MONOGRAFIES DE L'INSTITUT D´INVESTIGACIÓ EN INTEL·LIGÈNCIA ARTIFICIAL

Computationally Manageable Combinatorial Auctions for Supply Chain Automation

Andrea Giovannucci

Consell Superior d´Investigacions Científiques

MONOGRAFIES DE L'INSTITUT D'INVESTIGACIÓ
EN INTEL·LIGÈNCIA ARTIFICIAL
Number XXXII

Institut d'Investigació
en Intel·ligència Artificial

Consell Superior
d'Investigacions Científiques

Computationally Manageable Combinatorial Auctions for Supply Chain Automation

Andrea Giovannucci

Foreword by Juan Antonio Rodriguez-Aguilar and Jesus Cerquides

2008 Consell Superior d'Investigacions Científiques
Institut d'Investigació en Intel·ligència Artificial
Bellaterra, Catalonia, Spain.

Series Editor
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Foreword by Juan Antonio Rodriguez-Aguilar[a] and Jesus Cerquides[b]
[a] Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques
[b] Departament de Matemàtica Aplicada y Anàlisi
Universitat de Barcelona

Volume Author
Andrea Giovannucci
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Institut d'Investigació
en Intel·ligència Artificial

Consell Superior
d'Investigacions Científiques

© 2008 by Andrea Giovannucci
NIPO:653-08-087-X
ISBN:978-84-00-08666-4
Dip. Legal: B-30348-2008

Ordering Information: Text orders should be addressed to the Library of the IIIA, Institut d'Investigació en Intel·ligència Artificial, Campus de la Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain.

I dedicate this piece of life to my father Alberto, my mother Carla, and my brother Marco.

Contents

List of Figures

List of Tables

Nomenclature

δ	The overall number of SCOs mentioned anywhere in the bids with their multiplicities, page 102
ℓ	Length of a valid solution sequence for a MMUCA, page 124
$\mathbb{N}^G$	The set of multisets over the set G, page 102
$\mathcal{D}$	The *multiset* of the overall SCOs submitted by all bidders with their multiplicities, page 101
$\mathcal{D}_{ij}$	The multiset of SCOs offered in bid Bid_{ij}, page 101
$\mathcal{I}_{ijk}$	The input multiset of the SCO t_{ijk}, page 120
$\mathcal{M}$	Marking in a PTN, page 30
$\mathcal{M}^m$	The multiset indicating the resources available to the auctioneer at the $m-th$ step of a production process, page 102
$\mathcal{O}_{ijk}$	The output multiset of the SCO t_{ijk}, page 120
$\mathcal{U}_{in}$	The multiset indicating an auctioneer's initial stock, page 102
$\mathcal{U}_{out}$	The multiset indicating an auctioneer's final requirements, page 102
Σ	It represents an allcation sequence, i.e. a sequence of SCOs, page 102
Bid_{ij}	The $j-th$ bid submitted by the $i-th$ bidder, page 101
C_T	A vector representing the function C_{FS}, page 77
C_{FS}	Cost associated to a firing sequence on a WPTN, page 68
G	The set of goods at auction, page 102
M_k	A vector representing a marking $\mathcal{M}_k$, page 77
p_{ij}	The valuation associated to bid Bid_{ij}, page 101
PTN_E	Place Transition Net representing bids and internal production structure, page 63

PTN_I	Place Transition Nets representing an auctioneer's internal production structure, page 60
T	The *set* of the overall SCOs submitted by all the bidders disregarding their multiplicities, page 102
t_{ijk}	The $k-th$ SCO in the $j-th$ bid submitted by the $i-th$ bidder, page 101
DIP	Direct Integer Programming, page 138
CCIP	Connected Component Integer Programming, page 145
Auction Net	WPTN representing the MUCRAtR decision space, page 72
CA	Combinatorial Auction, page 41
CMWOSP	Constrained Maximum Weight Occurrence Sequence Problem, page 74
ILP	Integer Linear Programming. An optimisation technique, page 24
IP	Integer Programming. See ILP, page 24
MMUCA	Mixed Multi-unit Combinatorial Auction, page 104
MUCA	Multiunit Combinatorial Auctions, page 41
MUCRA	Multiunit Combinatorial Reverse Auctions, page 41
MUCRAtR	Multi-unit Combinatorial Reverse Auction with transformability Relationships among goods, page 51
OR	OR Bidding Language, page 42
PN	Petri Nets, page 27
PTN	Place Transition Nets, page 28
PTNS	Place Transition Net Structure, page 28
SCC	Strongly Connected Component, page 36
SCF	Supply Chain Formation, page 47
SCO	Supply Chain Operation, page 90
TNS	Transformability Network Structure, page 54
WDP	Winner Determination Problem, page 43
WPTN	Weighted Place Transition Net, page 67
WPTNS	Weighted Place Transition Net Structure, page 66
XOR	XOR Bidding Language, page 42

Foreword

Nowadays we are witnessing an important transformation of the way organizations operate to fulfill their objectives. We are moving from monolithic structures to collaborative structures whose components tend to reduce their sizes. This means that we are moving toward the paradigm of virtual organizations. In this setting, the ability to quickly and efficiently collaborate to design, develop, produce and sell a new product has become a key competitive advantage.

In this environment, enterprises face critical strategic decisions on whether to collaborate with other firms to complete some tasks across its supply chain. In this setting there is a need for an increased automation across the supply chain. Indeed, static and vertical integrated supply chains are quickly giving way to more flexible value chains composed of partners that can be assembled in real time to meet unique requirements.

This thesis is the result of a pioneer work on automating the process of collaborative supply chain network formation. At this aim, it proposes a novel combinatorial auction model, the so-called Mixed Multi-Unit Combinatorial Auction, that supports not only to trade and exchange goods but also to trade and exchange manufacturing operations. This model has achieved international recognition, has opened a new line of research in our institute and shows a high potential for industrial application.

We have been lucky to work with Andrea Giovannucci along these years. Our collaboration has been very fruitful and enjoyable both scientifically and personally. Thanks to his enthusiasm, generosity, friendliness, ambition for knowledge and team making capabilities, Andrea has been the PhD student every advisor would like to work with.

We wish the reader an experience as pleasant as the one we had while advising the author.

The supervisors

Juan Antonio Rodríguez Aguilar and Jesús Cerquides

Abstract

The need for automating the process of supply chain formation is motivated by the advent of Internet technologies supporting B2B and B2C negotiations: the speed at which market requirements change has dramatically increased. In this scenario enterprises must become flexible in the process of product customisation and order fulfilment. This can be only achieved if the supply chain formation process is agile, and thus the need for automation.

The main goal of this dissertation is to provide computationally efficient market-based auction mechanisms for automating the process of optimal supply chain partner selection. This is achieved by means of two progressive, non-trivial extensions of combinatorial auctions (CA).

On the one hand, we extend CAs to determine optimal outsourcing strategies. Thus, we provide computational means, via the so-called Multi-unit Combinatorial Auctions with Transformation Relationships (MUCRAtR), for an enterprise to optimise its *make-or-buy* decisions across the supply chain, namely to decide whether to outsource some production processes or not. At this aim, we add a new dimension to the goods at auction. A buyer can express its internal production and cost structure. Firstly, we introduce such information in the winner determination problem (WDP) so that an auctioneer/buyer can assess what goods to buy, from whom, and what internal operations to perform in order to obtain the required resources. In this way, an auctioneer can build his supply chain minimising its costs. Secondly, since the decision problem faced by the auctioneer is extremely hard, we also provide a formal framework to analyse the computational properties of the WDP and to facilitate the classification of WDPs, and hence to provide guidance for developing efficient solution algorithms.

On the other hand, we propose a novel CA, the so-called Mixed Multi-unit Combinatorial Auction (MMUCA), that automates the process of collaborative supply chain network formation. The outcome of such a new auction is the coordinated plan of a totally integrated supply chain (the selection of a set of supply chain partners along with the ordered set of operations that each partner must perform). We manage to provide computational means to optimise *make-or-buy-or-collaborate* decisions, and therefore to tightly link sourcing, outsourcing, and collaboration strategies. In this context, make, buy, and collaborate mean that a stakeholder of the supply chain decides whether to perform a set of services or operations by himself (make), to outsource them (buy), or to perform them in collaboration with other stakeholders (collaborate). A MMUCA allows agents to bid for bundles of goods to buy, to sell, and for bundles of (manufacturing)

operations across the supply chain. One such operation can be regarded as a step in a production process, and thus winner determination in a MMUCA amounts to choosing the sequence in which the winning bids must be implemented while minimising total cost. Furthermore, we introduce a bidding language for MMUCAs and analyse the corresponding WDP. Finally, we succeed in providing very efficient optimisations to the MMUCA WDP, based on a formal analysis of its topological structure, which can found their practical application to actual-world scenarios.

Acknowledgements

Innanzitutto grazie a tutta la mia famiglia, Carla, Alberto e Marco, per tutto il loro appoggio sia affettivo che economico in questo lungo e tortuoso viaggio che é stato il mio dottorato. Ai miei nonni, zii, zie e cugini che indirettamente hanno contribuito alla mia educazione e cultura.

A mis directores Jesús y Juan Antonio por lo que me han enseñado, no sólo desde el punto de vista profesional y técnico, sino también humano y moral. Gracias por su amistad y comprensión a lo largo de los momentos dificíles que he tenido durante estos años.

Thanks a lot to Ulle Endriss for his unsubstitutable help and contribution to the core of my PhD. In particular, thanks for his help in ideating and subsequently formalising the MMUCA auction and the associated bidding language.

A mi novia Ana, por el infinito apoyo moral y afectivo y por haber aguantado a un novio fantasma a lo largo de algunos meses. Gracias también a Arnau, Charo, y Cristina por el cariño y las enseñanzas que me han proporcionado.

A Claudio Baccigalupo y Manuel Atencia por sus cuidadosas revisiones y sus inestimables consejos. En general, un agradecimiento a todo el IIIA, por la ayuda ocasional pedida y concedida en algunos temas difíciles. Por el trato profesional y amistoso recibidos. Gracias en particular a Carles Sierra, Adrián Perrau (Eidrien), Félix Bou, Pedro Meseguer, Ramón López de Mántaras, Lluís Godo, Enric Plaza, Dani Pollack, José Luis y Tito Cruz.

Gracias a Bruno Rossell y Meritxell Vinyals por al ayuda en el desarrollo del software empleado en mis tesis y por los experimentos contenidos en esta tesis.

Many thanks to Gopal, Nick, Raj, Luke, Michelle, Sebastian, Xanna, Jiggar and more people from Southampton. Thanks to their help, friendship, support and teaching I enjoyed a wonderful experience in Southampton.

Grazie a tutti i miei amici italiani, senza i quali non sarei arrivato ad essere la persona che sono. Grazie per le lunghe chiacchierate, per le avventure vissute, e per essermi sempre stati vicini anche nella distanza. Grazie a Max, Francesca, Seve, Perrito, Ele, Michi, Checco Totti Abele, Andrea Calvi, e molti altri insostituibili amici.

Gracias a todos mis amigos de Barcelona, por aguantar a un amigo inexistente durante algunos meses. Gracias también por la ayuda, los consejos, la amistad incondicionada que siempre me han proporcionado. Gracias a Lorenzo, Manu, Sofía, Adrián, Tito, Francesca, Pablo, Becky, Nolwenn, Perrito, Juan Antonio, Jesus, Jordi, Ana, Marcos, Helena, Javier (maestro), y muchos mas.

Este trabajo ha sido parcialmente financiado por los proyectos de investigación (grants 2006-5-0I-099, TIN-2006-15662-C02-01, TIC-2003-08763-C02-01 and TIP-2003-08763-C02-01) y por una beca I3P (I3P-BDP2003 o BEC.09.01.04/05-164).

Chapter 1

Introduction

The main goal of this dissertation is to provide computationally efficient market-based auction mechanisms for automating the process of optimal supply chain partner selection. This is achieved by means of two progressive, non-trivial extensions of combinatorial auctions (CA). On the one hand, we extend CAs to determine optimal outsourcing strategies. Thus, we aim at providing a useful tool to optimise make-or-buy decisions across the supply chain. On the other hand, we propose a novel CA that automates the process of collaborative supply chain network design, planning[1], and formation. The outcome of such a new auction is the coordinated plan of a totally integrated supply chain (the selection of a set of supply chain partners along with the ordered set of operations that each partner must perform). Analogously, in the latter case we aim at providing a useful tool to optimise make-or-buy-or-collaborate decisions, and therefore to tightly link sourcing, outsourcing, and collaboration strategies. In this context, *make, buy,* and *collaborate* mean that a stakeholder of the supply chain decides whether to perform a set of services or operations by himself (make), to outsource them (buy), or to perform them in collaboration with other stakeholders (collaborate).

This chapter is organised as follows. In section 1.1 we explain why some think that our economy is undergoing profound changes in the next years. In section 1.2, we go back to reality and explain what is currently changing in our economy and what is required to adapt to such changes. In section 1.3 we recall some concepts and terminology related to supply chain management. In section 1.4, we specify and thoroughly exemplify the problems we cope with in this PhD thesis. In section 1.5 we highlight the contributions of this dissertation with respect to the state-of-the-art. Finally, in section 1.6, we elaborate on the structure of this dissertation.

1.1 A hypothesis for the future: Wikinomics

In his recent article, Burkeman (Burkeman, 2005) summarises and discusses the eye-opening new book of Don Tapscott called *WIKINOMICS: How Mass Collaboration Changes Everything* (Tapscott and Williams, 2006). According to Don Tapscott, a guru

[1]We remark that *supply chain planning* consists in assessing who will do what and when in a supply chain.

of the Web, “we have barely begun to imagine how the Internet will change the way we live and work”. We are living a revolution that is undermining the very basis of traditional economy. In his article, Burkeman recalls three examples of this transformation from the *Wikinomics* book:

- *Self-Organisers*: China’s flourishing motorbike industry is not composed of big organised firms hiring thousand of employees and outsourcing tasks to small subcontractors. Instead, a myriad of smaller companies collaborate and self-organise in order to share risks and profits. Their representatives meet in tea-shops or in on-line places and jointly plan a product, to which they contribute with the service they are best at. Even the final assembly is a service. A “self-organised system of design and production” has emerged.

- *Prosumers*: when amateurs began to hack the computerised parts at the heart of the Lego Mindstorm range (Shaeffer, 2007), the company initially threatened to sue them. Then, perceiving the wind of change, Lego started to encourage them to be *prosumers*, consumers that have an active role in the design of a product. This lead to an increased satisfaction of customers without harming the enterprise profit.

- *The new gold rush*: the Gold mine at the Red Lake in Ontario, owned by Goldcorp, was in a terrible crisis in 1999. When the chief executive Rob McEwen heard a talk about Linus Torvald, the inventor of Linux, he came up with a revolutionary idea. If developers collaboratively code on the Web, why not share the mining activity on the web? Then, he put Goldcorp secret geological data on the web and set a 575,000 $ prize to reward the discovery of new gold veins in Red Lakes’s mine. Around 80 valid targets were identified and the company value turned from $100m to $9bn.

Those three cases above aim at showing that the collaborative structure, recently emerged in social and collaborative networks as Wikipedia (Lih, 2003) and Sourceforge (SourceForge, S.F., 2007), could be far more radical and change the way we think about manufacturing. In his book, Tapscott introduces his revolutionary idea of “wikinomics”, an idea that originates in a work that dates back to 1937 (Coase, 1937). At that time, Ronald Coase, a Nobel prize economist, noticed something odd in capitalism. Capitalism predicates the free market and exchange. If capitalist theory was correct American or British people should do business among them as individuals in an open market, and not organise themselves in firms, as it happens. The motivation (Coase, 1937) is that making things requires collaboration, and that finding and linking up all the people who need to collaborate costs money. Companies emerge when it is cheaper gathering people, materials, and tools under the same roof, rather than going out looking for the best deal every time a few hours’ work is required. However, the Internet is radically lowering the cost of collaborating. Consequently, big companies are doomed to reduce their size in order to leave space to more agile and flexible collaborative structures. A symptom of this new collaborative reorganisation is that, for instance, large companies, from media outlets to clothes shops, are trying to make profit by incorporating final customers in the creation of their products. However, *Wikinomics* forecasts a further radical revolution: it is not given that the company will stay in the driving seat at all. Quoting Tapscott: “We are talking about a new means of production. Collaboration can occur at an astronomical scale, so if you can create an encyclopedia with a bunch of people, could you create a mutual fund, a motorcycle?”.

Tapscott is not the only one prohetising a wiki future. For instance, Laubaucher and Malone (Laubacher and Malone, 2003) claim that “The most radical new organisational form, the virtual corporation, involves small firms and free-lancers, or even e-lancers — electronically connected free-lancers, who post their qualifications and find assignments on the Internet — joining forces on a temporary basis, working together on a project, then disbanding when the work is completed. Virtual corporations of this sort have long characterised film production and construction and are increasingly prevalent in the most dynamic and fastest-growing sectors of the economy — computers and telecommunications, entertainment, biotechnology.”

Other terms employed to indicate analogous concepts are *virtual corporation*, *virtual organisation* (Mowshowitz, 2002), and *extended enterprise* (Dyer, 2000).

1.2 With the feet in the air & the head on the ground

The provocative title quotes The Pixies’ song *Where is my mind*. It aims at highlighting the fact that wikinomics is a far goal. However, any revolution takes its time to entirely develop, and probably several intermediate steps are required to approach the new economy envisaged by Tapscott and Couse. Then, in this section we stay with *the head on the ground* and we analyse what is going on in the business world now. We will summarise what is changing and why. At the same time we will comment on the requirements that originate from such changes.

We are witnessing an important transformation of the firm organisational structure. Today’s business world is experiencing a progressive disintegration of the traditional vertical integrity[2] of the enterprises’ organisational structure. This is witnessed by a heavy increment in the use of outsourcing. Quoting Greaver (Greaver, 1999), “Outsourcing is the act of transferring some of an organisation’s recurring internal activities and decision rights to the outside providers, as set forth in a contract”. Outsourcing is one of the success keys of western economies and is widely employed. Indeed, a recent on-line news (DMReview.com online news, 2005) about outsourcing claims that, “According to a newly released IDC study, the worldwide BPO (Business Process Outsourcing) market is vibrant and brimming with opportunity. The comprehensive BPO report finds that worldwide BPO spending will experience a five-year compound annual growth rate (CAGR) of 10.9 percent, growing from $382.5 billion in 2004 to $641.2 billion in 2009. This forecast covers eight BPO markets: human resources, procurement, finance & accounting, customer service, logistics, sales & marketing, product engineering, and training”. Another on-line news (DMReview.com online news, 2006) says that “According to a newly released IDC study, the business outsourcing market progressed positively in 2005, experiencing a 33 percent increase in the volume of deals signed. [...]. Small and mid-size deals are fuelling growth. Underlying this trend is

[2]In microeconomics and management the term *vertical integration* describes the degree to which a firm owns its upstream suppliers and its downstream buyers.

an increase in the share of new deals versus extensions and renewals, which indicates that a growing number of new organisations are buying into the business outsourcing model. [...]. Manufacturing, financial services, and government verticals registered the strongest adoption of business outsourcing overall".

The trend is quite clear. We are moving from vertically integrated structures to collaborative structures whose components tend to reduce their sizes (Lucking-Reiley and Spulber, 2001; Hammer, 2001). This means that we are slowly moving towards the paradigm of virtual enterprises. This is a symptom endorsing the *Wikinomics* theory. Such transformation is due to many factors.

Firstly, today's business environment is getting tougher and tougher. Indeed, nowadays customers are increasingly demanding better and innovative goods, as well as progressively more customised products. This new situation entails some implicit production requirements and constraints like timeliness, convenience, responsiveness, quality, and reliability. Moreover, ever lower prices are imposed by a fierce market competition.

Secondly, the rapid pace of innovation has entailed a shorter product and technology life cycle (for instance, the PC or phone industries where new models are introduced each 3 to 9 months), and an increased uncertainty in supply and demand. Notice that the presence of technology, in particular the Internet, has also made the work of modern organisations placeless. This has forced an increased specialisation of the operational activities across an organisation.

Thirdly, we are experiencing a worldwide increment in competition (hypercompetition). We are fastly moving from a best-in-class to a best-in-world paradigm, barriers are dropping quickly, competition is just one click away from any customer. Companies that recently were in separate fields now compete in the same narrow market (for instance, Apple with the iPod efficiently entered into the MP3 player market).

Finally, we are witnessing a rapid commoditisation of goods[3], due to the rapid price decline and to the increased pressure for improved performances.

Thus, the ability to quickly and efficiently design, develop, produce and sell a new product has become a key competitive advantage. That is why the structural integrity of organisations is breaking down; the traditional vertically integrated organisations, controlling as many of the production factors as possible, is being quickly replaced by better focused and more specialised organisations. An increased number of capable service providers, the pressure deriving from the hypercompetitivity, and the pervasive presence of technology impose a new strategic vision. As a result, new supply chain management (Simchi-Levi et al., 2000) strategies are emerging, like strategic outsourcing (Quinn and Hillmer, 1995; Greaver, 1999; Corbett, 2004) and collaborative supply chain network design (Viswanadham, 2002).

Notice that the intersection between portions of supply chains of different firms is often non empty. For instance, *original equipment manufacturers* (OEM) are typical in rapidly chaining markets. The term *original equipment manufacturer* (OEM) refers to a company that sells a manufacturing component to another company, that in turn resells it as its own, usually as a part of a larger product.

[3]In essence, commoditisation occurs as a good or service becomes undifferentiated across its supply base by the diffusion of the intellectual capital necessary to acquire or produce it efficiently. As such, many products which formerly carried premium margins for market participants have become commodities, such as generic pharmaceuticals and silicon chips (Schrage, 2007).

In this environment, the selection of the right business partners is critical, which are quickly moving from the role of suppliers, manufacturers, customers, to the role of *collaborators*. Hence, many enterprises now face critical *make-or-buy-or-collaborate* strategic decisions across their supply chain: different types of actors, as component suppliers, contract manufacturers, service purchasers, logistic providers, and final customers have to be efficiently integrated into the supply chain. In particular, one of the main objectives of current supply chain management (Simchi-Levi et al., 2000) is to integrate as much as possible the *back-end* of the supply chain (its production and manufacturing portion) to the *front-end* (the final customer).

Another fundamental requirement stemming from the business environmental changes explained above is a need for an increased automation across the supply chain. Indeed, static and vertically integrated supply chains are quickly giving way to more flexible value chains composed of partners that can be assembled in real time to meet unique requirements. This phenomenon is being accelerated by the Internet, that lowered the communication barriers transforming a game that was firm against firm into a game that is supply chain network against supply chain network (Viswanadham, 2002).

A spectrum of possible solutions is possibly needed by enterprises. On the one extreme, companies must make decisions about whether to outsource part of their production processes (buy/make decisions) in business environments characterised by myriads of possible partners (lower barriers caused an increment in competition). On the other extreme of the spectrum, virtual enterprises may need agile *decision support systems* (DSSs) that allow them to automatically form self-organising supply chains.

Indeed, we do believe that nowadays firms, or group of firms, require DSSs that allow them to nimbly and automatically select strategic business partners. With this goal, those DSSs should allow firms to:

- automate the process of partner selection, optimising critical *make-or-buy* decisions across the supply chain (i.e. trading off decisions of internal vs external production) with myriads of potential partners. Clearly this entails a tight integration of the procurement and outsourcing strategies.

- decide whether to collaborate with other firms to complete some tasks across its supply chain. In this case companies need to automate *make-or-buy-or-collaborate* critical decisions across the supply chain with myriads of potential partners.

- automate the process of collaborative supply chain network design and planning with a large number of potential partners. In particular, the decision support should allow them to self-organise by allowing to:

  – integrate and coordinate all the supply chain stakeholders;

  – include component suppliers, contract manufacturers, logistic providers and final customers into the supply chain design process;

  – optimise the overall performance of the supply chain (i.e. not a local optimisation);

- easily support mass customisation[4]; and
- integrate potential suppliers and final customers into new product developments.

Obviously, decisions like the ones considered above can emerge as long as the supply chain stakeholders collaborate and share information like capacity, schedule, and cost structures. However, full transparency and collaboration is rather unlikely. Then, all the previous requirements should come with the possibility to share only part of a stakeholder's internal information, without being forced to reveal every piece of critical production information.

With the above-mentioned requirements fulfilled, competitive companies could easily cope with a wide range of difficult business decisions: from the selection of optimal, tightly connected procurement, outsourcing, and collaboration strategies, to the formation of virtual enterprises.

In the next section, we briefly introduce the definition of supply chain and we provide some terminology that will be useful in the remaining of the chapter.

1.3 Supply Chain and Supply Chain Management

According to (Simchi-Levi et al., 2000), "In a typical supply chain, raw materials are procured and items are produced at one or more factories, shipped to warehouses, for intermediate storage, and then shipped to retailers and customers. [...] The supply chain, consists of suppliers, manufacturing centers, warehouses, distribution centers, and retail outlets.".

Supply chain management (SCM) "is a set of approaches utilised to efficiently integrate suppliers, manufacturers, warehouses, and stores, so that merchandise is produced and distributed at the right quantities, to the right locations, and at the right time, in order to minimise system-wide costs while satisfying service level requirements" (Simchi-Levi et al., 2000). One of the core objectives of the supply chain is to perform a global optimisation across the supply chain. But many features of the way businesses are run today prevent this from happening: the uncertainty underlying the supply, the demand, the transportation time, the vehicles and the tools breakdowns. Furthermore the various stakeholders across the supply chain locally maximise their utility disregarding the performances of the other elements within the supply chain. In fact, the different components often have even conflicting objectives. Traditional SCM deals with all these problems acting on different aspects of control: distribution network configuration, supply contracts, distribution strategies, supply chain integration and strategic partnering, inventory control, outsourcing and procurement strategies, information technology and DSSs, etc.

In particular, aspects relevant to our work are:

(1) outsourcing and procurement strategies considered in the first part of this dissertation; and

[4]According to (Simchi-Levi et al., 2000) "mass customisation involves the delivery of a wide variety of customised goods or services quickly and efficiently at low cost".

(2) supply chain integration and strategic partnering, considered in the second part of the PhD thesis.

Since our work mainly focuses on outsourcing issues, in what follows we provide some basic related terminology. Different operational aspects of the supply chain can be outsourced. More specifically, we classify the types of possible supply chain partners into four categories:

- *component suppliers*, also called providers, that supply raw or intermediate goods across the supply chain;
- *contract manufacturers*, that provide services or manufacturing operations across the supply chain;
- *service purchasers*, that require services or manufacturing operations across the supply chain;
- *logistic providers*, in charge of the transportation, distribution, and storage of raw, intermediate or manufactured goods; and
- *final customers*, at the end of the supply chain, be them either retailers, or, in the new Internet era, final clients.

In this dissertation we narrow the focus of the investigation to the collaboration of component suppliers, contract manufacturers, service purchasers, and final customers. We deem necessary the incorporation of the logistic portion into the problem. However, in this dissertation the collaboration with logistic providers is left out, and will be thoroughly discussed as a path of future work in chapter 9. Therefore, in this dissertation we assume that logistics are negotiated independently.

1.4 The Problem

Once outlined in section 1.2 the requirements originating from the vertiginous changes in today's business world, we focus on the requirements that we tackle in this dissertation. In particular, we present two motivating examples concerning the main issues we intend to face in this thesis: the problem of efficiently solving *make-or-buy* and *make-or-buy-or-collaborate* decisions across the supply chain. Both examples consider an imaginary company devoted to produce and sell apple pies called *Grandma & co*. The examples, along with the emerging implicit requirements, are thoroughly presented in sections 1.4.1 and 1.4.2.

1.4.1 Optimising make-or-buy decisions

The first example aims at making explicit the requirements regarding the automation of *make-or-buy* decisions.

Example 1.1. Consider a company, named *Grandma & co*, devoted to produce and sell apple pies. The internal production structure of the company, i.e. the way apple pies are prepared, is presented in figure 1.1. Each circle represents a raw, intermediate or manufactured good. Squares connecting goods represent manufacturing operations. An arc connecting a good to an operation indicates that the good is an *input* to the operation, whereas an arc connecting an operation to a good indicates that the good is an *output* of the operation. Then, *butter*, *sugar*, and *flour* are *input goods* to the *Make Dough* operation, whereas *dough* is an *output good* of the *Make Dough* operation. The labels on the arcs connecting *input goods* to operations, and the labels on the arcs connecting *output goods* to operations indicate the units required of each *input good* to perform an operation and the units generated per *output good* respectively. In our example, the preparation of two units of *dough* requires one unit of *butter*, three units of *sugar*, and two units of *flour*.

Each operation has an associated cost every time it is carried out. We label each operation with a cost. In our example, the *Make Dough* operation costs 5 €.

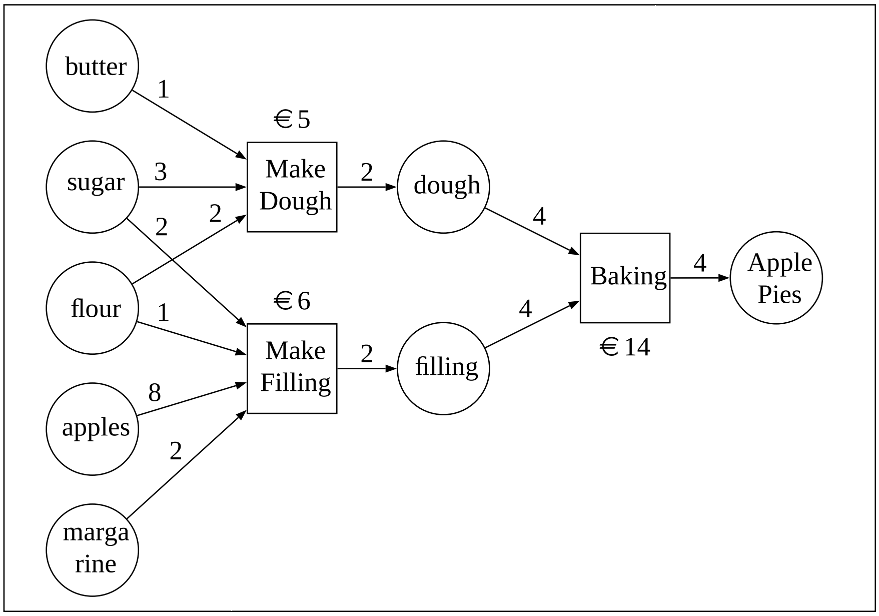


Figure 1.1: Apple pie production flow.

Consider that the marketing department at *Grandma & co* forecasts that two hundred apple pies will be sold within a month. Therefore, the company starts an automated sourcing (Minahan et al., 2002) process to acquire the basic ingredients needed for producing pies, namely *butter*, *sugar*, *flour*, *apples*, and *margarine*.

However, the production management staff decides to test a new sourcing process. Instead of limiting the procurement to basic ingredients, they decide to incorporate in the sourcing process intermediate and final goods as well, namely *dough*, *filling*, and *apple pies* in figure 1.1. More precisely, the production management wonders whether to *outsource* part of its production process. In fact, the executive staff noticed that more and more specialised enterprises are entering the organic food market. Since *Grandma & co* is a well-known brand for pies, it decides that in order to reduce costs, it could be suitable to negotiate and collaborate with those new brands.

As an additional constraint, the production management knows that strong complementarities among the negotiated goods exist on the supplier side. For instance, suppliers often sell margarine and butter as indivisible bundles. Thus, it is required that those complementarities are taken into account. □

Grandma & co realises that it faces a decision problem: shall it buy the required ingredients and internally produce apple pies, or buy already-made apple pies (outsource all its production), or opt for a *mixed purchase* and buy some ingredients for internal production and some already-made apple pies? This concern is reasonable since the cost of ingredients plus preparation costs may eventually be higher than the cost of already-made apple pies. *Grandma & co* must take a decision among many possible mutually exclusive options:

- buy all the basic ingredients to internally produce all the pies;
- buy from suppliers all the pies and resell them under its name;
- buy already-made dough and filling from suppliers , and bake itself the cake;
- prepare part of the dough and part of the filling, and buy the rest from suppliers;
- buy part of the pies from suppliers and produce the rest itself;
- and so on.

Grandma & co is interested in quantitatively assessing what to buy and from whom, as well as what to produce in house. Such assessment depends on many factors:

(1) the market cost of the basic ingredients (butter, sugar, flour, apples, and margarine);

(2) the market cost of dough, filling, and pies;

(3) the stock goods at *Grandma & co*;

(4) the finally required goods (the sales forecast);

(5) the cost for performing at *Grandma & co* the operations *Make Dough*, *Make Filling*, and *Baking* (the internal cost structure);

(6) the number of units of each good either produced or required for each operation (the internal production structure); and

(7) the complementarity relationships among goods holding on the suppliers' side.

Hence, *Grandma & co* requires a complex decision support system along with a negotiation mechanism that helps it in detecting which is the revenue maximising buying configuration and the internal operations to perform in order to obtain the finally required goods. It is easy to understand from the example that the procurement and outsourcing decisions are tightly linked. Notice that there is a mutual dependency among the outsourcing opportunity, the ingredients' market prices (as Dough, Apples,etc.) and other factors. This kind of dependencies must be absolutely captured by any proposed solution.

The literature on procurement has introduced combinatorial reverse auctions to deal with the problem of complementarities among goods on the bidders' side. In the following section we briefly recall some knowledge about electronic sourcing and combinatorial auctions.

The procurement phase

In the everyday business world, the sourcing process of goods and services usually involves complex negotiations. With the advent of the Internet, a plethora of commercial products to electronically support this process (e-sourcing tools) have started to be commercialised by a significant number of vendors (e.g. Ariba, Emptoris, Perfect, and iSOCO to name a few[5]). Thus, e-sourcing tools have become an established part of the business landscape (Team, 2001). Reverse[6] auctions are at the heart of most of these tools as the mechanism for buying companies to automate their negotiations with the qualified providers in their supply chains.

Although reverse auctions are certainly valuable to swiftly negotiate with providers, combinatorial (reverse) auctions may lead to more efficient allocations whenever complementarities among the goods at auction hold, as argued in (Sandholm, 2002). A combinatorial (reverse) auction (Cramton et al., 2006) is an auction where bidders can sell (buy) entire bundles of goods in a single transaction. Although computationally very complex, selling (buying) items in bundles has the great advantage of eliminating the risk for a bidder of not being able to obtain (sell) complementary items at a reasonable cost (price) in a follow-up auction (think of a combinatorial auction for a pair of shoes, as opposed to two consecutive single-item auctions for each of the individual shoes).

In particular, connected with the introduction of combinatorial auctions are bidding languages (Nisan, 2006) and the winner determination problem (WPD) (Lehmann et al., 2006). Winner determination is the problem, faced by the auctioneer, of choosing what goods to award to which bidder so as to maximise its revenue. The winner determination for combinatorial auctions is a complex computational problem. In particular, it has been shown that the WDP is NP-complete (Rothkopf et al., 1998). Bidding is the process of transmitting one's valuation function over the set of goods at offer to the auctioneer (or rather *some* valuation function — the bidders are of course not required to reveal their true valuation —).

[5]We refer the reader to (Bartels et al., 2005) for an analysis of e-sourcing tools.

[6]An auction is called *direct* when the auctioneer aims at selling goods, whereas we talk about *reverse* auction when the auctioneer is interested in buying goods.

Since *Grandma & co* aims at dealing with the case in which complementarities among goods hold at the bidder's side, combinatorial auctions is for sure the more suitable sourcing method. Then, in order to cope with *Grandma & co*'s problem, we employ combinatorial auctions. Anyway, combinatorial auctions cannot be directly employed for the problem explained in example 1.1 due to some intrinsic limitations.

To the best of our knowledge, no author directly dealt with the *make-or-buy* decision problem employing reverse combinatorial auctions. On the one hand, combinatorial reverse auctions solve the problem of procurement when complementarities among goods exist on the supplier side. On the other hand, operations research has studied the best *make-or-buy* decisions based on past production information, sell forecast, providers' offers, etc (Aissaoui et al., 2007)[7]. However, nobody embedded the decision problem into the procurement problem when complementarities among goods hold, nobody analysed the procurement decisions in conjunction with the outsourcing decisions in a combinatorial scenario. Then, in what follows, we analyse the requirements associated with the *make-or-buy* decision problem that are not fulfilled by combinatorial auctions, and we discuss the extensions required in order to deal with such decision problem.

Combinatorial Auction limitations

Say that *Grandma & co* opts for running a combinatorial reverse auction (Sandholm et al., 2002) with qualified providers for the procurement of all the required goods. Unfortunately, traditional combinatorial reverse auctions cannot be applied to solve such a problem for three reasons. Firstly, because of expressiveness limitations, namely an auctioneer (*Grandma & co*) cannot express:

- its internal manufacturing operations along with the producer/consumer relationships holding among them (for instance, in figure 1.1, the output of *Make Dough* is an input of *Baking*);
- the relationships between the manufacturing operations and the auctioned goods (for instance, in figure 1.1, the input to the *Make Dough* operation is three units of *sugar*, two units of *flour* and one unit of *butter*, whereas its output is two units of *dough*);
- the relationships between the received bids and the internal manufacturing operations;
- the requirements sent to bidders. This is clarified by observing that even though the final requirements of *Grandma & co* are two hundred apple pies, multiple request configurations fulfil such outcome, for instance:
  - two hundred already-made apple pies
  - the basic ingredients plus in-house production of two hundred apple pies

[7]For a general review on decision support to supply chain management refer to (Erenguc et al., 1999).

How can *Grandma & co* formally describe its requirements? What should be the requirements sent to bidders? In fact, the optimal requirements depends on the received offers, and therefore cannot be stated a priori.

- the cost associated to performing each internal operation or a set of internal operations.

The second problem is that the outcome of a combinatorial auction only provides information about what goods to buy and from whom. However, the information about which internal manufacturing operations to perform and the order in which the auctioneer has to perform them (in the example of figure 1.1, the auctioneer cannot perform the *Baking* operation before *Make Dough* or *Make Filling*) is not provided.

Table 1.1 summarises the requirements stemming from the *make-or-buy* decisions that are not supported by any state-of-the art solution.

TYPE	LIMITATION
Expressiveness	(1) internal manufacturing operations and the producer/consumer relationships among them
	(2) specification of an auctioneer's final requirements
	(3) relationships among the manufacturing operations, the auctioned goods, and the received bids
	(4) specification of an auctioneer's internal cost structure
WDP	(5) information about which in-house operations to perform and in which order

Table 1.1: Summary of unfulfilled requirements.

Although combinatorial auctions help set the market price of each good, they do not incorporate the notion of internal manufacturing operations. This is why all the above-mentioned difficulties arise.

Summarising, *Grandma & co* requires an extended combinatorial reverse auction that provides:

(1) a formal language to quantitatively express, analyse, and communicate its internal production structure and requirements; and

(2) an efficient cost minimising winner determination solver that not only assesses which goods to buy and from whom, but also the sequence of internal manufacturing operations needed to obtain the finally required goods.

1.4.2 Optimising make-or-buy-or-collaborate decisions

In what follows, we further increase the complexity of the scenario illustrated in example 1.1. Besides component suppliers, *Grandma & co* brings contract manufacturers, service purchasers, and final customers into the auction. We clarify what we stated above by means of the following example.

Example 1.2. Consider again the example of *Grandma & co*. The revolutionary production management (PM) staff decides that, besides all the goods , *Grandma & co* will negotiate all the operations along its supply chain. Thus, it invites to the auction suppliers of goods, suppliers of manufacturing operations (as *Make Dough* or *Baking*), and final customers/buyers of the final product (apple pies). Since *Grandma & co* is often asked to perform some service operations (as *Baking* for instance) for other companies, it decides to bring into the auction service purchasers as well. Summarising, *Grandma & co*, acting as auctioneer, receives offers from four types of bidders, namely:

(1) **component suppliers:** bidders that offer goods (for instance, two hundreds units of flour and a hundred units of sugar for 800 €);

(2) **contract manufacturers:** bidders that offer manufacturing operations (for instance, perform the operation *Make Dough* at 4 €);

(3) **service purchasers:** bidders that require manufacturing operations (for instance, willing to pay € 42 for having the operation *Make Filling* done seven times); and

(4) **final customers:** bidders that ask for goods (for instance, two hundred units of apple pies for 2400 €).

□

Resorting to example 1.2, in what follows we clarify what we intend for *make-or-buy-or-collaborate* decisions. Say that there is a contract manufacturer that is very able to efficiently and cheaply perform the *Baking* operation, i.e. at a cost of € 10. However, it performs very poorly the *Make Filling* and the *Make Dough* operations. In such a case, the way to optimally produce apple pies for both firms is to *collaborate*: i.e *Grandma & co* will be in charge of buying the basic ingredients to subsequently transform them into *Dough* and *Filling*, whereas the other firm of the *Baking* operation. Together they can offer a more competitive price.

Observe that it might be the case that *Grandma & co* acts as a pure intermediary for some or all the operations. Eventually, someone might perform the *Baking* operation and someone else might require the *Baking* operation. In this case the operation is performed *by* a bidder *for* another bidder, and *Grandma & co* acts just as an intermediary that makes profit by connecting the service provider and asker.

From example 1.2, we see that more stakeholders, besides component suppliers, have to be brought into the negotiation. In particular, we need to incorporate contract manufacturers, service purchasers, and final customers. Hence, it is compulsory to introduce a unified formal language for describing all the possible types of operations that supply chain stakeholder can negotiate upon. We classify such operations in four types:

(1) *Supply of manufacturing, assembly, disassembly operations*. For instance, the cost of assembling a personal computer given a mother board, a CPU, two memory units and a hard drive costs €12. This type of operation will typically describe services offered by contract manufacturers.

(2) *Demand of manufacturing, assembly, disassembly operations*. For instance, a bidder is willing to pay 5€ to have his PC assembled given that he provides the components (e.g. a mother board, a CPU, two memory units and an hard drive).

(3) *Supply of goods*. For instance, a supplier offers 100 units of RAM memories and 100 units of CPUs at €4000. This type of operation will typically describe services offered by component suppliers.

(4) *Demand of goods*. For instance, a customer is willing to pay €5000 for 20 PCs. This will typically describe operations associated to final customers.

We will refer to any of the possible operations mentioned above with the term *supply chain operation* (SCO).

Grandma & co faces a decision problem more complex than the one explained in section 1.4.1. Although the use of combinatorial reverse auctions may allow *Grandma & co* to improve its supply chain, there are further limitations that prevent its use:

(1) Even though combinatorial auctions allow to express offers or requests on bundles of goods, there exists no language to express offers or requests of manufacturing operations across the supply chain. Furthermore, along the lines of expressive commerce (Sandholm, 2006a)[8], it is desirable to provide bidders with a language rich enough to compactly express several possible offer alternatives.

(2) Besides complementarities among goods, further relationships must be taken into account. Those relationships link all the stakeholders of a supply chain by means of producer/consumer relationships. For instance, there is a producer/consumer relationship between any producer or supplier of *dough* and any supplier of the *Baking* operation since *dough* is requested to perform the *Baking* operation (see figure 1.1). Those relationships have only been partially taken into account by current combinatorial auction models despite being present in most real-world scenarios. In fact, the inputs and outputs of a production process are strongly connected since a manufacturer may risk:

- to produce unsold goods, thus losing money; and
- to fail to produce already sold goods when no able to obtain the required inputs, thus losing credibility on the market.

Hence, a supply chain can be regarded as an intricate network of suppliers, manufacturers (entities transforming input goods into output goods at a certain cost), and consumers interacting in a complex way. The complementarities arising

[8]Expressive commerce is a new sourcing paradigm in which supply and demand are expressed in greater detail than in traditional electronic commerce. A subsequent optimisation allows to discover the most profitable alternatives.

in the scenario of example 1.2 are different from the ones we do find in CAs. They arise because of the preconditions and postconditions of manufacturing processes: precedences and dependencies along the supply chain must be taken into account. Hence, whilst in CAs the complementarities can be simply represented as relationships among goods, in supply chains the complementarities involve not only goods, but also interrelated manufacturing relationships across several levels of the supply chain.

(3) Similarly to the case discussed in section 1.4.1, the outcome of a combinatorial auction does not provide an ordered sequence of supply chain operation to perform. However, an auctioneer must know the sequence of operations to perform in order to make its supply chain operate.

The most significant attempt to deal with the shortcomings exposed above has been undertaken by Walsh et. al (Walsh et al., 2000). Although they mainly focus on analysing the problem of distributed supply chain formation (SCF), in which no auctioneer is leading the formation process, the underlying problem is similar to a certain degree. Quoting Walsh et al. (Walsh and Wellman, 2003), "Supply Chain Formation is the process of determining the participants in a supply chain, who will exchange what with whom, and the terms of the exchanges". They define a new type of auction, the *combinatorial auction for supply chain formation*, which deals with scenarios in which multiple agents must form a supply chain. In order to cope with some of the above-mentioned combinatorial auction limitations, Walsh et al. (Walsh et al., 2000) introduce the notion of task dependency network (TDN). TDNs offer the means to express:

- offers on bundles of goods;
- demands of bundles of goods; and
- offers on a single manufacturing operation (with only one output product and multiple input components).

Furthermore, TDNs well describe the production complementarities we highlighted in point (2) of the combinatorial auctions shortcomings listed above, which is the possibility of expressing producer/consumer relationships.

Nonetheless, although TDNs are indeed valuable to model SCF, further requirements must be addressed to fully support automated negotiations across the supply chain. In fact, Walsh et al. (Walsh et al., 2000) mainly focus on game theoretical and economical issues, and do not elaborate on computational and expressiveness issues. Hence, due to some intrinsic limitations, TDNs cannot cope with all the requirements we exposed above. In particular, the requirements associated to the *make-or-buy-or-collaborate* decision problem that TDNs do not support are the following:

(1) the ability to represent all possible supply chain network topologies (TDNs only supports acyclic networks);

(2) the possibility to express complementarities among supply chain operations (for instance, if *Make Dough* and *Make Filling* share some machine, they can be cheaper if offered together) ;

(3) the possibility for bidders to require supply chain operations (TDNs only allow to offer them);

(4) the possibility to express resource sharing (for instance, an oven is a resource that can be shared);

(5) the possibility to express minimum/maximum capacity constraints on the number of times each supply chain operation can be performed (for instance, in presence of economies of scale[9] there is a critical number of operations that drastically reduce the price of a manufacturing process);

(6) the possibility to express any type of manufacturing operation (for instance, TDNs only allow operations with a single output);

(7) providing a coordinated scheduling plan among the supply chain stakeholders;

(8) solving *Make-or-Buy-or-Collaborate* decisions (i.e. not only supply chain formation problems);

(9) the ability to specify the configuration an auctioneer aims to end up with (the sales forecast for *Grandma & co*).

Then, although TDNs are indeed valuable to model SCF, further requirements (regarding *expressiveness* and *computability*) must be addressed to fully support automated supply chain network design and planning.

As to *expressiveness requirements*, we shall need:

(1) to support a wide range of supply chain topologies beyond acyclic nets;

(2) to provide bidders with means for expressing several types of preferences over supply chain operations;

(3) the configuration to end up with (i.e. the sale forecast);

As to *computational requirements*, we must ensure;

(1) that the outcome of the optimisation problem is not only the set of winning bids, but also a coordinated and integrated plan of all the supply chain stakeholders;

(2) the computational tractability of supply chain network design and planning while preserving optimality. This is an important requirement since, as explained in section 1.2, myriads of agents could potentially participate.

In table 1.2 we list the requirements associated to the *make-or-buy-or-collaborate* decision problem that are not currently supported by any state of the art methodology or tool. Summarising, *Grandma & co* needs:

[9]Economies of scale characterise a production process in which an increase in the scale of the firm causes a decrease in the long run average cost of each unit.

TYPE	REQUIREMENTS
Expressiveness	(1) support any supply chain topology (2) provide bidders with a language for expressing several types of preferences over supply chain operations (3) configuration to end up with
Computational	(4) compute the scheduled sequence of supply chain operations to perform (5) computational tractability of supply chain network planning while preserving optimality

Table 1.2: Requirements associated to *make-or-buy-or-collaborate* decisions.

(1) a language for expressing the offers/requests of the different actors involved in the auction. This language should be able to represent demands and offers of supply chain operation, and should be expressive enough to overcome the shortcomings of TDNs.

(2) a scalable winner determination solver that not only assesses the supply chain partners that maximise the auctioneer's revenue, but also provides an integrated coordination/scheduling plan for the emerging supply chain. That is, it should provide information about the synchronised sequence of supply chain operations that must be performed.

In the previous two sections, we introduced the requirements connected with the solution of *make-or-buy* and of *make-or-buy-or-collaborate* decisions. In the following section we will outline the approach we employed to fulfil such requirements.

1.5 Contributions

In this dissertation we contribute with two generalisations of combinatorial auctions providing support to the *make-or-buy* and *make-or-buy-or-collaborate* decision problems.

In the first part of this dissertation we present an extension to combinatorial auctions that we shall refer to as *Multi-Unit Combinatorial Reverse Auction with Transformability Relationships Among Goods* (MUCRAtR). MUCRAtR automates *make-or-buy* decision problems in scenarios characterised by combinatorial preferences. This new auction type provides an auctioneer with a framework to optimise its outsourcing and procurement strategy. In particular, it allows an auctioneer:

- to formally express its internal production structure; and
- to automatically and efficiently assess which goods to buy and from whom, along with the *sequence* of internal operations to perform in order to obtain some required resources.

In order to provide a language to express the internal production structure of an auctioneer, we extend Petri Nets (refer to section 2.3 or to (Murata, 1989)), a well-known graphical and formal tool to analyse discrete dynamical systems. We call such extended model *Weighted Place Transition Nets* (WPTNs). The semantics of WPTNs naturally captures:

- the producer/consumer relationships holding among manufacturing operations; and
- the relationships among goods at auction, auctioneer's internal operations, and bids.

Next, in order to provide a formal definition to the auctioneer's decision problem, we define a new optimisation problem on WPTNs, the *Constrained Maximum Weighted Occurrence Sequence Problem* (CMWOSP). The resulting optimisation problem perfectly captures the nature of the auctioneer's decision problem. We anticipate that the newly introduced optimisation framework allows to import a wide body of analysis methods from Petri Nets theory and apply them to our decision problem, thus providing methods and tools for its analysis. Subsequently, in order to practically solve the auctioneer's decision problem, we exploit analysis methods imported from Petri Nets theory and manage to provide an efficient Integer Linear Programming (ILP) (Hillier and Lieberman, 1986) formulation of the problem. However, this formulation only works when an auctioneer's internal production structure is acyclic. That is, there are no cycles in a production process.

In the second part of the dissertation we present another extension of combinatorial auctions, namely *Mixed Multi-Unit Combinatorial Auctions* (MMUCA), that allows to deal with *make-or-buy-or-collaborate* decisions. This new auction type provides an auctioneer with an automatic method to optimally select supply chain partners. Our contribution develops along three dimensions:

(1) **Bidding Language**. We provide a novel language that allows agents to express a range of preferences over complementary operations across the supply chain. We build this language by extending and generalising previous languages for Combinatorial Auctions. In particular, we introduce the notion of *supply chain operation* (SCO). The notion of SCO encompasses several types of operations across the supply chain. Then, we provide bidders an expressive language to trade SCOs.

(2) **Winner Determination Problem**. We provide a definition of the auctioneer's decision problem that selects, among the received bids, the revenue-maximising *ordered sequence* of SCOs to perform. More precisely, this definition, besides fulfilling the semantics of the newly introduced bidding language, provides a

sequence of SCOs that is feasible. A feasible sequence guarantees that every SCO can be performed whenever the preceding SCOs in the sequence are run. Moreover, the definition of WDP also allows to specify the quantity of goods initially available (the stock), and the quantity of goods the auctioneer aims to end up with.

(3) **Winner Determination Problem Solvers**. We provide three different ILP-based solvers to deal with the practical solution of the WDP for MMUCA.

 (a) We succeed in mapping the auctioneer decision problem to a CMWOSP (analogously to the case of MUCRAtR). In this way, we can import a body of analysis tools. In this case as well, by relying on these analysis tools, we obtain a very efficient way of solving the decision problem. Nonetheless, this method can only be applied when the supply chain operations do not form a cycle within the production process (acyclic supply chain network). We shall refer to this as the *CMWOSP-based* solver.

 (b) Afterwards, we show that limiting the supply chain network to be acyclic prevents MMUCA's application to many significant scenarios. Thus, we provide a new Integer Linear Programming solver, called DIP, that solves the winner determination problem in the general case.

 (c) Although very general, the introduced method is computationally hard to solve, and therefore hinders the applicability of MMUCA to small-size and middle-size scenarios. In order to overcome such a problem, we introduce a new solver, called CCIP, that exploits some domain specific knowledge to reduce the search space.

Finally, we provide two empirical evaluations. The first one empirically quantifies the scalability gain provided by the CCIP solver with respect to the DIP solver in terms of computational time and size of solvable instances. The second one assesses the performances of the CMWOSP-based solver. We test such methods on acyclic instances and then we compare the obtained results with the results for DIP and CCIP of the former experiment.

Finally, we claim that MMUCA generalises and extends a wide range of auction types, namely:

- single-unit direct, reverse and double auctions (Krishna, 2002);
- multi-unit direct, reverse, and double auctions (Krishna, 2002);
- multi-unit combinatorial direct, reverse and double auctions (Sandholm et al., 2002);
- MUCRAtR (the first contribution of this dissertation); and
- Combinatorial Auctions for Supply Chain Formation (Walsh and Wellman, 2003).

Therefore, all the results that we can derive for MMUCA can be directly applied to the above-listed auction types.

1.6 Dissertation Outline

The remaining of this dissertation is organised as follows.

Chapter 2. We provide some background knowledge on Integer Programming (IP), Place Transition Nets, order and graph theory. This chapter is needed for understanding the concepts developed in chapters 3, 4, 6, and 7.

Chapter 3. We put in context our work with respect to the state of the art. Our work is placed at the intersection of two sub-areas, combinatorial auctions and supply chain management. Thus, firstly we introduce auctions and combinatorial auctions. In particular we elaborate on bidding languages, winner determination problem and test suites. Next, we explore some aspects related to supply chain management. In particular, the problem of centralised supply chain formation and centralised supply chain scheduling and planning are thoroughly described.

Chapter 4. We present Multi-unit combinatorial auctions with transformability relationships among goods (MUCRAtR). This is an extension to Combinatorial Auctions that allows to solve the *make-or-buy* decision problem. In this chapter, we also introduce Weighted Place Transition Nets (WPTN) and we define a new optimisation problem on them, the Constrained Maximum Weighted Occurrence Sequence Problem (CMWOSP). Finally, we show that the CMWOSP on acyclic nets can be solved by means of IP. The material contained in this chapter has been published in:

- Giovannucci, A., Rodriguez-Aguilar, J. A. and Cerquides, J. *Auctioning substitutable goods*. Volume 131 of *Lecture Notes in Artificial Intelligence*, pages 381–388.

- Giovannucci, A., Rodríguez-Aguilar, J. and Cerquides, J. *Multi-unit combinatorial reverse auctions with transformability relationships among goods*. Proc. Workshop on Internet and Networking Economics (*WINE 2005*), pages 858–867. Volume 3828/2005 of Lecture Notes in Computer Science. Springer-Verlag.

- Giovannucci, A., Rodríguez-Aguilar, J. and Cerquides, J. *Multi-unit combinatorial reverse auctions with substitutability relationships among goods*. Proc. of the first Networking and electronic commerce research conference (*NAEC 2005*), pages 324–337. Riva del Garda, Italy, 2005.

- Giovannucci, A., Rodriguez-Aguilar, J. A. and Cerquides, J. *Benefits of combinatorial auctions with transformability relationships*. Proc. of the 17th european conference on artificial intelligence (*ECAI 2006*), pages 717–718. Riva del Garda, Italy, 7/2006.

- Giovannucci, A., Rodriguez-Aguilar, J. A. and Cerquides, J. *Savings in combinatorial auctions through transformation relationships*. In O. Sheory and M. Fasli, editors, The TADA AMEC joint workshop at aamas 2006 trading agent design and analysis & agent mediated electronic commerce VII, *Lecture Notes* *in Computer Science*. Hakodate, Japan, 5/2006, pages 17–30, volume 4452/2007 of Lecture Notes in Computer Science.

- Giovannucci, A., Rodriguez-Aguilar, J. A. and Cerquides, J. *Auctioning transformable goods*. Proc. of the fifth international joint conference on autonomous agents and multi-agent systems (*AAMAS 2006*), pages 893–895. Hakodate, Japan, 5/2006.

Chapter 5. We provide a further extension of combinatorial auctions, the *Mixed Multi-unit Combinatorial Auction* (MMUCA). By means of MMUCA, an auctioneer can automate *make-or-buy-or-collaborate* decisions. In particular, we provide an expressive bidding language and a definition of the winner determination problem for MMUCA. The material contained in this chapter has been published in:

- Cerquides, J., Endriss, U., Giovannucci, A. and Rodriguez-Aguilar, J. A. *Bidding languages and winner determination for mixed multi-unit combinatorial auctions*. Proc. of the 20th intl. joint conferences on artif. intelligence (*IJCAI 2007*), pages 1221–1226. Hyderabad, India, 1/2007.

Chapter 6. Analogously to chapter 4, we provide a mapping of the MMUCA WDP to CMWOSP. With this purpose we introduce the *Mixed Auction Net*, a WPTN that compactly represents the whole search space associated to the MMUCA WDP. We show the equivalence between the MMUCAs WDP and a CMWOSP on the *Mixed Auction Net*. As a consequence of this mapping, we obtain an IP formulation of the WDP for a wide class of supply chain network topologies (acyclic). After showing that the hypothesis that the underlying supply chain is acyclic sometimes may not hold, we introduce a general IP model of the MMUCA WDP that deals with any network topologies, namely the DIP. DIP is built applying a direct mapping of the definition of the WDP to IP. The result is a solver that can find a solution to any instance of WDP. The material explained in this chapter has been published in:

- Giovannucci, A., Rodriguez-Aguilar, J., Cerquides, J. and Endriss, U. *Winner determination for mixed multi-unit combinatorial auctions via petri nets*. Twentieth International Conference on Autonomous Agents and Multi Agent Systems (*AAMAS 2007*). Hawaai, USA, 5/2007. To appear.

Chapter 7. We present the CCIP, an IP formulation of the MMUCA WDP that boosts DIP. The new model exploits the precedence relationships among the SCOs to enforce an a-priori ordering of the solution. In this way, we can prune a great part of the search space. In this chapter we formally prove that CCIP is correct.

Chapter 8. The aim of this chapter is to empirically evaluate and compare the solvers presented in chapters 6 and 7. For this purpose, firstly we describe the state-of-the-art methodology for generating an MMUCA benchmark (Vinyals, 2007b). Then, we perform some preliminary experiments to compare the performances in terms of CPU time of the three solvers.

Chapter 9. We draw some conclusions and thoroughly describe paths to future research.

Chapter 2

Mathematical Background

In this chapter we introduce some technical background knowledge in order to ease the understanding of this dissertation. In section 2.1 we summarise what Integer Programming is, why it is useful for our purposes, and we argue on its pros and cons. Next, in section 2.2 we briefly recall what multisets are, and we discuss some of their properties. Next, in section 2.3, we thoroughly describe Petri Nets (PNs), and in particular Place Transition Nets (PTNs), a formalism for analysing and simulating discrete dynamical systems. Finally, in section 2.4, we summarise some properties of graphs and binary relations from the perspective of order theory.

2.1 Linear and Integer Programming

In this section we introduce some basic concepts regarding linear and integer programming. Both are widely employed for solving complex optimisation problems. The former can solve bigger problems (in terms of decision variables) but is limited in its expressiveness (only linear function can be employed), whereas the latter is more complex but allows to solve a wider class of problems.

2.1.1 Linear Programming

Linear programming has been considered one of the technological breakthroughs of the mid-20th century (Hillier and Lieberman, 1986). This standard tool has saved thousands or even millions of dollars to companies that have employed it. At the heart of linear programming lies the problem of "allocating *limited resources* among *competing activities* in the best possible (i.e *optimal*) way." (Hillier and Lieberman, 1986). In particular, linear programming helps in determining the level of each resource that is allocated to each activity. This pattern applies to several real-world problems such as allocation of production facilities to products, portfolio selection, shipping partners selection, etc.

Linear programming employs mathematical models to represent the above-mentioned problems. In particular, the adjective *linear* illustrates the fact that only linear functions can be employed to model problems. The word *programming* is intended as semantically equivalent to *planning*. Hence, *linear programming* "involves the planning of activities to obtain an optimal result, i.e., a result that reaches the specified best goal among all the feasible alternatives." (Hillier and Lieberman, 1986).

A very interesting characteristic of linear programming is that there exists a very efficient solution method called the *Simplex Method* (Papadimitriou and Steiglitz, 1982). In particular, the simplex method can be applied to problems of enormous size. Notice that it has been shown that linear programming is in the class of the polynomial algorithms (Papadimitriou and Steiglitz, 1982).

In what follows we present an example of linear programming model.

Example 2.1. A farmer has a piece of farm land, say A square kilometres large, to be planted with either wheat or barley or some combination of the two. The farmer has a limited permissible amount F of fertiliser and P of insecticide which can be used, each of which is required in different amounts per unit area for wheat (F_1, P_1) and barley (F_2, P_2). Let S_1 be the selling price of wheat, and S_2 the price of barley. If we denote the area planted with wheat and barley by x_1 and x_2 respectively, then the optimal number of square kilometres to plant with wheat vs barley can be expressed as a linear programming problem:

$$\text{maximise } S_1x_1 + S_2x_2 \qquad \text{revenue bound or } \textit{objective function} \tag{2.1}$$

$$\text{subject to } x_1 + x_2 \leq A \qquad \text{limit on total area} \tag{2.2}$$

$$F_1x_1 + F_2x_2 \leq F \qquad \text{limit on fertiliser} \tag{2.3}$$

$$P_1x_1 + P_2x_2 \leq P \qquad \text{limit on insecticide} \tag{2.4}$$

$$x_1 \geq 0,\ x_2 \geq 0 \qquad \text{cannot plant a negative area} \tag{2.5}$$

□

The linear program in example 2.1 can be directly solved by commercial or free solvers, like ILOG CPLEX (ILOG, 2007), LINDO (Lindo Systems Inc., 2007) (commercial), and GLPK (Makhorin, 2001) (free). The reader can understand the reasons of the tremendous impact of linear programming in recent decades: even knowing little of the technical details it is possible to solve massive and highly complex optimisation problems.

2.1.2 Integer Programming

One key limitation of linear programming is the fact that variables are allowed to take on any fractional value. In some circumstances, this does not constitute a great problem. For instance, if the result of the optimisation is that we have to build 400.5 bicycles, rounding the result to 400 does not change the result a lot. Instead, if the result is to employ 2.5 Boeing airplanes to perform a shipping, the rounding to 2 or 3 air planes is not an easy decision. Moreover, in some problems the solution makes sense only if some variables take on an integer value. Whenever the only deviation from a linear programming approach is the fact the variables can only hold integer values, we have the so called *Integer Programming*.

The modelling language underlying Integer Programming is exactly equivalent to the one of linear programming, except that some variables are constrained to be integer. In particular, if all the variables involved in a problem have to be integer, then we talk about *pure integer programming*, whereas if both integer and fractional variables are allowed we talk about *mixed integer programming*.

Other problems for which the use of integer programming is fundamental are the problems involving interrelated "yes-or-no decisions". For instance, should we make an investment? Should we buy a new truck? And so on.

In what follows we present an example of integer program. It explains how to model the knapsack problem (Kellerer et al., 2004).

Example 2.2 (Knapsack problem)**.** Given a set of items, each with an associated cost and value, the *knapsack problem* consists in determining the subset of items to include in a collection so that the total cost is less than a given limit and the total value is as large as possible. It is a very typical combinatorial problem, and can be easily expressed by means of integer programming.

Say that there are n items. Each item is indexed by $i \in [1, n]$. Then, say that each item i has associated the value v_i and the cost c_i. The problem is thus finding the set of items that costs less than a constant C and maximises the value. In order to model this decision we assign a variable x_i to each item i. x_i takes on value 1 if item i is selected and 0 otherwise. Then, the function that we have to maximise is the value associated to the selected items, namely:

$$\sum_{i=1}^{n} x_i v_i \tag{2.6}$$

this is called the *objective function* of the integer program.

Additionally, we have to make sure that the cost of the selected items does not exceed the permitted cost C. Then, the following constraint must hold:

$$\sum_{i=1}^{n} x_i c_i \leq C \tag{2.7}$$

These are called the *side constraints* of the integer program.

□

In the previous example the decision variables can only take on values 0 or 1 (it would make no sense accepting 0.33 of an item). All the problems sharing this feature are called *binary integer programming* problems and the corresponding variables are called *binary variables*. Analogously to linear programming, a lot of software packages are available to solve integer linear programming problems as well. For instance, Excel (Microsoft, 2007), ILOG CPLEX (ILOG, 2007), LINDO (Lindo Systems Inc., 2007) are commercial solvers, whereas GLPK (Makhorin, 2001) is a free solver.

The case presented in example 2.2 is very simple. However, integer programming models are often very difficult to formalise, since many different type of decision variables and constraints are required. Modelling languages are useful for easing the implementation of such complex models. There exist a few modelling languages, the most famous being AMPL (Fourer et al., 1989), MathProg (Makhorin,), and OPL (Van Hentenryck, 1999). For a survey on modelling languages refer to (Kallrath, 2004).

Solving Integer Programming Problems

It may be tempting to think that Integer Programming problems are easier to solve than Linear Programming problems. In fact, one could argue that since the decision variables can only hold few values instead of real values the search space is reduced, that only a finite number of solutions have to be enumerated. It is possible to demonstrate that this argument is not valid: solving integer programming problems is much more difficult than solving linear programming problems in most cases (Papadimitriou and Steiglitz, 1982)[1].

For this reason, most algorithms for solving Integer Programming incorporate the simplex method as a solution step. We will not get into those details, for a detailed treatment of the subject refer to (Papadimitriou and Steiglitz, 1982) and (Hillier and Lieberman, 1986). The only aspect we aim at highlighting is that the two factors determining the computational hardness of an integer programming problem instance are (Hillier and Lieberman, 1986):

(1) the number of integer decision variables; and

(2) any special structure in the problems.

Current solution methods and commercial solvers can deal with problems ranging from hundreds to thousands of decision variables. The structure of the problem can sometimes make smaller problems much more difficult to solve than bigger ones. In general, reducing the number of constraints can help as well, although with a minor effect.

We briefly mention that huge instances of integer programming can not be solved optimally. For this reason, meta-heuristics have been recently employed to solve those huge problems non-optimally. Even if they can not guarantee any bound on their performances, they usually perform quite well. For a review on meta-heuristics refer to (Blum and Roli, 2003).

2.2 Multi-sets

A *multi-set* (Blizard and File, 1988; Syropoulos,) is an extension to the notion of set, considering the possibility of *multiple appearances* of the same element. An example of multiset is[2] $\mathcal{A} = \{a, a, a, b, b, c\}$.

In general, a *multi-set* $\mathcal{A}$ over a set X is a function $\mathcal{A} : X \rightarrow \mathbb{N}$ mapping X to the cardinal numbers. In the example above $X = \{a, b, c\}$.

For any $x \in X$, $\mathcal{A}(x) \in \mathbb{N}$ is called the *multiplicity* of x. For instance, in the example above, the cardinality of the element a is 3 ($\mathcal{A}(a) = 3$).

[1]There are some particular problems having a special structure that makes them as easy as a linear program.

[2]Henceforth we employ calligraphic letters to indicate multi-sets.

There are different ways of denoting multisets. We will show them by means of the three representations below:

- $\mathcal{A} = \{a, a, a, b, b, c\}$
- $\mathcal{A} = \{(a, 3), (b, 2), (c, 1)\}$
- $\mathcal{A} = \{3'a + 2'b + 1'c\}$

An element $x \in X$ *belongs* to the multi-set $\mathcal{A}$ if $\mathcal{A}(x) \neq 0$ and we write $x \in \mathcal{A}$. We denote the set of multi-sets over X by $\mathbb{N}^X$.

The total number of elements in a multiset, including the repetitions is the *cardinality* associated to the multiset. The cardinality of multisets is denoted in the same way as in the case of sets. For instance, $|\mathcal{A}| = 6$ in the example above.

2.2.1 Operations on Multisets

In what follows we list the operations between multisets. Given two multisets $\mathcal{A}, \mathcal{B} \in \mathbb{N}^X$, we have the following operations or relations among them:

- *sum*: $\mathcal{A}(x) \uplus \mathcal{B}(x) = \mathcal{A}(x) + \mathcal{B}(x) \quad \forall x \in X$
- *intersection*: $\mathcal{A}(x) \cap \mathcal{B}(x) = min(\mathcal{A}(x), \mathcal{B}(x)) \quad \forall x \in X$
- *union*: $\mathcal{A}(x) \cup \mathcal{B}(x) = max(\mathcal{A}(x), \mathcal{B}(x)) \quad \forall x \in X$
- *subset*:$\mathcal{A}(x) \subseteq \mathcal{B}(x) \rightarrow \mathcal{A}(x) \leq \mathcal{B}(x) \quad \forall x \in X$

2.3 Petri Nets

In this section we introduce and describe carefully the Petri Nets formalism. Petri Nets are a powerful mathematical and graphical tool for the description of discrete distributed systems. Petri Nets (PNs) were firstly introduced in 1962 by Karl Adam Petri in his seminal dissertation ((Petri, 1966) in English and (Petri, 1962) in German). In particular PNs are suitable for describing systems in which parallelism, concurrency, and synchronisation play an important role. For a very good review on Petri nets, refer to (Murata, 1989).

Petri Nets can provide some distinctive advantages with respect to other approaches (Reisig, 1985) like finite state machines:

- Causal dependencies or independence among the different components of the system can be explicitly represented.
- They allow to describe a system that is not inherently sequential.
- They can represent different levels of abstraction without having to change the description language. These abstraction levels range from the representation of a single bit in a PC to the representation of the PC in its environment within the same framework.

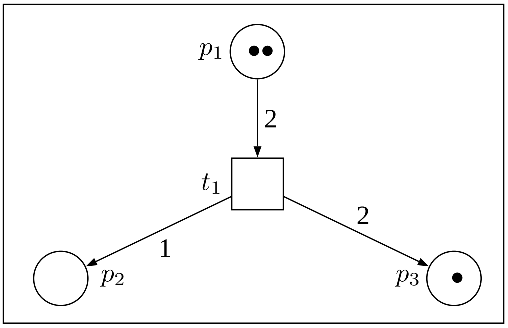


Figure 2.1: Example of a Place Transition Net.

- They provide a set of formal tools useful to analyse and describe discrete dynamical systems. For instance, it is possible to verify several system properties as *deadlock avoidance, boundedness*, etc.

I will not get into the details of all those properties. We only remark that a vast amount of analysis tools (Murata, 1989) are available for Petri nets. Thus, everything that is modelled by means of Petri nets can directly employ all those tools.

An example of Petri net is shown in figure 2.1. A PN is a bipartite graph: it has *place* nodes, *transition* nodes, and directed arcs connecting places to transitions and transitions to places. The places connected to a transition by means of input arcs are called the *input places* of the transition, and the ones connected by outgoing arcs from the transition are the *output places* of the transition. Places contain tokens. The graphical representation of a PTNS is composed of the following graphical elements: places are represented as circles, transitions are represented as rectangles, arcs connect places to transitions or transitions to places, and an *arc expression function* E labels arcs with values.

Different classes of Petri Nets exist. A survey of the different existing types of Petri nets is made in (Bernardinello and de Cindio, 1992). In this work, three levels of Petri nets are identified:

(1) *Level 1* nets whose places can contain at most one token;

(2) *Level 2* nets whose places can contain more than one token; and

(3) *Level 3* nets whose tokens are labelled by a type (tokens of different type within the same place can be distinguished).

We will focus on a particular *level 2* net called Place Transition Net (PTN). Place/Transition Nets are Petri Nets characterised by multiple tokens in the same place and arc weights[3]. More formally, following (Murata, 1989),

Definition 2.1 (Place/Transition Net Structure)**.** A *Place/Transition Net Structure* (PTNS) is a tuple $N = (P, T, A, E)$ such that:

[3]Actually, there should be a limit on the capacity of each place in term of contained tokens. However, it is not crucial in our work and we can set it to infinite.

(1) P is a set of *places*;

(2) T is a finite set of *transitions* such that $P \cap T = \emptyset$;

(3) $A \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*;

(4) $E : A \rightarrow \mathbb{N}^+$ is an *arc expression* function (it represents the weights associated to the arcs, standing for the number of input/output tokens consumed/produced by the transition).

□

Furthermore, we have that $t^\bullet = \{p \in P \mid (t, p) \in A\}$ are the *output places* of t, and that $^\bullet t = \{p \in P \mid (p, t) \in A\}$ are the *input places* of t.

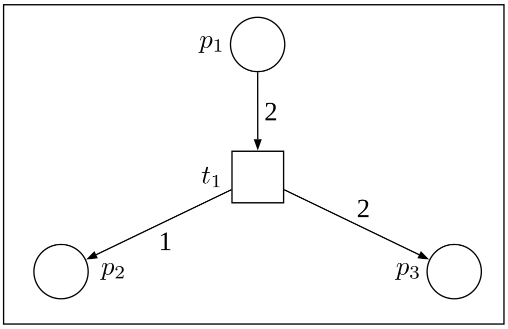


Figure 2.2: Example of a Place Transition Net Structure.

Example 2.3. In figure 2.2 we illustrate a PTNS defined as:

(1) $P = \{p_1, p_2, p_3\}$ is the set of *places*;

(2) $T = \{t_1\}$ is the set of *transitions*;

(3) $A = \{(p_1, t_1), (t_1, p_2), (t_1, p_3)\}$ is the set of *arcs*;

(4) $E(p_1, t_1) = 2; E(t_1, p_2) = 1; E(t_1, p_3) = 2;$ is the *arc expression function*.

Moreover, the input and output places of t_1 are:

$$t_1^\bullet = \{p_2, p_3\} \tag{2.8}$$

$$^\bullet t_1 = \{p_1\} \tag{2.9}$$

□

A distribution of tokens over the set of places is called a *marking*, and it stands for the state of the Petri net.

Definition 2.2 (Marking)**.** A *marking* $\mathcal{M} : P \rightarrow \mathbb{N}$ of a PTNS is a multiset over P. $\mathcal{M}(p) = k$ means that place $p \in P$ contains k tokens for marking $\mathcal{M}$.

□

Example 2.4. The marking $\mathcal{M}_0$ of figure 2.1 is

$$\mathcal{M}_0(p_1) = 2 \tag{2.10}$$
$$\mathcal{M}_0(p_2) = 0 \tag{2.11}$$
$$\mathcal{M}_0(p_3) = 1 \tag{2.12}$$

or equivalently, employing the notation for multisets, we can represent marking in a more compact form:

$$\mathcal{M}_0 = 2'p_1 + 1'p_3 \tag{2.13}$$

□

A PTNS S with a given initial marking $\mathcal{M}_0$ is called a *Place/Transition Net* (PTN) and is noted $(S, \mathcal{M}_0)$.

Given a marking $\mathcal{M}$, we say that a transition is *enabled* if all its input places contain at least as many tokens as required by the the transition's input arcs. If the transition is enable it can *fire* consuming tokens of the input places and producing tokens in the output places. Intuitively, a transition is enabled if enough tokens are present in its input places. In what follows we state more formally the concepts of *enabled transition* and *firing of a transition*.

A transition $t \in T$ is said to be *enabled* if each input place p of t is marked with at least $E(p,t)$ tokens, where $E(p,t)$ represents the weight of the arc connecting p to t. More formally,

Definition 2.3 (Enabled Transition)**.** Given a marking $\mathcal{M}$, a transition $t \in T$ is enabled iff:

$$\mathcal{M}(p) \geq E(p,t) \quad \forall p \in^\bullet t \tag{2.14}$$

□

Example 2.5. For instance in figure 2.1 transition t_1 is enabled in marking $\mathcal{M}_0$ since $E(p_1,t_1) = 2$ and $\mathcal{M}_0(p_1) = 2$, thus $\mathcal{M}_0(p_1) \geq E(p_1,t_1)$.

□

An enabled transition may or may not fire. If it fires, it changes the current marking to a new marking by removing tokens from the input places and putting tokens into the output places. More formally

Definition 2.4 (Firing of an enabled transition)**.** The *firing* of an enabled transition t removes $E(p_i,t)$ tokens from each input place p_i and adds $E(t,p_o)$ tokens to each output place p_o. The firing of a transition t changes marking $\mathcal{M}_{k-1}$ to a marking $\mathcal{M}_k$. The new marking can be computed employing the following equation[4]:

$$\mathcal{M}_k(p) = \mathcal{M}_{k-1}(p) + Z(t,p) \quad \forall p \in {}^\bullet t \cup t^\bullet \tag{2.15}$$

where $Z(t,p) = E(t,p) - E(p,t)$. In this case we write $\mathcal{M}_{k-1}[t > \mathcal{M}_k$ for denoting that the firing of transition t changes the $\mathcal{M}_{k-1}$ marking into the $\mathcal{M}_k$ marking.

[4]Henceforth, for simplicity, we implicitly assume that $E(p,t) = 0$ if $(p,t) \notin A$ and $E(t,p) = 0$ if $(t,p) \notin A$.

□

Example 2.6. Consider figure 2.1, the firing of t_1 in marking $\mathcal{M}_0$ leads to marking $\mathcal{M}_1 = 1'p_2 + 3'p_3$. We illustrate in figure 2.3 the state of the PTN of figure 2.1 after that t_1 fires.

□

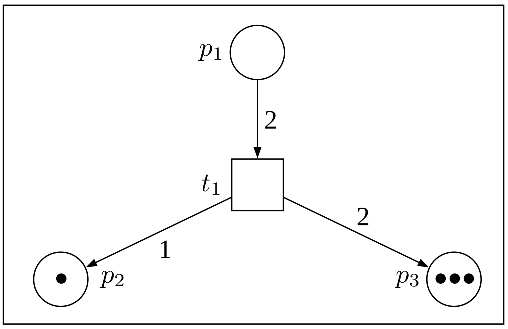


Figure 2.3: Place Transition Net of figure 2.1 after firing t_1.

2.3.1 Reachability

An important property we are interested in is whether we can reach a particular state of a PTN departing from a given initial state. This leads to the definition of *reachability*. Reachability is a fundamental concept that will be widely employed in this dissertation. In this section, we will introduce several concepts related to reachability. Intuitively, given an initial marking $\mathcal{M}_0$, and a final marking $\mathcal{M}_d$, the reachability problem consists in deciding if there exists a sequence of firings leading from $\mathcal{M}_0$ to $\mathcal{M}_d$.

The firing of an enabled transition changes the token distribution (marking) in a net according to the firing rule of definition 2.4. Then, a sequence of firings will result in a sequence of markings.

Definition 2.5 (Reachability)**.** A marking $\mathcal{M}_n$ is *reachable* from a marking $\mathcal{M}_0$ in a PTN structure S if there exists a sequence of firings that transforms $\mathcal{M}_0$ into $\mathcal{M}_n$. $\mathcal{M}_0$ is called the *start marking*, while $\mathcal{M}_n$ is called the *end marking*.

□

All the markings reachable from $\mathcal{M}_0$ in a PTN Structure S are noted as $R(S, \mathcal{M}_0)$, and are called the *reachable set* of a PTN.

Definition 2.6. (Firing Sequence) Given a PTN structure S and a marking $\mathcal{M}_0$, a *firing* or *occurrence sequence* $J : \mathbb{N} \rightarrow T$ is a sequence of transitions:

$$J = \langle t_1,\ t_2, \ldots, t_n \rangle$$

that changes the marking $\mathcal{M}_0$ into the marking $\mathcal{M}_n$. In this case we write $\mathcal{M}_0[J > \mathcal{M}_n$ as a shorthand to represent that the firing sequence J leads from $\mathcal{M}_0$ to $\mathcal{M}_n$.

□

Notice that in a *firing sequence* all the transitions must be enabled and fire with the order established by the very same sequence.

It can be shown that the start and end markings are related by the following equation:

$$\forall p \in P \qquad \mathcal{M}_n(p) = \mathcal{M}_0(p) + \sum_{t \in J} Z(t,p). \tag{2.16}$$

Definition 2.7. The *firing count multi-set* associated to a *firing* or *occurrence sequence* J is a multiset $\mathcal{K}_J \in \mathbb{N}^T$ such that the multiplicity of each transition stands for the number of times it appears in the firing sequence. That is:

$$\mathcal{K}_J(t) = |J^{-1}(t)| \qquad\qquad \forall t \in T \tag{2.17}$$

where $|J^{-1}(t)|$ is the number of times transition t is fired in the firing sequence J.

□

2.3.2 The state equation

In this section we aim at providing an algebraic representation of Petri nets. Such representation will allow us to compactly represent the reachability set in some cases.

For a Petri Net N with r transitions and n places, the *incidence matrix* $A = [a_{ij}]$ is an $r \times n$ matrix of integers. Each entry is given by $a_{ij} = a^+_{ij} - a^-_{ij}$, where $a^+_{ij} = E(t_i, p_j)$ stands for the weight of the arc connecting the t_i transition to its output place p_j, and $a^-_{ij} = E(p_j, t_i)$ stands for the weight of the incoming arc connecting place p_j to transition t_i.

Example 2.7. In the example of figure 2.2, the incidence matrix is:

$$a^- = [2\;0\;0] \tag{2.18}$$

$$a^+ = [0\;1\;2] \tag{2.19}$$

$$A = \begin{bmatrix} -2 & 1 & 2 \end{bmatrix} \tag{2.20}$$

□

It is straightforward that a^+_{ij}, a^-_{ij}, and a_{ij} represent the number of tokens added to, removed from, and changed in place j when transition i fires once.

Notice that in this new representation a transition t_i is enabled in a marking $\mathcal{M}$ iff

$$a^-_{ij} \leq \mathcal{M}(p_j) \quad j = 1, 2, \ldots, n$$

Example 2.8. In the example of figure 2.1, transition t_1 is enabled in marking $\mathcal{M}_0$ since $a^-_{11} = 2 \leq \mathcal{M}_0(p_1)$.

□

In order to obtain an algebraic representation of a Petri net, we can represent a marking $\mathcal{M}_k$ as an $n \times 1$ column vector M_k such that the $j-th$ entry of M_k represents the number of tokens present in place p_j after the $k-th$ firing in some firing sequence

($M_k[j] = \mathcal{M}_k(p_j)$). For instance, the markings of the nets in figures 2.1 and 2.3 can be represented as

$$\mathcal{M}_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

and

$$\mathcal{M}_1 = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

Finally, we define the *firing vector* u_k as an $r \times 1$ column vector of $r - 1$ zeros and one nonzero entry. By setting a a 1 in the $i - th$ position ($u_k[i] = 1$), we indicate that transition t_i fires at the k-th firing. We can now express equation (2.15) in matrix form:

$$M_k = M_{k-1} + A^T u_k \quad k = 1, 2, ... \tag{2.21}$$

Example 2.9. The state equation associated to the firing of transition t_1, transforming the PTN in figure 2.1 into the one in figure 2.3, is:

$$M_1 = M_0 + A^T u_1 = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix} \cdot 1 = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} \tag{2.22}$$

$u_1[1]$ takes on value 1 because transition t_1 is fired once.

□

Say that $\mathcal{M}_n$ is reachable from $\mathcal{M}_0$ via the firing sequence $J = \langle t_1, t_2, \ldots, t_n \rangle$. We represent the transitions in J by means of their firing vectors $\langle u_1, u_2, \ldots, u_n \rangle$. Then, by applying recursively equation (2.21), we obtain:

$$M_n = M_0 + \sum_{k=1}^{n} A^T \cdot u_k = M_0 + A^T \sum_{k=1}^{n} u_k = M_0 + A^T K_J \tag{2.23}$$

where K_J is an $r \times 1$ vector representing the firing count multiset $\mathcal{K}_J$, defined in equation (2.17), namely:

$$K_J[i] = \mathcal{K}_J(t_i) = |J^{-1}(t_i)| \qquad \forall i \in [1, r] \tag{2.24}$$

K_J is the *firing count vector* associated to the firing sequence J.

2.3.3 State equation and reachability

All the results that we report from here to the end of the chapter are taken from (Murata, 1989). Say that $\mathcal{M}_d$ is reachable from $\mathcal{M}_0$, then there exists a firing sequence $\langle u_1, u_2, ..., u_d \rangle$ bringing from $\mathcal{M}_0$ to $\mathcal{M}_d$. Therefore, a *necessary condition on reachability* can be expressed in terms of a matrix equation:

Theorem 2.1. *If $\mathcal{M}_d$ is reachable from $\mathcal{M}_0$, then the following equation has a non-negative integer solution $\mathbf{x}$:*

$$M_d = M_0 + A^T \mathbf{x} \tag{2.25}$$

where $\mathbf{x} = \sum_{k=1}^{d} u_k$ is the $r \times 1$ column vector of non-negative integers we called firing count vector.

$\square$

Notice that the $i-th$ entry of vector $\mathbf{x}$ encodes the number of times a transition t_i must be fired to transform $\mathcal{M}_0$ into $\mathcal{M}_d$.

Equation (2.25) is called the *State Equation*, since it describes the states that a Petri net would reach if the transitions encoded in $\mathbf{x}$ were fired. However, notice that not all the states encoded by the state equation are actually reachable. That means that there may exist solutions to equation (2.25) that are not reachable states of a Petri net. However, it can be shown that sometimes all the states reachable by a Petri net are described by the state equation. In particular, this happens when the net is acyclic.

Before defining the concept of acyclicity, we have to explain what is a cycle. Since a Petri Net is a bipartite graph, a cycle in a Petri net is a sequence of

Definition 2.8. A *directed cycle* in a Petri Net Structure (P, T, A, E) is a sequence of places and transitions $\langle p_1, t_1, p_2, t_2, \ldots, p_n, t_n, p_1 \rangle$ such that $\forall i \in [1, n]$ $(p_i, t_i) \in A$ and $(t_i, p_{i+1}) \in A$.

Definition 2.9 (Acyclicity)**.** A PTNS is said to be acyclic if it does not contain any directed circuit.

$\square$

In (Murata, 1989), it is shown that in an *acyclic* Petri Net, the condition expressed by theorem 2.1 is not only necessary, but also sufficient.

Theorem 2.2. *In an acyclic PTNS, $\mathcal{M}_d$ is reachable from $\mathcal{M}_0$ iff the following equation has a non-negative integer solution in $\mathbf{x}$:*

$$\mathcal{M}_d = \mathcal{M}_0 + A^T \mathbf{x} \tag{2.26}$$

$\square$

That is, if there exists a solution to equation (2.26), a firing sequence reaching $\mathcal{M}_d$ from $\mathcal{M}_0$ is guaranteed to exist, and $\mathbf{x}$ represents its firing count vector.

Moreover, Murata further extends the class of Petri nets for which the condition is still sufficient. These particular nets (trap-circuit and syphon-circuit nets) have special topologies with particular types of circuits. For such nets, the state equation represents all the reachable states if the initial marking $\mathcal{M}_0$ satisfies some constraints. Further efforts have been made for extending the validity of the state equation to more classes of Petri nets (Tarek and Lopez-Benitez, 2004).

2.4 Preliminaries on binary relations and graphs

In this section we recall some definitions about binary relations, graphs, and order relations. In section 2.4.1 we will recall binary relations and some of their properties. In section 2.4.2 we will recall the definition of directed graphs, directed acyclic graph, and we will summarise some concepts and properties related to graphs. In section 2.4.3 we recall some concepts related to order relations, and we connect them to graphs.

All the definitions and theorems contained in this section are taken from (Cormen, 2001), where the interested reader can find the the corresponding proofs.

2.4.1 Relations

In this section we will recall what a binary relation is along with the properties of such relations that we are interested in.

A *binary relation* R on two sets A and B is a subset of the Cartesian product $A \times B$. If $(a, b) \in A \times B$ we write aRb and we say that a is in relation with b. We say that R is a binary relation on A if it is a subset of $A \times A$.

Example 2.10. The *less than* is a binary relation defined on $\mathbb{N}$ as follows

$$\{(a, b) \in \mathbb{N} \times \mathbb{N} : a < b\} \tag{2.27}$$

□

There are special features that are particularly important for binary relations. Thus, a binary relation $R \subseteq A \times A$ is:

- *reflexive*: if $\forall a \in A \; aRa$. For instance, "$=$" and "$\leq$" are reflexive on A, while "$<$" is not.
- *symmetric*: if $aRb \Rightarrow bRa$. For instance, the "$=$" relation is symmetric, while "$\leq$" is not.
- *transitive*: aRb and bRc implies aRc. For instance, "$=$" is transitive.
- *antisymmetric*: if aRb and bRa then $a = b$. For instance, "$\leq$" is antisymmetric.

Equivalence classes

A binary relation that is *reflexive, symmetric* and *transitive* is called an *equivalence relation*. For instance, "$=$" is an equivalence relation, whereas "$<$" is not. If R is an equivalence relation on a set A, then for all $a \in A$ we denote with $[a]$ the set of element in relation with a, and we call it the *equivalence class* of a.

A well known result about equivalence classes is

Theorem 2.3 (An equivalence relation is the same as a partition)**.** *The equivalence classes of any equivalence relation R on a set A form a partition of A, and any partition of A determines an equivalence relation on A for which the sets in the partition are the equivalence class.*

□

2.4.2 Graphs and Paths

In this section, we introduce the definition of graph, path in a graph, and strongly connected components of a graph.

A *directed graph* G is a pair (V, E), where V is a finite set, and $E \subseteq V \times V$ is a binary relation on V. The set V is called the *nodes* or *vertexes* set, while E is the set of *arcs* or *edges*. If $(u, v) \in E$ we say that v is *adjacent* to u. Each edge of the type (u, u) is called a *self-loop*. In figure 2.4 we show the graphical representation of a graph.

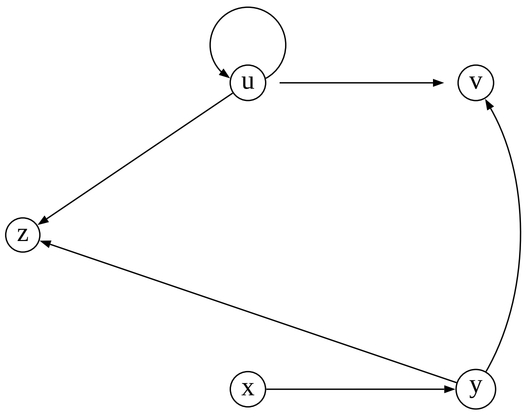


Figure 2.4: Example of a Graph

Definition 2.10 (Path in a graph)**.** A *path* of length k from a vertex v to a vertex v' in a graph (V, E), is a sequence $\langle v_0, \ldots, v_k \rangle$ of vertexes such that $v = v_o$ and $v' = v_k$, and $(v_i, v_{i+1}) \in E$ for $i = \{1, \ldots, k\}$. There is always a 0-length path from v to v. The path is said to be *simple* if all the vertexes in the path are distinct.

□

For instance, in the graph of figure 2.4, $\langle x, y, v \rangle$ is a path from x to v.

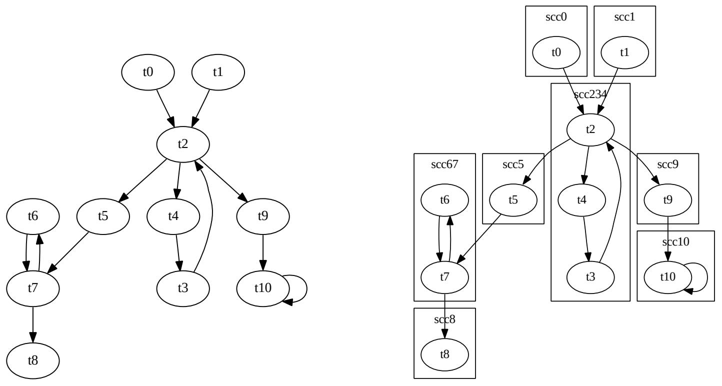
Definition 2.11 (Cycle in a Graph)**.** In a directed graph a path $\langle v_0, \ldots, v_k \rangle$ is a cycle if $v_0 = v_k$ and the path contains at least one edge. A cycle is *simple* if all the vertexes $\langle v_1, \ldots, v_k \rangle$ are distinct. A *self-loop* is a cycle of length one.

□

For instance, in the graph of figure 2.4, $\langle u, u \rangle$ is a self-loop.

Strongly Connected Components

A directed graph is *strongly connected* if for every pair of vertexes u and v there is a path from u to v and a path from v to u, i.e. if every two vertexes are connected by a directed path. The *strongly connected components* (SCC) of a graph are the *equivalence classes* of the vertexes under the "are mutually reachable" relation, or equivalently its maximal strongly connected sub-graphs (Cormen, 2001), (Harary, 1999). Figure 2.5(b) shows the SCCs of the graph in figure 2.5(a).



(a) A graph (b) SCCs of the graph

Figure 2.5: A graph and the corresponding SCCs

More formally, given a graph (V, E), we define a relation $R \subseteq V \times V$ such that uRv iff there exists a directed path from u to v and a path from v to u. It is easy to check that this relation is reflexive[5], transitive, and symmetric. Thus, it is an *equivalence relation*. The equivalence class associated to an element u is such that:

$$[u] = \{v \in V : \text{ exists a path from u to v and a path from v to u}\} \tag{2.28}$$

2.4.3 Order relations

A relation that is antisymmetric, reflexive and transitive is a *partial order*, and we call a set on which a partial order is defined a *partially ordered set*. In a partial order it is possible to have some elements that are not in relation among them. Then, a partial order R on a set A is a *total* or *linear order* if for all $a, b \in A$ we have aRb or bRa. Notice that, given a directed acyclic graph, we can define a partial order. The partial order is such that a node u *comes before* a node v if there exists a directed path from u to v. The relation is a partial order since it is trivially transitive and reflexive. It is also antisymmetric since the acyclicity hypothesis implies that if there is a path from a node u to a node v there cannot be a path from v to u without having a cycle.

Example 2.11. Consider the graph of figure 2.4 without the self-loop, it represents a partial order such that:

$$u < z \qquad u < v \tag{2.29}$$
$$x < y \qquad y < z \tag{2.30}$$
$$y < v \qquad x < z \tag{2.31}$$
$$x < v \tag{2.32}$$

□

[5]Recall that there exists always a 0-length path from a node to itself.

What happens when there is a cycle in the graph? Obviously, in this case the relation *exists a path* does not define anymore a partial order. We lose one property, the antisymmetric property. In fact, if a cycle is present, all the nodes along the cycle will be mutually connected by a two-way directed path. Consider for instance the graph of figure 2.5, which is the order among transitions t_2, t_3 and t_4? In such a case it is not possible to define an order among them. This type of relationship is called a *preorder* (Davey and Priestley, 2002).

More formally, a relation that is reflexive and transitive is a *preorder*. Normally, this is due to the presence of a cycle in the precedence relation. In order theory, a preorder is noted as $\lesssim$. Thus we will write $u \lesssim v$ when a path exists from v to u.

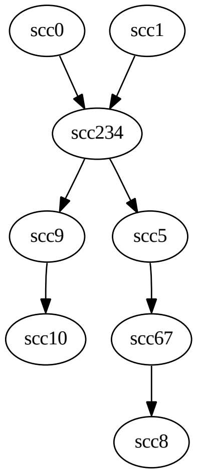


Figure 2.6: The strict order $\prec$

Given a preorder $\lesssim$ on a set V, several interesting properties (Davey and Priestley, 2002) hold:

- the relation $\sim$ on V such that $v \sim u$ if and only if $v \lesssim u$ and $v \lesssim u$ is an *equivalence relation*
- the relation $\prec$ on the quotient set $V/_{\sim}$ such that $[v] \prec [u]$ iff $v \lesssim u$ is a (strict) partial order. Intuitively, this operation eliminates the cycles by collapsing each SCC to a single element. We say then that the cyclic graph collapses into an acyclic graph,
- the equivalence classes defined by $\sim$ are the Strongly Connected Components (SCC) of the graph associated to the relation. Thus, from now on we will employ equivalently the terms SCC and equivalence class.

Figure 2.6 graphically represents the strict preorder relation $\prec$ corresponding to the graph of figure 2.5(b).

Chapter 3

Related Work

In this chapter, we recall some related work. The problem dealt within this dissertation is indeed original, and, to the best of our knowledge, has been only marginally treated by different disciplines like operations research, artificial intelligence, manufacturing engineering, and economics. Although a massive amount of work has been devoted to cope with different aspects of the *make-or-buy* or *make-or-buy-or-collaborate* problems, to the best of our knowledge nobody has entirely treated them. In this chapter we will summarise the literature close to our problem.

The chapter is organised as follows. In section 3.1, we will recall some basic concepts about auctions. Next, in section 3.2, we will thoroughly explain combinatorial auctions, a particular type of auctions. Then, in sections 3.3, we will introduce the problems of supply chain scheduling and supply chain formation respectively. Next, in section 3.4, we will put in relation the work presented in this dissertation with respect to the state of the art.

3.1 Auctions

The most employed definition of auction is due to McAfee et. al (McAfee and McMillan, 1987): ”*An auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants*”.

Auctions play an important role in economics. In their most basic form, they are one of the ways in which various commodities, financial assets and concession rights are allocated to individuals and firms, particularly in a market-oriented setting. Some very famous examples of auction houses are Sotheby’s (Sotheby’s, 2007), Christie’s (Christie’s, 2007), and Ebay (Ebay, 2007).

The introduction and use of auctions is motivated by the fact that the value of an item (or of a set of items) is often not known a-priori. Then, an auction is a way to “let the market decide” the value associated to the item. It is a very flexible mechanism that is employed with several different variations. Furthermore, it is dynamic, since it allows a meaningful interaction between buyers and sellers. From our point of view, the distinguishing feature of auctions is that they support full automation. In fact, they are a mechanism with predetermined rules. Hence, they are ideal for computer implementation. Finally, they are in most cases economically efficient (Milgrom, 2004).

3.1.1 Taxonomy of Auctions

Klemperer (Klemperer, 2004) classifies auctions in four basic groups based both on the modality the auction is run with and on the associated payment rule:

(1) the *ascending-bid* auction, also known as English or Open-outcry;

(2) the *descending-bid* auction, also known as Dutch;

(3) the *first-price sealed-bid* auction; and

(4) the *second-price sealed-bid* auction, also known as Vickrey auction.

In the ascending-bid auction, the price is raised successively until only one bidder remains. Such bidder wins the object and pays the final price.There are two variants of this auction. One (called Japanese) considers that the price is raised by the auctioneer, and the bidders that are not willing to pay the corresponding price at a given round quit the auction. The other one, known as English, let the bidders *call out* the prices.

The descending auction works in exactly the opposite way, the price starts at a very high price and it is successively decremented until some bidder expresses his willingness to accept that price.

In the first-price sealed-bid auction, all the bidders submit their offer without seeing the other bidders' offers. The bidder offering the highest bid wins paying his bid (that is the highest price, whereby the name *first-price*).

In the second-price sealed-bid, the process is similar, with the exception that the bidder pays the price offered in the second highest bid, whereby the name *second price*.

Another classification can be done based on the number of buyers and sellers, namely:

- *direct* auction when there is one seller and multiple buyers;
- *reverse* auction when there is one buyer and multiple sellers. In this case the item at auction is bought and not sold; and
- *double* auction when there are multiple buyers and multiple sellers.

Finally, a classification can be done based on the quantity of items sold/bought and on the features of the items (i.e. price is not the only discriminant of the value associated to an item). In this case we shall refer to *multidimensional auctions*. There are different types of multidimensional auctions:

- *multi-unit* auctions when multiple identical items are bought/sold;
- *multi-attribute* auctions when the value associated to an object is determined by a set of features (shipping time, quality, and so on); and

- *multi-item* or *combinatorial* auctions when multiple distinguishable items are bought/sold.

There exists a lot of hybrid auctions joining the features of different auction classes. In particular, very relevant to our work are (Cramton et al., 2006):

- combinatorial reverse auctions;
- multi-unit combinatorial reverse auctions.; and
- multi-unit combinatorial auctions.

Since combinatorial auctions (CAs) are of central importance in our work, in what follows we provide a detailed account on the state-of-the-art in CAs.

3.2 Combinatorial Auctions

A combinatorial (reverse) auction. (Cramton et al., 2006) is an auction where bidders can sell (buy) entire bundles of goods in a single transaction. Although computationally very complex (Sandholm et al., 2002), the fact that bidders can express their preferences over bundles of goods may help an auctioneer obtain better deals. In fact, buying items in bundles has the great advantage of eliminating the risk for a bidder of not being able to sell/buy complementary items at a reasonable price in a follow-up auction (think of a combinatorial auction to acquire a pair of shoes, as opposed to two consecutive single-item auctions for each of the individual shoes). Indeed, combinatorial auctions may lead to more efficient allocations whenever complementarities among the goods at auction hold. For a detailed survey on CAs refer to (Cramton et al., 2006; de Vries and Vohra, 2003; Kalagnanam and Parkes, 2003).

CAs have a high potential to be employed as an allocation mechanism in a wide variety of real-world domains. They have been proposed to be employed for allocating loads to trucks in the transportation market (Caplice and Sheffi, 2006), routes to buses (Cantillon and Pesendorfer, 2006), goods/services to buyers/providers in industrial procurement scenarios (Bichler et al., 2006), airport arrival and departure slots (Ball et al., 2006), and radio-frequency spectrum for wireless communications services (Pekec and Rothkopf, 2003). Walsh in (Walsh et al., 2000) employed them for supply chain formation.

The study of the mathematical, game-theoretical and algorithmic properties of combinatorial auctions has recently become a popular research topic in AI. This is not only due to their relevance to important application areas such as electronic commerce or supply chain management, but also to the range of deep research questions raised by this auction model.

In the last decades, different topics related to CAs have been considered, namely the design of auction mechanisms, bidding languages, and algorithms for the Winner Determination Problem. In the following sections, we summarise the most relevant contributions on those topics.

3.2.1 Mechanism Design

Auction theory studies the formal properties of auctions as shown in the surveys of (Krishna, 2002) and (Milgrom, 2004). Nonetheless CAs have recently attracted the attention of economists and game theorists. Associated to auction theory is also the design of auction *mechanisms*, devoted to study *how* to run an auction in order to guarantee some economic properties such as, for instance, efficiency, incentive compatibility, individual rationality, etc. For instance, (Ausubel and Milgrom, 2006b), (Parkes, 2006), (Ausubel and Milgrom, 2006a), (Cramton, 2006), (Ausubel et al., 2006), and (Land et al., 2006) describe some mechanisms for CAs.

3.2.2 Bidding Languages

Bidding is the process of transmitting one's valuation function over the set of goods on offer to the auctioneer (or rather *some* valuation function — the bidders are of course not required to reveal their true valuation —). In principle, it does not matter how the valuation function is being encoded, as long as sender (bidder) and receiver (auctioneer) agree on the semantics of what is being transmitted, *i.e.* as long as the auctioneer can understand the message(s) sent by the bidder. Indeed, it is possible to fully specify an auction mechanism (allocation and pricing rules) without reference to a concrete bidding language. In practice, however, the choice of a bidding language is of central importance.

Early work on combinatorial auctions has typically ignored the issue of bidding languages. The standard assumption used to be that if a particular bidder submits several atomic bids (a bundle together with a proposed price), then the auctioneer may accept any set of bids from that bidder for which the bundles do not overlap, and charge the sum of the specified prices. This is now sometimes called the *OR language*. But other interpretations of a set of atomic bids are possible. For instance, we may take it to mean that the auctioneer may accept at most one bid per bidder; this is now known as the *XOR language*.

The first systematic study of bidding languages is due to Nisan (Nisan, 2006) (an early version (Nisan, 2000) appeared in 2000). Nisan's papers provide an excellent introduction to the topic and clarify a number of issues that had previously remained somewhat fuzzy. Nisan classifies several types of bidding languages, providing expressiveness results for each of them. At the basis of his exposition lies the concept of *atomic bid*. Formally, an atomic bid is a pair (S, p), where S is a subset of the items at auction, and p is the price a bidder is willing to pay to obtain the goods in S. By combining in different ways atomic bids we obtain several bidding languages. The most widely employed are:

- *OR*. Each bidder submits an arbitrary number of atomic bids. The auctioneer is allowed to accept any disjoint subset of them.

- *XOR*. Each bidder submits an arbitrary number of atomic bids. The auctioneer is allowed to accept at most one among them.

- *OR-of-XOR*. Each bidder can submit any number of XOR bids. The auctioneer is allowed to accept any subset of these bids.

- *XOR-of-OR*. Each bidder can submit an arbitrary number of OR bids. The auctioneer is allowed to accept at most one of these bids.

Consider the following example explaining the semantics of OR and XOR bids.

Example 3.1. Say that the set of goods at auctions is $\{A, B, C\}$. Then, we have:

- *OR*: $(\{A\}, 3)$ OR $(\{B, C\}, 3)$ means that if the bidder is allocated $\{A, B, C\}$, then he will pay 6.

- *XOR*: $(\{A\}, 3)$ XOR $(\{B\}, 3)$ XOR $(\{A, B\}, 5)$ means that if he is allocated $\{A, B\}$ he will pay 5 (not 6).

Another interesting paper about bidding languages is (Boutilier and Hoos, 2001), where Boutilier et al. present a logical bidding language that allows the expression of complex utility functions in a natural and concise way. In this language bids are given by propositional formulae whose sub-formulae can be annotated with prices, thus allowing for a natural and concise formulation of bidders' utility functions.

To the best of our knowledge, no bidding language has considered so far services or manufacturing operations as entities that can be traded. As explained in chapter 1, in order to apply combinatorial auctions to the *make-or-buy* of *make-or-buy-or-collaborate* decisions, it is required to predicate about manufacturing operations and services across the supply chain.

3.2.3 Winner Determination Problem

Connected with the introduction of combinatorial auctions is the winner determination problem (WDP). Winner determination is the problem, faced by the auctioneer, of choosing which goods to award to which bidder so as to maximise its revenue. The winner determination for combinatorial auctions is a complex computational problem. Indeed, one of the fundamental issues limiting the applicability of CAs to real-world scenarios is the computational complexity associated to the winner determination problem. In particular, it has been proved that the WDP is NP-complete (Rothkopf et al., 1998). General IP solvers (Andersson et al., 2000) and special purpose algorithms (Sandholm, 2002; Fujishima et al., 1999; Leyton-Brown et al., 2000) have been employed to solve the WDP, but it is well known that a general solver that performs well in all situations does not exist. For an extended review on the winner determination problem and related issues refer to (Lehmann et al., 2006; Muller, 2006; Sandholm, 2006b).

Here we aim at presenting the traditional ILP (see section 2.1.2) formulation employed to model the combinatorial auction winner determination problem, given that its comprehension is required to understand the remaining of the dissertation.

ILP formulation for the Combinatorial Auction WDP

Say that an auctioneer wants to sell n goods. Each good is denoted as g_i, where $1 \leq i \leq n$. In a combinatorial auction bidders can send *all-or-nothing* offers over a set of goods. Say that each of the m bidders participating in the auction only submits one bid[1] b_j, $1 \leq j \leq m$. Each bid is represented by a pair $b_j = (S_j, p_j)$ such that p_j is the price that the bidder is willing to pay for obtaining the set of goods S_j. How can the auctioneer select the bids that maximise his revenue? This problem can be easily modelled by means of Integer Programming (refer to section 2.1.2 for a detailed explanation). We associate to each bid b_j a binary decision variable $x_j \in \{0, 1\}$ that takes on value 1 if bid b_j is selected, and 0 otherwise. Then, the function that the auctioneer wants to maximise is his revenue, namely:

$$\sum_{j=1}^{m} x_j p_j \tag{3.1}$$

that is the *objective function* of the integer program.

Additionally, we have to make sure that each good is sold to at most one bidder since the auctioneer only owns one copy of each good. Thus, we employ coefficients c_{ij} to model that either good g_i is required in bid b_j ($c_{ij} = 1$), or not ($c_{ij} = 0$). Then, the following constraints must hold:

$$\sum_{j=1}^{m} c_{ij} x_j \leq 1 \qquad\qquad 1 \leq i \leq n \tag{3.2}$$

In what follows we list some attempts carried out in the past to deal with the generation of benchmarks for testing combinatorial auctions WDP algorithms.

3.2.4 Test Suites

No real-world benchmark of CAs has been reported in the literature. Many efforts have been done so far to generate plausible data sets to be employed to test WDP algorithms. Some experiments have been run with human bidders (Banks et al., 1989). Nonetheless, as pointed out in (Leyton-Brown and Shoham, 2006), such data sets are not useful for assessing the WDP computational complexity. In the absence of test suites, it is common practice to artificially generate data sets. Some examples are (Fujishima et al., 1999; Boutilier et al., 1999; de Vries and Vohra, 2003) for single-unit CAs, and (Leyton-Brown et al., 2000) for multi-unit CAs. Multi-unit CAs have also been tested employing multidimensional knapsack problem benchmarks, borrowed from the operations research community. A more realistic approach to generate bids is presented in (Leyton-Brown and Shoham, 2006), where complementarity relationships among goods are made explicit at bid generation time. Another realistic approach is taken in (An et al., 2005), where the authors design bidding strategies that efficiently

[1]We do this simplified hypothesis for the sake of comprehension. The extension to the OR or XOR bidding language is easy (Lehmann et al., 2006).

identify desirable bundles in the framework of the transportation industry domain (focusing therefore on single-round, first-price, sealed-bid forward CAs).

Finally, a master student has elaborated on subjects related to this thesis. Vinyals (Vinyals, 2007b; Vinyals et al., 2007a; Vinyals et al., 2007b) has implemented a very powerful simulator of the behaviour of agents bidding in an MMUCA, and has tested the performances of some of the algorithms presented in this dissertation.

In what follows we change of subject and introduce the work in the state-of-the-art related to supply chain scheduling and supply chain formation.

3.3 Supply Chain Scheduling and Supply Chain Formation

In this section, we will talk about supply chain scheduling and planning, and supply chain formation. On the one hand, the problem of supply chain formation concerns the selection of the participants to the supply chain and the terms of the exchange, with the purpose of maximising the efficiency of the supply chain. Informally, supply chain formation is the problem of deciding *who* will supply *what*, *who* will do *what*, and *who* will buy *what*. On the other hand, the problem of supply chain scheduling and planning is more focused on the coordination among the different operations across the supply chain with the purpose of minimising the cost of performing operations and transportation, and the time required to perform all the operations. Informally, supply chain scheduling and planning is the problem of deciding *when* each agent within the supply chain has to perform a given operation or job in order to finish all the operations before a given deadline.

There is a fundamental difference between the problem of supply chain formation and the problem of supply chain scheduling and planning. The former deals with finding a set of supply chain partners, whereas the latter deals with the problem of coordinating them. Nevertheless, the two problems are tightly connected. In fact, in order to effectively select the participants to the supply chain, agents should make sure that there exists a feasible scheduling of their operations. This is needed since each stakeholder along the supply chain:

- provides resources subsequently employed by other stakeholders; or
- employs or consumes resources previously produced by other stakeholders; or
- produces resources subsequently employed by other stakeholder, requiring as inputs resources previously supplied by other stakeholders.

Then, the selection of partners can be greatly improved if the feasibility of the scheduling is taken into account.

In the literature there have been many attempts to solve the problem of supply chain planning and scheduling and some attempts dealing with the supply chain formation problem. However, to the best of our knowledge, no attempt to solve the problem of supply chain formation taking into account the feasibility of the scheduling has been done so far.

3.3.1 Supply Chain Scheduling and Planning

There exist two approaches to supply chain planning and scheduling (Lau et al., 2006): *centralised* and *distributed*.

The Centralised Approach

The centralised approach to supply chain scheduling has been investigated for many years. In this approach a central authority collects all the information from the peers and then computes the optimal planning (Cohen and Lee, 1988; Ertogral et al., 1998; Sabri and Beamon, 2000; Jayaraman and Pirkul, 2001; Lee et al., 2002). A good survey on centralised planning can be found in (Erenguc et al., 1999).

The information required to optimise the scheduling may either be centralised or distributed, according to the nature of the problem. For instance, inside an enterprise there may be a central repository of information, whereas in a consortium of enterprises each firm holds its private information. The information that must be provided in order to compute the planning concerns the production features of the participants (the required time to perform an operation, the associated cost, the precedence relationships among operations, and so on). One of the firm acts as a coordinator, and, after receiving the production data, computes an optimal plan, that is subsequently communicated to the other supply chain stakeholders. In this approach, there must be information sharing among the supply chain stakeholders in order to obtain an efficient plan.

Many methods have been proposed to solve the underlying planning problem, for instance metaheuristics (Kallrath, 2002), stochastic algorithms (Alonso-Ayuso et al., 2003), or mixed-integer programming (Gaonkar and Viswanadham, 2001).

The centralised approach suffers from some drawbacks. Firstly, some firms may be reluctant to share very sensitive internal information. Secondly, the computational time required to solve even small instances is huge. Finally, the centralised approach makes it difficult to react to fails and breakdowns across the supply chain. In case some of these events occur, the scheduled plan must be recomputed from scratch.

The Distributed Approach

In the distributed approach, the decisions about the scheduling are taken locally. That is, a supply chain stakeholder builds its schedule relying on the communications with its neighbours along the supply chain. The decision is based on the local information and objectives of each supply chain stakeholder. The interactions among the supply chain participants continue until a global scheduling is found or some termination condition is met.

The major advantages of the distributed approach versus the centralised one are:

- the information is shared only at a local level;
- the computational complexity of the problem is reduced, since the problem solved locally by each supply chain stakeholder is by far less difficult than the global optimisation problem; and

- since enterprises act locally, the capacity of reacting to breakdowns or shortcomings is increased with respect to the centralised approach.

Many methods have been proposed to solve the scheduling problem with a decentralised approach. For a good review, refer to (Lau et al., 2006). The most celebrated distributed approach for centralised supply chain scheduling and planning has been the Contract Net Protocol (CNP), along with all its variants. In his original formulation, the CNP specifies a bidding approach that enables task allocation among multiple agents (Smith, 1980). The multi agent system (Wooldridge and Jennings, 1995) based approach has been widely employed in the past as well (Collins, 2002; Zhang, 2002; Reis et al., 2001; Lee et al., 2003; Wagner et al., 2003; Lau et al., 2006; He et al., 2003; Norman et al., 2004).

The distributed approach suffers as well from some drawbacks. The main shortcomings regard the feasibility and optimality of solutions. It has been shown (Jennings and Wooldridge, 1998) that, since agents act and reason locally, they disregard the other agents' constraints and the global performance of the supply chain.

To conclude, choosing between a centralised or a distributed approach strongly depends on the problem to be solved and on the availability of computational resources.

3.3.2 Supply Chain Formation

Very little work has been devoted to the problem of automating supply chain formation. In this chapter, we will not consider the literature on non-automated supply chain formation because its contributions stem from the areas of economics and negotiation rather than from optimisation. Thus, it is out of the scope of the dissertation.

Supply chain formation studies the problem of automating the process of determining the supply chain partners, under the assumption that the information required by the decision making process is decentralised.

In the area of supply chain formation two approaches have been considered as well, namely the centralised and the distributed approach.

The Centralised Approach

As far as we are concerned, little effort has been devoted to the centralised approach to the supply chain formation problem.

A significant attempt to provide a mechanism to select the right business partners in a supply chain has been undertaken by (Gaonkar and Viswanadham, 2005). This work is probably at the edge between supply chain planning and supply chain formation. This very interesting paper focuses on the problem from a real-world point of view: what happens when there is a roll-over of products in a market? Should a firm maintain the same business partners? Should it change them? The authors provide a mixed integer programming formulation of the underlying decision problem. This approach suffers from some limitations:

(1) it is not completely automated, because the interaction between the supply chain stakeholders is performed through an Internet-enabled platform;

(2) there is no communication language among the supply chain stakeholders (like, for instance, a bidding language); and

(3) it has a high computational cost and subsequent poor scalability.

However, it substantially differs from our approach, since it is not based on a market mechanism. It is more a static decision support system to help strategic decision making under particular market conditions.

In (Walsh, 2001; Walsh et al., 2000), Walsh et. al introduce combinatorial auctions for supply chain formation. These represent an extension of combinatorial auctions in which a whole supply chain is negotiated via an auction. In such a context, askers, sellers and manufacturers participate and submit bids within the same auction. In order to cope with this new auction Walsh et. al introduce the Task Dependency Network (TDN), a network representing all the producer/consumer relationships among the bidders. We consider that this work has dealt with a problem very similar to ours. In fact both our and their work:

- are built upon a market-based mechanism, namely combinatorial auctions;
- explicitly represent producer-consumer relationships holding across the supply chain; and
- model resource contention (i.e. the fact that in the system there are less resources available than the overall required ones).

However, as explained in section 1.4.2, Task Dependency Networks and combinatorial auctions for Supply Chain Formation are limited along several dimensions: they do not possess the expressiveness, computational, and formal analysis tools required to deal with the *make-or-buy-or-collaborate* decision problem.

Collins et. al in (Collins, 2002; Babanov et al., 2003) deal with a problem similar to the supply chain formation introducing time and precedence constraints. However they do not explicitly model the multiple levels within a supply chain and the resource contention across it. They also provide a bidding language including information about the time required to perform operations.

Finally, Norman et. al (Norman et al., 2004) describe a combinatorial auction to form virtual organisations. They also provide an advanced bidding language for expressing offers in which the time dimension is considered as well. Although very innovative, we find that its applicability to the problem of supply chain formation is limited since neither resource contention nor the producer-consumer relationships present across a supply chain can be modelled.

The Distributed Approach

Distributed approaches to supply chain formation are not so closely related to our work. However, we will point out the two most relevant works in the field that employ a market based mechanism.

Rosenschein and Zlotkin in (Rosenschein and Zlotkin, 1994; Zlotkin and Rosenschein, 1996) introduce Task Oriented Domains (TODs). A

TOD is a set of tasks that must be completed, and a cost function over bundles of tasks. They fix a set of negotiation rules and provide some theoretical results on the properties of the negotiation outcome.

Walsh and Wellman in (Walsh and Wellman, 2003) provide a decentralised version of the auction mechanism provided in (Walsh et al., 2000), which is based on a variation of the Contract Net Protocol.

We stress that both approaches suffer from limitations. On the one hand, TODs do not incorporate nor implicitly neither explicitly the dependencies among operations across a supply chain, whereas the model in (Walsh and Wellman, 2003) suffers from the same expressiveness, computation and formal analysis shortcomings as combinatorial auctions for supply chain formation do.

3.4 Conclusions

Little work has been done so far to solve *make-or-buy* or *make-or-buy-or-collaborate* decisions with a centralised market-based approach. Many papers have focused on similar problems though none of them captures all the requirements expressed in sections 1.4.1 and 1.4.2.

Our work is placed somewhere in between *centralised supply chain planning* and *centralised supply chain formation*. Our work is not entirely included in the field of supply chain formation because we do not only assess the participants to a supply chain, but we also provide a feasible sequence of supply chain operations to perform. Analogously, our work is not completely included in the field of supply chain scheduling since the participants to the supply chain are not fixed a-priori, but determined on the fly based on a market mechanism. Furthermore, we do not need to include the time dimension into the problem to provide a feasible schedule. In fact, the precedence relationships among operations are implicitly represented in the formalism that we employ to model resource contention at each level of the supply chain.

Summarising, in the state of the art we find solutions to both supply chain scheduling and planning and to supply chain formation problems. However, none of the solutions we are aware of possesses all the features required to solve both *make-or-buy* and *make-or-buy-or-collaborate* decision via a market-based approach.

(1) As thoroughly explained in sections 1.4.1 and 1.4.2, *combinatorial auctions* lack of the possibility to express manufacturing operations, or equivalently production relationships among the goods at auction. However, they provide a good model to build upon because they allow to express complementarities among the goods at auction (Cramton et al., 2006); they can count on theoretically well-founded bidding languages (Nisan, 2006), and there have been significant contributions to the study of their winner determination problem (Lehmann et al., 2006).

(2) As detailed in section 1.4.2, combinatorial auctions for supply chain formation and the associated Task Dependency Networks help negotiating manufacturing operations. However, they suffer from *formal*, *computational*, and *expressiveness* limitations.

(3) We deem that distributed approaches are not suitable to our problem. In fact, to the best of our knowledge, they do not guarantee nor optimality nor feasibility. The literature in combinatorial auctions has thoroughly demonstrated the efforts in finding optimal solutions to the winner determination problem. In today's business world, to provide methodologies that sacrifice optimality when big quantities of money are in play is a risky business.

Chapter 4

MUCRAtR

In this chapter we deal with the *make-or-buy* decision problem when complementarities among goods hold at the bidders' side. With this aim, we introduce a new type of combinatorial auction, the *Multi-unit Combinatorial Reverse Auction with transformability Relationships among goods (MUCRAtR)*, extending traditional combinatorial auctions. We also provide a mapping of the MUCRAtR winner determination problem to an optimisation problem on Place/Transition Nets (PTN). Such a mapping allows to efficiently solve the WDP for some problem classes, and provides a set of powerful formal tools for describing the underlying optimisation problem.

This chapter is organised as follows. In section, 4.1 we introduce the problem we aim at solving and informally outline the proposed solution. In section 4.2, by means of some examples and intuitions, we introduce the limitations associated to CAs with respect to the *make-or-buy* decision problem in a combinatorial scenario. In section 4.3, we introduce a formalism, based on PTN, that overcomes part of such problems. In section 4.4 we extend PTN in order to amend the expressiveness shortcomings of the PTN model. In particular, we introduce a new type of PTN called *Weighted Place Transition Net* (WPTN). Moreover, we define a new reachability problem over WPTN, the *Constrained Maximum Weight Occurrence Sequence Problem* (CMWOSP). In section 4.5, relying on WPTNs, we succeed in formally representing an auctioneer's internal production and cost structure along with the set of received offers from bidders under a unified formalism. Building upon such framework, we formally define the WDP for the new auction as a particular CMWOSP. In section 4.7, we prove that the CMWOSP, and thus the WDP formalised in section 4.6, can be solved by means of IP under suitable conditions. Finally, section 4.8 draws some comments and concluding remarks.

4.1 Beyond Combinatorial Auctions

In the introductory chapter we mentioned that we are dealing with two main issues in this dissertation. The first one is the automation of *make-or-buy* decisions across the supply chain, and the second is the automation of *make-or-buy-or-collaborate* decisions across the supply chain. In this chapter we focus on the *make-or-buy* decision problem, namely the problem of selecting what to produce in-house and what to outsource in order to obtain some required goods. We argued in section 1.4.1 that this concern is reasonable because the cost of the raw materials plus the cost of the manufacturing operations could eventually be higher than the cost of already-made goods. As an additional constraint, we require that the complementarities among goods on the bidders' side are taken into account: bidders should be allowed to compose *all-or-nothing* offers over bundles of goods.

In section 1.4.1, through the example of the *Grandma & co* firm, we showed that the *make-or-buy* decision problem represents a challenging problem in a scenario with complementarities among the goods. We highlighted that *Grandma & co* requires a complex decision support system along with a combinatorial negotiation mechanism that helps it in detecting the cost-minimising buying configuration and the internal operations to perform in order to obtain the finally required goods.

For this reason, we decided to build upon combinatorial auctions to cope with the *make-or-buy* decision problem. We recall that the distinguishing feature of combinatorial auctions is that bidders can submit *all-or-nothing* offers over bundle of goods. This allows to mitigate the risks connected with markets with strong complementarities, like for instance depressed bidding[1].

Unfortunately, as we thoroughly showed in section 1.4.1, some limitations prevent the application of combinatorial auctions to the *make-or-buy* decision problem. That is mainly due to two types of limitations: *expressiveness* and *winner determination problem*. We recall in table 4.1 the limitations of combinatorial auctions thoroughly explained in section 1.4.1.

Hence, in this chapter we extend *Multiunit Combinatorial Reverse Auctions* (MUCRA)[2] in order to overcome the intrinsic limitations of CAs for dealing with the *make-or-buy* decision problem. The resulting auction model is called *Multi-unit Combinatorial Reverse Auction with transformability Relationships among goods (MUCRAtR)*. This new auction type allows a buyer/auctioneer to express and communicate to bidders its internal production structure and its final requirements. Bidders can then formulate appropriate offers and send them back to the auctioneer. Upon receiving the offers, an auctioneer can determine, by means of a public selection rule, the cost minimising combination of bids along with the internal operations leading to its final requirements.

Then, firstly we try to model an auctioneer's internal manufacturing operations by means of Place Transition Nets (PTNs, thoroughly described in section 2.3). They perfectly represent the manufacturing operations by specifying the quantity of resources both required an produced by each manufacturing operation. Furthermore, they naturally model the producer/consumer relationships holding among them. Then, the PTN representing the internal manufacturing operations fulfills requirement (1) in table 4.1.

Next, we incorporate the offers received by the auctioneer into the PTN encoding the auctioneer's production structure. This idea is based on the intuition that the selected offers inject goods into the auctioneer production process: without ingredients it is not possible to produce pies. This solves issue (3) in table 4.1.

[1]Depressed bidding is a phenomenon associated to the fact that bidders may risk to obtain only a part of a set of complementary goods, and therefore bid less aggressively.

[2]We recall that a MUCRA is simply a combinatorial reverse auction in which multiple copies of each item are auctioned[3] (Sandholm, 2002).

TYPE	REQUIREMENTS
Expressiveness	(1) specification of the internal manufacturing operations and the producer/consumer relationships among them (2) specification of an auctioneer's final requirements (3) relationships among the manufacturing operations, the auctioned goods, and the received bids (4) specification of an auctioneer's internal cost structure
WDP	(5) information about which in-house operations to perform and in which order

Table 4.1: Summary of requirements for the *make-or-buy* decision problem.

More in details, we build two PTNs: one representing the internal manufacturing operations of an auctioneer, that we name PTN_I (I from *Internal*), and another one extending PTN_I to incorporate offers, called PTN_E (E from *Extended*).

The dynamic behaviour of PTNs serves to describe the set of possible outcomes of a MUCRAtR. In particular, PTNs can naturally model:

(1) the preconditions of each manufacturing operation (the required inputs must be present);

(2) the resources consumed and produced by each manufacturing operation;

(3) the quantity of resources injected into the system when an offer is selected; and

(4) the quantity of resources available to an auctioneer after performing a given manufacturing operation.

According to item (4) in the list above, an auctioneer can model its resource availability at any step of its manufacturing process. Consequently, PTN_E can compactly encode the outcomes (in terms of finally available resources) of all the possible decisions an auctioneer may take[4]. The encoded information concerns the level of resources available at the end of a production process fed by a set of offers and composed of a sequence of internal operations. Furthermore, the rules governing the dynamics of PTNs enforce that each of the possible decisions is implementable, i.e. all the manufacturing operations are run in the correct order and only if the required input resources are provided.

[4]Notice that by decision we mean the selection of a set of offers and of a set of internal manufacturing operations.

However, an auctioneer is not simply interested in choosing the bids and the internal operations leading to a satisfactory level of available resources. Above all he is interested in minimising its costs while doing this. Unfortunately, PTNs allow to express neither the cost associated to performing manufacturing operations nor the cost associated to selecting a set of bids. Due to this expressiveness limitation, we decided to extend the notion of PTN to incorporate the cost associated to a manufacturing operation and the cost associated to a bid. Such extension, called *Weighted Place Transition Nets (WPTN)*, allows associating a cost to each transition of a PTN.

With this tool at hand, we firstly associate a cost to each transition of PTN_I. This creates a WPTN allowing to reason about the manufacturing operations internal to an auctioneer. I name such WPTN *Transformability Network Structure* (TNS). It incorporates the following information about an auctioneer's internal manufacturing operations:

(1) the required input goods;

(2) the produced output goods;

(3) the cost associated to each operation; and

(4) the eventual producer/consumer relationships with other operations.

By means of a TNS, an auctioneer can also compactly communicate to bidders all the possible RFQ configurations leading to its final requirements. Summarising, the information contained in a TNS along with the auctioneer's finally required goods provide to bidders sufficient information to compose meaningful offers. This overcome requirement (2) of table 4.1.

As mentioned above, our strategy shall be to map the internal manufacturing operations and the received offers into a PTN (PTN_E). If we associate a cost to each of its transitions, we obtain a WPTN that provides a unified description framework for the *make-or-buy* decision problem. I call such extension *Auction Net* because it permits to encode the information about an auctioneer's internal production and cost structures and about the offers it receives. The formal language offered by an *Auction Net* helps fulfill expressiveness requirements (1), (3) and (4) of table 4.1.

By means of an *Auction Net* an auctioneer can compactly express the outcome of any of its possible decisions (acceptance of some bids and execution of some internal operations), and also quantify the cost associated to each of such outcomes. Furthermore, the *Auction Net* allows to incorporate the information about an auctioneer's initial stock.

We recall that the goal of the auctioneer is selecting a cost minimising outcome fulfilling its final requirements. This can be achieved only if he can express constraints over possible outcomes. Then, the last requirement for expressing the decision problem is allowing an auctioneer to express constraints over the set of possible outcomes.

Against this background, the MUCRAtR winner determination problem can be stated as a problem over an *Auction Net* (a WPTN), where the goal is minimising the cost associated to a sequence of steps that brings to a final state fulfilling some constraints. Then, we define a new optimisation problem on WPTNs: the *Constrained*

Maximum Weighted Occurrence Sequence Problem (CMWOSP). The objective of a CMWOSP is finding a cost minimising sequence of steps leading to a final state fulfilling a set of constraints. This provides a solution to requirement (5) in table 4.1.

Notice that the result of a CMWOSP is a *firing sequence*, i.e. an ordered sequence of transitions. This reflects a critical feature of the *make-or-buy* decision problem. An auctioneer cannot run its internal manufacturing operations in a random order. Because of producer/consumer relationships among manufacturing operations, an auctioneer must be aware of the implementation order. For instance, if *Grandma & co* decides to only buy the basic ingredients and to perform all the manufacturing operations internally, it cannot perform the *Baking* operation before the *Make Dough* or *Make Filling* operations, since the latter ones provide the inputs to the former one (cf. figure 1.1).

Then, the definition of the winner determination problem does not only assess the optimal set of goods to buy, but also the optimal ordered sequence of in-house operations to perform in order to obtain the goods finally required by the auctioneer.

Two direct benefits stem from the mapping of the MUCRAtR WDP to WPTNs. Firstly, it is possible to directly import all the PTNs analysis tools and theoretical results and apply them to our problem. This provides the techniques for dealing with requirement (5) in table 4.1. In fact, we manage to model, for a wide class of problems, the WDP via integer programming (see section 2.1.2), and efficiently solve it by means of black-box solvers as ILOG CPLEX (ILOG, 2007) or GNU GLPK (Makhorin, 2001).

4.2 The problem

In what follows we further specify the extensions to CAs needed for dealing with the *make-or-buy* decision problem. With this aim we extend example 1.1 in chapter 1. We recall that the example was about *Grandma & co*, a company devoted to produce and sell apple pies. According to the example, the marketing department at *Grandma & co* has forecast a sale of two hundreds apple pies within the next month, and therefore *Grandma & co* starts an automated sourcing process. *Grandma & co* opts for running a combinatorial auction to source the required ingredients. However, as explained in section 1.4.1, besides inviting providers of basic ingredients (*butter, sugar, flour, apples, margarine*), *Grandma & co* invites providers of intermediate goods (*dough*, *filling*), and even of final goods (*apple pies*). The production management department aims at evaluating the opportunity to outsource part of the production process.

Unfortunately, *Grandma & co* faces a decision problem that cannot be solely treated by means of combinatorial auctions because of the intrinsic limitations listed in table 4.1. In example 4.1 we provide an extended version of example 1.1 that explicitly illustrates such limitations.

Example 4.1. The data characterising the *Grandma & co*'s decision problem are:

(1) The cost of its internal manufacturing operations:

(a) A *Make Dough* operation costs €5 each time it is carried out. It requires one unit of *butter*, three units of *sugar*, and two units of *flour* as inputs; and it produces two units of *dough* as output.

(b) A *Make Filling* operation costs €6 each time it is carried out It requires one unit of *flour*, eight units of *apple*, and two units of *margarine* as inputs; and it produces two units of *filling* as output.

(c) A *Baking* operation costs €14 each time it is carried out. It requires four units of *dough* and four units of *filling* as inputs; and it produces four units of *apple pie* as output.

(2) A sale forecast of 200 apple pies. This represents the final requirements of *Grandma & co*.

(3) A stock of one hundred units of *floor* and two hundreds units of *sugar*.

□

Then, if *Grandma & co* intends to run a combinatorial auction and to invite all its providers, it must be able to

- send them a request for quotes (RFQ) containing the number of required units for each good; and
- once received all bids, it must be able to determine which bids to accept and which internal manufacturing operations to perform in order to obtain the 200 apple pies.

Unfortunately, life is not that easy for *Grandma & co*. Firstly, it is not possible to a priori establish how many units of each good the auctioneer (*Grandma & co*) requires. In fact, this depends on the production plan, that can only be decided upon receiving the offers. Secondly, once received all bids, *Grandma & co* needs a winning rule for the optimal, efficient and automatic selection of the best set of bids and in-house operations. In the two following sections we illustrate the first and second problem.

4.2.1 Communicating the RFQ

In a traditional Multi-Unit Combinatorial Reverse Auction (MUCRA) scenario, a *Request for Quotation* (RFQ) (Reyes-Moro et al., 2003) expresses the number of required units for each good. However, whenever an auctioneer (*Grandma & co*) faces *make-or-buy* decision problems, it happens that the requirement sent to bidders (the RFQ) is not equivalent to the quantity of goods that the auctioneer actually requires (the 200 apple pies). When internal manufacturing operations are taken into account, an auctioneer has to distinguish between the objective quantity of goods at the end of its production process and what to ask providers for. This occurs because an auctioneer (*Grandma & co*) can opt for several, possible buying options and several, possible levels of internal production. All these options differ in the number of required units and in the level of internal production. For instance,

- if *Grandma & co* decides to buy only already-made pies without producing anything, then it must ask providers offers for two hundred units of apple pies. This results in the RFQ expressed in table 4.2(a) and in the internal operations quantified in table 4.2(b).

Resource	Required Units
butter	0
sugar	0
flour	0
apples	0
margarine	0
dough	0
filling	0
apple pies	200

(a) Request for quotes for apple pies only.

Resource	Required Units
butter	100
sugar	500
flour	300
apples	800
margarine	200
dough	0
filling	0
apple pies	0

(c) Request for quotes for basic ingredients only.

Operation	Quantity
Make Dough	0
Make Filling	0
Baking	0

(b) Internal operations to perform.

Operation	Quantity
Make Dough	100
Make Filling	100
Baking	50

(d) Internal operations to perform.

Table 4.2: Request for quotes for different scenarios.

- if *Grandma & co* decides to produce everything in house, then it must require for each ingredient the quantity needed for producing 200 apple pies, and must perform the *Make Dough*, *Make Filling*, and *Baking* operations as many times as required. This corresponds to the RFQ expressed in table 4.2(c) and in the internal operations quantified in table 4.2(d).

It is easy to understand why *Grandma & co* cannot completely specify its exact requirements a-priori (limitation (2) in table 4.1). The number of acquired units will depend on the received offers.

In order to overcome such difficulty *Grandma & co* should be able to communicate to bidders its internal production relationships along with the producer/consumer relationships among them (limitation (1) in table 4.1). When bidders have this information available, *Grandma & co* simply has to communicate to bidders the quantity of each good it aims at obtaining at the end of the production process (in our case two hundred *apple pies*). The bidders can then infer the required quantity for each good (limitation (2) in table 4.1).

4.2.2 Selecting the optimal decision

Even under the hypothesis that *Grandma & co* was able to uniquely communicate its requirements to bidders, once received the bids it would not be able to decide which bids to accept and which internal operations to perform in order to minimise its costs and to obtain the 200 apple pies. More importantly, it would not have any public rule stating how to win in the auction. How can bidders participate and submit bids if they are not aware of the winning bids' selection mechanism? If *Grandma & co* cannot determine who the winners are, there can be no auction.

In order to express all the possible outcomes of any of its possible decisions, an auctioneer must be able to link its internal production and cost structure, the received offers, and its final requirements (the 200 apple pies).

If it also wants to select the best among those possible decisions, then it must be able to quantify the cost associated to each of the above-mentioned decision outcomes.

Then, in the following section, we make a first attempt at solving the above-mentioned problems relying on PTNs (section 2.3). In this way, we will succeed in modelling all the possible decisions an auctioneer may take.

4.3 A first attempt: Place/Transition Nets

PTNs (see section 2.3) are a very powerful tool to describe discrete dynamical systems, like for instance operating systems, workflows, finite state machines, parallel activities, data-flow computation, producers-consumers systems with priority, and so on. The firing of a transition in PTNs represents a state change in a discrete system. Such a state change can only take place if some preconditions occur (i.e. the transition must be enabled). For instance, if we model manufacturing operations by means of transitions in a PTN, the execution of a manufacturing operation changes the state of the production system: some goods are consumed, while other goods are produced, whenever enough input goods are available.

In this section we try to model the problem of *Grandma & co* by means of PTNs. In section 4.3.1 we model via PTNs the internal production structure of an auctioneer, and in section 4.3.2, we complement such PTN model by incorporating the offers received by the auctioneer.

4.3.1 Modelling the internal production structure

In this section we model an auctioneer internal production structure by means of PTN. Consider the following example.

Example 4.2. In figure 4.1, we associate a Place/Transition Net Structure (PTNS[5]) to the internal production structure of *Grandma & co*, characterised in example 4.1. In doing this we associate *places* (P) to goods, *transitions* (T) to manufacturing operations, and input/output arcs (A) and their weights (E) to the quantity of goods consumed/produced by each manufacturing operation. Formally,

- The set of places is $P = \{butter, sugar, flour, apples, margarine, dough, filling, applepie\}$
- The set of transitions is $T = \{makedough, makefilling, baking\}$
- The set of arcs is $A = \{(butter, makedough), (sugar, makedough), (flour, makedough), (sugar, makefilling), (flour, makefilling),$

[5]Refer to definition 2.1.

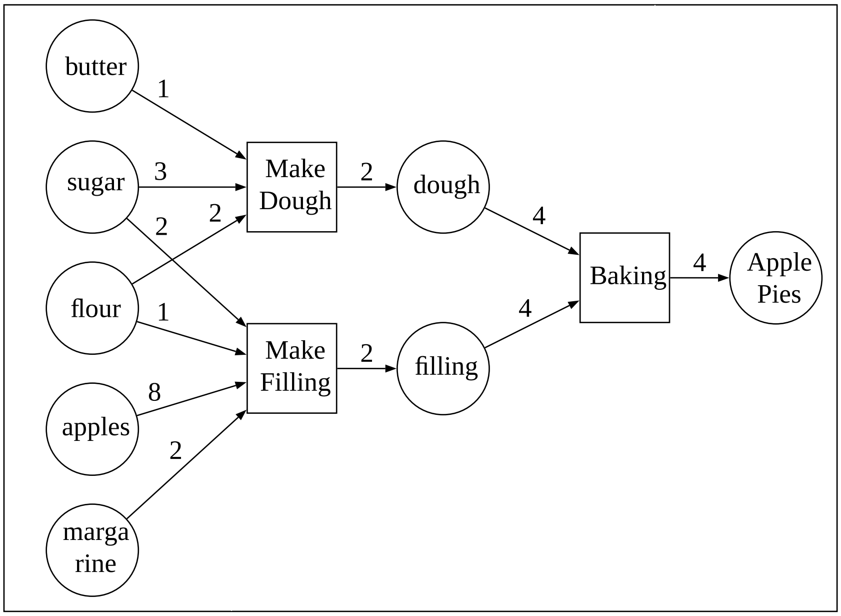


Figure 4.1: PTNS associated to example 4.1.

$(apples, makefilling), (margarine, makefilling), (makedough, dough),$
$(makefilling, filling), (filling, baking), (dough, baking),$
$(baking, applepie)\}.$

- The arc weight function E is:

$$\begin{aligned} E(butter, makedough) =& 1 & E(sugar, makedough) &= 3 \\ E(flour, makedough) =& 2 & E(sugar, makefilling) &= 2 \\ E(flour, makefilling) =& 1 & E(apples, makefilling) &= 8 \\ E(margarine, makefilling) =& 2 & E(makedough, dough) &= 2 \\ E(makefilling, filling) =& 2 & E(filling, baking) &= 4 \\ E(dough, baking) =& 4 & E(baking, applepie) &= 4 \end{aligned}$$

Then, with this tool at hand, we can quantitatively represent the input resources needed and consumed by each manufacturing operation, the output resources produced, and the producer consumer relationships among the manufacturing operations.

We recall that a PTN is a PTNS with associated an initial marking $\mathcal{M}_0$ (see section 2.3). The initial marking in a PTN usually represents the initial state of a discrete dynamic system. In the case of *Grandma & co* we can provide a similar semantics. The following example clarifies this point.

Example 4.3. The initial marking $\mathcal{M}_0$ stands for the initial stock at *Grandma & co*. Indeed, the stock of a firm represents the “initial state” of its supply chain. The initial stock at *Grandma & co* is two hundreds units of sugar and a hundred units of flour (see example 4.1). The multiset (refer to section 2.2) representation of the initial state would be:

$$\mathcal{M}_0 = 200'sugar + 100'flour$$

Thus, in figure 4.2, we graphically depict the initial marking of the PTN by means of numbers within places (circles). We call the resulting PTN PTN_I (I stands for Internal).

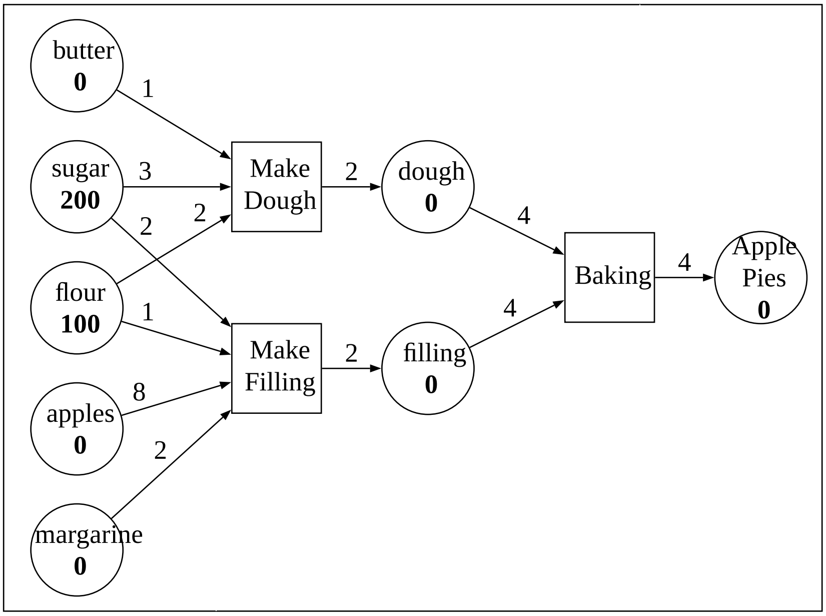


Figure 4.2: PTN_I associated to example 4.1.

Recall from section 2.3 that a transition in a PTN is enabled only if its input places contain enough tokens. For instance, in figure 4.2, transition *Make Dough* is *enabled* only if *at least* one unit of *butter*, three units of *sugar*, and two units of *flour* are within its input places. This is exactly what we require for a manufacturing operation to be *enabled*: it can not be performed unless the required goods are available. Moreover, looking at the *Baking* operation in figure 4.2, we observe that the producer/consumer relationships between *Make Dough* and *Baking* on one side, and between *Make Filling* and *Baking* on the other side, is quantitatively described by the PTN. Notice that the enabling condition guarantees that a producer/consumer relationship is not only quantitatively represented, but also it is constrained to be implemented in its dynamics.

If a transition is enabled in a marking it can *fire* (see definition 2.4). If a transition fires it consumes some input goods and produces some output goods. Once more, this is the semantics we require for a manufacturing operation: a manufacturing operation consumes a set of input resources and produces a set of output resources.

Example 4.4. In table 4.3 we show what happens when the *Make Dough* transition fires. In the left image *Make Dough* is enabled. The execution of *Make Dough* provides some inputs to the *Baking* operation, as shown in the image on the right, thus perfectly describing the producer/consumer relationship among them.

□

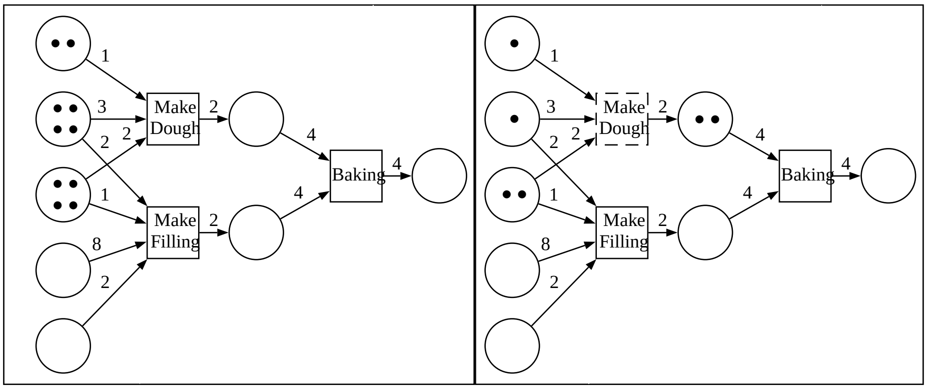


Table 4.3: Execution of a manufacturing operation on PTN_I.

What does it happen when there is a sequence of firings? As explained in section 2.3.1, the PTN will pass through a succession of markings (states). What does a marking represent in the case of *Grandma & co*? We recall that a *marking* is a distribution of tokens over the set of places. It associates an integer value to each place. What is the meaning of associating value 100 to flour? The answer is that a *marking* stands for the state of a production process, i.e. it describes the resources available at each state of the transformation process. In fact, it associates to each state the number of units of each good available to the auctioneer in that state. Accordingly, a manufacturing operation can be performed in a given state only if enough tokens are available in its input places in that state. The firing of a transition adds tokens into its output places likewise a manufacturing operation produces new available resources to the auctioneer.

If *markings* describe the level of resources currently available to an auctioneer, they naturally apply to describe the requirements of an auctioneer as well. An auctioneer aims at *reaching* a marking that fulfils its requirements (at least two hundreds tokens in the *applepie* place). This helps linking an auctioneer's requirements to its internal production structure.

In section 2.3.1, we illustrated the problem of reachability, i.e. the problem of reaching a given marking $\mathcal{M}_d$ departing from an initial marking $\mathcal{M}_0$. We explained that it is a well studied problem in the PTN literature. The reader can imagine that the auctioneer is dealing with a similar problem: reaching a *marking* that fulfils its needs.

Summarising, by means of the PTN representation we partially fulfill requirements (1) and (2) in table 4.1. However, we still need to express:

- the relationships between the internal manufacturing operations and the received offers (limitation (3) in table 4.1); and

- the information about the cost associated to bids' selection and to manufacturing operations' carrying out (limitation (4) in table 4.1).

Then, in the next section we incorporate the description of the received offers into PTN_I.

4.3.2 Incorporating Bids

In this section we cope with limitation (3) in table 4.1. That is, to establish a relationship among an auctioneer's internal production structure, the goods at auction, and the received offers. This entails relating the PTN description of section 4.3 (PTN_I) with the bidders' offers and the goods at auction.

Firstly, notice that the relation between the auctioned goods and the manufacturing operations is already accounted by PTN_I. It quantitatively specifies the goods required and produced by each manufacturing operation. Hence, it only remains linking the received combinatorial offers to the PTN_I. In fact, the utility of PTN_I is very limited if an auctioneer cannot link it to the received bids. For instance, the PTN (production process) described in figure 4.2 cannot work: there are not enough tokens (goods) to fire (run) any of the transitions (manufacturing operations). The problem is that the auctioneer (*Grandma & co*) needs to *buy* goods to feed its production process. Buying goods is equivalent to injecting tokens into the corresponding places. For instance, if *Grandma & co* decides to accept a bid offering 100 units of *butter*, this will inject 100 units into the *butter* place and will correspondingly increment the marking of the PTN. The counterpart of this operation would be putting a **100** into the *butter* place of figure 4.2.

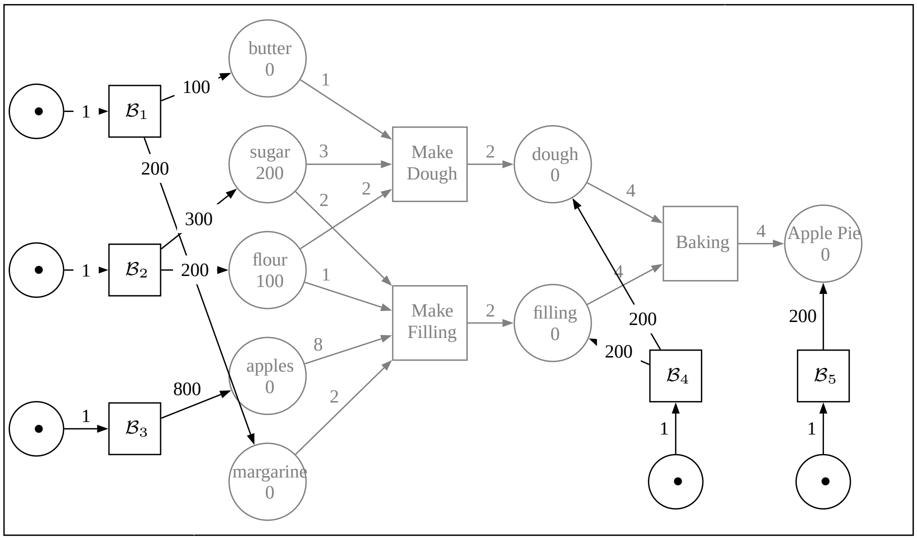


Figure 4.3: PTN_E. Incorporating bids into the PTN_I of figure 4.2.

As a consequence, incorporating bids into the PTN is quite natural. Indeed, they can be easily modelled by means of transitions as well. If a bid is selected, it must increase the amount of some available resources. Correspondingly, a transition adds tokens into its output places when fired. However, two features distinguish bids from manufacturing operations. Firstly, bids do not consume any resource. Secondly, bids can be run only once (it is not possible to accept a bid twice in our semantics). Therefore, each bid will be represented by a special type of transition, whose single input place will not be a good, but a sort of controller. Such a controller, named *bid place*, will enforce that a transition representing a bid is selected at most once. We will call this type of transitions *bid transitions*. In contrast, we will call the transitions corresponding to manufacturing operations *operation transitions*, and the places representing goods *good places*. We make clear the process of bid incorporation by means of an example.

Example 4.5. Say that *Grandma & co* receives the combinatorial offers in equations (4.1) to (4.5) below from bidders. We represent an offer sent by a provider as a multiset[6] $\mathcal{B} \in \mathbb{N}^G$, where G is the set of goods (in our case represented by places in figure 4.2), along with a cost. The multiplicity associated to each element of the multiset stands for the number of offered units for the element.

$$\mathcal{B}_1 \rightarrow 100'butter + 200'margarine \quad \text{at €200} \tag{4.1}$$

$$\mathcal{B}_2 \rightarrow 200'flours + 300'sugar \quad \text{at €100} \tag{4.2}$$

$$\mathcal{B}_3 \rightarrow 800'apples \quad \text{at €200} \tag{4.3}$$

$$\mathcal{B}_4 \rightarrow 200'dough + 200'filling \quad \text{at €1300} \tag{4.4}$$

$$\mathcal{B}_5 \rightarrow 200'apple\,pies \quad \text{at €2400} \tag{4.5}$$

For instance, $\mathcal{B}_4 \rightarrow 200'dough + 200'filling$ at € 1300 stands for a combinatorial bid offering two hundred units of *dough and* two hundred units of *filling* at € 1300.

In figure 4.3 we intuitively show how to incorporate bids in equations (4.1) to (4.5) into the PTN_I on figure 4.2. PTN_I is shadowed, whereas the incorporated bids are in dark black. We will refer to the PTN in figure as the PTN_E (E from Extended). Notice that:

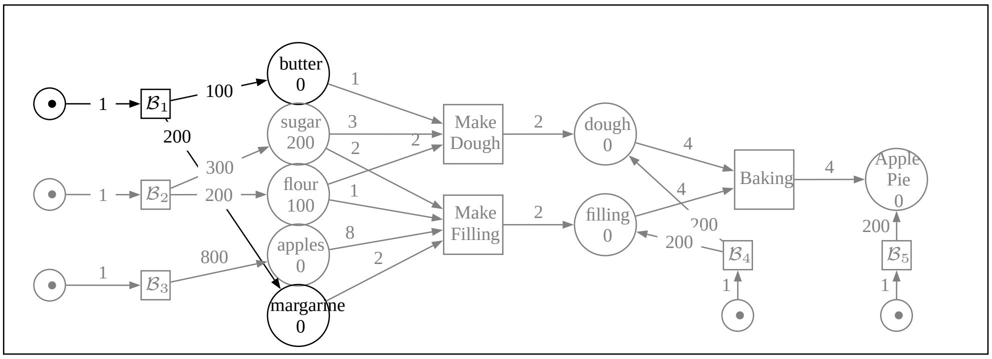
(1) The input places of *bid transitions* (transitions associated to bids and represented by $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4, \mathcal{B}_5$ in figure 4.3) only contain one token and their input arcs weigh one. Therefore, a bid transition can fire once at most.

(2) A *bid transition* does not have any other input place except from a *bid place*. Thus, it does not consume any resources.

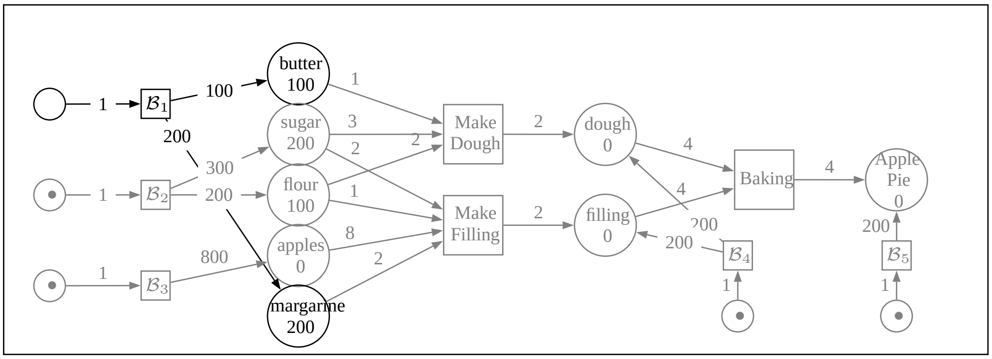
(3) The output places of *bid transitions* are the goods offered in the corresponding bids, whereas the output arcs' weights are the number of offered units.Therefore, they increase the number of tokens present on the net if fired.

In table 4.4 we graphically depict the evolution of the PTN in figure 4.2 when applying the firing sequence $J = \langle \mathcal{B}_1, makedough \rangle$. The upper picture shows the initial

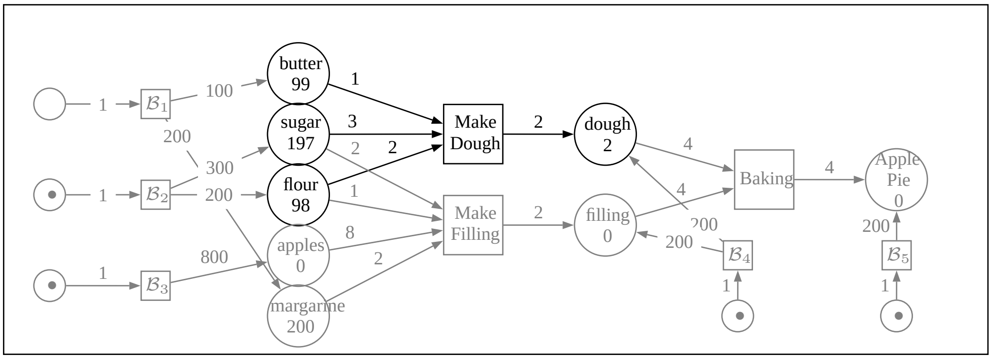
[6]Refer to section 2.2.



(a) Initially.



(b) After selecting bid $\mathcal{B}_1$.



(c) After performing *Make Dough*.

Table 4.4: Applying the firing sequence $J = \langle \mathcal{B}_1, makedough \rangle$.

marking $\mathcal{M}_0 = 100'butter + 200'margarine$ (the stock at *Grandma & co*). The central picture shows the marking obtained after firing transition $\mathcal{B}_1$ (i.e. after accepting bid $\mathcal{B}_1$). Finally, the lower picture shows the marking obtained after firing $makedough$ (after performing the *Make Dough* operation). Notice that in both cases transitions $\mathcal{B}_1$ and *Make Dough* are enabled. Notice also that transition $\mathcal{B}_1$ cannot fire anymore, whereas *Make Dough* can.

□

Summarising, with the PTN in figure 4.3 *Grandma & co* can express:

(1) its internal manufacturing operations along with the producer/consumer relationships among them (requirement (1) in table 4.1);

(2) the relations among the auctioned goods, the received offers, and the manufacturing operations (rquirement (2)); and

(3) its final requirements (requirement (3)).

Furthermore, it can obtain all the possible production states reachable by means of any legal combination of bids and internal operations. That is, it characterises the combinatorial problem by providing a formalism to enumerate all the possible solutions. This can be achieved thanks to the dynamics of PTN (the firings). This is a crucial point: the PTN_E in figure 4.3 compactly represents all the possible decisions that *Grandma & co* can take.

Unfortunately, *Grandma & co* is not interested in *simply* reaching a state that fulfils its final requirements, it wants to *minimise* its costs as well. How can we quantify that performing manufacturing operations costs money? How can we quantify that buying goods costs money? It is under this point of view that PTNs lack of the necessary expressiveness and need to be extended. In the next section, we explain how to deal with such extension.

4.4 Weighted Place Transition Nets

There is a feature of some discrete systems (in particular the one we consider) that, to the best of our knowledge, has never been considered so far in the PTN literature, and that we deem fundamental. A change in the state of a system may have an associated cost. For instance, in our case, a manufacturing operation has a cost associated to each time it is carried out. Thus, in order to model manufacturing operations, we need to extend Place Transition Nets to incorporate the notion of *transition cost*. Such extension will allow us not only to represent the fact that a cost is associated to each transition firing, but also to easily compute the cost associated to a *firing sequence*.

The extension of PTN to incorporate the costs of operations and bids is quite natural and consistent with all the properties of PTN. If we aim at representing the fact that performing a manufacturing operation costs money, we simply have to associate a cost to the firing of an *operation transition*. Similarly, if we aim at representing that buying goods costs money, we have to associate a cost to the firing of each *bid transition*. In general, since both bids and manufacturing operations can be represented by means of PTNs, we have to associate a cost to each transition in a PTN.

4.4.1 WPTNSs and WPTNs

We extend the notion of Place Transition Net (see section 2.3) by associating a *cost* to each transition. This leads us to the definition of *Weighted Place Transition Net Structure* (WPTNS) and *Weighted Place Transition Net* (WPTN).

Definition 4.1 (WPTNS)**.** A WPTNS is a a tuple (P, T, A, E, C) where:

- P, T, A, E are defined exactly like in a PTNS.
- $C : T \rightarrow \mathbb{R}$ is a cost function that associates a cost to each transition.

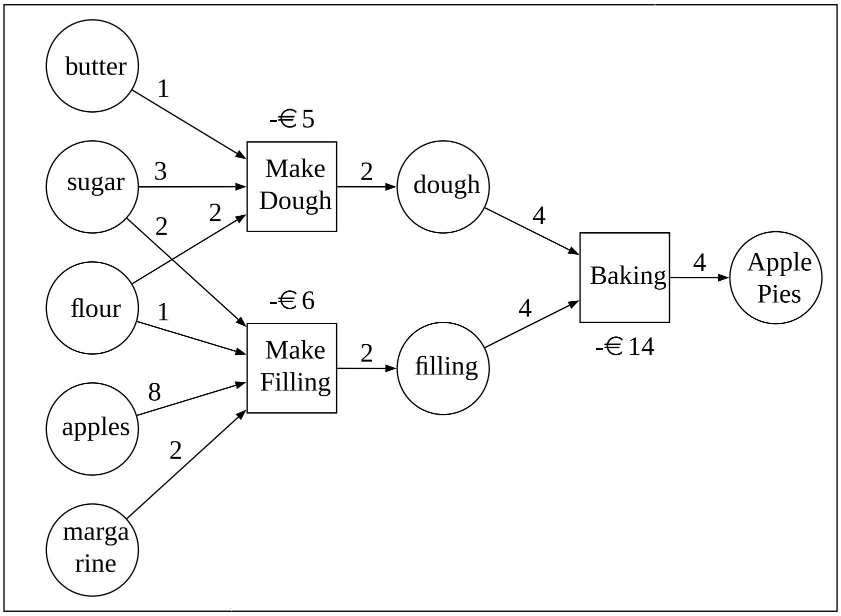


Figure 4.4: WPTNS associated to example 4.1.

Example 4.6. Let us associate a WPTNS to the internal production structure of *Grandma & co* specified in example 4.1. At this aim we associate *places* (P) to goods, *transitions* (T) to manufacturing operations, transition costs (C) to manufacturing costs, and input/output arcs (A) and their weights (E) to the quantity of goods consumed/produced by each manufacturing operation. A WPTNS employs the same graphical representation as a PTN (see section 2.3), the only difference being that a cost labels each transition. We depict in figure 4.4 the resulting WPTNS, formally defined as:

- The set of places is $P = \{butter, sugar, flour, apples, margarine, dough, filling, applepie\}$
- The set of transitions is $T = \{makedough, makefilling, baking\}$

- The set of arcs is $A = \{(butter, makedough), (sugar, makedough), (flour, makedough), (sugar, makefilling), (flour, makefilling), (apples, makefilling), (margarine, makefilling), (makedough, dough), (makefilling, filling), (filling, baking), (dough, baking), (baking, applepie)\}$.
- The arc weight function E is:

$$\begin{aligned} E(butter, makedough) =& 1 & E(sugar, makedough) &= 3\\ E(flour, makedough) =& 2 & E(sugar, makefilling) &= 2\\ E(flour, makefilling) =& 1 & E(apples, makefilling) &= 8\\ E(margarine, makefilling) =& 2 & E(makedough, dough) &= 2\\ E(makefilling, filling) =& 2 & E(filling, baking) &= 4\\ E(dough, baking) =& 4 & E(baking, applepie) &= 4 \end{aligned}$$

- The cost functions C is defined as[7]:

$$\begin{aligned} C(makedough) &= -€\ 5\\ C(makefilling) &= -€\ 6\\ C(baking) &= -€\ 14 \end{aligned}$$

□

In figure 4.4, the values of C and the values of E label respectively transitions and arcs.

Analogously to a PTNS, we define a WPTN by associating to a WPTNS an initial marking $\mathcal{M}_0$.

Definition 4.2 (WPTN)**.** A WPTN is a pair $(N, \mathcal{M}_0)$, where N is s WPTNS, and $\mathcal{M}_0$is a multiset of places that stands for its initial marking.

The initial marking in a PTN represents the initial state of a discrete dynamic systems. The very same semantics is inherited by WPTNs.

Example 4.7. The initial marking $\mathcal{M}_0$ for the WPTNS in figure 4.4 *Grandma & co* is:

$$\mathcal{M}_0 = 200'sugar + 100'flour$$

In figure 4.5, we graphically depict the initial marking of the WPTNS in figure 4.4.

4.4.2 Dynamics of WPTNs

WPTNSs and WPTNs preserve all the properties of PTNSs and PTNs respectively, but allow the quantitative representation of the cost of a transition. Therefore, we can naturally extend to them all the concepts employed for PTNs. Those include the concepts of

[7]The sign convention employed is negative values each time an auctioneer incurs in a cost.

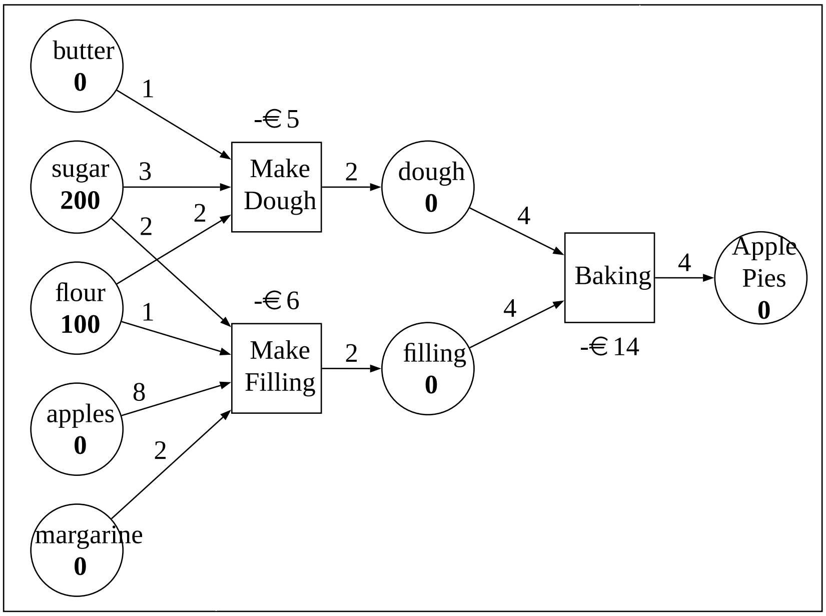


Figure 4.5: WPTN associated to example 4.1.

enabling of a transition, firing of a transition, marking, firing sequence, and so on (refer to section 2.3).

In a PTN, if a transition is enabled in a marking it can *fire*. If a transition fires it consumes some input goods and produces some output goods. In a WPTN, something more happens. If a transition fires it carries out a cost, the cost associated to the fired transition.

Example 4.8. In table 4.5 we show what happens when the *Make Dough* transition fires. The transition generates a cost of €5. In the upper right corner we show the quantity of money spent by the auctioneer in the corresponding state.

□

What does it happen when there is a sequence of firings? Firstly, the WPTN will evolve through a succession of markings (states); and secondly, a cost will be associated to such a sequence of transitions (*firing sequence* in section 2.3.1). Considering this, we can define the notion of *cost of a firing sequence* (C_{FS}) as:

Definition 4.3 (Cost of a firing sequence)**.** The cost C_{FS} associated to a firing sequence $J = \langle t_1, t_2, ..., t_d \rangle$ is the sum of all the costs of the transitions contained in the sequence:

$$C_{FS}(J) = \sum_{i=1}^{d} C(t_i) \tag{4.6}$$

If a transition fires more than once, say k times, then its cost will be added k times.

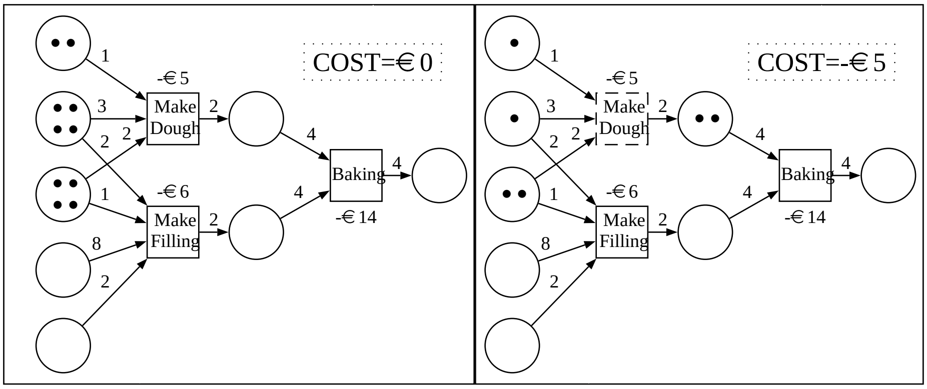


Table 4.5: Cost of executing a manufacturing operation on a WPTN.

Example 4.9. In figure 4.6, analogously to figure 4.3, we incorporate into a WPTN the bids expressed in equations (4.1) to (4.5). Notice that the costs labelling *bid transitions* is the cost associated to the bids. Furthermore, in table 4.6, we repeat the firing sequence of table 4.4 ($J = \{\mathcal{B}_1, makedough\}$) when a cost is associated to each transition. In this case, the cost associated to the firing sequence is $C_{FS}(J) = C(\mathcal{B}_1) + C(makedough)$ =-€ 200− € 5 =- € 205. In upper right corner of each frame of table 4.6 we highlight the cost associated to the corresponding firing.

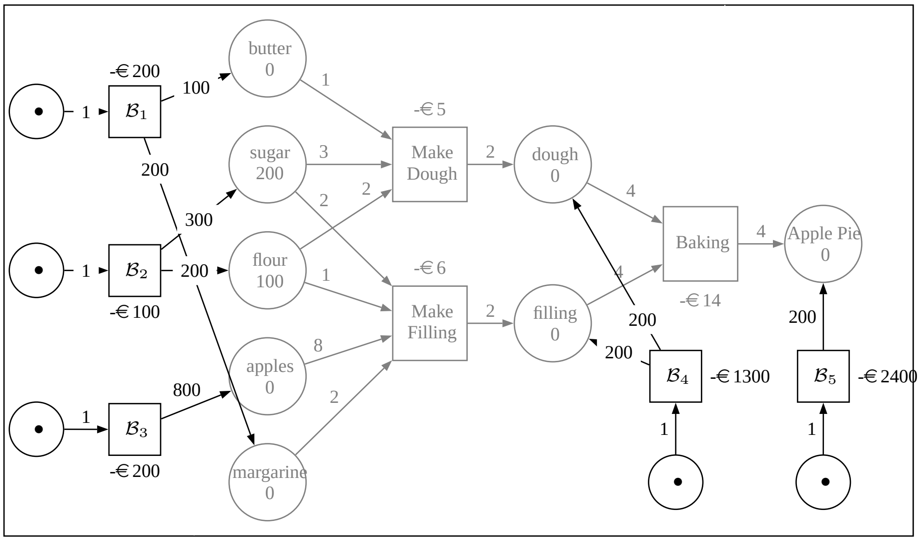
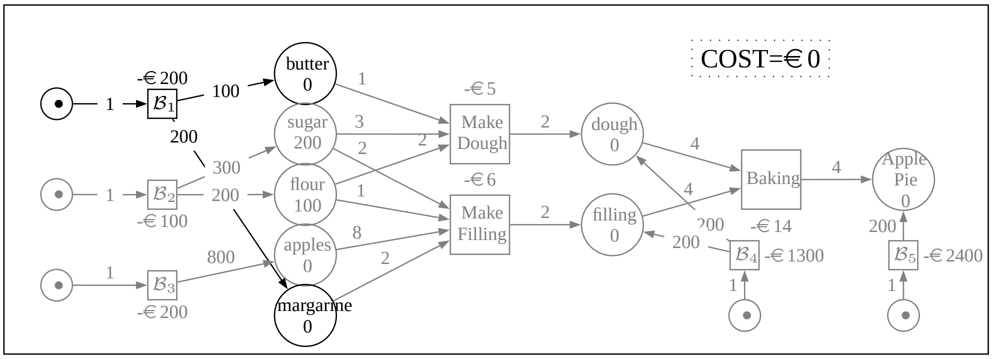
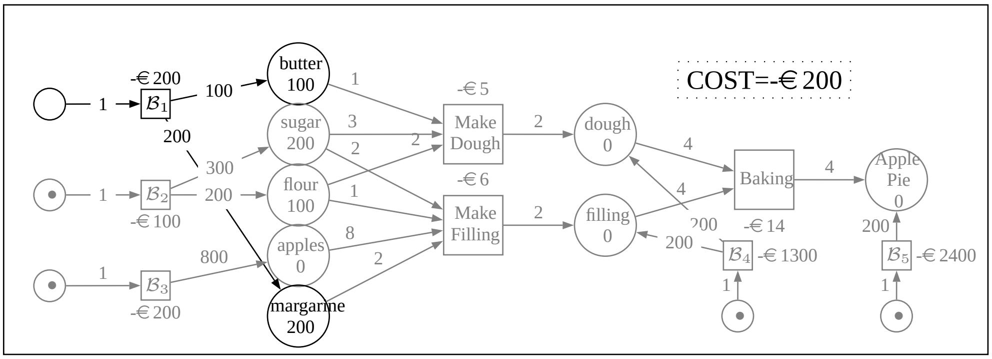


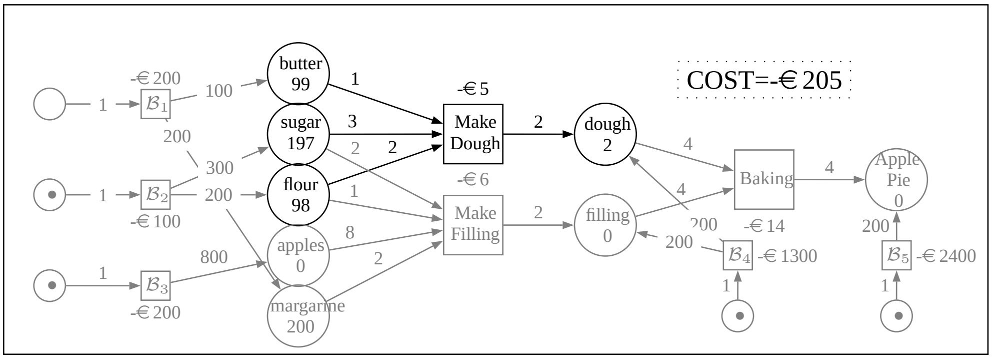
Figure 4.6: Incorporating bids into the WPTN of figure 4.5.



(a) Initially.



(b) After selecting bid $\mathcal{B}_1$.



(c) After performing *Make Dough*.

Table 4.6: Applying the firing sequence $J = \langle \mathcal{B}_1, makedough \rangle$.

4.5 Representing auction outcomes with WPTNs

In the previous section we introduced WPTNs and showed their powerful modelling features. The examples tried to give the intuitions behind the application of WPTN to our problem. In fact, we saw that the auctioneer faces a *make-or-buy* decision problem, and decides to solve it by means of combinatorial auctions. In this section, we aim at representing each of the outcomes of such auction given a description of the internal manufacturing operations, of the received bids, and of the auctioneer's final requirements . However, since an auctioneer is mostly interested in assessing the cost associated to each of such outcomes, we also associate an auctioneer's cost to each of the outcomes.

Then, firstly we introduce the *Transformability Network Structure* (TNS), a WPTN for modelling and communicating the internal manufacturing operations of an auctioneer. Secondly, we extend the TNS in order to incorporate the information regarding the received bids. This will result in the introduction of the *Auction Net*. This structure compactly expresses all the possible decisions an auctioneer may take, and quantifies the cost associated to each of such decisions. With those formal tools at hand, we can then define what a MUCRAtR is by providing an operational definition of valid auction outcome.

4.5.1 The Transformability Network Structure

In what follows we formally define the *Transformability Network Structure*. This corresponds to the net presented in figure 4.4. TNSs are useful for expressing the internal manufacturing operations of an auctioneer. This tool will have to quantitatively represent the input resources needed and consumed by each manufacturing operation, the output resources produced, the producer consumer relationships among the manufacturing operations, and the cost associated to each manufacturing operation. Summarising, a TNS describes the different ways in which goods can be transformed and at which cost. More formally,

Definition 4.4 (TNS)**.** A *transformability network structure* is a Weighted Place/Transition Net $N = (P, T, A, E, \mathcal{M}_0, C)$ such that we associate:

(1) the *places* in P to a set of goods G to negotiate upon[8].

(2) the *transitions* in T to a set of internal manufacturing operations;

(3) the *directed arcs* in A along with their weights E to the specification of the number of units of each good that are either consumed or produced by a manufacturing operation.

(4) the *initial marking* $\mathcal{M}_0$ to the quantity of each good initially available to the auctioneer (the stock). We indicate this particular initial marking with the multiset $\mathcal{U}_{in} \in \mathbb{N}^P$. Then, $\mathcal{M}_0 = \mathcal{U}_{in}$.

[8]Notice that a place represents a good. Thus, in what follows we will talk indifferently of *good places* and *goods*. That is, P and G are employed indifferently.

(5) a cost $C : T \to \mathbb{R}^+$ to each manufacturing operation.

In the next section we show how to incorporate the received bids into the TNS. The resulting WPTN is called *Auction Net*.

Example 4.10. The WPTN introduced in example 4.7 is the TNS associated to the problem of *Grandma & co*, previously described in example 4.1.

Notice that if an auctioneer communicates to the bidders its TNS along with some constraints on the final marking (for instance, at least 200 tokens in the apple pie place), the bidders have all the information for composing meaningful offers. This completely fulfills the CAs expressiveness limitation in communicating to bidders an auctioneer's requirements (issue (2) in table 4.1).

4.5.2 The Auction Net

In this section, we will thoroughly explain how to transform a TNS (figure 4.4) into an *Auction net* (figure 4.6). In the remaining of the chapter it is assumed that B is the set of received bids. Each bid is represented by a multiset $\mathcal{B} \in \mathbb{N}^P$ and has associated a cost encoded by function $C_B : B \to \mathbb{R}^+ \cup \{0\}$.

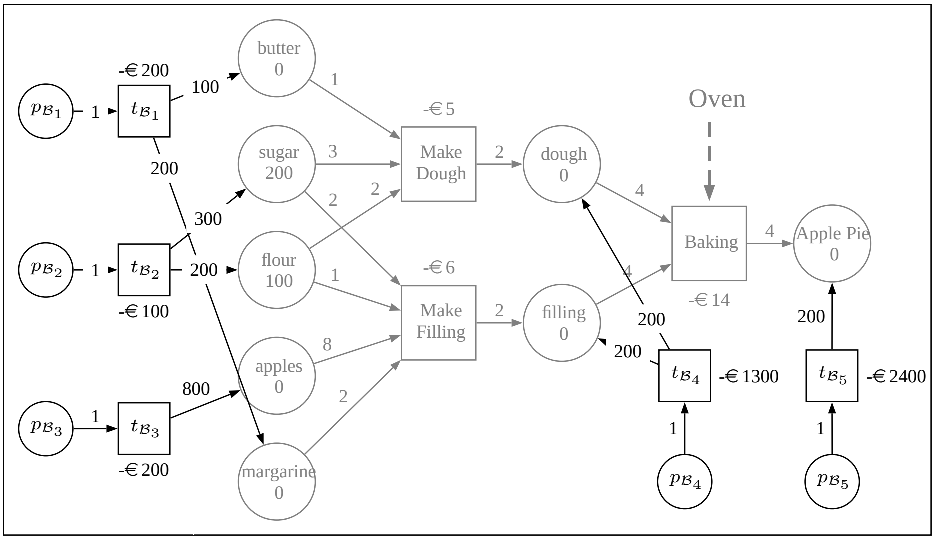


Figure 4.7: Auction Net of the MUCRAtR in example 4.1.

Definition 4.5 (Auction Net)**.** Given a set of bids B, and a TNS $N = (P, T, A, E, \mathcal{U}_{in}, C)$, an *Auction Net* is a WPTN $S^* = (P^*, T^*, A^*, E^*, \mathcal{M}_0^*, C^*)$ where:

$$\begin{cases} P^* &= P \cup P_B \\ T^* &= T \cup T_B \\ A^* &= A \cup A_B \end{cases}$$

(1) P_B is the set of *bid places*. That is, for each bid $\mathcal{B} \in B$ add a place $p_{\mathcal{B}}$.

(2) T_B is the set of *bid transitions*. That is, for each bid $\mathcal{B} \in B$ add a transition $t_{\mathcal{B}}$.

(3) A_B is the set of *bid arcs*. It is built as follows:

$$A_B = A_B^i \cup A_B^o$$

where

$$A_B^i = \{(p_{\mathcal{B}}, t_{\mathcal{B}}) \in P_B \times T_B \mid \forall \mathcal{B} \in B\} \quad (4.7)$$
$$A_B^o = \{(t_{\mathcal{B}}, p) \in T_B \times P \mid p \in \mathcal{B}\} \quad (4.8)$$

are the *input bid arcs* and *output bid arcs* respectively.

(4) The arc expression E^* function is built as follows:

$$E^*(x,y) = E(x,y) \qquad (x,y) \in A \quad (4.9)$$
$$E^*(t_{\mathcal{B}}, p) = \mathcal{B}(p) \qquad (t_{\mathcal{B}}, p) \in A_B^o \quad (4.10)$$
$$E^*(p_{\mathcal{B}}, t_{\mathcal{B}}) = 1 \qquad (p_{\mathcal{B}}, t_{\mathcal{B}}) \in A_B^i \quad (4.11)$$

(5) The cost function $C^* : T \cup T_B \to \mathbb{R}$ is built as follows:

$$C^*(t) = C(t) \qquad t \in T$$
$$C^*(t_{\mathcal{B}}) = C_B(\mathcal{B}) \qquad t_{\mathcal{B}} \in T_B$$

(6) The initial marking is defined as

$$\mathcal{M}_0^*(p) = \begin{cases} \mathcal{U}_{in}(p) & p \in P \\ 1 & p \in P_B \end{cases} \quad (4.12)$$

□

Example 4.11. We extend the *TNS* of example 4.6 with the bids listed in equations (4.1) to (4.5). This gives raise to the *Auction Net* in figure 4.7. $(P, T, A, E, \mathcal{M}_0, C)$ have been defined in example 4.6. Then, $S^* = (P^*, T^*, A^*, E^*, \mathcal{M}_0^*, C^*)$ is defined as follows:

(1) $P^* = P \cup \{p_{\mathcal{B}_1}, p_{\mathcal{B}_2}, p_{\mathcal{B}_3}, p_{\mathcal{B}_4}, p_{\mathcal{B}_5}\}$

(2) $T^* = T \cup \{t_{\mathcal{B}_1}, t_{\mathcal{B}_2}, t_{\mathcal{B}_3}, t_{\mathcal{B}_4}, t_{\mathcal{B}_5}\}$

(3) $A^* = A \cup A_B^i \cup A_B^o$ where

$$A_B^i = \{(p_{\mathcal{B}_1}, t_{\mathcal{B}_1}), (p_{\mathcal{B}_2}, t_{\mathcal{B}_2}), (p_{\mathcal{B}_3}, t_{\mathcal{B}_3}), (p_{\mathcal{B}_4}, t_{\mathcal{B}_4}), (p_{\mathcal{B}_5}, t_{\mathcal{B}_5})\}$$
$$A_B^o = \{(t_{\mathcal{B}_1}, butter), (t_{\mathcal{B}_1}, margarine), (t_{\mathcal{B}_2}, sugar), (t_{\mathcal{B}_2}, flour), \ldots\}$$

(4) $E^*(x,y) = E(x,y)$ if $(x,y) \in A$. When $(x,y) \in A_B$ we have:

$$\begin{array}{rr} E^*(t_{\mathcal{B}_1}, butter) = 100 & E^*(t_{\mathcal{B}_1}, margarine) = 200 \\ E^*(t_{\mathcal{B}_2}, sugar) = 300 & E^*(t_{\mathcal{B}_2}, flour) = 200 \\ \dots & \dots \\ E^*(p_{\mathcal{B}_1}, t_{\mathcal{B}_1}) = 1 & E^*(p_{\mathcal{B}_2}, t_{\mathcal{B}_2}) = 1 \\ \dots & \dots \end{array}$$

(5) $C^*(t) = C(t)$ when $t \in T$. When $t \in T_B$ we have:

$$\begin{array}{ll} C^*(t_{\mathcal{B}_1}) = \text{-€}200 & C^*(t_{\mathcal{B}_2}) = \text{-€}100 \\ C^*(t_{\mathcal{B}_3}) = \text{-€}200 & C^*(t_{\mathcal{B}_4}) = \text{-€}1300 \\ C^*(t_{\mathcal{B}_5}) = \text{-€}2400 & \end{array}$$

□

Recall that by means of the PTN defined in example 4.5, an auctioneer was able to compactly represent all the possible outcomes associated to any of its decisions. However, he had the problem to assess the cost associated to each of such outcomes. Notice that by means of the auction net, the auctioneer can now express both the outcomes of its decisions and the cost associated to each of them.

In order to define the winner determination problem for MUCRAtR one further step is required. We have to define an optimisation problem whose solution retrieves the optimal firing sequence to apply to the auction net in order to obtain a desired final marking (in the case of *Grandma & co* more than 200 tokens in the apple pie place). This is the purpose of the following section.

4.5.3 Constrained Maximum Weight Occurrence Sequence Problem

Since there is a cost associated to each transition, one may be interested in finding a maximum (minimum[9]) cost firing sequence leading from an initial marking to some final marking. More importantly, one may be interested in finding a maximum cost firing sequence leading from an initial marking $\mathcal{M}_0$ to a final marking $\mathcal{M}_d$ that *fulfils a set of inequality constraints*. For instance, we may want to impose that in a final marking $\mathcal{M}_d$ each place contains exactly one token ($\mathcal{M}_d(p) = 1, \forall p \in P$), or at least 200 tokens in a given place (for instance, the *Apple Pie* place in example 4.1 $\mathcal{M}_d(applepie) \geq 200$). With this aim we define the *Constrained Maximum Weight Occurrence Sequence Problem* (CMWOSP).

Definition 4.6 (CMWOSP)**.** Given a WPTN $N = (P, T, A, E, \mathcal{M}_0, C)$, a set of inequality/equality constraints that a final marking $\mathcal{M}_d$ must fulfil, expressed as:

$$\forall p \in P \;\; \mathcal{M}_d(p) \Delta_p h_p \tag{4.13}$$

[9]In any optimization problem maximising and minimising are two dual representations of the very same problem. We will talk about maximisation in what follows, but all the results can be easily applied to a minimisation.

where $\Delta_p \in \{<, \leq, =, \geq, >\}$ and $h_p \in \mathbb{N} \cup \{0\}$, find an occurrence sequence $J_{opt} = \langle u_1, u_2, ..., u_d \rangle$ that brings the initial marking $\mathcal{M}_0$ to a final marking $\mathcal{M}_d$ such that: (1) $\mathcal{M}_d$ fulfils all the constraints in equation (4.13); and (2) J_{opt} maximises the total cost C_{FS}.

We can express the inequations (4.13) in matrix form:

$$M_d \mathbf{\Delta h} \tag{4.14}$$

where M_d is a vector whose $i-th$ component represents the number of tokens in place i, $\mathbf{\Delta}$ is a vector whose $i-th$ element contains $\{<, >, \leq, \geq, =\}$, and $\mathbf{h}$ is a vector whose $i-th$ element contains h_p. We will call the constraints in equation (4.13) or (4.14) the *final marking constraints*.

Proposition 4.1. *CMWOSP is at least EXPSPACE-hard.*

Proof. The reachability problem for PTN can be reduced to a CMWOSP. It has been proved that the reachability problem is EXPSPACE-hard (Lipton, 1976). □

4.6 The Winner Determination Problem

In this section, we formally define the *winner determination problem* for MUCRAtR.

Informally, given a TNS expressing the internal manufacturing operations of an auctioneer over a set of goods G, an auctioneer's final requirements $\mathcal{U}_{out} \in \mathbb{N}^G$, and a set of received bids B, the *winner determination problem* amounts to finding the set of bids and internal operations that minimise the auctioneer's cost and produce at least the required goods.

The formal definition of the WDP relies on the *Auction Net*.

Definition 4.7 (Winner Determination Problem)**.** Given an auction expressed as $\langle N, \mathcal{U}_{out}, B \rangle$, where $N = (P, T, A, E, \mathcal{M}_0)$ is a TNS, $\mathcal{U}_{out} \in \mathbb{N}^G$ expresses the auctioneer final requirements, and B is the set of received bids. Let $S^* = (P^*, T^*, A^*, E^*, \mathcal{M}_0^*, C^*)$ be the corresponding *Auction Net*. The *Winner Determination Problem* amounts to selecting the set of bids B^* and the sequence of internal operations J^* that both minimise the auctioneer's cost and satisfy the the following *final marking constraints* on the *Auction Net*:

$$\mathcal{M}_d(p) \geq \mathcal{U}_{out}(p) \qquad \forall p \in P \tag{4.15}$$
$$\mathcal{M}_d(p) \geq 0 \qquad \forall p \in P_B \tag{4.16}$$

Proposition 4.2. *The WDP for a MUCRAtR $\langle N, \mathcal{U}_{out}, B \rangle$ can be reduced to a CM-WOSP on the corresponding auction net. Such a CMWOSP is characterised by the following* final marking constraints*:*

$$\mathcal{M}_d(p) \geq \mathcal{U}_{out}(p) \qquad \forall p \in P \tag{4.17}$$
$$\mathcal{M}_d(p) \geq 0 \qquad \forall p \in P_B \tag{4.18}$$

Proof. The proof is by construction:

(1) Solve the CMWOSP on the *Auction Net* N_B. We name the CMWOSP solution J^{min}.

(2) The set of winning bids B^* corresponds to the *bid transitions* contained in J^{min}:

$$B^* = \{\mathcal{B} \in B | t_{\mathcal{B}} \in J^{min}\} \tag{4.19}$$

(3) The sequence J^* of internal manufacturing operations that an auctioneer has to perform internally is obtained by removing from J^{min} all the transitions that are not *operation transitions*. We denote this as follows:

$$J^* = J^{min}_{|T} \tag{4.20}$$

□

Notice carefully that in a CMWOSP the sum of the weights associated to the overall transitions is maximised. However, since negative costs are associated to both bid transitions and operation transitions, maximising the sum of the weights implies minimising the auctioneer's costs.

Example 4.12. If *Grandma & co* receives the bids in equations (4.1) to (4.5), the decision minimising its costs and allowing it to obtain the 200 apple pies is:

(1) to select bid $\mathcal{B}_4$ to obtain *dough* and *filling*; and

(2) to subsequently bake them at *Grandma & co* after running fifty times the *Baking* operation.

If we look at it on the WPTN, this corresponds to the firing sequence

$$J = \langle \mathcal{B}_4, \underbrace{Baking, Baking, Baking, \ldots, Baking}_{\text{50 times}} \rangle \tag{4.21}$$

Then, the cost of this decision is assessed as follows:

$$cost(\mathcal{B}_4) + 50 \cdot cost(\textit{Baking}) = -€1300 - €\ 700 = -€2000. \tag{4.22}$$

The reader can check that this is the best possible option for the auctioneer: it exploits the initial stock, it brings to a marking that fulfils *Grandma & co* requirements, it minimises the costs.

□

Finally, the optimisation problem of the auctioneer is clearly stated, and there is a rule for selecting the winners. Thus, we have solved issue (4) in table 4.1 as well. Since we obtained this result by directly employing place transition nets, we can import all the techniques employed for them. As a first example, we show how to solve the winner determination problem by means of Integer Programming (see section 2.1). With this aim, we just show that some particular CMWOSP can be solved by means of Integer Programming.

4.7 Solving the WDP by means of IP

In this section, firstly we show that the CMWOSP can be solved by means of Integer Programming under some special conditions. Then, we show that those conditions are fulfilled when the underlying PTN is acyclic. Finally, we explicitly state the IP solving the WDP.

4.7.1 Solving the CMWOSP by means of IP

In section 2.3.2, we showed that under some hypothesis on a PTN, it is possible to express its overall reachability set by means of an equation, the state equation (see section 2.3.3). The state equation describes all the states that an acyclic PTN can reach, and it is a linear equation. That is all we need to generate our integer program.

We recall also that, by means of the *state equation*, it is possible to represent in matrix form the firings and markings of a PTN (see section 2.3.2):

- Let us associate to each place $p_i \in P$ a position i in a vector $M_k \in \mathbb{N}^{|P|}$. The integer contained in the $i-th$ position of the M_k vector corresponds to the number of tokens contained in a a place p_i after k firings in some sequence. Then, M_0 is the initial marking, M_1 is the marking obtained after the firing of some transition, and so on.

- Let us associate to each transition $t_j \in T$ a position j in a vector of integers $\mathbf{x} \in \mathbb{N}^{|T|}$. The integer contained in the $j-th$ position of $\mathbf{x}$ encodes the number of times transition t_j has been fired.

With this representation, the state equation can be written as:

$$M = M_0 + A^T\mathbf{x} \tag{4.23}$$

The very same formalism holds for WPTN. In fact, the only difference is that there is a cost associated to each transition. Then, can we represent in matrix form the cost of a sequence bringing from M_0 to M via the transitions encoded in $\mathbf{x}$ as well? The answer is quite easy. Notice that $\mathbf{x}$ in equation (4.23) stands for the number of times each transition is fired for transforming marking M_0 into marking M. Then, if we know the cost of each transition, according to definition in equation (4.6), we have to multiply the cost of each transition by the number of times it is fired. Then, we define a vector $C_T \in \mathbb{R}^{|T|}$ whose $j-th$ position represents the cost associated to transition t_j ($C_{FS}(t_j)$). Hence, the cost associated to the firing sequence represented by $\mathbf{x}$, noted as $J_{\mathbf{x}}$, is:

$$C_{FS}(J_{\mathbf{x}}) = \mathbf{x}^T C_T \tag{4.24}$$

The idea behind the mapping to IP is finding a set of linear equations that:

(1) constrains the decision variables associated to transitions to hold a value encoding a valid firing sequence;

(2) constrains the marking obtained by firing the selected transitions to fulfil a set of equality/inequality constraints; and

(3) maximises the sum of the costs associated to the selected transitions.

Notice that point (1) can be easily fulfilled when the net is acyclic by means of the *state equation*. Since the *state equation* represents all the reachable states, it is enough to apply to it a set of inequality/equality constraints to fulfil point (2). Finally, since in a WPTN a cost is associated to each transition, maximising the cost associated to the selected *firing sequence* we satisfy point (3) as well.

In what follows we go into the formal details of what we explained above. The following theorem states that if we can represent all the reachable states of a PTN by means of the state equation, then the CMWOSP can be solved by means of IP.

Theorem 4.1. *Consider an WPTN $(P, T, A, E, \mathcal{M}_0, C)$ with incidence matrix*[10] $\mathbf{A}$. *If the state equation describes all the reachable states M of the WPTN, then all the non-negative integer solutions the following integer program:*

$$\textit{max} \quad \mathbf{x}^T c_T \tag{4.25}$$

$$\textit{subject to} \ \ \mathcal{M}_0 + \mathbf{A}^T\mathbf{x} \ \mathbf{\Delta h} \tag{4.26}$$

represent the firing count vectors of all the optimal solutions to the CMWOSP defined by $\langle \sim, \mathbf{h} \rangle$

Proof. Notice that equation (4.26) simply imposes that the end marking fulfils the constraints defined by $\langle \sim, \mathbf{h} \rangle$ in equation (4.13). Equation (4.25) maximises the cost $C_{FS}(J_\mathbf{x})$, associated to the firing sequence represented by $\mathbf{x}$ (see equation (4.24)). As a result, a solution $\mathbf{x}^*$ to the IP defined by equations (4.25) and (4.26) optimises the sum of the costs associated to fired transitions, while ensuring that the final marking is reachable and fulfils the constraints defined by $\langle \sim, \mathbf{h} \rangle$. $\square$

According to the results stated in theorem 2.2, it is possible to express the reachability set with the state equation when the PTN is acyclic. Then, we apply this result to our problem via the following corollary:

Corollary 4.1. *Provided that a WPTN is acyclic, every CMWOSP defined on it can be mapped into integer linear programming.*

Proof. . Since the WPTN is acyclic, in virtue of theorem 2.2, all the reachable states M are the non-negative integer solutions of equation (2.26). Then, for theorem 4.1 the firing count vectors of all the solutions to the CMWOSP are the solutions to the IP in equations (4.25) and (4.26).

Hence, we solve the CMWOSP problem in two steps. First, we determine the optimal firing count vector $\mathbf{x}^{opt}$ by solving the Integer Linear Program (ILP) in equations (4.25) and (4.26). Then, we construct J_{opt} from $\mathbf{x}^{opt}$, for which each step is enabled. Since S is acyclic, we can establish a partial order among transitions so that $t_1 < t_2$ iff t_2 uses as input some output of t_1. We can construct an occurrence sequence J_{opt} by ordering the transitions in the firing count vector $\mathbf{x}_{J_{opt}}$ non-decreasingly according to our partial ordering. Every step in the so ordered occurrence sequence is guaranteed to be enabled. The occurrence sequence J_{opt} is consequently the solution to our CMWOSP. $\square$

[10] Refer to section 2.3.3.

Thus, we can also cope with requirement (5) in table 4.1.

4.7.2 The IP Formulation in practise

We have shown that the CMWOSP can be solved by means of an ILP in the case that the underlying WPTN is acyclic in section 4.7.1. We showed in section 4.6 that the winner determination problem for MUCRAtR is a CMWOSP. In this section we show that the WDP for MUCRAtR can be solved by means of IP when the auctioneer's TNS is acyclic. Furthermore we will explicitly write down the IP model.

The first assumption is that no cycles are added when we extend a TNS into an Auction Net. This is very easy to show.

Proposition 4.3. *Given an acyclic TNS* $(P, T, A, E, \mathcal{M}_0, C)$*, the corresponding Auction Net* $(P \cup P_B, T \cup T_B, A \cup A_B, E_B, \mathcal{M}_0^{Bid}, C_{Bid})$ *will also be acyclic.*

Proof. Say that there is a *bid transition* $t_{\mathcal{B}}$ that includes a cycle that was not present in the TNS. The output places of bid transitions are always in P (see definition 4.5). Then, in order to have a cycle, there should be a transition with input places in P that has the input place of $t_{\mathcal{B}}$ as an output place. However, this is impossible since, according to definition 4.5, the input places of *bid transitions* have only output arcs. □

Naturally, it follows that:

Corollary 4.2. *When the TNS is acyclic, the WDP can be solved by means of IP.*

Next, we explicitly express the IP model solving the *make-or-buy* decision problem, or equivalently solving the WDP for MUCRAtR.

The mathematical model is built according to the following rules:

(1) there are n goods, indexed with $i \in \{1, 2, \ldots, n\}$

(2) there are m internal manufacturing operations, indexed with $j \in \{1, 2, \ldots, m\}$

(3) there are l multi-unit combinatorial bids, indexed with $k \in \{1, 2, \ldots, l\}$

(4) a_{ij} is the difference between the weight of the arc connecting operation transition j to good i and the weight of the arc connecting good i to operation transition j in the auction net. Formally, in the WPTN language $a_{ij} = E(j, i) - E(i, j)$. Informally, this represents the flow of tokens in place i when transition j is fired.

(5) u_i^{in} is the quantity of good i initially available to the auctioneer (the stock).

(6) u_i^{out} is the quantity of good i finally required by the auctioneer (the sale forecast).

(7) b_{ki} is the weight of the arc connecting bid transition k to good i.

(8) bp_k is the weight of the arc connecting the bid place p to the bid transition k.

(9) bu_k^{in} is the quantity of tokens initially available in bid place of bid k[11].

[11]Notice that we know that this is always one. However, for the sake of generality we consider it as a parameter.

(10) c_k is the cost associated to internal operation j.

(11) p_k is the price associated to bid k.

(12) $y_k \in \mathbb{N} \cup \{0\}$ is an integer decision variable (for each bid $k \in \{1, 2, \ldots, l\}$) taking on value w if bid k has been selected w times[12].

(13) $x_j \in \mathbb{N} \cup \{0\}$ is an integer decision variable (for each transition $j \in \{1, 2, \ldots, m\}$) taking on value w if transformation j is fired w times in the optimal firing sequence.

With this in mind, the IP model is expressed with the following equations:

$$\max \quad \sum_k y_k \cdot p_k + \sum_j x_j \cdot c_j \tag{4.27}$$

$$u_i^{in} + \sum_k y_k \cdot b_{ki} + \sum_j x_j \cdot a_{ij} \geq u_i^{out} \qquad \forall i \tag{4.28}$$

$$bu_k^{in} - y_k \cdot bp_k \geq 0 \qquad \forall i \tag{4.29}$$

Equation (4.27) minimises (recall the the costs are negative) the sum of the costs associated to bids plus the costs associated to internal manufacturing operations. Equations (4.29) and (4.28) correspond to equation (4.13) of the CMWOSP. We split it into two equations since they implement different inequalities. This is made clear if compared with equations (4.12), (4.17), and (4.18). Indeed, equation (4.28) implements equation (4.17), whereas equation (4.29) implements equation (4.18).

If we observe equation (4.29), and recall that $bu_k^{in} = 1$ for all k (see equation 4.12), and that $bp_k = 1$ for all k (see equation 4.11), the equation becomes:

$$1 - y_k \geq 0 \qquad \forall k \tag{4.30}$$

Considering that y_k is an integer decision variable, it turns out clear that it becomes a binary decision variable $y_k \in \{0, 1\}$. Hence, the whole optimisation problem in equations (4.27) to (4.29) can be rewritten under this hypothesis:

$$\max \quad \sum_k y_k \cdot p_k + \sum_j x_j \cdot c_j \tag{4.31}$$

$$u_i^{in} + \sum_k y_k \cdot b_{ki} + \sum_j x_j \cdot a_{ij} \geq u_i^{out} \qquad \forall i \tag{4.32}$$

This ILP can be readily implemented with the aid of an optimisation library (see section 2.1.2). The number of decision variables needed to encode this problem is $|T| + |B|$, where T is the set of internal supply chain operations and B is the set of received bids. The number of required constraints is $|G|$, where G is the set of goods.

[12]Notice carefully that we know that this variable can take only value 0 or 1. Then, it is a binary decision variable. However, in order to be formal, we hypothesise that is an integer variable for the moment.

4.7.3 Comparison with a traditional MUCRA IP solver

In what follows we compare the IP formulation of the MUCRAtR WDP with the IP formulation of a traditional Multi-unit Combinatorial Reverse Auction (MUCRA) WDP. In order to solve the WDP for a MUCRA, as formalised in (Sandholm et al., 2002), we exploit the equivalence to the multi-dimensional knapsack problem pointed out in (Holte, 2001). Sandholm et al. in (Sandholm et al., 2002) show how MUCRA can be solved by means of IP. In this case the problem is stated by means of the following parameters and variables:

(1) there are n goods, indexed with $i = \{1, 2, \dots, n\}$

(2) there are l multi-unit combinatorial bids, indexed with $k = \{1, 2, \dots, l\}$

(3) u_i^{out} is the quantity of good i finally required by the auctioneer.

(4) b_{ki} is the quantity of good i offered in bid k.

(5) p_k is the price associated to bid k.

(6) $y_k \in \{0, 1\}$ is a binary decision variable (for each bid $k \in \{1, 2, \dots, l\}$) taking on value 1 if bid k has been selected and 0 otherwise.

Then, the problem of selecting the best offers can be expressed with the following IP model:

$$\max \sum_k y_k \cdot p_k \tag{4.33}$$

$$\sum_k y_k \cdot b_{ki} \geq u_i^{out} \qquad \forall i \tag{4.34}$$

In this case the number of decision variables is $|B|$, and the number of constraints is $|G|$. Then, our formulation of the WDP can be clearly regarded as an extension of the ILP we must solve for a MUCRA (as formalised above). In fact, the second component of expression 4.31 changes the overall cost as transformations are applied, whereas the second component of expression 4.32 makes sure that the units of the selected bids fulfil a buyer's requirements taking into account the units consumed and produced by transformations.

Observe the analogy between the IP in equations (4.31) and (4.32), and the IP in equations (4.33) and (4.34).

The first terms of both IP are equivalent. In the MUCRAtR IP we add the contributions due to the firing of transformations. It seems a trivial extension. However, notice carefully that we showed that this cannot be done for every possible class of nets.

4.8 Conclusions

In this chapter we dealt with the *make-or-buy* decision problem when combinatorial offers are received by an auctioneer. We identified and overcome all the limitations that prevent the applicability of CAs to *make-or-buy* decisions. These are grouped into *Expressiveness*, *Winner Determination*, and *Formal Analysis* limitations.

We showed that PTNs are very useful for overcoming some of the CAs expressiveness limitations. However, due to their inability to express costs, we had to extend PTNs in order to fully represent the internal production and cost structure of an auctioneer. This lead to the definition of Weighted Place/Transition Nets (WPTN), PTNs in which it is possible to associate a cost to each transition firing.

By means of WPTN all the expressiveness issues are solved. Then, an auctioneer can employ WPTNs to define its production and cost structure. We called such a representation a *Transformability Network Structure* (TNS).

However, the TNS must be linked someway with the offers received from bidders (the bids). Then, we extended the TNS in order to incorporate the information about the received combinatorial bids. This lead to the definition of the *Auction Net*. An *Auction Net* is a WPTN that incorporates all the information about the running auction: internal manufacturing operations and received offers.

Once the decision problem input is conveniently expressed, we formalise its output. With this aim, we introduced a new reachability problem on WPTNs, the *Constrained Maximum Weight Occurrence Sequence Problem* (CMWOSP). We subsequently employed the CMWOSP to formally define the auctioneer decision problem, or equivalently, the winner determination problem.

Next, via the exploitation of some well-known results of PTN theory, we succeeded in mapping the optimisation problem to an IP model, that can be directly solved by means of commercial or free optimisation libraries. However, such a solver can be applied only in the case that the net is acyclic.

Notice as well that the representation via WPTN allows to import a wide body of methods and tools associated to *PTNs*. As a first example of this powerful approach, we provided the above mentioned mapping to integer programming of the MUCRAtR WDP.

It seems quite natural at this point to consider an extension. If an auctioneer can incorporate into the auction its internal operations, why not to incorporate information about the bidders' internal operations as well? That is, in a MUCRAtR an auctioneer decides whether to produce in-house or to buy as already made the goods he requires. However, there is a third possibility, a bidder may offer to *perform* an operation for the auctioneer. In such a case, the auctioneer would be able to outsource not only goods, but manufacturing operations or services as well. In the following chapters we discuss in depth such extension.

Chapter 5

Mixed Multi unit Combinatorial Auctions

Along the lines of what we have done in chapter 4, where we introduced MUCRAtR to cope with the *make-or-buy* decision problem, in this chapter we provide a new type of combinatorial auction (CA) to deal with *make-or-buy-or-collaborate* decision problems. This new auction type is called *Mixed Multi-unit Combinatorial Auction* (MMUCA). It supports the trading of any operation across a supply chain: from supply and request of goods to the request and offer of manufacturing operations and services. In this chapter we introduce:

- a formal language that allows bidders to express offers and requests over such supply chain operations;
- a formalisation of the optimisation problem faced by an auctioneer when selecting the set of bids that maximises its revenue;

This chapter and the two following deal with different aspects of the *make-or-buy-or-collaborate* decision problem. In particular, this chapter deals with *expressiveness* requirements and formalises the *decision problem* faced by the auctioneer, whereas chapters 6 and 7 deal with computational and formal analysis aspects associated to the decision problem.

This chapter is organised as follows. In section 5.1, we describe the requirements of CAs when applied to the *make-or-buy-or-collaborate* decision problem. In section 5.2, we introduce an example that helps clarifying the *make-or-buy-or-collaborate* decision problem. In section 5.3 we introduce a novel formal language that supports the negotiation of supply chain operations. In section 5.4, we formally define an allocation rule that automates the *make-or-buy-or-collaborate* decision, that is, we formalise the decision problem that the auctioneer faces. In section 5.5 we list the auction models subsumed by MMUCA. Finally, in section 5.6, we draw some conclusions and remarks about the expressiveness of the defined formal language and about the types of auction subsumed by our model.

5.1 Beyond CAs for Supply Chain Formation

In chapter 4, we studied the *make-or-buy* decision problem under the hypothesis that complementarities among goods exist on the bidder side. In order to solve such a problem, we introduced a new type of combinatorial auction, the Multi Unit Combinatorial Reverse Auction with transformability Relationships among Goods (MUCRAtR). In this chapter instead, we deal with the *make-or-buy-or-collaborate* decision problem, namely the problem of selecting the most convenient supply chain partners. In this case, a new dimension is added to the decision problem. In order to find a profitable agreement, the parts negotiating a collaboration across the supply chain, have to make explicit and share some information about their internal production structure.

We approach this problem employing a market-based mechanism. Analogously to chapter 4, we build upon combinatorial auctions since they help capturing the production complementarities arising within a supply chain. We introduce a new type of combinatorial auction that allows an auctioneer to trade, besides goods, operations across the supply chain. As thoroughly explained in section 1.4.2, the operations that can be negotiated across a supply chain are:

(1) *Supply of manufacturing, assembly, disassembly operations.*

(2) *Request of manufacturing, assembly, disassembly operations.*

(3) *Supply of goods.*

(4) *Request of goods.*

Combinatorial auctions for supply chain formation (SCF), introduced by Walsh et al. in (Walsh and Wellman, 2003), have been the first attempt to deal with the problem of supply chain formation by means of combinatorial auctions. Supply chain formation is the problem of selecting the set of participants in a supply chain, and of assessing who will exchange what with whom, while maximising the utility of the participants. We consider the supply chain formation problem similar to a certain degree to the *make-or-buy-or-collaborate* problem. In fact, the objective of SCF is to provide to the supply chain stakeholders a mechanism to select the best way of collaborating among them. Combinatorial auctions for SCF relies on the Task Dependency Networks (TDN) to represent the production relationships among the supply chain stakeholders. However, as illustrated in chapter 1, some intrinsic limitations of TDNs hinder their application to the *make-or-buy-or-collaborate* decision problem.

In table 5.1 we illustrate the requirements that we aim at fulfilling when dealing with the *make-or-buy-or-collaborate* decision problem. In the table, we also mark the requirements that are fulfilled by CAs and TDNs. Summarising, we can classify the emerging requirements in three types:

(1) *expressiveness* requirements (1–8 in table 5.1);

(2) *WDP* requirements (9–13 in table 5.1); and

(3) *computational* requirements (14–19 in table 5.1); and

	Requirements	**CAs**	**TDN**
1	express an offer/request on bundles of goods	✓	✓
2	express an offer of a SCO with a single output product		✓
3	express an offer of a SCO with multiple output products		
4	express a request of a SCO		
5	express the offer/request of a bundle of SCOs		
6	express combinations of bids	✓	
7	express the min/max number of times SCOs are performed		
8	express resource sharing		
9	express an auctioneer's initial stock		
10	express the auctioneer's final requirements		
11	support *acyclic* supply chain networks		✓
12	support *cyclic* supply chain networks		
13	compute the *scheduled sequence* of SCOs to perform		
14	ensure computational tractability while preserving optimality		
15	solve SCF decision problem		✓
16	solve the *make-or-buy-or-collaborate* decision problem		
17	formally represent the search space		
18	graphically represent the search space		
19	assess the computational tractability based on the problem structure		

Table 5.1: Requirements associated to the *make-or-buy-or-collaborate* problem.

As to expressiveness requirements, it is clear that an auctioneer intending to trade any possible operation across the supply chain must provide bidders with a language for expressing their preferences over such operations (requirements 1–8 in table 5.1). Since we build upon CAs, in this chapter we firstly introduce a novel bidding language that extends and generalises bidding languages for combinatorial auctions (bidding languages for CAs are summarised in section 3.2.2). The purpose of CA bidding languages is to predicate about goods, in particular about bundles of goods. However, in our case the language must also allow to predicate about operations across the supply chain. In order to cope with this requirement, we define *supply chain operations* (SCOs). A supply chain operation is a concept that unifies under the same name some[1] of the supply chain operations, namely:

- *supply* of a manufacturing operation;
- *supply* of a bundle of goods; and
- *request* of a bundle of goods.

This abstraction considers that the only distinguishing features of a supply chain operation are:

- the set of required and consumed inputs
- the set of produced outputs

[1]We say *some* of the operations since the *request* of a supply chain operation cannot be expressed as an atomic operation. This point is clarified further on.

Thus, while in combinatorial auctions a bidder bids on bundles of goods, in this case the objects predicated in the bidding language are bundles of SCOs. More precisely, as pointed out by requirement (5) in table 5.1, bidders must be able to express preferences over bundles of supply chain operations, and in particular to utter offers for operations, requests of operations, and offer/request alternatives. Hence, we solve those problems by letting bidders:

(1) specify valuations over bundles of supply chain operations. An *atomic bid* will allow bidders to associate a value to a bundle of supply chain operations. The semantics of atomic bids will be rich enough to specify both requests and offers for bundles of supply chain operations (requirement (5) in table 5.1);

(2) specify combinations of atomic bids (requirement (6) in table 5.1).

Thus, on the one hand forming *atomic bids* joining *supply chain operations* perfectly captures potential complementarities among such operations. On the other hand, we provide bidders a way to express combinations of bids representing alternative offers. This is needed because the preferences of a bidder cannot be fully expressed only by atomic bids. For this purpose, we introduce a bidding language with several constructs allowing the representation of several types of preferences over set of atomic bids. For instance, *XOR bids* allow a bidder to express a set of atomic bids such that only one of them can be selected by an auctioneer. *OR bids* allow to express that any subset of its atomic bids can be selected by an auctioneer. Other constructs enable the representation of quantity ranges, volume-based discounts, and so on.

Next, we cope with the first *WDP* requirements of CAs for SCF, represented by requirements (9–13) in table 5.1. With a suitable language for representing the bidders' offers at hand, we can provide an operational definition of the problem of selecting the winning bids while respecting the bidders' constraints. In other words, we have to provide a definition of the winner determination problem. With respect to the traditional combinatorial auction WDP a new dimension comes into play and must be considered: the production preconditions of supply chain operations. In fact, when supply chain operations are dealt, not only it is important what SCOs to select, but also their execution order. In fact, it must happen that at each step of the production process each SCO has available the resources it requires to be performed. Since a supply chain is a *chain* of SCOs, it may be the case that some SCOs provide the required inputs to other SCOs. Hence, the former ones must be performed before the latter ones. Then, an auctioneer must select a *sequence* of SCOs such that it:

(1) fulfils the constraints imposed by the bidders through the bidding language (e.g. if two atomic bids are in XOR, the auctioneer has to select at most one of them);

(2) is scheduled correctly, i.e. that each SCO has available the required input resources; and

(3) produces as output *at least* as many resources as required by the auctioneer (i.e. after performing the sequence of supply chain operations, the auctioneer ends up with the quantity of goods he initially required).

Only in the case that the solution fulfils the above conditions can be considered *valid* and implementable. Then, we will consider that a *valid* solution that maximises the auctioneer's revenue is a solution to the winner determination problem.

The provided definition of winner determination problem is not limited to any particular supply chain topology or SCO type. Then, by means of the definitions of bidding language and WDP we overcome requirements 1–13 in table 5.1. The remaining requirements (14–19) in table 5.1 are not considered in this chapter, and will be solved in the next chapter. Summarising, in this chapter we focus on *expressiveness* requirements, and formalise the *decision problem* faced by the auctioneer.

With a bidding language and an allocation rule (winner determination problem), the new auction type is completely defined. We shall call the resulting auction model *Mixed Multi-unit Combinatorial Auctions* (MMUCAs).

5.2 The problem

In this section we continue the example of *Grandma & co* introduced in section 1.4.2. Relying on such example, we specify the auctioneer's problem we aim at solving.

Example 5.1. *Grandma & co* is a company devoted to producing and selling apple pies. Traditionally, it was used to buying the basic ingredients to internally produce apple pies ready to sell. However, its revolutionary sourcing department is experimenting the most bizarre innovations. In example 1.1, we explained that *Grandma & co* decided to bring into the sourcing process producers of intermediate (*Dough*, *Filling*) and final goods (*Apple Pies*) across the supply chain. This led to the introduction of a new type of auction, MUCRAtR, as explained in chapter 4.

In this example, we show how the restless sourcing department decides to implement a newer sourcing process. Besides inviting to the sourcing event suppliers of all the goods across the supply chain, it also invites suppliers and requesters of *manufacturing services* such as, for instance, *Make Dough* or *Baking*. Then, *Grandma & co* runs a new type of combinatorial auction that involves:

- providers of goods (dough, filling, flour, and so on);
- requesters of goods (apple pies);
- providers of manufacturing operations (e.g. of *Make Dough*, *Make Filling*, or the *Baking* operations); and
- requesters of manufacturing operations.

All these potential supply chain partners are bidders in the auction.

The data regarding *Grandma & co* internal production costs is equal to the one defined in example 4.1 and is expressed in figure 5.1. We summarise it in the following:

(1) the *Make Dough* operation costs €5 each time it is carried out, it requires as inputs one unit of *butter*, three units of *sugar*, and two units of *flour*, and it produces two units of *dough* as output;

(2) the *Make Filling* operation costs €6 each time it is carried out, it requires as inputs one unit of *flour*, eight units of *apple*, and two units of *margarine*, and it produces two units of *filling* as output; and

(3) the *Baking* operation costs €14 each time it is carried out, it requires as inputs four units of *dough* and four units of *filling*, and it produces four units of *apple pie* as output.

Furthermore, the data about the initial stock and the final requirements are:

(1) a stock of a hundred units of *flour* and two hundred units of *sugar*;

(2) *Grandma & co* wants to end up with at least two hundred apple pies in its warehouse.

Say that *Grandma & co* receives the following bids (expressed in natural language) from all the invited bidders:

(1) *Bidder 1* offers 100 units of butter *AND* 200 units of margarine at €200. Bidders 1 to 4 express multi-unit bids that offer combinations of goods.

(2) *Bidder 2* offers 200 units of flours *AND* 300 units of sugar at €100.

(3) *Bidder 3* offers 800 units of apple pies at €200.

(4) *Bidder 4* offers 200 units of dough *AND* 200 units of *filling* at €1300.

(5) *Bidder 5* requests 200 units of apple pies for €2400. This bidder express a multi-unit request of goods.

(6) *Bidder 6* offers 100 units of butter at €150 *OR (non-exclusive)* offers 200 units of margarine at €100. This bidder proposes two alternative, not mutually exclusive, multi-unit offers. Notice that if the auctioneer accepts both bids, it must pay €250.

(7) *Bidder 7* offers 200 units of margarine at €200 *XOR (exclusive OR)* offers 200 units of butter at €200. This bidder proposes two alternative mutually, exclusive, multi-unit offers. The auctioneer can accept at most one of them.

(8) *Bidder 8* offers 200 units of filling at €1400 *XOR* requests 100 units of apple pies for €200.

(9) *Bidder 9* offers to perform the *Make Dough* operation 50 times at €200. This bidder, a contract manufacturer, offers to perform a SCO for the auctioneer exactly fifty times.

(10) *Bidder 10* requests to have the operation *Baking* performed 50 times, and he is willing to pay €210 for it. This bidder requests that an operation is performed for him exactly fifty times.

(11) *Bidder 11* offers to transform 2 units of dough and 2 units of filling into 1 unit of apple pie at €20 each time the operation is performed. He offers to perform the operation at most 50 times and at least 10 times. This bidder offers an operation that was not previously present in the auctioneer's internal supply chain (figure 5.1). Moreover, he expresses the operation can be performed a minimum and a maximum number of times.

(12) *Bidder 12* offers the *Baking* operation:

- at €10 each time it runs if the operation is performed between 10 and 30 times; and
- at €8 each time it runs if the operation is performed between 31 and 50 times.

This bidder issues an offer for an operation that includes a value-based discount.

(13) *Bidder 13* offers between 100 and 200 units of apples in bundles of 4 units at €2 per bundle. This bidder expresses quantity ranges.

(14) *Bidder 14* offers to transform 3 units of flour, 2 units of sugar, 1 unit of butter, 4 units of apples, and 2 units of margarine into 2 units of dough and 2 units of filling at €10 each time the operation is performed. The operation can be performed at least 10 and at most 40 times. Similar to the offer of bidder 11, with the difference that this operation has multiple output goods.

(15) *Bidder 15* offers to perform both the *Make Dough AND* the *Make Filling* operation at €20. This bidder issues an offer over a *bundle* of SCOs.

(16) *Bidder 16* offers to perform the *Make Dough* operation at €20 only if provided with an oven (it will give the oven back after performing the operation). This bidder expresses an offer in which there is a resource shared (the oven). In fact, it can be employed again afterwards.

□

The reader can understand that not only the requirements are difficult to express, but also the underlying decision problem is actually very complex. Which is the best option for *Grandma & co*? How to select the bids that maximise its revenue? The problem of *Grandma & co* is thus twofold: on the one hand to provide a bidding language for expressing the bidders' preferences, and on the other hand to find an allocation rule for assessing the revenue maximising sequence of SCOs that allows it to obtain at least two hundred apple pies at the end of the production process.

5.3 Bidding Language

In this section, we firstly define the notions of *supply chain operation* and *valuation* over *supply chain operations*, and subsequently we define a bidding language that can be used to transmit an agent's valuation (which may or may not be its true valuation) to the auctioneer. We also formally define the semantics of the language and introduce a number of additional language constructs that allow for the concise encoding of typical features of valuation functions. Finally, we discuss the expressive power of the language.

5.3.1 Supply Chain Operation

In what follows we provide a formal definition of supply chain operation.

Definition 5.1. Let G be the finite set of all the types of goods under consideration. A *Supply Chain Operation* (SCO) is a pair of multisets[2] over G: $(\mathcal{I}, \mathcal{O}) \in \mathbb{N}^G \times \mathbb{N}^G$.

□

An agent offering the SCO $(\mathcal{I}, \mathcal{O})$ declares that it can deliver $\mathcal{O}$ *after* having received $\mathcal{I}$. As we mentioned in section 5.1, in our setting bidders can offer any number of such SCOs, including several copies of the same SCO. That is, agents will be negotiating over *multisets of SCOs*, formally over elements of $\mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)}$.

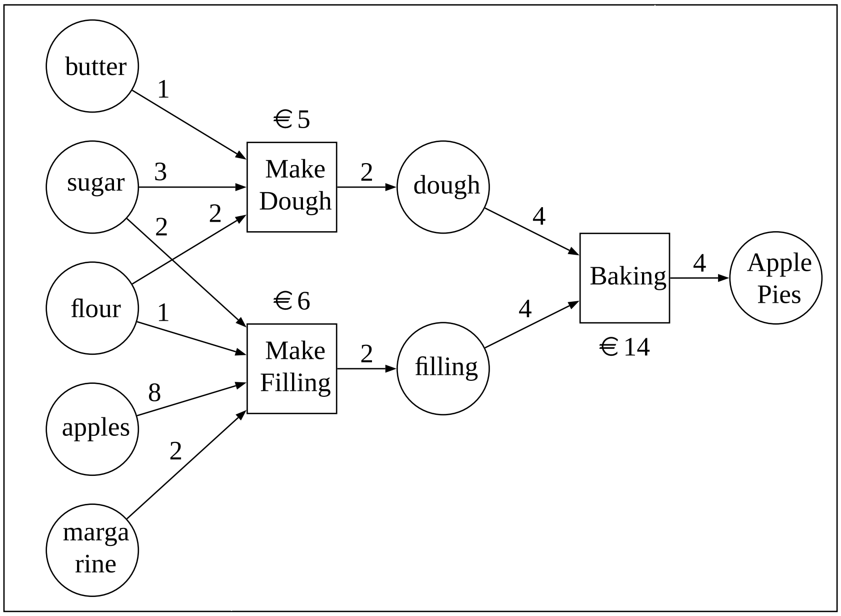


Figure 5.1: TNS associated to example 5.1.

Example 5.2. In figure 5.1 we graphically represent the internal manufacturing operations of *Grandma & co* employing the TNS introduced in in section 4.5.1. The *Make Dough* operation is represented as the following SCO:

$$\textit{Make Dough} = (1'butter + 3'sugar + 2'flour, 2'dough) \tag{5.1}$$

[2]Refer to section 2.2 for some background on multi-sets.

The *Make Filling* operation is represented as:

$$\textit{Make Filling} = (2'sugar + 1'flour + 8'apples + 2'margarine, 2'filling) \quad (5.2)$$

$\square$

Example 5.3. $\{(\emptyset, 1'a), (1'b, 1'c)\}$ means that the agent in question is able to deliver a (no input required) and that it is able to deliver c if provided with b. Note that this is not the same as $\{(1'b, 1'a + 1'c)\}$. In the former case, if another agent is able to produce b if provided with a, we can get c from nothing; in the latter case this would not be possible.

$\square$

Notice that the formalism employed for describing SCOs allows the representation of:

- *offers for bundles of goods*, expressed as SCOs with no inputs. That means that nothing is taken as input ($\mathcal{I} = \emptyset$), and $\mathcal{O}$ is provided as output. For instance, the offer of 200 units of butter *and* 100 units of margarine can be expressed as:

$$\{1'(\emptyset, 100'margarine + 200'butter)\}$$

- *requests of bundles of goods*, expressed as SCOs with no output. That means that $\mathcal{I}$ is taken as input, and nothing ($\mathcal{O} = \emptyset$) is provided as output. For instance, the request of 200 units of apple pie can be expressed as:

$$\{1'(200'applepie, \emptyset)\}$$

- *offers for bundles of SCOs*, expressed as:

$$\{\alpha'_1(\mathcal{I}_1, \mathcal{O}_1) + \alpha'_2(\mathcal{I}_2, \mathcal{O}_2) + \ldots + \alpha'_m(\mathcal{I}_m, \mathcal{O}_m)\}$$

  where $\alpha'_i \in \mathbb{N}$ represents the multiplicity of the SCO $(\mathcal{I}_i, \mathcal{O}_i)$. For instance, an offer to perform 10 times the *Make Dough* operation *and* 5 times the *Make Filling* operation can be expressed as:

$$\begin{aligned}&\{10'\textit{Make Dough} + 5'\textit{Make Filling}\} = \qquad (5.3)\\ &\{10'(1'butter + 3'sugar + 2'flour, 2'dough)+\\ &5'(2'sugar + 1'flour + 8'apples + 2'margarine, 2'filling)\}\end{aligned}$$

- *requests of bundles of SCOs*. In order to understand how to represent this type of request, we have to define what is meant by requiring a service. In fact, a bidder requiring the *Baking* service (see figure 5.1) provides the inputs to perform the *Baking* operation (dough and filling), and he is expected to receive the output of the required operation (apple pie).

Example 5.4. Consider the following multi-set of SCOs:

$$\{1'(\emptyset, 4'dough + 4'filling), 1'(4'\textit{apple pie}, \emptyset)\}$$

This means that a bidder provides the dough and the filling *and* he is expected to receive 4 apple pies. Notice that no precedence constraint between the two operations is specified. The bidder is happy receiving the apple pies both before and after giving away the dough and filling.

If a bidder expresses his willingness to pay €20 for having this multiset of SCOs allocated, this means that he is requiring the *Baking* operation for €20.

Notice that this is *not* the same as

$$\{1'(4'\textit{apple pie}, 4'dough + 4'filling)\}$$

In this case, the meaning would be that the bidder *requires* the apple pies as input *before* giving away the dough and filling. It is not what the bidder means.

□

In direct multi-unit combinatorial auctions, thoroughly explained in chapter 3, it is typical to assume *free-disposal* for bidders. Say that a bidder is willing to pay €10 for three units of dough. The free-disposal assumption says that the bidder is willing to pay *at least* €10 for four units of dough. This is a reasonable assumption, since the bidder receives more than he has required paying the same amount. Conversely, in a multi-unit combinatorial *reverse* auction the *free-disposal* assumption says that if a bidder is willing to be paid €10 for three units of dough, then it is willing to be paid *at most* €10 for two units of dough. This is reasonable as well since the bidder gives away less than offered and receives the same payment.

In the general case, the *free-disposal* assumption says that a bidder is willing to pay/be paid at least/at most the same amount if he is allocated a superset/subset of the required/offered goods.

In what follows, we generalise this idea to supply chain operations first, and then to multisets of supply chain operations. The idea of superset/subset is substituted with the idea of subsumption.

We define a *subsumption relation* $\sqsubseteq$ over supply chain operations as follows:

$$(\mathcal{I}, \mathcal{O}) \sqsubseteq (\mathcal{I}', \mathcal{O}') \Leftrightarrow \mathcal{I} \subseteq \mathcal{I}' \wedge \mathcal{O} \supseteq \mathcal{O}' \tag{5.4}$$

Intuitively, this means that the second supply chain operation is at least as good as the first (for the bidder), because he receives more and has to give away less.

Example 5.5. For instance, we have that:

$$(2'a + 2'b, 1'c) \sqsubseteq (3'a + 3'b, 1'c) \tag{5.5}$$

□

The following definition extends this subsumption relation to multisets of supply chain operations. It applies to multisets of the same cardinality, where for each SCO in the first set there exists a (distinct) SCO in the second set subsuming the former.

Definition 5.2 (Subsumption)**.** Let $\mathcal{D}, \mathcal{D}' \in \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)}$. We say that $\mathcal{D}$ is subsumed by $\mathcal{D}'$ ($\mathcal{D} \sqsubseteq \mathcal{D}'$) iff:

(i) $\mathcal{D}$ and $\mathcal{D}'$ have the same cardinality: $|\mathcal{D}| = |\mathcal{D}'|$.

(ii) There exists a surjective mapping $f : \mathcal{D} \to \mathcal{D}'$ such that, for all SCOs $t \in \mathcal{D}$, we have $t \sqsubseteq f(t)$.

□

Example 5.6. Employing a simplified notation for the innermost sets, we have[3]:

$$\{(a, bb), (cc, dd)\} \sqsubseteq \{(cc, d), (aaa, b)\} \tag{5.7}$$

Notice that the function f is such that the element (a, bb) maps to (aaa, b), and the element (cc, dd) maps to (cc, d). In fact, we have that $(a, bb) \sqsubseteq (aaa, b)$ and $(cc, dd) \sqsubseteq (cc, d)$.

□

Property (i) of definition 5.2 is needed, because giving less supply chain operations in some cases may diminish the valuation of a bidder. This is clarified by the following example.

Example 5.7. Consider that a bidder is willing to pay €10 for receiving two units of b and two units of c, namely for the SCO $\{(bb, \emptyset), (cc, \emptyset)\}$. Most probably, the bidder *is not* willing to pay at least the same quantity for having the multiset $\{(bb, \emptyset)\}$, since he is receiving less goods! Alternatively, consider the case in which a bidder is willing to be paid €10 for providing two units of b and two units of c, namely $\{(\emptyset, bb), (\emptyset, cc)\}$. Most probably, the bidder in this case *is* willing to pay at least the same quantity for being allocated the multiset $\{(\emptyset, bb)\}$, since he is giving away less goods!

□

Then, as shown in example 5.7, there is not a general rule stating that less supply chain operations allocated is considered a better outcome for a bidder.

5.3.2 Valuations

Our goal is having agents negotiating over bundles of SCOs. Then, we have to introduce a formalism that allows an agent to express preferences over bundles of SCOs. Hence, in what follows we provide a definition of valuation.

Definition 5.3. A *valuation* $v : \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)} \to \mathbb{R}$ is a (typically partial) mapping from multisets of SCOs to the real numbers.

[3] This is equivalent to

$$\{(1'a, 2'b), (2'c, 2'd)\} \sqsubseteq \{(2'c, 1'd), (3'a, 1'b)\} \tag{5.6}$$

□

Intuitively, $v(\mathcal{D}) = p$ means that the agent equipped with valuation v is willing to make a payment of p in return for being allocated all the SCOs in $\mathcal{D}$ (in case p is a negative number, this means that the agent will accept the deal if it *receives* an amount of $|p|$).

Example 5.8 (Valuations)**.**

- $v(1'(1'oven + 1'dough, 1'oven + 1'cake)\}) = -20$ means that a bidder can produce a cake for € 20 if given an oven and some dough, and that it will return the oven again afterwards.
- $v(\{1'(1'butter + 3'sugar + 2'flour, 2'dough)\}) = -4$ means that a bidder is able to perform the *Make Dough* operation for € 4.

□

We write $v(\mathcal{D}) = \perp$ to express that v is *undefined* over the multiset $\mathcal{D}$. Again intuitively, this means the agent would be unable to accept the corresponding deal. Valuation functions can often be assumed to be both *normalised* and *monotonic:*

Definition 5.4 (Normalised valuation)**.** A valuation v is normalised iff $v(\mathcal{D}) = 0$ whenever $\mathcal{I} = \mathcal{O}$ for all $(\mathcal{I}, \mathcal{O}) \in \mathcal{D}$.

□

That is, a valuation is normalised iff exchanging a multiset of goods for an identical multiset does not incur any costs (this includes the special case of $\mathcal{I} = \mathcal{O} = \emptyset$, *i.e.* the case of not exchanging anything at all). The next definitions refer to our subsumption relation $\sqsubseteq$ (see Definition 5.2).

Definition 5.5 (Monotonic valuation)**.** A valuation v is monotonic iff $v(\mathcal{D}) \leq v(\mathcal{D}')$ whenever $\mathcal{D} \sqsubseteq \mathcal{D}'$.

□

That is, an agent with a monotonic valuation does not mind taking on more goods and giving fewer away. This assumption is the generalisation of the *free-disposal* assumption we mentioned above when supply chain operations are traded.

Any given valuation function can be *turned into* a monotonic valuation by taking its monotonic closure[4]:

Definition 5.6 (Monotonic closure)**.** The monotonic closure $\hat{v}$ of a valuation v is defined as $\hat{v}(\mathcal{D}) = \max\{v(\mathcal{D}') \mid \mathcal{D}' \sqsubseteq \mathcal{D}\}$.

□

As we are working with multisets of goods, observe that there could be infinitely many bundles an agent may want to assign a (defined) value to. As we shall see in Section 5.3.6, our bidding languages can only express valuations that are *finitely-peaked* (or that are the monotonic closure of a finitely-peaked valuation):

[4]Here and throughout this chapter, we assume that any occurrences of $\perp$ are being removed from a set before computing its maximum element, and that the maximum of the empty set is $\perp$.

Definition 5.7 (Finitely-peaked val.)**.** A valuation v is finitely-peaked iff v is only defined over finite multisets of pairs of finite multisets and $\{\mathcal{D} \in \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)} \mid v(\mathcal{D}) \neq \bot\}$ is finite.

□

5.3.3 Atomic Bids

An *atomic bid* $\text{BID}(\{\alpha'_1(\mathcal{I}_1, \mathcal{O}_1) + \ldots + \alpha'_n(\mathcal{I}_n, \mathcal{O}_n)\}, p)$, where $\alpha'_i \in \mathbb{N}$ represents the multiplicity of the SCO $(\mathcal{I}_i, \mathcal{O}_i)$, specifies a finite multiset of finite SCOs and a price. To make the semantics of such an atomic bid precise, we need to decide whether or not we want to make a *free disposal* assumption. We can distinguish two types of free disposal:

- Free disposal *at the bidder's side* means that a bidder would always be prepared to accept more goods and give fewer goods away, without requiring a change in payment. This affects the definition of the valuation functions used by bidders.
- Free disposal *at the auctioneer's side* means that the auctioneer can freely dispose of additional goods, *i.e.* accept more and give away fewer of them. This affects the definition of what constitutes a valid solution to the winner determination problem (see Section 5.4).

Under the assumption of free disposal at the bidder's side, the bid $Bid = \text{BID}(\mathcal{D}, p)$ defines the following valuation:

$$v_{Bid}(\mathcal{D}') = \begin{cases} p & \text{if } \mathcal{D} \sqsubseteq \mathcal{D}' \\ \bot & \text{otherwise} \end{cases}$$

To obtain the valuation function defined by the same bid without the free disposal assumption, simply replace $\sqsubseteq$ in the above definition by equality.

5.3.4 Combinations of Bids

A suitable *bidding language* should allow a bidder to encode choices between alternative bids and the like. To this end, several operators for combining bids have been introduced in the literature (Nisan, 2006), which we are going to adapt to our purposes here. Informally, an OR-combination of several bids signifies that the bidder would be happy to accept that any combination of the sub-bids specified is selected by the auctioneer, if he gets paid/pays the sum of the associated prices. An XOR-combination of bids expresses that the bidder is prepared to accept that at most one of them is selected[5].

We also suggest the use of an IMPLIES operator to express that accepting one bid forces the auctioneer to also take the second. We shall take an AND-combination to mean that the bidder will only accept if the respective sub-bids are selected together.

As it turns out, while all these operators are very useful for specifying typical valuations in a concise manner, any complex bid can alternatively be represented by an

[5] As Nisan (Nisan, 2006) put it, "purists may object" to the name XOR, as this is not the same as the exclusive-or operator familiar from propositional logic (ONE-OF may be a better name).

XOR-combination of atomic bids. To simplify presentation, rather than specifying the exact semantics of all of our operators directly, we are simply going to show how any bid can be translated into such a *normal form*. Firstly, any occurrences of IMPLIES and OR can be eliminated by applying the following rewrite rules:

$$\begin{aligned} X \text{ IMPLIES } Y &\rightsquigarrow (X \text{ AND } Y) \text{ XOR } Y \\ X \text{ OR } Y &\rightsquigarrow X \text{ XOR } Y \text{ XOR } (X \text{ AND } Y) \end{aligned}$$

Note that for single-unit auctions, OR cannot be translated into XOR like this (if X and Y overlap, then they cannot be accepted together; in an MMUCA this depends on the supply of the auctioneer). Next we show how to distribute AND over XOR, so as to push AND-operators to the inside of a formula:

$$(X \text{ XOR } Y) \text{ AND } Z \rightsquigarrow (X \text{ AND } Z) \text{ XOR } (Y \text{ AND } Z)$$

Finally, we need to define how to turn an AND-combination of atomic bids into a single atomic bid:

$$\text{BID}(\mathcal{D}, p) \text{ AND } \text{BID}(\mathcal{D}', p') \rightsquigarrow \text{BID}(\mathcal{D} \uplus \mathcal{D}', p + p')$$

Recall from section 2.2.1 that the $\uplus$ symbol is a *sum of multiset*, meaning that the multiplicity of the sum multiset for an element is the sum of of the multiplicities of the addend multisets.

Observe that these rewrite rules together allow us to translate any expression of the bidding language into an equivalent XOR-combination of atomic bids. We also call this the ***XOR-language***. To formally define the semantics of this language, it suffices to define the semantics of the XOR-operator. Suppose we are given n bids Bid_i, with $i \in \{1..n\}$. Let $Bid = Bid_1 \text{ XOR } \cdots \text{ XOR } Bid_n$. This bid defines the following valuation:

$$v_{Bid}(\mathcal{D}) = \max\{v_{Bid_i}(\mathcal{D}) \mid i \in [1, n]\}$$

That is, XOR simply selects the atomic bid corresponding to the valuation giving maximum profit for the auctioneer.

5.3.5 Representing Quantity Ranges

As we prove in the next section, the XOR-language is expressive enough to describe any (finitely-peaked) valuation. Nevertheless, it may not be possible to express a given valuation in a succinct manner. From a practical point of view, it is therefore useful to introduce additional constructs that allow us to express typical features more succinctly. Here we consider the case of quantity ranges. We want to be able to express that a certain number of copies of the same SCO are acceptable to a bidder.

Let $n \in \mathbb{N}$. To express that up to n copies of the same Bid are acceptable, we use the following notation:

$$Bid^{\leq n} = \underbrace{(Bid \text{ OR } \cdots \text{ OR } Bid)}_{n \text{ times}}$$

This allows us to express *bundling constraints* in a concise manner: the bid $(\{a, a, a, b\}, -10)^{\leq 50}$ expresses that we can sell up to 50 packages containing three items of type a and one item of type b each, for 10 € a package (for simplicity, we omit $\mathcal{O}$ here). We also use the following shorthand:

$$Bid^n \quad = \quad \underbrace{(Bid \text{ AND } \cdots \text{ AND } Bid)}_{n \text{ times}}$$

Now we can express quantity ranges. Let $n_1, n_2 \in \mathbb{N}$ with $0 < n_1 < n_2$. The following expression says that we may accept between n_1 and n_2 copies of the same Bid:

$$Bid^{[n_1,n_2]} \quad = \quad Bid^{\leq(n_2-n_1)} \text{ IMPLIES } Bid^{n_1}$$

These constructs also allow us to express important concepts such as quantity discounts in a concise manner. For instance, the bid

$$[(a, 20)^{\leq 100} \text{ IMPLIES } (a, 25)^{50}] \text{ XOR } (a, 25)^{\leq 50}$$

says that we are prepared to buy up to 50 items of type a for 25 € each, and then up to 100 more for 20 € each.

5.3.6 Expressive Power

Next we are going to settle the precise expressive power of the XOR-language, and thereby of the full bidding language. We have to distinguish two cases, as we have defined the semantics of the language both with and without free disposal.

Proposition 5.1. *The XOR-language without free disposal can represent all finitely-peaked valuations, and only those.*

Proof. Let v be any finitely-peaked valuation. To express v in the XOR-language, we first compose one atomic bid for each $\mathcal{D} = \{\alpha'_1(\mathcal{I}_1, \mathcal{O}_1) + \ldots + \alpha'_n(\mathcal{I}_n, \mathcal{O}_n)\}$ with $v(\mathcal{D}) = p \neq \bot$:

$$\text{BID}(\{\alpha'_1(\mathcal{I}_1, \mathcal{O}_1) + \ldots + \alpha'_n(\mathcal{I}_n, \mathcal{O}_n)\}, p)$$

Joining all these bids together in one large XOR-combination yields a bid that expresses v. Vice versa, it is clear that the XOR-language cannot express any valuation that is not finitely-peaked. □

Proposition 5.2. *The XOR-language with free disposal can represent all valuations that are the monotonic closure of a finitely-peaked valuation, and only those.*

Proof. The construction of a bid representing any given valuation works in analogy to the proof of Proposition 5.1. Note that for the semantics with free disposal we precisely obtain the monotonic closure of the valuation we would get if we were to drop the free disposal assumption. □

These results correspond to the expressive power results for the standard XOR-language for direct single-unit combinatorial auctions. With free disposal (the standard assumption), the XOR-language can express all monotonic valuations (Nisan, 2006); and without that assumption it can represent the complete range of valuations (note that *any* valuation is finitely-peaked if we move from multisets to sets). Notice that this result on the expressiveness shows that the provided bidding language overcomes successfully requirements (1–8) of table 5.1.

Given those expressiveness results, in the remaining of the dissertation we assume that bidders express their preferences by means of the XOR language.

5.3.7 Examples of Bids

In this section we provide some examples of bids in order to highlight the better expressiveness offered by our bidding language. For this reason, we encode the bids presented in example 5.1.

(1) *Bidder 1* offers 100 units of butter *AND* 200 units of margarine at €200:

$$\textsc{bid}(1'(\emptyset, 100'butter + 200'margarine), -200)$$

(2) *Bidder 2* offers 200 units of flours *AND* 300 units of sugar at €100:

$$\textsc{bid}(1'(\emptyset, 200'flour + 300'sugar), -100)$$

(3) *Bidder 3* offers 800 units of apple pies at €200:

$$\textsc{bid}(1'(\emptyset, 800'\textit{apple}), -200)$$

(4) *Bidder 4* offers 200 units of dough *AND* 200 units of *filling* at €1300:

$$\textsc{bid}(1'(\emptyset, 200'dough + 200'filling), -1300)$$

(5) *Bidder 5* requests 200 units of apple pies for €2400:

$$\textsc{bid}(1'(800'\textit{apple pie}, \emptyset), 2400)$$

(6) *Bidder 6* offers 100 units of butter at €150 *OR (non-exclusive)* offers 200 units of margarine at €100:

$$\textsc{bid}(1'(\emptyset, 100'\textit{butter}), -150)\ \textit{OR}\ \textsc{bid}(1'(\emptyset, 200'\textit{margarine}), -100)$$

(7) *Bidder 7* offers 200 units of margarine at €200 *XOR (exclusive OR)* offers 200 units of butter at €200:

$$\textsc{bid}(1'(\emptyset, 200'\textit{margarine}), -200)\ \textit{XOR}\ \textsc{bid}(1'(\emptyset, 200'\textit{butter}), -200)$$

(8) *Bidder 8* offers 200 units of filling at € 1400 *XOR* requests 100 units of apple pies for € 200:

$$\textsc{bid}(1'(\emptyset, 200'\textit{filling}), -1400)\ \textit{XOR}\ \textsc{bid}(1'(100'\textit{apple pie}, \emptyset), 200)$$

(9) *Bidder 9* offers to perform the *Make Dough* operation 50 times at € 200:

$$\textsc{bid}(50'(1'butter + 3'sugar + 2'flour, 2'dough), -200)$$

(10) *Bidder 10* requests the operation *Baking* performed 50 times, and he is willing to pay € 210 for it:

$$\textsc{bid}(1'(\emptyset, 4'dough + 4'filling) + 1'(4'\textit{apple pie}, \emptyset), 4.2)^{50}$$

(since $4.2 * 50 = 210$)

(11) *Bidder 11* offers to transform 2 units of dough and 2 units of filling into 1 unit of apple pie for € 20 each time the operation is performed (whenever the operation is performed at least 10 times and at most 50 times):

$$\textsc{bid}(1'(2'dough + 2'filling, 1'\textit{apple pie}), 20)^{[10,50]}$$

(12) *Bidder 12* offers the *Baking* operation at:

$$\textsc{bid}(1'(4'dough + 4'filling, 4'\textit{apple pie}), 10)^{[10,30]}$$
$$\textit{XOR}$$
$$\textsc{bid}(1'(4'dough + 4'filling, 4'\textit{apple pie}), 8)^{[31,50]}$$

(13) *Bidder 13* offers between 100 and 200 units of apples in bundles of 4 units at € 2 per bundle:

$$\textsc{bid}(1'(\emptyset, 4'\textit{apple pie}), -2)^{[25,50]}$$

(14) *Bidder 14* offers to transform 3 units of flour, 2 units of sugar, 1 unit of butter, 4 units of apples, 2 units of margarine into 2 units of dough and 2 units of filling at € 10 each time the operation is performed. The operation can be performed at least 10 and at most 40 times:

$$\textsc{bid}(1'(1'butter + 2'sugar + 3'flour + 4'apple + 2'margarine,\\ 2'dough + 2'filling), -10)^{[10,40]}$$

(15) *Bidder 15* offers to perform both the *Make Dough AND* the *Make Filling* operation at € 20:

$$\textsc{bid}(1'(1'butter + 3'sugar + 3'flour, 2'dough)+\\ 1'(8'apple + 2'margarine + 1'flour, 2'filling), -20)$$

Notice carefully that the offer of bidder 15 cannot be rewritten as

$$\textsc{bid}(1'(1'butter + 3'sugar + 3'flour + 8'apple + 2'margarine + 1'flour,\\ 2'dough + 2'filling), -20)$$

The two bids do not represent the same thing. In the former case, the two operations could be performed at different steps in the production process. In the latter case, it is a one shot operation that needs *at the same time* all the input resources

$$1'butter + 3'sugar + 3'flour + 8'apple + 2'margarine + 1'flour$$

available.

(16) *Bidder 16* offers to perform the *Make Dough* operation for €20 only if provided with an oven (it will give the oven back after performing the operation):

$$\textsc{bid}(1'(4'dough + 4'filling + 1'oven, 4'\textit{apple pie} + 1'oven), 10)$$

In what follows we provide an example showing that the introduced bidding language can be employed not only to express bidders' preferences, but also to encode information about a particular market. As an example, we consider how to incorporate into the auction information about the expected sales in function of the sale price. Consider the following example.

Example 5.9. Here we extend example 5.1 taking into account that more information about the apple pie market becomes available to *Grandma & co*. Such information is the sale forecast in function of the sale price:

- two hundreds apple pies if the sale price is €12 each, for a total of €2400;
- a hundred and thirty apple pies if the selling price is set to €13, for a total of €1690;

This information can be easily included in the auction by means of *bids from the markets*:

$$\textsc{bid}(1'(\emptyset, 200'\textit{apple pies}), 2400)\ \textit{XOR}\ \textsc{bid}(1'(\emptyset, 130'\textit{apple pies}), 1690) \tag{5.8}$$

□

5.4 Winner Determination

In this section, we define the winner determination problem (WDP) for MMUCAs. We first give an informal outline of the problem, and then a formal definition. We also briefly comment on mechanism design issues.

5.4.1 Informal Definition

The *input* to the WDP consists of a complex bid expression for each bidder, a multiset $\mathcal{U}_{in}$ of goods the auctioneer holds to begin with, and a multiset $\mathcal{U}_{out}$ of goods the auctioneer expects to end up with.

In standard combinatorial auctions, a solution to the WDP is a set of atomic bids to accept. In our setting, however, the *order* in which the auctioneer "uses" the accepted SCOs matters. For instance, if the auctioneer holds a to begin with, then checking whether accepting the two bids $Bid_1 = (\{1'(1'a, 1'b)\}, 10)$ and $Bid_2 = (\{(1'b, 1'c)\}, 20)$ is feasible involves realising that we have to use the SCO contained in Bid_1 before the one contained in Bid_2. Thus, a *solution* to the WDP will be a *sequence of SCOs*. A *valid* solution has to meet two conditions:

(1) *Bidder constraints:* The multiset of SCOs in the sequence has to *respect the bids* submitted by the bidders. This is a standard requirement. For instance, if a bidder submits an XOR-combination of SCOs, at most one of them may be accepted.

(2) *Auctioneer constraints:* The sequence of SCOs has to be *implementable:*

  (a) check that $\mathcal{U}_{in}$ is a superset of the input set of the first SCO (there are enough goods available to perform the first SCO);

  (b) then update the set of goods held by the auctioneer after each SCO and check that it is a superset of the input set of the next SCO (at each step there are enough goods available to perform the remaining SCOs);

  (c) finally check that the set of items held by the auctioneer in the end is a superset of $\mathcal{U}_{out}$ (i.e. the auctioneer ends up with the resources initially required).

Requirement 2 is specific to MMUCAs. An *optimal* solution is a valid solution that maximises the sum of prices associated with the atomic bids selected.

5.4.2 Formal Definition

For the formal definition of the WDP, we restrict ourselves to bids in the XOR-language, which we have showed to be fully expressive (over finitely-peaked valuations) in proposition 5.1. For each bidder i, let Bid_{ij} be the jth atomic bid occurring within the XOR-bid submitted by i.

Recall that each atomic bid consists of a multiset of SCOs and a price: $Bid_{ij} = (\mathcal{D}_{ij}, p_{ij})$, where $\mathcal{D}_{ij} \in \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)}$ is a multiset of SCOs and $p_{ij} \in \mathbb{R}$ is the associated cost/price. We will employ the following notation:

- For each Bid_{ij}, let t_{ijk} be the kth SCO in $\mathcal{D}_{ij}$.
- Let $\mathcal{D}_{ij}(t_{ijk})$ be the multiplicity of t_{ijk} in $\mathcal{D}_{ij}$.
- Let $\mathcal{D} = \biguplus_{ij} \mathcal{D}_{ij}$ be the multiset of the overall SCOs received with their multiplicity.

- Let δ be the overall number of SCOs mentioned anywhere in the bids, i.e.

$$\delta = \sum_{ij} |\mathcal{D}_{ij}| = \sum_{ijk} \mathcal{D}_{ij}(t_{ijk})$$

- Let $T = \{t_{ijk} : \forall ijk\}$ be the set of the overall SCOs in the bids disregarding their multiplicity.
- Let G be the set of negotiated goods.
- $\mathcal{U}_{in} \in \mathbb{N}^G$ is a multiset of goods standing for the initial stock of the auctioneer.
- $\mathcal{U}_{out} \in \mathbb{N}^G$ is a multiset of goods standing for the number of goods the auctioneer desires to end up with.
- $\mathcal{M}^m \in \mathbb{N}^G$ is a multiset of goods standing for the number of goods available to the auctioneer after applying m supply chain operations in a production process.

The auctioneer has to decide which SCOs to accept and in which order to implement them. Thus, we define an *allocation sequence* as

Definition 5.8 (Allocation Sequence)**.** An *allocation sequence* Σ is a sequence of SCOs:

$$\Sigma : \{1, 2, \ldots, \ell\} \to T$$

where $\ell \in \mathbb{N}$ is the length of the sequence.

□

We will say that a SCO t_{ijk} is contained in the allocation sequence to say that the kth SCO in the jth atomic bid of bidder i belongs to the allocation sequence. More formally, with an abuse of notation, we will write

$$t_{ijk} \in \Sigma \iff \exists m \in \{1, \ldots, \ell\} \text{ s.t. } \Sigma(m) = t_{ijk} \tag{5.9}$$

Furthermore, let $(\mathcal{I}_{\Sigma(m)}, \mathcal{O}_{\Sigma(m)})$ be the input and output multisets of the transition holding the m-th position of Σ; and let $|\Sigma^{-1}(t_{ijk})|$ be the number of times t_{ijk} occurs within the sequence Σ.

Given an allocation sequence Σ we can obtain the set of goods held by the auctioneer after each SCO. We illustrate this fact by means of the following example.

Example 5.10. Say that an auctioneer begins with $\mathcal{U}_{in} = \{2'a + 2'd\}$. If we apply the first SCO in a sequence $(\mathcal{I}_{\Sigma(1)}, \mathcal{O}_{\Sigma(1)}) = (2'a, 1'c)$ (from two units of a produce one unit of c), the auctioneer ends up with $\mathcal{M}^1 = \{1'c + 2'd\}$. Formally, we can express this operation as an equation over multisets:

$$\mathcal{M}^1(g) = \mathcal{U}_{in}(g) + \mathcal{O}_{\Sigma(1)}(g) - \mathcal{I}_{\Sigma(1)}(g)$$

The application of the SCO above is only possible because two units of good a are available. This condition maps to:

$$\mathcal{U}_{in}(g) \geq \mathcal{I}_{\Sigma(1)}(g)$$

□

Let $\mathcal{M}^m \in \mathbb{N}^G$ be the goods held by the auctioneer after applying the mth SCO in an allocation sequence Σ. We can generalise the two equations above as follows (let $\mathcal{M}^0 = \mathcal{U}_{in}$):

$$\mathcal{M}^m(g) = \mathcal{M}^{m-1}(g) + \mathcal{O}_{\Sigma(m)}(g) - \mathcal{I}_{\Sigma(m)}(g) \tag{5.10}$$

$$\mathcal{M}^{m-1}(g) \geq \mathcal{I}_{\Sigma(m)}(g) \tag{5.11}$$

Notice that the length of the solution sequence $\ell = |\Sigma|$ will be at most equal to the overall number of atomic transformations submitted, i.e. $\ell \leq \delta$.

Equation 5.11 can be written in a more synthetic form by embedding into one formula its recursive structure:

$$\mathcal{U}_{in}(g) + \sum_{l=1}^{m-1} \big(\mathcal{O}_{\Sigma(l)}(g) - \mathcal{I}_{\Sigma(l)}(g)\big) \geq \mathcal{I}_{\Sigma(m)}(g) \tag{5.12}$$

since

$$\mathcal{M}^m(g) = \mathcal{U}_{in}(g) + \sum_{l=1}^{m} \big(\mathcal{O}_{\Sigma(l)}(g) - \mathcal{I}_{\Sigma(l)}(g)\big) \tag{5.13}$$

Notice that an allocation sequence will not necessarily be a valid solution to the MMUCA WDP. We are now ready to define under what circumstances a sequence of SCOs constitutes a valid solution:

Definition 5.9 (Valid Solution Sequence)**.** Given a multiset $\mathcal{U}_{in}$ of available goods and a multiset $\mathcal{U}_{out}$ of required goods, an allocation sequence Σ for a given set of XOR bids over SCOs t_{ijk} is said to be a *valid solution sequence* iff:

(1) Σ either contains all or none of the SCOs belonging to the same atomic bid. That is, the semantics of the BID operator is fulfilled:

$$\exists k : t_{ijk} \in \Sigma \;\Rightarrow\; \forall k \; |\Sigma^{-1}(t_{ijk})| = \mathcal{D}_{ij}(t_{ijk})$$

Intuitively, this means that if Σ contains a SCOs t_{ijk} of bid $Bid_{ij} = (\mathcal{D}_{ij}, p_{ij})$, then it must contain all the $t_{ijk'} \in \mathcal{D}_{ij}$ with the corresponding multiplicity.

(2) Σ does not contain two SCOs belonging to different atomic bids by the same bidder. That is, the semantics of the XOR operator is fulfilled:

$$t_{ijk}, t_{ij'k'} \in \Sigma \;\Rightarrow\; j = j'$$

(3) Equation (5.11) holds at each step of the solution sequence Σ:

$$\mathcal{M}^{m-1}(g) \geq \mathcal{I}_{\Sigma(m)}(g) \qquad \forall m \in [1, \ell], \forall g \in G \tag{5.14}$$

that is equivalent to equation (5.12):

$$\mathcal{U}_{in}(g) + \sum_{l=1}^{m-1} \big(\mathcal{O}_{\Sigma(l)}(g) - \mathcal{I}_{\Sigma(l)}(g)\big) \geq \mathcal{I}_{\Sigma(m)}(g) \tag{5.15}$$

$$\forall m \in [1, \ell], \forall g \in G$$

This condition ensures that all SCOs have enough input goods available at each step of the SCO sequence.

(4) The set of goods held by the auctioneer after implementing the SCO sequence is a superset of the goods the auctioneer is expected to end up with:

$$\mathcal{M}^{\ell}(g) \geq \mathcal{U}_{out}(g) \qquad \forall g \in G \tag{5.16}$$

that is equivalent to:

$$\mathcal{U}_{in}(g) + \sum_{m=1}^{\ell} \left(\mathcal{O}_{\Sigma(m)}(g) - \mathcal{I}_{\Sigma(m)}(g)\right) \geq \mathcal{U}_{out}(g) \qquad \forall g \in G$$

□

The *revenue* for the auctioneer associated with a valid *solution sequence* Σ is the sum of the prices of the bids associated to the supply chain operations in the solution sequence. Then, according to item (1) of definition 5.9, a bid $Bid_{ij} = (\mathcal{D}_{ij}, p_{ij})$ is in the winning set if all the transitions in $\mathcal{D}_{ij}$ are in the winning set. Then, it is easy to see that the set of winning bids can be expressed as $B^* = \{Bid_{ij} \in B | \exists k \text{ s.t. } t_{ijk} \in \Sigma\}$. Then, the revenue of the auctioneer is computed as:

$$\sum_{Bid_{ij} \in B^*} p_{ij} \tag{5.17}$$

Definition 5.10 (WDP)**.** Given a set of XOR bids and multisets $\mathcal{U}_{in}$ and $\mathcal{U}_{out}$ of initial and final goods, respectively, the winner determination problem is the problem of finding a valid solution sequence Σ that maximises the revenue for the auctioneer.

□

Before going on, a comment on the definition of allocation sequence is in place. In the definition given above, we make the hypothesis that only one SCO is performed at each step of the solution sequence. However, it is clear that the nature of our problem admits an eventual concurrency of SCOs. For instance, it may be the case that two SCOs can be performed in parallel, i.e. at the very same step. For this reason we notice that it is possible to extend the definition of allocation sequence to capture concurrency. We leave out such generalisation as a matter of future work.

With this allocation rule at hand, plus the bidding language introduced in section 5.3, the *Mixed Multi-unit Combinatorial Auction* model is completely defined.

5.4.3 Mechanism Design

An important issue in auction design concerns their *game-theoretical* properties. We note here that the central result in *mechanism design*, on the incentive-compatibility of the Vickrey-Clarke-Groves (VCG) mechanism (Ausubel and Milgrom, 2006b), carries over from standard combinatorial auctions to MMUCAs. Recall that the VCG mechanism allocates goods in the most efficient manner and then determines the price to be paid by each bidder by subtracting from their offer the difference of the overall value of the winning bids and the overall value that would have been attainable without that bidder taking part. That is, this "discount" reflects the contribution to the overall production of value of the bidder in question. The VCG mechanism is strategy-proof: submitting their true valuation is a (weakly) dominant strategy for each bidder. As an inspection of standard proofs of this result reveals (Mas-Colell et al., 1995), this does not depend on the internal structure of the agreements that agents make. Hence, it also applies to MMUCAs.

However, notice that our focus is centred on an efficient allocation rule, and we do not argue about mechanism design issues.

5.5 Subsumed Auction Models

Our model of mixed multi-unit combinatorial auctions subsumes a range of combinatorial auction models discussed in the combinatorial auctions literature (see section 3.2.1), namely:

- *Single-unit* direct and reverse auctions;
- *Multi-unit* direct and reverse auctions, where there may be several indistinguishable copies of the same good available in the system;
- *Multi-unit* direct and reverse combinatorial auctions;
- *Double auctions*, or *combinatorial exchanges*, where the auctioneer will be able to both sell and buy goods within a single auction. We should stress that there are important differences between our mixed auctions and models known as *double auction* (Wurman et al., 1998) or *Combinatorial exchanges* (Sandholm et al., 2002). The most important difference is that these models do not incorporate the concept of a *sequence* of exchanges, which is required if the intention is to model some sort of production process. In the formulation of the WDP for combinatorial exchanges given by Sandholm *et al.* (Sandholm et al., 2002), for instance, accepting "circular" bids such as $\text{BID}(\{(1'a, 1'b)\}, 10)$ and $\text{BID}(\{(1'b, 1'a+1'c)\}, 10)$, to obtain c for 20 €, would be considered a solution sequence. With our semantics in mind, however, this allocation sequence is *not* valid: the first agent needs to receive a before it can produce b, but the second agent needs to receive b before it can produce a and c. Hence, no deal should be possible. In fact, the MMUCA can be used to simulate combinatorial exchanges (and double auctions). For instance, the bid $\text{BID}(1'(1'a, 1'b), 10)$ can be rewritten as $\text{BID}(1'(1'a, \emptyset) + 1'(\emptyset, 1'b), 10)$ to express that a bidder will only deliver b if it receives a, but that the order does not matter. Of course, if no true SCOs (imposing an order) are used, then the simpler model of combinatorial exchanges is to be preferred.
- *Multi-Unit Combinatorial Reverse Auctions with transformability Relationships among Goods*. A MUCRAtR, as proposed in chapter 4, can be modelled by allowing the auctioneer to submit bids representing its internal SCOs along with

their costs. In fact it increments its expressiveness as well, allowing to represent XOR combinations of multisets of internal SCOs.

- *Combinatorial Auctions for supply chain formation*, introduced in (Walsh and Wellman, 2003). Walsh and Wellman (Walsh and Wellman, 2003) tackle a similar problem to ours, focusing on supply chain formation. Although their contribution is very significant, we find limitations along three dimensions. Firstly, they do not allow a provider to submit bids on bundles of SCOs. Secondly, they do not define a bidding language (in fact, their agents submit a bid with a single SCO each). Finally, the SCO net that defines the supply chain has to fulfil strict criteria: acyclicity, SCOs can only produce one output good, etc.

Our bidding language as well can be viewed as a generalisation of the state-of-the-art bidding languages for combinatorial auctions (Nisan, 2006). In fact, it can be easily applied as well to all the above mentioned auctions[6]:

- in *single-unit* direct auctions, we only have atomic bids of the type

$$\text{BID}(\{(\mathcal{I}, \emptyset)\}, p)$$

  where $\mathcal{I}$ is a set such that $|\mathcal{I}| = 1$.

- in *multi-unit* direct auctions $|\mathcal{I}| = \mathcal{I}(g) \leq n$, where n is the number of units of good g at auction (there is a single good at auction).

- in *multi-unit* direct combinatorial auctions $\mathcal{I}(g) \leq n_g$ where n_g is the number of units of good g at auction (there are multiple goods at auction).

- in *combinatorial exchanges* we have bids of the type:

$$\text{BID}(\{(\mathcal{I}, \emptyset)\}, p)$$

  and

$$\text{BID}(\{(\emptyset, \mathcal{O})\}, p)$$

- in *MUCRAtR* bidders can send bids of the type

$$\text{BID}(\{(\emptyset, \mathcal{O})\}, p)$$

  and the auctioneer itself can send bids in the form:

$$\text{BID}(\{(\mathcal{I}, \mathcal{O})\}, p)^{\leq \gamma}$$

  where γ is an upper bound on the maximum number of times an operation can be performed by the auctioneer. We clarify this point by means of the following example:

[6]We provide here the direct cases. The reverse cases are easily obtained with small changes.

Example 5.11 (MMUCA as MUCRAtR)**.** This example aims at representing, by means of the bidding language introduced in section 5.3, the auction described in example 4.5. The bids, sent by the very same auctioneer, and representing its internal production structure, are (from figure 1.1):

$$\textsc{bid}(1'(1'butter + 3'sugar + 2'flour, 2'dough), -5)^{\leq\gamma_1}\ \textit{OR}$$
$$\textsc{bid}(1'(2'margar. + 2'sugar + 1'flour + 8'apples, 2'filling), -6)^{\leq\gamma_2}\ \textit{OR}$$
$$\textsc{bid}(1'(4'filling + 4'dough, 4'\textit{apple pie}), -14)^{\leq\gamma_3}$$

They represent the *Make Dough*, *Make Filling*, and *Baking* operations respectively. Notice that γ_1, γ_2, and γ_3 represent the maximum number of times each internal operation can be performed by the auctioneer. This is an example of the richer expressiveness of our bidding language, since in the case of MUCRAtR we had $\gamma_1 = \gamma_2 = \gamma_3 = \infty$. It is obvious that there always exists an upper bound on the number of times each physical operation is performed. The bids sent by the bidders, as expressed in equations (4.1) to (4.5), can be easily encoded in our bidding language as follows:

$$\mathcal{B}_1 = \textsc{bid}(1'(\emptyset, 100'butter + 200'margarine), -200) \tag{5.18}$$
$$\mathcal{B}_2 = \textsc{bid}(1'(\emptyset, 200'flours + 300'sugar), -100) \tag{5.19}$$
$$\mathcal{B}_3 = \textsc{bid}(1'(\emptyset, 800'apples), -200) \tag{5.20}$$
$$\mathcal{B}_4 = \textsc{bid}(1'(\emptyset, 200'dough + 200'filling), -1300) \tag{5.21}$$
$$\mathcal{B}_5 = \textsc{bid}(1'(\emptyset, 200'\textit{apple pies}), -2400) \tag{5.22}$$

□

- in *Combinatorial Auctions for SCF* bidders can send bids in the form:

$$\textsc{bid}(\{(\mathcal{I}, \mathcal{O})\}, p)$$

  such that:

  – there are not cycles in the supply chain network topology
  – $|\mathcal{O}| = 1$

5.6 Conclusions

In this chapter we provided a solution to requirements 1–13 of table 5.2. In what follows we list the solution provided by MMUCAs to the requirements associated to the *make-or-buy-or-collaborate* decision problem:

(1) MMUCAs support the representation of both cyclic and acyclic supply chain network topologies, since the bidding language and the definition of the WDP are independent on the topology of the network;

(2) MMUCAs allow to express complementarities among supply chain operations, since they allow the submission of bids on multisets of SCOs;

(3) MMUCAs allow bidders to require supply chain operations, as explained by means of example 5.4;

(4) MMUCAs allow to express resource sharing, as showed in item 16 of section 5.3.7;

(5) MMUCAs allow to express minimum/maximum capacity constraints on the number of times each supply chain operation can be performed via the constructs introduced in section 5.3.5;

(6) MMUCAs allow to express manufacturing operations with multiple outputs, as shown in item 14 of section 5.3.7;

(7) MMUCAs provide a coordinated scheduling plan among the supply chain stakeholders: the output of the MMUCAs WDP is an ordered and implementable sequence of SCOs;

(8) MMUCAs allow to solve *Make-or-buy*, *Make-or-buy-or-collaborate*, and SCF decision problems;

(9) MMUCAs support the specification of the configuration the auctioneer expects to end up with via the $\mathcal{U}_{out}$ multiset; and

(10) MMUCAs support the specification of the initial stock via the $\mathcal{U}_{in}$ multiset.

Summarising, the main extension introduced in this chapter with respect to CAs for SCF is that bidders can send bids in the form:

$$\text{BID}(\{\alpha'_1(\mathcal{I}_1, \mathcal{O}_1) + \ldots + \alpha'_n(\mathcal{I}_n, \mathcal{O}_n)\}, p) \; \textit{XOR} \; \text{BID}(\ldots)$$

i.e. to submit XOR combinations of atomic bids on *multisets of supply chain operations*. Hence, in particular, it improves the expressiveness and the range of solvable problems when employing Combinatorial Auctions for SCF.

Another important contribution of this chapter is the incorporation of the concept of a *sequence* of supply chain operations as a solution to the WDP. This is required if the intention is to model some sort of production process. We provide as a solution to the WDP the sequence of operations maximising an auctioneer's revenue and fulfilling the bidders' constraints.

Notice that there are two different ways in which an MMUCA can be employed. The hypothesis underlying both possibilities is that there is a mutual agreement between bidders and providers on which goods are negotiated[7]. Given this, we envisage two possibilities:

- the first one is that bidders are constrained to bid on a fixed set of previously defined supply chain operations. For instance, an auctioneer may constrain the bidding on supply chain operations like the ones in figure 5.1.

[7]We call the set of goods at auction the *negotiated goods*.

	Requirements	**MMUCA**	**TDN**
1	express an offer/request on bundles of goods	✓	✓
2	express an offer of a SCO with a single output product	✓	✓
3	express an offer of a SCO with multiple output products	✓	
4	express a request of a SCO	✓	
5	express the offer/request of a bundle of SCOs	✓	
6	express combinations of bids	✓	
7	express the min/max number of times SCOs are performed	✓	
8	express resource sharing	✓	
9	express an auctioneer's initial stock	✓	
10	express the auctioneer's final requirements	✓	
11	support *acyclic* supply chain networks	✓	✓
12	support *cyclic* supply chain networks	✓	
13	compute the *scheduled sequence* of SCOs to perform	✓	
14	ensure computational tractability while preserving optimality	?	
15	solve SCF decision problem	✓	✓
16	solve the *make-or-buy-or-collaborate* decision problem	✓	
17	formally represent the search space	?	
18	graphically represent the search space	?	
19	assess the computational tractability based on the problem structure	?	

Table 5.2: Requirements associated to the *make-or-buy-or-collaborate* problem.

- the second one is that there is a complete freedom of bidding on any supply chain operation, as long as it only involves the *negotiated goods* as inputs or outputs. For example, unlike in the previous point, a bidder may send a bid offering the supply chain operation *Make Pie*, that takes as inputs all the basic ingredients and provides a finished apple pie. Notice that this operation is not present in figure 5.1.

With the introduction of MMUCAs and of the associated bidding language, we consider solved requirements 1–13 in table 5.2. However, we have not provided any computational method for solving the WDP (requirements (14-19) in table 5.2). In the following chapters we provide some solutions to this issue.

Chapter 6

Solving the MMUCA Winner Determination Problem

By means of the MMUCA bidding language we make possible to express any possible type of supply chain operation over any type of supply chain network topology. Moreover, the MMUCA winning rule on the one hand accounts for the semantics of the bidding language, and on the other hand automates the supply chain formation and planning process. However, the auctioneer lacks of a computational method to solve the WDP. In this chapter, we provide a solution to such issue.

Firstly, applying a technique similar to the one employed for MUCRAtR in section 4.7, we succeed in mapping the MMUCA WDP to a *Constrained Maximum Weight Occurrence Sequence Problem* (CMWOSP). Likewise MUCRAtR, two benefits stem from this mapping. As a first benefit, we can inherit and import all the Place Transition Nets theoretical and formal results. As a second benefit, we succeed in efficiently solving the MMUCA WDP by means of Integer Programming (IP) for a wide class of supply chain network topologies, namely the acyclic ones.

The fact that the WDP can be solved by means of IP only for acyclic supply chain network topologies poses a serious requirement to the applicability of MMUCAs to some real-world scenarios. Thus, as a second result of this chapter, we extend the class of solvable MMUCA WD problems at the price of an efficiency decrement. We provide an IP model, built directly upon the definition of MMUCA WDP, that allows solving any class of problem on any network topology. However, the price to be paid is that the computational complexity of the underlying optimisation problem significantly increases.

This chapter is organised as follows. In section 6.1.1 we present a mapping of the MMUCA WDP to a CMWOSP. In section 6.2 we provide an IP formulation of the MMUCA WDP that applies to any network topology. Next, in section 6.3 we discuss briefly on computational complexity. Finally, in section 6.4 we draw some conclusions.

6.1 Mapping MMUCA to WPTN

In this section we demonstrate that an instance of the MMUCA WDP can be transformed into an instance of the Constrained Maximum Weight Occurrence Sequence Problem (CMWOSP), introduced in section 4.5.3. We recall that a CMWOSP is an optimisation problem defined on Weighted Place Transition Nets (WPTNs). WPTNs are an extension of Place Transition Nets (PTNs) in which a cost is associated to the firing of each transition (see section 4.4). We introduce this mapping because it allows:

- to incorporate analysis methods to analyse behavioural properties of WPTNs;
- exploiting such analysis methods we provide an IP formulation for some classes of WPTNs, and therefore some classes of supply chain network topologies.

6.1.1 The intuitions behind the mapping

The idea behind the mapping of the WDP to a CMWOSP is that an atomic *supply chain operation* (SCO) can be viewed as a transition in a WPTN. Consider the following offer, expressed by a bidder in the bidding language introduced in section 5.3:

$$Bid_1 = \text{BID}(1'(2'H_2O, 1'O_2 + 2'H_2), -8) \tag{6.1}$$

This represents an offer over an hydrolysis process: 2 moles of water are transformed into 1 mole of oxygen and two moles of hydrogen at a price of €8. Then, consider the transition depicted in figure 6.1, and say that each place represents a good. Let the place labelled with H_2O be water , H_2 be hydrogen , and O_2 be oxygen. The transition in figure perfectly captures the semantics of a supply chain operation: the input places of the transitions are the input goods of the SCO, its output places are the output goods of the SCO, and the transition cost is the cost associated to the SCO. Analogously, an SCO offering goods can be represented as a transition with only output places, whereas an SCO asking for goods as a transition with only input places.

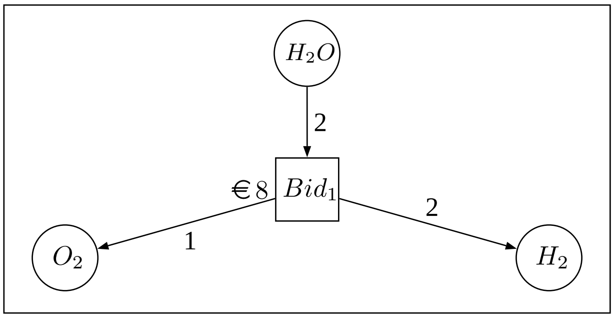


Figure 6.1: Example of an SCO represented as a transition in a WPTN.

Example 6.1. Say that the following bids, expressed in the bidding language of section 5.3 and graphically represented in the WPTN of figure 6.2, are submitted to an MMUCA:

(1) Bid bid_1 offers two moles of water at €10 (the minus represents the fact that the bidder gets paid):

$$bid_1 = \text{BID}(1'(\emptyset, 2'H_2O), -10) \tag{6.2}$$

(2) Bid bid_2 offers two moles of water at €14:

$$bid_2 = \text{BID}(1'(\emptyset, 2'H_2O), -14) \tag{6.3}$$

(3) Bid bid_3 stands for an offer to perform the hydrolysis process for €8:

$$bid_3 = \text{BID}(1'(2'H_2O, 1'O_2 + 2'H_2), -8) \tag{6.4}$$

(4) Bid bid_4 represents an offer to buy the products resulting from the reaction for €23 (the positive cost represents the fact that the bidder pays money):

$$bid_4 = \text{BID}(1'(2'H_2 + 1'O_2, \emptyset), 23) \tag{6.5}$$

(5) Bid bid_5 represents an offer to buy the products of the reaction for €25:

$$bid_5 = \text{BID}(1'(2'H_2 + 1'O_2, \emptyset), 25) \tag{6.6}$$

$\square$

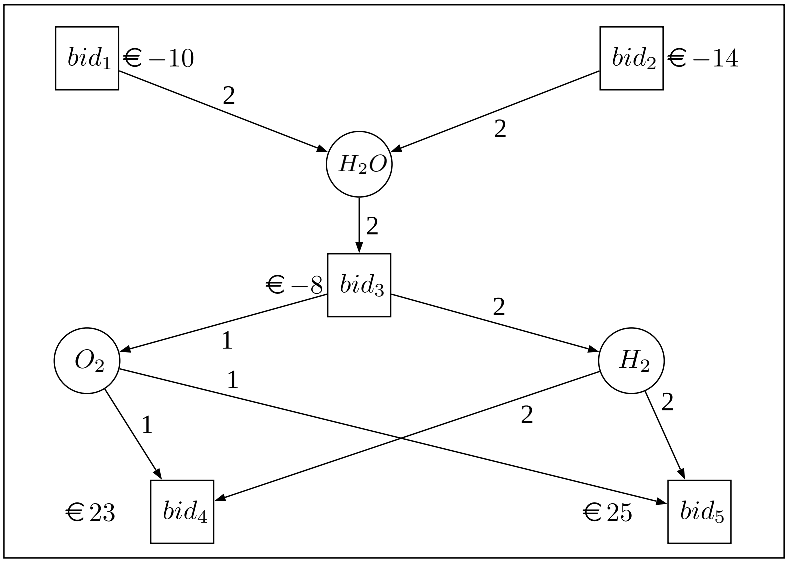


Figure 6.2: Example of bids in a MMUCA represented as a WPTN.

In example 6.1 finding the revenue maximising solution is straightforward. Firstly, buy two moles of water from bid_1, then process the water through the SCO in bid_3, and then sell the products of the reaction to bid_5. The total revenue of the supply chain is

$25 - 8 - 10 =$€ 7. Notice carefully that this solution is the solution to the CMWOSP[1] defined on the WPTN in figure 6.2 with initial marking empty and destination marking $\mathcal{M}_d$ satisfying the following constraints[2]:

$$\mathcal{M}_d(H_2O) \geq 0 \tag{6.7}$$

$$\mathcal{M}_d(O_2) \geq 0 \tag{6.8}$$

$$\mathcal{M}_d(H_2) \geq 0 \tag{6.9}$$

Given the example above, we argue that if we:

(1) build a WPTN joining all the atomic SCOs received within bids;

(2) set the initial marking to the goods initially available to the auctioneer; and

(3) set some constraints on the final marking,

then the solution to the CMWOSP corresponds to the solution to the MMUCA WDP.

Informally, this is the kind of mapping we intend to demonstrate. We obtain several advantages from this mapping. We can readily import a series of results and tools valid for PTNs, as for instance tools to analyse the reachability problem on the PTN. As a major benefit, we manage to efficiently encode the MMUCA WDP by means of IP. In particular, in response to the requirements 17–21 of table 5.1 of chapter 5, this mapping also allows us: (1) to visually and formally explicit the search space associated to the WDP; (2) to assess the computational tractability of the WDP based on the problem structure; and (3) to study structural and behavioural properties of the resulting supply chain. However, we have to take some more details into account:

- In the previous example, given the WPTN representation, each SCO can be used an arbitrarily number of times. Instead, the semantics of the bidding language imposes that SCOs must be used a limited number of times. Recall that we solved such problem in the case of MUCRAtR introducing *bid places* (see figure 4.3).

- How can we express on the WPTN an offer/demand over a bundle of (complementary) SCOs? That is, how could we express a bid like

$$\begin{aligned}\textsc{bid}(1'(1'butter + 3'sugar + 3'flour, 2'dough)+\\ 1'(8'apple + 2'margarine + 1'flour, 2'filling), -20)\end{aligned} \tag{6.10}$$

  offering to perform both the *Make Dough* and *Make Filling* operations?

- How can we express on the WPTN a set of mutually exclusive (XOR) atomic bids? That is, how could we express a bid like:

$$\textsc{bid}(1'(\emptyset, 200'\textit{margarine}), -200)\ \textit{XOR}\ \textsc{bid}(1'(\emptyset, 200'\textit{butter}), -200)$$

In the following section we provide an answer to all these questions.

[1]So far under the hypothesis that transitions can fire at most once. We will solve the issue of limiting the number of times each transition can fire further on.

[2]In case of no free-disposal on the auctioneer side replace $\geq$ with $=$.

6.1.2 Representing Bids

Firstly, we must recall the notation employed in section 5.4.2:

- The $j-$th an atomic bid of bidder i is represented by a pair

  $$Bid_{ij} = (\mathcal{D}_{ij}, p_{ij})$$

  where p_{ij} is the price a bidder is willing to pay/be paid to have allocated the multiset of SCOs $\mathcal{D}_{ij}$.
- For each Bid_{ij}, let t_{ijk} be the kth SCO in $\mathcal{D}_{ij}$.
- Let $\mathcal{D}_{ij}(t_{ijk})$ be the multiplicity of t_{ijk} in $\mathcal{D}_{ij}$.
- Let $\mathcal{D} = \cup_{ij}\mathcal{D}_{ij}$ be the multiset of the overall SCOs received with their multiplicity.
- Let δ be the overall number of SCOs mentioned anywhere in the bids, i.e. $\delta = |\mathcal{D}| = \sum_{ij} |\mathcal{D}_{ij}|$.
- Let T be the set of the overall SCOs in the bids without their multiplicity, that is $T = \{t_{ijk} : \forall ijk\}$;
- G is the set of negotiated goods
- $\mathcal{U}_{in} \in \mathbb{N}^G$ is a multiset of goods standing for the initial stock of the auctioneer.
- $\mathcal{U}_{out} \in \mathbb{N}^G$ is a multiset of goods standing for the number of goods the auctioneer desires to end up with.
- $\mathcal{M}^m \in \mathbb{N}^G$ is a multiset of goods standing for the number of goods available to the auctioneer after applying m supply chain operations in a production process.

In example 6.1, we restrict ourselves to the case in which agents can only submit one atomic bid. Moreover, we only consider bids over a single atomic SCO, i.e. $|\mathcal{D}_{ij}| = 1$. Next, we progressively relax all these constraints. First of all, we explain how to represent on a WPTN a bid on a bundle (multiset) of SCOs.

Expressing bids on bundles of SCOs

For a bid Bid_{ij}, combinatorial on SCOs, we have to ensure that:

- if an atomic SCO t_{ijk} in bid Bid_{ij} is included in the solution sequence,
  - it must be included in the solution $\mathcal{D}_{ij}(t_{ijk})$ times;
  - all the other atomic SCOs $t_{ijk'}$ within the same atomic bid (all the SCOs in $\mathcal{D}_{ij}$) must be included $\mathcal{D}_{ij}(t_{ijk'})$ times as well;

  this maps to item (1) of the definition of valid solution sequence (definition 5.9);

- the price that has to be paid to (received by) the bidder is the price of the whole bid (p_{ij}).

We achieve this by introducing some auxiliary places and transitions. The example in figure 6.3 represents the following bid:

$$Bid_{ij} = \textsc{bid}(1'(2'p_1, 1'p_2 + 2'p_3) + 3'(1'p_4, 1'p_6 + 1'p_7) + 2'(1'p_5, 1'p_8 + 1'p_9), -20)$$

If we refer to the three atomic SCOs as:

$$\begin{aligned} t_{ij1} &= (2'p_1, 1'p_2 + 2'p_3) \\ t_{ij2} &= (1'p_4, 1'p_6 + 1'p_7) \\ t_{ij3} &= (1'p_5, 1'p_8 + 1'p_9) \end{aligned}$$

We can rewrite the bid in a more readable way:

$$Bid_{ij} = \textsc{bid}(1't_{ij1} + 3't_{ij2} + 2't_{ij3}, -20)$$

This is a bid on a bundle of SCOs $\{t_{ij1}, t_{ij2}, t_{ij3}\}$ with associated price $p_{ij} = -20$€.

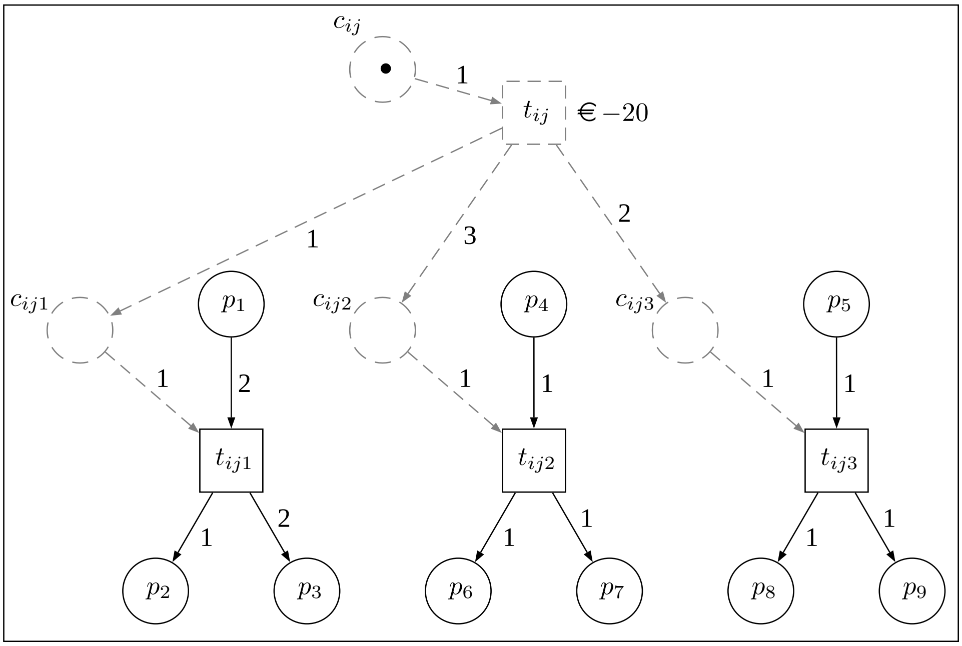


Figure 6.3: Bids on bundles of SCOs.

In general, in order to incorporate a bid over multiple SCOs we proceed as follows:

- for each bid Bid_{ij} we introduce an auxiliary transition t_{ij} (*bid transition*) and an auxiliary place c_{ij} (*bid place*).
- for each atomic SCO t_{ijk} within bid Bid_{ij}, we add an auxiliary place c_{ijk} (c_{ij1}, c_{ij2}, and c_{ij3} in figure 6.3), called *SCO place*.

- we attach the valuation p_{ij} of bid Bid_{ij} to the corresponding *bid transition* t_{ij}. In the example, we associate the bid cost p_{ij}=-€20 to transition t_{ij}. Hence, whenever t_{ij} fires, the cost/gain p_{ij} is added to the cost of the firing sequence.

It is easy to check that the WPTN if figure 6.3 allows for firing any subset of $\{t_{ij1}, t_{ij2}, t_{ij3}\}$ (depending on the tokens) with the corresponding multiplicities $(1, 3, 2)$. Notice also that firing at least one of the three transitions requires to previously fire transition t_{ij}, because this guarantees having the required tokens in the input places c_{ijk}. In this way, we guarantee that firing at least one of the transitions implies firing also t_{ij}, and therefore that the corresponding money is added to the overall cost/revenue (recall that we are dealing with a WPTN).

Any legal firing sequence on the WPTN in figure 6.3 guarantees that selecting at least one of the t_{ijk} implies also selecting t_{ij}. However, we need a further requirement: either none of the t_{ijk} fires, or all of them fire. If they all fire, they have to fire as many times as expressed by their multiplicities in the bids. In the figure, we have to enforce that if t_{ij} fires, then t_{ij1} fires once ($\mathcal{D}_{ij}(t_{ij1}) = 1$), t_{ij2} three times ($\mathcal{D}_{ij}(t_{ij2}) = 3$), and t_{ij3} twice ($\mathcal{D}_{ij}(t_{ij3}) = 2$).

The topology in figure 6.3 cannot guarantee such property by itself. For instance, a firing sequence in which only transitions t_{ij1} and t_{ij2} fire (not t_{ij3}) is legal but does not comply with our all-or-none assumption. In order to enforce it, we simply impose some constraints on the final configuration of the net. Say that we impose that in the final configuration c_{ij1}, c_{ij2}, and c_{ij3} contain no tokens. More formally, the final marking should fulfil the constraints:

$$\begin{cases} \mathcal{M}_d(c_{ij1}) & = 0 \\ \mathcal{M}_d(c_{ij2}) & = 0 \\ \mathcal{M}_d(c_{ij3}) & = 0 \end{cases} \tag{6.11}$$

This implies that all the legal firing sequences leading to the final configuration $\mathcal{M}_d$ contain either none or the three transitions $t_{ij1}, t_{ij2}, t_{ij3}$ with multiplicities 1, 2, and 3 respectively. In fact the only possible firing sequences are either no firings $J = \{\}$, or

$$J = \langle t_{ij}, t_{ij1}, t_{ij2}, t_{ij2}, t_{ij2}, t_{ij3}, t_{ij3} \rangle \tag{6.12}$$

$$J = \langle t_{ij}, t_{ij3}, t_{ij3}, t_{ij2}, t_{ij1}, t_{ij2}, t_{ij2} \rangle \tag{6.13}$$

$$J = \dots \tag{6.14}$$

We remark that the semantics of multiplicity of the SCOs offered in bid Bid_{ij} is completely captured by the provided WPTN. The weights of the arcs connecting bid transitions t_{ij} and SCO places c_{ijk}, along with the constraints on the final marking, enforces that none of the SCOs in $\mathcal{D}_{ij}$ is used, or all of them are used as many times as indicated by their multiplicities in $\mathcal{D}_{ij}$.

Expressing XOR of atomic bids

We are now able to represent an atomic bid on a WPTN. However, we still have to express the XOR relationships among the atomic bids that come from the same bidder

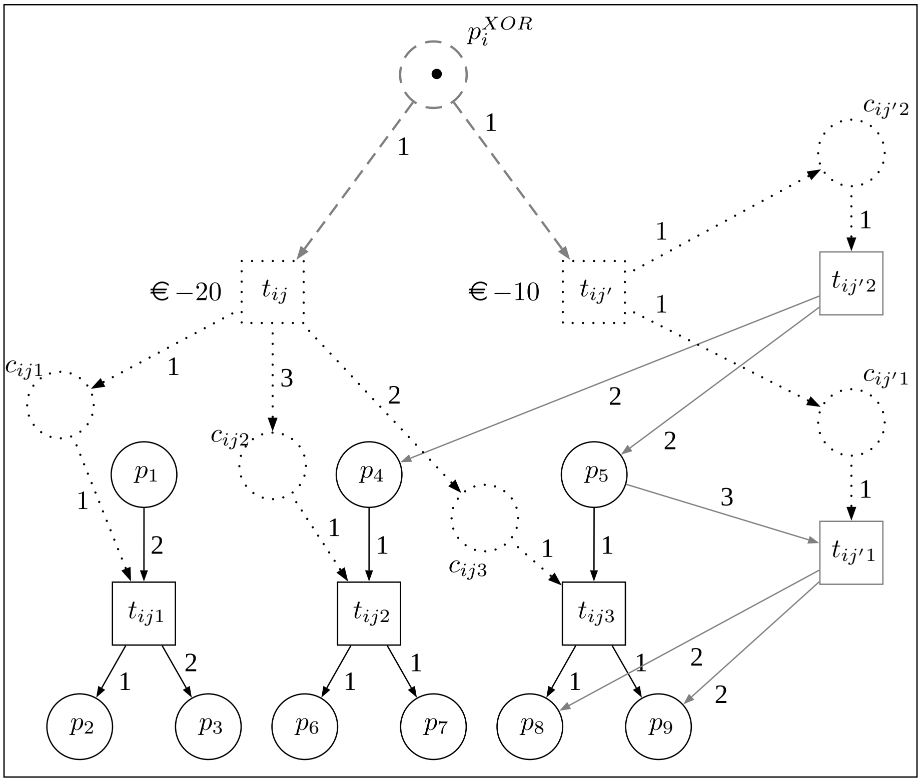


Figure 6.4: XOR of atomic bids

to fully represent our bidding language. Consider the following bid:

$$\text{BID}(1'(2'p_1, 1'p_2 + 2'p_3) + 3'(1'p_4, 1'p_6 + 1'p_7) + 2'(1'p_5, 1'p_8 + 1'p_9), -20)$$
$$XOR$$
$$\text{BID}(1'(\emptyset, 2'p_4 + 2'p_5) + 1'(3'p_5, 2'p_8 + 2'p_9), -10)$$

If we refer to the five atomic SCOs as:

$$\begin{aligned} t_{ij1} &= (2'p_1, 1'p_2 + 2'p_3) \\ t_{ij2} &= (1'p_4, 1'p_6 + 1'p_7) \\ t_{ij3} &= (1'p_5, 1'p_8 + 1'p_9) \\ t_{ij'1} &= (3'p_5, 2'p_8 + 2'p_9) \\ t_{ij'2} &= (\emptyset, 2'p_4 + 2'p_5) \end{aligned}$$

We can rewrite the bid in a more readable way as follows:

$$\text{BID}(1't_{ij1} + 3't_{ij2} + 2't_{ij3}, -20) \tag{6.15}$$

$$XOR \tag{6.16}$$

$$\text{BID}(1't_{ij'1} + 1't_{ij'2}, -10) \tag{6.17}$$

We refer to the two bids submitted by a bidder i in XOR as to Bid_{ij} and $Bid_{ij'}$. This means that an auctioneer can select at most one of them (see section 5.3).

Figure 6.4 depicts bids Bid_{ij} and $Bid_{ij'}$. Bid Bid_{ij} is over SCOs t_{ij1}, t_{ij2}, and t_{ij3}, whereas bid $Bid_{ij'}$ is over SCOs $t_{ij'1}$ and $t_{ij'2}$. The cost associated to Bid_{ij} is $c(t_{ij}) = -€\,20$, and the cost associated to $Bid_{ij'}$ is $c(t_{ij'}) = -€\,10$.

In order to incorporate the semantics of the XOR operator into the WPTN, we introduce a new place, labelled with p_i^{XOR}, called *XOR place*. Notice that *bid places* c_{ij} and $c_{ij'}$ have been substituted by the *XOR place*. This WPTN topology enforces that at most one of the two transitions t_{ij} and $t_{ij'}$ can fire. When either of them fires, it consumes the unique token in p_i^{XOR} inhibiting the firing of the other one. It is clear from previous section that transition t_{ij} represents bid Bid_{ij} and transition $t_{ij'}$ represents bid $Bid_{ij'}$. This corresponds to selecting at most one bid out of bids Bid_{ij} and $Bid_{ij'}$. This reasoning applies to the case of m bids in XOR among them as well.

6.1.3 The Mixed Auction Net

In the previous section, we showed the intuitions behind the mapping of the MMUCA WDP to a CMWOSP. We recall that the CMWOSP is an optimisation problem defined on WPTNs, thoroughly explained in section 4.5.3.

In section 4.6 we succeeded in mapping the MUCRAtR Winner Determination Problem to a CMWOSP. In order to perform such mapping we had to build a WPTN departing from the internal production structure of an auctioneer and from the received bids. This WPTN was called the *Auction Net*. Along the lines of such strategy, in this section we build a WPTN with a similar function for the MMUCA WDP. We shall call such WPTN the *Mixed Auction Net*, a WPTN that shall allow us to define the MMUCA WDP as a CMWOSP.

We will now provide the definition of *Mixed Auction Net*. Informally, such net is composed of three types of places, namely:

- *good places*, representing goods at auction;
- *SCO places*, useful to control the number of times each SCO is performed
- *XOR places*, useful to control that at most one bid per bidder is selected

Then, it is composed of two types of transitions, namely:

- *SCO transitions*, that represent the SCOs submitted by the bidders
- *bid transitions*, useful to control the number of times each SCOs is employed

The arc weights are associated so that some properties are fulfilled. Finally, we associate a cost to each *bid transition*, corresponding to the valuation associated to a bundle of SCOs.

In what follows the notation employed for describing bids is the one defined at the beginning of section 6.1.2. Moreover, we indicate with $\mathcal{I}_{ijk}$ and $\mathcal{O}_{ijk}$ respectively the input and output multisets of SCO t_{ijk}.

Definition 6.1. Given a finite set of bids B in the *XOR* bidding language over a set of goods G, a *Mixed Auction Net* is a WPTN $S^* = (P^*, T^*, A^*, E, \mathcal{M}_0, C)$ where

$$\begin{cases} P^* &= P_G \cup P_{SCO} \cup P_{XOR} \\ T^* &= T_B \cup T_{SCO} \\ A^* &= A_{SCO} \cup A_B \cup A_{XOR} \end{cases}$$

and

(1) P_G is the set of *good places*. For each good $g \in G$ add a place p_g.

(2) P_{SCO} is the set of *SCO places*. For each atomic SCO $t_{ijk} \in T$ add a place c_{ijk}.

(3) P_{XOR} is the set of *XOR places*. For each bidder i add a place p_i^{XOR}.

(4) T_B is the set of *bid transitions*. For each bid $Bid_{ij} \in B$ add a transition t_{ij}.

(5) T_{SCO} is the set of *SCO transitions*. For each atomic SCO[3] $t_{ijk} \in T$ add a transition t_{ijk}.

(6) A_{SCO} is the set of *SCO arcs*. It is built as follows:

$$A_{SCO} = A^i_{SCO} \cup A^o_{SCO}$$

where

$$\begin{aligned} A^i_{SCO} &= \{(p_g, t_{ijk}) \in P_G \times T_B \mid g \in \mathcal{I}_{ijk}\} \\ A^o_{SCO} &= \{(t_{ijk}, p_g) \in T_B \times P_G \mid g \in \mathcal{O}_{ijk}\} \end{aligned}$$

are the *input SCO arcs* and *output SCO arcs* respectively.

(7) A_B is the set of *bid arcs*. It is built as follows

$$A_B = A^i_B \cup A^o_B$$

where

$$\begin{aligned} A^o_B &= \{(t_{ij}, c_{ijk}) \in T_B \times P_{SCO}\} \\ A^i_B &= \{(c_{ijk}, t_{ijk}) \in P_{SCO} \times T_{SCO}\} \end{aligned}$$

are the *input bid arcs* and *output bid arcs* respectively.

[3]Henceforth, we indicate with the same label transitions on the WPTN and the corresponding supply chain operations.

(8) A_{XOR} is the set of *XOR arcs*. It is built as follows:

$$A_{XOR} = \{(p_i^{XOR}, t_{ij}) \in P_{XOR} \times T_B\}$$

(9) The arc expression function is built as follows:

$$E(p_g, t_{ijk}) = \mathcal{I}_{ijk}(g) \qquad (p_g, t_{ijk}) \in A^i_{SCO} \tag{6.18}$$
$$E(t_{ijk}, p_g) = \mathcal{O}_{ijk}(g) \qquad (t_{ijk}, p_g) \in A^o_{SCO} \tag{6.19}$$
$$E(c_{ijk}, t_{ijk}) = 1 \qquad (c_{ijk}, t_{ijk}) \in A^i_B \tag{6.20}$$
$$E(t_{ij}, c_{ijk}) = \mathcal{D}_{ij}(t_{ijk}) \qquad (t_{ij}, c_{ijk}) \in A^o_B \tag{6.21}$$
$$E(p_i^{XOR}, t_{ij}) = 1 \qquad (p_i^{XOR}, t_{ij}) \in A_{XOR} \tag{6.22}$$

(10) The bid cost function $C : B \to \mathbb{R}$ is built as follows:

$$C(t_{ijk}) = 0 \qquad t_{ijk} \in T_{SCO}$$
$$C(t_{ij}) = p_{ij} \qquad t_{ij} \in T_B$$

(11) The initial marking is defined as

$$\mathcal{M}_0(p) = \begin{cases} \mathcal{U}_{in}(g) & p_g \in P_G \\ 1 & p \in P_{XOR} \\ 0 & p \in P_{SCO} \end{cases} \tag{6.23}$$

□

Informally, P_G represents the set of negotiated goods, T_B the set of atomic bids, P_{XOR} the set of bidders, T_{SCO} the set of atomic supply chain operations, and P_{SCO} the set of places that *controls* the execution of SCOs.

Then, A_{SCO} connects the places representing the input goods and output goods of each atomic SCOs t_{ijk} to the transition representing it (t_{ijk}). The input goods are connected by incoming arcs whereas the output goods by outgoing arcs. For instance, transition t_{ij1} in figure 6.4 corresponds to the atomic SCO $t_{ij1} = (\mathcal{I}_{ij1}, \mathcal{O}_{ij1}) = (2'p_1, 1'p_2{+}2'p_3)$. Therefore, place p_1, representing the input good to t_{ij1}, is connected to transition t_{ij1} by means of an incoming arc; and places p_2 and p_3, representing its output goods, are connected to t_{ij1} by means of outgoing arcs.

Then, A_B is a set of arcs such that: (1) bid transition t_{ij} is connected to SCO places c_{ijk}; and (2) SCO places c_{ijk} are connected to atomic transitions t_{ijk}:

A_{XOR} is the set of arcs that connects all the p_i^{XOR} places to the bid transitions t_{ij} corresponding to bids coming from the same provider i.

The bid cost function $C : B \to \mathbb{R}$ is built in a way such that:

- the cost of a SCO is 0: $c(t_{ijk}) = 0$; and
- the cost of a bid transitions t_{ij} is the price of bid Bid_{ij} ($c(t_{ij}) = p_{ij}$).

Example 6.2. The *Mixed Auction Net* associated to example in figure 6.4 is defined as follows:

- $P_G = \{p_1, ..., p_9\}$.
- $T_B = \{t_{ij}, t_{ij'}\}$.
- $P_{XOR} = \{p_i^{XOR}\}$.
- $T_{SCO} = \{t_{ij1}, t_{ij2}, t_{ij3}, t_{ij'1}, t_{ij'2}\}$.
- $P_{SCO} = \{c_{ij1}, c_{ij2}, c_{ij3}, c_{ij'1}, c_{ij'2}\}$.
- $A_{SCO} = \{(p_1, t_{ij1}), (t_{ij1}, p_2), (t_{ij1}, p_3), \ldots, (p_5, t_{ij'1}), (t_{ij'1}, p_8), (t_{ij'1}, p_9)\}$.
- $A_B = \{(t_{ij}, c_{ij1}), (c_{ij1}, t_{ij1}), (t_{ij}, c_{ij2}), (c_{ij2}, t_{ij2}), ..., (t_{ij'}, c_{ij'1}), (c_{ij'1}, t_{ij'1})\}$.
- $A_{XOR} = \{(p_i^{XOR}, t_{ij}), (p_i^{XOR}, t_{ij'})\}$.
- The E function is[4]:

$$\begin{aligned} E(p_1, t_{ij1}) &= 2 \\ E(t_{ij1}, p_2) &= 1 \\ E(t_{ij1}, p_3) &= 2 \\ E(t_{ij}, c_{ij2}) &= 3 \\ &\cdots \end{aligned}$$

- The cost function is:

$$C(t) = \begin{cases} -20 & t = t_{ij} \\ -10 & t = t_{ij'} \\ 0 & otherwise \end{cases} \tag{6.24}$$

- The initial marking $\mathcal{M}_0$ is:

$$\mathcal{M}_0(p) = \begin{cases} 1 & p = p_i^{XOR} \\ 0 & otherwise \end{cases} \tag{6.25}$$

$\square$

6.1.4 Expressing the MMUCA WDP as a CMWOSP

In this section we introduce a CMWOSP on the *Mixed Auction Net*, whose solution can be easily transformed into a solution to the corresponding MMUCA WDP. In this way, we can exploit several results valid for CMWOSPs, WPTNs and PTNs. In particular, by means of this mapping, we can solve the MMUCA WDP by means of ILP whenever the associated *Mixed Auction Net* is acyclic (see section 4.7). Our aim in this section is showing that, from the firing sequence associated to a particular CMWOSP on the *Mixed Auction Net*, we can derive an optimal solution sequence to the corresponding MMUCA WDP.

[4]We only provide a sample of its definition. The whole definition is represented in figure 6.4.

This mapping is based on the analogy between a valid solution sequence and a firing sequence solution to a CMWOSP. In fact, we can prove that a sequence of SCOs solution to the MMUCA WDP and a sequence of transitions solution to a CMWOSP are objects fulfilling similar constraints. In fact, we want to show that there is a strong analogy between the SCOs in a MMUCA and the SCO transitions in the Mixed Auction Net, as well as between the bids in a MMUCA and the bids transitions in the Mixed Auction Net. In section 6.1.2 we provided some intuitions about this mapping. Obviously, the *Mixed Auction Net* will play a fundamental role in this sense. The central point is that, as mentioned in section 6.1.2, we have to impose some conditions on the number of tokens each place contains at the end of the firing sequence (sections 6.1.2 and 6.1.2) in order to ensure that:

- the auctioneer fulfils its requirements; and
- the semantics of the bidding language is fulfilled.

In particular, we have to ensure that:

- (*Auctioneer*) the *good places* will contain *at least*[5] the number of tokens corresponding to the number of goods the auctioneer expects to end up with (specified by $\mathcal{U}_{out}$).
- (*Bidding language*)
  - the *XOR place* will contain *at least* zero tokens. This ensures that at most one among the XOR bids is selected[6]. We say *at least* since it may be that no bid is selected, thus leaving a token in the place.
  - the *SCO places* will contain *exactly* zero tokens. This will enforce that SCOs of a same atomic bid are either all selected with the correct multiplicity, or none of them is selected[7].

With these constraints in mind when considering solutions to a MMUCA WDP, we can finally link the solutions to the MMUCA WDP with the solutions to a CMWOSP over a *Mixed Auction Net* as follows.

Theorem 6.1. *Given a MMUCA with a multiset of available goods $\mathcal{U}_{in}$, a set of required goods $\mathcal{U}_{out}$, and a set of bids B in the XOR language*[8] *over the goods in G, solving MMUCA WDP amounts to solving the CMWOSP defined on the* Mixed Auction Net $S^* = (P^*, T^*, A^*, E, \mathcal{M}_0, C)$*, with destination marking $\mathcal{M}_d$ fulfilling the following constraints*[9]*:*

$$\mathcal{M}_d(p) \geq \mathcal{U}_{out}(g) \qquad p_g \in P_G \tag{6.26}$$

$$\mathcal{M}_d(p) = 0 \qquad p \in P_{SCO} \tag{6.27}$$

$$\mathcal{M}_d(p) \geq 0 \qquad p \in P_{XOR} \tag{6.28}$$

[5] *Substitute* at least *for* Exactly *in the case of* no-free-disposal *on the auctioneer's side.*

[6] Under the hypothesis that the *XOR place* contains one token in the initial marking.

[7] Under the hypothesis that the *SCO place* contains zero tokens in the initial marking.

[8] Notice that in the case of OR language we could state exactly the same if we make appropriate changes to the WPTN. We should just represent all the bids as in figure 6.3, i.e. omitting the XOR places.

[9] In case of no free disposal on the auctioneer's side simply substitute $=$ for $\geq$ in equation (6.26).

Proof. $\Rightarrow$) First, we begin proving that a solution to the CMWOSP can be transformed into a solution to the MMUCA WDP. Recall that each atomic bid consists of a multiset of SCOs and a price: $Bid_{ij} = (\mathcal{D}_{ij}, p_{ij})$, where $\mathcal{D}_{ij} \in \mathbb{N}^{(\mathbb{N}^G \times \mathbb{N}^G)}$ is a multiset of SCOs and $p_{ij} \in \mathbb{R}$ is the associated cost/price. The notation employed is the one introduced in section 6.1.2.

We recall that a *solution sequence* is a mapping Σ from positions to SCOs:

$$\Sigma : \{1, 2, \ldots, \ell\} \rightarrow T$$

where $\ell \in \mathbb{N}$ is the length of the sequence, and T is the overall set of SCOs contained in all bids. Then, in WPTN terms, we can regard the *solution sequence* Σ as a sequence of *SCO transitions* on the mixed auction net.

Say that J^* is the solution to the CMWOSP described in the theorem we are proving and that Σ^* is the sequence obtained by J^* restricted to the elements of T_{SCO} (or, equivalently, without the elements of T_B). Recall that J^* contains both *bid transitions* and *SCO transitions*.

$$\Sigma^* = J^*|_{T_{SCO}} \tag{6.29}$$

and say that B^* is the *set* of transitions removed from J^* to obtain Σ^* :

$$B^* = \{t_{ij} \in J^*|_{T_B}\} \tag{6.30}$$

Obviously, $B^* \subseteq T_B$ is a subset of the *bid transitions*.

We aim at showing that Σ^* is the solution to the corresponding MMUCA WDP and that B^* is the set of selected bids. Recall that each transition in T_{SCO} represents an SCO. Then Σ^* can be seen as a sequence of SCOs as well:

$$\Sigma^* : \{1, 2, \ldots, \ell\} \rightarrow T_{SCO} \tag{6.31}$$

Notice that $T_{SCO} \equiv T$.

For this reason we have to check that Σ^* it is a *valid solution sequence*. That is, it must fulfil each of the constraints expressed in definition 5.9:

(1) Σ^* either contains all or none of the SCOs belonging to the same atomic bid, so that the semantics of the BID-operator is fulfilled:

$$\exists k : t_{ijk} \in \Sigma^* \;\Rightarrow\; \forall k \; |\Sigma^{*-1}(t_{ijk})| = D_{ij}(t_{ijk})$$

In section 6.1.2 we gave the intuitions that this is the case. However, to prove it formally, we write the state equation (see equation (2.25)) at a generic *SCO place* $c_{ijk} \in P_{SCO}$:

$$\mathcal{M}_d(c_{ijk}) = \mathcal{M}_0(c_{ijk}) + A^T \cdot \mathbf{x} \tag{6.32}$$

Notice from figure 6.4 that that both transitions $t_{ij} \in T_B$ and $t_{ijk} \in T_{SCO}$ can add/remove tokens to/from c_{ijk}. Notice also that according to equation (6.23) no tokens are present initially in c_{ijk}. Then, we can rewrite the equation as:

$$\mathcal{M}_d(c_{ijk}) = 0 + x_{t_{ijk}} - x_{t_{ij}} \cdot \mathcal{D}_{ij}(t_{tjk}) \tag{6.33}$$

where $x_{t_{ij}}$ and $x_{t_{ijk}}$ stands for the number of times t_{ij} and t_{ijk} fire in the firing sequence J^* respectively.

Then, applying constraint (6.27) over place c_{ijk} we obtain:

$$\mathcal{M}_d(c_{ijk}) = 0 \qquad \forall ijk \tag{6.34}$$

Merging with equation (6.33) we obtain:

$$0 = x_{t_{ijk}} - x_{t_{ij}} \cdot \mathcal{D}_{ij}(t_{tjk}) \qquad \forall ijk \tag{6.35}$$

$$x_{t_{ijk}} = x_{t_{ij}} \cdot \mathcal{D}_{ij}(t_{ijk}) \qquad \forall ijk \tag{6.36}$$

From equation (6.36) we can derive the following chain of implications:

$$\exists k : t_{ijk} \in \Sigma^* \Rightarrow \exists k : t_{ijk} \in J^* \Rightarrow t_{ij} \in J^* \Rightarrow t_{ij} \in B^* \Rightarrow \ldots \tag{6.37}$$

$$\ldots \Rightarrow \forall k |J^{*-1}(t_{ijk})| = \mathcal{D}_{ij}(t_{ijk}) \Rightarrow \forall k |\Sigma^{*-1}(t_{ijk})| = \mathcal{D}_{ij}(t_{ijk}) \tag{6.38}$$

Then, taking the first premise and the last consequence, we have:

$$\exists k : t_{ijk} \in \Sigma^* \Rightarrow |\Sigma^{*-1}(t_{ijk})| = \mathcal{D}_{ij}(t_{ijk}) \forall k \tag{6.39}$$

That is what we wanted to show.

(2) Σ^* does not contain two SCOs belonging to different atomic bids by the same bidder, and thus the semantics of the XOR operator is fulfilled:

$$t_{ijk}, t_{ij'k'} \in \Sigma^* \;\Rightarrow\; j = j'$$

In order to demonstrate this result, we write the state equation at each of the $p_i^{XOR} \in P_{XOR}$ place and we proceed similarly to the previous demonstration. We obtain:

$$\mathcal{M}_d(p_i^{XOR}) = 1 - \sum_j x_{t_{ij}} \tag{6.40}$$

and then applying the constraint in equation (6.28), we have:

$$1 - \sum_j x_{t_{ij}} \geq 0 \tag{6.41}$$

$$\sum_j x_{t_{ij}} \leq 1 \tag{6.42}$$

From equation (6.37) we know that:

$$t_{ijk} \in \Sigma^* \Rightarrow t_{ij} \in J^* \tag{6.43}$$

$$t_{ij'k'} \in \Sigma^* \Rightarrow t_{ij'} \in J^* \tag{6.44}$$

However, for the constraint in equation (6.42) we have that:

$$t_{ij}, t_{ij'} \in J^* \Rightarrow j = j' \tag{6.45}$$

Then, joining the implications we have:

$$t_{ijk}, t_{ij'k'} \in \Sigma^* \Rightarrow j = j' \tag{6.46}$$

As we wanted to demonstrate.

(3) Equations (5.10) and (5.11) hold at each step of the solution sequence Σ^*.

$$\mathcal{M}^m(g) = \mathcal{M}^{m-1}(g) + \mathcal{O}_{\Sigma^*(m)}(g) - \mathcal{I}_{\Sigma^*(m)}(g) \tag{6.47}$$
$$\mathcal{M}^{m-1}(g) \geq \mathcal{I}_{\Sigma^*(m)}(g) \tag{6.48}$$

This condition ensures that all SCOs have enough input goods available at each step of the SCO sequence.

We recall that the places in P_G represent the goods in G. For the sake of clarity we rewrite here both equations:

$$\mathcal{M}^m(p) = \mathcal{M}^{m-1}(p) + E(t,p) - E(p,t) \qquad \forall p \in {}^\bullet t \cup t^\bullet \tag{6.49}$$
$$\mathcal{M}^{m-1}(p) \geq E(p,t) \qquad p \in {}^\bullet t \tag{6.50}$$

We recall that these equations represent the change in the $\mathcal{M}^{m-1}$ marking after the firing of a transition t (equation (6.49)), and the condition of activation of transition t in marking $\mathcal{M}$ (equation (6.50)).

Next, we aim at writing equations (2.15) and (2.14) at each place in P_G and at each step of the firing sequence J^*. Notice from figure 6.4 that the only transitions that add/remove tokens from/to the places in P_G are the transitions in P_{SCO}. Then, instead of J^*, we can employ Σ^* of equation (6.29):

$$\begin{cases} \mathcal{M}^m(p_g) & = \mathcal{M}^{m-1}(p_g) + \mathcal{O}_{\Sigma^*(m)}(p_g) - \mathcal{I}_{\Sigma^*(m)}(p_g) \\ \mathcal{M}^{m-1}(p_g) & \geq \mathcal{I}_{\Sigma^*(m)}(p_g) \end{cases} \tag{6.51}$$

That is exactly what we required.

(4) The set of goods held by the auctioneer after implementing the SCO sequence is a superset of the goods the auctioneer is expected to end up with :

$$\mathcal{U}_{in}(g) + \sum_{m=0}^{\ell} \left(\mathcal{O}_{\Sigma^*(m)}(g) - \mathcal{I}_{\Sigma^*(m)}(g)\right) \geq \mathcal{U}_{out}(g)$$

Considering the constraints on the final marking of equation (6.26) and the initial marking of equation (6.23), we obtain:

$$\mathcal{U}_{in}(p_g) + \sum_{l=0}^{\ell} (\mathcal{O}_{\Sigma^*(l)}(p_g) - \mathcal{I}_{\Sigma^*(l)}(p_g)) \geq \mathcal{U}_{out}(p_g) \tag{6.52}$$

Observe that since the only transitions that have associated non-null costs are the *bid transitions* in T_B, and according to equation (6.39), the cost associated to the firing sequence is:

$$C_T(J^*) = \sum_{t_{ij} \in J^*} C(t_{ij}) = \sum_{t_{ij} \in B^*} C(t_{ij}) = \sum_{t_{ij} \in B^*} p_{ij} \tag{6.53}$$

It is obvious from equation (6.37) that having transition t_{ij} in the solution sequence means that bid Bid_{ij} is in the winning set. Then, we have:

$$\sum_{t_{ij} \in B^*} p_{ij} = \sum_{Bid_{ij} \in \textit{Winning Set}} p_{ij} \tag{6.54}$$

Then, the quantity maximised by the CMWOSP is equivalent to the auctioneer's revenue. Hence, Σ^* is a valid solution and maximises the auctioneer revenue. Then, it is the solution to the MMUCA WDP according to definition 5.10.
$\Leftarrow$) We prove the converse as well. Given a solution to the MMUCA WDP, it can be transformed into a solution to the CMWOSP described in the theorem we are proving. Say Σ is the solution to the MMUCA WDP. Then, consider the following constructs.

- The sequence of SCO transitions $\Sigma^* : \mathbb{N} \rightarrow T_{SCO}$ such that:

  $$\forall m \in [1, |\Sigma|] \qquad \Sigma^*(m) = t_{ijk} \Longleftrightarrow \Sigma(m) = t_{ijk} \tag{6.55}$$

  Recall that $T \equiv T_{SCO}$.

- The set of bid transitions $B^* \subseteq T_B$ such that:

  $$t_{ij} \in B^* \Longleftrightarrow \exists k \textit{ s.t. } t_{ijk} \in \Sigma^* \tag{6.56}$$

- The sequence of bid transitions $J_B^* : \mathbb{N} \rightarrow T_B$ formed by arranging in a random order the elements of B^*. More formally, the sequence must be such that:

  $$|J_B^{*-1}(t_{ij})| = \begin{cases} 1 & \forall t_{ij} \in B^* \\ 0 & otherwise \end{cases} \tag{6.57}$$

- $J^* : T_B \cup T_{SCO} \rightarrow [1, |J_B^*| + |\Sigma^*|]$ is a sequence of transitions obtained concatenating the sequences J_B^* and Σ^*. Observe that the sequences are concatenated in such a way that the elements of J_B^* are placed before the elements of Σ^*.

Σ^* corresponds to the sequence of SCOs solution to the MMUCA WDP, whereas the sequence J_B^* contains the bid transitions corresponding to the winning bids. That is, if $t_{ij} \in J_B^*$, then Bid_{ij} is in the winning set.

Then, we aim at showing that the sequence J^* is a solution to the CMWOSP on the mixed auction net with final constraints in equations (6.26), (6.27), and (6.28). With this purpose, we have to perform three steps.

(1) we have to make sure that the final marking constraints are fulfilled;

(2) we have to make sure that all the transitions in J^* are enabled at the step they are executed; and

(3) we have to make sure that the solution is optimal, i.e. that there is not another solution to the CMWOSP with higher associated cost.

In order to solve item (1), we make the hypothesis that all the transitions in J^* are enabled. This hypothesis will be confirmed later on. Under this hypothesis, we can write equations (2.14) and (2.15) at the $m-th$ step of J^* in the following form:

$$\mathcal{M}^{m-1}(p) \geq E(p, J^*(m)) \tag{6.58}$$

$$\mathcal{M}^{m}(p) = \mathcal{M}^{m-1}(p) + E(J^*(m), p) - E(p, J^*(m)) \tag{6.59}$$

Embedding the recursion, we can obtain the marking at step m as:

$$\mathcal{M}^{m}(p) = \mathcal{M}_0 + \sum_{l=1}^{m} (E(J^*(l), p) - E(p, J^*(l))) \tag{6.60}$$

Then, we write this equation in the final state for all the places in the mixed auction net. Then, say that $\ell = |J^*|$ is the length of the sequence J^*. Analogously, we note $\ell_B = |J^*_B|$ and $\ell_\Sigma = |\Sigma^*|$ We know from section 6.1.3 that the auction net has three types of places (P_G, P_{sco} and P_{XOR}).

- P_{XOR} places. We know that equation (6.28) must hold. Then, we must have that:

$$\mathcal{M}^{\ell}(p_i^{XOR}) \geq 0 \qquad \forall i \tag{6.61}$$

  $\mathcal{M}^{\ell}$ takes the following form for all $p_i^{XOR} \in P_{XOR}$:

$$\mathcal{M}^{\ell}(p_i^{XOR}) = \mathcal{M}_0(p_i^{XOR}) + \sum_{m=1}^{\ell} \left(E(J^*(m), p_i^{XOR}) - E(p_i^{XOR}, J^*(m))\right)$$

  that taking into account definition 6.1 becomes:

$$\begin{aligned}\mathcal{M}^{\ell}(p_i^{XOR}) &= 1 - \sum_{m=1}^{\ell} E(p_i^{XOR}, J^*(m)) = 1 - \sum_{j} |J^{*-1}(t_{ij})| = \\ &= 1 - \sum_{j} |J_B^{*-1}(t_{ij})|\end{aligned}$$

  The intuition behind this are provided by figure 6.4. No transitions are incoming into place p_i^{XOR}, and the only outgoing transitions are t_{ij} and $t_{ij'}$. It is easy to see that since Σ is a solution to the MMUCA WDP, condition (2) of definition 5.9 holds, and then we have that:

$$\mathcal{M}^{\ell}(p_i^{XOR}) = 1 - \sum_{j} |J_B^{*-1}(t_{ij})| \geq 0 \tag{6.62}$$

  Then, equation (6.28) is fulfilled.

- P_{SCO} places. We know that equation (6.27) must hold. Then, we have that for all $c_{ijk} \in P_{SCO}$:

$$\mathcal{M}^\ell(c_{ijk}) = \mathcal{M}_0(c_{ijk}) + \sum_{m=1}^{\ell} \left(E(J^*(m), c_{ijk}) - E(c_{ijk}, J^*(m))\right)$$

  that considering the mapping of section 6.1.3 becomes:

$$\begin{aligned}\mathcal{M}^\ell(c_{ijk}) = 0 + \sum_{m=1}^{\ell} \left(E(J^*(m), c_{ijk}) - E(c_{ijk}, J^*(m))\right) = \\ = \sum_{m=1}^{\ell_B} E(B^*(m), c_{ijk}) - \sum_{s=1}^{\ell_\Sigma} E(c_{ijk}, \Sigma^*(s)) \qquad (6.63)\\ = |B^{*-1}(t_{ij})| \cdot \mathcal{D}(t_{ijk}) - |\Sigma^{*-1}(t_{ijk})| \qquad (6.64)\end{aligned}$$

  Equation (6.63) results from considering that only bid transitions have output places in c_{ijk}, and that only SCO transitions have input places in c_{ijk} (see figure 6.4). Equation (6.64) follows from the fact that t_{ij} is the only input transition to c_{ijk} and that t_{ijk} is the only output transition of c_{ijk}. Hence, from condition (1) of definition 5.9, we have the following final marking:

$$\mathcal{M}^\ell(c_{ijk}) = \begin{cases} 0 & t_{ijk} \notin \Sigma^* \\ \mathcal{D}(t_{ijk}) - \mathcal{D}(t_{ijk}) & t_{ijk} \in \Sigma^* \end{cases}$$

  Then $\mathcal{M}^\ell(c_{ijk}) = 0$ for all $c_{ijk} \in P_{SCO}$.

- P_G places. Equation (6.26) must hold. Analogously to the previous cases, we write for all $p_g \in P_G$:

$$\begin{aligned}\mathcal{M}^\ell(p_g) = \mathcal{M}_0(p_g) + \sum_{m=1}^{\ell} \left(E(J^*(m), p_g) - E(p_g, J^*(m))\right) = \\ = \mathcal{U}_{in}(p_g) + \sum_{m=1}^{\ell_\Sigma} \left(E(\Sigma^*(m), p_g) - E(p_g, \Sigma^*(m))\right) = \qquad (6.65)\\ = \mathcal{U}_{in}(p_g) + \sum_{m=1}^{\ell_\Sigma} \left(\mathcal{O}_{\Sigma(m)}(p_g) - \mathcal{I}_{\Sigma(m)}(p_g)\right) \qquad (6.66)\end{aligned}$$

  Equation (6.65) follows from the fact that the only transitions that can add/remove tokens to/from places in P_G are the SCO transitions (see figure 6.4). Equation (6.66) substitutes the SCO transitions input/output arc weights for the input/output multisets of the corresponding SCOs. Following condition (4) of definition 5.9, we have that:

$$\mathcal{M}^\ell(p_g) = \mathcal{U}_{in}(p_g) + \sum_{m=1}^{\ell_\Sigma} \left(\mathcal{O}_{\Sigma(m)}(p_g) - \mathcal{I}_{\Sigma(m)}(p_g)\right) \geq \mathcal{U}_{out}(p_g) \qquad (6.67)$$

Next, we show that all the transitions in J^* are enabled.

- The transitions in J^*_B are trivially enabled, because:
  (1) at most one of the transitions outgoing from an *XOR place* can fire (according to equation (6.62)); and
  (2) the only token required to fire such a transition is present in the initial marking ($\mathcal{M}_0(p_i^{XOR}) = 1$ according to equation (6.23)).
- In order to have the transitions in Σ^* enabled as well, it must happen that:

$$\mathcal{M}^{m-1}(p) \geq E(p, J^*(m)) \qquad \forall m \in [1, \ell], \forall p \in P_G \cup P_{sco} \tag{6.68}$$

Recall that the XOR places are neither input nor output of the SCO transitions. Then, the only places modified by transitions in T_{SCO} are P_G and P_{sco}:

(1) P_{sco} places. Observe that only bid transitions can add tokens into the SCO places, and bid transitions are fired before the SCO transitions[10]. We also know that if a SCO transition t_{ijk} is in Σ^*, then the corresponding bid transitions t_{ij} is in B^* (equation (6.56)). Then, t_{ij} has added $\mathcal{D}_{ij}(t_{ijk})$ into the c_{ijk} places before any of the transitions in Σ^* has fired. As a consequence, transition t_{ijk} has available in place c_{ijk} the tokens to be fired at most $\mathcal{D}_{ij}(t_{ijk})$ times.

(2) P_G places. We write the enabling condition at the generic step m:

$$\mathcal{M}^{m-1}(p_g) \geq E(p_g, \Sigma^*(m)) \qquad \forall m \in [1, \ell_\Sigma], \forall p_g \in P_G \tag{6.69}$$

Analogously to what we have done in equation (6.66), we substitute the input multiset of the SCO for the input arc weights of the corresponding SCO transition:

$$\mathcal{M}^{m-1}(p_g) \geq \mathcal{I}_{\Sigma(m)}(p_g) \qquad \forall m \in [1, \ell_\Sigma], \forall p_g \in P_G \tag{6.70}$$

That is fulfilled at each step because of condition (3) of definition 5.9.

Finally, we have to prove that there is no other solution with a higher associated cost. Notice that, as shown in equation (6.54), the cost maximised in the CMWOSP is the auctioneer revenue. Then, say per absurd there exists another solution J' to the CMWOSP with a cost c' higher than the revenue associated to the corresponding MMUCA WDP. For the $\Rightarrow$) side of the demonstration, this would be a solution to the corresponding MMUCA WDP as well, since it has a higher revenue. This is impossible for the optimality of the solution to the MMUCA WDP.

$\square$

Summarising, each firing sequence J^* solution to the CMWOSP can be transformed into an optimal solution sequence of the MMUCA WDP. This can be done simply by removing from J^* the *bid transitions* (T_B). The obtained subsequence is a solution to the MMUCA WDP.

[10]Recall that J^* is a concatenation of J^*_B and Σ^*.

6.1.5 Solving the MMUCA WDP with IP

Thanks to theorem 6.1 we can exploit all the results proved for WPTN and the CMWOSP in section 4.7.1. In that section we showed that if a WPTN is acyclic, any CMWOSP on it can be efficiently solved by means of IP (see corollary 4.1).

In this section we explicitly present the IP formulation of the MMUCA WDP when the corresponding mixed auction net is acyclic.

The mathematical model is built according to the following rules:

(1) there are n *good places*, indexed with $g \in \{1, 2, \ldots, n\}$ (for each good $g \in G$)

(2) there are l *XOR places*, indexed with $i \in \{1, 2, \ldots, l\}$ (for each bidder i)

(3) the *bid transitions* t_{ij} are indexed with $i \in \{1, 2, \ldots, l\}, j \in \{1, 2, \ldots, m_i\}$ (for each bid j of each bidder i)

(4) the *SCO transitions* t_{ijk} are indexed with:

$$i \in \{1, 2, \ldots, l\} \tag{6.71}$$
$$j \in \{1, 2, \ldots, m_i\} \tag{6.72}$$
$$k \in \{1, 2, \ldots, f_{ij}\} \tag{6.73}$$

(for each SCO k of each bid j of each bidder i).

(5) $x_{ijk} \in \mathbb{N}$ is an integer decision variable (for each SCO transition t_{ijk}) taking on value w if SCO labelled by ijk is present w times in the optimal firing sequence. Namely, the SCO is used w times.

(6) $x_{ij} \in \{0, 1\}$ is a binary decision variable (for each *bid transition* t_{ij}), taking on value 1 if transition t_{ij} is in the optimal firing sequence.

With this in mind, the CMWOSP can be expressed by the following integer programming:

$$\begin{cases} \max \sum\limits_{ij} x_{ij} \cdot C(t_{ij}) & \\ \mathcal{M}_0(p_g) + \sum\limits_{ijk} x_{ijk} \cdot (E(t_{ijk}, p_g) - E(p_g, t_{ijk})) \geq \mathcal{U}_{out}(p_g) & \forall p_g \in P_G \\ 0 + x_{ij}E(t_{ij}, c_{ijk}) - x_{ijk}E(c_{ijk}, t_{ijk}) = 0 & \forall c_{ijk} \in P_{SCO} \\ 1 - \sum\limits_{j} x_{ij}E(p_i^{XOR}, t_{ij}) \geq 0 & \forall p_i^{XOR} \in P_{XOR} \end{cases}$$

The first equation maximises the cost associated to the optimal firing sequence, the second, third and fourth inequalities correspond to equations (6.26), (6.27), and (6.28) respectively.

Then, considering the mapping proposed in definition 6.1, this IP turns into:

$$\begin{cases} \max \sum_{ij} x_{ij} \cdot p_{ij} & \\ \mathcal{U}_{in}(g) + \sum_{ijk} x_{ijk} \cdot (\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)) \geq \mathcal{U}_{out}(p_g) & \forall g \in G \\ x_{ijk} = x_{ij}\mathcal{D}(t_{ijk}) & \forall ijk \\ 1 - \sum_{j} x_{ij} \geq 0 & \forall i \end{cases} \tag{6.74}$$

Finally, setting $a_{ijkg} = \mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)$, $u_g^{in} = \mathcal{U}_{in}(g)$ and $u_g^{out} = \mathcal{U}_{out}(g)$, we have:

$$\begin{cases} \max \sum_{ij} x_{ij} \cdot p_{ij} & \\ u_g^{in} + \sum_{ijk} x_{ijk} a_{ijkg} \geq u_g^{out} & \forall g \\ x_{ijk} = x_{ij}\mathcal{D}(t_{ijk}) & \forall ijk \\ 1 - \sum_{j} x_{ij} \geq 0 & \forall i \end{cases} \tag{6.75}$$

The interpretation of the model above is rather intuitive. The first equation maximises the auctioneer revenue. The second one ensures that at least as many goods as required by the auctioneer are produced at the end of the production process. The third equation enforces that the semantics of atomic bids is selected, i.e. all the SCOs with the corresponding multiplicity are selected or none of then. The fourth one ensures that the semantics of complex bids is fulfilled, i.e. that at most one atomic bid per bidder is selected.

In appendix A.1 we present this model encoded in the OPL language (see section 2.1.2 and (Van Hentenryck, 1999)).

Notice that the solution to the IP above is represented by the value assigned to decision variables x_{ijk} and x_{ij}. Recall that in such a solution the information about the order in which the SCOs must be performed is not included. However, according to corollary 4.1, this information can be easily extracted by the solution to the IP since the Mixed Auction Net is acyclic.

Problem Size

Next, we assess the number of decision variables and constraints required by the above IP model:

- for each bid transition t_{ij}, corresponding to bid Bid_{ij}, we create a binary decision variable x_{ij}, to total $|B|$ binary decision variables; and
- for each transition $t_{ijk} \in T$ (corresponding to a SCO), we create an *integer* decision variable, for a total of $|T|$ integer decision variables.

Then, the total number of decision variables is $|B| + |T|$. Finally, we assess the number of required constraints:

- for each good $g \in G$ we create a constraint, for a total of $|G|$ constraints;
- for each transition $t_{ijk} \in T$, we create a constraint, for a total of $|T|$ constraints; and
- for each bidder $i \in L$ we create a constraint.

The total number of constraints is then $|G| + |T| + |L|$.

6.1.6 Advantages of the mapping to CMWOSP

Before going on, we aim at highlighting the advantages brought about by the mapping of the MMUCA WDP to WPTNs. In particular such a mapping allows to import all the PTNs tools and properties presented in the literature to analyse structural and behavioural properties of the emerging supply chain. Some examples of application are listed in what follows.

(1) One can very efficiently solve the underlying IP when the supply chain is acyclic; this is obtained exploiting an important PTN analysis tool, the state equation.

(2) One may be interested in maintaining under a certain threshold the level of resources present in each place (for instance, for inventory capacity constraints). This can be mapped to a well known behavioural property of PTN, called boundedness (Murata, 1989).

(3) Thanks to the very appealing and intuitive WPTN graphical representation, we can compactly encode and visualise the search space associated to the MMUCA WDP. This is obtained thanks to the the fact that the semantics of transitions on PTN naturally accommodates for the representation of SCOs.

(4) Once obtained a solution sequence to the MMUCA WDP, one can visualise it by means of a token game showing the evolution of the supply chain at any step of the SCO sequence (as we did in table 4.6).

(5) One can graphically visualise the MMUCA WDP problem. This provides a very helpful guidance in obtaining insights about such problem. For instance, by visualising the MMUCA WDP by means of WPTN, one can incorporate new bidding language constructs with a minimum effort. For instance, consider the following example.

Example 6.3. We explained that switching to the *OR* language instead of the *XOR* bidding language is as simple as removing the *XOR* place from figure 6.4 (as done in figure 6.3). However, there is another widely employed bidding language that is very compact and human readable. Is is called the *XOR-of-OR* bidding language (refer to section 3.2.2). Such a language is such that any XOR combination of OR combinations of atomic bids can be selected. For instance, the bid:

$$((a, 1) \textit{ OR } (a, 1) \textit{ OR } (a, 1) \textit{ OR } (a, 1) \textit{ OR } (a, 1)) \textit{ XOR } (b, 2) \tag{6.76}$$

means that an auctioneer can select from 0 to 5 copies of the atomic bid $(a, 1)$ or (exclusive) one copy of the atomic bid $(b, 2)$.

In figure 6.5, we graphically show how to incorporate the *XOR-of-OR* bidding language. In figure we depict the following bid:

$$(\text{BID}(1't_{ij1} + 3't_{ij2} + 2't_{ij3}, -20) \textit{ OR} \tag{6.77}$$
$$\text{BID}(1't_{ij'1} + 1't_{ij'2}, -10)) \textit{ XOR} \tag{6.78}$$
$$\text{BID}(1't_{ij''1}, -2) \tag{6.79}$$

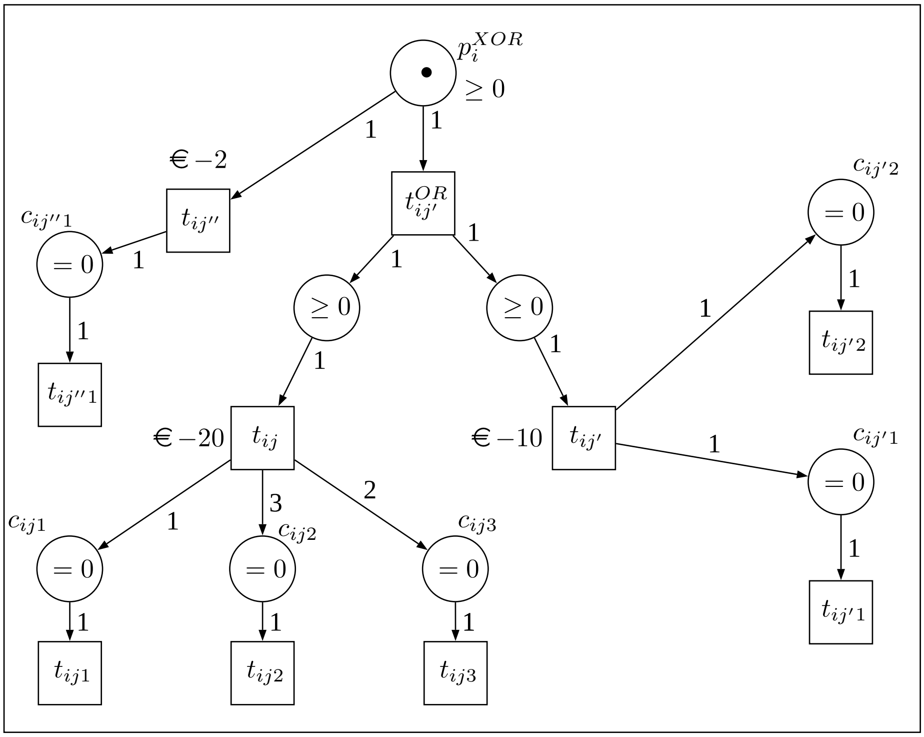


Figure 6.5: XOR-of-OR of atomic bids

The reader can check that this topology allows either firing $t_{ij''}$ or (exclusive) any of the t_{ij} and $t_{ij'}$ if the final contraints represented by inequalities in places in figure 6.5 are fulfilled.

$\square$

Notice that in this dissertation we only exploit directly advantages (1) and (3), and we envisage a promising path for future developments the study of all the implications connected with advantage (2),(4), and (5). We did not deepen into considerations connected with the study of behavioural and structural properties of the resulting supply chain. Nevertheless, by means of the mapping to WPTNs, we provide all the theoretical and practical tools to deal with such a study.

6.2 Solving the WDP on Cyclic Mixed Auction Nets

So far, we have not been concerned about whether a Mixed Auction Net is cyclic or not. Is it a reasonable hypothesis considering that a mixed auction net does not contain any cycle? The answer is that it depends. One could see a market as a big production cycle. However, when we consider local production processes, one could think that it is possible to avoid considering cycles in the topology. Unfortunately, this is not always the case. Even locally, production cycles are often characterised by cycles. Moreover, we will see that, in our semantics, cycles are required to represent shared resources or resources that can be employed more than once. This is the case, for instance, of a piece of software or of a tool that is not *consumed* but *used*. That is, at the end of the supply chain operation the resource is still present, but the operation cannot take place without it. With the purpose of clarifying this concept we slightly modify the example of figure 6.2.

Example 6.4. We recall that in example 6.2 five bidders participate in a MMUCA. We modify bid bid_3 introducing the fact that a bidder needs a machine MC to perform the hydrolysis operation. Bid bid_3, which stood for a bid on the hydrolysis process for €8, namely:

$$bid_3 = \text{BID}(1'(2'H_2O, 1'O_2 + 2'H_2), -8) \tag{6.80}$$

turns now into:

$$bid_3^* = \text{BID}(1'(2'H_2O + 1'MC, 1'O_2 + 2'H_2 + 1'MC), -3) \tag{6.81}$$

Notice that the bidder only requires the MC machine to run the hydrolysis process, and it will release it afterwards. Obviously, we have to include a bid that offers machine MC as well. This is bid bid_6:

$$bid_6 = \text{BID}(1'(\emptyset, 1'MC), -5) \tag{6.82}$$

The new configuration of the Mixed Auction Net substituting bid_3^* with bid_3 is shown in figure 6.6[11].

[11]In the figure we have omitted all the *XOR places*, *bid transitions*, and *SCO places* for the sake of comprehension.

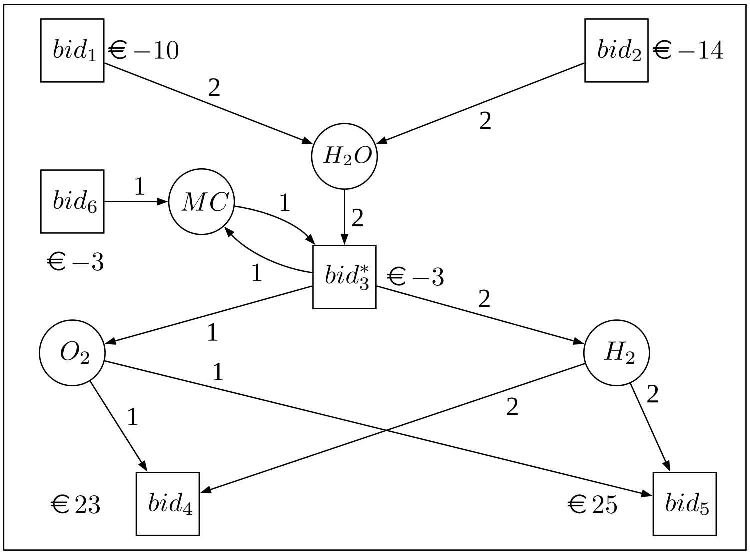


Figure 6.6: Example of a MMUCA in form of WPTN.

We can think about other types of resources that have this type of behaviour, as for instance an oven, a piece of software, a consultant, and so on. Those type of resources are not consumed, and eventually can be shared. In fact, we can see in figure 6.6 that transition bid_3^* requires the MC machine, and that after using it, the machine is still available (and could eventually be employed by another supply chain operation). Generalising, we can model *resource usage*, namely the machinery that production processes require.

Before explaining how to solve this new problem, we would like to show why the IP introduced in section 6.1.5 does not work in this case. We know from theorem 2.2 that it is not guaranteed to work since the Mixed Auction Net contains a cycle. Then, we write the IP in equations (6.75) as if the mixed auction net was acyclic to detect and show the problem. We can get rid of side constraints 2 and 3 in equation (6.75) since we consider that each bidder submits a bid over a single SCO. Then, we assign the binary decision variable x_i to bid bid_i. We also hypothesise that the auctioneer has no preferences on the number of goods available at the end of the production process ($\mathcal{U}_{out} = \mathcal{U}_{in} = \emptyset$). Then, we have:

$$\begin{cases} \max\ -10x_1 - 14x_2 - 3x_3 - 3x_6 + 23x_4 + 25x_5 & \\ 2x_1 + 2x_2 - 2x_3 \geq 0 & \text{place } H_2O \\ x_6 - x_3 + x_3 \geq 0 & \text{place } MC \\ x_3 - x_4 - x_5 \geq 0 & \text{place } O_2 \\ 2x_3 - 2x_4 - 2x_5 \geq 0 & \text{place } H_2 \end{cases} \tag{6.83}$$

If we simplify the equations above we obtain:

$$\begin{cases} \max\ -10x_1 - 14x_2 - 3x_3 - 3x_6 + 23x_4 + 25x_5 & \\ 2x_1 + 2x_2 - 2x_3 \geq 0 & \text{place } H_2O \\ x_6 \geq 0 & \text{place } MC \\ x_3 - x_4 - x_5 \geq 0 & \text{place } O_2 \end{cases} \tag{6.84}$$

the optimal solution is $x_5 = x_3 = x_1 = 1$ and the remaining x_i are 0. However, this solution is not valid! Let us apply the solution. At a first step, SCO bid_1 is used, providing two units of H_2O to the auctioneer. The following supply chain operation should be bid_3^*. However, it cannot be used without one MC, which is currently unavailable because we can only obtain it through bid_6. Thus, it is unfeasible to use bid_3^* because bid_6 is not part of the winning bid set.

□

Then, the solution to the IP *is not* a valid solution to the MMUCA WDP. This happens because the circularity of the net causes the elimination of the x_3 variable from the equation of place MC. This is not the only problem. Say that one is lucky and the IP solution matches the solution to the MMUCA WDP, he still should find the *ordered sequence* of operations. In this case the net is not acyclic and therefore a unique order among transitions cannot be ensured (as stated in corollary 4.1).

We end up this section with a remark that, though neither developed nor formally proved, can be useful in practice. Say that we compute the IP shown in section 6.1.5 for a cyclic mixed auction net (likewise in example 6.4). Say that we find a solution represented by $\mathbf{x}^*$ (the decision variables x_{ijk} with assigned a value). Say also that $S_{\mathbf{x}^*}$ is the subnet obtained by the mixed auction net by removing all the transitions not included in the solution $\mathbf{x}^*$ (i.e. removing the t_{ijk} such that $x_{ijk} = 0$). It is intuitive to think that if $S_{\mathbf{x}^*}$ is acyclic, then the solution is a valid solution sequence. The sketch of the demonstration follows. Recall that a necessary condition for a state to be reachable in a PTN is that $\mathbf{x}^*$ is a solution to the state equation (see section 2.3.2). However, since the hypothesis is that the mixed auction net is cyclic, we cannot guarantee that the state is reachable. Nevertheless, observe that x^* is a solution to the state equation associated to the subnet $S_{\mathbf{x}^*}$ as well. Then, if $S_{\mathbf{x}^*}$ is acyclic, the state is reachable in virtue of corollary 4.1. Then, $\mathbf{x}^*$ is a valid solution.

Although this observation may seem very powerful, in practise the situation described above is rather unusual. However, it should be taken into account.

6.2.1 Modifying the representation

By means of example 6.4 we showed that on cyclic nets the IP defined in section 6.2 cannot be applied. In the example we have also shown that the circularity of the net may cause an elimination of some decision variables. This elimination acts so that a check on the feasibility of a given solution is required. In order to overcome such problem, we modify the IP presented in section 6.2 in such a way that it is possible to check at each step of the SCO sequence whether enough resources are available to perform the selected SCO. In particular, we modify the way the SCOs are represented.

The new SCO encoding incorporates some information about the order in which the SCOs must be performed. In order to obtain this new representation, we build directly upon the definition of WDP (definition 5.10). However, notice that building upon the mixed auction net or on the CMWOSP one can obtain similar conclusions. The improved IP model resulting from the new representation is called *Direct Integer Programming* (DIP) solver.

According to definition 5.10, a solution to the WDP is a mapping Σ from the positions in the solution sequence to the atomic SCOs. Based on this, we define an IP model with the following decision variables: $x_{ijk}^m \in \{0,1\}$ is a binary decision variable that takes on value 1 if SCO t_{ijk} holds position m in the solution sequence, and 0 otherwise. These variables are the mathematical representation of something similar to Σ. In fact, we can associate to an element t_{ijk} a position m in a sequence if $x_{ijk}^m = 1$. However, we can have some *empty positions*. The problem is that prevents from having a solution such that $x_{ijk}^m = 0 \; \forall ijk$. This would leave position m empty. Then, we call a sequence with empty positions *partial sequence*. Obtaining the corresponding sequence from a partial sequence is as easy as removing the empty elements from the partial sequence. Thus, in what follows we will consider that Σ is a partial sequence, and if we want to retrieve the corresponding sequence we simply remove from Σ the empty positions.

Σ is obtained from the variables x_{ijk}^m in the following way:

$$\Sigma(m) = \begin{cases} t_{ijk} & x_{ijk}^m = 1 \\ \perp & otherwise \end{cases} \tag{6.85}$$

Obviously we do not know a priori how long the solution sequence will be. Then, we rely on the fact that if δ SCOs are submitted overall by all bidders, the length of the solution sequence will be at most δ (there can not be more SCOs in the sequence than the ones overall offered).

Observe that employing the binary decision variables above, we can state the following relationships[12]:

$$\mathcal{O}_{\Sigma(m)}(g) = \sum_{ijk} x_{ijk}^m \mathcal{O}_{ijk}(g) \qquad \forall g \tag{6.86}$$

$$\mathcal{I}_{\Sigma(m)}(g) = \sum_{ijk} x_{ijk}^m \mathcal{I}_{ijk}(g) \qquad \forall g \tag{6.87}$$

$$\mathcal{M}^m(g) = \mathcal{M}^{m-1}(g) + \mathcal{O}_{\Sigma(m)}(g) - \mathcal{I}_{\Sigma(m)}(g) \qquad \forall m, g \tag{6.88}$$

$$\mathcal{M}^m(g) = \mathcal{M}^{m-1}(g) + \sum_{ijk} x_{ijk}^m (\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)) \qquad \forall m, g \tag{6.89}$$

Equation (6.89) can be expanded into the following equation by making explicit its

[12]We anticipate that the following constraints must be added to ensure that Σ is a *function*:

$$\sum_{ijk} x_{ijk}^m \leq 1$$

recursive structure:

$$\mathcal{M}^m(g) = \sum_{l=1}^{m} \sum_{ijk} x_{ijk}^l (\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)) \qquad \forall m, g \tag{6.90}$$

6.2.2 The general IP formulation

We now show how to map the WDP in definition 5.10 into integer programming (IP). Therefore, the issue is to decide for each SCO whether it is selected for the solution sequence, and if so, to choose its position in the solution sequence. Thus, we define a set of binary decision variables $x_{ijk}^m \in \{0, 1\}$, where x_{ijk}^m takes on value 1 if the SCO t_{ijk} is selected at the m-th position of the solution sequence ($t_{ijk} = \Sigma(m)$), and 0 otherwise. Here and in what follows:

- m always ranges from 1 to δ, the maximum length of the solution sequence;
- i ranges over all bidders;
- for each bidder i, j ranges from 1 to the number of atomic bids submitted by i;
- for each atomic bid j of bidder i, k ranges from 1 to the number of SCO in that atomic bid;
- g ranges over all goods.

We also introduce several sets of auxiliary binary decision variables:

- $x_{ijk} \in \mathbb{N}$ is an integer decision variables that takes on value w iff transition t_{ijk} is present anywhere in the sequence w times ($|\Sigma^{-1}(t_{ijk})| = w$);
- $x_{ij} \in \{0, 1\}$ takes on value 1 iff any of the SCOs in the jth atomic bid of bidder i are selected. Equivalently, x_{ij} takes on value 1 iff bid Bid_{ij} is selected.

In what follows, we define the set of constraints that the solution sequence must fulfil:

(1) We enforce the constraints expressed by condition (1) of definition 5.9. Thus, if bid Bid_{ij} is selected, all the SCOs t_{ijk} in that bid must be selected exactly $\mathcal{D}_{ij}(t_{ijk})$ times. In other words, if bid Bid_{ij} is selected, all the SCOs in it must be selected with the required multiplicity. Formally,

$$x_{ij} \cdot \mathcal{D}_{ij}(t_{ijk}) = \sum_m x_{ijk}^m \quad (\forall ijk) \tag{6.91}$$

(2) We enforce that the atomic bids submitted by each bidder are exclusive (XOR). This amounts to satisfying the following constraints (cf. condition (2) of Definition 5.9):

$$\sum_j x_{ij} \leq 1 \quad (\forall i) \tag{6.92}$$

Observe that in the case of the *OR* bidding language we simply have to remove this constraint.

(3) We also impose that at most one SCO is selected at each position of the sequence:

$$\sum_{ijk} x^m_{ijk} \leq 1 \quad (\forall m) \tag{6.93}$$

This equation encodes the hypothesis of no simultaneous firings and enforces that the Σ built with the x^m_{ijk} is a function, i.e. it does not have two images associated the same element (cfr. equation (6.85))

(4) Next, we capture condition (3) of Definition 5.9: enough goods must be available at step m to perform the next SCO (cf. equation (5.15)). We recall that this maps to the following condition:

$$\mathcal{M}^{m-1}(g) \geq \mathcal{I}^m(g) \qquad\qquad \forall g$$

which is translated, according to equations (6.87) and (6.90), into:

$$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} \sum_{ijk} x^l_{ijk} \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \sum_{ijk} x^m_{ijk} \cdot \mathcal{I}_{ijk}(g) \tag{6.94}$$
$$\forall g, \forall m$$

(5) And finally, after having performed all the selected SCOs, the set of goods held by the auctioneer must be a superset of the final goods $\mathcal{U}_{out}$ (cf. condition (4) of Definition 5.9):

$$\mathcal{M}^{\delta}(g) \geq \mathcal{U}_{out}(g) \qquad \forall g$$

that turns into

$$\mathcal{U}_0(g) + \sum_{m=0}^{\delta} \sum_{ijk} x^m_{ijk} \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \mathcal{U}_{out}(g) \qquad \forall g \tag{6.95}$$

Now solving the WDP for MMUCAs with XOR-bids amounts to solving the following integer program:

$$\max \sum_{ij} x_{ij} \cdot p_{ij} \quad \text{subject to constraints (6.91)– (6.95)} \tag{6.96}$$

In table 6.1, we summarise the DIP formulation employing the same notation as the IP in equation (6.75), with the exception of the symbol I_{ijkg}, that stands for $\mathcal{I}_{ijk}(g)$.

Finally, a valid solution according to definition 5.10 is obtained from the solution of the IP by making transition t_{ijk} the m-th element of the partial sequence Σ iff $x^m_{ijk} = 1$, and then removing the empty positions. In appendix A.2 we present this model encoded in the OPL language (see section 2.1.2 and (Van Hentenryck, 1999)).

Observe that our proposed implementation can easily be amended so as to directly encode the constraints imposed by language constructs other than the XOR-operator. This would remove the need for translating into the XOR-language first and thereby greatly improve efficiency.

(a)	$\forall ijk$	$x_{ijk} = \sum_m x^m_{ijk}$
(b)	$\forall ijk$	$x_{ijk} = x_{ij} \cdot \mathcal{D}_{ij}(t_{ijk}) \quad \forall ijk$
(c)	$\forall i$	$\sum_j x_{ij} \leq 1$
(d)	$\forall m$	$\sum_{ijk} x^m_{ijk} \leq 1$
(e)	$\forall g$	$u^{in}_g + \sum_m \sum_{ijk} x^m_{ijk} \cdot a_{ijkg} \geq u^{out}_g$
(f)	$\forall g, \forall m$	$u^{in}_g + \sum_{l=0}^{m-1} \sum_{ijk} x^l_{ijk} \cdot a_{ijkg} \geq \sum_{ijk} x^m_{ijk} \cdot I_{ijkg}$
(g)		$\max \sum_{ij} x_{ij} \cdot p_{ij}$

Table 6.1: Resume of the IP formulation of solver DIP.

Problem Size

The number of decision variables in the above integer program is of the order of $O(|T| \cdot \delta)$ (corresponding to x^m_{ijk}). More in details, we create a binary decision variable x_{ij} for each bid $Bid_{ij} \in B$, for a total of $|B|$ binary decision variables. Then, we create a decision variable x^m_{ijk} for each SCO $t_{ijk} \in T$ and for each position m in the solution sequence, for a total of $|T| \cdot \ell$ binary decision variables. Assuming, in the general case, that the maximum length of the solution sequence is $\ell = \delta$, then we have $|T| \cdot \delta = |T| \cdot \sum_{ij} |\mathcal{D}_{ij}|$ decision variables. Then, we create a total of

$$|B| + |T|(1 + \delta) \in O(|T| \cdot \delta)$$

decision variables. With a similar process, we compute the total number of constraints, that is:

$$|T| + |L| + \delta + |G|\delta + |G| \in O(|G|\delta) \tag{6.97}$$

Example 6.5. For the problem presented in figure 6.4, we have the following data:

$$\begin{aligned} |L| &= 1 & |B| &= 2 \\ |G| &= 9 & |T| &= 5 \\ \delta &= 8 \end{aligned}$$

Then, in the case of the IP in section 6.1.5 the number of decision variables created is 7, and the number of constraints is 9+5+1=15. In the case of the IP presented in this section, we have 45 decision variables and 5+1+8+56+9=79 constraints.

6.3 Computational Complexity

In his master thesis (Ottens, 2007), Ottens provides a detailed proof of the NP-completeness of the decision problem underlying the MMUCA WDP. We briefly recall the employed argumentation in what follows.

The (decision problem underlying the) WDP for standard combinatorial auctions is known to be NP-complete, with respect to the number of goods(Rothkopf et al., 1998). NP-hardness can, for instance, be shown by reduction from the well-known SET PACKING problem. As our mixed auction model generalises standard combinatorial auctions, winner determination remains NP-hard also here. NP-membership (and thereby NP-completeness) of the problem of checking whether there exists a solution exceeding a given revenue (for finite bids) follows from the fact that a candidate solution provided by an oracle can always be verified in polynomial time. That is, despite of the generalisations we have introduced, the computational complexity of the WDP does not increase, at least not with respect to the polynomial hierarchy.

6.4 Conclusions

In this chapter we dealt with the problem of solving the MMUCA WDP, as defined in chapter 5. With this aim, we provided a mapping of the MMUCA WDP to a CMWOSP on the Mixed Auction Net. Some benefits stemed from this mapping. Firstly, since the mixed auction net is a WPTN, it provides a very powerful theoretical framework for analysing the MMUCA WDP computational, structural and behavioural properties. Secondly, consequence of the first benefit, we provide an efficient mapping of the MMUCA WDP to ILP for acyclic mixed auction nets. Thirdly, since WPTNs have associated a very appealing graphical representation, they provide a graphical framework to compactly represent both the search space and the solutions associated to the MMUCA WDP. This is due to the perfect matching between the semantics of transitions and the semantics of SCOs. We recall that we focus on the computational advantages provided by the mapping to CMWOSP, and leave out for future developments the analysis of the structural and behavioural properties of the solutions to the MMUCA WDP. However, we remark that the mapping to CMWOSP provides the needed theoretical and practical tools to perform such analysis.

Next, we show that the hypothesis that the mixed auction net is acyclic sometimes may not hold. In such a case, the ILP based on the CMWOSP cannot be employed.

Hence, we provide a general IP solver, the *Direct Integer Programming* (DIP) solver, that directly builds upon the definition of the MMUCA WDP. This solver allows to solve the MMUCA WDP on any supply chain network topology. However, it has the disadvantage to be computationally more costly. In fact, it requires more decision variables to be encoded.

Notice that the mixed auction net provides a framework to *a priori* assess the solver to employ, either the CMWOSP-based, if no cycles are present in the mixed auction net, or the DIP otherwise. With this tool at hand, one can build computationally efficient MMUCAs. For instance, one approach could be constraining the participants to an MMUCA to bid on sets of SCOs that do not form cycles. This would ensure the absence of cycles in the corresponding *Mixed Auction Net*, thus allowing the use of the CMWOSP-based solver. However, as motivated by some examples provided in this chapter, sometimes it is not possible to avoid cycles in the *Mixed Auction Net*.

Recent contributions on computationally efficient WDP solvers for different auction types (namely, (Lehmann et al., 2006) for CAs and (Engel et al., 2006) for multi-attribute double auctions) agree on and defend that a careful, formal analysis of the structure of WDPs can provide guidance for developing efficient winner determination solvers. Along the lines of these works, in the next chapter, we propose an IP for MMUCAs that dramatically improves the computational efficiency of the DIP solver.

Chapter 7

Connected Component-based Solver

In the previous chapter we presented DIP, an ILP that can solve MMUCA WDPs on any network topology. Then, in section 6.2.2 we showed that DIP requires O($|T|\delta$) decision variables to be represented. This means that the associated search space is very large. In this section we reduce the search space associated the MMUCA WDP.

Recent contributions on computationally efficient WDP solvers for different auction types (namely, (Lehmann et al., 2006) for CAs, (Engel et al., 2006) for multi-attribute double auctions, as well as our contribution in section 4 for MUCRAtR) agree on and defend that a careful, formal analysis of the structure of WDPs can provide guidance for developing efficient winner determination solvers. Along the lines of these works, in this chapter we present a technique to reduce the search space associated to the MMUCA WDP. This will result in an ILP formulation for MMUCA WDPs that dramatically improves the computational efficiency of the DIP solver presented in section 6.2.2.

At this aim, we found our analysis on observing the structure of the WDP that results after establishing *dependence relationships* among transformations. For instance, in the example of *Grandma & co* (depicted in figure 1.1) the *Baking* SCO potentially depends on the *Make Dough* SCO, since the output provided by *Make Dough* may be required to perform *Baking*. The analysis of the WDP based on dependency relationships helps design an IP that *a priori* establishes *when* to *use* each transformation. Therefore, the search space reduction is achieved by enforcing MMUCA solutions to fulfil a template. The template reduces the possible orderings among transformations without losing solutions.

This chapter is organised as follows. In section 7.1 we explain the intuitions underlying the improvement we propose by means of examples, and in the remaining sections we develop a rigorous description of those intuitions. In section 7.2 we introduce the solution template allowing a reduction in the search space along with some mathematical tools required in the chapter. In section 7.3 we present the *Connected Component Integer Programming* (CCIP) solver, an ILP formulation improving the DIP solver by exploiting the solution template. Then, in section 7.4, we prove that the search space reduction imposed by the solution templates does not cause a loss of solutions. Finally, in section 7.5 we draw some conclusions.

7.1 Motivation and Example

In this chapter we introduce a technique to reduce the search space associated to the solution of MMUCA WDPs. Then, we apply this new representation to encode a new ILP solver for MMUCA, the CCIP. CCIP substantially reduces the number of variables and constraints used by DIP.

The search space reduction is obtained by observing that DIP produces several *equivalent* solutions. We regard two solutions as *equivalent* if they select the same bids. As a consequence, equivalent solutions contain the same supply chain operations (SCOs) (even if arranged in different order), and they have associated the same cost. In what follows we provide the rationale to achieve such reduction and to found CCIP.

Example 7.1. Recall from section 5.4.2 that in a MMUCA WDP the input is composed of: (1) the initially available goods ($\mathcal{U}_{in} \in \mathbb{N}^G$); (2) the finally required goods ($\mathcal{U}_{out} \in \mathbb{N}^G$); and (3) a set of bids in the XOR bidding language ($Bid_{ij} = (\mathcal{D}_{ij}, p_{ij})$). Hence, let us consider an MMUCA WDP scenario characterised as follows:

- $\mathcal{U}_{in} = \emptyset$ and $\mathcal{U}_{out} = \emptyset$: no goods are initially available and no goods are required at the end of the auction.

- Eight bidders submit the eight bids showed in equations 7.1 to 7.8.

$$Bid_{11} = (3't_0 + 1't_1, -3\text{ USD}) \tag{7.1}$$
$$Bid_{21} = (2't_2, 9\text{ USD}) \tag{7.2}$$
$$Bid_{31} = (1't_3, -2\text{ USD}) \tag{7.3}$$
$$Bid_{41} = (1't_4, -1\text{ USD}) \tag{7.4}$$
$$Bid_{51} = (1't_5, -8\text{ USD}) \tag{7.5}$$
$$Bid_{61} = (2't_6 + 2't_7, -3\text{ USD}) \tag{7.6}$$
$$Bid_{71} = (1't_8, -12\text{ USD}) \tag{7.7}$$
$$Bid_{81} = (1't_9 + 2't_{10}, -4\text{ USD}) \tag{7.8}$$

We recall that $Bid_{ij} = (\alpha'_h t_h, p_{ij})$ means that bidder i offers α_h copies of SCO t_h ($\mathcal{D}_{ij}(t_h) = k$) at price p_{ij} in his j-th bid. For instance, bid Bid_{11} offers in a bundle (combinatorial bid) three units of t_0 and one unit of t_1 at a price of 3 USD[1]. More formally, $\mathcal{D}_{11} = \{3't_0 + 1't_1\}$.

Recall from section 5.4.2 that $\mathcal{D} = \uplus_{ij}\mathcal{D}_{ij}$ is the union of multisets submitted in each bid. For the bids in equations 7.1 to 7.8, we have:

$$\mathcal{D} = \{3't_0 + 1't_1 + 2't_2 + 1't_3 + 1't_4 + 1't_5 + 2't_6 + 2't_7 + 1't_8 + 1't_9 + 2't_{10}\} \tag{7.9}$$

[1]Recall that the negative sign means that a bidder is willing to be paid.

Recall also that the maximum length ℓ of the solution sequence will be at most equal to the overall number of atomic SCOs submitted, namely

$$\ell = \delta = \sum_{ij} |\mathcal{D}_{ij}| = 17$$

Finally, recall that T is the set of all the received SCOs (disregarding their multiplicity).

$$T = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\} \tag{7.10}$$

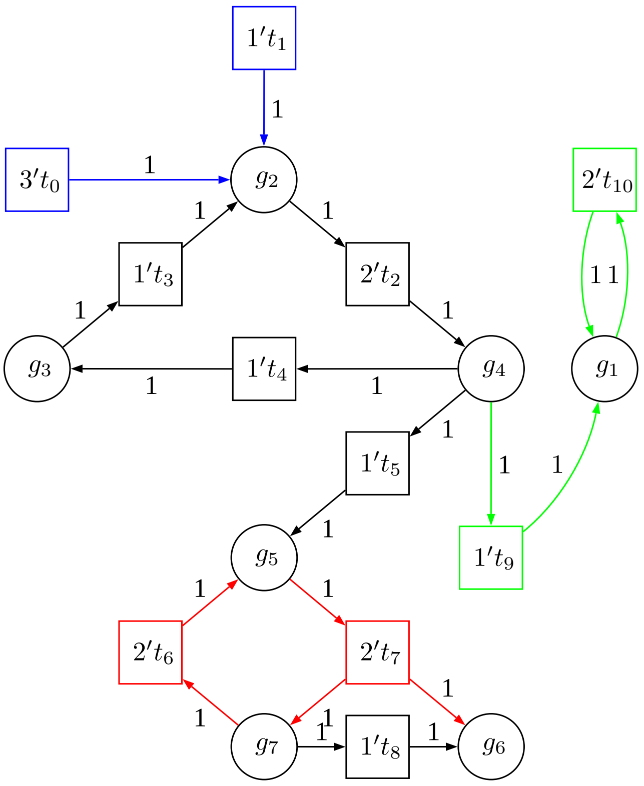


Figure 7.1: Graphical representation for the SCOs in bids in equations 7.1 to 7.8

In figure 7.1 we graphically represent SCOs $t_1, \ldots, t_{10}$ contained in the bids of equations 7.1 to 7.8. The formalism employed in the figure is similar to the one employed in chapter 6. Figure 7.1 represents a Petri Net Structure (PTNS) in which each transitions represents a SCO and each place a good. The input/output arcs from/to SCOs depict the input/output multisets of each SCO. We recall that the arc weights represent the input and output multiplicity of each SCO (for instance, according to figure 7.1, the input and output multisets of SCO t_7 are respectively $\mathcal{I}_{t_7} = \{g_5\}$ and $\mathcal{O}_{t_7} = \{g_7, g_6\}$). Notice that in our example every arc has weight 1.

Unlike the formalism employed in chapter 6, in the PTNS of figure 7.1 the information about complementarities among SCOs and the XOR relationships is omitted. Furthermore, we label each SCO with its multiplicity in each bid. For instance, $3't_0$ means that three units of t_0 have been submitted in a bid.

At this point consider that solver DIP solves the WDP with the input expressed by equations 7.1 to 7.8, and finds the solution sequence in table 7.1. The first row in table 7.1 represents a position within the solution sequence (the m index in variables x_{ijk}^m of section 6.2), whereas the second row shows the SCO assigned to the each position within the solution sequence. For instance, the fact that in the second row and second column we find SCO t_2 means that position 2 of the solution sequence is assigned to t_2 (in DIP this means that in the solution $x_{t_2}^2 = 1$).

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**Revenue**
Sequence 1	t_0	t_2			t_1	t_0	t_4	t_0	t_2					t_3				+3 USD

Table 7.1: Example of solution found by solver DIP.

Thus, according to definition 5.9 of section 5.4.2, the solution of table 7.1 corresponds to the following solution sequence:

$$\Sigma = \langle t_0, t_2, t_1, t_0, t_4, t_0, t_2, t_3 \rangle \tag{7.11}$$

Notice that the solution sequence Σ, according to what explained in section 6.2.1, is obtained by removing the *empty positions* in *Sequence 1* of table 7.1.

Accordingly, the winning bids are Bid_{11}, Bid_{21}, Bid_{31}, and Bid_{41}, and the revenue associated to this solution is $-3 + 9 - 2 - 1 = 3$. As the reader can check, this solution is valid, since the semantics of the bidding language is fulfilled, and at each step of the solution sequence there are enough input goods available to perform the corresponding SCO. Now consider all the solutions in table 7.2. These solutions are all valid and optimal (they have associated the same revenue) as much as the one in table 7.1. They are simply a rearrangement of the very same solution along different positions of the solution sequence, without modifying the relative order among them (i.e. they all represent the same Σ of equation 7.11).

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**Revenue**
Sequence 2	t_0	t_2	t_1	t_0	t_4	t_0	t_2	t_3										+3 USD
Sequence 3						t_0	t_2	t_1	t_0	t_4	t_0	t_2					t_3	+3 USD
Sequence 4								t_0	t_2			t_1	t_0	t_4	t_0	t_2	t_3	+3 USD
Sequence 5	t_0		t_2		t_1		t_0		t_4		t_0		t_2			t_3		+3 USD
Sequence 6										t_0	t_2	t_1	t_0	t_4	t_0	t_2	t_3	+3 USD

Table 7.2: Solutions equivalent to the solution in table 7.1 with same relative order.

Now consider the solutions in table 7.3. They are still valid solutions equivalent to *Sequence 1* in table 7.1, though not only the positions assigned to SCOs are different, but also the relative order among them has been altered. Although those solutions correspond to valid solution sequences different from the one in equation 7.11, it is easy to check that those solutions are still valid. Indeed, at each step of the solution sequence there are enough inputs to perform the corresponding SCOs. Hence, all the considered solutions are Pareto optimal, and equivalent among them.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**Revenue**
Sequence 7				t_0		t_1	t_0		t_2	t_0		t_2	t_4	t_3				+3 USD
Sequence 8	t_0	t_0	t_0	t_1	t_2	t_2	t_4	t_3										+3 USD

Table 7.3: Solutions equivalent to the solutions in table 7.1 with different order.

□

At this point, the reader is ready to understand what we mean by *equivalent solutions*. Solutions of solver DIP that select the same bids (and thus the same SCOs) have associated the same cost. Thus, we hypothesise that they are indistinguishable for an auctioneer.

Notice that, as shown in example 7.1, the search space of solver DIP contains a huge amount of equivalent solutions. Hence, in order to reduce the complexity of our problem, we aim at understanding why all those redundant solutions are found. As explained in section 6.2, the IP formulation of solver DIP is founded on the hypothesis that a SCO can hold any position within the solution sequence (recall that we create decision variables x^m_{ijk} for each position m and for each SCO $t_{ijk} \in T$), and we set the length of the solution sequence equal to the overall number of received SCOs, namely δ. Thus, in principle, each SCO can take one among δ available positions. This explains why all those equivalent solutions are contained in the DIP search space: a large number of equivalent rearrangements of SCOs within the solution sequence are allowed.

The fact that many equivalent solutions can be found implies a larger search space than needed, and thus an increased computational cost. Such computational cost is reflected in the number of decision variables employed for solver DIP. In the case of example 7.1, for instance, t_0 has the possibility to take on any of the 17 positions of the solution sequence. Hence, DIP requires 17 decision variables for t_0. Then, for all the SCOs it requires $\delta|T| = 11 * 17 = 187$ decision variables. If we manage to reduce the number of equivalent solutions contained in the search space, we cut down the number of decisions, and consequently the search space.

Then, the strategy we follow to reduce the search space consists in limiting the possible positions each SCO may take on in a solution sequence. In this way the number of feasible solutions is reduced. Obviously, if we limit the positions each SCO can take on we lose solutions as well. The main point here is *losing solutions that are equivalent to solutions that in turn are found*. For instance, an auctioneer is willing to lose all but one of the solutions in tables 7.1, 7.2, and 7.3 . If at least one is found, we assume that an auctioneer is not bothered by losing all the other equivalent solutions.

We assume that if two solutions are equivalent, from an auctioneer's point of view eliminating one of them from the space of feasible solutions does not constitute a problem. However, given a set of equivalent solutions, the auctioneer needs that at least one of them is included in the space of feasible solutions.

We employ a terminology related to equivalence classes in order to explain this concept. It is easy to verify that the relation *is equivalent to* on the set of DIP solutions is an equivalence relation (refer to section 2.4.1 for the theory underlying equivalence relations). In these terms, our goal consists in finding a solution template that:

- reduces the number of equivalent solutions contained in the search space, and
- ensures that at least one feasible solution for each equivalence class is found.

That is, we must ensure that no solution class is completely removed from the search space. Hereafter, with an abuse of terminology, we say that we *lose a solution class* when we lose a whole equivalence class of solutions.

In what follows we explain how to reduce the search space. We employ a function that reduces the possible positions any SCOs can take on in the solution sequence.

Example 7.2. Say that we constrain t_1 to hold only the first position in a solution sequence. All the solutions in tables 7.1, 7.2, and 7.3 are still valid if we push t_1 ahead in front of the sequence. For instance, the solution sequences in table 7.3 produces the equivalent solutions represented in table 7.4.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**Revenue**
Sequence 9	t_1			t_0			t_0		t_2	t_0		t_2	t_4	t_3				+3 USD
Sequence 10	t_1	t_0	t_0	t_0	t_2	t_2	t_4	t_3										+3 USD

Table 7.4: Solutions equivalent to the solutions in table 7.3 pushing t_1 ahead.

In general, every solution found by DIP to the considered problem can be reordered into a solution with t_1 in the first position. Then, we push t_2 in the first position of the solution sequences in table 7.3, and we obtain the sequences in table 7.5.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**Revenue**
Sequence 11	t_2			t_0			t_0		t_2	t_0		t_1	t_4	t_3				+3 USD
Sequence 12	t_2	t_0	t_0	t_0	t_1	t_2	t_4	t_3										+3 USD

Table 7.5: Solutions equivalent to the solutions in table 7.3 pushing t_2 ahead.

None of the sequences in table 7.5 is a valid solution to solver DIP since t_2 cannot operate without input goods (g_2). In this case, placing t_2 at the first position is not *safe*, since the SCOs that can provide it with input goods are not performed before it.

Then, all the solutions found by solver DIP can be reordered into solutions having t_1 at the first position without losing solution classes. Oppositely, not all the solutions found by DIP can be reordered into a solution having t_2 at the first position. Therefore, if we constrain t_2 to take on the first position, we lose solution classes.

Why it is possible to push ahead t_1 and not t_2? The reason is that t_2 may *depend* on other SCOs to be performed. In fact t_2 may need some inputs that in turn are produced by other SCOs (t_0 and t_1 in the case of t_2). Then, if we want provide a solution template that limits the positions that each SCO can hold without causing a loss of solutions, then we have to consider those dependencies among SCOs.

A SCO t' *depends* on t if any of the output goods of t is an input good of t'. In such a case, t' may need the output of t to operate. Hypothesising that t' *depends* on t and t does not depend on t' we have that:

- if in a solution t' comes before t, then the solution remains valid by moving t before t'.
- if in a solution t comes before t', then the solution may not be feasible anymore by moving t' before t.

Along this line, given two SCOs, we can differentiate three cases:

- t *depends* on t' and t' does not depend on t: $\begin{tikzcd} t' \bigcirc \arrow[r] & \bigcirc t \end{tikzcd}$
- t depends on t' and t' depends on t: $\begin{tikzcd} t' \bigcirc \arrow[r, leftrightarrow] & \bigcirc t \end{tikzcd}$

  That is, they are *mutually dependent*.
- otherwise (the case of no dependence at all): $\begin{tikzcd} t' \bigcirc & \bigcirc t \end{tikzcd}$

By analysing the dependencies above we can limit the positions each SCO can assume without losing solutions.

Example 7.3. Consider once more example 7.1, graphically depicted in figure 7.1. Notice that t_1 does not have any input good. Then, it does not depend on any SCO. Then, we can constrain SCO t_1 to hold the first position within the solution sequence (as in example 7.2): any solution with t_1 at a different position than the first one can be reordered into a solution in which t_1 is at the first position. We can assign only one position in a solution sequence since only one unit of t_1 is offered. Position 1 is *safe* to t_1. Then, we assign position 1 to[2] t_1.

Next, things are different with t_0. Recall that three units of t_0 are offered by bid Bid_{11}, and thus t_0 might appear up to three times within a solution sequence. Then, we cannot simply assign t_0 to position 2. Since t_0 can be performed three times, it needs at least three positions in a solution sequence. Then, we assign positions $2, 3$ and 4 of a solution sequence to t_0, as represented in table 7.6.

Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Solution Template	t_1	t_0	t_0	t_0	...	...	...	...	...								

Table 7.6: Assigning positions to t_0 within a solution sequence.

At this point we wonder whether we can carry on with t_2, t_3, and t_4 and so on. Unfortunately, we cannot. This is because t_2, t_3, and t_4 form a loop, i.e. they are *mutually dependent*. Observing carefully figure 7.1 we can say that:

- t_2, t_3, and t_4 depend on t_0 and t_1;

[2]In terms of decision variables for DIP this means that we are not generating $x_{t_1}^m$ for all the positions $m \in \{1, \ldots, 17\}$, but we generate only one decision variable $x_{t_1}^1$, since t_1 is allowed to hold only position 1.

- t_2, t_3, and t_4 do not dependent[3] on $t_5, \dots, t_{10}$;
- t_2, t_3, and t_4 are mutually dependent[4]; and
- $t_5, \dots, t_{10}$ dependends on t_2, t_3, and t_4.

Then, t_2, t_3, and t_4 must come before $t_5, \dots, t_{10}$ and after t_0 and t_1. However, we cannot establish an order among them since they are mutually dependent. Thus, we must consider all their possible orderings. For instance, we can assign to t_2, t_3, and t_4 positions $5, 6, 7$, and 8 (since two units of t_2 are available, we must assign two positions to t_2). Table 7.7 outlines a *template* of a solution built in this way.

Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Solution Template	t_1	t_0	t_0	t_0	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_5	t_9	t_{10}	t_{10}	t_6 t_7	t_6 t_7	t_6 t_7	t_6 t_7	t_8
#Variables	1	1	1	1	3	3	3	3	1	1	1	1	2	2	2	2	1

Table 7.7: Positions within the solution sequence assigned a-priori to SCOs.

Notice that we now need a decision variable for each of the elements in table 7.7 (as expressed in the last row of the table). As to t_2, t_3, and t_4, the possible choices can be encoded by the following variables:

$$\{x^5_{t_2}, x^6_{t_2}, x^7_{t_2}, x^8_{t_2}, x^5_{t_3}, x^6_{t_3}, x^7_{t_3}, x^8_{t_3}, x^5_{t_4}, x^6_{t_4}, x^7_{t_4}, x^8_{t_4}\} \tag{7.12}$$

where $x^5_{t_2} = 1$ means that t_2 is performed at the 5-th position.

Then, the total number of decision variables required to represent the problem amounts to:

$$1 + 1 + 1 + 1 + 3 \cdot 4 + 1 + 1 + 1 + 1 + 2 \cdot 4 + 1 = 29 \tag{7.13}$$

In contrast, table 7.8 illustrates the assignment of positions as required by DIP. DIP employs $11 * 17 = 187$ variables overall. Therefore, the difference when constraining SCOs to limited number of positions is very significant (29 versus 187 in the example).

To conclude, we have to detect the dependencies present in the structure induced by the SCOs and apply the process described above: we assign an a-priori limited number of positions within the solution sequence to each SCO (or group of SCOs).

In what follows, we formally analyse how we can extend the intuitions above to the general case in order to yield a new IP, the so called CCIP, by relying on the notion of dependence among transformations, and using it to constrain the positions at which a transformation can be used.

[3]No matter the ordering among t_2, t_3, and t_4, we can always assign to them positions before t_5 or t_9 without losing solutions.

[4]Notice that t_2, t_3, and t_4 lie on a cycle in the net. For this reason, each of them could contribute to provide goods to the inputs of the other.

Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Solution Template	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0	t_0
	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1	t_1
	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2	t_2
	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3	t_3
	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4	t_4
	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5	t_5
	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6	t_6
	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7	t_7
	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8	t_8
	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9	t_9
	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}	t_{10}
#Variables	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11

Table 7.8: Positions assigned a-priori without constraints.

7.2 SCO Dependencies and Solution Template

In this section we formally introduce a solution template that limits the possible positions each SCOs can take (like in table 7.7) without losing any solution class. With this purpose, firstly we formally introduce the concept of *dependency* among SCOs. Next, we introduce a function that constrain the SCOs to hold a limited number of positions within a solution sequence, that is a *solution template*.

But before that we would like to clarify the concept of *dependency*. The fact that an SCO t depends on another SCO t' does not enforce that t' must be forcedly executed before t. In fact this could happen. The fact that t depends on t' only means that it is always possible to change the relative order of t and t' bringing t' in front without losing solutions classes.

7.2.1 The SCO Dependency Graph (SDG)

In this section we formally capture the concept of *dependency* among SCOs. All the background knowledge required to understand this section is summarised in section 2.4.2.

An SCO dependency graph (SDG) is a graph that encodes the dependencies, in terms of precedence relationships, between the SCOs[5] in T. The SDG is a directed graph whose nodes stand for SCOs, and an edge from SCO t to SCO t' reflects that there exists a good that is both output of t and input to t'.

Example 7.4. The SDG associated to example 7.1 (see figure 7.1) is depicted in figure 7.2(b). For the sake of comprehension, we include a copy of figure 7.1 in figure 7.2(a).

Definition 7.1 (SCO Dependency Graph)**.** Given a set of bids in the XOR bidding language, such that T is the overall set of SCOs, the associated SCO Dependency Graph (SDG) is a graph $SDG = (V, E)$ such that:

[5]Recall that, given the input to a MMUCA WDP, T is the set of overall SCOs present in all bids. The T corresponding to the bid of example 7.1 is represented in equation 7.10.

- Each SCO is a vertex: $V = T$,
- A directed arc connects two SCOs t and t' iff there exists a good that is both output of t and input to t'. More formally,

$$(t,t') \in E \textit{ iff } O_t \cap \mathcal{I}_{t'} \neq \emptyset$$

An SDG may or may not contain cycles. However, we have to assume that the graph is cyclic in the general case. As explained in section 2.4.3, a (cyclic) graph defines a preorder $\lesssim$ over T. We denote this preorder as a pair $(T, \lesssim)$. The semantics of the preorder is that $t \lesssim t'$ iff a path exists between t and t'. As illustrated in section 2.4.3, a preorder allows the existence of pairs (t,t') such that $t \lesssim t'$ and $t' \lesssim t$.

Example 7.5. In the order defined by the graph in figure 7.2(b) we have that $t_1 \lesssim t_5$ and $t_2 \lesssim t_8$. However, considering that $t_2 \lesssim t_3$ and $t_3 \lesssim t_2$, t_2 and t_3 cannot be ordered among them.

Figures 7.2 (a) and (b) depict the PTN structure representing the SCOs of example 7.1 along with the corresponding SDG. We recall from chapter 2 that, given a set T equipped with a preorder $\lesssim$, we can define an equivalence relation $\sim$ on T as follows:

$$t \sim t' \textit{ iff } t \lesssim t' \textit{ and } t' \lesssim t \tag{7.14}$$

Example 7.6. Regarding the example of figure 7.2(b), the equivalence classes are:

$$[t_0] = \{t_0\} \tag{7.15}$$
$$[t_1] = \{t_1\} \tag{7.16}$$
$$[t_2] = \{t_2, t_3, t_4\} \tag{7.17}$$
$$[t_5] = \{t_5\} \tag{7.18}$$
$$[t_6] = \{t_6, t_7\} \tag{7.19}$$
$$[t_8] = \{t_8\} \tag{7.20}$$
$$[t_9] = \{t_9\} \tag{7.21}$$
$$[t_{10}] = \{t_{10}\} \tag{7.22}$$
$$\tag{7.23}$$

Recall also that it is possible to define a strict partial order over the quotient set $(T/_\sim, \prec)$ such that:

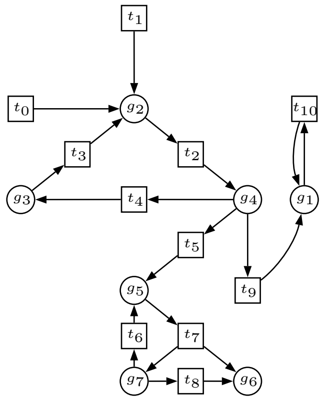
$$[t] \prec [t'] \textit{ iff } t \lesssim t' \textit{ and } t \not\gtrsim t' \tag{7.24}$$

Equivalently, we define a strict order on the set T $(T, \prec)$ such that:

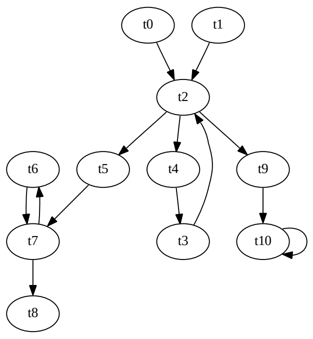
$$t \prec t' \textit{ iff } [t] \prec [t']$$

Example 7.7. As to the example in figure 7.2(b), $[t_2] \prec [t_5]$ ($t_2 \prec t_5$) since there exists a simple path from t_2 to t_5 ($\langle t_2, t_5 \rangle$)[6]. However $[t_2] \prec [t_4]$ does not hold since, although a simple path exists from t_2 to t_4 ($\langle t_2, t_4 \rangle$), we have that $t_2 \sim t_4$. In fact there are cycles ($\langle t_2, t_4, t_3, t_2 \rangle$).

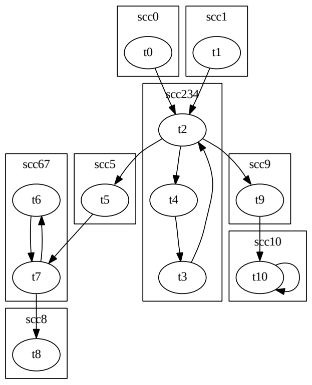
[6]Recall that according to the notation employed in section 2.4.2 a path in a graph is noted as $\langle v_1, \ldots, v_n \rangle$, where the v_i are the nodes belonging to the path.



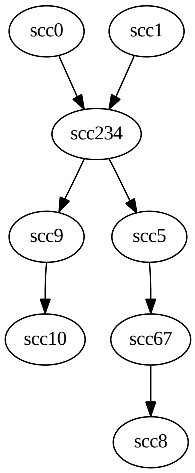
(a) A PTNS representing SCOs



(b) SDG



(c) SCCs of the SDG



(d) The strict order

Figure 7.2: A PTN structure, the corresponding SDG, SCC, and Order Relation.

Then, we are now ready to formally define the concept of *dependence*. We recall that two SCOs t, t' can be such that: (1) t depends on t' and t' does not depend on t; or (3) t and t' are mutually dependent; or (4) t and t' are not dependent on one another. More formally, we can distinguish the following three cases:

(1) $t \prec t'$: t *depends* on t'. A one-way directed path between t and t' exists in the SDG. Then, all the SCOs along the path connecting t to t' can contribute to increase the input goods of t'. Then, t' *depends* on their execution. For instance, in example 7.7, we have that t_5 depends on t_2. Therefore, pushing t ahead of t'

in a solution sequence does not cause a loss of solution classes.

(2) $t' \sim t$: t and t' are *mutually dependent*. There exist both a simple path from t to t' and another one from t' to t. Therefore, they lie on a simple cycle of the SDG. For instance, in figure 7.2(b), we have that $t_2 \sim t_4$. Obviously, we cannot order them since the circularity of the relationship implies that they depend on each other. Then, we may risk to lose some solution class if we change their relative order in a solution sequence.

(3) $t \not\precsim t'$ and $t' \not\precsim t$: no path exists between t and t'. The relative positions of t and t' within the solution sequence does not affect the feasibility of the solution in any case. Then, it does not matter the relative order of t and t' in the solution sequence, and it can be changed arbitrarily.

In what follows we present three examples referring to the three items in the list above.

Example 7.8 (Dependence)**.** In example 7.2 we were able to move t_1 in the first position of the solution sequence without losing solutions, whereas we could not do the same for t_2. This happens since t_1 does not depend on any SCO ($\nexists t$ such that $t \prec t'$), whereas t_2 depends on t_1 ($t_1 \prec t_2$) and t_0 ($t_0 \prec t_2$). Then, t_1 and t_0 must hold positions previous to t_2. This is why in the solution template in table 7.7, t_2 comes after t_1 and t_0.

Example 7.9 (Mutual Dependence)**.** In example 7.3, we saw that in the case of t_2, t_3, and t_4 we could not assign to each of these SCOs only one place in the solution sequence. In fact, we have that $t_2 \sim t_3 \sim t_4$. Then, in order to consider all the possible orderings among them, we assigned to them positions $5, 6, 7$, and 8 in the solution template of table 7.7.

Example 7.10 (Independence)**.** In example 7.2 we were able to move t_1 in the first position of the solution sequence without losing solutions. The reader can check that equivalently t_0 can be brought to the first position without affecting the validity of the solution. Thus, the solution template of table 7.7 can be modified by switching the positions of t_0 and t_1 as shown in table 7.9.

Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Solution Template	t_0	t_0	t_0	t_1	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_5	t_9	t_{10}	t_{10}	t_6 t_7	t_6 t_7	t_6 t_7	t_6 t_7	t_8

Table 7.9: Interchanging the positions of t_1 and t_0.

Hence, while we can a-priori establish an order among SCOs belonging to different equivalence classes, for SCOs within the same equivalence class we cannot since they are mutually dependent. As to the case of SCOs, we can chose any ordering.

7.2.2 Computing the equivalence classes

As shown in section 2.4, the definition of Strongly Connected Component (SCC) in graph theory coincides with the notion of equivalence class we defined above. The very good news is that there exists an algorithm that can find the SCCs of a graph (V, E) in polynomial time ($\Theta(V + E)$), as explained in (Cormen, 2001). The fact that we have available this algorithm significantly simplifies the first of our subproblems, that is the problem of finding an execution order among SCOs. In fact, once obtained the strongly connected components, enforcing a suitable ordering among them amounts to building a solution template.

Henceforth, we will refer indifferently to equivalence classes or SCCs.

Example 7.11. The strongly connected components of the graph in figure 7.2(b) are:

$$\begin{aligned}
scc0 &= \{t_0\} = [t_0] & scc67 &= \{t_6, t_7\} = [t_6] \\
scc1 &= \{t_1\} = [t_1] & scc8 &= \{t_8\} = [t_8] \\
scc234 &= \{t_2, t_3, t_4\} = [t_2] & scc9 &= \{t_9\} = [t_9] \\
scc5 &= \{t_5\} = [t_5] & scc10 &= \{t_{10}\} = [t_{10}]
\end{aligned}$$

They are graphically depicted in figure 7.2(c). As mentioned in section 2.4, it is also possible to define a strict order among equivalence classes (SCCs), graphically depicted in figure 7.2(d).

7.2.3 Order Enforcing Function

We mentioned at the beginning of this section that our aim is to build a *template* that allows us to a-priori limit the set of positions that each SCO can hold within a solution sequence in such a way that no solution class is lost. As illustrated by the template in table 7.7, there is a link between the dependencies among SCOs and their relative order. Most precisely, a solution template must comply with the strict order stemming from dependencies. Next, we provide a formal definition of solution template, the so-called *$\mathcal{D}$-bounded Order Enforcing Function*.

Definition 7.2 ($\mathcal{D}$-bounded Order Enforcing Function)**.** Given a strict order $(T/_\sim, \prec)$ and a multi-set $\mathcal{D} \in \mathbb{N}^T$, a $\mathcal{D}$-bounded Order Enforcing Function $S : \{1, \ldots, |\mathcal{D}|\} \rightarrow T/_\sim$ is a sequence of equivalence classes satisfying the following constraints:

$$S(i) \prec S(j) \Rightarrow i < j \tag{7.25}$$

$$|S^{-1}([t])| = \sum_{t' \in [t]} \mathcal{D}(t') \quad \forall [t] \in T/_\sim \tag{7.26}$$

Where $|S^{-1}([t])|$ is the number of times the equivalence class $[t]$ appears in the sequence S. Henceforth, S will denote a $\mathcal{D}$-bounded order enforcing function for $(T/_\sim, \prec)$.

Equation 7.25 guarantees that all the position assigned to the equivalence classes are in increasing order with respect to $(T/_\sim, \prec)$. This means that if $[t]$ comes before

$[t']$ according to $(T/_\sim, \prec)$, then $[t]$ comes before $[t']$ in S. Equation 7.26 ensures that enough positions in S are available to contain all the SCOs in $\mathcal{D}$ with their multiplicity. For instance, if three units of t_0 are offered, it means that up to three copies of t_0 may be present in the solution sequence. Then, three positions must be assigned to t_0 in S. Notice that there is no overlapping among the positions assigned to different equivalence classes in virtue of equation 7.25.

Example 7.12. If $\mathcal{D}$ is the multi-set of the overall SCOs received in the MMUCA of example 7.1. We define a $\mathcal{D}$-bounded enforcing function S as follows:

$$\begin{aligned}
S(1) &= [t_1]; & S(2) &= [t_0]; \\
S(3) &= [t_0]; & S(4) &= [t_0]; \\
S(5) &= [t_2]; & S(6) &= [t_2]; \\
S(7) &= [t_2]; & S(8) &= [t_2]; \\
S(9) &= [t_5]; & S(10) &= [t_9]; \\
S(11) &= [t_{10}]; & S(12) &= [t_{10}]; \\
S(13) &= [t_6]; & S(14) &= [t_6]; \\
S(15) &= [t_6]; & S(16) &= [t_6]; \\
S(17) &= [t_8];
\end{aligned}$$

Departing from solution template in table 7.7 we can represent function S as shown in table 7.10. The solution template readily leads to the mapping in table 7.10 by substituting each set of elements for the equivalence class it belongs to. For instance $\{t_2, t_3, t_4\}$ for $[t_2]$ and $\{t_6, t_7\}$ for $[t_6]$.

Positions	1	2	3	4	5	6	7	8	...
Equiv. classes	$[t_1]$	$[t_0]$	$[t_0]$	$[t_0]$	$[t_2]$	$[t_2]$	$[t_2]$	$[t_2]$	...

...	9	10	11	12	13	14	15	16	17
...	$[t_5]$	$[t_9]$	$[t_{10}]$	$[t_{10}]$	$[t_6]$	$[t_6]$	$[t_6]$	$[t_6]$	$[t_8]$

Table 7.10: $\mathcal{D}$-bounded enforcing function for example 7.1.

We employ S^{-1} to indicate the inverse of an enforcing function S. $S^{-1}([t])$ indicates the set of integers (positions) that map to the equivalence class $[t]$ via S. More formally:

$$S^{-1}([t]) = \{m \in \{1, \ldots, |\mathcal{D}|\} \mid S(m) = [t]\}$$

Example 7.13. Regarding example 7.12,

$$S^{-1}([t_6]) = S^{-1}([t_7]) = \{13, 14, 15, 16\}$$

In what follows, we show that it is always possible to build at least one solution template. We prove this by construction. This result is fundamental to our purposes since the S function is employed to encode our problem.

Lemma 7.1. *Given a strict order $(T/_\sim, \prec)$ and a multi-set $\mathcal{D} \in \mathbb{N}^T$ such that $\forall t \in T\ \mathcal{D}(t) \geq 1$, at least a $\mathcal{D}$-bounded order enforcing function S exists.*

Proof of lemma 7.1 Let $(q_1, q_2, ..., q_k)$, where $k = |T/_\sim|$, be an ordering of the elements of $T/_\sim$ satisfying $\prec$. Then, we build S as follows:

$$\begin{array}{llll}
S(1) = q_1 & S(2) = q_1 & \dots & S(\lambda_1) = q_1 \\
S(\lambda_1 + 1) = q_2 & S(\lambda_1 + 2) = q_2 & \dots & S(\lambda_1 + \lambda_2) = q_2 \\
\dots & \dots & \dots & \dots \\
S(\sum_{l=1}^{k-1} \lambda_l + 1) = q_k & S(\sum_{l=1}^{k-1} \lambda_l + 2) = q_k & \dots & S(\sum_{l=1}^{k-1} \lambda_l + \lambda_k) = q_k
\end{array}$$

where $\lambda_i = \sum_{t \in q_i} \mathcal{D}(t)$ Notice that S satisfies the constraints specified by equations 7.25 and 7.26. □

Notice that this proof also explains how to practically build a solution template given the SCCs.

S is thus a function that assigns positions within a sequence to set of SCOs. The main property of S is that every solution that DIP finds can be reordered into an *equivalent* and *feasible* solution that fulfils the solution template S. In order to formally define the concept of *fulfilment*, we have to introduce some notation. In fact, we need to link a solution to DIP to the solution template S. We begin by introducing partial sequences, a generalisation of the concept of sequence that captures the formal representation of a solution to solver DIP (see table 7.1).

7.2.4 Partial Sequences

We begin by recalling the definition of sequence.

Definition 7.3 (Sequence)**.** A *Sequence* over a non-empty finite set T is a function $K : [1, n] \to T$, with $n \in \mathbb{N}$.

Notice that in table 7.1 we represented a solution as a mapping from positions within a sequence to SCOs. In what follows we illustrate the concept of *partial sequence*, which intuitively is a sequence with "holes", meaning that there could be some positions of the sequence that are *empty*. This notion will be employed to formally capture solution sequences like the ones in tables 7.1, 7.2, and 7.3.

Definition 7.4 (Partial Sequence)**.** A *Partial Sequence* over a non-empty finite set T is a **partial** function $K : [1, n] \to T$, with $n \in \mathbb{N}$.

The fact the function is **partial** implies that some integers may have no image, representing the holes in the sequence.

From now on, we will employ the following notation:

(1) $|K|$ the length of the sequence. Henceforth, we will assume $|K| = n$;

(2) $K^{-1} : T \to 2^{[1,n]}$ is a partial injective function such that $m \in K^{-1}(t)$ iff $K(m) = t$ (inverse function);

(3) $|K^{-1}(t)|$ is the number of times t appears in sequence K;

(4) Given a multi-set $\mathcal{D} : T \to \mathbb{N}$, we will note as $\mathcal{D}(t)$ the multiplicity of t in $\mathcal{D}$;

(5) $Dom(K)$ is the subset of $[1, n]$ that admits an image via K (domain);

(6) $Im(K) = \{t \in T \mid K(m) = t \textit{ for some } m \in [1, n]\}$ (image)

Example 7.14. The partial sequence representing the solution sequence in table 7.1 is:

$$
\begin{aligned}
K(1) &= t_0 \\
K(2) &= t_2 \\
K(5) &= t_1 \\
K(6) &= t_0 \\
K(7) &= t_4 \\
K(8) &= t_0 \\
K(9) &= t_2 \\
K(14) &= t_3
\end{aligned}
$$

Obviously, a solution sequence can not contain more SCOs than the ones submitted in bids overall. Then, we further refine the representation of solutions by limiting the number of times each SCO can appear within a partial sequence.

Definition 7.5 ($\mathcal{D}$-bounded Partial Sequence)**.** Given a *partial sequence* K over a set T and a multi-set $\mathcal{D} \in \mathbb{N}^T$, we say that K is $\mathcal{D}$-bounded if:

$$|K^{-1}(t)| \leq \mathcal{D}(t) \qquad \forall t \in Im(K) \tag{7.27}$$

Example 7.15. The partial function defined in example 7.14 is bounded by the multi-set $\mathcal{D}$ in equation 7.9:

$$\mathcal{D} = \{t_0, t_0, t_0, t_1, t_2, t_2, t_3, t_4, t_5, t_6, t_6, t_7, t_7, t_8, t_9, t_{10}, t_{10}\}$$

this happens since:

$$\big(|K^{-1}(t_0)| = 3 \textit{ and } \mathcal{D}(t_0) = 3\big) \textit{ implies } |K^{-1}(t_0)| \leq \mathcal{D}(t_0) \tag{7.28}$$

$$\ldots\ldots \tag{7.29}$$

and a similar equation applies to all the other elements in $Im(K)$.

□

Notice that the multi-set $\mathcal{D} = \uplus_{ij}\mathcal{D}_{ij}$ associated to an MMUCA bounds all of its solutions, as state in the following observation.

Remark 7.1. Every solution to an MMUCA is a $\mathcal{D}$-bounded partial sequence.

Now that we have all the formal tools to describe a solution, we can define when a solution *fulfils* a solution template S (order enforcing function). This is a central point and leads us to the definition of S-fulfilment:

Definition 7.6 (S-fulfilment)**.** Given a $\mathcal{D}$-bounded partial sequence K and a $\mathcal{D}$-bounded order enforcing function S, we say that K fulfils S iff:

$$\forall i \in dom(K) \qquad K(i) \in S(i) \tag{7.30}$$

This means that a solution K complies with a solution template if each SCO in K takes on a position *allowed* by S.

Example 7.16. COnsider table 7.11

- the partial sequence K in does not fulfil the order enforcing function (solution template) **S**, since $K(1) = t_2$ and $S(1) = [t_1]$, but $t_2 \notin [t_1]$;
- the partial sequence K' fulfils **S**.

In table the highlighted SCOs do not hold the positions enforced by the solution template.

□

Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
(S) Solution Template	t_1	t_0	t_0	t_0	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_5	t_9	t_{10}	t_{10}	t_6 t_7	t_6 t_7	t_6 t_7	t_6 t_7	t_8
K	t_0	$\underline{\mathbf{t_2}}$			$\underline{t_1}$	$\underline{t_0}$	t_4	$\underline{t_0}$	$\underline{\mathbf{t_2}}$					$\underline{\mathbf{t_3}}$			
K'	t_1	t_0	t_0	t_0	t_2	t_4	t_2	t_3									

Table 7.11: Partial sequence fulfilling (K') and not fulfilling (K') S in table 7.10.

Then, now we can explain why the solution template represented by S is of central importance. We will formally prove in section 7.4 that each partial sequence, solution to the MMUCA WDP, can be reordered into an *equivalent* and *feasible* solution that fulfils S. Consequently, if we limit the search space so that *only* the solutions fulfilling S are included, we guarantee that no solution class is lost. Therefore, we achieve what we intended, obtaining a space search reduction without sacrificing solution classes.

In the next section we show how to apply an ordering enforcing function to ILP. We will present a new solver for the MMUCA WDP, that employs considerably less decision variables than DIP by exploiting S.

7.3 The improved IP formulation

The aim of this section is to introduce a new IP that improves solver DIP. We call the improved solver, described in the remaining of this section, solver CCIP. The idea underlying the improvement of solver DIP is to consider as possible solutions only partial sequences fulfilling a $\mathcal{D}$-bounded order enforcing function S and excluding all other solutions. With this purpose, we employ the order enforcing function introduced in definition 7.2.

In section 7.3.1, we introduce a preliminary version of the IP formulation of solver CCIP. In section 7.3.2 we introduce a further simplification that allows to eliminate part of the constraints. In section 7.3.3 we show that CCIP turns into a CMWOSP solver when the Mixed Auction Net is acyclic. Finally, in section 7.3.4 we show that CCIP turns into the DIP solver when the SDG is connected.

7.3.1 The Model

As usual, the input to the MMUCA WDP is a set of bids Bid_{ij}, each one over a multi-set $\mathcal{D}_{ij}$ along with some price p_{ij}. $\mathcal{D} = \bigcup_{ij} \mathcal{D}_{ij}$ is the multi-set of all the submitted SCOs. Then, the maximum length of the solution sequence is $\delta = |\mathcal{D}|$.

According to remark 7.1, a partial sequence representing a solution to the WDP is always bounded by $\mathcal{D}$, since the partial sequence will at most contain all the submitted SCOs. Then, we consider a $\mathcal{D}$-bounded order enforcing function S. The associated order relation $(T/_{\sim}, \prec)$ is the one defined by the SDG graph on T.

In solver DIP we employed binary decision variable x_{ijk}^m taking on value 1 iff SCO t_{ijk} is selected at the m-th position within the solution sequence. In the case of DIP, m ranges in all the positions of the solution sequence ($m \in [1, \delta]$). However, now we can assign a limited number of positions to each SCO via S. If we want to allow as feasible solutions only partial sequences fulfilling S, we only create decision variables for the positions each SCO can hold. More precisely we create decision variables x_{ijk}^m for all $m \in S^{-1}([t_{ijk}])$. By means of this operation we manage to drastically reduce the search space.

Next, analogously to section 6.2, we employ the following auxiliary decision variables. Firstly, x_{ijk} is an integer variable that represents the number of positions that SCO t_{ijk} holds in the solution sequence. Secondly, x_{ij} is a binary decision variables taking on value one if bid Bid_{ij} is selected and 0 otherwise. Then, we impose the following constraints.

(1) We obtain the number of positions that SCO t_{ijk} holds in the solution sequence (x_{ijk}) by summing up x_{ijk}^m over all the positions m assigned to $[t_{ijk}]$:

$$x_{ijk} = \sum_{m \in S^{-1}([t_{ijk}])} x_{ijk}^m \ \forall ijk \tag{7.31}$$

Example 7.17. Regarding example 7.1 we have:

$$x_{t_2} = x_{t_2}^5 + x_{t_2}^6 + x_{t_2}^7 + x_{t_2}^8 \tag{7.32}$$

and

$$x_{t_0} = x_{t_0}^2 + x_{t_0}^3 + x_{t_0}^4 \tag{7.33}$$

and so on.

(2) We are interested in that at most one SCO can hold each position. Consequently, we impose that:

$$\sum_{t_{ijk} \in S(m)} x_{ijk}^m \leq 1 \ \ \forall m \tag{7.34}$$

Notice that the sum is only over the SCOs of a single equivalence class. These constraints enforce that the solution is a partial sequence. Without such a constraint we could have more than one SCO assigned to the same position of the sequence.

Example 7.18. Following example 7.1, at step 5 ($m = 5$) the following constraints hold:

$$x_{t_2}^5 + x_{t_3}^5 + x_{t_4}^5 \leq 1 \tag{7.35}$$

(3) We need decision variables controlling if a given bid has been selected. As we know, the semantics of a bid implies that selecting at least one SCO within a bid implies selecting all the SCOs within the same bid with the corresponding multiplicity. That is:

$$x_{ijk} = x_{ij} \cdot \mathcal{D}_{ij}(t_{ijk}) \qquad \forall ijk \tag{7.36}$$

Example 7.19. For SCO t_0 in bid Bid_{11} of the MMUCA of example 7.1 we have:

$$x_{t_0} = x_{11} \cdot 3 \tag{7.37}$$

and so on.

(4) We impose that the XOR semantics of a bid is fulfilled, i.e. at most one bid per bidder can be selected:

$$\sum_j x_{ij} \leq 1 \qquad \forall i \tag{7.38}$$

(5) We need to encode the constraint enforcing that each SCO selected is enabled at any step of the solution sequence.

$$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} \sum_{t_{ijk} \in S(l)} x_{ijk}^l \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \sum_{t_{ijk} \in S(m)} x_{ijk}^m \cdot \mathcal{I}_{ijk}(g) \tag{7.39}$$

$$\forall g \in G, \forall m \in [1, \delta]$$

(6) We express the constraint enforcing that the goods available to the auctioneer at the end of the solution sequence is at least $\mathcal{U}_{out}$:

$$\mathcal{U}_0(g) + \sum_{m=0}^{\ell} \sum_{t_{ijk} \in S(m)} x_{ijk}^m \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \mathcal{U}_{out}(g) \qquad \forall g \in G \tag{7.40}$$

In table 7.12 we summarise the CCIP ILP formulation.

Finally, solving the MMUCA WDP is equivalent to optimise the objective function:

$$\max \sum_{ij} x_{ij} \cdot p_{ij} \tag{7.41}$$

subject to constraints (a-f) in table 7.12.

(a)	$\forall ijk$	$x_{ijk} = \sum_{m \in S^{-1}([t_{ijk}])} x^m_{ijk}$
(b)	$\forall ijk$	$x_{ijk} = x_{ij} \cdot \mathcal{D}_{ij}(t_{ijk}) \quad \forall ijk$
(c)	$\forall i$	$\sum_j x_{ij} \leq 1$
(d)	$\forall m$	$\sum_{t_{ijk} \in S(m)} x^m_{ijk} \leq 1$
(e)	$\forall g \in G$	$\mathcal{U}_0(g) + \sum_{m=0}^{\ell} \sum_{t_{ijk} \in S(m)} x^m_{ijk} \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \mathcal{U}_{out}(g)$
(f)	$\forall g \in G$ $\forall m \in [1, \delta]$	$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} \sum_{t_{ijk} \in S(l)} x^l_{ijk} \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \sum_{t_{ijk} \in S(m)} x^m_{ijk} \cdot \mathcal{I}_{ijk}(g)$
(g)		$\max \sum_{ij} x_{ij} \cdot p_{ij}$

Table 7.12: Resume of the IP formulation of solver CCIP.

7.3.2 Eliminating some Equations

There is a further simplification that we can add. Not only we can reduce the number of variables, but we are also able to eliminate some redundant constraints. It follows from some considerations on the IP structure that we can remove some constraints from solver CCIP because redundant. In what follows we provide the corresponding intuitions.

Equation (7.39) ensures that enough goods are present to perform the selected SCOs at each step of the solution sequence. It must be applied at each step of the solution sequence.

$$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} \sum_{t_{ijk} \in S(l)} x_{ijk}^l \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \sum_{t_{ijk} \in S(m)} x_{ijk}^m \cdot \mathcal{I}_{ijk}(g) \qquad (7.42)$$

$$\forall g \in G, \forall m \in [1, \delta]$$

Equation (7.40) states that at the end of the sequence at least $\mathcal{U}_{out}$ goods are available to the auctioneer.

$$\mathcal{U}_0(g) + \sum_{m=0}^{\ell} \sum_{t_{ijk} \in S(m)} x_{ijk}^m \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \mathcal{U}_{out}(g) \qquad \forall g \in G \qquad (7.43)$$

The application of the two constraints above plus the fact that S limits the possible position assignments makes some of those constraints redundant. In particular, we can get rid of constraint 7.42 at each step m if the group of SCO assigned to step m via S does not belong to any cycle of the graph. The following example will clarify the statement.

Example 7.20. Consider the MMUCA WDP presented in example 7.1. In particular, we will focus on the equations regarding good g_4 just before the firing of t_5. Then, considering that the only SCOs that can add or remove tokens to g_4 are $\{t_2, t_4, t_5, t_9\}$, equation 7.43 becomes[7]:

$$x_{t_2} - x_{t_4} - x_{t_5} - x_{t_9} \geq 0 \qquad (7.46)$$

Notice that the x_{t_i} in in equation 7.46 are integer, not binary variables. Now we consider equation 7.42 at step 9 and for good g_4. According to table 7.7, SCO t_5 is assigned to position 9. Then, equation 7.42 becomes[8]:

$$x_{t_2} - x_{t_4} \geq x_{t_5} \qquad (7.49)$$

[7]Equation 7.43 can be rewritten as

$$0 + \sum_{l=0}^{\ell} \sum_{t \in \{t_2, t_4, t_5, t_9\}} x_t^l \cdot [\mathcal{O}_t(g_4) - \mathcal{I}_t(g_4)] \geq 0 \qquad (7.44)$$

that can be rewritten as

$$0 + \sum_{t \in \{t_2, t_4, t_5, t_9\}} x_t \cdot [\mathcal{O}_t(g_4) - \mathcal{I}_t(g_4)] \geq 0 \qquad (7.45)$$

since $x_t = \sum_{l=0}^{\ell} x_t^l$ (equation 6.91). Expanding equation 7.45, we obtain 7.46.

[8]Equation 7.42 can be rewritten as:

$$0 + \sum_{l=0}^{8} \sum_{t \in T} x_t^l \cdot [\mathcal{O}_t(g_4) - \mathcal{I}_t(g_4)] \geq x_{t_5}^9 \cdot \mathcal{I}_{t_5}(g_4) \qquad (7.47)$$

Once again, since the only SCOs that can add or remove tokens to g_4 are $\{t_2, t_4, t_5, t_9\}$ and their assigned positions in table 7.7 are $\{5, 6, 7, 8, 9, 10\}$, equation 7.47 becomes:

$$0 + \sum_{l \in \{5,6,7,8\}} \sum_{t \in \{t_2, t_4\}} x_t^l \cdot [\mathcal{O}_t(g_4) - \mathcal{I}_t(g_4)] \geq x_{t_5}^9 \cdot \mathcal{I}_{t_5}(g_4) \qquad (7.48)$$

The reader can check that expanding this expression we obtain equation 7.49.

Notice that equation (7.49) is automatically satisfied if equation (7.46) is fulfilled. Then, equation 7.49 is redundant, and we can get rid of it.

Example 7.20 above shows an important properties of our model. This property is not very intuitive because it depends on the equations and on their relationships. Anyway, as we mentioned above, this property does not hold if the SCO lies on a simple cycle or on a self-loop of the SDG. A counter example will clarify this sentence.

Example 7.21. We consider again example 7.1, but in this case we focus on the equations for good g_1. Notice that a a self-loop is present on t_{10}. This self-loop prevents to apply the same reasoning of example 7.20. First, we write the equivalent of equation 7.46 and we obtain:

$$x_{t_9} + x_{t_{10}} - x_{t_{10}} \geq 0 \Longrightarrow x_{t_9} \geq 0 \tag{7.50}$$

Next, we consider equation 7.42 at step 11. We obtain:

$$x_{t_9} - x_{t_{10}} \geq 0 \tag{7.51}$$

It is easy too see that equation 7.50 does not imply equation 7.51. Then, we cannot get rid of equation 7.51.

Intuitively, equation 7.43 is a *global* condition enforcing that at the end of the sequence the global input-output balance at each good of the net in figure 7.1 is positive. On the other hand, equation 7.42 is *local* to each step, and enforces that enough input goods are available at each step.

As showed by example 7.21, we have to check constraint 7.42 only when the SCOs assigned to position m belong to a cycle. Notice that by definition each time an equivalence class contains $n > 1$ SCOs, each SCO in the equivalence class belongs to a simple cycle of length n.

Example 7.22. (1) t_{10} in figure 7.2(a) has good g_1 both as input and output. Then it belongs to a self-loop ($\langle t_{10}, t_{10}\rangle$). (2) t_2 belongs to the simple cycle $\langle t_2, t_4, t_3, t_2\rangle$. (3) t_1 does not belong to any cycle.

To conclude, with respect to the encoding of solver CCIP presented in section 7.3.1, we can get rid of a set of inequations. Thus, besides reducing the number of variables, we decrements the number of constraints of solver DIP.

Employing the terminology introduced in section 7.3.2, we can say that equation 7.39 must be added only if the SCOs assigned to step m by S belong to a simple cycle. Then, defining L_F as:

$$L_F = \{m \in \{1, \dots \ell\} \mid S(m) \text{ contains a simple cycle}\}$$

We can rewrite equation (7.39) as:

$$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} \sum_{t_{ijk} \in S(l)} x_{ijk}^{l} \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \sum_{t_{ijk} \in S(m)} x_{ijk}^{m} \cdot \mathcal{I}_{ijk}(g) \qquad \forall g \in G, \forall m \in L_F \tag{7.52}$$

In appendix A.3 we present the CCIP model with reduced constraints encoded in the OPL language (see section 2.1.2 and (Van Hentenryck, 1999)).

Problem Size

The number of decision variables in the above integer program is of the order of $O(\sum_{ijk}|S^{-1}(t_{ijk})|)$ (corresponding to x^m_{ijk}). More in details, we create a binary decision variable for each bid $Bid_{ij} \in B$, for a total of $|B|$ binary decision variables. Then, we create a decision variable x_{ijk} for each SCO $t_{ijk} \in T$, for a total of $|T|$ decision variables. Then, we create a decision variable for each SCO $t_{ijk} \in T$ and for each position it is allowed to take on within the solution sequence, for a total of $\sum_{ijk}|S^{-1}(t_{ijk})|$ binary decision variables. Then, we create a total of

$$|B| + \sum_{ijk}|S^{-1}(t_{ijk})| \in O(\sum_{ijk}|S^{-1}(t_{ijk})|)$$

decision variables.

With a similar process, we compute the total number of constraints, that is:

$$2|T| + |L| + \delta + |G|(1 + |L_F|) \tag{7.53}$$

7.3.3 The CMWOSP-based solver is a special case of CCIP

In this section, we show that the CMWOSP-based solver introduced in section 6.1.5 is a special case of CCIP. Say that we know that no cycles are present in the TDG. That means that we will have exactly as many SCCs as the number of SCOs. Then, we have that equation (a) in table 7.15 turns into:

$$x_{ijk} = x^m_{ijk} \tag{7.54}$$

Considering this,

- equation (d) becomes redundant and we can eliminate it;
- equation (e) turns into

$$\mathcal{U}_0(g) + \sum_{m=0}^{\ell} \sum_{t_{ijk}\in S(m)} x_{ijk} \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \mathcal{U}_{out}(g) \quad \forall g \in G \tag{7.55}$$

  that can be rewritten as:

$$\mathcal{U}_0(g) + \sum_{ijk} x_{ijk} \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \mathcal{U}_{out}(g) \qquad \forall g \in G \tag{7.56}$$

Since the net has no cycles, it happens that $|L_F| = 0$. Then, equation (f) can be eliminated. Since x^m_{ijk} is not employed in any equation, we can eliminate equation (7.54) as well.

Then, joining equations (b), (c), and (g) in table 7.15 with equation (7.56), we obtain exactly the same ILP model as the one in equations (7.55).

7.3.4 CCIP amounts to DIP when the SDG is connected

Analogously, we show that when the SDG is connected CCIP turns into DIP. If the SDG is connected, then there is a big SCC encompassing all the SCOs. It happens that:

$$S(m) = T \qquad \forall m \in [1, \delta] \tag{7.57}$$

$$L_F = \{1, \ldots, \delta\} \tag{7.58}$$

Then, we create decision variables x^m_{ijk} $\forall t_{ijk} \in S(m)$. Then, basically, we create the same decision variables as in DIP. Next, the constraint in equation (7.52) must be applied at each step of the solution sequence. Then, we obtain the same formulation as DIP.

7.4 Equivalence between solvers DIP and CCIP

In this section we formally prove that no solution class is lost limiting the possible positions of SCOs via S. This result is indirectly proved by showing that:

- each solution found by solver DIP can be reordered into a solution to CCIP; and
- each solution to solver CCIP is also a solution to DIP.

If this holds, then we are guaranteed that: (1) for each solution in DIP solution space there is always an equivalent solution in CCIP solution space; and (2) the CCIP solution space is a subset of the DIP solution space. Then, CCIP does not lose solutions nor create new solutions not fulfilling DIP.

Such proof is rather complex. Then, before going on, we introduce some definitions and constructs that will be employed in the demonstration. To this end, in sections 7.4.1 and 7.4.2 we provide formal tools to capture the notion of reordering of a solution. Then, in section 7.4.4 we introduce some properties of partial sequences of SCOs to be employed for the proof. Finally, in section 7.4.5, we provide the formal proof.

7.4.1 Subsequences

In what follows we introduce the concept of subsequence of a partial sequence. A subsequence of some partial sequence is a new *sequence* obtained from the former one by removing some of the elements and all the empty positions without disturbing the relative positions of the remaining elements. An example will clarify the sentence:

Example 7.23. Consider the partial sequence in table 7.13.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**Revenue**
Sequence 1	t_0	t_2			t_1	t_0	t_4	t_0	t_2					t_3				+3 USD

Table 7.13: Example of solution found by solver DIP.

Example of subsequences of the partial sequence of table 7.13 are showed in the following. At the left hand side we have shown the subsequence, while on the right hand side the original partial sequence with highlighted the elements selected to form the subsequence. The element ϵ indicates a *hole* in the partial sequence.

$$\langle t_1, t_4, t_2\rangle \qquad \langle t_0, t_2, \epsilon, \epsilon, \underline{\mathbf{t_1}}, t_0, \underline{\mathbf{t_4}}, t_0, \underline{\mathbf{t_2}}, \epsilon, \epsilon, \epsilon, \epsilon, t_3, \epsilon, \epsilon, \epsilon\rangle \tag{7.59}$$

$$\langle t_1, t_0, t_0, t_2\rangle \qquad \langle t_0, t_2, \epsilon, \epsilon, \underline{\mathbf{t_1}}, \underline{\mathbf{t_0}}, t_4, \underline{\mathbf{t_0}}, \underline{\mathbf{t_2}}, \epsilon, \epsilon, \epsilon, \epsilon, t_3, \epsilon, \epsilon, \epsilon\rangle \tag{7.60}$$

$$\langle t_1\rangle \qquad \langle t_0, t_2, \epsilon, \epsilon, \underline{\mathbf{t_1}}, t_0, t_4, t_0, t_2, \epsilon, \epsilon, \epsilon, \epsilon, t_3, \epsilon, \epsilon, \epsilon\rangle \tag{7.61}$$

$$\langle t_4, t_2\rangle \qquad \langle t_0, t_2, \epsilon, \epsilon, t_1, t_0, \underline{\mathbf{t_4}}, t_0, \underline{\mathbf{t_2}}, \epsilon, \epsilon, \epsilon, \epsilon, t_3, \epsilon, \epsilon, \epsilon\rangle \tag{7.62}$$

$$\langle t_2, t_1, t_4\rangle \qquad \langle t_0, \underline{\mathbf{t_2}}, \epsilon, \epsilon, \underline{\mathbf{t_1}}, t_0, \underline{\mathbf{t_4}}, t_0, t_2, \epsilon, \epsilon, \epsilon, \epsilon, t_3, \epsilon, \epsilon, \epsilon\rangle \tag{7.63}$$

$$\langle t_0, t_4, t_2, t_3\rangle \qquad \langle \underline{\mathbf{t_0}}, t_2, \epsilon, \epsilon, t_1, t_0, \underline{\mathbf{t_4}}, t_0, \underline{\mathbf{t_2}}, \epsilon, \epsilon, \epsilon, \epsilon, \underline{\mathbf{t_3}}, \epsilon, \epsilon, \epsilon\rangle \tag{7.64}$$

Notice that the order among the elements is maintained. That is, for instance t_3 comes after t_2 in the partial sequence of table 7.1, then the same must happen in equation 7.64.

Definition 7.7 (Subsequence of a partial sequence)**.** Say $K : [1, n] \rightarrow T$ is a partial sequence. We say that K' is a subsequence of K iff:

- K' is a *sequence* of elements of T. More formally, $K' : \{1, \ldots, m\} \rightarrow T$ where $m \in \mathbb{N}$ and[9] $m \leq n$.
- There is a strictly increasing function (called characteristic function of the sequence) $f : \{1, \ldots, m\} \rightarrow [1, n]$ such that:

$$K'(i) = K(f(i))\ \forall i \in \{1, \ldots, m\}$$

Example 7.24. The characteristic function of subsequence 7.59 is:

$$\begin{aligned} f(1) = 5 \rightarrow &\quad \text{the first element of } K' \text{corresponds to the fifth element of } K \\ f(2) = 7 \rightarrow &\quad \text{the second element of } K' \text{corresponds to the seventh element of } K \\ f(3) = 9 \rightarrow &\quad \text{the third element of } K' \text{corresponds to the nine-th element of } K \end{aligned}$$

We call the *inverse characteristic function* of a subsequence K' the function retrieving the position of an element of the subsequence within the original sequence. We denote it as $f_{K'}^{-1} : [1, n] \rightarrow \{1, \ldots, m\}$. For instance $f_{K'}^{-1}(j) = k$ means that the position within the original sequence of the $j - th$ element of the subsequence was k.

Example 7.25. The inverse of the characteristic function of example 7.24 is:

$$\begin{aligned} f^{-1}(5) &= 1 \\ f^{-1}(7) &= 2 \\ f^{-1}(9) &= 3 \end{aligned}$$

Given a partial sequence $K : \mathbb{N} \rightarrow T$ and a set $T' \subseteq T$, we define the subsequence of K restricted to T', denote as $K_{|_{T'}}$, as the subsequence of K obtained removing from K all the elements not belonging to T'. More formally:

[9]Notice that K' is a *sequence*, not a *partial sequence*.

Definition 7.8 (Sequence restricted to a subset)**.** Given a partial sequence $K : \mathbb{N} \rightarrow T$ and a set $T' \subseteq T$, $K_{|T'}$ (to be read as K restricted to T') is a subsequence of K such that:

$$|K_{|T'}^{-1}(t)| = |K^{-1}(t)| \qquad \forall t \in T' \qquad (7.65)$$

$$|K_{|T'}^{-1}(t)| = 0 \qquad \forall t \notin T' \qquad (7.66)$$

Example 7.26. Given the partial sequence of table 7.13, $K_{|\{t_2,t_3,t_4\}} = \langle t_2, t_4, t_2, t_3 \rangle$.

In the following section we introduce a formalism to describe how to order a partial sequence in order to make it comply with a solution template.

7.4.2 Reordering Sequences

The main goal of this section is to introduce the theoretical tools to check whether the reordering of a partial sequence complies with a solution template. We recall that this is useful since we have to prove that, for each solution to DIP, there always exists a reordering of it that complies with the solution template. Then, in this section, as a first step we provide a definition of reordering of a partial sequence that fulfils the solution template S.

Intuitively, an S-fulfilling reordering is a reordering of K into a new partial sequence K' that fulfils S and that preserves the order defined by K among the SCOs within the same equivalence class. Before giving the formal details, we present an example.

Example 7.27. Consider order enforcing function S and the partial sequence K in table 7.14. In table 7.14 we present also K', an $S-$fulfilling reordering of K, and K'' a partial sequence that is not an $S-$fulfilling reordering of K. Notice that in K' the elements t_0 and t_1 have been reordered, whereas the elements of the equivalence class $[t_2]$ maintain the same order as in K. Observe that K'' fulfils S, but the elements of the equivalence class $[t_2]$ are in a different relative order than in K (t_2 comes after t_4).

Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
(S) Solution Template	t_1	t_0	t_0	t_0	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_2 t_3 t_4	t_5	t_9	t_{10}	t_{10}	t_6 t_7	t_6 t_7	t_6 t_7	t_6 t_7	t_8
K	t_0	$\underline{\mathbf{t_2}}$			t_1	t_0	$\underline{\mathbf{t_4}}$	t_0	$\underline{\mathbf{t_2}}$					$\underline{\mathbf{t_3}}$			
K'	t_1	t_0	t_0	t_0	$\underline{\mathbf{t_2}}$	$\underline{\mathbf{t_4}}$	$\underline{\mathbf{t_2}}$	$\underline{\mathbf{t_3}}$									
K''	t_1	t_0	t_0	t_0	$\underline{\mathbf{t_4}}$	$\underline{\mathbf{t_2}}$	$\underline{\mathbf{t_2}}$	$\underline{\mathbf{t_3}}$									

Table 7.14: Examples of S-fulfilling (K') and not S-fulfilling (K'') reordering of K.

Thus, we proceed to the formal definition of $S-$fulfilling reordering.

Definition 7.9 (S-fulfilling reordering)**.** Given a $\mathcal{D}$-bounded partial sequence K and a $\mathcal{D}$-bounded order enforcing function S, K' is an S-fulfilling reordering of K iff:

(1) K' fulfils S

(2) $K'_{|q} = K_{|q} \qquad \forall q \in T/_{\sim}$

Point (1) implies that K' complies with the solution template. Point (2) of definition 7.9 implies that the order among SCOs belonging to the same equivalence class in K' is the same as in K.

In section 7.4 we will be interested in retrieving the original position in of the elements of a reordered sequence. That is, given K' S-fulfilling reordering of K, we are interested in retrieving the original position i in K of the j-th element of K', as shown in the following example.

Example 7.28. Consider the partial sequences K and K' employed in example 7.27. Say that we are interested in retrieving the original positions within K of the $7th$ element of K'. From table 7.14 we know that $K'(7) = t_2$. The natural way of doing it would be to look for the position of SCO t_2 in K. But this does not work since t_2 appears more than once in K. Indeed, we have that $K^{-1}(t_2) = \{5, 7\}$. Then, we have to recur to the characteristic functions that are employed to build the subsequences $K'_{|q}$ and $K_{|q}$ (point (2) of definition 7.9).

Remark 7.2. Say K' is an S fulfilling reordering of K. Then, the original position $\tilde{s}$ in K of the $\tilde{m}$-th element of K' is:

$$\tilde{s} = f_{K_{|q}}\left(f^{-1}_{K'_{|q}}(\tilde{m})\right) \tag{7.67}$$

Where $q = [K'(\tilde{m})]$ is the equivalence class that contains the $\tilde{m}$-th element of K', $f^{-1}_{K'_{|q}}$ is the characteristic function associated to the subsequence ${K'}_{|q}$, and $f_{K_{|q}}$ is the inverse characteristic function associated to the subsequence $K_{|q}$ (see definitions 7.7 and 7.8).

Now we are going to provide an existence result: no matter which is the partial sequence, there always exists an S-fulfilling reordering of it. Hence, in what follows we provide both a theorem of existence and a way to build an S-fulfilling reordering.

Proposition 7.1. *Given a $\mathcal{D}$-bounded partial sequence K and a $\mathcal{D}$-bounded order enforcing function S, an S-fulfilling reordering K' always exists.*

Proof of proposition 7.1 The demonstration is carried out by construction. For each equivalence class $[t] \in T/_{\sim}$ we define two sequences. One contains all the integers mapping to $[t]$ via S (i.e. $S^{-1}([t])$) ordered in increasing value. The other one is the sequence K restricted to the set $[t]$ (the subsequence $K_{|[t]}$). Then, $\forall [t] \in T/_{\sim}$:

$$\text{be } (s_1, s_2, \ldots, s_i, \ldots, s_a) \text{ s.t. } s_i < s_{i+1} \text{ and } \{s_i\}_{i=1}^{a} = S^{-1}([t]) \tag{7.68}$$

$$\text{be } (t_1, t_2, \ldots, t_j, \ldots, t_b) = K_{|[t]} \tag{7.69}$$

Then, we define K' as $K'(s_i) = t_i \; \forall i \in [1, \ldots, b]$ and $\forall [t] \in T/_{\sim}$.

Point (2) of definition 7.9 is trivially satisfied by construction. Point (1) of definition 7.9 is satisfied if $K(s_i) \in S(s_i)$. But per construction we have that $K(s_i) = t_i, t_i \in [t]$ and $S(s_i) = [t]$. $\square$

In the following section we add some complementary definitions about sequences and order relationships.

7.4.3 Order Fulfilling Sequences

Our aim is to assess the positions to a-priori assign to SCOs in such a way that the order established by the SDG is not violated. We explained in section 7.2.1 that we have to make sure that all the SCOs such that $t \prec t'$ must be assigned positions such that t comes before t' in the sequence. Thus, the first step is knowing which ones, among the possible solutions, do not violate the strict order imposed by $(T/_{\sim}, \prec)$. With this purpose, we give a definition to decide whether a partial sequence fulfils a strict order relationship.

Definition 7.10 (Order Fulfilling Sequence)**.** We say that a partial sequence K over T fulfils an order relation $(T/_{\sim}, \prec)$ iff:

$$\forall i, j \in dom(K) \qquad [K(i)] \prec [K(j)] \Rightarrow i < j \tag{7.70}$$

This definition formally states that a partial sequence K fulfils the order relationship $\prec$ only if the relative order among SCOs within K does not violate $\prec$.

Example 7.29. The partial function K in table 7.14 does not fulfil the order relationship defined by the SDG in figures 7.2(b) and (c). K violates the order relationship in various points. For instance, we have that t_2 appears before t_1 although $t_1 \prec t_2$ holds. Observe that this does not mean that this solution is not valid, but only that it does not fulfil the order relation. On the opposite, partial sequence K' and K'' in table 7.14 fulfil the order relationship $\prec$.

As mentioned above, S is a template that a partial sequence must adhere in order to fulfil $\prec$. Hence, we must define the conditions for a partial sequence to satisfy a given solution template.

In what follows we formally show that a sequence fulfilling S also fulfils the order relationship $\prec$.

Lemma 7.2. *If K fulfils S, then K fulfils $(T/_{\sim}, \prec)$.*

Proof of lemma 7.2

From equations 7.30 and 7.25 it follows that $[K(i)] \prec [K(j)] \Rightarrow i < j$. □

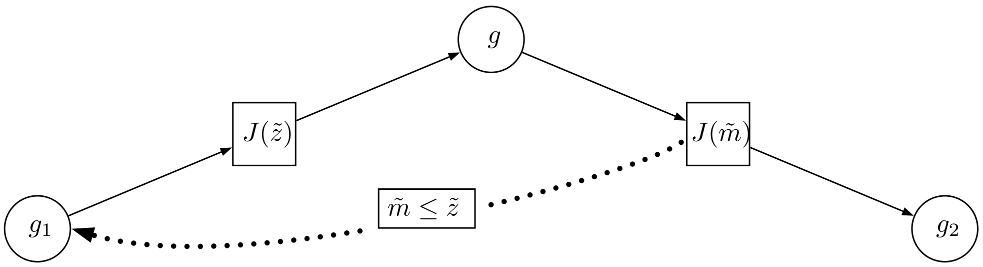
The order enforcing function is exactly the solution template we were looking for. Any partial sequence (and thus any solution) fulfilling it also fulfils the order relationship $(T/_{\sim}, \prec)$. This is a very important property, since it means that the precedence relationship among SCOs are fulfilled within such a partial sequence.

In what follows, we detail some definition and properties employed in the proof of section 7.4.5.

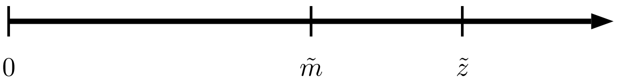
7.4.4 Properties of partial sequences of SCOs

In this section we demonstrate some properties of partial sequences of SCOs that fulfil an order relationship $(T/_{\sim}, \prec)$. Those properties will be useful in section 7.4.5. In particular we will deal with a special case: the case in which two SCOs t and t' are such that $t \lesssim t'$ but t comes after t' in a partial sequence fulfilling $(T/_{\sim}, \prec)$. We will call it the case of *forward swapping*. Notice that this can happen only if t and t' are in the same equivalence class. Then, in what follows:

- T is a set of SCOs equipped with the preorder defined by its SDG $(T, \lesssim)$ (as in section 7.2.1). Recall that a SCO $t \in T$ is composed by a pair of multisets of goods: $t = (\mathcal{I}_t, \mathcal{O}_t)$, where $\mathcal{I}_t, \mathcal{O}_t \in \mathbb{N}^G$.
- $J : \mathbb{N} \to T$ is a partial sequence of SCOs that fulfils the order relationship $(T/_\sim, \prec)$.



(a) g is output of $J(\tilde{z})$ and input of $J(\tilde{m})$.



(b) Position $\tilde{m}$ comes before position $\tilde{z}$ in the J partial sequence.

Figure 7.3: $J(\tilde{z})$ is *forwardly swapped* with $J(\tilde{m})$ in g.

J has some important properties that we detail in the following. But before that, we give an important definition, the definition of SCOs *forwardly swapped*.

We provide an example with some intuitions of the definition of *forwardly swapped*. For instance, figure 7.3 graphically depicts the case in which $J(\tilde{z})$ is forwardly swapped with $J(\tilde{m})$. Intuitively, say that $\tilde{m}$ is a position of the partial sequence J such that the associated SCO $J(\tilde{m})$ has g as an input good[10] ($\mathcal{I}_{J(\tilde{m})}(g) > 0$). Say also that further ahead in the sequence, at position $\tilde{z} \geq \tilde{m}$, there is a SCO that has g as output good ($\mathcal{O}_{J(\tilde{z})}(g) > 0$). In such a case we say that SCO $J(\tilde{z})$ is forwardly swapped with $J(\tilde{m})$. In figure 7.3(a) we show that g is both input of $J(\tilde{m})$ and output of $J(\tilde{z})$, whereas in figure 7.3(b) we graphically represent that $\tilde{z}$ comes after $\tilde{m}$ in the J solution sequence.

[10]The notation here is such that $\mathcal{O}_{J(l)}$ means $\mathcal{O}_{ijk}$ where the corresponding SCO is $t_{ijk} = J(l)$

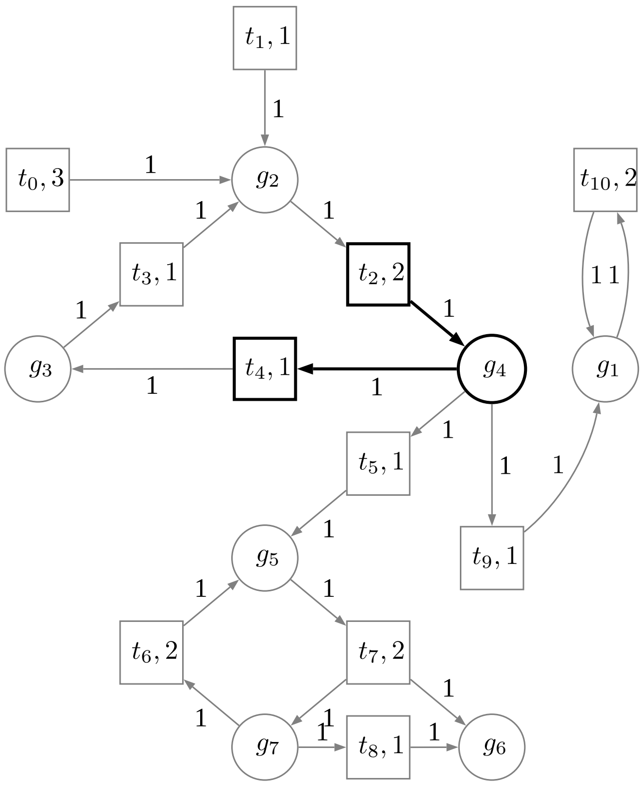


Figure 7.4: Part of the SDG of example 7.1

Definition 7.11 (Forwardly Swapped)**.** Given a partial sequence J that fulfils an order relationship $(T/_\sim, \prec)$, a good $g \in G$ and two positions $\tilde{m}, \tilde{z} \in dom(J)$ such that:

(1) $\mathcal{I}_{J(\tilde{m})}(g) > 0$

(2) $\tilde{m} \leq \tilde{z}$

(3) $\mathcal{O}_{J(\tilde{z})}(g) > 0$

then we say that $J(\tilde{z})$ is *forwardly swapped* with $J(\tilde{m})$ in g. If $\tilde{m} = \tilde{z}$ we say that $J(\tilde{m})$ has a *self-loop*.

In what follows we present a lemma that describes three important properties of partial sequences with swapped SCOs. The lemma is very intuitive. The intuition behind this is explained by the following example.

Example 7.30. Consider the partial sequence K' in table 7.14 . By definition it fulfils S. Say $\tilde{m} = 6$, then we have that $K'(\tilde{m}) = t_4$. We also have that $\mathcal{I}_{t_4}(g_4) > 0$ (see figure 7.4). In figure 7.4 we extend figure 7.1 by highlighting in thick black the SCOs that are concerned in this example. Observe that t_2, whose output good is g_4, is at position $\tilde{z} = 7$ of K', after t_4. The reader can check that:

(1) t_2 and t_4 belong to a simple cycle;

(2) $t_4 \prec t_5$, $t_4 \prec t_9$; and

(3) $t_2 \sim t_4$.

This means that if $J(\tilde{z})$ is forwardly swapped with $J(\tilde{m})$ in g, then it holds that:

(1) $J(\tilde{m})$ and $J(\tilde{z})$ belong to the same equivalence class;

(2) any SCO t that has g as input good must be such that either $J(\tilde{m}) \prec t$ or $J(\tilde{m}) \sim t$;

(3) any SCO t that has g as output good must be such that either $t \prec J(\tilde{m})$ or $t \sim J(\tilde{m})$.

These properties are generalised in the following lemma.

Lemma 7.3. *If $J(\tilde{z})$ is forwardly swapped with $J(\tilde{m})$ in g, then:*

(1) $J(\tilde{m}) \sim J(\tilde{z})$;

(2) for all t such that $\mathcal{I}_t(g) > 0$, either $J(\tilde{m}) \sim t$ or $J(\tilde{m}) \prec t$;

(3) for all t such that $\mathcal{O}_t(g) > 0$, either $J(\tilde{m}) \sim t$ or $t \prec J(\tilde{m})$.

Proof of lemma 7.3

(1) If $\tilde{z} = \tilde{m}$ this is trivially true. Otherwise, we have that $J(\tilde{z}) \lesssim J(\tilde{m})$ since g is output of $J(\tilde{z})$ and input of $J(\tilde{m})$. If we had that $J(\tilde{m}) \not\sim J(\tilde{z})$, then $[J(\tilde{z})] \prec [J(\tilde{m})]$ would hold. Hence, from definition 7.9:

$$[J(\tilde{z})] \prec [J(\tilde{m})] \Rightarrow \tilde{z} \prec \tilde{m} \tag{7.71}$$

that is against the initial hypothesis. Then, we can conclude that $J(\tilde{m}) \sim J(\tilde{z})$.

(2) $J(\tilde{z}) \lesssim t$ since g is output of $J(\tilde{z})$ and input of t. But we know from the previous point that $J(\tilde{m}) \sim J(\tilde{z})$. Then, it cannot be $J(\tilde{m}) \not\lesssim t$ by transitivity. At this point only two possibilities remain, either $J(\tilde{m}) \sim t$ or $J(\tilde{m}) \prec t$. If $\tilde{m} = \tilde{z}$ the same discussion holds setting $J(\tilde{m}) = J(\tilde{z})$.

(3) It is clear that $t \lesssim J(\tilde{m})$ since g is output of t and input of $J(\tilde{m})$. But we know from the previous point that $J(\tilde{m}) \sim J(\tilde{z})$. At this point only two possibilities remain, either $t \sim J(\tilde{m})$ or $t \prec J(\tilde{m})$. If $\tilde{m} = \tilde{z}$ the same discussion holds setting $J(\tilde{m}) = J(\tilde{z})$.

□

Notice that from lemma 7.3 follows that, under the hypothesis considered above, there cannot be any SCO with g as input or output good that is not in relation with $J(\tilde{m})$ via $\lesssim$.

Corollary 7.1. *If $J(\tilde{z})$ is forwardly swapped with $J(\tilde{m})$ in g, then for all t such that $\mathcal{O}_t(g) > 0$ or $\mathcal{I}_t(g) > 0$: it cannot be $t \not\lesssim J(\tilde{m})$.*

7.4.5 Equivalence between solvers

In this section we prove that no solution class is lost by limiting the positions via an ordering enforcing function. We prove this result indirectly. Instead of relying on the definition of MMUCA WDP, we build our proof departing from the corresponding ILP formulation, that is DIP (see section 6.2).

In fact, if we prove that (1) each solution to DIP can be reordered into a solution to CCIP, and that (2) each solution to CCIP is also a solution to solver DIP, then we demonstrate that no solution class is lost. In fact, there is a reason for employing this indirect proof. By doing this, we also prove that the operation performed in section 7.3.2 — the elimination of part of the constraints of solver CCIP— is legal.

With this in mind we demonstrate the following two theorems:

Theorem 7.1. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils all the constraints of solver CCIP.*

Theorem 7.2. *Given a partial sequence J, solution to solver CCIP, it fulfils all the constraints of DIP.*

Theorems 7.1 and 7.2 will be proved in the remaining of the chapter. Relying on those theorems, we can prove that:

Corollary 7.2. *Any solution found by solver DIP can be reordered into a solution to solver CCIP.*

Proof of corollary 7.2 Say H is a solution to solver DIP with objective value c_H. Assume that there exists an S-fulfilling reordering J of H that is not a solution to solver CCIP. The cost associated to solution sequence J is equal to the cost c_H associated to solution H since J is a reordering of H. For theorem 7.1 J fulfils all the constraints of CCIP. Since J is not an optimal solution to solver CCIP, there must exist another solution J' to solver CCIP with objective value $c_{J'} > c_J = c_H$. However, from theorem 7.2, we have that each solution to solver CCIP fulfils all the constraints imposed by DIP . Then, J' should be an optimal solution to DIP since it fulfils all its constraints and its objective value is larger than c_H. This is against the hypothesis that the solution is H. It follows that J is an optimal solution to CCIP. □

Corollary 7.3. *Any solution found by solver CCIP is a solution to solver DIP.*

Proof of corollary 7.3 Say J is an optimal solution to solver CCIP with cost c_J. Assume that it is not an optimal solution to solver DIP. From theorem 7.2 we have that J fulfils all the constraints of solver DIP. Then, there should be another solution H of DIP with cost $c_H > c_J$. In this case, in virtue of theorem 7.1, it could be reordered into a solution sequence fulfilling the constraints of solver CCIP. Since H fulfils the constraints of CCIP and has cost $c_H > c_J$, it should be an optimal solution to solver CCIP. This is against the hypothesis that the solution is J with objective value c_J. It follows that J is an optimal solution to DIP. □

7.4.6 Proof of theorem 7.1

As mentioned above, a solution to the IP model of the MMUCA WDP defined in section 6.2 can be expressed by means of a partial sequence $H : \{1, \ldots, |\mathcal{D}|\} \to T$ such that[11] $H(m) = t$ iff $x_t^m = 1$. A solution to solver CCIP, too, can be expressed by means of a partial sequence $J : \{1, \ldots, |\mathcal{D}|\} \to T$ such that $J(m) = t$ iff $x_t^m = 1$.

The first step is thus proving that for each solution to DIP there exists a reordering of it that fulfils the constraints of CCIP. The intuition behind this theorem is illustrated by the following example:

Example 7.31. Consider again example 7.27. The partial sequence K' in table 7.14 is an S-fulfilling reordering of sequence K in table 7.14 . Recall that K is a solution to the MMUCA of example 7.1 found by solver DIP. K' is a still valid solution to the MMUCA of example 7.1.

With theorem 7.1 we aim at demonstrating the universality of the result of example 7.31: any S-fulfilling reordering of a solution is still a valid solution to the MMUCA WDP. For the sake of simplicity we resume the IP CCIP in table 7.15.

Since the proof of theorem 7.1 is complex and rather long, we begin by demonstrating several lemmas.

First, we show that J fulfils equations (a),(b), and (c) in table 7.15. Equation (a) sums into x_{ijk} the number of times SCO t_{ijk} appears in a solution sequence. Equation (b) enforces that either all or none of the SCOs within a bid are selected. Equation (c)enforces that at most one bid is selected for each bidder.

Lemma 7.4. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils constraints (a),(b), and (c) in table 7.15.*

Proof of lemma 7.4 x_{ijk} of equation 6.91 counts the number of times t_{ijk} appears in the solution H. Thus, we have that $x_{ijk} = |H^{-1}(t_{ijk})|$. On the other hand, x_{ijk} in equation 7.15(a) counts the number of times SCO t_{ijk} appears in J. Then, we have $x_{ijk} = |J^{-1}(t_{ijk})|$. Since J is an *S-fulfilling* reordering of H, from point (2) of definition 7.9 we can derive that $|H^{-1}(t_{ijk})| = |J^{-1}(t_{ijk})| \; \forall t_{ijk}$. Then, the variables x_{ijk} assume the same values in equation 6.91 for H as in equation (a) in table 7.15 for J.

From this follows trivially that equations (b) and (c) in table 7.15 are fulfilled by J if equations 6.91 and 6.92 are fulfilled by H. Furthermore, the objective values of the two IPs DIP and CCIP assume the same optimal value, i.e. equations 6.96 and (g) in table 7.15 assume the same values for H and J respectively. □

Next we consider equation (d) in table 7.15. Equation (d) enforces that at most one SCO can hold each position of the solution sequence.

Lemma 7.5. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils constraint (d) in table 7.15.*

Proof of lemma 7.5 Constraint (d) in table 7.15 is fulfilled iff at most one SCO is selected at each position of the solution sequence. Since J is a partial sequence, it

[11]In order to ease the notation we will henceforth employ t for indicating the generic SCO t_{ijk}. Equivalently, we will employ t_p to indicate $t_{i_p j_p k_p}$ and t' to indicate $t_{i'j'k'}$.

(a)	$\forall ijk$	$x_{ijk} = \sum_{m \in S^{-1}([t_{ijk}])} x_{ijk}^m$
(b)	$\forall ijk$	$x_{ijk} = x_{ij} \cdot \mathcal{D}_{ij}(t_{ijk}) \quad \forall ijk$
(c)	$\forall i$	$\sum_j x_{ij} \leq 1$
(d)	$\forall m$	$\sum_{t_{ijk} \in S(m)} x_{ijk}^m \leq 1$
(e)	$\forall g \in G$	$\mathcal{U}_0(g) + \sum_{m=0}^{\ell} \sum_{t_{ijk} \in S(m)} x_{ijk}^m \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \mathcal{U}_{out}(g)$
(f)	$\forall g \in G$ $\forall m \in L_F$	$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} \sum_{t_{ijk} \in S(l)} x_{ijk}^l \cdot [\mathcal{O}_{ijk}(g) - \mathcal{I}_{ijk}(g)] \geq \sum_{t_{ijk} \in S(m)} x_{ijk}^m \cdot \mathcal{I}_{ijk}(g)$
(g)		$\max \sum_{ij} x_{ij} \cdot p_{ij}$

Table 7.15: Resume of the IP formulation of solver CCIP.

cannot be the case that more than one SCOs is associated to a single position. Then it is always fulfilled. □

Next, we consider equation (e) of table 7.15. Equation (e) enforces that the goods available to the auctioneer at the end of the solution sequence is at least $\mathcal{U}_{out}$.

Lemma 7.6. *Given a partial sequence H, solution to solver DIP, any S-fulfilling re-ordering J of H fulfils constraints (e) of table 7.15.*

Proof of lemma 7.6 We can rewrite equation (e) of table 7.15 considering that the solution is J. Then we have:

$$\mathcal{U}_0(g) + \sum_{m \in dom(J)} [\mathcal{O}_{J(m)}(g) - \mathcal{I}_{J(m)}(g)] \geq \mathcal{U}_{out}(g) \tag{7.72}$$

Now we rewrite equation 6.95 of solver DIP considering that the solution is H:

$$\mathcal{U}_0(g) + \sum_{m \in dom(H)} [\mathcal{O}_{H(m)}(g) - \mathcal{I}_{H(m)}(g)] \geq \mathcal{U}_{out}(g) \tag{7.73}$$

Notice that $\forall t \in T \; |H^{-1}(t)| = |J^{-1}(t)|$ since J is an $S - fulfilling$ reordering of H. Then, the Left Hand Side (LHS) of equations 7.72 and 7.73 assume the same value. Then, trivially, equation (e) of table 7.15 is fulfilled by J if equation 6.95 is fulfilled by H. $\square$

The most complex demonstration is ensuring that all the selected SCOs are enabled at each step. This means checking that equation (f) of table 7.15 is always fulfilled by J given that 6.94 is fulfilled by H. Then, we further divide the demonstration of this lemma in some sub lemmas. But before that, we rewrite equations 6.94 and (f) of table 7.15 considering that the solutions are H and J respectively[12]:

$$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} [\mathcal{O}_{H(l)}(g) - \mathcal{I}_{H(l)}(g)] \geq \mathcal{I}_{H(m)}(g) \tag{7.74}$$
$$\forall g \in G, \forall m \in [1, ..., \ell]$$

for equation 6.94 and

$$\mathcal{U}_0(g) + \sum_{l=0}^{m-1} [\mathcal{O}_{J(l)}(g) - \mathcal{I}_{J(l)}(g)] \geq \mathcal{I}_{J(m)}(g) \tag{7.75}$$
$$\forall g \in G, \forall m \in [1, ..., \ell]$$

for equation (f) of table 7.15[13].

In the demonstration, we will prove that equation 7.75 is fulfilled by J for a general good g and at a step $\tilde{m}$ that complies with different hypothesis. The different hypothesis will be treated in the different lemmas that follow.

The first non trivial case is obviously when the SCO associated to step $\tilde{m}$ requires input goods from g. In this case, too, we have to distinguish two sub-cases that are described in lemmas 7.7 and 7.9. Lemma 7.7 deals with the case in which there is no SCO that adds tokens into g holding a position after $\tilde{m} - 1$ in partial solution sequence J, whereas lemma 7.9 considers the complementary case.

[12] In order not to overcharge the notation, we write $\sum_{l=0}^{m-1}$ instead of $\sum_{l \prec m : l \in dom(J)}$.

[13] We should add constraint 7.75 only for steps m associated to SCOs belonging to cycles ($m \in L_F$). Instead, we add it for every m. However, this case is more restrictive and then, if it is satisfied in this case, it will be also fulfilled if we remove the equations corresponding to $m \notin L_F$.

Lemma 7.7. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils constraints (f) of table 7.15 at a step $\tilde{m}$ and at a good g such that:*

- $\mathcal{I}_{J(\tilde{m})}(g) > 0$
- $\forall \tilde{z} \geq \tilde{m}\ O_{J(\tilde{z})}(g) = 0$

Proof of lemma 7.7 In this case we can write equation 7.75 as[14]:

$$\mathcal{U}_0(g) + \sum_{l=0}^{\ell} \mathcal{O}_{J(l)}(g) \geq \sum_{l=0}^{\tilde{m}} \mathcal{I}_{J(l)}(g) \tag{7.77}$$

since after step $\tilde{m}$ no SCO can add further contributions to good g. We can also write the following inequation:

$$\sum_{l=0}^{\tilde{m}} \mathcal{I}_{J(l)}(g) \leq \sum_{l=0}^{\ell} \mathcal{I}_{J(l)}(g) \tag{7.78}$$

since there could be SCOs with positions after $\tilde{m}$ that remove tokens from g. Then, equation 7.77 is fulfilled if the following equation is fulfilled:

$$\mathcal{U}_0(g) + \sum_{l=0}^{\ell} \mathcal{O}_{J(l)}(g) \geq \sum_{l=0}^{\ell} \mathcal{I}_{J(l)}(g) \tag{7.79}$$

Considering that equation 7.72 holds, equation 7.79 is satisfied. □

Now we deal with the most problematic sub-case, i.e. when there exists a SCO that can add tokens into g and holds a position after $\tilde{m}$. Notice that this case is someway connected to the case of forwardly swapped SCOs (see definition 7.11). In order to cover this case, we have to demonstrate two lemmas. The first follows:

Lemma 7.8. *Given a partial sequence H, solution to solver DIP, and an S-fulfilling reordering of it J, assume that at a step $\tilde{m} \in dom(J)$ and for a good $g \in G$ it holds that:*

(1) $\mathcal{I}_{J(\tilde{m})}(g) > 0$, i.e. g is an input good to transition $J(\tilde{m})$;

(2) $\exists \tilde{z} \geq \tilde{m}$ such that $O_{J(\tilde{z})}(g) > 0$, i.e. $J(\tilde{z})$ is forwardly swapped with $J(\tilde{m})$ in g;

(3) $q = [J(\tilde{m})]$, i.e q is the equivalence class of $J(\tilde{m})$;

[14]Equation 7.75 can be rewritten in an equivalent form if we bring to the right hand side of the equation all the terms containing $\mathcal{I}$:

$$\mathcal{U}_0(g) + \sum_{l=0}^{\tilde{m}-1} \mathcal{O}_{J(l)}(g) \geq \sum_{l=0}^{\tilde{m}} \mathcal{I}_{J(l)}(g) \tag{7.76}$$

(4) *$\tilde{s} = f_{H_{|q}}\left(f^{-1}_{J_{|q}}(\tilde{m})\right)$, i.e. $\tilde{s}$ is the position in H corresponding to the position $\tilde{m}$ in J*[15].

Then, we have that:

$$\sum_{l=0}^{\tilde{s}-1} \mathcal{O}_{H(l)}(g) \leq \sum_{l=0}^{\tilde{m}-1} \mathcal{O}_{J(l)}(g) \tag{7.80}$$

and

$$\sum_{l=0}^{\tilde{s}-1} \mathcal{I}_{H(l)}(g) \geq \sum_{l=0}^{\tilde{m}-1} \mathcal{I}_{J(l)}(g) \tag{7.81}$$

Proof of lemma 7.8 We begin by checking that equation 7.80 holds. Recall that since hypothesis (2) of lemma 7.8 holds, from lemma 7.3 we have that all the SCOs t with g as output good ($\mathcal{O}_t(g) > 0$) that exist are such that $t \prec J(\tilde{m})$ or $t \sim J(\tilde{m})$. Then, only those SCOs can contribute to increase the Right Hand Side (RHS) and LHS of equation 7.80.

With this in mind we show that equation 7.80 is fulfilled. Then, denoting $\tilde{t} = J(\tilde{m}) = H(\tilde{s})$, we have that:

(1) All the SCOs t=J(p) such that $t \prec \tilde{t}$ and $O_t(g) > 0$ have added their contribution to the RHS of equation 7.80, but not necessarily to the LHS. This is because for point (1) of definition 7.9 we have that:

$$[J(p)] \prec [J(\tilde{m})] \Rightarrow p \prec \tilde{m} \tag{7.82}$$

Oppositely, not necessarily all the SCOs $t \prec \tilde{t}$ have added their contribute to the LHS of equation 7.80.

(2) Any SCO $t \sim \tilde{t}$ that has added its contribute to the RHS of equation 7.80 has also given its contribute to the LHS either. This is because per hypothesis of S-fulfilling reordering $H_{|[J(\tilde{m})]} = J_{|[J(\tilde{m})]}$, i.e. the order in which the SCO within the same equivalence class are executed is the same for H and J.

Similarly, we check that equation 7.81 is fulfilled. Recall that since hypothesis (2) of lemma 7.8, from lemma 7.3 we have that all the SCOs t with g as input good ($\mathcal{I}_t(g) > 0$) that exist are such that $J(\tilde{m}) \prec t$ or $t \sim J(\tilde{m})$. Then, only those SCOs can contribute to increase the RHS and LHS of equation 7.80.

With this in mind we show that equation 7.81 is fulfilled since, denoting $\tilde{t} = J(\tilde{m}) = H(\tilde{s})$, we have that:

[15]Since H is a reordering of J, there must exists a step $\tilde{s}$ to which-is associated the SCO corresponding to the $\tilde{m}$-th position of J. Step $\tilde{s}$, corresponding to the original position in H of the $\tilde{m}$-th element of J, can be computed as explained in proposition 7.2. Recall that $f_{H_{|[J(\tilde{m})]}}$ is the characteristic function of the sequence H restricted to the set $[J(\tilde{m})]$, whereas $f^{-1}_{J_{|[J(\tilde{m})]}}$ is the inverse characteristic function of the sequence J restricted to the set $[J(\tilde{m})]$.

(1) No SCO t such that $t > \tilde{t}$, and $I_t(g) > 0$ has given its contribute to the RHS of equation 7.81. This is because, say $t = J(p)$, then for point (1) of definition 7.9 we have that:

$$[J(\tilde{m})] \prec [J(p)] \Rightarrow \tilde{m} \prec p \tag{7.83}$$

Oppositely, some of the SCOs $\tilde{t} \prec t$ may have contributed to increase the LHS of equation 7.81

(2) Any SCO t such that $t \sim \tilde{t}$ that has not executed in the RHS of equation 7.80 has not been executed in the LHS either. This is because per hypothesis $H_{|_{[J(\tilde{m})]}} = J_{|_{[J(\tilde{m})]}}$, i.e. the order in which the SCO within the same equivalence class are executed is the same for H and J.

□

With the result of lemma 7.8 at hand we can proceed to deal with the most problematic sub-case:

Lemma 7.9. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils constraints (f) of table 7.15 at a step $\tilde{m}$ and at a good g such that:*

- $\mathcal{I}_{J(\tilde{m})}(g) > 0$
- $\exists \tilde{z} \geq \tilde{m}$ *such that* $O_{J(\tilde{z})}(g) > 0$

Proof of lemma 7.9 Notice that we are under the hypothesis (1) and (2) of lemma 7.8. .As we mentioned in footnote 15, since J is a reordering of H, there must exists a position $\tilde{s}$ of H corresponding to position $\tilde{m}$ in J. That is $\tilde{s} = f_{H_{|q}}\left(f^{-1}_{J_{|q}}(\tilde{m})\right)$, as in hypothesis (4) of lemma 7.8.

Now consider that equation 7.74 is fulfilled for all m, since H is a solution to solver DIP. In particular it will hold at position $\tilde{s}$. Then, if we rewrite expressions 7.74 and 7.75 at those steps $\tilde{s}$ and $\tilde{m}$

$$\mathcal{U}_0(g) + \sum_{l=0}^{\tilde{s}-1} \mathcal{O}_{H(l)}(g) - \mathcal{I}_{H(l)}(g) \geq \mathcal{I}_{H(\tilde{s})}(g) \tag{7.84}$$

$$\mathcal{U}_0(g) + \sum_{l=0}^{\tilde{m}-1} \mathcal{O}_{J(l)}(g) - \mathcal{I}_{J(l)}(g) \ \geq \ \mathcal{I}_{J(\tilde{m})}(g) \tag{7.85}$$

an we check that the LHS of equation 7.84 is smaller than the LHS of equation 7.85, we are sure that equation 7.85 is fulfilled at step $\tilde{m}$, since it is fulfilled in equation 7.84 per hypothesis. Then, we check if the following equation is fulfilled:

$$\sum_{l=0}^{\tilde{s}-1} \mathcal{O}_{H(l)}(g) - \mathcal{I}_{H(l)}(g) \leq \sum_{l=0}^{\tilde{m}-1} \mathcal{O}_{J(l)}(g) - \mathcal{I}_{J(l)}(g) \tag{7.86}$$

With easy algebraic SCOs is is easy to check that equation 7.86 is satisfied if both equation

$$\sum_{l=0}^{\tilde{s}-1} \mathcal{O}_{H(l)}(g) \leq \sum_{l=0}^{\tilde{m}-1} \mathcal{O}_{J(l)}(g) \tag{7.87}$$

and equation

$$\sum_{l=0}^{\tilde{s}-1} \mathcal{I}_{H(l)}(g) \geq \sum_{l=0}^{\tilde{m}-1} \mathcal{I}_{J(l)}(g) \tag{7.88}$$

are satisfied. In virtue of lemma 7.8 the equations above are satisfied. □

The last lemma deals with a trivial case. That is, when we consider a good g and a step $\tilde{m}$ of the partial sequence J for which g in not an input good to the selected SCO $J(\tilde{m})$. Since g is not an input good, equation 7.75 assume a trivial form.

Lemma 7.10. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils constraints (f) of table 7.15 at a step $\tilde{m}$ and at a good g such that $\mathcal{I}_{J(\tilde{m})}(g) = 0$*

Proof of lemma 7.10 The SCO enabled at step $\tilde{m}$ does not require input goods from good g. It is a trivial case since if the equation was enabled at step $\tilde{m} - 1$ it is enabled at step $\tilde{m}$. At step 1 it is enabled since $\mathcal{U}_0(g) \geq 0$. □

In conclusion, we showed that equation 7.76 is fulfilled for every m and for every g when J is the solution sequence. Then we can now give a further lemma:

Lemma 7.11. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils constraints (f) of table 7.15.*

Proof of lemma 7.11 all the possible cases are covered by lemmas 7.10, 7.7, and 7.9. □

Finally, after proving all the parts of the theorem, we restate it and prove it.

Theorem 7.3. *Given a partial sequence H, solution to solver DIP, any S-fulfilling reordering J of H fulfils all the constraints of solver CCIP.*

Proof of theorem 7.1 All the equations are covered by lemmas 7.4, 7.5, 7.6, 7.11 □

7.4.7 Proof of theorem 7.2

At this point we have to check that the other way around is true, too. That is, given a solution to solver CCIP, this fulfils all the constraints of solver DIP.

Proof of theorem 7.2 First, notice that imposing that $x_{ijk}^{m} = 0 \; \forall m \notin S([t_{ijk}])$ for DIP we obtain the same equations as for solver CCIP, excepting expression (f) of table 7.15. The fact the we do apply the expression only when the position m is associated to a SCO belonging to a cycle ($m \in L_F$) creates an asymmetry between the two problems. However, in what follows we show that equation (f) of table 7.15 is automatically fulfilled for J in solver CCIP when $m \notin L_F$. We rewrite equation (f) of table 7.15 considering that the solution is J (as in equation 7.75) for a generic step $\tilde{m} \notin L_F$ to which a SCO $\tilde{t} = J(\tilde{m})$ is associated:

$$\mathcal{U}_0(g) + \sum_{l=0}^{\tilde{m}-1} [\mathcal{O}_{J(l)}(g) - \mathcal{I}_{J(l)}(g)] \geq \mathcal{I}_{J(\tilde{m})}(g) \tag{7.89}$$

Per absurd, say that for a solution J of solver CCIP this does not hold:

$$\mathcal{U}_0(g) + \sum_{l=0}^{\tilde{m}-1} [\mathcal{O}_{J(l)}(g) - \mathcal{I}_{J(l)}(g)] < \mathcal{I}_{J(\tilde{m})}(g) \tag{7.90}$$

Notice that constraint 7.72 enforces that at the end of the sequence the units of good g available must be at least 0:

$$\mathcal{U}_0(g) + \sum_{m=0}^{\ell} [\mathcal{O}_{J(m)}(g) - \mathcal{I}_{J(m)}(g)] \geq \mathcal{U}_{out}(g) \geq 0 \tag{7.91}$$

Then, if equation 7.90 holds at step $\tilde{m}$, there must exist some SCO $t = J(\tilde{z})$, holding a position $\tilde{z} \geq \tilde{m}$ in the solution sequence that adds tokens into g ($O_t(g) > 0$). we have two cases:

(1) $\tilde{z} = \tilde{m}$: in this case $J(\tilde{m})$ has a self-loop, and thus $J(\tilde{m})$ belongs to the cycle $\langle J(\tilde{m}), J(\tilde{m})\rangle$.

(2) $\tilde{z} > \tilde{m}$: In this case $J(\tilde{z})$ is forwardly swapped with $J(\tilde{m})$ in g. From this follows that, in virtue of point (1) of lemma 7.3, $J(\tilde{z}) \sim J(\tilde{m})$. Then, $J(\tilde{m})$ belongs to a cycle.

Both of the possibilities contradicts the initial hypothesis $\tilde{m} \notin L_F$. Then, we can conclude that any solution to solver CCIP fulfils equation 7.89 when $\tilde{m}$ does not belong to a cycle. ☐

7.5 Conclusions

In this chapter we have proposed a representation of the MMUCA WDP that considerably reduces the search space. This is obtained by reducing the space of feasible solutions. We have showed that the pruned solutions can be always reordered into equivalent solutions belonging to the reduced solution space. Such a reduction in the solution space entails a reduction in the size of the search space.

Notice also that computing the order enforcing function is computationally easy. In fact, there is a very efficient algorithm to compute the SCCs of the SDG graph (Cormen, 2001). This is an important point to consider. Thus, we managed to divide the MMUCA WDP problem into two subproblems, and one of those subproblems is solvable in polynomial time.

Obviously, the number of decision variables and the size of the search space depend on the size of cycles in a mixed auction net. In fact, we showed that the number of required decision variables for CCIP is $O(\sum_{ijk} |S^{-1}(t_{ijk})|)$. Thus, the bigger the strongly connected components, the more the number of decision variables. In the next section we provide a preliminary empirical test that confirms that the reduction in the search space corresponds to a reduction in the solving time of CCIP with respects to DIP.

Finally, notice that when all the SCOs form a unique Strongly Connected Component (i.e. the SDG is connected), DIP and CCIP provide exactly the same ILP model. Whereas when the SCOs do not form any cycle, CCIP is equivalent to the CMWOSP-based solver. Then. we can infer that CCIP perfectly exploits the topology associated to SCOs and generalises both solvers CCIP and DIP.

Chapter 8

Empirical Evaluation

The purpose of this chapter is to perform a preliminary empirical evaluation of the CMWOSP-based (presented in section 6.1.5), DIP, and CCIP solvers. In fact, our goal is to provide some useful hints on the applicability of MMUCAs.

The chapter is structured as follows. In section 8.1, we motivate the experiments provided in this chapter. In section 8.2 we summarise the artificial data set generator for MMUCAs presented in (Vinyals, 2007b), and detail the corresponding algorithm. In section 8.3, we analyse some early, empirical results after:

- running and comparing DIP and CCIP solvers on arbitrary network topologies;
- running the CMWOSP-based solver on acyclic network topologies[1].

Finally, we draw some conclusions in section 8.4.

8.1 Motivation

Despite its potential for application, and like CAs, little is known about the practical application of MMUCAs since no real-world data is available to test WD algorithms. Such results are unlikely to come up unless researchers are provided with algorithms or test suites to generate artificial data representative of the auction scenarios a WD algorithm is likely to encounter.

In the very recent past, there have been some attempts to empirically evaluate the performances of MMUCA WDP algorithms. In particular, Vinyals et. al. (Vinyals et al., 2007a; Vinyals et al., 2007b; Vinyals, 2007b) carefully analyse the performances of the DIP solver, after providing an algorithm to generate artificial data sets that are representative of the sort of scenarios a WD algorithm is likely to encounter. In those works Vinyals et al. show that DIP scales up to small and medium scenarios depending on the testing parameters.

[1]We recall that the CMWOSP-based solver, introduced in section 6.1.5, can only deal with acyclic network topologies.

In this chapter, we employ Vinyals' bid generator algorithm to generate artificial data and subsequently compare the performances of Integer Programming (IP) implementations of the DIP, CCIP, and CMWOSP-based solvers. In appendix A we present those models encoded in the OPL language (see section 2.1.2 and (Van Hentenryck, 1999)).

Firstly, we compare DIP and CCIP on arbitrary network topologies. Recall that, as proved in chapter 7, CCIP provides a more concise IP formulation than DIP in terms of both number of decision variables and constraints. In this chapter we empirically quantify the computational cost reduction deriving from the reduction of the number of decision variables. Secondly, we run the CMWOSP-based solver to assess the performances of CCIP on acyclic networks.

Notice that our empirical evaluation focuses on a proof-of-concept scenario. Therefore, an accurate quantitative comparison would require a much wider range of scenarios. However, this is left out for future work because it is beyond the scope of this thesis.

8.2 The Artificial Data Set Generator

In order to perform our evaluation, we employ a test set generator designed and implemented by Vinyals in her master's thesis (Vinyals, 2007b), and thoroughly explained in other publications (Vinyals et al., 2007a; Vinyals et al., 2007b). Vinyals et. al present an algorithm to generate artificial data that is representative of the sort of scenarios a winner determination algorithm is likely to encounter and provide a very detailed analysis of the computational performance of DIP. The empirical evaluation contained in this chapter has been developed in close collaboration with Vinyals.

In what follows, we summarise the details of the generator proposed by Vinyals et. al. Firstly, we specify the requirements the generator is expected to fulfil, and then we present some implementation details.

Since the bid generator is not a contribution of this dissertation, we only briefly summarise some of the bid generator features. The reader that is interested in the bid generator details should refer to the above mentioned publications.

8.2.1 Bid Generator Requirements

In order to test and compare MMUCA WD algorithms, researchers must be provided with algorithms or test suites to generate artificial data that is representative of the auction scenarios a WD algorithm is likely to encounter. Hence, WD algorithms can be accurately tested, compared, and improved. Unfortunately, we cannot benefit from any previous results in the literature since they do not take into account the notion of SCO introduced in chapters 4 and 5. In this section, we make explicit the requirements for a bid generation technique considering that in MMUCA agents trade SCOs instead of goods.

A naive approach to artificial bid generation would be to create bids uniformly at random. However, this approach would generate unrealistic bids and therefore unrealistic scenarios. Let us consider a random bid $b = (1'(\mathcal{I}, \mathcal{O}), p)$. If goods appearing in sets $\mathcal{I}$ and $\mathcal{O}$ are selected uniformly at random, there is little chance that they will represent a realistic SCO. Also, if p is chosen uniformly at random, it will not be related with the actual values of the goods in the sets $\mathcal{I}$ and $\mathcal{O}$ and consequently the SCO would be either too profitable or too expensive for the auctioneer, unrealistically easing the problem.

If individual bids uniformly at random generated may be unrealistic, bundles of random bids also present similar drawbacks.

Then, testing WD algorithms on these scenarios is almost useless, because any extracted conclusion cannot be used in real settings. The bid generator has to satisfy a number of requirements to make the artificial bids close to the bids that are likely to appear in a real-world auction.

In what follows, we introduce an example to illustrate the requirements the generator must fulfil.

Example 8.1. Consider the assembly of a car's engine, whose structure is depicted in Figure 8.1. In the figure, we employ a graphical representation analogous to Place Transition Nets. Notice that each part in the diagram, in turn, is produced form further components or raw materials. For instance, a cylinder ring (part 8) is produced by transforming some amount of stainless steel with the aid of an appropriate machine. Therefore, there are several production levels involved in the making of a car's engine. A MMUCA allows to run an auction where bidders can bid over bundles of parts, bundles of SCOs, or any combination of parts and SCOs. Notice that the result of an MMUCA WD algorithm would be an ordered sequence of bids making explicit how bidders coordinate to progressively transform goods till producing engines as final products. Therefore, an MMUCA would allow to assemble a supply chain from bids.

□

Since MMUCAs generalise CAs, as discussed in chapter 5, the approach is to depart from artificial data sets generators for CAs, keeping the requirements summarised in (Leyton-Brown and Shoham, 2006), namely:

(1) there is a finite set of goods;

(2) certain goods are more likely to appear together than others;

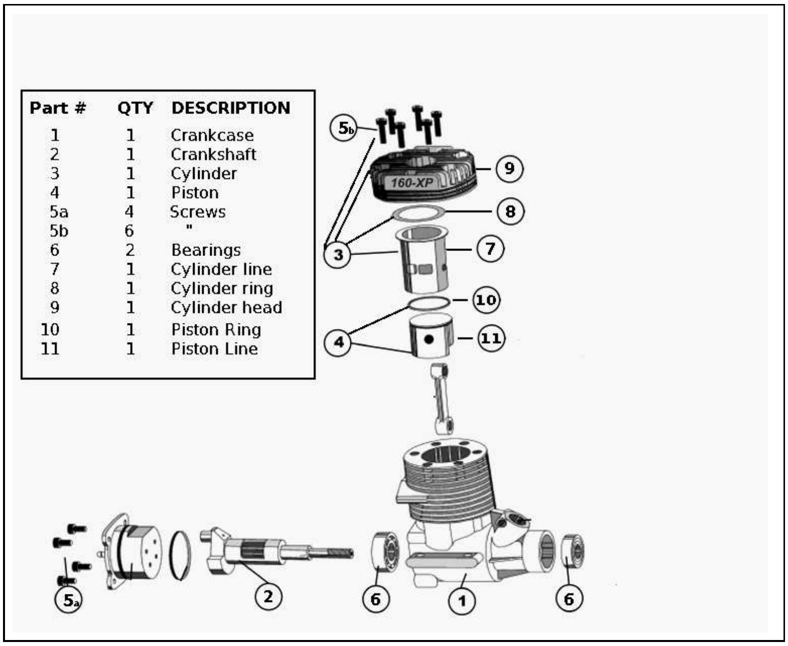
(3) the number of goods in a bundle is often related to which goods compose the bundle;

(4) valuations are related to which goods appear in the bundle;

(5) valuations can be configured to be sub-additive, additive or super-additive in the number of goods requested; and

(6) sets of XOR'ed bids are constructed on a per-bidder basis.

Notice though that the requirements above must be reformulated, and eventually extended, in terms of SCOs since a bidder in a MMUCA bids over a bundle of SCOs, whereas a bidder in a CA bids over a bundle of goods. Hence, in what follows we discuss the CA requirements listed above reformulated for MMUCA.



Part #	QTY	DESCRIPTION
1	1	Crankcase
2	1	Crankshaft
3	1	Cylinder
4	1	Piston
5a	4	Screws
5b	6	"
6	2	Bearings
7	1	Cylinder line
8	1	Cylinder ring
9	1	Cylinder head
10	1	Piston Ring
11	1	Piston Line

Figure 8.1: Components of a car engine.

1. There is a finite set of SCOs. A CA generator bundles goods from a given set of goods to construct bids. What is the set of SCOs from which a MMUCA generator constructs bids? In order to provide a proper answer we must take inspiration on realistic scenarios faced by buyers and providers. If so, within a given market we expect several producers to offer the very same or similar services (SCOs) at different prices, as well as several consumers to require the very same or similar services (SCOs) valued at different prices. In other words, within a given market we can identify a collection of common services that companies request and offer. For instance, in the example in Figure 8.1, several providers may offer to assemble a cylinder through the very same SCO:

$$t = (6'\text{screws} + 1'\text{cylinder\_line} + 1'\text{cylinder\_rig} + 1'\text{cylinder\_head}, 1'\text{cylinder})$$

Eventually, a provider may either offer to perform such SCO several times (e.g. as many times as cylinders are required), or to bundle it with other SCOs, or the two. Hereafter, we shall consider the common goods and services in a given market to be represented as a collection of SCOs that we shall refer to as *market SCOs*. Therefore, *market SCOs* are equivalent to the *goods* in a combinatorial auction, that is the object providers and buyers can request and offer. Hence, bids for MMUCAs shall be composed as combinations of market SCOs. In this generator, the set of market SCOs is always finite and includes at least two market SCOs for every good in G, ensuring that every good is individually available to buy and/or sell. As an example, Figure 8.2 depicts a sample of market SCOs if intending to build the car engine in Figure 8.1.

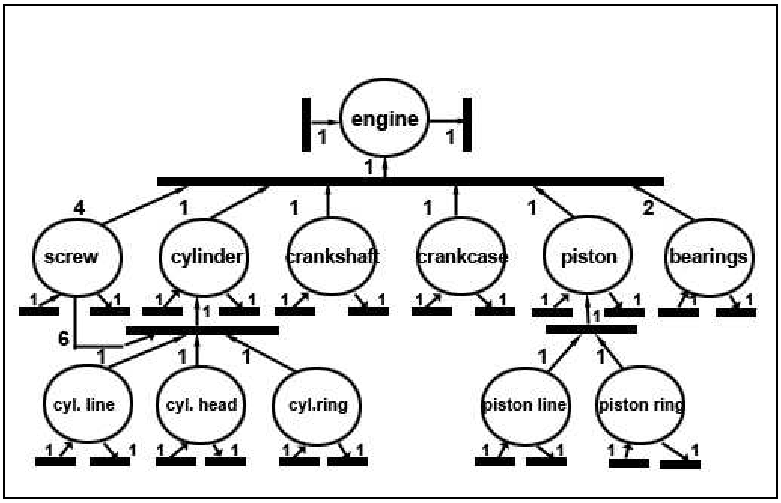


Figure 8.2: Market SCOs for a car's engine.

2. Certain SCOs are more likely to appear together than others. In any market, services and goods are related to each other. For example, the production process for a good can also generate some by-products that can be sold with it or used in another industrial process. Also, some services or products are usually bought together by the final customer.

3. There could be multiple copies of similar SCOs in a bundle. Since bids are composed as combinations of market SCOs, we must introduce the notion of *SCO multiplicity* as the counterpart of good multiplicity (the number of units of a given good within an offer or a request). Say that in a CA a bidder submits a bid for the goods in multi-set $\{2'engine + 1'piston\}$. It is clear that the multiplicity of good *engine* in this bundle is two, whereas the multiplicity of good *piston* is one.

When SCOs are considered things change slightly. In fact, there are two ways to assign a multiplicity to SCOs, one is repeating the SCO several times , and the other one is to simultaneously increase the required input goods and produced output goods while maintaining the same input/output ratio. For instance, consider the following supply chain operation

$$t = (3'a, 2'b) = (\mathcal{I}_t, \mathcal{O}_t) \tag{8.1}$$

Then, we could repeat three times it either offering three times operation t, namely:

$$\mathcal{D} = 3't = 3'(3'a, 3'b) \tag{8.2}$$

or triplicating both the input and output goods:

$$\mathcal{D} = (3 \cdot \mathcal{I}_t, 3 \cdot \mathcal{O}_t) = (9'a, 6'b) \tag{8.3}$$

Notice that the semantics of the two types of multiplicity are different. On the one hand, equation (8.2) means that three copies of the same SCOs are offered and can be *separately* used at three different steps of the solution sequence. On the other hand, equation (8.3) implies that the three copies must be employed at the very same step. That is, while in the former case there can be three different steps in the solution sequence in which 3 copies of "a" are available, in the latter case the bidder needs *nine* copies if "a" available at a given step to perform the operation. Then, we will refer to the multiplicity intended as in equation (8.3) as *repetition multiplicity*, whereas to the one in equation (8.2) as *component-wise multiplicity*.

4. Valuations are related to which SCOs appear in the bundle; furthermore SCO valuations keep consistency with respect to bidder valuations for goods involved in each SCO. A further issue has to do with the way bidders value SCOs and bundles of SCOs. Notice that performing a SCO to assemble the engine in Figure 8.1 results in a new product that has more market value than its parts. Therefore, a car maker values the SCO according to his expected benefits, namely the difference between the expected market value of the engine and the cost of its parts. Therefore, if the parts cost $\$850$ and the expected market value of the engine is $\$1000$, the car maker should be willing to offer to pay less than $\$150$ for the SCO. On the other hand, any provider is expected to request less than $\$150$ in order to perform the SCO. In general, buyers and providers in a MMUCA should value a SCO on the basis of the difference between the expected market value of its output goods and the cost of its input goods. Notice though that we are not assuming here that such difference must always be positive. Likewise bidders should value bundles of SCOs considering the values of SCOs included in it.

5. Appropriate valuations can be configured to be sub-additive, additive or super-additive in the number of SCOs requested. This requirement tries to capture the multiplicity-based (volume- based) discounts policies that are applied in real world. Significant discounts are applied in real markets when goods and services are traded at certain number of units. For example in figure 8.1, we observe that screws are usually traded in higher quantities than full engines. Thus, not surprisingly the same (percentage) discount may apply to an offer for 100 screws than to an offer for 5 engines. Hence, an offer to produce more than 5 engines, being more unlikely, should reflect higher discounts.

6. Sets of XOR'ed bids are constructed on per-bidder basis. We recall from chapter 5 that when a bidder submits different bids in XOR he declares that they are mutually exclusive offers. For example, the following offer

$$\textsc{bid}_1(1'(1'engine, \emptyset), 100) \ \ \textsc{xor} \tag{8.4}$$

$$\textsc{bid}_2(1'(2'engine, \emptyset), 190) \tag{8.5}$$

stands for a bidder that offers to buy two engines or one engine but in any case three engines. On the other hand when a bidder expresses complementarity he translates the OR bids as XOR bids. For example if a bidder wants to buy one engine or one cylinder he submits the following XOR-bid:

$$\text{BID}_1(1'(1'engine, \emptyset), 100) \quad \text{XOR} \tag{8.6}$$
$$\text{BID}_2(1'(1'cylinder, \emptyset), 30) \quad \text{XOR} \tag{8.7}$$
$$\text{BID}_3(1'(1'cylinder + 1'engine, \emptyset), 120) \tag{8.8}$$

As you can observe in both cases bids submitted in the same XOR bid are likely to have similarities and, consequently, combining bids into XOR bids uniformly at random does not capture this property.

7. Unrequested goods by the auctioneer may become involved in the auction. Finally, we add a last requirement that stems from the fact that, unlike auctioneers in CAs, not all goods involved in a MMUCA must be requested by the auctioneer. Back to our example of a car maker in need of engines depicted in Figure 8.1, it can run a MMUCA only requesting engines. Thereafter, bidders may offer already-assembled engines, or other goods (e.g. parts like crankcases, crankshafts, or screws) that jointly with SCOs over such goods help produce the requested goods.

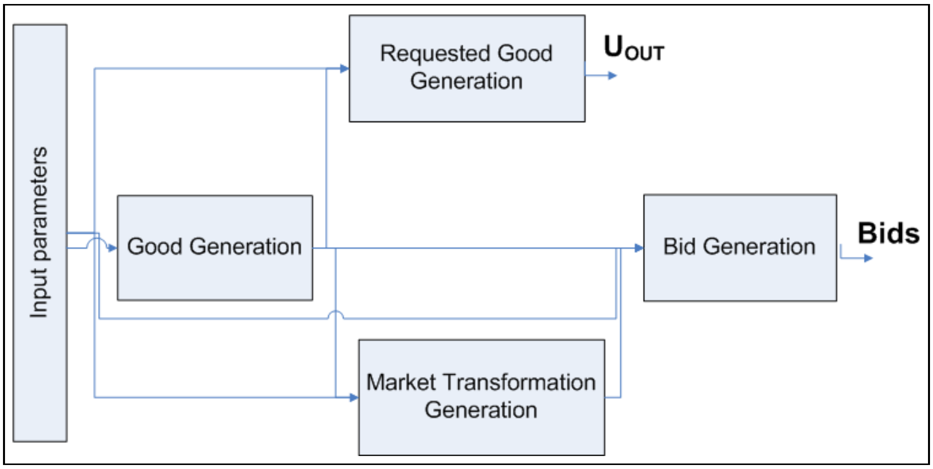


Figure 8.3: Modules of the bid generator and their interaction.

8.2.2 An Algorithm for Artificial Data Set Generation

In what follows we describe a bid generation algorithm that automates the generation of artificial data sets for MMUCA while capturing the requirements above. The algorithm's purpose is to generate MMUCA WDP (each one composed of a collection of XOR bids and the set of goods available to and requested by the auctioneer) that can be subsequently fed into an MMUCA WD algorithm. The algorithm starts by generating the set of goods involved in MMUCA. Next, it generates the goods the auctioneer requests. After that, it creates a subset of atomic SCOs, which are the market SCOs to employ for bid generation. Thereafter, it generates bids as linear combinations of market SCOs, which are subsequently priced according to a pricing policy. The resulting bids are further composed into XOR (mutually exclusive) bids because the XOR language is fully expressive (as proved in section 5.3.6). Hence, the bid generation algorithm[2] assumes that each bidder formulates a single XOR bid, being the number of bidders equal to the number of XOR bids. In figure 8.3 we depict the different modules of the generator and their interaction (Vinyals, 2007a).

Good Generation. This process requires the number of different goods (n_{goods}) involved in an auction along with the maximum price any good can take on ($maxPrice$). Based on these values, it assesses for each good g: (1) its average market price (μ_g) drawn from a uniform distribution $U[1, maxPrice]$ where $maxPrice$ stands for the maximum market price any good can take on; and (2) the distribution to assess its multiplicity, or more precisely, the success probability ($g_{geometric}$) of a geometric probability distribution from which the good multiplicity can be drawn.

Requested Goods Generation. This process assesses the number of units of each good the auctioneer requests, namely the multiset $\mathcal{U}_{out}$. Since the auctioneer must not request all goods, this process selects a subset of the goods in G to be part of $\mathcal{U}_{out}$. Firstly, it determines whether a good g is requested by the auctioneer by comparing the value drawn from a uniform distribution $U[0,1]$ with $p_{good\_requested}$, the probability of adding a new good to $\mathcal{U}_{out}$. Once a given good g is included in $\mathcal{U}_{out}$, the number of units requested for g is drawn from a geometric distribution with the success probability $g_{geometric}$ obtained by the good generation process. Notice that by selecting a subset of the goods we fulfil the requirement 7 listed in section 8.2.1 *unrequested goods by the auctioneer may be involved in the auction.*

Market SCOs Generation. This process generates *a finite set of SCOs* to be employed as the building blocks to subsequently compose bids and consequently fulfilling requirement 1 listed in section 8.2.1. For each good, this procedure constructs two market SCOs, one with only input goods (*I-SCO*) and one with only output goods (*O-SCO*). Each SCO involves a single good with multiplicity one. For instance, $(\{engine\}, \{\})$ and $(\{\}, \{engine\})$ stand respectively for the I- SCO and O-SCO for good $engine$. After that, the algorithm generates a limited number of market SCO (IO-SCOs) with both input and output goods ($n_{IO\_market\_SCOs}$). In order to generate each market IO-SCO, this procedure chooses the goods to include in its input and output set employing the probabilities of adding some good to the input and output set respectively ($p_{good\_in\_input}$ and $p_{good\_in\_output}$). Whenever a good is included to either the input or output set, its multiplicity is calculated from a geometric distribution parametrised by $g_{geometric}$.

Finally, we attach to each market SCO a probability distribution to draw its *component-wise multiplicity*. It is assumed that the bid generation process, detailed by algorithm 1, uses a geometric distribution to calculate the *component-wise multiplicity* of each market SCO. Hence, the generation of market SCOs assesses the success

[2]Here we only provide the bid generation algorithm. The interested reader must refer to (Vinyals, 2007b) for a complete description of all algorithms required by the artificial data set generator.

probability to be employed by such geometric distributions, namely the probability of adding an extra unit of a SCO already included in a bundle bid. Thus, each SCO t is assigned a success probability $t_{geometric}$. However, success probabilities cannot be uniformly at random generated because SCOs are defined over multisets of goods, and therefore consistency must be kept with respect to the success probabilities assigned to each good by the good generation process. Therefore, the success probability for each SCO is set as follows. Given a SCO $t = (\mathcal{I}, \mathcal{O})$, for each good g involved in the SCO, The success probability of t is set to:

$$t_{geometric} = \min_{g \in G} g_{geometric}^{|m_{\mathcal{I}}(g) - m_{\mathcal{O}}(g)|} \tag{8.9}$$

where $m_{\mathcal{I}}(g)$ (respectively $m_{\mathcal{O}}(g)$) stands for the number of occurrences of g in $\mathcal{I}$ (respectively $\mathcal{O}$).

Bid Generation. The bid generation algorithm (algorithm 1) generates bids that are subsequently combined into XOR bids, each one encoding the offer or request of a bidder. This process makes explicit:

(1) which SCOs and how many of them to offer/request in a bundle;

(2) how to price the bundle; and

(3) which bids to combine in an XOR bid.

In what follows we detail each of this functions:

(1) *Selecting the SCOs requested in a bundle and their multiplicities.* Firstly, for each XOR bid ($XORBid$) the algorithm composes each bid (Bid) by combining the market SCOs (MTS) returned by the market SCO generation process. The number of market SCOs ($nTransfBid$) to compose each bid is obtained from a normal distribution $\mathcal{N}(\mu_{add\_new\_SCO}, \sigma_{add\_new\_SCO})$ (line 12).

Market SCOs are chosen from the set of market SCOs (MTS) and their *component-wise multiplicity* in the bundle bid is obtained from a geometric

distribution with success probability $t_{geometric}$ (line 15-16). By assessing the number of units to include in a bundle using a probabilistic distribution that depends on each SCO we partially fulfil requirement 3: *there could be multiple copies of similar SCOs in a bundle*. In fact, in this way SCOs repeated as in equation (8.3) are likely to appear. In this way the authors provide a *repetition multiplicity* associated to SCOs as well.

We also consider that, given an existing bundle, not all SCOs are equally likely to be requested because *certain SCOs (for which complementarities hold) will be more likely to appear together than others*, as stated by requirement 2 in section 8.2.1. To ease these complementarities we assume that the probability of adding a new market SCO to an existing bundle only depends on the last SCO added and not on the whole bundle (Markov property).

It is clear that different copies of the same SCOs may be included in the solution. That is, we may have *repetitions* of SCOs (as the one in equation (8.2)). Then, requirement 3 is completely fulfilled.

(2) *Pricing the bundle.* Next, the algorithm prices the SCO according to its *component-wise multiplicity* (lines 17-21). To fulfil valuations requirements listed in section 8.2.1, a pricing policy must provide the means to price a good,

Algorithm 1 Bid Generation(MTS, n_{XOR_bids}, μ, σ_{prices}, $\mu_{add\_new\_XOR\_clause}$, $\sigma_{add\_new\_SCO}$, $\mu_{add\_new\_SCO}$, $\sigma_{add\_new\_SCO}$, α)

```
 1: for $g = 1$ to $n_{goods}$ do
 2:   for $b = 1$ to $n_{XOR\_bids}$ do
 3:     $p_{prices\_bid}[b, g] \leftarrow \mu[g] \cdot N(1, \sigma_{prices})$
 4:   end for
 5: end for
 6: $Bids \leftarrow \emptyset$
 7: for $b = 1$ to $n_{XOR\_bids}$ do
 8:   $XORBid \leftarrow EmptyXORBid()$
 9:   $nXORClauses \leftarrow \mathcal{N}(\mu_{add\_new\_XOR\_clause}, \sigma_{add\_new\_XOR\_clause})$
10:   for $x = 1$ to $nXORClauses$ do
11:     $Bid \leftarrow EmptyCombinatorialBid()$
12:     $nTransfBid \leftarrow \mathcal{N}(\mu_{add\_new\_SCO}, \sigma_{add\_new\_SCO})$
13:     if $x == 1$ then
14:       for $t = 1$ to $nTransfBid$ do
15:         $MT \leftarrow$ Select a SCO using Markov model from $MTS$ with state $MT$
16:         $multiplicity \leftarrow Geometric(MT.t_{geometric})$ bid $B$.
17:         $T.inputs \leftarrow MT.inputs \cdot multiplicity$
18:         $T.outputs \leftarrow MT.outputs \cdot multiplicity$
19:         $T.price \leftarrow \sum_{g \in T.outputs} p_{prices\_bid}[b, g] - \sum_{g \in T.inputs} p_{prices\_bid}[b, g]$
20:         $p_{offer} \leftarrow (T.t_{geometric})^{multiplicity}$
21:         $discount \leftarrow \alpha \frac{1-e^{1-p_{offer}}}{1-e}$
22:         $Bid.t \leftarrow Bid.t \cup T$
23:         $Bid.price \leftarrow Bid.price + T.price \cdot (1 - discount)$
24:       end for
25:     else
26:       $model$ ←Uniformly At Random generate a number between 1 and x-1
27:       $Bid \leftarrow XORBid(model)$
28:       if $nTransfBid \geq length(XORBid(model).t)$ then
29:         $Bid \leftarrow removeRandomTransition(Bid)$
30:         $Bid \leftarrow recalculatePrices(Bid)$
31:       end if
32:       if $nTransfBid \leq length(XORBid(model).t)$ then
33:         $Bid \leftarrow addRandomTransition(Bid)$
34:         $Bid \leftarrow recalculatePrices(Bid)$
35:       end if
36:     end if
37:     $XORBid \leftarrow XORBid \cup \{Bid\}$
38:   end for
39:   $Bids \leftarrow Bids \cup \{XORBid\}$
40: end for
41: return Bids
```

a SCO, multiple units of the very same SCO, and a bundle of SCOs in a realistic manner. As to pricing goods, in order to vary prices among bidders, the algorithm generates a price for bidder b for good g, represented as $p_{prices\_bid}[b,g]$, from a normal distribution $\mathcal{N}(\mu[g], \sigma_{prices})$, where $\mu[g]$ stands for good g's average price in the market and σ_{prices} for the variance among bidders' prices (lines 2-4). Thereafter, a SCO's price for bidder b is assessed in terms of the difference from his valuation of its output goods with respect to his valuation of its input goods (line 19). Accordingly, *SCO valuations keep consistency with respect to bidder valuations for goods involved in each SCO* as stated by requirement 5 in section 8.2.1. Each bid valuation is obtained by adding the prices of its SCOs (line 23). Hence *valuations are related to which SCOs compose the bundle* as stated by requirement 6 although varying among different bidders. Furthermore we propose to introduce super-additivity by applying multiplicity-based discounts to SCOs addressing the requirement that *valuations can be configured to be sub-additive, additive o super-additive in the number of SCOs requested.* In other words, as a general rule, the more unlikely for a SCO to be traded at certain units (multiplicity), the higher the discount to apply to its overall price. In this way we try to capture in a realistic manner the way multiplicity-based (volume-based) discounts are applied in the real world. Therefore, given SCO t, we firstly assess the probability p_{offer} of the SCO to be traded with *component-wise multiplicity* m from a geometric distribution with success probability $t_{geometric}$ as follows: $p_{offer} = t_{geometric}{}^{multiplicity}$ (line 20). Secondly, we compute the discount to apply ($discount$) as follows: $discount = \alpha\frac{1-e^{1-p_{offer}}}{1-e}$. Indeed, in this way we manage to apply higher discounts to more unlikely offers within the range $[0, \alpha]$. Notice too that setting α to zero leads to no discounts, and thus to no super-additivity.

(3) *Which bids to combine in an XOR bid.* Finally, after creating each bid, the algorithm adds it to the XOR bid under construction (line 37). The number of bids that compose an XOR bid is obtained from a normal distribution $\mathcal{N}(\mu_{add\_new\_XOR\_clause}, \sigma_{add\_new\_XOR\_clause})$ (line 9). We consider here requirement 7 listed in section 8.2.1 and since different bids in XOR-relationships stand for different alternatives or options for the bidder we propose to generate similar bids for the same XORBid. The first bid of each XORBid is generated uniformly at random (lines 13-24) whereas the rest of bids are created applying some modifications over one existing bid in the bundle (lines 25-36). The number of modifications depends on the difference between the number of SCOs assigned to the new bid and the existent one:

- if it is less we remove randomly one SCO;
- if it is greater we add uniformly at random new SCOs; and
- if it is equal we apply once both operations.

In all cases we finally recalculate the prices following the proposed price policy. Hence the requirement that *sets of XOR'ed bids are constructed on a per-bidder basis* is fulfilled.

n_{goods}	20
$n_{IO\_market\_SCOs}$	$n\_SCOs/3$
max_price	100
σ_{prices}	0.05
$p_{good\_requested}$	0.3
$\mu_{add\_new\_SCO}$	1.0
$\sigma_{add\_new\_SCO}$	0
$\mu_{add\_new\_XOR\_clause}$	1.0
$\sigma_{add\_new\_XOR\_clause}$	0
$p_{good\_in\_input}$	0.2
$p_{good\_in\_output}$	0.1
α	0.1
p_ISCOs	0.6
p_OSCOs	0.1
allow_cycles	1

Table 8.1: Artificial generator parameter values.

8.3 Empirical Evaluation

In this section, we firstly provide a preliminary comparison of DIP and CCIP on arbitrary supply chain network topologies. Next, we run the CMWOSP-based solver on acyclic network topologies.

8.3.1 DIP versus CCIP

In what follows, we provide a preliminary experiment to quantitatively assess the size of the supply chain formation scenarios that CCIP allows to solve compared to DIP.

According to (Hillier and Lieberman, 1986), the number of decision variables is a good index of the difficulty of an optimisation problem (although not the only one). Since the number of decision variables of both DIP and CCIP depend on the number of SCOs within the submitted bids, in order to compare their computational performances and to analyse their scalability, we have chosen to observe their solving times as the number of SCOs increases. In order to compare the DIP and CCIP MMUCA WD algorithms, we have employed randomly generated MMUCA WDPs using the artificial data set generator presented in section 8.2. We have set the generator parameters for this experiment as listed in Table 8.1.

We ran our experiments as follows. We generated MMUCA WDP instances with SCOs within the range $[0, 300]$. We sampled the interval to generate 50 WDP instances every 20 SCOs. Both solvers DIP and CCIP were fed with the very same WDP instances. We solved each WDP instance using implementations of both solvers on CPLEX 10.1 (ILOG, 2007), recording both the solutions and solving times. Moreover, we set a maximum time limit to 4800 seconds for each solver to find a solution for each WDP instance. Whenever any of the solvers exceeded the time limit, we marked the WDP as *time exceeded* and assigned. After that, we set its solving time to the time

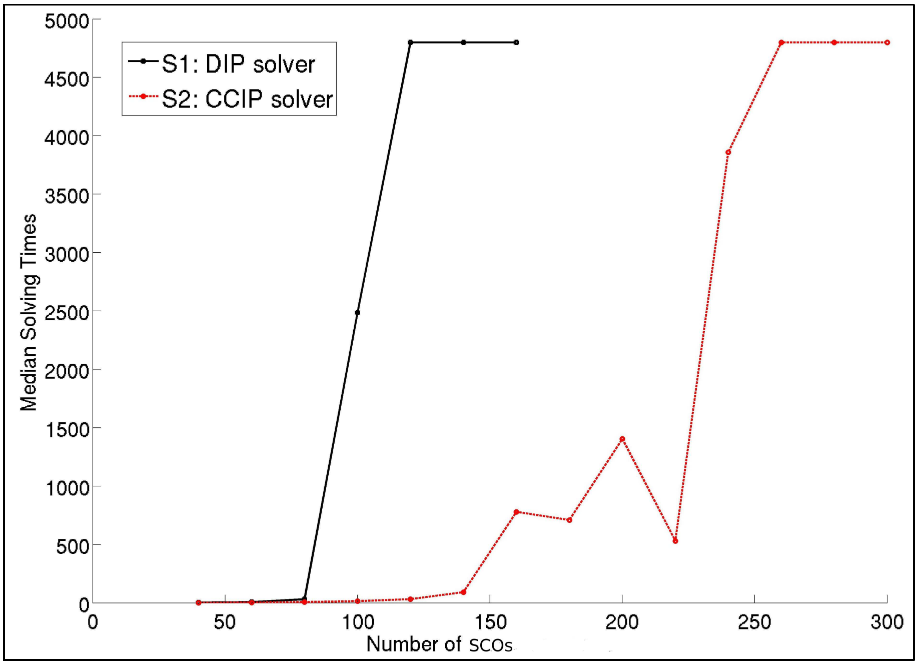


Figure 8.4: Comparison between DIP and CCIP.

limit to subsequently record it. Notice that we only considered feasible WDP instances to calculate solving times since the time required by CPLEX to prove unfeasibility is (usually) significantly lower than the time required to find an optimal solution. Finally, we ran all tests on a Dell Precision 490 with double processor Dual-Core Xeon 5060 running at 3.2 GHz with 2Gb RAM on a Linux 2.6.

Figures 8.4 and 8.5 summarise the results of this experiment. Figure 8.4 depicts the median of the solving times obtained when varying the number of SCOs. Figure 8.5 shows for both DIP and CCIP the number of instances that have been solved within the time limit. Then, given a time limit, CCIP was able to solve problems with more than twice the number of SCOs than DIP did solve. Indeed, whereas 120 represents the empirical limit ($\sim$50% of solved instances) on the number of SCOs for DIP, CCIP starts reaching the time limit when solving WDP instances containing more than 250 SCOs. Furthermore, for WDPs with close to 100 SCOs, DIP is in median about 70 times slower than CCIP. This ratio rapidly increases as the number of SCOs gets close to 120 in the presented scenario.

The observations stemming from this experiment are very promising. They indicatethat we can obtain substantial reductions in the solving time when employing CCIP instead of DIPdepending on the features of the scenario. The search space reduction obtained with solver CCIP translates into a significant decrease in computational solving time complexity.

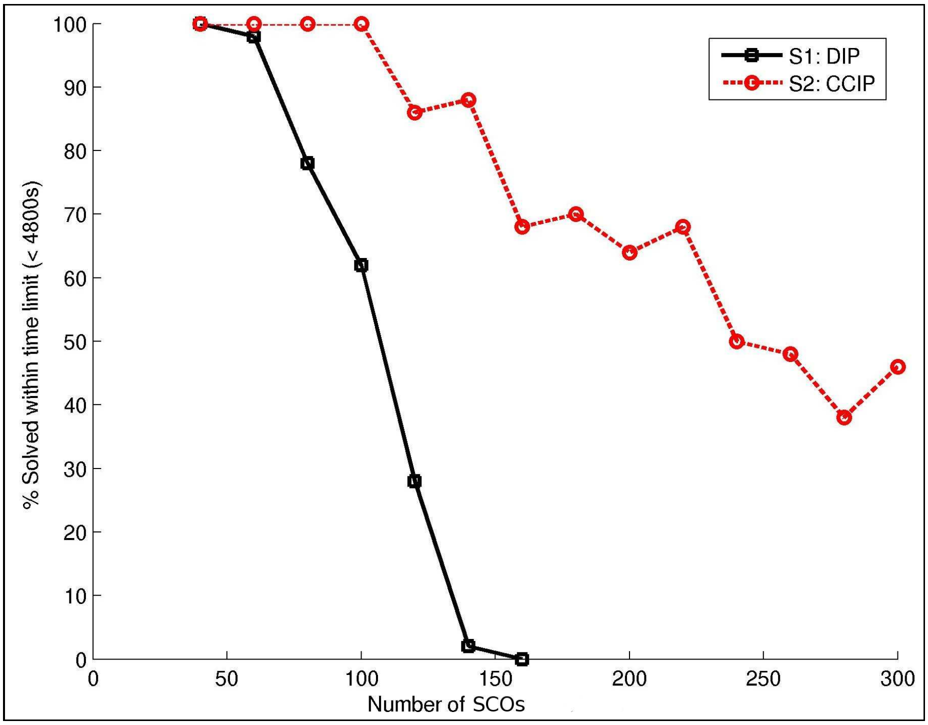


Figure 8.5: Number of instances solved within the time limit (4800 sec.).

8.3.2 Performances of the CMWOSP-based solver

In this section we aim at testing the performances of the CMWOSP algorithm (that can be used only when the *Mixed Auction Net* is acyclic). Recall that (section 7.3.3) at the theoretical level the CMWOSP-based and CCIP solvers are equivalent when the *Mixed Auction Net* is acyclic. However, if we employ CCIP we should compute the strongly connected components beforehand. In this case, since we a-priori know that the *Mixed Auction Net* is acyclic, we directly employ the CMWOSP-based solver.

We have run this experiment as described in section 8.3.1, but enforcing the bid generator to build instances with no cycles. Figure 8.6 shows the CPU time required to solve problem instances on acyclic nets for the CMWOSP-based solver.

Notice that the axis time scale in figure 8.6 is nearly four orders of magnitude smaller than in figure 8.4. Hence, cycles in the mixed auction net may lead to a significant increase in computational cost.

8.4 Conclusions

In this chapter we presented a preliminary empirical comparison of the CMWOSP-based, DIP and CCIP solvers. Firstly, we compared DIP and CCIP on any type of

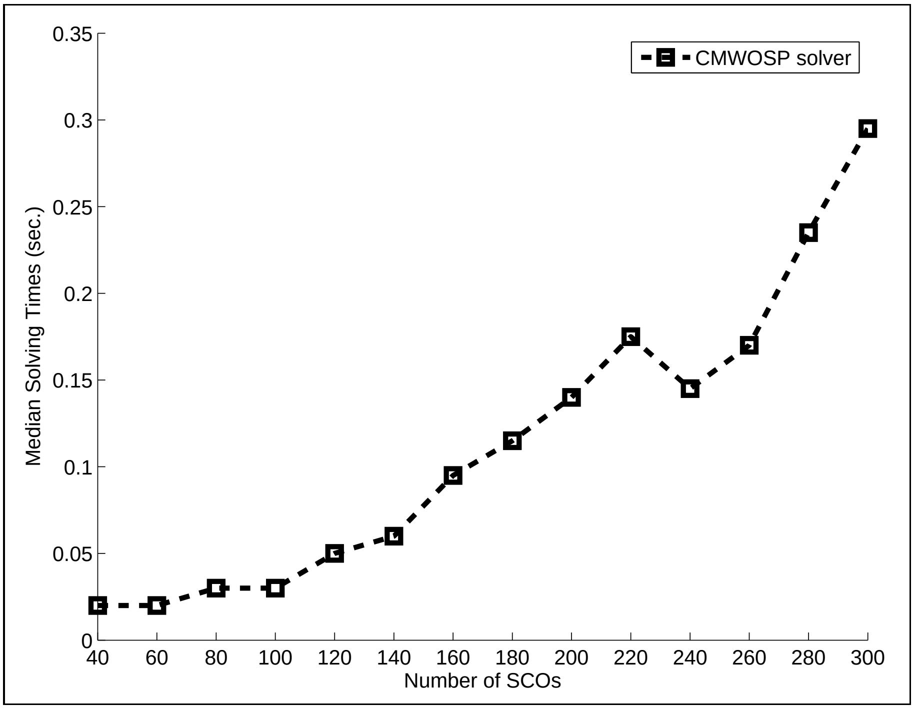


Figure 8.6: Experiments with acyclic network topologies (reduced time scale).

network topology. Next, we empirically assessed the performances of the CMWOSP-based solver on acyclic *Mixed Auction Nets*.

In chapter 7, we proposed CCIP, a solver that dramatically improves the computational efficiency of DIP by taking advantage of the topological characteristics of WDPs. At the theoretical level, we proved that CCIP brings a drastic reduction in the number of decision variables required to solve the WDP. In this chapter, we have empirically observed that in the presented scenario CCIP:

(1) can deal with WDPs with more than twice SCOs than DIP;

(2) can significantly reduce the computation time (by a factor larger than 70).

Finally, we observed that in the considered parameter setting the CMWOSP-based solver is four orders of magnitude faster in solving acyclic instances than DIP and CCIP in solving instances produced by Vinyals' generator.

Chapter 9

Conclusions and Future Work

In this chapter, we draw some conclusions about the work developed in this dissertation and we show some open paths to future development.

9.1 Conclusions

Most of the currently studied and employed combinatorial auctions deal with the negotiation of goods, disregarding eventual production relationships holding among them. The information about such relationships helps improve the outcome of a negotiation. In order to fill this gap, we introduced two novel combinatorial auction extensions that help in determining the revenue-maximising strategy for partner selection in supply chain network design and planning. The former, called *Multi Unit Combinatorial Reverse Auctions with Transformability Relationships among Goods (MUCRAtR)*, copes with *make-or-buy* decisions. The latter, called *Mixed Multi-unit Combinatorial Auctions (MMUCA)*, deals with *make-or-buy-or-collaborate* decisions. Below, we separately summarise our two contributions.

9.1.1 Make-or-Buy Decisions

In chapter 1, we thoroughly described the requirements that must be fulfilled in order to solve *make-or-buy* decision problems. Since we built upon combinatorial auctions, we also explained the of CAs limitations that hinder their application to our problem. In table 9.1 we recall both the requirements and the corresponding CAs limitations associated with the *make-or-buy* decision problem. We observed that all the CAs limitations stem from the fact that they can neither express nor represent an auctioneer's internal manufacturing operations.

The first requirement hindering the application of CAs to our problem is that they can neither represent internal manufacturing operations nor the producer/consumer relationships among them. In order to apply CAs to solve the *make-or-buy* decision problem, we provided a formal framework to represent internal manufacturing operations. At this aim we decided to employ Place/Transition Nets (PTNs) (Reisig, 1985) because:

	Requirements	**CAs**	**MUCRAtR**
1	express a request on bundles of goods	✓	✓
2	express an auctioneer's initial stock		✓
3	express producer/consumer relationships among internal operations		✓
4	specify an auctioneer's final requirements		✓
5	express relationships among manufacturing operations, auctioned goods, and received bids		✓
6	formally and graphical represent the search space associated to the auctioneer's decision problem		✓
7	specify the auctioneer's internal cost structure		✓
8	information about which in-house operations to perform and in which order		✓

Table 9.1: Requirements of to the *make-or-buy* problem.

(1) they naturally help us capture the notion of manufacturing operation;

(2) they have a well-defined semantics that can naturally accommodate the notion of sequence of operations and consumer/producer relationships;

(3) they have an integrated description of both states and actions to characterise the search space where operations occur;

(4) they have a large number of formal analysis methods that allow the investigation of structural and behavioural (dynamic) properties of the net; and

(5) they have a graphical representation that is intuitively very appealing to study problems related to the topology of the supply chain.

Thus, we modelled the internal production structure of an auctioneer by means of a PTN, that we referred to as PTN_I. Not only does this formal representation allow us to describe the quantity of resources either produced or consumed by a manufacturing operation, the producer/consumer relationships among operations, and the quantity of goods available to an auctioneer after each operation, but it also allow us to express preconditions over a manufacturing operation by means of a *firing rule*. By the application of the firing rule, we impose that a manufacturing operation can *only* be run if its input goods are available. This property is critical for the correct representation of a production process: the implementation order of a production process is constrained by the availability of resources at each step.

Then, a PTN_I completely specifies an auctioneer's internal manufacturing operations and the producer/consumer relationships among them (requirement (5) in table 9.1). Moreover, a PTN_I allows an auctioneer to specify his requirements and communicate them to bidders (requirement (4) in table 9.1). This is obtained by specifying a configuration (marking) to end up with. If an auctioneer communicates to a set of bidders his PTN_I along with a description of the final state of such PTN describing his requirements, then the bidders can infer all the possible configurations of offers fulfilling such requirements.

Next, in order to express the relationships among internal manufacturing operations, auctioned goods, and received bids (requirement (3) in table 9.1), we incorporated the received bids into PTN_I. At this aim, we exploited the fact that a bid that offers goods can be regarded as a transition (*bid transition*) that injects tokens into PTN_I. Unlike transitions corresponding to manufacturing operations, bid transitions do not consume input resources and can be fired only once. The PTN_I augmented with bid transitions was called PTN_E (where E stands for *Extended*).

By means of a PTN_E, an auctioneer can compactly express all the possible outcomes of any of his possible decisions. By decision we mean the selection of bids together with a sequence of internal operations to perform. Thus, a PTN_E both formally and graphically represents the search space associated to the auctioneer's decision problem (requirement 6 in table 9.1). We successfully linked bids, manufacturing operations, and goods at auction, and we fully represent all the possible decisions an auctioneer may take in a unified representation.

However, the goal of the auctioneer is not only to find a feasible outcome, but also to find an outcome that minimises his costs. Thus, an auctioneer needs to quantify the cost associated to each decision. With this aim, he has to associate a cost to the selection of a bid, and a cost to the performance of a manufacturing operation. Unfortunately, in their original definition, place transition nets do not incorporate the notion of cost associated to the firing of a transition. Then, we defined a new type of Place Transition Net, the so-called *Weighted Place Transition Nets* (WPTN), to express the notion of cost associated to transition firings or to the firing of sequences of transitions.

Then, we transformed both PTN_I and PTN_E into WPTNs by associating to each *operation transition* the cost of the corresponding manufacturing operation and to each *bid transition* the bid cost. The resulting WPTNs were called *Transformability Network Structure* (TNS) and *Auction Net* respectively: a TNS completely describes an auctioneer's internal manufacturing operations, whereas an Auction Net compactly represents the set of possible auctioneer's decisions along with the corresponding cost. Then, *Transformability Network Structure* and *Auction Net* allow the auctioneer to express his internal cost structure and to incorporate it into his decision problem (requirement (7) in table 9.1).

The auctioneer needs to select the set of offers along with the sequence of internal manufacturing operations to perform that minimise his costs and allow him to obtain his final requirements (point (8) in table 9.1). With this purpose, we defined the auctioneer's decision problem as an optimisation problem on the *Auction Net*. Thus, we introduced a new type of reachability problem over WPTNs, and called this new optimisation problem the *Constrained Maximum Weight Occurrence Sequence Problem* (CMWOSP). Intuitively, this optimisation problem involves finding an optimal cost sequence of transitions on a WPTN that leads to a final state which fulfils some constraints.

Additionally, we provided an important result on the CMWOSP. We showed that the CMWOSP can be solved by means of Integer Programming on acyclic WPTNs, namely on WPTNs that do not contain any directed cycle.

The CMWOSP perfectly captures the semantics of the auctioneer's decision problem in a MUCRAtR: to find the set of *bid and operation transitions* that minimises an auctioneer's revenue. Thus, we formalised the auctioneer's decision problem in a MUCRAtR as a CMWOSP on the Auction Net. Two major benefits, and therefore contributions, stemmed from the formalisation of the MUCRAtR WDP by means of a CMWOSP:

(1) the CMWOSP provides as a result both the set of bids to accept and the sequence of operations to perform in order to obtain the auctioneer's final requirements (requirement (8) in table 9.1);

(2) the *make-or-buy* decision problem can be solved by means of Integer Programming for a large class of supply chain network topologies (acyclic).

Summarising, we provide to the auctioneer with a formalism to express his requirements and communicating them to bidders; and a rule for determining the optimal allocation, i.e. the set of winning bids and the sequence of internal operations to perform. In this way we provide a solution to all the requirements needed for extending combinatorial auctions for dealing with the *make-or-buy* decision problem.

The solution to the WDP that we provide can be employed as a decision support system in different settings:

- *Combinatorial auctions*. As a winner determination solver in a MUCRAtR.
- *Negotiation*. A buyer, after receiving a set of offers from his providers, can compute the best offers and eventually counter-offer.
- *What-if supply chain analysis*. A buyer, aware of the prices and capacities of his providers, can test different configurations of his supply chain.

Summary of MUCRAtR contributions

To summarise, the main contributions related to the *make-or-buy* decision problem are:

- **MUCRAtR**, an extension of combinatorial auctions that allows dealing with *make-or-buy* decision problems in scenarios characterised by combinatorial preferences. This new auction type provides an auctioneer with a framework to optimise his outsourcing strategy.
- **Weighted Place Transition Nets (WPTN)**, an extension of Place Transition Nets. In WPTNs it is possible to associate a weight (cost) to the firing of each transition. WPTN is a formal framework introduced to represent the MUCRAtR space of auctioneer's decisions and associated revenues.
- **Constrained Maximum Weighted Occurrence Sequence Problem (CMWOSP)**, a new reachability problem defined on WPTNs. It formalises the problem of selecting a cost-maximising sequence of actions leading from an initial state to a set of possible final states.
- **ILP solution to the CMWOSP**. We prove that the CMWOSP can be solved by means of ILP when the underlying WPTN is acyclic. We obtain this contribution by exploiting results imported from the literature on Place Transition Nets.

- **Formalisation of the MUCRAtR WDP as a CMWOSP on an Auction Net**. We show that CMWOSP perfectly captures the features of the auctioneer's decision problem in a MUCRAtR. This results provides three important benefits: (1) an ILP formulation for a wide class of MUCRAtR WDPs (acyclic); (2) a formalism (PTN) to analyse the decision problem; and (3) the result of the WDP provides both the set of selected bids and the *sequence* of operations to perform.

9.1.2 Make-Or-Buy-Or-Collaborate

In the second part of this dissertation we dealt with the *make-or-buy-or-collaborate* decision problem. As thoroughly explained in section 1.4.2, most of the requirements arising in the *make-or-buy-or-collaborate* decision problem are currently not supported by state-of-the-art methodologies and tools. In table 9.2 we summarise the requirements and the limitations of two state-of-the-art solutions, namely combinatorial auctions and task dependency networks.

MMUCA

In order to overcome such limitations, we introduced an extension to combinatorial auctions, called *Mixed Multi-unit Combinatorial Auctions* (MMUCA), that fulfils all the requirements of this decision problem.

	Requirements	**CAs**	**TDN**
1	express an offer/request on bundles of goods	✓	✓
2	express an offer of a SCO with a single output product		✓
3	express an offer of a SCO with multiple output products		
4	express a request of a SCO		
5	express the offer/request of a bundle of SCOs		
6	express combinations of bids	✓	
7	express the min/max number of times SCOs are performed		
8	express resource sharing		
9	express an auctioneer's initial stock		
10	express the auctioneer's final requirements		
11	support *acyclic* supply chain networks		✓
12	support *cyclic* supply chain networks		
13	compute the *scheduled sequence* of SCOs to perform		
14	ensure computational tractability while preserving optimality		
15	solve SCF decision problem		✓
16	solve the *make-or-buy-or-collaborate* decision problem		
17	formally represent the search space		
18	graphically represent the search space		
19	assess the computational tractability based on the problem structure		

Table 9.2: Requirements of the *make-or-buy-or-collaborate* problem.

MMUCAs support the trading of operations across the supply chain: from the supply and demand of components to the supply and demand of manufacturing operations or services. With the aim of making MMUCA operative:

(1) we provided a formal language allowing bidders to express offers and requests over supply chain operations; and

(2) we formalised the optimisation problem faced by an auctioneer aiming at selecting the subset of the offered supply chain operations maximising his revenue.

As to the formal language, we introduced a general-purpose concept that can represent any operation or service negotiated across the supply chain by any supply chain stakeholder, the so-called *supply chain operation* (SCO). The characterising features of SCOs are the required and consumed input resources and the output resources produced by the service. According to requirements (1-8) in table 9.2, the different actors involved in a MMUCA require a language to express the offer/request of supply chain operations. Then, we extended traditional bidding languages in the literature to deal with *SCOs*.

Bidding Language

We set SCOs as the atomic entities that can be negotiated across a supply chain. Building upon such building blocks, we defined a new bidding language that allows bidders:

(1) associating a value to bundles of SCOs within an *atomic bid*; and

(2) combining *atomic bids* into complex expressions encoding a wide variety of preferences over SCOs.

The provided bidding language generalises state-of-the art bidding languages and provides supports to express bids in the following auction types:

- *Multi-unit* combinatorial auctions, where there may be several indistinguishable copies of the same good available in the system.

- *Double auctions* where there are multiple buyers and multiple sellers. We integrate direct and reverse auctions, *i.e.* the auctioneer will be able to both sell and buy goods within a single auction. Or considering the bidders' point of view, a bidder can submit both offers and demands on sets of goods.

- *Combinatorial exchanges*. Combinatorial case of double auctions. In this auction type both buyers and seller submit combinatorial bids.

- *Multi-unit Combinatorial Reverse Auctions with Transformability Relationships among Goods* (MUCRAtR). We integrate the notion of internal manufacturing operation into MMUCAs.

- *Combinatorial auctions for supply chain formation* introduced in (Walsh et al., 2000).

The novelty of our bidding language with respect to the above-mentioned auction types is that we further extend the idea of manufacturing operations by allowing agents to also bid for *supply chain operations*. Since in our language a bidder is allowed to bid over *bundles* of supply chain operations, such language captures potential complementarities among such operations. This extension offers a higher degree of expressiveness and allows to generalise bidding languages for the above-listed auction types.

Summarising, the proposed bidding language can express several types of complex bids and allows for bids on bundles of SCOs. By means of the introduced bidding language we overcome requirements (1-8) in table 9.2.

The Winner Determination Problem

As to the Winner Determination Problem, we cannot rely on previous definitions in the literature. According to requirement (13) in table 9.2, and similarly to MUCRAtR, a new dimension comes into play: the *order* among SCOs. For this reason, we provided a new and general definition of winner determination problem that builds upon our SCO-based bidding language. The WDP describes the rules to:

- select the winning bids that maximise an auctioneer's revenue; and
- assess the *execution order* of the SCOs contained in the winning bids.

Notice that the winning bids are those that:

- they fulfil the constraints specified by bidders via the bidding language;
- they maximise the auctioneer revenue.

and the sequence of SCOs representing the execution order is such that:

- it contains all and only the SCOs included in the winning bids;
- it is implementable, i.e. each SCO in the sequence must have its required inputs available at the position where it is scheduled; and
- it produces at its end at least the set of goods required by the auctioneer.

Observe that the *order* in which agents consume and produce goods is of central importance in our model and affects the definition of the winner determination problem.

The rule to assess the set of winners provides a solution to requirements 10, 13, 15, and 16 in table 9.2. By including the constraint that the goods available after running all the selected supply chain operations are the ones specified by the auctioneer, we provided a solution to requirement (9) in table 9.2. Since the definition does not depend on the particular topology of the supply chain network, we provided a solution to requirement (11-12) in table 1.2.

The new WDP definition extends and generalises the definition of winner determination for:

- *Multi-unit combinatorial auctions*

- *Double auctions*
- *Combinatorial exchanges*
- *MUCRAtR*
- *Combinatorial auctions for supply chain formation*

Thus, we provided both a bidding language and a definition of winner determination problem that extends and generalises all the above-mentioned auction types. The bidding language along with the winning rule fully characterises *Mixed Multi-unit Combinatorial Auctions* (MMUCAs).

A mathematical framework for the MMUCA WDP

Analogously to the MUCRAtR WDP, we provided a mapping of the MMUCA winner determination problem to a *Constrained Maximum Weight Occurrence Sequence Problem* on the *Mixed Auction Net*. Via this mapping, we obtained the same advantages as in the case of MUCRAtR: (1) the solution is expressed as a *sequence* of SCOs; (2) we provide a formal framework to analyse the properties of the decision problem; and (3) we obtain an ILP-based formulation of the MMUCA WDP for acyclic Mixed Auction Nets.

The *Mixed Auction Net* provides a formalism to reason about MMUCAs, and therefore also about all the WDPs associated to auctions subsumed by MMUCA (requirements (17-18) in table 9.2). In particular, we showed that the *Mixed Auction Net* subsumes the TNS and the Transformability Network Structure (Walsh and Wellman, 2003).

Solving the MMUCA WDP

In this dissertation we provided three different IP solvers for computing the solutions to the MMUCA WDP. The first one is based on the mapping to CMWOSP. This solver deals with acyclic supply chain network topologies. The second one (DIP) was directly built upon the definition of MMUCA WDP and applies to arbitrary network topologies. The third one (CCIP) improves the performances of the second solver by exploiting some domain knowledge.

The CMWOSP-based Solver. By mapping to CMWOSP we obtain an ILP-based formulation of the MMUCA WDP with integer programming for a wide class of WPTN topologies.

The DIP Solver. We showed that restricting the Mixed Auction Net to be acyclic is a significant limitation in some scenarios: it does not allow representing cyclic operations, resource sharing, and so on. We provided a new IP formulation, called *Direct Integer Programming* (DIP) solver, that is directly built upon the definition of MMUCA WDP. DIP solves the WDP associated to any supply chain network topology, thus broadening the classes of solvable problems.

The CCIP solver. The main drawback of DIP is that it generates a number of decision variables and constraints that limit its applicability to small-size and medium-size scenarios. DIP guarantees optimality, but decrements the computational tractability. We proposed an ILP-based formulation for MMUCA WDP, namely the *Connected Component Based Integer Programming* Solver (CCIP), that dramatically improves the computational efficiency of DIP. A search space reduction is achieved by analysing and exploiting the precedence relationships among SCOs.

We conclude by observing that our approach solves some of the problems related to centralised approaches to supply chain formation and scheduling (see section 3.3.1). Firstly, we can reduce the complexity associated to optimise the scheduling problem. In fact, we have that

- the complexity of the scheduling problem is reduced due to the absence of time dimension, without losing the possibility to express precedence relationships among operations; and
- we provide a very efficient optimisation problem solver (CCIP).

Secondly, agents are not forced to reveal all their information truthfully. The part of information revealed by agents (the bidders) is regulated by the bidding language and the market-based mechanism. In market-based mechanisms agents can act strategically, hide or lie on critical information, decide what to communicate and what not.

Empirical Evaluation

In the last part of this dissertation we provided a preliminary proof of concept about the performances of the CMWOSP-based, DIP and CCIP solvers in a single scenario. On the one hand, we compared DIP and CCIP on arbitrary network topologies. On the other hand, we empirically assessed the performances of the CMWOSP-based solver on acyclic *Mixed Auction Nets*. We observed that in the considered parameter setting:

- CCIP outperforms DIP
- acyclic instances are much easier to solve

Summary of MMUCA Contributions

To summarise, the contributions in this dissertation related to the *make-or-buy-or-collaborate* decision problem are listed in what follows.

- **MMUCA** is a new type of auction that allows to deal with *make-or-buy-or-collaborate* decisions. This new auction type provides an auctioneer with a framework to optimally select supply chain partners. MMUCA generalises and extends several types of auctions (including MUCRAtR). Our contribution develops along two dimensions:
  - **MMUCA Bidding Language**. We provide a novel bidding language that allows agents to trade any type of operation across the supply chain. Such a language extends and generalises several previous bidding languages.

  - **MMUCA Winner Determination Problem**. We provide a definition of winner determination problem that selects, among the received bids, the revenue-maximising *ordered sequence* of supply chain operations to perform. The definition of MMUCA WDP extends and generalises the definition of winner determination problem of several existing auction types.

- **Mixed Auction Net** is a WPTN that compactly represents the space of possible decisions an auctioneer may take, along with the revenue associated to each decision. Thus, it formally and graphically represents the search space associated to the MMUCA WDP, and therefore of the *make-or-buy-or-collaborate* decision problem.

- **Mapping the MMUCA WDP to the CMWOSP**. We succeeded in mapping the MMUCA WDP to a CMWOSP on the *Mixed Auction Net*. As in the case of MUCRAtR, three benefits stemmed from this mapping: (1) the provided solution is a sequence of SCOs; (2) a whole corpus of theoretical results from the PTN literature can be imported to analyse the decision problem; and (3) we obtain an ILP formulation of the WDP for acyclic *Mixed Auction Nets*.

- **MMUCA WDP Solvers**.
  - **CMWOSP-Based Solver**. This ILP-based solver is based on the mapping of MMUCA and MUCRAtR to a CMWOSP and applies only to acyclic mixed auction nets.
  - **DIP solver**. This ILP-based solver works on arbitrary supply chain network topologies. It overcomes a set of limitations connected with the use of the CMWOSP-based solver.
  - **CCIP solver**. This ILP-based solver dramatically improves the performances of DIP solver because it allows a more concise representation of the optimisation problem. This is obtained by exploiting the precedence relations among supply chain operations.

  Finally, consider that MUCRAtR and Combinatorial Auctions for supply chain formation are a special case of MMUCA. Thus, the three solvers presented above can be used to solve problems on any network topologies for them as well. Thus, besides broadening the applicability of MMUCAs, we have also broadened the applicability of MUCRAtR and CAs for SCF. MUCRAtR and CAs for SCF can be extended to any network topology!

9.2 Future Work

We believe that this dissertation opens several paths to future developments. The most interesting extension we envisage to MMUCAs is the incorporation of time and uncertainty in the MMUCA model.

On the one hand, we envisage the possibility to express the release time and duration of a supply chain operation. This information should be also included within the winner determination problem. In this way, an auctioneer would be able to fix deadlines to have his production process completed. Moreover, the participants to the supply chain would synchronise their operations by fulfilling not only the producer/consumer relationships, but also eventual time constraints.

On the other hand, an auctioneer may be interested in assigning a success probability to each supply chain operation. In this way, he could minimise the incidence of failures and shortcoming across the supply chain.

Next, in order to outperform CCIP we plan to explore the design of a local algorithm. Although solutions may be sub-optimal with a local approach, the number of transformations that can be dealt with is expected to be larger, and hence the size of the supply chain scenarios we could tackle.

Furthermore, a more realistic setting requires to incorporate logistic providers, besides component suppliers, contract manufacturers, and final customers. We do believe that, by relying on the intuitions provided by the graphical representation of WPTNs, we can easily incorporate constructs dealing with this kind of problems. Along the same line, we aim at assessing the value of our approach in actual scenarios with real-world data (for instance in the automotive industry). If this was not possible, we plan to improve the artificial bid generator summarised in section 8.2 by incorporating actual-world supply chain topologies, following the strategy of (Leyton-Brown and Shoham, 2006). Furthermore, we need to perform extensive experiments with different parameter settings in order to empirically assess the improvement of CCIP over DIP under different market conditions.

As to bidding languages, we have seen that the XOR-language is fully expressive (over finitely-peaked valuations) in section 5.3. Future work should address the expressive power of different fragments of the bidding language and compare the succinctness of different fragments for certain classes of valuations: which languages can express what valuations, and which languages can do so using less space than others? As to the case of direct single-unit combinatorial auctions, several results are given by Nisan (Nisan, 2006), and some of these results may be relatively easy to transfer to our model.

Theoretically, as to mechanism design, we do believe that we provided to game theorists a new interesting and difficult problem. An interesting question to consider in future work would be what exactly the auctioneer should *announce* when opening an MMUCA. In the case of direct auctions this is the set of goods to be sold. If bidding for transformations is possible, however, it may be difficult to foresee what types of goods will be relevant to a solution, as this depends on the transformation capabilities of the bidders in the market. Notice also that we have not provided any suggestion on how to run a MUCRAtR. This is not within the scope of this dissertation since it is a subject of mechanism design. However, in order to illustrate how our contribution can be used by a given mechanism, we offer an example about how a MUCRATR could be run:

(1) the auctioneer sends to bidders a WPTN representing his internal cost structure along with some constraints on the final state of the WPTN (its requirements)

(2) the bidders compose and send back to the auctioneer meaningful combinatorial offers based on the received information

(3) the auctioneer builds an *auction net* and solves a CMWOSP on it

(4) from the CMWOSP solution the auctioneer can extract the set of winning bids and the sequence of internal operations to subsequently perform

As to the mapping of the MMUCA WDP to CMWOSP, we have only exploited a small portion of its potentiality. We recall that we employed it to provide an ILP formulation for solving the WDP when the underlying topology is acyclic. However, we do believe that we can exploit further theoretical tools derived by our mapping along several dimensions. Some examples of this idea follows. First, it is known from the literature that it is possible to increase the classes of Petri nets for which the state equation represents the whole reachability set. As an example one may add linear side constraints to the state equation (Esparza and Melzer, 2004). Therefore, we would like to assess the applicability of these types of techniques to our problem. Secondly, the validity of the mapping from MMUCA WDP to WPTNS is not restricted to bids in the XOR language, but in fact it can easily cope with other languages. For instance, as explained in section 5.5, the extension to the OR-of-XOR bidding language is trivial. Third, and most importantly, our mapping allows to analyse structural and behavioural properties of the solutions to the MMUCA WDP. Thus, we aim at exploring the Petri net techniques that is possible to import in the context of MMUCAs.

Finally, we do strongly believe that CMWOSP can be employed to study other optimisation problems. In fact, the extension of CMWOSP to a broader class of optimisation problems that share similar features is a path that deserves much attention. In particular, we talk about domains characterised by preconditions and postconditions on variables interacting at multiple levels. The most promising of those domains is surely deterministic planning.

Appendix A

OPL models of the MMUCA WDP solvers

In this appendix, we present the ILP models of the solvers presented in this dissera-tion expressed in the OPL modeling language (Van Hentenryck, 1999). We present the CMWOSP-based (section 6.1.5), the DIP (section 6.2.2), and the CCIP (section 7.3.1) solvers.

A.1 The CMWOSP-based Solver

```
{string} Goods=...;
int nBids=...;
int nTransformations=...;
int nGoods=...;
int nBidders=...;

range Bids = 1..nBids;
range Transfs = 1..nTransformations;
range Bidders= 1..nBidders;

//DECLARATIONS

//Input goods of each transformation
int T_in[Transfs][Goods]=...;
//Output goods of each transformation
int T_out[Transfs][Goods]=...;
// Associates to each SCO the multiplicity it appears
//  within the bid
int multiplicity[Transfs]=...;
//A set contains the transitions indexes corresponding
//to the same atomic bid
```

```
{int} transf_same_bids[Bids]=...;

//Associates to each SCO its bid
int transf_to_bids[Transfs]=...;

//Which bids compose which XORbid
{int} xor_bids[Bidders]=...;

//Initial marking provided by the auctioneer for free
int U_in[Goods]=...;

//RFQ goods required by the auctioneer
int U_out[Goods]=...;

//The cost associated to each bid
float costs[Bids]=...;

//Variables associated to eac SCO is fired at each step
dvar boolean x_t[Transfs];

//Variables associated to atomic bids
dvar boolean x_b[Bids];


//THE MODEL

minimize

   sum(b in Bids) x_b[b]*costs[b];

subject to {

//(1) Each SCO can be fired as many times as its
// multiplicity if only if its bid is activated.
// This condidition also controls that selecting
// at least one SCO within a bid implies selecting
// all the SCOs within the same bid
 forall(t in Transfs)
  ctOnePositionSelected:
   (x_t[t]) ==(x_b[transf_to_bids[t]]*multiplicity[t]);

//(2) We enforce that the atomic bids submitted
// by each bidder are exclusive (XOR)
 forall(b in Bidders)
  ctXORbid:
   (sum ( j in xor_bids[b]) x_b[j]) <=1;

//(3) After having performed all the selected SCOs,
// the set of goods held by the auctioneer must be
// a superset of the final goods Uout
 forall(g in Goods)
  ctFinalConfiguration:
   (U_in[g] + sum(j in Transfs) x_t[j]* ...
            ...*(T_out[j][g]-T_in[j][g]))>=U_out[g];
}
```

A.2 The DIP solver

```
{string} Goods=...;
int nBids=...;
int nTransformations=...;
int nGoods=...;
int nBidders=...;
int nSteps=...;

range Bids = 1..nBids;
range Transfs = 1..nTransformations;
range Bidders= 1..nBidders;
range Steps= 1..nSteps;

//Input goods of each transformation
int T_in[Transfs][Goods]=...;

//Output goods of each transformation
int T_out[Transfs][Goods]=...;

// Associates to each transformation the multiplicity it
//appears within the bid
int multiplicity[Transfs]=...;

//A set contains the transitions indexes corresponding
// to the same atomic bid
{int} transf_same_bids[Bids]=...;

//Associates to each transformation its bid
int transf_to_bids[Transfs]=...;

//Which bids compose which XORbid
{int} xor_bids[Bidders]=...;
```

```
//Initial marking provided by the auctioneer for free
int U_in[Goods]=...;

//RFQ goods required by the auctioneer
int U_out[Goods]=...;

//The cost associated to each bid
float costs[Bids]=...;

//Variables associated to which transformation is fired
// at each step
dvar boolean x_t[Steps][Transfs];

//Variables associated to atomic bids
dvar boolean x_b[Bids];


minimize
   sum(b in Bids) x_b[b]*costs[b];
subject to {


//(1) Each transformation can be fired as many times
//as its multiplicity if only if its bid is activated.
//This condidition also controls that selecting at
//least one transformation within a bid implies selecting
// all the transformations within the same bid
forall(t in Transfs)
 ctOnePositionSelected:
  (sum(p in Steps) x_t[p][t])==...
           ...==(x_b[transf_to_bids[t]]*multiplicity[t]);

//(2) We impose that at most one transformation is
// selected at each position of the sequence
forall(p in Steps)
 ctOneTransformationSelected:
  sum(t in Transfs) x_t[p][t] <=1;

//(3) We enforce that the atomic bids submitted
// by each bidder are exclusive (XOR)
forall(b in Bidders)
 ctXORbid:
  (sum(j in xor_bids[b]) x_b[j]) <=1;
//(4) Check that each transition selected is enabled
// at steps in the solution sequence where more than
// one transition can be fired
forall(s in Steps,g in Goods)
 ctEnoughInputs:
  (U_in[g]+sum(k in 1..s-1,j in Transfs) x_t[k][j]*...
     ...*( T_out[j][g] - T_in[j][g]) )>= ...
     ...>= sum(l in Transfs) x_t[s][l]*T_in[l][g];

//(5) After having performed all the selected
//transformations, the set of goods held by the
// auctioneer must be a superset of the final goods
// Uout
forall(g in Goods)
 ctFinalConfiguration:
  (U_in[g]+sum(s in Steps,j in Transfs) x_t[s][j]*...
        ...*(T_out[j][g]-T_in[j][g]))>=U_out[g];

}
```

A.3 The CCIP Solver

```
{string} Goods=...; //Good names
int nBids=...; //Number of bids
int nTransformations=...; //Number of transformations
int nGoods=...; //Number of goods
int nBidders = ...; //Number of bidders or XORbids
//Number of solutions positions
// = number of transformations * multiplicities
int nSteps=...;

range Bids = 1..nBids;
range Transfs = 1..nTransformations;
range Bidders=1..nBidders;
range Steps=1..nSteps;

//Input goods of each transformation
int T_in[Transfs][Goods]=...;

//Output goods of each transformation
int T_out[Transfs][Goods]=...;

// Associates to each transformation the multiplicity
// it appears within the bid
int multiplicity[Transfs]=...;
```

```
//A set contains the transitions indexes corresponding
// to the same atomic bid
{int} transf_same_bids[Bids]=...;

//Associates to each transformation its bid
int transf_to_bids[Transfs]=...;

//A set contains the transitions indexes corresponding
// to bids of the same bidder
{int} xor_bids[Bidders]=...;

//Initial marking provided by the auctioneer for free
int U_in[Goods]=...;

//RFQ goods required by the auctioneer
int U_out[Goods]=...;

//The cost associated to each bid
float costs[Bids]=...;

//This array associates to each position in the solution
// the set of transformations that can fire
int  S[Steps][Transfs] = ...;

//This array associates 1 when the set of transformations
// that might be fired at this position is >1 or is one
// transformation that contains a self-loop
int steps_to_check[Steps]=...;

//Variables associated to which transformation is fired
// at each step
dvar boolean x_t[Steps][Transfs];

//Variables associated to atomic bids
dvar boolean x_b[Bids];


minimize
   sum(b in Bids) x_b[b]*costs[b];
subject to {

//(1) Each transformation can be should be fired as many
// times as its multiplicity if only if its bid is activated.
//This condidition also controls that selecting at least
// one transformation within a bid implies selecting all
// the transformations within the same bid
forall(t in Transfs)
 ctOnePositionSelected:
  (sum(p in Steps) x_t[p][t])==(x_b[transf_to_bids[t]]*...
                                ...*multiplicity[t]);

//(2) At most one transformation can fire at each position
   forall(p in Steps)
    ctOneTransformationSelected:
sum(t in Transfs) x_t[p][t] <=1;

//(3) XOR semantics of a bid is fulfilled, at most one bid
// per bidder can be selected
   forall(b in Bidders)
    ctXORbid:
         (sum(j in xor_bids[b]) x_b[j])<=1;

//(4) Check that each transition selected is enabled at
// steps in the solution sequence where more than one
//  transition can be fired
forall(s in Steps:steps_to_check[s]==1,g in Goods)
 ctEnoughInputs:
  (U_in[g]+sum(k in Steps:k<s,j in Transfs) x_t[k][j]*...
    ...*(T_out[j][g]-T_in[j][g]) )>= ...
    ...>= sum(l in Transfs) x_t[s][l]*T_in[l][g];

//(5) After having performed all the selected,
// transformations the set of goods held by the
// auctioneer must be a supersetof the final goods Uout
forall(g in Goods)
 ctFinalConfiguration:
  (U_in[g]+sum(s in Steps, j in Transfs) x_t[s][j]*...
      ...*(T_out[j][g]-T_in[j][g]))>=U_out[g];

//(6) Transformations that can fired in each position
//  of the solution sequence are restricted by function S
forall(p in Steps, j in Transfs)
 ctTransformationsPosition:
  x_t[p][j]<=S[p][j];


}
```

Bibliography

Aissaoui, N., Haouari, M., and Hassini, E. (2007). Supplier selection and order lot sizing modeling: A review. *Computers and Operations Research*, 34(12):3516–3540.

Alonso-Ayuso, A., Escudero, L., Garín, A., Ortuño, M., and Pérez, G. (2003). An Approach for Strategic Supply Chain Planning under Uncertainty based on Stochastic 0-1 Programming. *Journal of Global Optimization*, 26(1):97–124.

An, N., Elmaghraby, W., and Keskinocak, P. (2005). Bidding strategies and their impact on revenues in combinatorial auctions. *Journal of Revenue and Pricing Management*, 3(4):337–357(21).

Andersson, A., Tenhunen, M., and Ygge, F. (2000). Integer programming for combinatorial auction winner determination. In *Fourth International Conference on Multiagent Systems (ICMAS 2000)*, pages 39–46, Boston, MA.

Ausubel, L. M., Cramton, P., and Milgrom, P. (2006). *The Clock-Proxy Auction: A Practical Combinatorial Auction Design*, chapter 5. Combinatorial Auctions. MIT Press.

Ausubel, L. M. and Milgrom, P. (2006a). *Ascending Proxy Auctions*, chapter 3. Combinatorial Auctions. MIT Press.

Ausubel, L. M. and Milgrom, P. (2006b). *The Lovely but Lonely Vickrey Auction*, chapter 1. Combinatorial Auctions. MIT Press.

Babanov, A., Collins, J., and Gini, M. (2003). Asking the right question: Risk and expectation in multiagent contracting. *Artificial Intelligence For Engineering Design Analysis And Manufacturing*, 17:173–186.

Ball, M. O., Donohue, G. L., and Hoffman, K. (2006). *Auctions for the Safe, Efficient, and Equitable Allocation of Airspace System Resources*, chapter 20. Combinatorial Auctions. MIT Press.

Banks, J., Ledyard, J., and Porter, D. (1989). Allocating Uncertain and Unresponsive Resources: An Experimental Approach. *The RAND Journal of Economics*, 20(1):1–25.

Bartels, A., Ragsdale, J., Pohlmann, T., Young, G. O., and Brown, K. (2005). The forrester wave: e-sourcing suites, Q4 2005. Technical report, Forrester. Tech Notes.

Bernardinello, L. and de Cindio, F. (1992). A survey of basic net models and modular net classes. In *Advances in Petri Nets 1992, The DEMON Project*, pages 304–351, London, UK. Springer-Verlag.

Bichler, M., Davenport, A., Hohner, G., and Kalagnanam, J. (2006). *Industrial Procurement Auctions*, chapter 23. Combinatorial Auctions. MIT Press.

Blizard, W. and File, P. (1988). Multiset theory. *Notre Dame J. Formal Logic*, 30(1):36–66.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308.

Boutilier, C., Goldszmidt, M., and Sabata, B. (1999). Sequential Auctions for the Allocation of Resources with Complementarities. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence table of contents*, pages 527–523.

Boutilier, C. and Hoos, H. (2001). Bidding languages for combinatorial auctions. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 1211–1217.

Burkeman, O. (5 september 2005). The wiki way. Guardian Unlimited `www.guardian.co.uk/technology/2007/sep/05/news.netrich`.

Cantillon, E. and Pesendorfer, M. (2006). *Auctioning Bus Routes: The London Experience*, chapter 22. Combinatorial Auctions. MIT Press.

Caplice, C. and Sheffi, Y. (2006). *Combinatorial Auctions for Truckload Transportation*, chapter 21. Combinatorial Auctions. MIT Press.

Christie's (2007). Online. `http://www.christies.com/`.

Coase, R. (1937). The Nature of the Firm. *Economica*, 4(16):386–405.

Cohen, M. and Lee, H. (1988). Strategic Analysis of Integrated Production-Distribution Systems: Models and Methods. *Operations Research*, 36(2):216–228.

Collins, J. (2002). *Solving Combinatorial Auctions with Temporal Constraints in Economic Agents*. PhD thesis, University Of Minnesota.

Corbett, M. (2004). *The Outsourcing Revolution: Why it Makes Sense and how to Do it Right*. Dearborn Trade Pub.

Cormen, T. (2001). *Introduction to Algorithms*. MIT Press.

Cramton, P. (2006). *Simultaneous Ascending Auctions*, chapter 4. Combinatorial Auctions. MIT Press.

Cramton, P., Shoham, Y., and Steinberg, R., editors (2006). *Combinatorial Auctions*. MIT Press.

Davey, B. and Priestley, H. (2002). *Introduction to Lattices and Order*. Cambridge University Press.

de Vries, S. and Vohra, R. (2003). Combinatorial auctions: A survey. *INFORMS Journal of Computing*, 15(3):284–309.

DMReview.com online news (March 22, 2006). Business outsourcing makes strong push forward with small and midsize deals. Online News.

DMReview.com online news (October 31, 2005). Worldwide bpo market steadily expands. Online News.

Dyer, J. (2000). *Collaborative Advantage: Winning Through Extended Enterprise Supplier Networks*. Oxford University Press US.

Ebay (2007). Online. `http://www.ebay.com`.

Engel, Y., Wellman, M. P., and Lochner, K. M. (2006). Bid expressiveness and clearing algorithms in multiattribute double auctions. In *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*, pages 110–119, New York, NY, USA. ACM Press.

Erenguc, S., Simpson, N., and Vakharia, A. (1999). Integrated production/distribution planning in supply chains: An invited review. *European Journal of Operational Research*, 115(2):219–236.

Ertogral, K., Wu, S., and Burke, L. (1998). Coordination Production and Transportation Scheduling in the Supply Chain. *Dept. IMSE, Lehigh Univ., Bethlehem, PA, Tech. Rep*.

Esparza, J. and Melzer, S. (2004). Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, 16:159–189.

Fourer, R., Gay, D., and Kernighan, B. (1989). *AMPL: A Mathematical Programming Language*. AT&T Bell Laboratories.

Fujishima, Y., Leyton-Brown, K., and Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceeding of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 548–553.

Gaonkar, R. and Viswanadham, N. (2001). Collaboration and information sharing in global contractmanufacturing networks. *Mechatronics, IEEE/ASME Transactions on*, 6(4):366–376.

Gaonkar, R. and Viswanadham, N. (2005). Strategic sourcing and collaborative planning in Internet-enabled supply chain networks producing multigeneration products. *Automation Science and Engineering, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, 2(1):54–66.

Greaver, M. (1999). *Strategic Outsourcing: A Structured Approach to Outsourcing Decisions and Initiative*. AMACOM.

Hammer, M. (2001). The Superefficient Company. *Harvard Business Review*, 79(8):82–91.

Harary, F. (1999). *Graph Theory*. Perseus Books.

He, M., Jennings, N., and Leung, H. (2003). On agent-mediated electronic commerce. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):985–1003.

Hillier, F. and Lieberman, G. (1986). *Introduction to Operation Research*. McGraw-Hill.

Holte, R. C. (2001). Combinatorial auctions, knapsack problems, and hill-climbing search. In *Lecture Notes in Computer Science*, volume 2056, pages 57 – 66. Springer-Verlag, Heidelberg.

ILOG (2007). Cplex. `http://www.ilog.com/products/cplex/`.

Jayaraman, V. and Pirkul, H. (2001). Planning and coordination of production and distribution facilities for multiple commodities. *European Journal of Operational Research*, 133(2):394–408.

Jennings, N. and Wooldridge, M. (1998). Applications of intelligent agents. *Agent technology: foundations, applications, and markets table of contents*, pages 3–28.

Kalagnanam, J. and Parkes, D. (2003). Auctions, Bidding and Exchange Design. Chapter 10. *Supply Chain Analysis in the eBusiness Era*. Edited by David Simchi-Levi, S. David Wu and Z. Max Shen.

Kallrath, J. (2002). Planning and scheduling in the process industry. *OR Spectrum*, 24(3):219–250.

Kallrath, J. (2004). *Modeling Languages in Mathematical Optimization*. Kluwer Academic Pub.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack problems*. Springer New York.

Klemperer, P. (2004). *Auctions: Theory and Practice*. Princeton University Press.

Krishna, V. (2002). *Auction Theory*. Academic Press.

Land, A., Powell, S., and Steinberg, R. (2006). *PAUSE: A Computationally Tractable Combinatorial Auction*, chapter 6. Combinatorial Auctions. MIT Press.

Lau, J., Huang, G., Mak, K., and Liang, L. (2006). Agent-based modeling of supply chains for distributed scheduling. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 36(5):847 – 861.

Laubacher, R. and Malone, T. (2003). Retreat of the Firm and the Rise of the Guilds: The Employment Relationship in an Age of Virtual Business. *Inventing the Organization of the 21st Century. Malone TW, Laubacher R, Scott-Morton MS (eds). MIT Press.*

Lee, Y., Jeong, C., and Moon, C. (2002). Advanced planning and scheduling with outsourcing in manufacturing supply chain. *Computers & Industrial Engineering*, 43(1-2):351–374.

Lee, Y., Kumara, S., and Chatterjee, K. (2003). Multiagent based dynamic resource scheduling for distributed multiple projects using a market mechanism. *Journal of Intelligent Manufacturing*, 14(5):471–484.

Lehmann, D., Mueller, R., and Sandholm, T. M. (2006). *The winner determination problem,* chapter 12. Combinatorial Auctions. MIT Press.

Leyton-Brown, K. and Shoham, Y. (2006). *Combinatorial Auctions,* chapter 18. A Test Suite for Combinatorial Auctions. MIT Press.

Leyton-Brown, K., Shoham, Y., and Tennenholtz, M. (2000). An algorithm for multi-unit combinatorial auctions. In *Proceedings of the American Association for Artificial Intelligence Conference (AAAI)*, pages 56–61.

Lih, A. (2003). Wikipedia as Participatory Journalism: Reliable Sources? Metrics for evaluating collaborative media as a news resource. *Nature*, 2004.

Lindo Systems Inc. (2007). Lingo. `http://www.lindo.com/products/lingo`.

Lipton, R. (1976). The reachability problem requires exponential space. Technical Report 62, Yale University.

Lucking-Reiley, D. and Spulber, D. (2001). Business-to-Business Electronic Commerce. *The Journal of Economic Perspectives*, 15(1):55–68.

Makhorin, A. GNU Linear Programming Kit: Modelling Language GNU MathProg.

Makhorin, A. (2001). Glpk – gnu linear programming toolkit. `http://www.gnu.org/directory/GNU/glpk.html`. [Online; accessed 25-Oct-2007].

Mas-Colell, A., Whinston, M., and Green, J. (1995). *Microeconomic Theory*. Oxford University Press.

McAfee, R. and McMillan, J. (1987). Auctions and Bidding. *Journal of Economic Literature*, 25(2):699–738.

Microsoft (2007). Excel. `http://office.microsoft.com/en-us/excel/`.

Milgrom, P. (2004). *Putting Auction Theory to Work*. Cambridge University Press.

Minahan, T., Howarth, F., and Vigoroso, M. (2002). Making e–sourcing strategic. *Research report, Aberdeen Group, Boston*.

Mowshowitz, A. (2002). *Virtual Organization: Toward a Theory of Societal Transformation Stimulated by Information Technology*. Quorum/Greenwood.

Muller, R. (2006). *Tractable Cases of the Winner Determination Problem*, chapter 13. Combinatorial Auctions. MIT Press.

Murata, T. (1989). Petri nets: Properties, analysis and applications. In *IEEE*, volume 77, pages 541–580.

Nisan, N. (2000). Bidding and allocation in combinatorial auctions. In *Proc. 2nd ACM Conference on Electronic Commerce (EC-2000)*. ACM Press.

Nisan, N. (2006). *Bidding Languages for Combinatorial Auctions*, chapter 9. Combinatorial Auctions. MIT Press.

Norman, T., Preece, A., Chalmers, S., Jennings, N., Luck, M., Dang, V., Nguyen, T., Deora, V., Shao, J., Gray, W., et al. (2004). Agent-based formation of virtual organisations. *Knowledge-Based Systems*, 17(2-4):103–111.

Ottens, B. (2007). Comparing winner determination algorithms for mixed multi-unit combinatorial auctions. Master's thesis, Universiteit van Amsterdam., Netherlands.

Papadimitriou, C. and Steiglitz, K. (1982). *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

Parkes, D. C. (2006). *Iterative Combinatorial Auctions*, chapter 2. Combinatorial Auctions. MIT Press.

Pekec, A. and Rothkopf, M. H. (2003). Combinatorial auction design. *Manage. Sci.*, 49(11):1485–1503.

Petri, C. (1962). *Kommunikation mit Automaten*. PhD thesis, Rheinisch-Westfälisches Institut f. instrumentelle Mathematik an d. Univ.

Petri, C. (1966). Kommunikation mit automaten. Schriften des iim nr. 2, Institut fur Instrumentelle Mathematic, 1962. Technical report, English translation: Technical Report RADC-TR-65-377, Griffiths Air Base, New York.

Quinn, J. and Hillmer, F. (1995). Strategic Outsourcing. *The McKinsey Quarterly*, (1).

Reis, J., Mamede, N., and O Neill, H. (2001). Locally perceiving hard global constraints in multi-agent scheduling. *Journal of Intelligent Manufacturing*, 12(2):223–236.

Reisig, W. (1985). *Petri nets: an introduction*. Springer-Verlag, New York, NY, USA.

Reyes-Moro, A., Rodríguez-Aguilar, J. A., López-Sánchez, M., Cerquides, J., and Gutiérrez-Magallanes, D. (2003). Embedding decision support in e-sourcing tools: Quotes, a case study. *Group Decision and Negotiation*, 12:347–355.

Rosenschein, J. and Zlotkin, G. (1994). *Rules of encounter*. MIT Press Cambridge, Mass.

Rothkopf, M. H., Pekec, A., and Harstad, R. M. (1998). Computationally manageable combinational auctions. *Management Science*, 44(8):1131–1147.

Sabri, E. and Beamon, B. (2000). A multi-objective approach to simultaneous strategic and operational planning in supply chain design. *Omega*, 28(5):581–598.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54.

Sandholm, T. (2006a). Expressive commerce and its application to sourcing. *Proceedings of the Innovative Applications of Artificial Intelligence*.

Sandholm, T. (2006b). *Optimal Winner Determination Algorithms*, chapter 14. Combinatorial Auctions. MIT Press.

Sandholm, T., Suri, S., Gilpin, A., and Levine, D. (2002). Winner determination in combinatorial auction generalizations. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 69–76, Bologna, Italy. ACM Press.

Schrage, M. (2007). The Myth of Commoditization. *Sloan management review*, 48(2):10–14.

Shaeffer, L. D. (September 11, 2007). Five More Keys to Engaging the Customer to Produce Real Innovation: Lessons From LEGO.

Simchi-Levi, D., Kaminsky, P., and Simchi-Levi, E. (2000). *Designing and Managing the Supply Chain: Concepts, Strategies, and Case Studies*. Irwin/McGraw-Hill.

Smith, R. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 29(12):1104–1113.

Sotheby's (2007). Online. `http://www.sothebys.com/`.

SourceForge, S.F. (2007). SourceForge.net Home Page. `http://www.sourceforge.net`.

Syropoulos, A. Mathematics of Multisets. *Pre-proceedings of the Workshop on Multiset Processing*, pages 286–295.

Tapscott, D. and Williams, A. (2006). *Wikinomics: how mass collaboration changes everything*. Portfolio.

Tarek, A. and Lopez-Benitez, N. (2004). Optimal legal firing sequence of petri nets using linear programming. *Optimization and Engineering*, 5(1):25–43.

Team, I. S. C. (2001). Strategic sourcing: Applications to turn direct materials procurement into competitive advantage. Technical report, Stephens Inc. Industry Report.

Van Hentenryck, P. (1999). *The OPL optimization programming language*. MIT Press Cambridge, MA, USA.

Vinyals, M. (2007a). Bid generator figures. Internal Communication.

Vinyals, M. (2007b). On the empirical evaluation of mixed multi-unit combinatorial auctions. Master's thesis, Universitat Autonoma de Barcelona.

Vinyals, M., Cerquides, J., and Rodriguez-Aguilar, J. A. (2007a). On the empirical evaluation of mixed multi-unit combinatorial auctions. *Workshop on Agent Mediated Electronic Commerce IX,AMEC*.

Vinyals, M., Giovannucci, A., Cerquides, J., Meseguer, P., and Rodriguez-Aguilar, J. A. (2007b). Towards a realistic bid generator for mixed multi-unit combinatorial auctions. *14th Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion RCRA*.

Viswanadham, N. (2002). The past, present, and future of supply-chain automation. *IEEE Robotics & Automation Magazine*, 9(2):48–56.

Wagner, T., Guralnik, V., and Phelps, J. (2003). TAEMS agents: enabling dynamic distributed supply chain management. *Electronic Commerce Research and Applications*, 2(2):114–132.

Walsh, W. and Wellman, M. (2003). Decentralized supply chain formation: A market protocol and competitive equilibrium analysis. *Journal of Artificial Intelligence Research*, 19:513–567.

Walsh, W. E. (2001). *Market protocols for decentralized supply chain formation*. PhD thesis. Chair-Michael P. Wellman.

Walsh, W. E., Wellman, M. P., and Ygge, F. (2000). Combinatorial auctions for supply chain formation. In *Proc. of the 2nd ACM Conference on Electronic Commerce*.

Wooldridge, M. and Jennings, N. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152.

Wurman, P., Walsh, W., and Wellman, M. (1998). Flexible double auctions for electionic commerce: Theory and implementation. *Decision Support Systems*, 24:17–27.

Zhang, X. (2002). *Sophisticated Negotiation in Multi-agent Systems*. PhD thesis, University of Massachusetts at Amherst.

Zlotkin, G. and Rosenschein, J. (1996). Mechanism design for automated negotiation, and its application to task oriented domains. *Artificial Intelligence*, 86(2):195–244.

Num. 21	J. Cerquides, *Improving Algorithms for Learning Bayesian Network Classifiers*, (2005).
Num. 22	M. Villaret, *On Some Variants of Second-Order Unification*, (2005).
Num. 23	M. Gómez, *Open, Reusable and Configurable Multi-Agent Systems: A Knowledge Modelling Approach*, (2005).
Num. 24	S. Ramchurn, *Multi-Agent Negotiation Using Trust and Persuasion*, (2005).
Num. 25	S. Ontañon, *Ensemble Case-Based Learning for Multi-Agent Systems*, (2006).
Num. 26	M. Sánchez, *Contributions to Search and Inference Algorithms for CSP and Weighted CSP*, (2006).
Num. 27	C. Noguera, *Algebraic Study of Axiomatic Extensions of Triangular Norm Based Fuzzy Logics*, (2007).
Num. 28	E. Marchioni, *Functional Definability Issues in Logics Based on Triangular Norms*, (2007).
Num. 29	M. Grachten, *Expressivity-Aware Tempo Transformations of Music Performances Using Case Based Reasoning*, (2007).
Num. 30	I. Brito, *Distributed Constraint Satisfaction*, (2007).
Num. 31	E. Altamirano, *On Non-clausal Horn-like Satisfiability Problems*, (2007).
Num. 32	A. Giovannucci, *Computationally Manageable Combinatorial Auctions for Supply Chain Automation*, (2008).