Investigation of Matching Problems using Constraint Programming and Optimisation Methods

Danuta Sorina Chisca

JOINT PhD BETWEEN
UNIVERSITY COLLEGE CORK
AND
ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY COLLEGE CORK

Thesis submitted for the degree of Doctor of Philosophy

November 2019

Head of School: Professor Cormac Sreenan

Supervisors: Professor Barry O'Sullivan (UCC)
Professor Michela Milano (UNIBO)

Research supported by Insight Centre for Data Analytics and Science Foundation Ireland under Grant No. 12/RC/2289 which is co-funded under the European Regional Development Fund.

Contents

List of Figures

List of Tables

I, Danuta Sorina Chisca, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere. This is a Joint PhD between University College Cork, School of Computer Science and Information Technology, Cork, Ireland and Alma Mater Studiorum - Università di Bologna, Computer Science and Engineering (DISI) department, Bologna, Italy. Parts of this work have appeared in the following publications which have been subject to peer review:

- Danuta Sorina Chisca, Mohamed Siala, Gilles Simonin, and Barry O'Sullivan. *A CP-based approach for popular matching"*. Published in the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona, USA, pages 4202–4203, 2016. ISBN 978-1-57735-760-5 [Chisca et al., 2016]
- Danuta Sorina Chisca, Mohamed Siala, Gilles Simonin and Barry O'Sullivan. *New Models for Two Variants of Popular Matching*. Published in: 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), Boston, USA, pages 752–759, 2017. DOI: 10.1109/ICTAI.2017.00119 [Chisca et al., 2017]
- Danuta Sorina Chisca, Michele Lombardi, Michela Milano and Barry O'Sullivan. *From Offline to Online Kidney Exchange Optimization*. Published in: 2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI), Volos, Greece, pages 587–591. DOI: 10.1109/ICTAI.2018.00095 [Chisca et al., 2018]
- Danuta Sorina Chisca, Michele Lombardi, Michela Milano and Barry O'Sullivan. *A Sampling-free Anticipatory Algorithm for the Kidney Exchange Problem*. In Proceedings of the 16th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2019, Thessaloniki, Greece, pages 146–162, 2019. [Chisca et al., 2019b]
- Danuta Sorina Chisca, Michele Lombardi, Michela Milano and Barry O'Sullivan. *Logic-Based Benders Decomposition for Super Solutions: an Application to the Kidney Exchange Problem*. In Proceedings of 25th International Conference on Principles and Practice of Constraint Programming, CP 2019, Stamford, USA, pages 108–125, 2019. [Chisca et al., 2019a]

This research was undertaken at the Insight Centre for Data Analytics and supported by Science Foundation Ireland under Grant Number SFI/12/RC/2289, which is co-funded under the European Regional Development Fund.

Danuta Sorina Chisca

Acronyms

AEG - Abstract Exchange Graph

CHA - Capacitated House Allocation problem

CP - Constraint Programming

CSBA - Collective Scenario-Based Algorithm

CSP - Constraint Satisfaction Problem

gcc - global cardinality constraint

HA - House Allocation

HAT - House Allocation with Ties

HR - Hospitals/Residents Problem

KEP - Kidney Exchange Problem

LBBD - Logic-Based Benders Decomposition

MP - Master Problem

SP - Sub Problem

SM - Stable Marriage Problem

SR - Stable Roommates problem

SRI - Stable Roommate with Incomplete lists

To M.

Acknowledgements

"It's not that I'm so smart, it's just that I stay with problems longer." Einstein

There are many people that have helped me with doing this thesis and to whom I owe a debt of gratitude. Firstly, I would like to thank immensely my both supervisors, Barry O'Sullivan and Michela Milano, for their constant support and guidance; without their vision and expertise this thesis would certainly not be what it is today.

I'm very grateful to be able to work with the "popular" postdocs Gilles Simonin and Mohamed Siala, for their help in my search of "popularity". Collaborating with them allowed me to acknowledge theoretical algorithms, which I was unaware at the moment. Huge thanks to Michele Lombardi for his dedication to seek/try different algorithms and his continuous sustain over the years. I gained much from Michele knowledge, insight and enthusiasm.

I've been fortunate to share great times, delicious food, and in-depth conversations with many friends. In Bologna I had the pleasure to share the desk and to maintain a close friendships with Allegra, you have been a constant source of everything from deep life advice to drinking, spritz and orange juice. I'm also grateful to my "Insight" supporters: to Chrys - you always pushed me to do better, spinning and drinking coffee; to Samreen - so grateful for all the dinners we had when visiting you, P.S.: you make the most amazing biryani; to Caitríona - your patience was enormous for "rompipalle" me when asking you to remind me if I have all the documents and always forgetting one or two. It was a pleasure to share the apartment with Sarah and Aida, and so many thanks to my ex-housemates for "wining" together at the end of the day. Even if we don't live together Sarah you are still the best runner at 5 am, and Aida you definitely know how to shake it off at the end of the day.

Lastly, and most importantly, I would like to thank my family and second family for their love, understanding, support and encouragement through all aspects of my project.

Abstract

This thesis focuses on matching under ordinal preferences, i.e. problems where agents may be required to list other agents that they find acceptable in order of preference. In particular, we focus on two main cases: the popular matching and the kidney exchange problem. These problems are important in practice and in this thesis we develop novel algorithms and techniques to solve them as combinatorial optimisation problems. The first part of the thesis focuses on one-sided matching on a bipartite graph, specifically the popular matching. When the participants express their preferences in a cardinal order, the most common desire is to maximise a utility function, for example finding the maximum weight or minimum cost matching. If preferences are ordinal in nature, one might want to guarantee that no two applicants are inclined to form a coalition in order to maximise their welfare, thus finding a stable matching is needed. Therefore, negotiating the size and the optimality of the matching with respect to agents' preferences is a problem that occurs naturally. Popularity is a concept that offers an attractive trade-off between these two notions. In short, a popular matching is a matching in which the majority of the participants will decide on a matching M. In particular, we examine the popular matching in the context of constraint programming using global constraints. We discuss the possibility to find a popular matching even for the instances that does not admit one.

The second part of the thesis focuses on non-bipartite graphs, i.e. the kidney exchange problem, a recent innovation that matches patients in need of a kidney to willing living donors. Kidney transplant is the most effective treatment to cure end-stage renal disease, affecting one in every thousand European citizens. Chronic kidney disease is a life threatening health issue that affects millions of people worldwide. Motivated by the observation that the kidney exchange is inherently a stochastic online problem, first, we give a stochastic online method. This methos is more general and provides an expected value estimation that is correct within the limit of sampling errors. Second, we show that by taking into consideration a probabilistic model of future arrivals and drop-offs, we can get reduce sampling scenarios, and we can even construct a sampling-free probabilistic model, called the Abstract Exchange Graph (AEG). A final contribution of this thesis is related to finding robust solutions when uncertainty occurs. Uncertainty is inherent to most real world problems. For instance in the case of Kidney Exchange Problem (KEP), if a pair pools out from the solution we need to find a repair solution quickly to limit the impact to other pairs. We propose a method that exploits a Logic-Based Benders Decomposition to find super solutions to an optimisation problem.

Sommario

L'obiettivo di questa tesi è quello di studiare problemi di matching (accoppiamento) con preferenze ordinali, i.e. problemi dove è necessario che gli agenti esprimono le loro preferenze in una lista ordinata.

La prima parte della tesi si concentra su problemi di matching unilaterale in un grafo bipartito: il popular matching. Se i partecipanti esprimono le loro preferenze in ordine cardinale, il modo più comune è di massimizzare una funzione di utilità, per esempio trovare il peso massimo oppure il costo minimo del matching. Se le preferenze sono fornite in modo ordinale, si dovrebbe garantire che nessuno dei due partecipanti creasse una coalizione in modo da massimizzare il proprio welfare, quindi trovare un matching stabile è necessario. Quindi negoziare il matching tra la dimensione e ottimalità con rispetto alle preferenze degli agenti è un problema che si verifica in modo naturale. Il concetto di popolarità (popularity) offre un compromesso di queste due nozioni. Brevemente, un matching popolare è un matching in cui la maggioranza degli agenti scelgono il matching M. In particolare, esaminiamo il matching popolare nel contesto di constraint programming (programmazione a vincoli) facendo uso dei vincoli globali. Discutiamo la possibilitá di trovare un matching popolare anche per le istanze che non ne ammettono una.

La seconda parte della tesi si concentra su un grafo non-bipartito, i.e. lo scambio dei reni - kidney exchange problem (KEP), un'innovazione recente che accoppia i pazienti bisognosi di un rene a donatori viventi disponibili. Il trapianto dei reni è il trattamento più efficace per chi soffre di un'insufficienza renale all'ultimo stadio, che colpisce una persone su mille nei paesi europei. Molto spesso un paziente bisognoso di un trapianto ha un amico o familiare che sono disposti a donare il loro rene. Tuttavia, a causa delle differenze del tipo di sangue e la presenza di certe proteine nel sangue, il donatore potenziale potrebbe risultare incompatibile con il paziente. Quindi un trapianto non può essere effettuato, oppure se si decide di procedere comunque il paziente sarebbe ad alto rischio di respingere il rene del donatore. La malattia renale cronica è un gravissima problema per la salute che colpisce milioni di persone al mondo. Motivati dall'osservazione che lo scambio dei reni è un problema online intrinsicamente stocastica, Per primo, diamo un metodo stocastico online che è più generale e forniamo una stima del valore atteso che è corretta entro i limiti degli errori di campionamento. Secondo, facciamo vedere che prendendo in considerazione un modello probabilistico di nuovi arrivi e uscite, ci liberiamo degli scenari e possiamo costruire un modello probabilistico senza campionamento, chiamato Abstract Kidney Exchange (AEG). Un

contributo finale di questa tesi e' relativo a trovare soluzioni robuste considerando delle incertezze. Proponiamo un metodo sfruttando la Logic Based Benders Decomposition per trovare le super soluzioni di un problema di ottimizzazione. Il master si occupa del problema originale, mentre i sottoproblemi cercano di trovare soluzioni da riparare ogni perturbanza.

Chapter 1

Introduction

"In God we trust, all others must bring data." W. Edwards Deming

1.1 Motivation

Matching problems involving preferences are motivated in practice in widespread applications, such as the assignment of children to schools, junior doctors to hospitals, kidney transplant patients to donors and so on [Manlove et al., 2007, Kavitha and Nasre, 2009, Irving, 2007a, Gardenfors, 1975, Biró, 2008]. Given the applications of matching problems, and the implications of a participant's allocation in a matching for their quality of life, it is of paramount importance that the matching algorithms that drive such applications should optimise in some sense the satisfaction of the participants according to their preferences [Manlove, 2013, Gale and Shapley, 1962]. Since execution time is a very important requirement in practice, a lot of the research effort has gone into developing efficient algorithms for solving these matching problems [Hebrard et al., 2004b, Holland and O'Sullivan, 2005a, Hebrard and Walsh, 2005]. These factors have led to a lot of research activity in the area.

This thesis investigates two important variants of matching theory, and shows how Constraint Programming and Operational Research can be applied to combinatorial optimisation problems. These problems are growing in importance and we address the design, analysis and real-world applications of matching problems of indivisible goods,

such as the popular matching problem and the kidney exchange problem. We focus on the creation of new mathematical models for these problems that more accurately reflects the reality, and the development of optimal models that can be deployed in practice.

1.2 Matching Under Ordinal Preferences

Matching problems involve the assignment of a set (or sets) of agents to one another, subject to some criteria. In many cases the agents form two disjoint sets, and we seek to assign the agents in one set to those in the other.

We focus on the case where agents have ordinal preferences over a subset of the others, meaning that there is a notion of first choice, second choice, and so on. For example a student who is applying for admission to university might rank in order of preference a small list of available universities, i.e. four universities. Similarly, the universities might form a ranking of their applicants according to some academic criteria.

Matching problems with preferences can be classified into three major groups:

A. bipartite matching problems with two-sided preferences, where the agents can be partitioned into two disjoint sets, and each member of one set ranks a subset of the members of the other set in order of preference. Example applications include assigning junior doctors to hospitals [NRMP, 2019, car, 2019], pupils to schools [Abdulkadiroğlu et al., 2005] and school- leavers to universities [Biró, 2008].

B. bipartite matching problems with one-sided preferences consist of partitioning the agents into two disjoint sets, but each member of only one set ranks a subset of the members of the other set in order of preference. Example applications include campus housing allocation problem [Chen and Snmez, 2002] , DVD rental markets [Abraham et al., 2006] and assigning reviewers to conference papers [Garg et al., 2010].

C. non-bipartite matching problems with preferences, where the agents form a single homogeneous set, and each agent ranks a subset of the others in order of preference. Example applications include stable roommates problem, forming pairs of agents for chess tournaments [Kujansuu et al., 1999], finding kidney exchanges involving incompatible patient–donor pairs [Roth et al., 2004, Wallis et al., 2011, Dickerson et al., 2012b].

For bipartite matching problems with preferences, an extensively studied problem is the classical Stable Marriage Problem (SM)[Gale and Shapley, 1962], in which the participants consist of two disjoint sets of agents, say n men and n women, each of whom ranks all members of the opposite sex in order of preference and a matching is just a one-one mapping between the two sets. Note that we henceforth use the term agents to refer to those participants in matching problems who have preference lists. Hence, the agents in a SM instance are the men and women.

In [Gale and Shapley, 1962] the authors showed that, given an SM instance of size n, a stable matching can always be found in polynomial-time, and moreover they also provided an $O(n^2)$ algorithm to find such a matching. The algorithm involves a sequence of marriage proposals made by the members of one sex to the members of the other. Each proposal is followed by either an acceptance or a rejection. Proposals are accepted if the person being proposed to is either unmatched or prefers the proposer to his/her current partner, in which case his/her current partner is rejected in favour of the new proposal. Proposals are rejected if the person being proposed to already has a partner he/she prefers to the proposer. The algorithm can be executed with the men doing the proposing (man-oriented) or with the women doing the proposing (woman-oriented). Agents of the proposing set start with the most preferred agents on their preference lists and make proposals until they are accepted. If they ever get rejected due to their partner getting a better proposal from someone else, they continue by making a proposal to the next agent on their preference list. The algorithm terminates when all parties involved are matched.

Men's preferences:	Women's preferences:
$m_1 : w_1, w_2, w_3, w_4$	$w_1 : m_4, m_3, m_2, m_1$
$m_2 : w_2, w_1, w_4, w_3$	$w_2 : m_3, m_4, m_1, m_2$
$m_3 : w_3, w_4, w_1, w_2$	$w_3 : m_2, m_1, m_4, m_3$
$m_4 : w_4, w_3, w_2, w_1$	$w_4 : m_1, m_2, m_3, m_4$

Figure 1.1: An instance of SM

For example in Figure 1.1, we have a set of 4 men and a set of 4 women, seeking for a match, and there are 10 stable matchings [Knuth, 1976]:

$$M_1 = \{(m_1, w_1), (m_2, w_2), (m_3, w_3), (m_4, w_4)\},$$
$$M_2 = \{(m_1, w_2), (m_2, w_1), (m_3, w_3), (m_4, w_4)\},$$
$$M_3 = \{(m_1, w_1), (m_2, w_2), (m_3, w_4), (m_4, w_3)\},$$
$$M_4 = \{(m_1, w_2), (m_2, w_1), (m_3, w_4), (m_4, w_3)\},$$
$$M_5 = \{(m_1, w_2), (m_2, w_4), (m_3, w_1), (m_4, w_3)\},$$
$$M_6 = \{(m_1, w_3), (m_2, w_1), (m_3, w_4), (m_4, w_2)\},$$
$$M_7 = \{(m_1, w_3), (m_2, w_4), (m_3, w_1), (m_4, w_2)\},$$
$$M_8 = \{(m_1, w_3), (m_2, w_4), (m_3, w_2), (m_4, w_1)\},$$
$$M_9 = \{(m_1, w_4), (m_2, w_3), (m_3, w_1), (m_4, w_2)\},$$
$$M_{10} = \{(m_1, w_4), (m_2, w_3), (m_3, w_2), (m_4, w_1)\}.$$

Alternatively to SM, preference lists for bipartite matching problems can be one-sided. An example of this type of problem is the House Allocation problem (HA) [Abraham et al., 2004], where an attempt is made to allocate a set H of objects (e.g., houses, posts etc) using a one-one mapping among a set A of agents, each of whom ranks a subset of H in order of preference.

In addition to bipartite matching problems, non-bipartite matching problems are also widely studied. In the classical Stable Roommates problem (SR) [Gale and Shapley, 1962, Irving, 1985], the participants consist of a single set of agents each of whom ranks the others in order of preference, and a matching is a partition of the set into disjoint pairs of roommates.

1.3 The Popular Matching Problem

In the one sided bipartite graph, each applicant has an ordinal preference list, ranking the subset of items in order of his preference (first choice, second choice, and so on). We seek to match applicants to items in which no applicant is matched with more than one item, and no item is matched with more than one applicant. We will consider that the preference lists are not strictly ordered, for example an applicant might have two items as first choice in his preference list (a tie).

Consider the example below with three applicants and three items, and the respective preference list:

$$a_1 : p_1 \quad p_2 \quad p_3$$
$$a_2 : p_1 \quad p_2 \quad p_3$$
$$a_3 : p_1 \quad p_2 \quad p_3$$

Figure 1.2: An instance of popular matching

Let M and M′ be two arbitrary matchings in a given instance I and let P(M,M′) denote the set of agents who prefer M to M′. We say that M is more popular than M′ if |P(M,M′)| > |P(M′,M)|, i.e. the number of agents who prefer M to M′ is greater than the number of agents who prefer M′ to M. A matching M in I is popular if there is no other matching M′ in I that is more popular than M [Abraham et al., 2007a]. In Figure 1.2, consider the three symmetrical matchings:

$$M_1 = \{(a_1, p_1), (a_2, p_2), (a_3, p_3)\}$$
$$M_2 = \{(a_1, p_3), (a_2, p_1), (a_3, p_2)\}$$
$$M_3 = \{(a_1, p_2), (a_2, p_3), (a_3, p_1)\}$$

It is easy to verify that none of these matchings is popular, since $M_1 < M_2$, $M_2 < M_3$, and $M_3 < M_1$. In fact, this instance admits no popular matching, the problem being that the more popular than relation is not acyclic. The popular matching problem is to determine if a given instance admits a popular matching and to find such a matching, if one exists.

Gardenfors [Gardenfors, 1975] first introduced the notion of a popular matching, the author referres to this concept as a majority assignment in the context of voting theory. We remark that the more popular concept can be traced back even further to the Condorcet voting protocol. Popular matchings were then considered by [Sng and Manlove, 2010] in the context of the House Allocation (HA) problem. They showed that popular matchings need not exist, given an instance of HA, and also noted that popular matchings can have different cardinalities. In [Abraham et al., 2007a] authors described an $O(n + m)$ algorithm for finding a maximum cardinality popular matching (henceforth a maximum popular matching) if one exists, given an instance of HA. They also described an $O(\sqrt{n}m)$ counterpart for the House Allocation with ties (HAT) problem.

There exist in the literature a number of efficient algorithms for solving popular matching problems, e.g. [Abraham et al., 2007b]. In [Huang and Kavitha, 2013, Biró, 2008] the authors design the popular matching for the stable marriage problem and the roommates problem. In [Acharyya et al., 2014] they count the number of popular matchings

in the House Allocation Problem. In [Mestre, 2006] the author studied the weighted version of popular matching problem. However, in real world situations, additional constraints are often needed. In many cases, the new problem is intractable and the original algorithms become useless.

1.4 Kidney Exchange Problem

The Kidney exchange is a recent innovation that matches patients in need of a kidney to willing donors. In a non-bipartite graph, we can model the kidney exchange problem by constructing a pair (patient, donor) for each patient, and an directed edge between any two pairs where the incompatible donor for one patient is compatible with the other patient, and viceversa. An edge (p_i,p_j) in a matching corresponds to a pairwise kidney exchange, in which p_j receives a kidney from p_i's donor in exchange for p_i receiving a kidney from p_j's donor. Preference lists can be constructed on the basis of compatibility profiles between donors and patients.

For example, (Mary and Carlos) and (Amir and Shauna) are two incompatible pairs, where Mary wants to donate a kidney to Carlos, but they do not match; Amir wants to donate to Shauna, but they do not match. Mary is a match for Shauna, and Amir matches Carlos. Swapping donors and patients in this case allows both transplants to happen. An exchange cycle (or simply "cycle") involves a sequence of matches where the donor of one pair donates to the candidate in the next pair along the cycle. The cycle is completed when the donor in the last pair gives a kidney to the candidate in the first pair [Roth et al., 2005]. An "altruistic" donor can initiate chains of transplants in the pool that end by transplanting a candidate on the deceased donor waiting list, called domino-paired donation (DPD) [Roth, 2008].

Traditionally, a KEP pool is managed through a sequence of match runs, whereby at regular intervals, the pool is assessed and a solution consisting of cycles and chains is determined such that no pair is simultaneously involved in more than one cycle or chain. In larger pools, there are many cycles and/or chains and consequently many possible solutions. The preferred selection would ideally be determined by some objective function, such as maximising the total number of potential transplants [Roth et al., 2004]. In cycles there is a practical limitation that all transplants should be performed simultaneously in order to avoid the possibility of a scheduled donor opting to leave the pool prior to donation. Without this restriction, the possibility exists that a donor will donate a kidney without the associated candidate obtaining a

transplant. For this reason, if any one of the transplants in a proposed cycle cannot be completed, none of the selected transplants in the cycle can proceed. On the other hand, if one of the transplants in a proposed chain segment cannot be completed, transplants prior to the point of failure can still proceed since the issue of an untransplanted candidate with no donor does not arise [Ashlagi et al., 2011, Anderson et al., 2015]. Failure to proceed with a proposed solution can occur for a number of reasons, including a positive laboratory crossmatch, a candidate or physician declining an assigned donor, or donors or candidates having to leave the pool due to illness or other reasons [Dickerson et al., 2013].

Recent studies suggest that optimisation methods which take into account the probability that selected transplants fail to proceed to actual transplantation can improve upon schemes that ignore this uncertainty [Dickerson et al., 2013, Alvelos et al., 2015, Pedroso, 2014]. However, these programs run offline, at prefixed intervals, we argue that the problem is intrinsically a dynamic problem which evolves in time. In [Ünver, 2010] he provides initial analysis on dynamic kidney exchange, but does not include chains in the program. In [Pedroso, 2014] they develop a dynamic scheme, considering even the case of compatible pairs, however the program still runs at fixed terms. In [Dickerson and Sandholm, 2015] the authors build a future match framework, learning the potentials of pairs in an offline graph.

1.5 Contribution and Thesis Structure

In this section, we discuss the structure of this thesis, along with the contributions of each chapter and the overall contributions of the methods presented in the thesis.

Background

Chapter 2 begins by presenting an introduction to the field of optimisation in combinatorial problems. We give a brief overview of several important results from matching theory; some of these will subsequently be used by the algorithms that we will describe in this thesis. Instances of constraint programming, integer programming and Logical-based Benders Decomposition are shown as high level encoding.

Popular Matching Problem

In Chapter 3 we address the popular matching problem. First we show how to encode the problem in a constraint programming context. Second, we show how to optimally

solve the popular matching with ties in the preference list. Third, we design solutions for the instances that does not admit a popular matching, which we call this problem the popular matching problem with copies. All the popular matching problems mentioned will be solved using global constraints, and this material has appeared in the following publications:

- *Danuta Sorina Chisca, Mohamed Siala, Gilles Simonin, and Barry O'Sullivan. "A CP-based approach for popular matching". Published in the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona, USA, pages 4202–4203, 2016. [Chisca et al., 2016]*
- *Danuta Sorina Chisca, Mohamed Siala, Gilles Simonin and Barry O'Sullivan. "New Models for Two Variants of Popular Matching". Published in: 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), Boston, USA. [Chisca et al., 2017]*

Kidney Exchange Problem

In most matching schemes of the kidney exchange problem, the main goal is to maximise the number of transplants, running the program at fixed intervals.

However, in Chapter 4 we consider a *dynamic* setting of kidney exchange matching problem, where patients come and leave over time. Online decision making can be framed as a form of multi-stage stochastic optimisation. This approach typically estimates future developments by sampling scenarios and optimise the expected value of current decisions. In Chapter 4 we build an online anticipatory algorithm and we provide an expected value estimation that is correct within the limit of sampling error. Secondly, we propose a new anticipatory algorithm which is sampling-free based and develop an abstract exchange graph for a probabilistic method. These contributions have appeared in the following publications:

- *Danuta Sorina Chisca, Michele Lombardi, Michela Milano and Barry O'Sullivan. From Offline to Online Kidney Exchange Optimization. Published in: 2018 IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, Volos, Greece. [Chisca et al., 2018]*
- *Danuta Sorina Chisca, Michele Lombardi, Michela Milano and Barry O'Sullivan. A Sampling-free Anticipatory Algorithm for the Kidney Exchange Problem. In Proceedings of the 16th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2019, Thessaloniki, Greece. [Chisca et al., 2019b]*

We will also consider uncertainty when matching, i.e. a pair worsening once a solutions is found. Many researchers focused on pair/edge failure, though we seek to develop a method that finds a solution which is robust to these type of failures. If an incompatibility involving one of the donors and her recipient occurs in an identified solution, the swap is cancelled and the remaining pair must wait for the next run.

In Chapter 5, we investigate the use of Logic Based Benders Decomposition to find a super solution for general disruptions, with application to the KEP. The master deals with the original problem, while subproblems try to find repair solutions for each possible disruption. The results are presented in the following publication and are elaborated in In Chapter 5:

- *Danuta Sorina Chisca, Michele Lombardi, Michela Milano and Barry O'Sullivan. Logic-Based Benders Decomposition for Super Solutions: an Application to the Kidney Exchange Problem. In Proceedings of 25th International Conference on Principles and Practice of Constraint Programming, CP 2019, Stamford, USA. [Chisca et al., 2019a]*

Conclusions Finally, in Chapter 6 we present some general concluding remarks and outlook of future extensions of the work presented in this dissertation.

Chapter 2

Background

"I have been impressed with the urgency of doing. Knowing is not enough; we must apply. Being willing is not enough; we must do." Leonardo da Vinci

In this chapter we provide the necessary background to understand the matching problems. Moreover, we will focus on optimisation problems, popular matchings, and kidney exchanges. For each of them we introduce the necessary background.

2.1 Optimisation

Numerous industrial applications of combinatorial optimisation require going beyond a single satisfiable solution. Frequently the interest is in finding good, or the absolute best, quality solution. For example, we might wish to define the objective function to maximise profit, produce customer satisfaction, or to minimise cost, loss function. Due to the potential size of the applications in practice, it is usually infeasible to compute optimal allocations by hand. Centralised matching schemes automate this task by employing algorithms to compute optimal matchings based on the input data supplied by the participants.

A combinatorial optimisation model of a problem consists of a declarative specification, separating the problem in the formulation and the search strategy. However, modelling a problem is difficult as a solvable model is needed to quickly finds optimal solutions or determines that none exist. Naturally, there are many alternative models for a single problem; often it is not clear which one is best. In an optimisation setting,

an objective function needs to be specified and optimised, i.e. maximise or minimise an utility function. Once the model is defined it can be passed off to a solver which will search for an optimal solution. Often, we may need to specify some heuristics about how the solver should perform the search, such as the variable or value ordering before the solver can effectively solve the problem.

Two common issues arise in the development of a solution: the model either does not accurately represent the problem, or a solution is not found by the solver in reasonable time. The former is more of a real-world problem requiring the assistance of a domain expert, while latter may require the input of an expert in combinatorial optimisation.

2.1.1 Constraint Programming

The field of constraint programming (CP) is rapidly progressing. Constraint programming [Rossi et al., 2006] is a powerful paradigm for solving combinatorial search problems using a wide range of techniques from artificial intelligence, computer science, and operations research. Constraint programming is currently applied with success to many domains, such as scheduling, planning, vehicle routing, etc. The basic idea in constraint programming is that the user states the constraints and a general purpose constraint solver is used to solve them. Constraints are just relations, and a constraint satisfaction problem (CSP) states which relations should hold among the given decision variables. For example, in scheduling activities in a company, the decision variables might be the starting times and the duration of the activities and the resources needed to perform them, and the constraints might be on the availability of the resources and on their use for a limited number of activities at a time. Constraint solvers take a real-world problem like this, represented in terms of decision variables and constraints, and find an assignment to all the variables that satisfies the constraints.

The constraint satisfaction problem (CSP) [Rossi et al., 2006] is defined by a tuple $\langle X, D, C \rangle$, defining the variables, domains, and constraints respectively. A solution to a CSP consists of a mapping from each variable in X to one of the values in its domain, such that all constraints in C are satisfied. A CP solver works by systematically trying all possible assignments of values to the variables in a problem, to find the feasible solutions.

Example: the 4-Queens Problem[1] consists in placing four queens on a 4×4 chessboard, see Figure 2.1 such that no two queens are attacking each other.

[1] https://developers.google.com/optimization/cp/queens

There are two key elements to a constraint programming search:

- Propagation — Each time the solver assigns a value to a variable, the constraints add restrictions on the possible values of the unassigned variables. For example, in the 4-queens problem, each time the solver places a queen, it can't place any other queens on the row and diagonals the current queen is on. Propagation can speed up the search significantly by reducing the set of variable values the solver must explore.
- Backtracking occurs when either the solver can't assign a value to the next variable, due to the constraints, or it finds a solution, therefore the solver backtracks to a previous stage and changes the value of the variable at that stage to a value that hasn't already been tried. In the 4-queens example, this means moving a queen to a new square on the current column.

For instance, to model the 4-Queens Problem, we can create four variables $queens$, each of them with domain $D = \{1, 2, 3, 4\}$. For any solution, queens[i] = j means there is a queen in column i and row j.

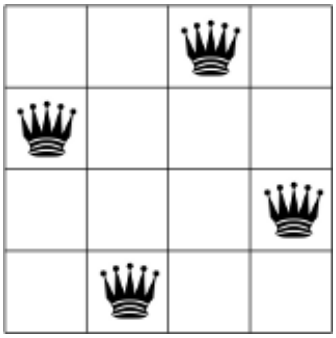


Figure 2.1: 4-Queens problem

These constraints guarantee the three conditions for the 4-queens problem:

- no two queens on the same row
- no two queens on the same column
- no two queens on the same diagonal

These constraints can be added efficiently to the CP model. The 4-Queens problem can be extended to N-queens, but the number of solutions goes up roughly exponentially with the size of the board.

2.1.1.1 Global Constraints

Global constraints define constraints over an arbitrarily sized set of variables, presenting many benefits for constraint programming [van Hoeve and Katriel, 2006]. Notably, they can convey complex relationships between variables, allowing for a concise specification of a problem. More importantly, from a pragmatic perspective, this enables higher levels of reasoning to be performed by dedicated inference algorithms, reducing the search space significantly. For example, propagation for global constraints such as *AllDifferent* and cardinality constraints can be achieved in low polynomial time using flow-based algorithms [Régin, 1994].

The Global Constraint Catalogue [Beldiceanu et al., 2007] collects definitions for all global constraints defined in the CP literature. Such a vast catalogue provided many opportunities for the application of constraint programming, however one practical issue faced by users is in identifying which one is appropriate for their problem.

All-Different

One of the most widely known, intuitive, and well studied global constraints is the AllDifferent constraint [Régin, 1994] which simply specifies that a set of variables must be assigned distinct values. Such a relation arises in many practical applications such as resource allocation, e.g. to state that a resource may not be used more than once at a single time point.

The model for the example in Figure 2.1 can can use a global constraint such as *AllDifferent* to efficiently solve the problem. One solution of 4-Queens is shown in Figure 2.1.

Global Cardinality

The global cardinality constraint [Régin, 1996] places lower and upper bounds on the number of occurrences of certain values amongst a set of variables. The global cardinality constraint $gcc(x_1, \ldots, x_n, c_{v_1}, \ldots, c_{v_{n'}})$ is a generalization of AllDifferent. While AllDifferent requires that every value is assigned to at most one variable, the gcc is specified on n assignment variables $x_1, \ldots, x_n$ and n' count variables $c_{v_1}, \ldots, c_{v_{n'}}$ and specifies that each value v_i is assigned to exactly c_{v_i} assignment variables. AllDifferent, then, is the special case of gcc in which the domain of each count variable is $\{0, 1\}$. For any tuple $t \in D^n$ and value $v \in D$, let $occ(v, t)$ be the number of occurrences of v in t.

The global cardinality constraint models restrictions in applications such as timetabling when there may be a limit on the number of consecutive activity types. An example of a problem that can be modeled with a gcc is the shift assignment problem [Régin, 1994, van Hoeve and Katriel, 2006] in which we are given a set of workers $W = \{W_1, \ldots, W_s\}$ and a set of shifts $S = \{S_1, \ldots, S_t\}$ and the problem is to assign each worker to one of the shifts while fulfilling the constraints posed by the workers and the boss: Each worker W_i specifies in which of the shifts she/he is willing to work and for each shift S_i the boss specifies a lower and upper bound on the number of workers that should be assigned to this shift. In the gcc, the workers would be represented by the assignment variables and the shifts by the count variables. The domain of an assignment variable would contain the set of shifts that the respective worker is willing to work in and the interval corresponding to each count variable would match the lower and upper bounds specified by the boss for this shift.

2.1.2 Integer Programming

An integer programming problem is a mathematical optimisation or feasibility problem, in which some or all the variables are restricted to be integers. Example of an optimisation problem:

$$\max \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

subject to:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad \forall i \in m \tag{2.2}$$

$$x_j \geq 0 \qquad \forall j \in n \tag{2.3}$$

$$x_j \text{ integer} \qquad \text{for some or all } j \in n \tag{2.4}$$

This problem is called the linear integer-programming problem. It is said to be a *mixed integer program* when some, but not all, variables are restricted to be integer, and is called a pure integer programming when all decision variables must be integers. The case where the integer variables are restricted to be 0 or 1 comes up surprising often. Such problems are called pure (mixed) 0-1 programming problems or pure (mixed) binary integer programming problems.

Integer-programming models arise in practically every area of application of mathematical programming. One particularly simple integer problem is the Knapsack prob-

lem [Martello and Toth, 1990]. The traditional story is that there is a knapsack, (with capacity w_0), there are a number of items, each with a weight (w) and a value (c). The objective is to maximise the total value of the items in the knapsack. These can be written as:

$$\max \sum_{j=1}^{n} c_j x_j \tag{2.5}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq w_0 \tag{2.6}$$

$$x_j \in \{0,1\} \forall j \in n \tag{2.7}$$

Assume for example we have a knapsack (with capacity 14), and 4 items, i.e. e item 2 has weight 7 and value 11: A practical example is

$$\max 8x_1 + 11x_2 + 6x_3 + 4x_4 \tag{2.8}$$

$$\text{s.t.} 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \tag{2.9}$$

$$x_j \in \{0,1\} \tag{2.10}$$

The solution $x_1 = 1$, $x_2 = 1$, $x_3 = 0.5$, $x_4 = 0$, with value 22 is the optimal solution to the linear program, even though is not integral. An integer solution is a $x_1 = 0$, $x_2 = 1, x_3 = 1, x_4 = 1$ with a value of 21.

2.1.3 Super Solutions

When dealing with real world problems, efficiency and optimality may not be the only concerns. For instance, in the kidney exchange problem further tests are performed just hours before the surgery, and this may affect the solution found, i.e. the patient/donor worsening. The problem is to find a matching for the kidney exchange problem resilient to small perturbation. The notion of fault tolerance has been introduced in the propositional satisfiability framework by [Ginsberg et al., 1998]. A solution is fault tolerant if a small change in the model is guaranteed not to result in a large perturbation of the solution.

However, one might want the new solution to be as close as possible to the old solution. Indeed it might not be practical to change too many surgeries on short notice. A

solution where a small perturbation, such as a pair pooling out from the solution, can be repaired with small changes. The notion of super-models [Ginsberg et al., 1998] or σ-models [Roy, 2001] has first been defined for Boolean satisfiability problems:

> "To model this concept of fault tolerance we introduce the notion of σ-models: these are satisfying assignments of Boolean formula for which any small alteration, such as a single bit flip, can be repaired by another small alteration, yielding a nearby satisfying assignment."

An (a, b) super model α is a model of a Boolean formula such that for any set of atoms A, if $|A| \leq a$, then there exists a disjoint set B such that $|B| \leq b$ and if we negate $\alpha(A \cup B)$ we obtain another model. Below is the definition of (1, 1) super models from [Roy, 2001, Roy, 2006]:

Definition 2.1. *A super-model of a Boolean formula F is a satisfying assignment α of F, $F(\alpha) = 1$, such that for every i, if we negate the i-th bit of α, there is another bit $j \neq i$ of α which we can negate to get another satisfying assignment.*

Flipping a bit of a σ-model is called a *break*, corresponding to a "bad" event. The bit that is flipped to get another satisfying assignment is a *repair*.

In [Roy, 2006] the author extended the notion of single repairability to repairability of a sequence of breaks to a model.

Definition 2.2. *A σ(r, s)-model of a Boolean formula Θ is a model of Θ such that for every choice of at most r bit flips (the "break" set) of the model, there is a disjoint set of at most s bits (the "repair" set) that can be flipped to obtain another model of Θ.*

[Hebrard et al., 2004b] generalise this definition to constraint satisfaction on non-Boolean domains.

Definition 2.3. *A solution to a CSP is (a, b)-super solution iff the loss of the values of at most a variables can be repaired by assigning other values to these variables, and modifying the assignment of at most b other variables.*

Let us consider the following CSP example from [Hebrard et al., 2004b]: $X, Y, Z \in \{1, 2, 3\}$, $X \leq Y \wedge Y \leq Z$. The solutions to this CSP are:

<1, 1, 1>, <1, 1, 2>, <1, 1, 3>, <1, 2, 2>, <1, 2, 3>, <1, 3, 3>, <2, 2, 2>, <2, 2, 3>, <2, 3, 3>, <3, 3, 3>.

The subsets of the solutions that are (1, 1)-super solutions are:

<1,1,2>, <1,1,3>, <1,2,2>, <1,2,3>, <1,3,3>, <2,2,2>, <2,2,3>, <2,3,3>

and (1, 0)-super solutions are:

<1, 2, 3>, <1, 2, 2>, <2, 2, 3>.

The solution <1,1,1> is not a (1,0)-super solution. If X loses the value 1, there is no repair value for X that is consistent with Y and Z since neither <2, 1, 1> nor <3, 1, 1> are solutions. Also, solution <1, 1, 1> is not a (1, 1)- super solution since when X loses the value 1, there is no repair by changing the value assigned to at most one other variable, i.e., there exists no repair solution when X breaks since none of <2,1,1>, <3,1,1>, <2,2,1>, <2,3,1>, <2,1,2>,<2, 1, 3> is a solution. On the other hand, <1, 2, 3> is a (1, 0)-super solution given that when X breaks the repair solution is <2, 2, 3>, when Y breaks the repair solution is <1, 1, 3>, and when Z breaks the repair solution is <1, 2, 2>.

A number of properties follow immediately from the definition. For example, a (c, d)-super solution is a (a, b)-super solution if $(a \leq c$ or $d \leq b)$ and $c + d \leq a + b$. Deciding if a SAT problem has an (a,b)-supermodel is NP-complete [Ginsberg et al., 1998]. In [Hebrard et al., 2004b] the authors shows that deciding if a CSP has an (a,b)-super solution is also NP-complete, even when restricted to binary constraints:

Theorem 2.1. *Deciding if a CSP has an (a, b)-super solution is NP-complete for any fixed* a.

Furthermore they proposed a method to find (1, 0)-Super Solutions via Search. Therefore, by choosing a (1,0)-super-solution over other solutions, we deal with one perturbation without changing other assignments, which is often a valuable property.

In [Holland and O'Sullivan, 2004] the authors proposed a novel application of super solutions to combinatorial auctions in which a bid may be disqualified or withdrawn after the winners are announced. In [Holland and O'Sullivan, 2005b] they proposed a Weighted Super Solutions to find a repair solution whose revenue exceeds some threshold while limiting the cost associated with forming such a repair.

A more recent paper on fault tolerant solutions [Bofill et al., 2013] using SAT encoding, improves the work of [Ginsberg et al., 1998] using weighted Partial MaxSAT formulation. Their approach strengthens the complexity to $\mathcal{O}(n^a)$ for each handled variable change, where n is the number of variables, and a is the number of breaks, instead of $\mathcal{O}(n^{a+b})$ of [Ginsberg et al., 1998], where b is the number of repairs.

2.1.4 Logic-based Benders Decomposition

The Logic-based Benders Decomposition (LBBD) approach has been used so solve allocation and scheduling problems [Hooker and Ottosson, 2003, Hooker, 2007] and resource allocation [Benini et al., 2011]. LBBD is a classic operational research technique, which assigns some of the variables trial values and finds the best solution consistent with these values. In the process it learns informations to reduce the number of solutions. This strategy is known as "learning from one's mistakes", as exclude superfluous solutions. The central element of Benders decomposition is the derivation of Benders cuts that exclude superfluous solutions. Classical Benders cuts are formulated by solving the dual of the subproblem that remains when the trial values are fixed. The subproblem must therefore be one for which dual multiplers are defined, such as a linear or nonlinear programming problem.

Fixing some primary variables to these values, solves a subproblem for each set of values. The solution of the subproblem is used to generate a cut, i.e. Bender's cut, that the primary variables must satisfy in all solutions enumerated.

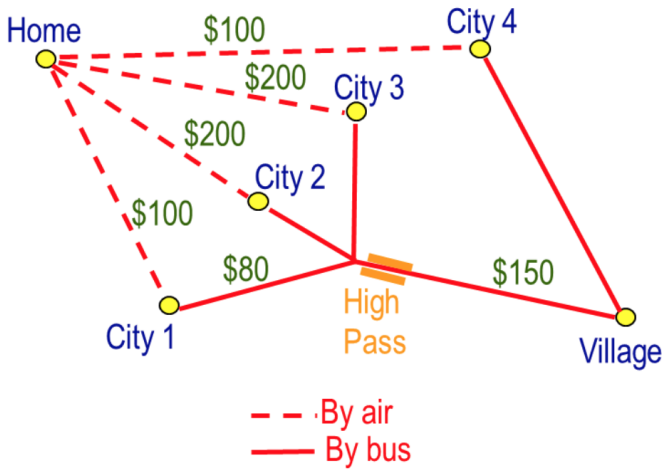


Figure 2.2: Example: remote village

Assume the following example [Hooker, 2016]: Find the cheapest route to a remote village with the values shown in Figure 2.2, the village can be reached by bus or air. Let be x flight destination and y the bus route, we seek to find cheapest route (x,y). The master problem solve the cheapest flight x, subject to Benders cuts generated so far, while the subproblem search to find the cheapest bus route from the airport to the village. The master problem and the subproblem interactions as in Figure 2.3.

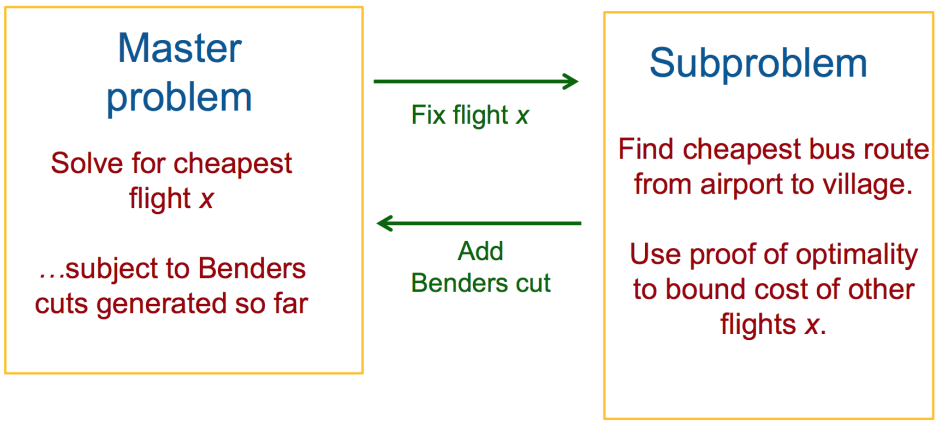


Figure 2.3: Decomposition of master - subproblem

We start with x = City 1 and pose the subproblem: find the cheapest route given that x = City 1. Optimal cost is \$100 + 80 + 150 = \$330.

The dual problem of finding the optimal route is to prove optimality. The proof is that the route from City 1 to the village must go through High Pass. So cost $\geq$ airfare + bus from city to High Pass + \$150. But this same argument applies to City 1, 2 or 3. Specifically the Benders cut is:

$$cost \geq B_{city1}(x) = \begin{Bmatrix} \$100 + 80 + 150 & \text{if x = City 1} \\ \$200 + 150 & \text{if x = City 2,3} \\ \$100 & \text{if x = City 4} \end{Bmatrix}$$

Now we need to solve the master problem, thus we need to pick the city x to minimize cost.

Clearly the solution is x = City 4, with cost \$100. Now let x = City 4 and pose the subproblem: Find the cheapest route given that x = City 4. Optimal cost is \$100 + 250 = \$350.

Again solve the master problem: Pick the city x to minimize cost subject to:

$$cost \geq B_{city4}(x) = \begin{cases} \$350 & \text{if x = City 1} \\ \$0 & \text{otherwise} \end{cases}$$

The solution is x = City 1, with cost $330. Because this is equal to the value of a previous subproblem, we are done.

Classical Benders cuts are formulated by solving the dual of the subproblem, when some values are fixed. Formally a general optimization problem can be written as:

$$\min f(x) \qquad OP \tag{2.11}$$
$$s.t. \tag{2.12}$$
$$x \in S \tag{2.13}$$
$$x \in D \tag{2.14}$$

where $f(x)$ is a generic function, each variable with domain D and a set of constraints $S \subseteq D$.

Assume P and Q to be two propositions whose truth or false is a function of x. Then P implies Q with respect to D if Q is true for any $x \in D$ for which P is true, and can be written as P $\xrightarrow{D}$ Q.

The inference dual of OP_0 can be written as:

$$\max \beta \qquad ID \tag{2.15}$$
$$s.t. \tag{2.16}$$
$$x \in S \xrightarrow{D} f(x) \geq \beta \tag{2.17}$$

The dual problem finds the tightest possible bound on the objective function value.

The optimal value of the inference dual is the same as the original problem, given the strong duality property.

LBBD Structure

Benders decomposition views elements of the feasible set as pairs (x, y) of objects that belong respectively to domains D_x , D_y .

Therefore the optimisation problem OP_0 becomes:

$$\min f(x, y) \qquad OP \tag{2.18}$$
$$s.t. \tag{2.19}$$
$$(x, y) \in S \tag{2.20}$$
$$x \in D_x, y \in D_y \tag{2.21}$$

A general LBBD algorithm begins by fixing the variables y to some values $\bar{y} \in D_y$ in the master problem (MP), see Figure 2.4. This leads to the following *subproblem*, (SP):

$$\min f(x, \bar{y}) \qquad SP \tag{2.22}$$
$$s.t. \tag{2.23}$$
$$(x, \bar{y}) \in S \tag{2.24}$$
$$x \in D_x \tag{2.25}$$

Rather than solving the subproblem directly, we can solve the corresponding inference dual, known to have the same optimal value.

$$\max \beta \qquad DP \tag{2.26}$$
$$s.t. \tag{2.27}$$
$$(x, \bar{y}) \in S \xrightarrow{D_x} f(x, \bar{y}) \geq \beta \tag{2.28}$$

The dual problem is to find the best possible lower bound β^* on the optimal cost that can be inferred from the constraints, assuming y is fixed to $\bar{y}$. The optimal solution of subproblem is the same as the tightest bound on $f(x, y)$ for any fixed value of y, due to strong duality, which can be inferred from the constraint.

Just how this is done varies from one context to another, but essentially the idea is this. Let β^* be the value obtained for DP, which means that β^* is a lower bound on the optimal value of OP, given that $y = \bar{y}$. The solution of OP is a proof of this fact. This bound is expressed as a function $\beta_{\bar{y}y}$ of y, producing a Benders cut $z \geq \beta_{\bar{y}y}$. The subscript $\bar{y}$ of the bounding function denotes the y values used for its construction.

The algorithm proceeds as follows. At each iteration the Benders cuts so far generated comprise the constraints of a master problem,

$$\min z \qquad MP \tag{2.29}$$

$$s.t. \tag{2.30}$$

$$z \geq \beta_{\bar{y}k} \qquad \forall k = 0, 1... \tag{2.31}$$

$$y \in D_y \tag{2.32}$$

where $\bar{y}_1, \bar{y}_2, ...$ are the trial values till now obtained. The next trial value $(\bar{z}, \bar{y})$ of (z, y) is obtain by solving the master problem. If the optimal value $\beta*$ of the resulting subproblem DP is equal to $\bar{z}$. Otherwise a new Benders cut $z \geq \beta_{\bar{y}y}$ is added to the master problem. At this point the algorithm repeats. The algorithm terminates when the master problem and subproblem value converge, that is $\bar{z} = \beta^*$.

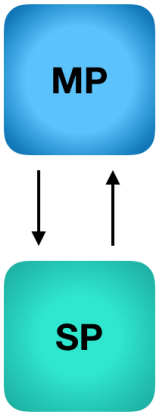


Figure 2.4: Master problem and sub-problem

In Figure 2.4 we show how the MP and SP communicates. First, the problem at hand is first decomposed into a master problem (MP) and a subproblem (SP). After the method works by iteratively solving MP and feeding SP with the partial solution from MP. If such solution cannot be extended to a complete one, a cut (Benders' cut) is generated forcing MP to yield a different solution. In case the extension is successful we have a feasible solution. The efficiency of the technique greatly depends on the possibility to generate strong cuts.

Theorem 2.2. *Suppose that in each iteration of the generic Benders algorithm, the bounding function $\beta_{\bar{y}}$ satisfies the following:*

The Benders cut $z \geq \beta_{\bar{y}}$ is valid; i.e., any feasible solution (x,y) of OP satisfies $f(x, y) \geq \beta_{\bar{y}}$.

If the algorithm terminates with a finite optimal solution $(z, y) = (\bar{z}, \bar{y})$ in the master problem, OP has an optimal solution $(x, y) = \bar{x}, \bar{y})$ with value $f(\bar{x}, \bar{y}) = \bar{z}$. If the algorithm terminates with an infeasible master problem, then OP is infeasible. If the algorithm terminates with an infeasible subproblem dual, then OP is unbounded. In general, the cuts can be refined to a specific problem and needs to be valid. Because all Benders cuts are valid, infeasibility of the master problem implies that the original problem is infeasible. Finally, if the subproblem dual is infeasible, then the subproblem is unbounded, which means the original problem is unbounded.

2.2 Matching Under Ordinal Preferences

Stable matching problems, introduced by [Gale and Shapley, 1962], have been exhaustively studied over recent decades. Different formulations are proposed, distinguishing between one-sided matching [Garg et al., 2010] and two-sided matching, e.g. the stable marriage (SM) problem [Gale and Shapley, 1962].

An instance of the classical Stable Marriage problem (SM) comprises a set of men and women, and each person ranks each member of the opposite sex in strict order of preference, and we seek to match one to another taking in consideration their preferences.

A generalisation of SM is the Hospitals / Residents problem (HR) [Gale and Shapley, 1962], where each man corresponds to a resident and each woman corresponds to a hospital which can potentially be assigned multiple residents up to some fixed capacity.

The Stable Roommates problem (SR) is the non-bipartite generalisation of SM in

which each agent ranks all of the others in strict order of preference. Stability is once again relevant in this context, and the definition of a stable matching is a straightforward extension of the definition in the SM case.

2.2.1 Stable Matching for Hospitals / Residents problem

The Hospitals / Residents problem (HR) [Gale and Shapley, 1962, Gusfield and Irving, 1989, Manlove, 2008] (or known as the College (or University or Stable) Admissions problem, or the Stable Assignment problem) was first defined by Gale and Shapley in their seminal paper "College Admissions and the Stability of Marriage" [Gale and Shapley, 1962].

An instance of HR involves a set $R = \{r_1, ..., r_{n1}\}$ of residents and a set $H = \{h_1, ..., h_{n2}\}$ of hospitals. Each hospital $h_j \in H$ has a capacity c_j, which indicates the number of posts that h_j has. Also there is a set $E \subseteq R \times H$ of acceptable resident–hospital pairs. Each resident $r_i \in R$ has an acceptable set of hospitals $A(r_i)$, where $A(r_i) = \{h_j \in H : (r_i, h_j) \in E\}$.

In the same way, each hospital $h_j \in H$ has an acceptable set of residents $A(h_j)$, where $A(h_j) = \{r_i \in R : (r_i, h_j) \in E\}$.

Residents' preferences	Hospitals' preferences
$r_1 : h_2 \quad h_1$	$h_1 :(2): r_1 \quad r_3 \quad r_2 \quad r_5 \quad r_6$
$r_2 : h_1 \quad h_2$	$h_2 :(2): r_2 \quad r_6 \quad r_1 \quad r_4 \quad r_5$
$r_3 : h_1 \quad h_3$	$h_3 :(2): r_4 \quad r_3$
$r_4 : h_2 \quad h_3$	
$r_5 : h_2 \quad h_1$	
$r_6 : h_1 \quad h_2$	

Figure 2.5: An instance of the Hospitals/Residents problem

An example with six residents and three hospitals, and their respective preferences, is shown in Figure 2.5.

A matching M is an assignment such that $|M(r_i)| \leq 1$ for each $r_i \in R$ and $|M(h_j)| \leq c_j$ for each $h_j \in H$, that is no resident is assigned to an unacceptable hospital, each resident is assigned to at most one hospital, and no hospital is oversubscribed).

Definition 2.4. *Let I be an instance of HR and let M be a matching in I [Manlove, 2013]. A pair $(r_i, h_j) \in E \setminus M$ blocks M, or is a blocking pair for M, if the following conditions are satisfied:*

A. r_i is unassigned or prefers h_j to $M(r_i)$;

B. h_j is undersubscribed or prefers r_i to at least one member of $M(h_j)$ (or both).

M is said to be stable if it admits no blocking pair. In Figure 2.5 the instance admits a stable matching $M = \{(r_1, h_2), (r_2, h_1), (r_3, h_1), (r_4, h_3), (r_6, h_2)\}$.

The algorithm in [Gale and Shapley, 1962], known as the resident-oriented Gale–Shapley algorithm (RGS), showed that every instance I of HR admits at least one stable matching.

Theorem 2.3. *Given any instance of HR, the RGS algorithm constructs, in $O(m)$ time, the unique resident-optimal stable matching, where m is the number of acceptable resident–hospital pairs [Gusfield and Irving, 1989, Manlove, 2013].*

The RGS algorithm returns the unique resident-optimal stable matching, in which each assigned resident has the best hospital that she could achieve in any stable matching, whilst each unassigned resident is unassigned in every stable matching [Gale and Shapley, 1962].

Equivalently to the RGS algorithm, also the hospital-oriented Gale–Shapley algorithm (HGS), involves hospitals offering posts to residents. The HGS algorithm returns with the unique hospital-optimal stable matching. In this matching, every hospital $h_j \in H$ is assigned its c_j best stable partners, whilst every undersubscribed hospital is assigned the same set of residents in every stable matching [Gusfield and Irving, 1989].

2.2.2 Stable Matching for Hospitals/Residents Problem with Indifference

In large-scale matching schemes, participants may not be able to provide a genuine strict preference order over what may be a very large number of hospitals, and may prefer to express indifference in their preference lists, (ties). The concept of indifference can be applied to instances of HR [Manlove, 2013].

A matching in the context of Hospitals/Residents problem with indifference (HRI), is a set of (resident, hospital) pairs such that no resident is assigned to more than one hospital and no hospital is over-subscribed. A matching is stable if it admits no

blocking pairs. We can define a blocking pair for an instance of HRI with respect to three different levels of stability, since the presence of ties forces us to extend the definition. In fact an hospital h can prefer r_i over r_j, r_j over r_i or to be indifferent between them, as we can see in the example below:

A pair $(r_i, h_j) \in R \times H$ is said to block a matching M for an instance of HRI, and is called a blocking pairs when:

A. weak stability:

   (a) r_i is unassigned or prefers h_j to his/her assigned hospital in M, and

   (b) h_j is undersubscribed or prefers r_i to its worst assigned resident in M;

B. strong stability: either

   (a) r_i is unassigned or prefers h_j to his/her assigned hospital in M, and

   (b) h_j is undersubscribed or prefers r_i to its worst assigned resident in M or is indifferent between them;

   or

   (a) r_i is unassigned or prefers h_j to his/her assigned hospital in M or is indifferent between them, and

   (b) h_j is undersubscribed or prefers r_i to its worst assigned resident in M;

C. super-stability:

   (a) r_i is unassigned or prefers h_j to his/her assigned hospital in M or is indifferent between them, and

   (b) h_j is undersubscribed or prefers r_i to its worst assigned resident in M or is indifferent between them.

M is said to be be weakly stable, strongly stable or super-stable if it admits no blocking pair with respect to the relevant definition above.

[Gale and Shapley, 1962] showed that an instance of Stable roommates (SR) needs not admit a stable matching. The following result, due to Irving in [Irving, 1985], indicates that it is possible to determine in polynomial time whether a given instance admits a stable matching, and if so, to find such a matching.

Similarly, other stable matching problems arise naturally in practical applications, where preference lists need not include indifference. These include the Student /

Project Allocation problem (SPA), a generalisation of HR, where students are to be assigned to projects offered by lecturers, subject to capacity constraints involving both projects and lecturers, and preference lists supplied by both students and lecturers.

2.2.3 Stable Matching for Stable Roommates Problem

The Stable Roommates problem (SR) [Gale and Shapley, 1962, Irving, 1985, Gusfield and Irving, 1989] is a non - bipartite generalisation of Stable Matching.

An instance of SR is a single set $A = \{a_1, ..., a_n\}$ of agents, and n denotes the size of the instance. Also there is a set E of un-ordered pairs of agents, called acceptable pairs. Each agent $a_i \in A$ has an acceptable set of agents $A(a_i)$, where $A(a_i) = \{a_j \in E : \{a_i, a_j\} \in E\}$. An assignment M is a subset of E. If $\{a_i, a_j\} \in M$, a_i is said to be assigned to a_j. With respect to M, if a_i is not assigned to any agent then a_i is said to be unassigned, otherwise a_i is assigned. An instance of six roommates and their preferences is shown in Figure 2.6.

$$
\begin{array}{lllllll}
a_1: & a_3 & a_4 & a_2 & a_6 & a_5 \\
a_2: & a_6 & a_5 & a_4 & a_1 & a_3 \\
a_3: & a_2 & a_4 & a_5 & a_6 & a_6 \\
a_4: & a_5 & a_2 & a_3 & a_6 & a_1 \\
a_5: & a_3 & a_1 & a_2 & a_4 & a_6 \\
a_6: & a_5 & a_1 & a_3 & a_4 & a_2
\end{array}
$$

Figure 2.6: An instance of the Stable Roommates problem

A matching M is an assignment such that no agent belongs to more than one pair of M. If $\{a_i, a_j\} \in M$ then we let $M(a_i)$ denote a_j. Let A_M denote the set of agents who are assigned in M.

A pair $\{a_i, a_j\} \in E \setminus M$ blocks a matching M, or is a blocking pair for M, if the following conditions:

- a_i is unassigned or prefers a_j to $M(a_i)$;
- a_j is unassigned or prefers a_i to $M(a_j)$.

A matching M is said to be stable if it admits no blocking pair. An instance I of SR is said to be solvable if I admits a stable matching; I is unsolvable otherwise. The solution of Figure 2.6 is the matching $\{(a_1, a_6), (a_2, a_4), (a_3, a_5)\}$, which is stable.

Recently, an application of an extension of SR, called the Stable Roommates problem with Incomplete lists (SRI).

2.3 One-sided Case: Popular Matching

The notion of *popularity* was introduced by Gardenfors [Gardenfors, 1975] in the full stable marriage problem, where it is a desirable property in finding a weakly stable matching, but it has its origins from as far back as 1785 with the notion of Condorcet winner. Popular matching and its extensions have been an exciting area of research studied seriously over the last decade [Abraham et al., 2007b].

In the *popular matching* problem, a set of applicants $\mathcal{A}$ have to be matched to a set of posts $\mathcal{P}$ where each applicant has an ordinal list of preference over posts, ranking a subset of posts in order of preference. An example with three applicants and three posts and their preferences is shown in Figure 2.7.

$$a_1 : p_1\, p_2\, p_3$$
$$a_2 : p_1\, p_2\, p_3$$
$$a_3 : p_1\, p_2\, p_3$$

Figure 2.7: An instance of preference list

Notice that only the applicants have preferences over posts. Therefore, the popular matching problem considers only one-sided preferences. If applicants can be indifferent between posts, we say that preference lists contain ties.

We say that an applicant a prefers a matching M' to M if *(i)* a is matched in M' and unmatched in M, or *(ii)* a prefers $M'(a)$ to $M(a)$. M' is said *more popular* than M, denoted by $M' \succ M$, if the number of applicants that prefer M' to M exceeds the number of applicants that prefer M to M'. A matching M is popular if there exists no matching more popular than M.

Informally, a matching M can be seen as a set containing pairs $\langle a, p\rangle$ where a is an applicant and p is a post and each applicant/post appears at most once in M. Given two matchings M and M', we use the notation $M \prec M'$ if (strictly) more applicants prefer M' to M. A matching M is *popular if and only if* there is no matching M' such that $M \prec M'$.

We can easily check that the instance in Figure 2.7 does not admit a popular matching since for any matching M, there exists a matching M' such that $M \prec M'$. For instance,

consider the three (symmetrical) matchings $M_1 = \{(a_1, p_1), (a_2, p_2), (a_3, p_3)\}$, $M_2 = \{(a_1, p_3), (a_2, p_1), (a_3, p_2)\}$, and $M_3 = \{(a_1, p_2), (a_2, p_3), (a_3, p_1)\}$. We have $M_1 \prec M_2, M_2 \prec M_3$, and $M_3 \prec M_1$. In fact, it turns out that this instance admits no popular matching, the problem being that the more popular than relation is not transitive.

Popular matchings have been studied in house allocation problem (HA), house allocation with ties (HAT), capacitated house allocation (CHA), stable roommates with indifference (SRI). The house allocation problem (HA) is the variant of SM in which the women do not have preferences over the men. The men are referred to *applicants* and women are referred to as *houses*. A many-one extension of HA is the capacitated house allocation (CHA), where each house can accommodate multiple applicants up to some fixed capacity. CHA can also be seen as the variant of the HR where hospitals do not have preferences over residents.

2.3.1 Popular Matching for the House Allocation Problem

Many economists and game theorists have studied the problem of allocating a set of indivisible goods among a set of applicants [Manlove, 2013]. Each applicant $a_i \in \mathcal{A}$ may have ordinal preferences over a subset of H (houses), i.e. the acceptable goods for a_i. Many models have considered the case where there is no monetary transfer.

Formally, an instance I of HA comprises a set $A = \{a_1, a_2, ..., a_{n_1}\}$ of applicants and a set $H = \{h_1, h_2, ..., h_{n_2}\}$ of houses. There is a set $E \subseteq A \times H$ of acceptable applicant–house pairs. Let $m = |E|$. Each applicant $a_i \in A$ has an acceptable set of houses $A(a_i)$, where $A(a_i) = \{h_j \in H : (a_i, h_j) \in E\}$, as in Figure 2.8. Similarly each house has an acceptable set of applicants.

$$
\begin{aligned}
a_1 &: h_1, h_4 \\
a_2 &: h_2, h_5 \\
a_3 &: h_3, h_4, h_6 \\
a_4 &: h_1 \\
a_5 &: h_2 \\
a_6 &: h_3
\end{aligned}
$$

Figure 2.8: An istance of HA

Each applicant $a_i \in A$ has a preference list in which he/she ranks $A(a_i)$ in strict order. Given any applicant $a_i \in A$, and given any houses $h_j, h_k \in H$, a_i is said to prefer h_j to h_k if $\{h_j, h_k\} \subseteq A(a_i)$, and h_j precedes h_k on a_i's preference list. Notice that houses do not have preference lists over applicants.

An assignment M is a subset of E. If $(a_i, h_j) \in M$, a_i and h_j are said to be assigned to one another. For each $k \in A \cup H$, the set of assignees of k in M is denoted by $M(k)$. If $M(k) = 0$, k is said to be unassigned, otherwise k is assigned. A matching M is an assignment such that $|M(k)| \leq 1$ for each $k \in A \cup H$.

For each applicant $a_i \in A$ we create a new unique last resort house l_i and append l_i to a_i's preference list. This ensures that every applicant is assigned in a given matching in I. However in reality, $(a_i, l_i) \in M$ means that a_i is unassigned. For each applicant $a_i \in A$, we define $f(a_i)$ to be the first house in a_i's preference list. Given any house $h_j \in H$, we define $f(h_j)$ to be the set of applicants a_i such that $f(a_i) = h_j$. Then h_j is called an f-house if $f(h_j) \neq 0$. For a given applicant $a_i \in A$, we define $s(a_i)$ to be the first non f-house in a_i's preference list. A house $h_j \in H$ is called an s-house if $h_j = s(a_i)$ for some $a_i \in A$. By definition, the sets of f-houses and s-houses are disjoint.

Abraham et al. in [Abraham et al., 2007b] proved that $f(a_i)$ and $s(a_i)$ are the only candidate houses to which an applicant a_i can be assigned in a popular matching M. They also showed that every f-house must be assigned in M. Furthermore these two necessary conditions are sufficient for a matching to be popular.

Lemma 2.1. *[Abraham et al., 2007b] Let I be an instance of ha and let M be a matching in I. Then M is popular if and only if:*

- *every f-house is assigned in M;*
- *for each applicant $a_i \in A$, $M(a_i) \in \{f(a_i), s(a_i)\}$.*

Let the reduced graph G' be a subgraph of the underlying graph G of an istance I containing the vertices in $A \cup H$ and the edges $\{a_i, f(a_i)\}$ and $\{a_i, s(a_i)\}$ for each $a_i \in A$, meaning the edges of f-house and s-house. From this graph the theorem below follows:

Theorem 2.4. *[Abraham et al., 2007b] Let I be an instance of HA and let M be a matching in I. Then M is popular if and only if:*

- *every f-house is assigned in M;*
- *M is an applicant-complete matching in the reduced graph G′.*

Corollary 2.1. *[Abraham et al., 2007b] Let I be an instance of HA and let M be a matching in I. There is an $O(m)$ algorithm to check if M is popular in I, where m is the number of acceptable applicant–house pairs in I.*

The example in Figure 2.8 have the following popular matchings:

$$M_1 = \{(a_1, h_1), (a_2, h_2), (a_3, h_3)\}$$
$$M_2 = \{(a_1, h_1), (a_2, h_2), (a_3, h_4), (a_6, h_3)\}$$
$$M_3 = \{(a_1, h_1), (a_2, h_5), (a_3, h_4), (a_5, h_2), (a_6, h_3)\}$$

Popular matchings can have different sizes as we can see. However it may also be verified that the unique perfect matching in Figure 2.8 is not popular (as M_2 is more popular), and hence a maximum popular matching can be smaller than a maximum cardinality matching in the underlying graph.

2.3.2 Popular Matching for House Allocation Problem with Ties

The definition of $f(a_i)$ for an applicant $a_i \in A$ is straightforward, i.e. it is the set of a_i's most-preferred houses, first rank position. For each applicant $a_i \in A$, define $s(a_i)$ to be the most-preferred set of houses in a_i's preference list that are even in G_1, with G_1 being the reduced graph with only $f(a_i)$ edges. Also, in contrast to the HA case, it is possible that $s(a_i) \subseteq f(a_i)$.

Any popular matching must be a maximum matching in G_1. These two necessary conditions for a matching to be popular are also sufficient.

Lemma 2.2. *[Abraham et al., 2007a] Let I be an instance of the house allocation problem with ties (HAT) and let M be a matching in I . Then M is popular if and only if:*

- *M is a maximum matching in G_1;*
- *for each applicant $a_i \in A$, $M(a_i) \in f(a_i) \cup s(a_i)$.*

Theorem 2.5. *Let I be an instance of HAT and let M be a matching in I. Then M is popular if and only if:*

- *M is a maximum matching in G_1;*
- *M is an applicant-complete matching in the reduced graph G'.*

Theorem 2.6. *There is an $\mathcal{O}(\sqrt{n_1}m)$ algorithm to find a popular matching in an instance of HAT, or report that there is none, n_1 is the number of applicants and m is the number of acceptable applicant-house pairs.*

2.3.3 Maximum Popular Matchings

We would like to remark that popular matchings can have different sizes, and the authors of [Abraham et al., 2007a] showed how to extend their algorithm in order to find a maximum popular matching without altering the time complexity.

Theorem 2.7. *[Abraham et al., 2007a] Let I be an instance of HA. There is an $\mathcal{O}(n + m)$ algorithm to find a maximum popular matching or report that no popular matching exists, where n is the number of applicants and houses, and m is the number of acceptable applicant–house pairs.*

They also defined a more complex algorithm, with $O(\sqrt{n}m)$ complexity, which can be used to find a maximum popular matching or report that no popular matching exists, in the case that preference lists include ties.

2.3.4 Popular Matching for Capacitated House Allocation Problem with Ties

Popular matchings can be defined in instances of capacited house allocation (CHA) and capacited house allocation with ties (CHAT), where each house h_j has a capacity $c_j > 0$.

An instance I of CHA comprises a bipartite graph $G = (A, H, E)$, defined similary as HA in Section 2.3.1. Each house $h_j \in H$ has a capacity $c_j \geq 1$ which indicates the maximum number of agents that may be matched to it. We assume that $m \geq max\{n_1, n_2\}$, i.e. no agent has an empty preference list and each house is acceptable to at least one agent. We also assume that $c_j \leq n_1$ for each $h_j \in H$. A matching M in I is a subset of E such that

- each agent is matched to at most one house in M,
- each house $h_j \in H$ is matched to at most c_j agents in M.

Manlove and Sng [Manlove and Sng, 2006] established the following characterisation of popular matchings in CHA.

Theorem 2.8. *Let I be an instance of CHA and let M be a matching in I. Then M is popular if and only if:*

- *for every f-house $h_j \in H$,*

- *if* $f_j \leq c_j$ *then* $f(h_j) \subseteq M(h_j)$*;*
- *if* $f_j > c_j$ *then* $|M(h_j)| = c_j$ *and* $M(h_j) \subseteq f(h_j)$*;*

• *M is an applicant-complete matching in the reduced graph* G'.

The same authors gave an efficient algorithm for computing a maximum popular matching (or reporting that none exists).

Theorem 2.9. *[Manlove and Sng, 2006] Given an instance of CHA, we can find a maximum popular matching, or determine that none exists, in* $\mathcal{O}(\sqrt{C}n1 + m)$ *time, where* C *is the sum of the capacities of the houses,* n_1 *is the number of agents and m is the total length of the agents' preference lists.*

$$a_1 : h_1 \quad h_2 \quad h_3$$
$$a_2 : h_1 \quad h_2 \quad h_3$$
$$a_3 : h_1 \quad h_2 \quad h_3$$

Figure 2.9: An istance of HA with no popular matching

2.3.4.1 Popular Matching with Copies

Many instances of the popular matching problem admit no solution. Suppose that we are allowed to add extra copies or duplicate some items and that these duplicates are indistinguishable from the original item. If there were no bounds on the number of copies that we could add for an item, a straightforward solution would be to have as many copies of an item as the number of people that demand the item as their top choice.

For example, in the above Figure 2.7 that does not admit a popular matching, say we have 2 additional copies of h_1 (call them h_1' and h_1''). Thus, we have 3 copies of h_1 in our instance now and it is easy to see that the matching $M = \{(a_1, h_1), (a_2, h_1'), (a_3, h_1'')\}$ is a popular matching for the new instance. Fortunately, popular matchings do not require each applicant to be matched to her top choice item. In this sense, we expect to improve the situation in terms of popularity by having additional copies of some items within the specified bounds.

This problem is called the *Popular Matching with Post Copies*. An example for the popular matching problem is the training program. A training program can be run for a single person, but some training programs may be able to accommodate more

than one person. We wish to fix the capacity of each training program so as to enable the resulting instance to admit a popular matching. An other relevant scenario is to organise bus tours at a conference. People can express their preference over the tours and it is possible to increase the size of a bus at a cost.

However Kavitha and Nasre [Kavitha and Nasre, 2011] gave an example to show that giving each house its maximum possible capacity does not always help to ensure that a popular matching exists. Specifically, they gave a solvable CHA instance where each house has capacity 1, but if we raise the capacity of every house to 2 then the instance is no longer solvable.

Consider the following graph, $G = (A \cup P, E)$ where $A = \{a_1, a_2, a_3, a_4, a_5\}$ and $P = \{f_1, f_2, s_1, s_2, s_3, s_4\}$ and the preference lists of people are described in Figure 2.10(a). In Figure 2.10(b) we have the same set of applicants and preference lists except that every item has 2 copies. That is, $P' = \{f_1, f_1', f_2', f_2, s_1, s_1', s_2, s_2', s_3, s_3', s_4, s_4'\}$ where we have explicitly included an identical copy b' of b, for every $b \in B$. We refer to this instance with duplicates as the instance $H = (A \cup P', E')$. The first column in the figure denotes the set of people and the row adjoining it denotes the preference list of the particular person. For example, a_1 treats f_1 as its rank-1 item, f_2 as its rank-2 item and s_1 as its rank-3 item according to Figure 2.10(a). When there are multiple items in the same cell as in case of Figure 2.10(b), the items are tied at that rank. That is, a_1 treats both f_1 and f_1' as its rank-1 item according to Figure 2.10(b).

	1	2	3
a_1 :	f_1	f_2	s_1
a_2 :	f_1	f_2	s_2
a_3 :	f_1	f_2	s_3
a_4 :	f_1	f_2	s_4
a_5 :	f_2		

(a)

	1	2	3
a_1 :	$f_1 f_1'$	$f_2 f_2'$	$s_1 s_1'$
a_2 :	$f_1 f_1'$	$f_2 f_2'$	$s_2 s_2'$
a_3 :	$f_1 f_1'$	$f_2 f_2'$	$s_3 s_3'$
a_4 :	$f_1 f_1'$	$f_2 f_2'$	$s_4 s_4'$
a_5 :	$f_2 f_2'$		

(b)

Figure 2.10: An instance with copies which does not admit a popular matching.

Consider the subgraph $G_1 = (A \cup B, E_1)$ of G where every person has edges only to her top-choice item. It is easy to see that every maximum matching of G_1 has to match the following vertices: the items $\{f_1, f_2\}$ and the person a_5. Thus, the items f_1 and f_2 are first choice and $s_1, ..., s_4$ are second choice in G_1. Hence s_i is the most preferred second choice vertex in G_1 for person a_i, for $i = 1, ..., 4$. The instance in Figure 2.10 admits a popular matching: for example $M = \{(a_1, f_1), (a_5, f_2), (a_2, s_2), (a_3, s_3), (a_4, s_4)\}$ is a

popular matching.

Let us now consider the subgraph H_1 of H where every person has edges to her rank-1 items. The first choice vertices in $H_1 = (A \cup P', E'_1)$ are f_1 and f'_1 while f_2 and f'_2 are now second choice in H_1. Thus f_2 and f'_2 become the most preferred second choice vertices for $a_1, ..., a_4$ - hence any popular matching has to match each of $a_1, ..., a_4$ to one of $\{f_1, f'_1, f_2, f'_2\}$. Also, a_5 has to be matched to f_2 or f'_2. Since there are 5 people and only 4 items that they can be matched to in any popular matching, there exists no popular matching now. Thus the instance shown in Figure 2.10(b) where each item has 2 copies does not admit a popular matching while the instance in Figure 2.10(a) where each item has a single copy does.

Kavitha and Nasre in [Kavitha and Nasre, 2011] proved the following algorithmic results for the *Popular Matching with Post Copies* (1,2).

Theorem 2.10. *[Kavitha and Nasre, 2011] The* FIXINGCOPIES *(1,2) is NP-complete. The result holds even in the following separate cases:*

- *each applicant $a_i \in A$ has a unique first choice, at most two (tied) second choices, and no house of rank > 2;*
- *each applicant $a_i \in A$ has a strictly-ordered preference list L_i of length at most 3 (excluding l_i), where L_i is derived from a master list L of all houses in H.*

In [Kavitha and Nasre, 2011] the authors show that this problem remains $\mathcal{NP}$-complete when the preference lists are derived from a master list or have length 2 with ties only in the first position. The same authors consider the variant where we only have to maintain an upper bound k on the total number of extra copies of all items, rather than an upper bound on the number of copies of each item.

The problem consists of a graph $G = (A \cup P, E)$ and an integer $k \geq 0$. The task is to decide whether there exist $\Delta_1, \ldots, \Delta_{|P|}$ where each $\Delta_i \geq 1$ and $\sum_i \Delta_i \leq k$, such that by having $\Delta_i + 1$ copies of the i-th item, for all $1 \leq i \leq |P|$, the resulting instance admits a popular matching. This problem can be solved in polynomial time.

2.3.5 Alternative Optimal Criteria

Various definitions of optimality for one-side matching problem have also been studied. The most prevalent concept is *Pareto*-optimality. Informally, a matching is Pareto-optimal if there is no other matching in which at least one applicant is better off, whilst no applicant is worse off.

Other optimality concepts were also studied. A matching is *rank-maximal* [Irving et al., 2006] if it matches the maximum number of applicants to their first choice and then, subject to this, the maximum number to their second choice, and so on. Similarly to Pareto-optimal matchings, rank-maximal matchings always exist and can be found in polynomial time [Irving et al., 2006].

The notion of a *popular* matching [Manlove and Sng, 2006] is also well-known. A matching M is Pareto optimal if there is no other matching M' such that $|P(M, M')| = 0$ and $|P(M', M)| \geq 1$. Hence a popular matching is Pareto optimal. In contrast of the Pareto matching, a popular matching does not always exists.

To see this, consider the example in Figure 2.9. It is clear that a matching cannot be popular unless all applicants are assigned. The unique matching up to symmetry in which all applicants are assigned is $M = \{(a_i, h_i) : 1 \leq i \leq 3\}$, however, $M' = \{(a_2, h_1), (a_3, h_2)\}$ is preferred by two applicants, which is a majority. The potential absence of a popular matching in a given HA instance can be related all the way back to the observation of Condorcet (1785) that, given k voters who each rank n candidates in strict order of preference, there may not exist a "winner," namely, a candidate who beats all others in a pairwise majority vote.

Manlove et al. describe the popular matching in the capacitated house allocation problem in [Manlove et al., 2006] and its variant with weights in [Sng and Manlove, 2010]. McCutchen [McCutchen, 2008] defined two versions of "near-popular" matchings, a least unpopularity factor matching and a least unpopularity margin matching. Also [Kavitha et al., 2011] studied a popular mixed matching, which is a probability distribution over matchings that is popular in a precise sense. A mixed matching is simply a probability distribution over matchings in the input graph. The function Φ that compares two matchings generalized in a natural manner to mixed matchings by taking expectation. A mixed matching P is popular if $\Phi(P, Q) \geq \Phi(Q, P)$ for all mixed matchings Q.

The author of [Mestre, 2006] draw an efficient algorithm for the weighted popular matching problem, where each applicant is assigned a priority or weight, and the definition of popularity takes into account the priorities of the applicants. In this case the algorithm for the no-ties case has $\mathcal{O}(n+m)$ complexity, and for the version that allows ties, the complexity is $\mathcal{O}(min(k\sqrt{n}, n)m)$, where k is the number of distinct weights assigned to applicants. The *optimal* popular matching [Kavitha and Nasre, 2009] where it is possible to return the most fair matching.

Kavitha et al. [Kavitha and Nasre, 2011, Kavitha et al., 2014] solved the problem of

finding a popular matching in a setting where additional copies of items could be introduced at a cost. Some attention from the constraint programming community has focused on encoding many variants of the stable matching problem and its variants [Gent et al., 2001, Manlove, 2005, Prosser, 2014].

[McCutchen, 2008] proposed two quantities to measure the unpopularity of a matching and showed polynomial time algorithms to compute these measures for any fixed matching. He also showed that the problem of computing a matching that minimizes either of these measures is NP-hard. Huang et al. [Huang et al., 2011] gave algorithms to compute matchings with bounded values of these unpopularity measures in certain graphs.

2.4 Kidney Exchange Problem

Damage from kidney disease can cause irreparable loss of organ function and, eventually, kidney failure. Such failure requires either continual dialysis or an organ transplant to sustain life. Kidney transplant is the only effective treatment to cure end stage renal disease, affecting one in every thousand European citizens. Waiting for a compatible deceased donor can involve impractically long waiting times, so living donor transplant is a good alternative: a close relative can give one of its kidneys to a patient. As approximately 40% of living donors [Roth et al., 2004] are incompatible with their specified recipient, several countries have independently developed kidney exchange programs to overcome this issue. However, even when a patient finds a willing living donor, sometimes they are not always compatible, do to blood type compatibility or other specific tests. A kidney exchange arise when a pair (patient, donor) is incompatible with each other and they seek to find a compatible match with other incompatible pairs. In such a program, a patient with an incompatible donor can "swap" its donor with another patient in a similar condition.

Centralised matching programs exists in USA, in the Netherlands, Canada, the United Kingdom, Portugal, Australia, and Israel to organise kidney exchanges between incompatible patient-donor pairs.

South Korea performed the first kidney exchange in 1991 [Park et al., 1999, Kwon et al., 1999], followed by the Switzerland in 2001 [Thiel et al., 2001]; these swaps were organised on a small scale and by hand.

The first kidney exchange in USA was in 2000, involving two blood type incompatible pairs [Wallis et al., 2011]. Altruistic donors can also join the program, i.e. donors that

do not have an associated patient and do not expect anything in return. A report in 2009 counted 13-way exchange involving altruistic donor chains [2].

A 4-way domino kidney transplant involving three cities across Canada was performed soon after the registry was activated in 2009. Since then, multiple domino and paired exchanges have been successfully undertaken and most parts of the country are now participating.

In 2016 a cost action has been established to support an European Network for Collaboration on Kidney Exchange Programmes [3]. The project brought together 28 European countries and policy makers, clinicians, economists, social scientists and optimisation experts in order to establish and channel the various themes that have to be addressed for the implementation of a collaborative KEP for the EU.

2.4.1 Compatibility Graph

The pairs (donor, patient) must be compatible when undergoing organ transplantations [Ashlagi and Roth, 2012, Roth et al., 2004, Roth et al., 2005]. One donor must be blood type compatible with their recipient, the second compatibility is related to the tissue type and the third one is the crossmatching. Below we will explain both of them.

Blood type

In living donation, the following blood types are compatible:

- donors with blood type *A* can donate to patient with blood types A and AB
- donors with blood type *B* can donate to recipients with blood types B and AB
- donors with blood type *AB* can donate to recipients with blood type AB only
- donors with blood type *O* can donate to recipients with blood types A, B, AB and O

Therefore, O is the universal donor: donors with O blood are compatible with any other blood type, and AB is the universal patient: patients with AB blood are compatible with any other blood type.

Tissue type

[2] https://www.washingtontimes.com/news/2009/dec/14/13-way-kidney-transplant-sets-record/
[3] http://www.enckep-cost.eu/

The final test is tissue type, also known as HLA type, between a kidney donor and patient, which is used to identify the presence of preformed antibodies that would damage the kidney, i.e. cause rejection from that specific donor. Tissue matching is a very complex area involving testing the similarity of certain proteins, called antigens, between the donor and patient, which are defined through blood tests. Once compatible blood type is established, a HLA (Human Leukocyte Antigen) crossmatch [4] is completed. For kidney transplantation, we are looking for 6 antigens, called histo-compatibility complex or HLA antigens. By analysing which of these specific antigens both individuals have, we are able to determine the closeness of tissue matching. A six-antigen match, both people have the same set of six antigens, is the best compatibility between a (donor, patient) pair who are not identical twins. This match occurs 25 percent of the time between siblings having the same mother and father and also occurs from time-to-time in the general population.

If the crossmatch is positive, the patient has responded to the donor, antibodies were present and killed the cells, and transplantation should not be carried out. A negative crossmatch means the patient has not responded, and transplantation may proceed. Therefore, the crossmatch is the final and very important test which is done just before the surgery and indicates a go or no go with a transplant operation.

2.4.2 Graph Model

Formally, the KEP is a non-bipartite matching problem on a directed graph, where nodes correspond to patient/donor pairs or simply to willing ("altruistic") donors [Dickerson et al., 2012b, Roth et al., 2005].

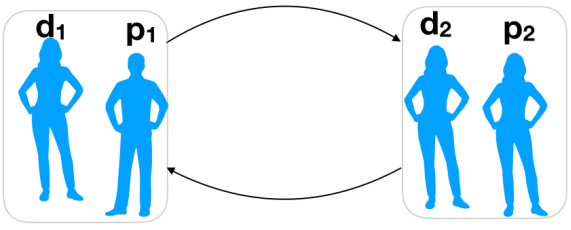


Figure 2.11: A pairwise exchange

An arc from node i to j means that donor i is compatible with patient j, after a blood type compatible test was performed, and a cycle corresponds a viable set of exchanges.

[4] https://www.lifespan.org/centers-services/transplant-center/donor-guide

In Figure 2.11, the pairs are incompatible between them, but donor d_1 can give to patient p_2 and donor d_2 can give to patient p_1, forming a pairwise exchange.

An altruistic donor can initiate a cycle by donating his organ to a patient, whose paired donor donates her organ to another patient, and so on, with no kidney in return. These type of cycles are called chains. In Figure 2.12, we have a 3-way cycle $\{(d_1, p_1), (d_3, p_3), (d_2, p_2)\}$, a 2-way cycle $\{(d_3, p_3), (d_2, p_2)\}$ and a 3-way chain $\{(a), (d_1, p_1), (d_3, p_3), (d_2, p_2)\}$.

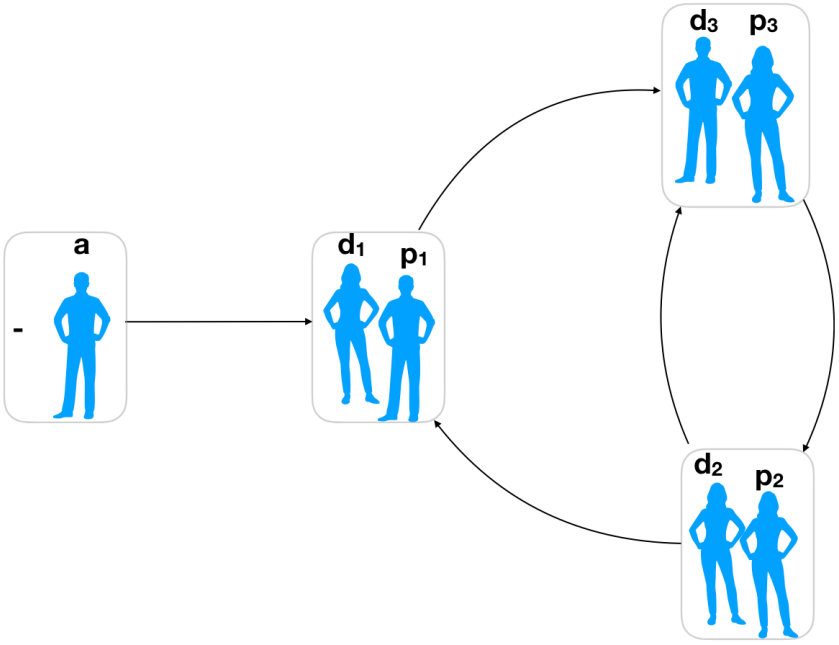


Figure 2.12: A 3-way chain example

An altruistic donor initiates a chain by donating his organ to a patient, whose paired donor donates her organ to another patient, and so on. With altruistic donors, the KEP graph becomes very complex. Sometimes, an exchanges that begins from an altruistic donor can terminates with the last pair giving the donor kidney to the "waitlist".

Consider the chain $(a_1, p_3, p_2, b_2, p_1, p_9, p_8)$ in Figure 2.13 as an example: the incompatible (patient,donor) pair represented by b_2 is called a bridge donor. What it means is that the transplants from altruistic donor a_1 to patient p_3, from donor p_3 to patient p_2, and from donor p_2 to patient b_2 are all performed simultaneously. Donor b_2 will not donate his/her kidney at the same time patient b_2 receives his/hers, but will wait until the transplants involving p_1, p_9, and patient p_8 are ready. These transplants will then be performed simultaneously. The kidney of donor p_8 will then go to the deceased donor waiting-list. Altruistic donors and bridge donors are collectively called non-directed donors (NDDs) in some studies.

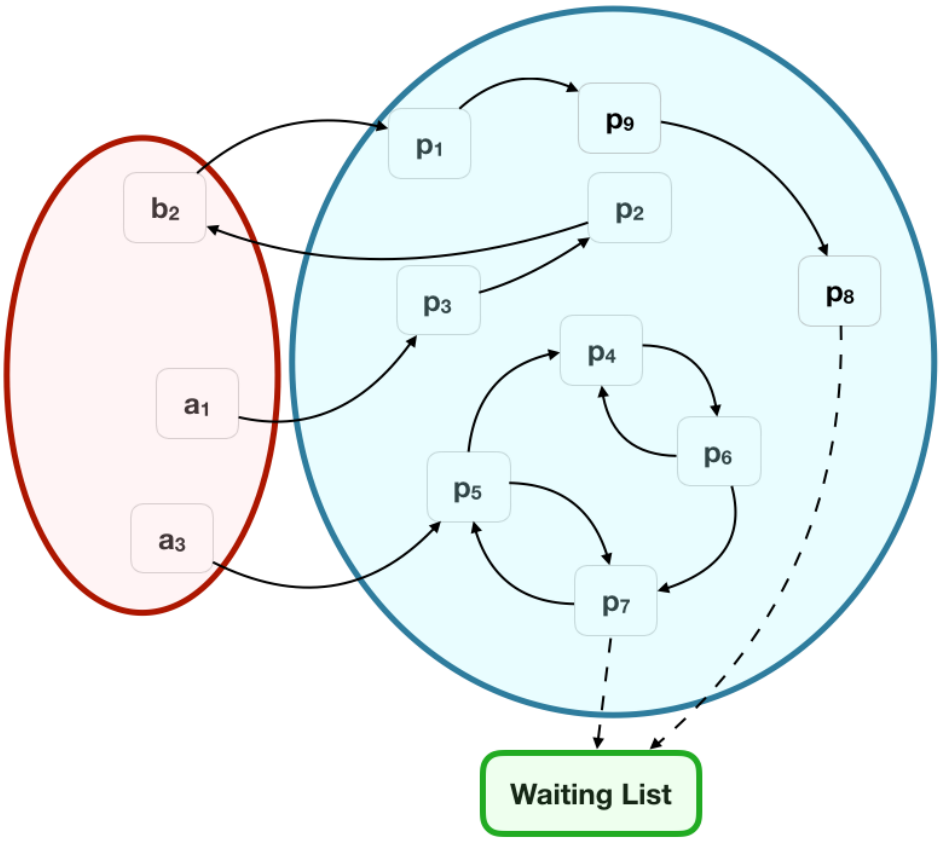


Figure 2.13: An example of KEP pool [Mak-Hau, 2015]

A matching is a collection of disjoint cycles/chains, where no vertex can give out more than one item (e.g., more than one kidney). Finding cycles of these possible donations in large groups of people is a computationally difficult problem. The goal of the programmes are typically to save as many patients as possible, so the optimal solution has maximum size under some constraints and priority criteria.

Even though short cycles/chains are preferred to the long chains/cycles, as the surgeries must take place simultaneous, a 9-way cycle swap has been performed [5].

2.4.3 Classic KEP Formulations

The two fundamental IP models for kidney exchange are the cycle formulation [Roth et al., 2004, Roth et al., 2005, Abraham et al., 2007a, Manlove and O'Malley, 2014], which includes one binary decision variable for each feasible cycle or chain, and the the edge formulation [Dickerson et al., 2016, Constantino et al., 2013,

[5]https://www.sfgate.com/health/article/9-way-kidney-swap-involving-18-surgeries-at-2-6307975.php

Anderson et al., 2015, Mak-Hau, 2015], which includes one decision variable for each compatible pair of agent.

In the cycle formulation, the number of constraints is sublinear in the input size, but the number of variables is exponential. In the basic edge formulation, the number of variables is linear but the number of constraints is exponential. Optimally solving these models has been an ongoing challenge for the past decade.

The cycle formulation

Given a set of vertices $V = (P \cup A)$, where P is the number of incompatible pairs and A the number of altruistic donors, the number of cycles of length at most l is $\mathcal{O}(|P|^l)$, the number of uncapped chains is exponential in $|P|$ if $A \neq 0$, and the number of capped chains of length at most k is $\mathcal{O}(|A||P|^k)$. Let $\mathcal{C}(l,k)$ be the set of cycles of length at most l and chains of length at most k. The KEP can be modeled as an IP formulation [Roth et al., 2005, Dickerson and Sandholm, 2013, Mak-Hau, 2015, Manlove and O'Malley, 2014] with one decision variable per each cycle/chain.

The objective function is to maximise some utility function, i.e. the weight of the cycles, or the number of transplants. Equation (2.34) prevents a pair to be selected in more than one cycle.

$$\max z = \sum_{c \in \mathcal{C}} w_j x_j \tag{2.33}$$

$$\text{s.t.} \sum_{c \in \mathcal{C}: i \in c} x_j \leq 1 \qquad \forall i \in \{P \cup A\} \tag{2.34}$$

$$x_c \in \{0,1\} \qquad \forall c \in \mathcal{C} \tag{2.35}$$

Figure 2.14 illustrates an example of five incompatible pairs. In Figure 2.14, we have four cycles, three 2-way cycles, c_1, c_2, c_3, and one five-way cycle, c_4, therefore there are four binary variables, x_1, x_2, x_3, x_4 with weight ={2, 2, 2, 5}. An optimal solution is to perform cycle c_4 with five transplants.

Unfortunately, the cycle formulation has the number of variables increasing exponentially. Branch and price with column generation approaches [Barnhart et al., 1998] can be used to reduce the model in memory. This approach has been used in [Abraham et al., 2007a, Glorie et al., 2014, Dickerson et al., 2013]. Indeed, the number of cycles can grow exponentially with k.

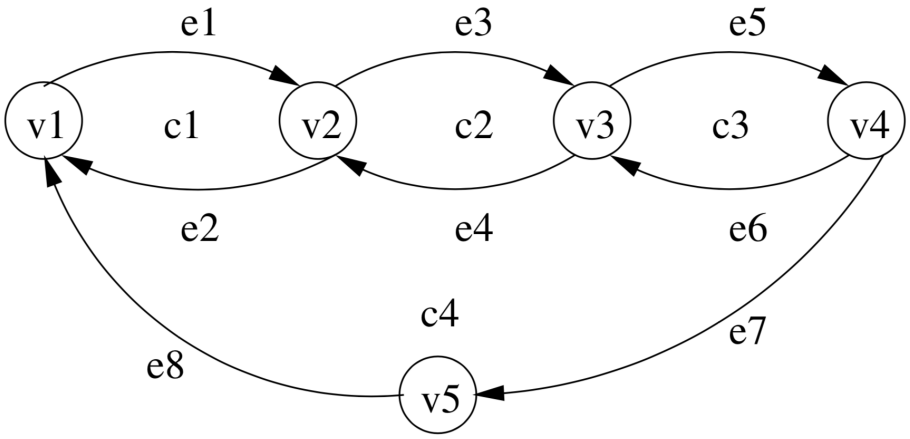


Figure 2.14: Example of KEP graph [Abraham et al., 2007a].

The edge formulation

The model is defined over a directed graph $G = (V, A(V))$, where V is the set of nodes (pairs and altruistic donors) and $A(V)$ is the set of arcs. In the edge formulation [Roth et al., 2007, Mak-Hau, 2015, Anderson et al., 2015], a variable x_{ij} is associated each edge $(i, j) \in A$. The model is defined as:

$$\max \sum_{(i,j)\in A} w_{ij} x_{ij} \tag{2.36}$$

subject to:

$$\sum_{j:(j,i)\in A} x_{ji} = \sum_{j:(i,j)\in A} x_{ij}, \qquad \forall i \in V \tag{2.37}$$

$$\sum_{j:(j,i)\in A} x_{ij} \leq 1, \qquad \forall i \in V \tag{2.38}$$

$$\sum_{1\leq p\leq k} x_{i_p j_{p+1}} \leq k-1, \qquad \forall paths(i_1, i_2, ..i_k, i_{k+1}) \tag{2.39}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A \tag{2.40}$$

The objective function 2.36 maximises the weighted sum of the exchange, i.e. in the case of unitary weights. This corresponds to maximising the total number of transplants. Constraints 2.37 assure that patient i receives a kidney iff donor i donates a kidney. Constraints 2.38 guarantee that a donor can only donate one kidney and constraints 2.39 enforce the cycle-length: to exclude cycles larger than k, we need to make sure that every path of length k arcs does not have more than $k-1$ arcs in a feasible exchange.

In Figure 2.14, there are eight edges, $e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8$, thus eight binary variables. In this case the optimal solution is e_1, e_3, e_5, e_7, e_8 for a total of five transplants.

In general the number of such paths can grow exponentially with k and computationally the size of the problem may become a bottleneck for solving large problem instances [Constantino et al., 2013].

Other formulations

Other compact formulations are presented in [Constantino et al., 2013, Alvelos et al., 2015, Manlove and O'Malley, 2014, Klimentova et al., 2014]. Their extended edge formulation has shown to be effective in finding the optimal solution where the cycle cap is greater than 3. However, each of the compact formulations introduced in their work has a weaker linear program (LP) relaxation than the cycle formulation, even when altruistic donors are not included. Dickerson et al [Dickerson et al., 2016] introduced a variant of the edge formulation, position-indexed edge formulations for kidney exchange, considering even the variant with altruistic donors.

Real-world exchanges suffer to some degree from last minute failures. This can happen for many reasons, including more extensive medical testing before the surgery, a patient or donor becoming too sick to participate, or a patient receiving an organ from the deceased donor waiting list. In [Dickerson et al., 2013] the authors considered the failure aware for the edge formulation setting.

We can model the kidney exchange as a market by constructing an agent for each patient, and an undirected edge between any two agents where the incompatible donor for one patient is compatible with the other patient, and vice versa. An edge $\{p_i, p_j\}$ in a matching corresponds to a pairwise kidney exchange, in which p_j receives a kidney from p_i's donor in exchange for p_i receiving a kidney from p_j's donor. Preference lists can be constructed on the basis of varying degrees of compatibility between patients and potential donors.

Exchanges involving an arbitrary number of nodes (i.e. k-way exchanges) have been considered in the literature. Dealing with more than three/four pairs is hard in practice, since all the operations often take place simultaneously to avoid withdrawal of donors after their pair received the organ.

Most kidney exchange clearing houses try to maximise the (weighted) number of transplants without attention to whether some incompatible pairs can be matched internally by the hospitals that entered them into the database. Thus, it may not be individually rational for a hospital to contribute those pairs it can match internally (see, e.g., [Roth, 2008]). Some weighted matching algorithms put some weight on internal exchanges, but this does not solve the problem.

In some cases sharing the information of all the pairs may lead to find a longer chain of compatibilities. Some work regarding this is presented in [Ashlagi and Roth, 2012].

In [Manlove and O'Malley, 2014], the authors introduces some optimal criteria to the kidney exchanges for the NLDKSS, as determined by the Kidney Advisory Group of NHSBT.

Definition 2.5. *A set of exchanges π is optimal if:*

A. the number of effective 2-cycles in π is maximised;

B. subject to (1), π has maximum size;

C. subject to (1)-(2), the number of 3-cycles in π is minimised;

D. subject to (1)-(3), the number of back-arcs in the 3-cycles in π is maximised;

E. subject to (1)-(4), the overall weight of the cycles in π is maximised.

In barter-exchange markets, agents seek to swap their items with one another in order to improve their social welfare [Abraham et al., 2007a, Ashlagi and Roth, 2012], meaning they try to maximise the exchange in an weighted directed graph. Some proposed models for kidney exchange programs have exponential numbers of constraints or variables, which makes them fairly difficult to solve when the problem size is large. [Constantino et al., 2013] propose two compact formulations for the problem, explain how these formulations can be adapted to address some problem variants, and provide results on the dominance of some models over others.

The author of [Mak-Hau, 2015] reviews existing Integer Programming (IP) and mixed integer linear programming (MILP) approaches to different variations of the KEP and propose a polynomial-sized and an exponential-sized MILP. The author also distinguish between Cardinality Constrained Multi-cycle Problem (CCMcP) and the Cardi-

nality Con- strained Cycles and Chains Problem (CCCCP). The cardinality for cycles is restricted in both CCMcP and CCCCP. The CCMcP can be viewed as an Asymmetric Travelling Salesman Problem that does allow subtours, however these subtours are constrained by cardinality, and it is not necessary to visit all vertices.

Chapter 3

Popular Matchings

We choose to go to the Moon in this decade and do the other things, not because they are easy, but because they are hard. J.F. Kennedy

In this chapter we study several variants of the popular matching problem. We study the problem of matching a set of applicants to an other set, i.e. items/posts, where each applicant has an ordinal preference list, which may contain ties, ranking a subset of the members of the other set. Several notions of optimality are studied in the literature for the case of strictly ordered preference lists [Manlove, 2013, Manlove et al., 2006, Irving et al., 2006]. A matching M is *popular* [Abraham et al., 2007a] if there exists no matching M' where more applicants prefer M' to M.

We propose a constraint programming approach to the popular matching problem. We show that one can use the Global Cardinality Constraint to encode the problem even in cases that involve ties in the ordinal preferences of the applicants. We propose novel algorithmic and complexity results for this variant. Next, we focus on the NP-hard case where additional copies of posts can be added in the preference lists, called *Popular Matching with Copies*. We define new dominance rules for this problem and present several novel graph properties characterising the posts that should be copied with priority.

In the literature there exist different optimal popular matching models, but only for the case where no ties are considered [Kavitha and Nasre, 2011]. We show that the optimal popular matching with ties can be solved in $O(n(m + n\log(n)))$ time, where n is the set of applicants and m is the sum of the length of the preference lists.

In Section 3.1 we gave the necessary background in graph theory, constraint programming. In Section 3.2 we show how the popular matching problem can be modelled using a constraint programming (CP) approach [1] . Using CP is extremely beneficial to tackle the problem with side constraints.

In Section 3.3, we propose an integer linear programming (ILP) model and a flow model for the optimal popular matching, where ties are allowed in the preference list.

Section 3.4 focuses on instances that does not admit a popular matching. In particular, we consider the case where we can add copies of posts in order to find a popular matching [2]. This problem is known to be NP-Hard [Kavitha and Nasre, 2011] and it remains $\mathcal{NP}$-complete even when preferences are derived from the same master preference list [Irving et al., 2008]. Kavitha et al. in [Kavitha and Nasre, 2011] show that this problem remains $\mathcal{NP}$-complete when the preference lists have length 2 and ties are allowed only in first position. However, their results are not applicable to the general case. To the best of our knowledge, our contribution is the first work that considers more than 2 copies for each post. We develop a number of new graph properties, which are used to define new dominance rules for the popular matching with copies problem. These dominance rules have a significant impact for the efficiency of the branching strategy. Last, Section 3.5 experimentally shows the key aspects of a good branching strategy and show the efficiency of our dominance rules for the variant of the popular matching problem where copies are possible.

3.1 Background

Abraham et al. in [Abraham et al., 2007b] showed that the standard popular matching problem is polynomial to solve even if the preference lists contain ties. In particular, they describe a number of efficient algorithms for finding popular matchings.

Several variants of the popular matching problem exist in the literature. For example, let r be the largest preference list size. A popular matching is *fair* [Kavitha and Nasre, 2009] if it matches the least number of applicants to their r-th choice and then, subject to this, the least number to their $(r-1)$-th choice, and so on. A matching is *rank-maximal* [Irving et al., 2006] if it matches the maximum number of applicants to their first choice and then, sub-

[1] Danuta Sorina Chisca, Mohamed Siala, Gilles Simonin, and Barry O'Sullivan. *A CP-based approach for popular matching"*. Published in the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona, USA, pages 4202–4203, 2016. ISBN 978-1-57735-760-5

[2] Danuta Sorina Chisca, Mohamed Siala, Gilles Simonin and Barry O'Sullivan. *New Models for Two Variants of Popular Matching*. Published in: 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), Boston, USA. DOI: 10.1109/ICTAI.2017.00119

ject to this, the maximum number to their second choice, and so on. The authors of [Kavitha and Nasre, 2011, Kavitha et al., 2014] studied the problem of finding a popular matching in a setting where additional copies of posts can be introduced at a cost. A variant called popular matching in the capacitated house allocation problem is described in [Manlove et al., 2006] and its variant with weights is introduced in [Sng and Manlove, 2010].

3.1.1 Graph Theory

We denote by $G = (A \cup P, E)$ an undirected bipartite graph with two disjoint sets of vertices A and P, and a set of edges E. A set $M \subseteq E$ is a matching in G if no two edges in M are adjacent in G. Let $\mathcal{M}$ denote the set of matchings in G.

A maximum cardinality matching (for short a maximum matching) is a matching $M \in \mathcal{M}$ that maximises $|M|$. A matching M is perfect if every vertex of G is incident to an edge of M. $N(v)$ denotes the neighbourhood of any $v \in A$. For any maximum matching M in a graph G, let $\mathcal{P} = v_0, v_1, \ldots, v_l$ be a M-alternating path as defined by Berge [Berge, 1957]. An alternating path $\mathcal{P}$ with respect to M is a simple path in G that contains edges in M and not in M alternately. If the end-vertices of $\mathcal{P}$ are exposed then M is an augmenting path. We will use the notion of an augmenting M-alternating path, and the following well-known theorem from [Berge, 1957].

Theorem 3.1. *Let M be a matching in a graph G. M admits a maximum cardinality if and only if G does not contain an augmenting M-alternating path.*

We will use the Gallai-Edmonds decomposition [Lovász and Plummer, 1986, Abraham et al., 2007b] for bipartite graphs, which involves labelling vertices with respect to a maximum matching M.

3.2 Modelling Popular Matching in CP

Constraint programming (CP) is a rich declarative paradigm to tackle these situations. Using a CP-based approach, the problem is formulated as a set constraints (or subproblems) defined over a set of variables. CP solvers rely on propagating these constraints to reduce the search space as efficiently as possible.

An instance of the *popular matching* problem involves an undirected bipartite graph $G = (A \cup P, E)$, where A is called the set of applicants and P is called the set of posts,

and each applicant a_i ranks all posts in $\{p_j|(a_i, p_j) \in E\}$. We assume in the rest of this chapter that every applicant $a_i \in A$ has in its preference list an extra unique post l_i, called last resort. Every last resort l_i is assumed to be worse than any post in P. Note that the use of l_i as an abbreviation of $l(a_i)$. In this way, we can assume that every applicant is matched, since any unmatched applicant can be assigned to her unique last resort. Therefore, we can focus on matchings that are *applicant-complete*.

For each matched node u in a matching M, we denote by $M(u)$ the node linked to u by an edge in M. An applicant a_i prefers a matching M to a matching M' if a_i is matched in M and unmatched in M', or if a_i is matched in both M and M', but prefers $M(a_i)$ to $M'(a_i)$. A matching M is *more popular* than a matching M' if there are more applicants preferring M to M'. A matching M is *popular* if there exists no matching more popular than M. The popular matching problem is the question of deciding if a popular matching exists, and to find one if it is the case.

We show that this problem can be formulated as a single Global Cardinality Constraint [Régin, 1996]. We begin with a CP model for the classic popular matching problem. Let $\mathcal{I}$ be a popular matching instance and $G = (A \cup P, E)$ the associated undirected bipartite graph. We use one integer variable x_i per applicant $a_i \in A$. The domain of each x_i represents all the potential matches for a_i, thus its domain represents all posts that are neighbours of a_i plus the unique last post l_i. We decide to represent the assignment of a_i by the index of the post (each l_i is indexed by a unique value $|P| + i$). Therefore, the domain is initialised by $\mathcal{D}(x_i) = \{j|p_j \in N(a_i)\} \cup \{|P| + i\}$. Assigning a value j to x_i is interpreted as a_i is matched to post p_j if $j \leq |P|$, and to l_i otherwise.

3.2.1 Strict Preference Lists

We consider here the case where preferences are given without ties. For each applicant a_i, we denote by $f(a_i)$ the first-ranked post in its preference list. We denote by $f(p_j)$ the set of applicants a_i where $f(a_i) = p_j$. A post $p_j \in P$ is called *f-post* if $\exists a_i \in A$ such that $f(a_i) = p_j$. We denote by $s(a_i)$ the best (most preferred) post for a_i that is not an f-post. Such post is called *s-post*. The least preferred post l_i guarantees the existence of $s(a_i)$.

Figure 3.1 shows an example with 6 applicants and 6 posts. The f-posts for this instance are p_2, p_3, p_4 and p_5, and $f(p_2) = \{a_1\}$, $f(p_3) = \{a_6\}$, $f(p_4) = \{a_3\}$, $f(p_5) = \{a_2, a_4, a_5\}$. The bold entries in the preference lists are the f-posts and the underlined entries are the s-posts.

Let $G' = (A \cup P, E')$ be a the reduced graph of G where $E' = \{(a_i, p)|\ p \in f(a_i) \cup s(a_i), a_i \in A\}$, as you can see in Figure 3.1(b).

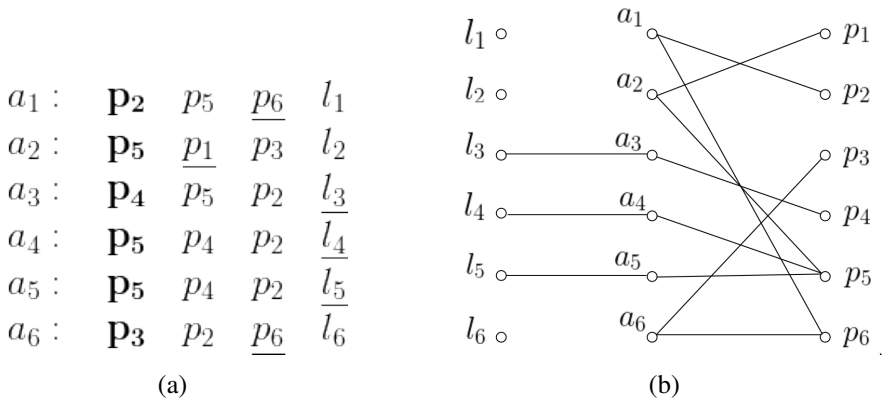


Figure 3.1: An example without ties in the preference lists (a) and the reduced graph G' (b).

Our CP model is based on the following lemma:

Lemma 3.1. *[Abraham et al., 2007b] A matching M is popular iff the following conditions hold:*

(a) Every f-post is matched in M,

(b) For each applicant a_i, $M(a_i) \in \{f(a_i), s(a_i)\}$.

Using Lemma 3.1, we show that one can model the popular matching problem using one global cardinality constraint (GCC) constraint. We define the mapping lb as follows: $lb(j) = 1$ if p_j is an f-post and $lb(j) = 0$ otherwise. This means that the lower bound on the f-posts must be 1, from Lemma 3.1, and the lower bound on the s-posts can be 0, since not all the s-posts can be in the matching. We define the mapping ub as follows:

A. $ub(j) = 0$ if $\forall a_i \in A, f(a_i) \neq p_j$ and $s(a_i) \neq p_j$,

B. $ub(j) = 1$ otherwise.

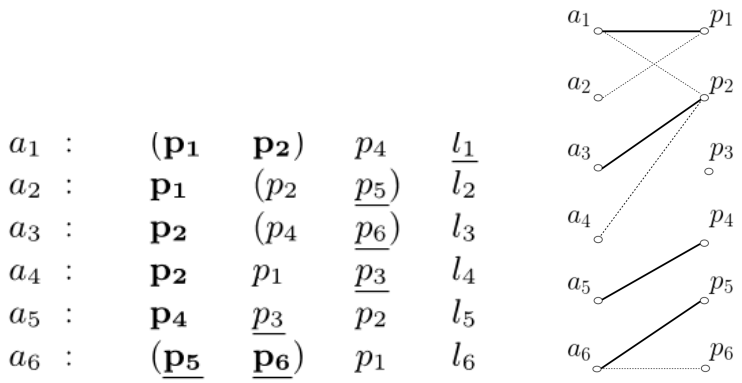
Given a complete assignment of the variables $[x_1, \ldots, x_{|A|}]$, we define a set of edges $M = \{(a_i, p_j)|\mathcal{D}(x_i) = \{j\} \wedge j \leq |P|\}$.

Theorem 3.2. GCC$(lb, ub, [x_1, \ldots, x_{|A|}])$ *is satisfiable iff M is a popular matching.*

Proof. Observe first that if GCC$(lb, ub, [x_1, \ldots, x_{|A|}])$ is satisfiable then all variables will have pairwise different values. Therefore M is a matching. The rest of the proof is a direct application of Lemma 3.1. □

3.2.2 Preference List with Ties

We consider the case where preferences can contain ties (see Figure 3.2(a) [Abraham et al., 2007b]). The notion of f-post and s-post are defined in the case with ties similar to the preferences list without ties. The definition of $f(a_i)$ becomes the set



$$\mathcal{E} = \{a_1, a_2, a_3, a_4, p_3, p_5, p_6, l_1, l_2, l_3, l_4, l_5, l_6\}$$
$$\mathcal{O} = \{a_6, p_1, p_2\}$$
$$\mathcal{U} = \{a_5, p_4\}$$

(c)

Figure 3.2: An example with ties in the preference lists and the graph G_1 with a maximum matching in bold and the $\mathcal{E}, \mathcal{O}, \mathcal{U}$ labelling sets associated.

of top choices for applicant a_i. However the definition of $s(a_i)$ is no longer the same and in this case it may contain a number of surplus f-posts.

Let M be a popular matching of some instance graph $G = (A \cup P, E)$. We define the *first-rank* graph of G as $G_1 = (A \cup P, E_1)$, where E_1 is the set of all rank-one edges.

Let M_1 be a maximum matching in G_1. We recall the Gallai-Edmonds decomposition [Lovász and Plummer, 1986] in Lemma 3.2 using the following notation. Using M_1 we can partition $A \cup P$ into three disjoint sets: $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$. These sets are defined as following: $\mathcal{E}$ (respectively $\mathcal{O}$) is the set of nodes having an even (respectively odd) alternating path with respect to M_1 from an unmatched node; and $\mathcal{U}$ is the set of nodes that are not in $\mathcal{E} \cup \mathcal{O}$.

Lemma 3.2 (Gallai-Edmonds decomposition)**.** *Let $\mathcal{E}$, $\mathcal{O}$ and $\mathcal{U}$ be the vertices sets defined by G_1 and M_1 above. Then:*

A. $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ are a partition of $A \cup P$, and any maximum matching in G_1 leads to exactly the same sets $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$.

B. *Every node in $\mathcal{O}$ (resp. $\mathcal{U}$) is matched to a node in $\mathcal{E}$ (resp. $\mathcal{U}$), and $|M| = |\mathcal{O}|+|\mathcal{U}|/2$.*

C. *No maximum matching of G_1 contains an edge between two nodes in $\mathcal{O}$, a node in $\mathcal{O}$ and a node in $\mathcal{U}$, or between a node in $\mathcal{E}$ and a node in $\mathcal{U}$.*

In [Abraham et al., 2007b], $s(a_i)$ is defined as the top choice(s) for a_i that is in $\mathcal{E}$ (see Figures 3.2(a) where bold entries represent the f-posts and underlined entries represent the s-posts). In this example the nodes label are: $\mathcal{E} = \{a_1, a_2, a_3, a_4, p_3, p_5, p_6, l_1, l_2, l_3, l_4, l_5, l_6\}$, $\mathcal{O} = \{a_6, p_1, p_2\}$, and $\mathcal{U} = \{a_5, p_4\}$.

We use the following theorem from [Abraham et al., 2007b] to model the popular matching with ties.

Theorem 3.3. *[Abraham et al., 2007b] A matching M is popular iff:*

A. *$M \cap E_1$ is a maximum matching of G_1,*

B. *For each applicant a_i, $M(a_i) \in f(a_i) \cup s(a_i)$.*

Using the reduced graph model, we have the following lemma:

Lemma 3.3. *From an instance $G^0 = (\mathcal{A} \cup \mathcal{P}^0, E^0)$ that does not admit a popular matching, adding a copy of a post p_i, that has already a number of copies equal to the degree of the vertex p_i in G^0, leads to an instance without popular matching.*

Proof. In the graph G^0, the number of applicants connected to any post p_i defines the maximum number of copies we should allow for that post. Indeed in other case there will not be enough applicants to match with the new copy. □

From this Lemma, we apply this new pre-pruning rule on our model:

- $\forall i, \mathcal{D}(x_i) \leftarrow \mathcal{D}(x_i) \setminus \{d(p_i)+1, \ldots, c_i\}$, where $d(p_i)$ is the degree of the item p_i in the subgraph G^0.

We show how to model the popular matching problem with ties using one GCC constraint. The domain of every variable x_i is pruned to contain only values corresponding to posts in $f(a_i) \cup s(a_i)$.

Next, by using the properties of $\mathcal{E}, \mathcal{O}, \mathcal{U}$ from Lemma 3.2 we can apply several preprocessing steps to our model (note that a last resort can be only a member of $\mathcal{E}$):

A. By definition, every vertex in $\mathcal{U}$ is matched to another vertex in $\mathcal{U}$. Let $\Psi = \{j|p_j \in \mathcal{U}\}$ and $\Omega = \{i|a_i \in \mathcal{U}\}$ be the two indexes of vertices in $\mathcal{U}$. This property leads to the following preprocessing steps:

(a) $\forall i \in \Omega, \mathcal{D}(x_i) \leftarrow \mathcal{D}(x_i) \cap \Psi$,

(b) $\forall j \in \Psi, \forall i \in [1, |A|] \setminus \Omega, \mathcal{D}(x_i) \leftarrow \mathcal{D}(x_i) \setminus \{j\}$.

B. By definition, every vertex in $\mathcal{O}$ is matched to a vertex in $\mathcal{E}$. Let $\Theta = \{j | p_j \in \mathcal{E} \vee l_{j-|P|} \in \mathcal{E}\}$ and $\Upsilon = \{i | a_i \in \mathcal{E}\}$ be the two indexes of vertices in $\mathcal{E}$.

Let $\Phi = \{k | a_k \in \mathcal{O}\}$ and $\Lambda = \{l | p_l \in \mathcal{O}\}$ be the two indexes of vertices in $\mathcal{O}$. This property leads to the following preprocessing steps:

(a) $\forall k \in \Phi, \mathcal{D}(x_k) \leftarrow \mathcal{D}(x_k) \cap \Theta$,

(b) $\forall l \in \Lambda, \forall i \in [1, |A|] \setminus \Upsilon, \mathcal{D}(x_i) \leftarrow \mathcal{D}(x_i) \setminus \{l\}$.

The mappings lb and ub are defined as follows:

A. $lb(j) = 1, \forall j$ such that $p_j \in \mathcal{O} \cup \mathcal{U}$, otherwise $lb(j) = 0$;

B. $ub(j) = 1$ for all j such that p_j is an f-post, p_j is an s-post, or $j > |P|$ and $l_{j-|P|}$ is an s-post. Otherwise $ub(j) = 0$.

For every complete assignment of $[x_1, \ldots, x_{|A|}]$, we define a unique set of edges $M = \{(a_i, p_j) | \mathcal{D}(x_i) = \{j\} \wedge j \leq |P|\}\}$.

Theorem 3.4. GCC$(lb, ub, [x_1, \ldots, x_{|A|}])$ *is satisfiable iff* M *is a popular matching with ties.*

Proof. $\Rightarrow$ First, if GCC is satisfiable, then $\forall a_i \in A$, $M(a_i) \in f(a_i) \cup s(a_i)$. Second, the preprocessing steps enforce the properties (b) and (c) from Lemma 3.2. Therefore $M \cap E_1$ is a maximum matching of G_1.

$\Leftarrow$ If M is popular, then by construction, the correspondent GCC constraint is satisfiable. □

3.3 Optimal Popular Matching

Several notions of optimality are studied in the literature for the case of preferences without ties [Kavitha and Nasre, 2009, McDermid and Irving, 2009]. The most common way is to associate each edge to a weight and then find a minimum/maximum weight popular matching.

The authors of [McDermid and Irving, 2009] improve the algorithms of Kavitha [Kavitha and Nasre, 2009] for finding a mincost popular matching in $O(n + m)$ time

and rank-maximal/fair popular matchings in $O(n \log n + m)$ time, where $n = |A|$ and m is the sum of preference lists.

We consider the more general case where preferences can contain ties. We assume that the weight function is monotonically increasing with respect to the rank. To the best of our knowledge, the complexity of finding an optimal popular matching where ties are allowed in the preference lists, is unknown. However, recently we found a technical report in ArXiv [Matsui and Hamaguchi, 2016] showing the result with the same worst-case time complexity. Our approach is, however, much simpler.

The solution of the popular matching problem is a matching between the set of applicants A and the set of posts Δ^P corresponding to the values in $\Delta = \bigcup_{i=1}^{n} \mathcal{D}(x_i)$.

Let n_1 be the number of applicants, and $n_2 \leq n_1$ be the number of posts in Δ^P. We denote by Δ_β the set of posts in $\mathcal{O}_p \cup \mathcal{U}_p$, where β is the size of this set. Let $G' = (A \cup \Delta^P, E')$ be the bipartite graph modelling our problem, where the weight on each edge $(a_i, p_j) \in E'$ represents the rank of p_j in the preference lists of a_i.

We use the well-known Minimum Weight Perfect Matching in Bipartite Graph problem. This problem is equivalent to the classic assignment problem with cost, which has the following ILP model to solve it:

$$min \sum_{(a_i,p_j)\in E'} w_i^j x_i^j \tag{3.1}$$

$$\sum_j x_i^j = 1 \quad \forall i \tag{3.2}$$

$$\sum_i x_i^j = 1 \quad \forall j \tag{3.3}$$

$$x_i^j = \begin{cases} 1 & \text{if the edge } (a_i, p_j) \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

We transform the graph G' ito an instance of the Minimum Weight Perfect Matching in a Bipartite Graph problem. In order to obtain a perfect matching, we need to add $K = n_2 - n_1$ dummy-applicant vertices in A, each of which is linked to all posts in $\Delta^P \setminus \Delta_\beta$ with a weight equal to a large constant C, e.g. $C = |P|^2$. Note that the weight on the edges between $a_i \in A$ and $p_j \in \Delta_\beta$ is equal to 1 (first rank). This linear transformation is illustrated in Figure 3.3.

If we compute a Minimum Weight Perfect Matching with the ILP and find a solution, we will obtain a weight equal to $KC + \epsilon$, where ϵ represents the weights sum from the n_1 normal applicants. Since KC is constant, ϵ is the minimum weight that we

are looking for. Thus, the matching obtained, without the dummy-applicant vertices, corresponds to a solution of optimal popular matching problem.

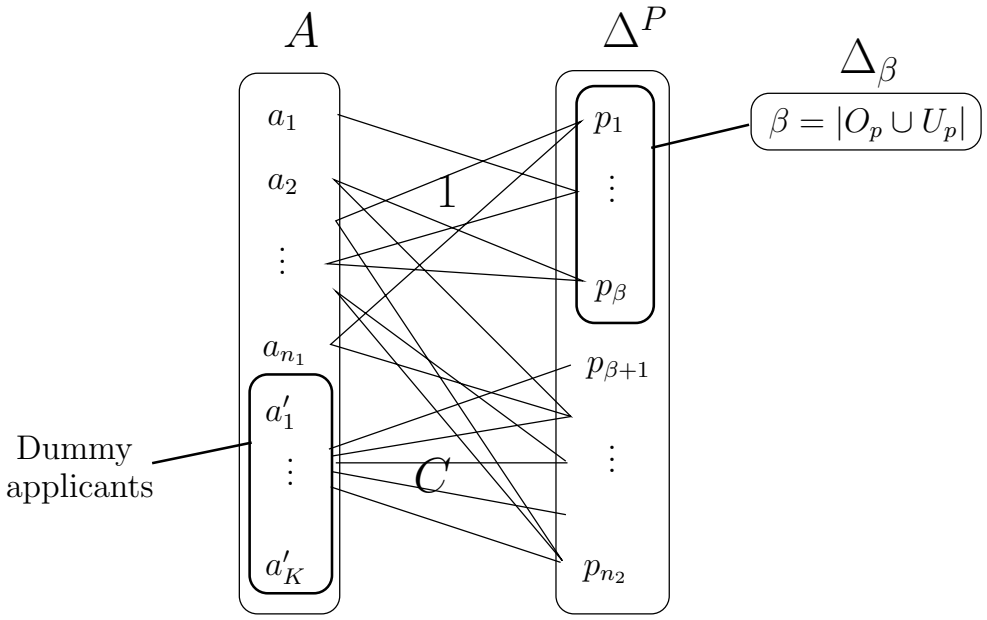


Figure 3.3: A perfect matching instance.

Kuhn and Yaw [Kuhn and Yaw, 1955] gave the first polynomial algorithm, called the Hungarian method, to solve the Minimum Weight Perfect Matching in Bipartite Graph problem. The algorithm had a running time of $O(n^3)$. Then, in [Edmonds and Karp, 1972] the authors gave a reduction of this problem to the Flow Circulation problem. As we did with our linear transformation to obtain a perfect matching instance, we can create a flow circulation instance in a special network, and from this new model, we can use better existing complexity results.

Let $G = (V, E)$ be a network, the Flow Circulation problem is define as follow:

$$lb(v,w) \quad \text{lower bound on flow on the edge } (v,w)$$
$$ub(v,w) \quad \text{upper bound on flow on the edge } (v,w)$$
$$c(v,w) \quad \text{cost of one flow unit on the edge } (v,w)$$
$$lb(v,w) \leq f(v,w) \leq ub(v,w) \tag{3.4}$$
$$min \sum_{(v,w)\in E} c(v,w).f(v,w) \tag{3.5}$$

We also need to keep track of the flow of the individual commodities:

$$f_i(v,w) \quad \text{the flow of commodity } i \text{ from } v \text{ to } w. \tag{3.6}$$

$$f(v,w) = \sum_i f_i(v,w) \quad \text{the total flow.} \tag{3.7}$$

The conservation constraint must be upheld individually for the commodities:

$$\sum_{w \in V} f_i(v,w) = 0. \tag{3.8}$$

3.3.1 Linear Transformation

In order to force the assignment of the β posts in Δ_β, we fix by $[1,1]$ the lower and upper bounds of the arcs going out from these posts. We also force the flow passing by the other posts to not exceed $n_1 - \beta$, otherwise all the flow could pass by these arcs and not by the posts in Δ_β. An illustration of the construction is presented in Figure 3.4. In [Fredman and Tarjan, 1987], the authors proposed an algorithm to solve the flow problem in $O(n(m + n\log n))$ time. Therefore, an optimal popular matching with ties can be computed in $O(n(m + n\log n))$ time.

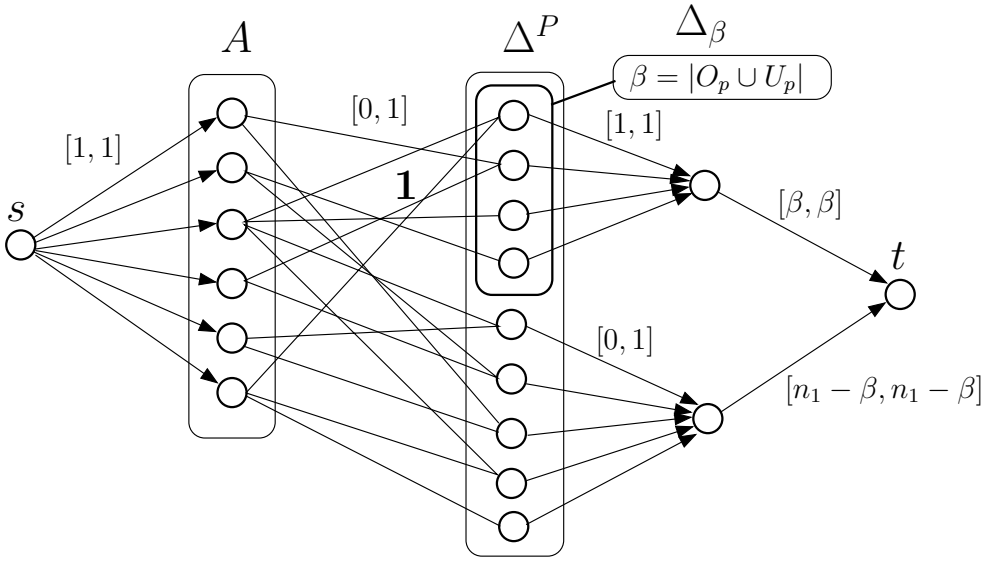


Figure 3.4: Illustration of the Flow Circulation problem.

3.4 Popular Matchings with Post Copies

Many instances of the popular matching problem admit no solution. In the literature, authors have studied such situations where extra copies of posts are allowed in order to find a solution [Kavitha and Nasre, 2011]. The 1-or-2 copies problem can be seen as a generalisation of the popular matching problem.

The problem is related to many real-world applications such as DVD rental stores, training programs, and bus tours. According to client demand, additional copies of an item can be purchased in order to satisfy their preferences. A training program can be run for a single person, but some training programs may be able to accommodate more than one person. To organise bus tours for a given conference, people express their preference over the tours and it is possible to increase the size of a bus at a cost, or with additional buses.

This problem is called the FIXINGCOPIES problem:

Instance: Given a graph $G = (A \cup P, E)$ and a list $\langle c_1, \ldots, c_{|P|} \rangle$ of upper bounds on the number of copies possible for each post.

Question: Does there exist an $\langle \chi_1, \ldots, \chi_{|P|} \rangle$ such that for each $i \in \{1, \ldots, |P|\}$, having χ_i copies of the i-th post, where $1 \leq \chi_i \leq c_i$, enables the resulting graph to admit a popular matching?

The FIXINGCOPIES problem is known to be $\mathcal{NP}$-complete [Kavitha and Nasre, 2011], and it remains $\mathcal{NP}$-complete even when preferences are derived from the same master preference list [Irving et al., 2008].

3.4.1 Properties of the FIXINGCOPIES Problem

It is assumed that the FIXINGCOPIES problem does not admit a popular matching for the basic instance $\mathcal{I}^0$ where the number of copies of each post is equal to 1.

The bipartite graph associated with $\mathcal{I}^0$ is denoted by $G^0 = (A \cup P^0, E^0)$. Thus, our problem can be divided into two parts:

(a) The decision of the number of copies for each post that defines an instance $\mathcal{I}^i$ of the popular matching problem with ties (containing a graph $G^i = (A \cup P^i, E^i)$);

(b) Solving the popular matching problem given by $\mathcal{I}^i$.

We propose to study the consequences of creating post copies for the instances that do not admit popular matching according to the labels.

First we will show that the properties from [Kavitha and Nasre, 2011] for the 1-or-2 copies problem can not be applied to our case. We will describe several new rules for the general copies problem.

In [Kavitha and Nasre, 2011], the authors define two notions: the critical items that are the items in $\mathcal{U} \cup \mathcal{O}$, and the non-critical that are the items in $\mathcal{E}$. Then they gave details

about which items from the $\mathcal{E}, \mathcal{O}, \mathcal{U}$ labels can or must not be copied. Assume that $G = (A \cup P, E)$ is a graph and K is a set of items that can be duplicated. Assume also that G does not admit a popular matching, and there exist a way of duplicating items in K such that the augmented instance $\tilde{G}$ admits a popular matching, and then the following lemma:

Lemma 3.4. *[Kavitha and Nasre, 2011] There exists an augmented instance $\tilde{G}$ such that every item that is critical in G_1 is also critical in $\tilde{G}_1$ and $\tilde{G}$ admits a popular matching.*

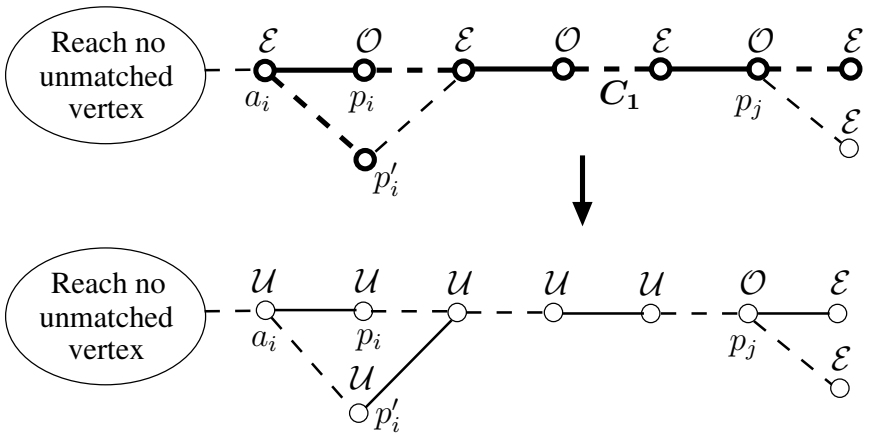
This lemma could be useful for our pruning rules for the general case, but unfortunately this lemma is not true for the general copies problem.

Proof. Indeed, we can use the existing proof and try to do the same, we will see that cannot work in this case. Suppose that we have a popular matching due to a copy of an item b_i such that b_i is critical in G_1 but becomes non critical (as its copy) in $\tilde{G}_1$ after the copy ($\tilde{G}$ is the instance with the copy). As in the proof of the lemma, the maximum matching in $\tilde{G}_1$ is the same without the copy p_i'. Now for the second condition to have the popularity, is that we obtain it with this copy, it is because an applicant is assigned to p_i'. Due to the non-criticality of p_i', it is an assignment out of $\tilde{G}_1$, so out of $\tilde{E}_1$. Thus there exists an applicant a not assigned (and non-critical) such that b_i is the best choice from $\mathcal{E}$. If we remove p_i', the graph obtained makes p_i critical but a is still non-critical. a has no non-critical items in E_1 otherwise the maximum matching will be not maximum. In fact the best choice in $\mathcal{E}$ will be after p_i (now critical), it can be a tie or from a lower rank, but this item can be assigned to another applicant already. Thus the solution might not be popular and the lemma is not applicable for the general problem. □

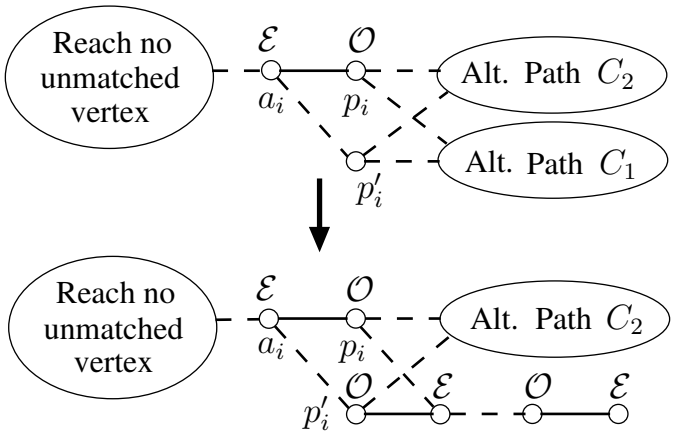
Now, we introduce several properties of the new instances obtained by copying a post according to the $\mathcal{E}, \mathcal{O}, \mathcal{U}$ labelling. In the following, a post will be said to be in $\mathcal{E}$, $\mathcal{O}$ or $\mathcal{U}$ if it is labelled by the corresponding set.

Lemma 3.5. *In any instance obtained from the copy of a post p_i in $\mathcal{O}$, the post and its copies will remain in $\mathcal{O}$ or be in $\mathcal{U}$. From this copy, any other post in $\mathcal{O}$ will be also in $\mathcal{U}$ or will remain in $\mathcal{O}$, and all posts in $\mathcal{E}$ or $\mathcal{U}$ will remain the same.*

Proof. Let M be the maximum matching used to obtain the sets $\mathcal{E}$, $\mathcal{O}$ and $\mathcal{U}$. By definition, if p_i is in $\mathcal{O}$ then it is linked to an M-alternating path and matched to an applicant a_i in $\mathcal{E}$ (see Figure 3.5). We recall that all posts in these paths are in $\mathcal{O}$ by definition. Therefore there are two cases: 1) there are only non-disjoint M-alternating



(a) Non-disjoint alternating paths



(b) Disjoint alternating paths

Figure 3.5: Illustrations of the copy of a post in $\mathcal{O}$ in G_1.

paths linked to p_i (see Figure 3.5(a)), and 2) at least two disjoint M-alternating paths are connected to p_i (see Figure 3.5(b)).

In both cases, the copy p'_i is also connected to a_i and to at least one M-alternating path. There exists one augmenting M-alternating path called C_1 from p'_i. By changing the membership in M of all edges of C_1, we can increase the matching. After this modification, there is no other augmenting M-alternating path and the matching is maximum by Berge's Theorem [Berge, 1957]. Since all posts of C_1 are in $\mathcal{O}$, no post in $\mathcal{E}$ or $\mathcal{U}$ changes from this copy. From this new matching M, we can recompute the labels.

In the first case, let p_j be the closest vertex to p_i on C_1 which creates a joint with other M-alternating paths. To simplify the proof we split C_1 in two parts: the one at the left of p_j (containing p_i) and the right one. The right part of C_1 remains unchanged, but on the left part of C_1, every vertex changes into $\mathcal{U}$ (see Figure 3.5(a)). Thus from this

copy, all posts in $\mathcal{O}$ will remain in $\mathcal{O}$ or will change into $\mathcal{U}$.

In the second case, there exists at least one distinct M-alternating path C_2 connected to p_i and p'_i. The concatenation of C_2 and the modified C_1 creates a unique M-alternating path. Therefore, the labels from the modified C_1 remain the same. Thus in both cases, after the copy of a post in $\mathcal{O}$, all posts in $\mathcal{O}$ will remain the same or will be in $\mathcal{U}$, and all posts in $\mathcal{E}$ or $\mathcal{U}$ will remain the same. □

In a similar way, we have the following proprieties for the applicants' labels.

Lemma 3.6. *In any new instance obtained from the copy of a post p_i in $\mathcal{O}$, any applicant in $\mathcal{E}$ will remain in $\mathcal{E}$ or will be in $\mathcal{U}$. And all applicants in $\mathcal{U}$ or $\mathcal{O}$ will remain the same.*

Proof. This proof is complementary of the previous one. First, one can remark that from a_i there can exist M-alternating paths starting by O (at the left of a_i on the Figure 3.5). After the modification of C_1, if a_i is in $\mathcal{U}$ then these paths may change also in $\mathcal{U}$. We remark that only applicants in $\mathcal{E}$ can change like this. The other applicants with a different label will remain the same. □

Corollary 3.1. *In any instance obtained from a copy of a post in $\mathcal{O}$, the size of the maximum matching will be increased by* 1.

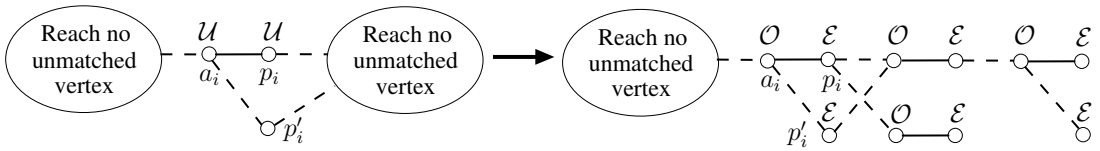


Figure 3.6: Illustration of the copy of a post in $\mathcal{U}$ in G_1.

Lemma 3.7. *In any instance obtained from the copy of a post p_i in $\mathcal{U}$, the post and its copies will be in $\mathcal{E}$. From this copy, any other post in $\mathcal{U}$ will be in $\mathcal{E}$ or will remain in $\mathcal{U}$. And all posts in $\mathcal{E}$ or $\mathcal{O}$ will remain the same.*

Proof. Let M be the maximum matching used to obtain the sets $\mathcal{E}$, $\mathcal{O}$ and $\mathcal{U}$. By definition, if p_i is in $\mathcal{U}$ then it cannot be linked to an M-alternating path, and it is matched to an applicant a_i who is also in $\mathcal{U}$ (see Figure 3.6). Since the copy p'_i has the same properties as p_i, no augmenting M-alternating path is created from it, and p'_i remains unmatched and in $\mathcal{E}$. Thus the maximum matching remains the same, so all vertices that are not a member of any M-alternating path from p'_i will keep their label.

All the vertices in $\mathcal{U}$ that are a member of an M-alternating path from this p'_i will be in $\mathcal{O}$ or $\mathcal{E}$. It is obvious that the applicant vertices will be in $\mathcal{O}$ and the post ones in $\mathcal{E}$.

All vertices not in $\mathcal{U}$ that are also a member of this path will remain the same. Thus after the copy of a post in $\mathcal{U}$, all posts in $\mathcal{U}$ will remain the same or will be in $\mathcal{E}$, and all posts in $\mathcal{E}$ or $\mathcal{O}$ will remain the same. □

Lemma 3.8. *In any new instance obtained from the copy of a post p_i in $\mathcal{U}$, any applicant in $\mathcal{U}$ will remain in $\mathcal{U}$ or will be in $\mathcal{O}$. And all the applicants in $\mathcal{E}$ or $\mathcal{O}$ will remain the same.*

Proof. This proof is complementary to the previous one. First, after the changing of labels on the new M-alternating paths (see Figure 3.6), the only applicants changing will be the one associated with this path, and more precisely the $\mathcal{U}$ ones. Note that from a_i, the neighbourhood in $\mathcal{U}$ will stay the same. The only vertices changing label are on these new M-alternating paths. Thus, the other applicants with a different label will remain the same. □

Corollary 3.2. *In any instance obtained from the copy of a post in $\mathcal{U}$, the maximum matching will remain the same.*

Corollary 3.3. *From any instance $\mathcal{I}$ with a popular matching, every instance $\mathcal{I}^*$ obtained from $\mathcal{I}$ by copying just posts in $\mathcal{E}$, remains popular*

Proof. From Lemma 3.9, adding more copies from $\mathcal{E}$ will not change the labels. In best case we will obtain more s-posts. Thus $\mathcal{I}$ is a sub-solution of any $\mathcal{I}^*$. □

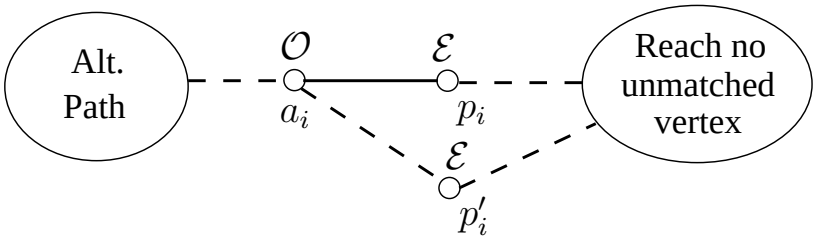


Figure 3.7: Illustration of the copy of a post in $\mathcal{E}$ in G_1.

Lemma 3.9. *In any instance obtained from the copy of a post p_i in $\mathcal{E}$, the maximum matching and the label of any vertex will remain the same.*

Proof. Let M be the maximum matching used to obtain the sets $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$. First, if p_i is unmatched then any copy of this post will be in $\mathcal{E}$. Second, if p_i is matched, then by definition, it cannot be linked to an M-alternating path, and it is matched to an applicant a_i who is in $\mathcal{O}$ (see Figure 3.7). Obviously, p'_i will be only connected to an applicant vertex that is in $\mathcal{O}$. Thus, we will have only new M-alternating paths from p'_i but none of them will be an augmenting path. Thus after the copy of a post in $\mathcal{E}$, all vertices will remain the same as the cardinality of the maximum matching. □

3.4.2 Two Automatons

From the lemmas established above, we can deduce important results about the behaviour of the label of any post after a copy.

Theorem 3.5. *From any copy of a post in $\mathcal{E}$, $\mathcal{O}$ or $\mathcal{U}$, every post uses the following automaton to determine its label:*

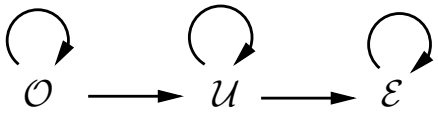


Figure 3.8: Illustration of the posts automaton.

Proof. By merging the Lemmas 3.5, 3.7 and 3.9, one can see that after any copy of a post with any label we always obtain the following rules:

(a) The label of any post in $\mathcal{O}$ will stay the same or will change into $\mathcal{U}$.

(b) The label of any post in $\mathcal{U}$ will stay the same or will change into $\mathcal{E}$.

(c) The label of any post in $\mathcal{E}$ will never change.

The automaton obtained is immediate from these rules. □

Theorem 3.6. *From any copy of a post in $\mathcal{E}$, $\mathcal{O}$ or $\mathcal{U}$, every applicant follows the following automaton to determine its label:*

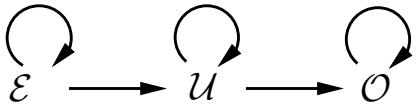


Figure 3.9: Illustration of the applicants automaton.

Proof. By merging the Lemmas 3.6, 3.8 and 3.9, one can see that after any copy of an applicant with any label we always obtain the following rules:

(a) The label of any applicant in $\mathcal{E}$ will remain the same or will change into $\mathcal{U}$.

(b) The label of any applicant in $\mathcal{U}$ remain stay the same or will change into $\mathcal{O}$.

(c) The label of any applicant in $\mathcal{O}$ will never change.

The automaton obtained is immediate from these rules. □

Corollary 3.4. *Any applicant matched and labelled by $\mathcal{O}$ or $\mathcal{U}$ will always remain matched in the children instances.*

3.4.3 Properties of G'

Now we can give properties for the reduced graph G'. In the reduced graph, first, we search for a maximum matching, and secondly, we define the $\mathcal{E}, \mathcal{O}, \mathcal{U}$ labels. On the reduced graph, we still need to have a maximum matching M in G_1 and an assignment-complete in G'. If the instance is not popular, from the maximum matching M and the labels, we can just study the set $\mathcal{NA}$ of non-assigned applicants. If we want to increase the assignment, we need to find an augmenting path with new copies. Thus the only alternating path studied are the ones starting from the unmatched applicants, i.e. $a \in \mathcal{NA}$. Therefore, we have the following lemmas:

Lemma 3.10. *We will never do a copy of posts in $\mathcal{E}$ or $\mathcal{U}$ in order to obtain a popular matching.*

Proof. On these alternating paths $\mathcal{NA}$, the posts are only in $\mathcal{O}$.

Lemma 3.11. *A copy of a post in $\mathcal{O}$ from $\mathcal{NA}$ can not assign two more applicants.*

Proof. Let a_1 and a_2 be the two applicants assigned with the copy. If we are changing the belonging from the augmenting alternating path $\{a_1, \ldots, p'_i\}$, from a_2 we have only an alternating path where $p'_i \in \mathcal{O}$. This leads having $a_1 \in \mathcal{E}$ and p_i still in $\mathcal{O}$. So there is no other unmatched vertex to have an augmenting alternating path. □

From this lemma we obtain a Lower Bound:

Theorem 3.7 (Lower Bound)**.** *If there are k non-assigned applicants in G', then we need at least k copies to reach a popular matching. This is the lower bound for the number of copies required to make the instance popular.*

An instance can be popular only if the maximum matching in G' is applicant-complete, thus only if we do useful copies to find augmenting alternating paths from the non-assigned applicants. These copies can only be done from these alternating paths $\mathcal{NA}$, and only from $\mathcal{O}$ posts.

From this remark we can define the maximum number of copies to the problem (Upper Bound):

Theorem 3.8 (Upper Bound)**.** *If there are k non-assigned applicants in G', and from them there are several alternating paths $\mathcal{P}$, with exactly p posts in $\mathcal{O}$, in order to find a popular matching, we can create a limited number of copies equal to:*

$$UB = \sum_{p_i \in \mathcal{P}} [d(p_i) - 1] = \sum_{p_i \in \mathcal{P}} d(p_i) - p \tag{3.9}$$

where $d(p_i) - 1$ is the degree of the node p_i minus the original copy already assigned to an applicant.

3.4.4 Dominance Rules

We introduce new dominance rules. For any instance $\mathcal{I}$, we denote by $\mathcal{F}^{\mathcal{I}}$ the set of posts that are not f-posts or s-posts.

Lemma 3.12. *Let $\mathcal{I}$ and $\mathcal{I}^*$ be two instances where $\mathcal{I}^*$ is built from $\mathcal{I}$ by copying some posts. Any post in $\mathcal{F}^{\mathcal{I}}$ is in $\mathcal{F}^{\mathcal{I}^*}$.*

Proof. Let p_i be such a post. Clearly p_i is in $\mathcal{E}$. Now since $p_i \in \mathcal{F}^{\mathcal{I}}$, then for all the applicants, their best choice in $\mathcal{E}$ is never p_i but other posts in $\mathcal{E}$. From Theorem 3.5 any post in $\mathcal{E}$ will always remain in $\mathcal{E}$ after any kind of copy. Thus, the other posts will remain in $\mathcal{E}$ and p_i will never become an $s\text{-}post$. □

Theorem 3.9. *Let $\mathcal{I}$ be an instance without a popular matching, and let $p_i \in \mathcal{F}^{\mathcal{I}}$. Every instance with a popular matching, obtained from $\mathcal{I}$ by copying some posts, remains popular even with the original number of copies of p_i from $\mathcal{I}$.*

Proof. If $p_i \in \mathcal{F}^{\mathcal{I}}$, it cannot be assigned to an applicant in a popular matching. By Lemma 3.12, the proof is immediate. □

The first dominance rule is working for any copy of any post. Using the Corollary 3.3 another dominance rule can be described for the posts in $\mathcal{E}$.

Theorem 3.10. *Let $\mathcal{I}$ be an instance who admits or not a popular matching. Every instance obtained from $\mathcal{I}$ by copying any post p_i in $\mathcal{E}$, remains the same or can admit a popular matching.*

Proof. By using Lemma 3.9 and Corollary 3.3, one can conclude easily that any copy of a post in $\mathcal{E}$ will not change the labels, and may add more $s\text{-}posts$ in the instance. This can lead to find a popular matching. □

Remark 3.1. *These theoretical results lead us to consider only instances with a maximum number of copies for the $s\text{-}posts$ and to avoid the posts in $\mathcal{F}^{\mathcal{I}}$. These properties highlight the difficulty in identifying posts that do not need to be copied to find a popular matching. Indeed, the $s\text{-}posts$ can change only after a copy of $\mathcal{U}$ posts.*

3.4.5 Modelling the FIXINGCOPIES Problem

We show how we can use the popular matching CSP model and the results from Section 3.4.1 in order to solve the problem of fixing copies and to introduce an effective pruning rule. Let $\boldsymbol{\chi^k(p_i)}$ be defined as the number of copies fixed for the post p_i in the instance $\mathcal{I}^k$.

We saw that the FIXINGCOPIES problem can be split in two steps: first, fixing the number of copies for each post that creates an instance of the popular matching problem with ties, and second, by checking the popularity of this instance. For the second step, it is sufficient to use the GCC constraint as shown in Theorem 3.4.

We propose to explore a search tree where the root node represents $\mathcal{I}^0$ and any other node is associated with an instance obtained from its parent by increasing the number of copies of one post. In this tree structure, any descendant $\mathcal{I}^l$ of a node $\mathcal{I}^k$ has the following property:

$$\forall p_i \in P,\ \chi^l(p_i) \geq \chi^k(p_i). \tag{3.10}$$

Thus from each node, we can apply all the results obtained in Section 3.4.1, and in particular the Theorem 3.9. From this theorem, we can fix the number of copies of some posts for an instance node and its children. Therefore we apply the following pruning rule at each instance node in the tree for our model. Let $\mathcal{I}^l$ be a child of $\mathcal{I}^k$, then:

$$\forall i, \text{if } p_i \in \mathcal{F}^{\mathcal{I}^k},\ \text{then } \forall \mathcal{I}^l, \chi^l(p_i) = \chi^k(p_i). \tag{3.11}$$

As we explained in Remark 3.1, the identification of useful/useless posts in the fixing copies problem is complex. We obtained only one pruning rule for exploring the instance tree. However with the intention of improving the exploration of the tree and to reduce the number of instances visited with the pruning rule, we discuss the impact of the branching strategies based on the $\mathcal{O}, \mathcal{E}, \mathcal{U}$ labelling.

In particular, we highlight the impact of the copy of a post in a specific label. From Lemma 3.9 and Theorem 3.10 the copy from a post in $\mathcal{E}$ does not change the labels

and cannot remove popularity from other instances with copies from different labels. This result shows that branching on posts in $\mathcal{E}$ does not avoid some potential solutions in another branches of the search tree.

From Corollary 3.2 the copy from a post in $\mathcal{U}$ does not change the maximum matching, but changes at least two posts in $\mathcal{E}$. We can conclude that only the posts in $\mathcal{U}$ have a significant impact on the branching decision, indeed exploring the nodes, where labels $\mathcal{U}$ are considered last, should reduce the backtracking.

3.5 Experiments

In this section we will present the experimental results, first, for the popular matching and, second, the FixingCopies problem.

We have successfully implemented the CP models presented in this chapter using the Numberjack platform [Hebrard et al., 2010] using the constraint programming solver Mistral as a backend. Numberjack is a modelling package written in Python providing a flexible interface for problem specification and the ability to solve such a problem using mixed integer programming, satisfiability, or constraint programming techniques. All the tests run on Intel Xeon E5-2640 processors.

For our experiments we created, in Python, a parameterised instance generator for the popular matching and fixing copies problems. This generator allows us to analyse and compare the different methods proposed for our CP model. For this purpose, we have varied the following parameters: the number of the applicants and the number of posts; the lengths and completeness of the preference lists; and the probability of ties in the lists. To keep the empirical generation of the instances manageable, we restricted our attention to cases where the number of posts are equal to the number of applicants, represented by n, and all the preference lists have the same length. We characterised the ties by a single parameter t that an entry in a preference lists is tied with its predecessor.

In order to obtain a 'controlled' scale graph we generated a bipartite graph using a preferential attachment mechanism [Guillaume and Latapy, 2004]. The graph is initialised with an input set of $|A|$ vertices (i.e. the number of applicants for our model), then at each step according to a probability r an edge is added between the applicant selected and an existing post, or a new bottom post p_i is created. Secondly, from this generated graph, we create the preference lists for each applicant and each post is ranked following a probability computed from the generated graph. For the instances with ties, we

gave a probability denoted by tr to decide if an entry in a preference lists is tied with its predecessor.

3.5.1 Results for the Popular Matching

We implemented the standard popular matching algorithms (with and without ties) [Abraham et al., 2007b] in C++. The CP models are implemented in Mistral-2.0 [Hebrard, 2008], also written in C++. We generated large random instances where the number of applicants is equal to the number of posts. Four sets of instances are generated with 8500, 9000, 9500, and 10000 applicants. We generated 10 instances without ties for each set with different seeds. For the case with ties, we generated 15 instances per set with 5 different seeds and 3 different probabilities for distributing the ties.

The results are summarised in Table 3.1. Each column shows the average CPU runtime in seconds for one set of instances averaged over successful runs. The first part of the table concerns the case without ties and the second part shows the results for the case with ties. Note that every instance is solved by both configurations. The results of the standard algorithm are shown in the row 'Standard'. The CP model is presented with the row 'Mistral'. Best results are shown in bold face fonts.

Table 3.1: Popular matching: Summary of the results

Instances	8.5k	9k	9.5k	10k
		Without ties		
Standard	144	160	177	195
Mistral	152	168	191	215
		With ties		
Standard	153	170	**162**	211
Mistral	**144**	**162**	181	**204**

Table 3.1 shows that the CP models are very competitive. Indeed, the performance of the CP model is slightly worse but very close to the standard algorithm for the case without ties. Best results for the case with ties are, however, always obtained with Mistral, which is around 10 seconds faster than the standard algorithm..

3.5.2 Results for the FixingCopies Problem

We study the FixingCopies problem to evaluate the different branching strategies as well as the dominance rules. The exploration strategies are the six permutations to rank in priority the three labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$. The search tree is implemented in Python using Numberjack [Hebrard et al., 2010].

The timeout is fixed to 20 minutes. We first run all the configurations on purely random instances (general preference list), but observed that these instances are easy to solve. Therefore, we studied another family of instances based on a notion of a master preference list, as described below.

3.5.2.1 General Preference List

In Section 3.4.5, regarding the difficulty to prune the domains of variables and to identify the the posts that do not need to be copied, we split the fixing copies problem into two steps. The first one involves exploring the enumeration of each instance obtained by increasing the number of copies of the posts. In the second step, we use our CP model for solving the instance at hand using the GCC constraint. The experiments deal with the different methods of exploration presented previously. In order to use the properties introduced in Section 3.4.1, the exploration of the enumerated instances must be done such that each node explored respects the property (3.10). The principle is to explore the instances, in regard to the property (3.10), by putting priorities on the post labels that must be copied in first. The different exploration strategies come from the six permutations to rank the three labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$. We show empirically that the automaton obtained theoretically is the best ranking strategy to explore the instances.

The experimental results are presented in Table 3.2 and Table 3.3. With respect of our description, we generated instances with $n \in \{35, 40, 45, 50, 140, 160, 180\}$ applicants. For each set, we create 60 instances as follows: for each set, we have four probabilities $r \in \{50\%, 60\%, 70\%, 80\%\}$ for the preferential attachment bipartite graph, five seeds, and three probabilities for having ties: $tr \in \{10\%, 15\%, 20\%\}$. Then we present the six ranking strategies based on the permutation of the labels. For instance *eou* means that we copy the nodes in $\mathcal{E}$ before $\mathcal{O}$, and then the nodes in $\mathcal{O}$ before $\mathcal{U}$. There is a Boolean $0, 1$ at the end of each configuration to say whether we use the pruning rule that we proposed (0=no pruning, 1 = pruning). For each configuration, we have reported the percentage of finding successful solutions, the number of decisions (or subproblems tested with the CP model) (D) and the runtime (T) in seconds for the

Table 3.2: Summary of the results for small instances

	35			40			45			50		
Labels	%sol	D	T	%sol	D	T	%sol	D	T	%sol	D	T
euo-0	**100**	384.93	20	94	1269.94	30	94	81.50	8	96	161.80	59
euo-1	**100**	319.12	12	94	1269.40	33	94	81.50	13	96	77.29	22
eou-0	**100**	384.93	17	96	4.54	0	94	5.50	0	96	161.80	58
eou-1	**100**	384.81	16	96	4.47	0	94	5.28	0	96	161.69	50
ueo-0	92	2912.04	57	96	4.47	0	94	5.28	0	96	161.69	50
ueo-1	92	2912.04	57	98	4102.18	110	77	7016.60	286	55	240.23	6
uoe-0	92	2912.04	58	96	4102.18	154	77	7016.60	219	54	240.23	10
uoe-1	92	2912.04	57	98	4102.18	119	79	7016.60	288	55	240.23	7
oeu-0	**100**	5.55	0	96	4102.18	153	77	7016.60	216	**100**	200.65	11
oeu-1	**100**	5.55	0	**100**	3.00	0	**100**	5.75	0	**100**	200.65	6
oue-0	**100**	5.55	0	**100**	3.00	0	**100**	5.75	0	**100**	200.65	11
oue-1	**100**	5.55	0	**100**	3.00	0	**100**	5.75	0	**100**	200.65	6

Table 3.3: Summary of the results for large instances

	140			160			180		
Labels	%sol	D	T	%sol	D	T	%sol	D	T
euo-0	36	11.00	51	13	13.00	1026	13	20.12	162
euo-1	36	11.00	51	11	13.00	1025	13	18.62	166
eou-0	36	11.00	51	11	13.00	1029	13	20.12	162
eou-1	36	11.00	51	11	13.00	1024	13	18.62	166
ueo-0	5	7.66	4	1	10.00	6	1	10.00	781
ueo-1	5	7.66	4	1	10.00	6	1	10.00	775
uoe-0	5	7.66	4	1	10.00	6	1	10.00	780
uoe-1	5	7.66	4	1	10.00	6	1	10.00	771
oeu-0	**81**	8.00	15	**50**	14.00	4	**48**	16.13	81
oeu-1	**81**	8.00	15	**50**	14.00	4	**48**	16.13	81
oue-0	**81**	8.00	15	**50**	14.00	4	**48**	16.13	81
oue-1	**81**	8.00	15	**50**	14.00	4	**48**	16.13	81

successful instances.

The first clear observation from Tables 3.2 and 3.3 is that all enumeration strategies starting from posts in $\mathcal{O}$ outperform the other strategies. For instance, with 140 applicants, any strategy starting by copying posts in $\mathcal{O}$ solves 81% of the instances, whereas the best strategy for the rest solves no more than 36%. This confirms that the automaton obtained theoretically is indeed the best ranking strategy for guiding the exploration of the search space.

It should be noted also that the strategies starting from posts in $\mathcal{U}$ behaves poorly in all sets, and in fact is the worst strategy overall. This is not surprising since we know by Corollary 3.2 that in any new instance obtained from the copy of a post in $\mathcal{U}$, the maximum matching will remain the same. So the chances of having a popular matching are low.

Last, we observe that the pruning rule that we introduced does not help in the general preference list. Therefore we construct preference lists derived from a master preference list, as described in [Irving et al., 2008].

3.5.2.2 Master Preference List

A master list models real-world matching problems in which posts are ranked according to some global objective criterion. The master list may be allowed to contain ties or may be strict. Irving et al. [Irving et al., 2008] considered the stable marriage problem in the presence of master preference lists and proved that many interesting variants

remain hard under this master list model. In the same vein, we consider the popular matching problem in the presence of a master list.

We generated the instances as follows: the number of applicants $a \in \{100, 150, 200\}$; the number of posts $p \in \{\frac{a}{2}, a\}$; the upper bound $ub \in \{\frac{a}{20}, \frac{a}{10}\}$ for each post copies; and each post has a probability $h \in \{50\%, 70\%, 90\%\}$ to be in the applicant preference list (following the order given by the master preference list). For each configuration we use 40 different randomised seeds. For instance in Table 3.4, $100a\_50p\_ub10$ is associated with the set of 100 applicants with 50 posts with 10 upper bounds of copies for each post. Each row summarises the results of each branching strategy. For example, for eou any post in $\mathcal{E}$ has the priority to be copied first, followed by the posts in $\mathcal{O}$ and the posts in $\mathcal{U}$. The ending in -0 means we did not use the dominance rule and -1 otherwise. A run is said to be *successful* when a solution is found or unsatisfiability is proven within the time limit. For each configuration, we report the percentage of successful runs $\%sol$; the number of decisions (D); and the runtime (T) in seconds of the successful instances. The results shows the average over successful runs, which means the instances that have returned a solution, either SAT or UNSAT.

Our dominance rules are extremely beneficial as we can see in the results. Any configuration is better when using the dominance rule (e.g. $eou-0$ and $eou-1$ in Table 3.4, Table 3.5, Table 3.6, Table 3.7). Second observation, in all the tables is that if we increase the upper bound of the copies we notice an increase in the number of successful runs.

Observations we made previously are also true for the experiments that are not reported here.

Table 3.4: FIXINGCOPIES: 100 applicants, 50 and 100 posts, 10% and 20% upper bound, and $h = 50\%$

	$100a\_50p\_ub10\_50h$			$100a\_50p\_ub20\_50h$			$100a\_100p\_ub10\_50h$			$100a\_100p\_ub20\_50h$		
Labels	%sol	D	T	%sol	D	T	%sol	D	T	%sol	D	T
euo-0	10%	431.25	39.24	90%	863.33	193.41	7%	236.40	879.00	2%	1041.67	1578.00
euo-1	15%	113.33	5.82	100%	176.42	10.07	10%	276.92	2363.50	100%	14.80	182.82
eou-0	15%	420.83	38.35	100%	814.82	163.30	7%	228.54	871.66	2%	1028.60	1578.00
eou-1	15%	113.33	5.81	100%	176.42	10.04	10%	276.79	2363.50	100%	14.82	182.82
ueo-0	10%	431.25	39.34	90%	863.41	191.09	7%	237.29	879.00	2%	1024.70	1578.00
ueo-1	10%	132.00	7.38	90%	211.61	13.64	7%	7.86	108.00	97%	18.57	211.64
uoe-0	10%	298.25	34.42	80%	618.25	130.00	5%	203.90	743.50	62%	494.19	977.00
uoe-1	10%	93.50	6.33	87%	270.57	25.14	5%	10.82	113.50	85%	13.80	141.17
oeu-0	15%	265.83	25.57	100%	414.32	67.05	7%	148.75	632.00	95%	363.29	844.44
oeu-1	15%	91.83	5.90	100%	119.82	8.54	10%	275.55	2334.50	100%	13.98	129.80
oue-0	10%	298.25	34.42	80%	618.18	130.66	5%	203.49	743.50	62%	491.73	976.92
oue-1	10%	93.50	6.28	87%	270.51	25.24	5%	10.85	113.50	85%	13.99	141.11

If we consider the Table 3.5, and Table 3.7 increasing the similarity between the preference lists will not make the instances easier in general (i.e., when $h \in \{70\%, 90\%\}$). Therefore, increasing the number of applicants, Table 3.4 and Table 3.6, the dominance rule is more obvious.

Having a closer look at the branching strategies we see in Table 3.4 that the rows $eou-0$, $eou-1$, $oeu-0$, $oeu-1$, the percentage of successful runs is higher than the others strategies. Moreover, if we have a closer look at the decision number and running time we can notice a huge difference between using the dominance rule $eou-1$ and not $eou-0$. Second, if the dominance is turned on, then the best strategies are the ones prioritising the branching of posts in $\mathcal{E}$ (e.g. $eou-1$ and $euo-1$,). Also, when the posts in $\mathcal{E}$ are not given the top priority, it is always better to branch on them in the second place (e.g. $oeu-1$ is better than $oue-1$). As explained in Section 3.4.5, branching on $\mathcal{E}$ posts first reduces the search effort, notice the number of decisions. However, if the post is either an f-post or s-post (as enforced by the first dominance rule), the new copy will be included in some domains and will increase the likelihood of finding a solution.

An important observation is about the branching on posts in $\mathcal{U}$. Indeed as we deduce in Section 3.4.5, any copy of these posts has an important impact. Branching on $\mathcal{U}$ for the last will decrease the number of decisions and increase the number of solutions.

Table 3.5: FIXINGCOPIES: 100 applicants, 50 and 100 posts, 10% and 20% upper bound, and $h = 70\%$

	$100a\_50p\_ub10\_70h$			$100a\_50p\_ub20\_70h$			$100a\_100p\_ub10\_70h$			$100a\_100p\_ub20\_70h$		
Labels	%sol	D	T	%sol	D	T	%sol	D	T	%sol	D	T
euo-0	15%	1277.50	216.40	92%	854.94	185.04	10%	235.22	873.75	12%	907.67	1501.20
euo-1	20%	126.12	6.97	100%	172.80	9.71	15%	9.16	121.00	100%	14.07	174.85
eou-0	20%	427.62	39.32	100%	815.47	164.37	15%	233.87	875.50	12%	919.32	1501.20
eou-1	20%	126.12	6.96	100%	172.80	9.75	15%	9.18	121.00	100%	13.92	174.85
ueo-0	15%	1277.50	217.04	92%	864.81	189.37	10%	236.58	873.75	12%	823.22	1421.40
ueo-1	15%	965.50	79.1	92%	200.32	12.28	12%	164.41	1667.60	92%	16.73	195.83
uoe-0	7%	360.00	41.44	70%	612.89	124.37	2%	179.68	726.00	72%	428.15	900.93
uoe-1	7%	99.00	6.61	77%	187.19	15.59	10%	28.75	315.25	90%	12.78	132.88
oeu-0	20%	387.62	44.36	100%	398.92	57.77	15%	195.04	746.33	92%	291.73	750.64
oeu-1	20%	123.25	8.48	100%	122.00	8.67	15%	10.35	116.33	97%	12.85	129.00
oue-0	7%	360.00	40.40	72%	594.75	121.51	2%	180.11	726.00	72%	424.58	890.65
oue-1	7%	99.00	6.64	80%	184.06	15.46	10%	28.80	315.25	90%	12.91	132.52

Table 3.6: FIXINGCOPIES: 200 applicants, 100 and 200 posts, 20% and 40% upper bound, and $h = 50\%$

	$200a\_100p\_ub20\_50h$			$200a\_100p\_ub40\_50h$			$200a\_200p\_ub20\_50h$			$200a\_200p\_ub40\_50h$		
Labels	%sol	D	T	%sol	D	T	%sol	D	T	%sol	D	T
euo-0	0%	-	-	0%	-	-	0%	-	-	0%	-	-
euo-1	17%	88.28	283.14	100%	411.95	135.93	17%	119.78	268.42	100%	187.80	397.55
eou-0	0%	-	-	0%	-	-	0%	-	-	0%	-	-
eou-1	17%	88.15	283.14	100%	411.95	135.25	17%	123.33	268.42	100%	187.80	397.55
ueo-0	0%	-	-	0%	-	-	0%	-	-	0%	-	-
ueo-1	12%	89.78	308.60	87%	482.25	186.02	12%	114.20	277.20	92%	254.59	472.81
uoe-0	0%	-	-	17%	497.54	609.28	0%	-	-	7%	292.39	189.33
uoe-1	10%	82.48	222.50	70%	334.21	155.38	7%	95.28	198.33	92%	190.25	298.35
oeu-0	2%	737.76	910.00	20%	461.35	613.75	0%	-	-	12%	359.52	435.00
oeu-1	17%	112.91	279.28	100%	293.52	137.69	17%	140.09	258.14	97%	176.82	290.46
oue-0	2%	743.26	910.00	17%	500.43	608.71	0%	-	-	5%	97.96	187.00
oue-1	10%	83.45	222.50	70%	334.07	159.30	7%	96.01	198.33	90%	180.42	301.25

Table 3.7: FIXINGCOPIES: 200 applicants, 100 and 200 posts, 20% and 40% upper bound, and $h = 90\%$

	$200a\_100p\_ub20\_90h$			$200a\_100p\_ub40\_90h$			$200a\_200p\_ub20\_90h$			$200a\_200p\_ub40\_90h$		
Labels	%sol	D	T	%sol	D	T	%sol	D	T	%sol	D	T
euo-0	0%	-	-	0%	-	-	0%	-	-	0%	-	-
euo-1	15%	271.50	78.65	100%	419.10	144.03	17%	112.50	267.85	100%	179.47	389.25
eou-0	0%	-	-	0%	-	-	0%	-	-	0%	-	-
eou-1	15%	271.50	79.06	100%	419.10	43.31	17%	112.36	267.85	100%	179.26	389.25
ueo-0	0%	-	-	0%	-	-	0%	-	-	0%	-	-
ueo-1	5%	308.00	85.89	95%	492.39	198.31	10%	145.62	322.00	97%	257.48	473.10
uoe-0	0%	-	-	17%	372.00	234.24	0%	-	-	10%	150.48	245.75
uoe-1	5%	295.50	120.29	87%	316.60	144.61	7%	121.32	232.66	82%	196.80	327.90
oeu-0	2%	965.00	921.70	25%	564.80	441.0	2%	540.29	607.00	12%	184.20	275.80
oeu-1	15%	268.83	105.39	97%	308.43	150.64	17%	128.73	244.71	100%	156.55	267.07
oue-0	0%	-	-	20%	349.25	216.11	0%	-	-	12%	146.09	231.60
oue-1	5%	295.50	121.25	90%	313.08	145.22	7%	122.38	232.66	85%	198.76	323.41

3.6 Chapter Summary

In this chapter we proposed a CP formulation for the popular matching problem that can handle cases in which there are ties in the applicants' preference lists. We reported the first polynomial time algorithm to find optimal popular matching in the presence of ties in the preference lists. Next, we showed novel graph properties and new dominance rules for the popular matching with copies. Our experiments on hard instances of the popular matching problem with copies showed essentially the key aspects of a good branching strategy as well as the effectiveness of our dominance rules in improving efficiency of search.

Chapter 4

Collective Scenario-Based Algorithm for the Kidney Exchange Problem

"And suddenly, just like that, hope became knowledge. I was going to win. It was just a matter of when." Khaled Hosseini, The Kite Runner

4.1 Introduction

In this chapter we present a method to build an online anticipatory algorithm from an offline Kidney Exchange Problem (KEP) model, with the objective to maximize the number of transplants. We call this method the *Collective Scenario-Based Algorithm* (CSBA). We compare our method against: a baseline that reflects the current practice, for example, solving the offline model; an oracle operating under perfect information; and against the best performing algorithms from [Awasthi and Sandholm, 2009].

The KEP is a stochastic online problem, and can greatly benefit from the use of anticipatory algorithms. Unfortunately, most such algorithms suffer from scalability issues due to the reliance on scenario sampling, limiting their practical applicability. Instead, we recognize that the KEP allows for a sampling-free probabilistic model of future arrivals and drop-offs, which we capture via a so-called Abstract Exchange Graph (AEG). We show how an AEG-based approach can outperform sampling-based algorithms in terms of quality, while being comparable to a myopic algorithm in terms of scalability.

4.2 An Overview of Kidney Exchange Programmes

Organ transplantation is an effective solution for people suffering of kidney failure [Dickerson and Sandholm, 2013], but finding a viable organ can be very difficult because of the scarcity of donors. Waiting for the organ of a deceased person can take more than a couple of years, and buying and selling of organs is illegal in most countries. However, people have two kidneys and a person can live fine with just one. This has encouraged voluntary donors (e.g. family members), which however are often incompatible with the patient due to, for example, blood or tissue type. A solution to this involves having the incompatible pairs join a system that allows for the swapping of donors donors, for example, a kidney exchange program.

Centralised kidney exchange programs exist in many countries, including the US, the Netherlands and the UK [Manlove and O'Malley, 2014]. The program requires to solve at regular intervals an optimization problem (the Kidney Exchange Problem – KEP) to find a matching for the enrolled pairs that saves the largest possible number of lives. This variant of the KEP has been addressed via a number of effective approaches. In reality however, new pairs arrive (and unfortunately drop off) over time, making the problem *inherently on-line and stochastic*. In this setting, it is known that exploiting information about future outcomes, by means of an *anticipatory on-line algorithm* can lead to substantially better results. The availability of extensive medical data makes the estimation of future outcomes viable in practice.

On-line decision making can be framed as a form of multi-stage stochastic optimization. This class of problem is extremely hard to solve exactly, but high-quality solutions can be found via scalable sub-optimal methods (see [Van Hentenryck and Bent, 2009] and references therein). These approaches typically estimate future developments by sampling *scenarios* and optimize the (estimated) expected value of current decisions. A few of the main anticipatory algorithms from [Van Hentenryck and Bent, 2009] have been adapted and extended to the KEP in [Awasthi and Sandholm, 2009], but the task was not straightforward and forced the authors to introduce heuristic approximations.

In this chapter we present a simple method to build a on-line anticipatory algorithm starting from an off-line model for the KEP, with the objective to maximize the number of transplants. Compared with the algorithms from [Awasthi and Sandholm, 2009], our method is more general and provides an expected value estimation that is *correct within the limit of sampling errors*.

We compare our method against a baseline that reflects the current practice (for ex-

ample, solving the off-line model with the current pairs), against an oracle operating under perfect information, and against the best performing algorithms from [Awasthi and Sandholm, 2009]. On instances obtained via a realistic donor pool generation method, our approach improves over the state-of-the-art in terms of both number of transplants and scalability.

4.3 Formulations for the Kidney Exchange Problem

Most works in the literature have focused on finding the best matching for a given graph, for example, *on the off-line version of the KEP*. In the Operations Research community the problem has been considered in [Constantino et al., 2013, Anderson et al., 2015, Mak-Hau, 2015, Manlove and O'Malley, 2014], and more recently in [Alvelos et al., 2015, Dickerson et al., 2016]. The (off-line) KEP can also be seen as a generalization of the Stable Roommates Problem (SR) that is shown to be NP-complete in [Irving, 2007b] as long as at least 3-way exchanges are considered. The problem is also related to barter-exchange markets (in which agents seek to swap their items with one another in order to improve their social welfare) [Abraham et al., 2007a, Ashlagi and Roth, 2012].

4.3.1 The Cycle Formulation

One popular Mathematical Programming model for the KEP is the so-called *cycle formulation* [Roth et al., 2005]. The model is defined over a directed graph $\langle V, A(V)\rangle$, where V is the set of nodes (pairs and altruistic donors) and $A(V)$ is the set of arcs. We assume that $A(\cdot)$ is a function that maps a set of nodes to a set of arcs, based on their compatibilities.

Figure 4.1 shows an example of such a graph, with four patient/donor pairs and one altruistic donor (i.e. $\langle -, d_4\rangle$). A two-way exchange is given by: $(d_0 \rightarrow p_2, d_2 \rightarrow p_0)$; $(d_0 \rightarrow p_1, d_1 \rightarrow p_0)$; $(d_1 \rightarrow p_3, d_3 \rightarrow p_1)$, a three way exchange by $(d_0 \rightarrow p_2, d_2 \rightarrow p_1, d_1 \rightarrow p_0)$, a chain starting from an altruistic donor by d_4 give rise to the following chains: $(d_4 \rightarrow p_3, d_3 \rightarrow p_1)$; $(d_4 \rightarrow p_3, d_3 \rightarrow p_1, d_1 \rightarrow p_0)$; $(d_4 \rightarrow p_3, d_3 \rightarrow p_1, d_1 \rightarrow p_0, d_0 \rightarrow p_2)$, with the last donor giving his kidney to the system. We remark that most countries prefer to have a limit on the length of the chains as all the surgeries must take place simultaneously as no donor can pull out from the program once their patient pair has their kidney. Hence, in this example there are three two-way cycles, one three-way

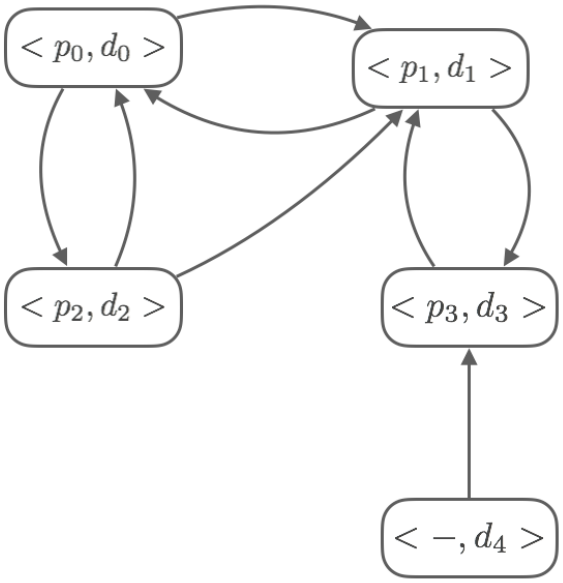


Figure 4.1: Example graph of a cycle formulation

cycle, and one two-way chain, one three-way chain, and one four-way chain.

The model requires enumerating the set $\mathcal{C}$ of all cycles that respect the constraints of the problem. Paths starting from altruistic donors are regarded as cycles. Note that in this formulation variables represent exchanges. Each cycle C_j corresponds to a set of nodes and is associated with a weight $w(C_j)$, equal to its cardinality when the goal is to maximize the number of transplants. The cycle formulation requires the introduction of a binary decision variable x_j for each cycle $c_j \in \mathcal{C}$ such that $x_j = 1$ iff the corresponding exchange has been chosen.

The model is then given by:

$$\max z = \sum_{j \in \mathcal{C}} w(C_j) x_j \tag{4.1}$$

$$\text{s.t.} \sum_{j \in \mathcal{C}, i \in C_j} x_j \leq 1, \qquad \forall i \in V \tag{4.2}$$

$$x_j \in \{0, 1\}. \qquad \forall j \in \mathcal{C} \tag{4.3}$$

The objective is to maximize the sum of weights, i.e. the total number of transplants. Constraint (4.2) prevents the model from using two cycles that share a com-

mon patient-donor pair, or altruistic donor. In the example of Figure 4.1, the enumaration of all cycles is: $\mathcal{C} = \{(d_0 \rightarrow p_2, d_2 \rightarrow p_0), (d_0 \rightarrow p_1, d_1 \rightarrow p_0), (d_1 \rightarrow p_3, d_3 \rightarrow p_1), (d_0 \rightarrow p_2, d_2 \rightarrow p_1, d_1 \rightarrow p_0), (d_4 \rightarrow p_3, d_3 \rightarrow p_1), (d_4 \rightarrow p_3, d_3 \rightarrow p_1, d_1 \rightarrow p_0), (d_4 \rightarrow p_3, d_3 \rightarrow p_1, d_1 \rightarrow p_0, d_0 \rightarrow p_2)\}$, and the weights of the cycles are: $w(C_j) = \{2, 2, 2, 3, 2, 3, 4\}$, and thus, the variables are the following $x_j = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$. A solution for this example is $x_1 = 1$ and $x_5 = 1$ with weight 5, meaning the cycle $(d_0 \rightarrow p_2, d_2 \rightarrow p_0)$ and the chain $(d_4 \rightarrow p_3, d_3 \rightarrow p_1)$.

The model is simple and capable of easily capturing complex constraints on cycle formation, e.g. cardinality restrictions. The main drawback is limited scalability, since the number of cycles is worst-case exponential in the maximum graph size. This issue has been addressed via column generation in [Abraham et al., 2007a, Dickerson et al., 2013, Glorie et al., 2014, Klimentova et al., 2014, Plaut et al., 2016]. Recently, alternative compact models have been proposed in [Alvelos et al., 2015, Dickerson et al., 2016] to improve scalability without resorting to column generation.

4.3.2 The Online Kidney Exchange Problem

In the online setting pairs appear and expire at each time step. A number of studies [Pedroso, 2014, Alvelos et al., 2015] have taken into account the effect of potential failures, but not of entering pairs. A simulator for the online KEP is presented in [Santos et al., 2017], but the authors still rely on periodic execution of an offline approach. To the best of our knowledge, the online version of the KEP has been targeted using anticipatory algorithms only in [Awasthi and Sandholm, 2009].

Formally, the on-line KEP is a multi-stage stochastic problem whose exact solution is a *policy tree*, which specifies recursively the best matching for each step and possible uncertain outcome. This approach has clear scalability issues, which are overcome in [Awasthi and Sandholm, 2009] via *scenario sampling*, based on the ideas from [Van Hentenryck and Bent, 2009] and references therein.

A scenario refers here to a set of nodes that may enter in the next h steps, where h is called the look-ahead horizon. A number n_s of scenarios can be obtained by sampling the probability distribution for pair entries (estimated from medical data). All scenarios are considered equally likely after sampling. We refer as V_0 to the nodes currently in the program and as V_s (with $s \in \{1, \ldots, n_s\}$) to the those entering under scenario s. An example of two simple scenarios is shown in Figure 4.2.

The two best performing algorithms from [Awasthi and Sandholm, 2009] are based

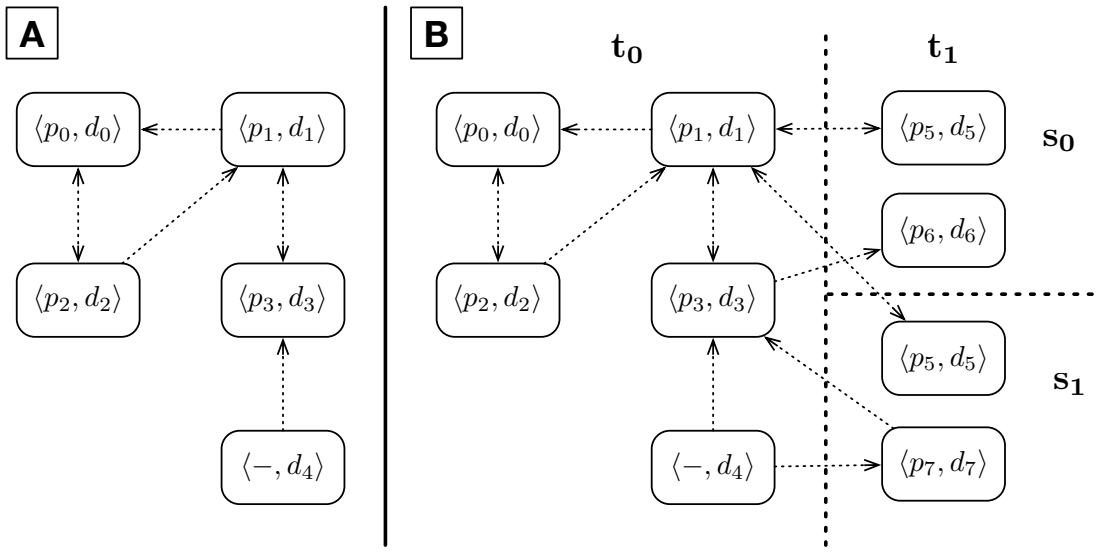


Figure 4.2: A: Example graph for an off-line KEP. B: Example graph for a scenario-based anticipatory approach

on: 1) sampling a subset of scenarios; 2) using the scenarios to compute scores for the cycles in the current time step; 3) solving a cycle-based off-line KEP for the current time step, with the computed scores. Both algorithms attempt to estimate the expected impact of choosing a cycle in the current time step, but they differ in how the estimate is computed.

The first algorithm in [Awasthi and Sandholm, 2009] (an adaptation of the REGRETS method from [Van Hentenryck and Bent, 2009]) computes cycle scores *by solving an off-line KEP for each scenario*. Cycles (in the current graph) that are chosen in the solution are rewarded with the value of the solution, cycles that are not chosen receive a fixed penalty δ. We refer to this methods as APST1 (from the initials of the authors): the pseudo-code is reported in Algorithm 1.

Algorithm 1: APST1 (adaptation of REGRETS)

```
Set score_j = 0 for each cycle C_j in ⟨V_0, A(V_0)⟩
for all scenario s ∈ {1 ... n_s} do
    Solve KEP on graph ⟨V_0 ∪ V_s, A(V_0 ∪ V_s)⟩
    for all cycle C_j in graph ⟨V_0, A(V_0)⟩ do
        if C_j is in the solution then
            score_j = score_j + sol. value
        else
            w_j = w_j − δ
Solve KEP on ⟨V_0, A(V_0)⟩ with the computed scores
```

The second algorithm in [Awasthi and Sandholm, 2009] (which shares ideas with the EXPECTATION method from [Van Hentenryck and Bent, 2009]) computes scores *by*

solving an off-line KEP for each cycle and scenario. The cycle scores are obtained by summing the solution values. We refer to this method as APST2 and its pseudo-code is in Algorithm 2.

Algorithm 2: APST2 (loosely based on EXPECTATION)

```
Set score_j = 0 for each cycle C_j in ⟨V_0, A(V_0)⟩
for all cycle C_j in graph ⟨V_0, A(V_0)⟩ do
    for all scenario s ∈ {1 ... n_s} do
        Solve KEP on ⟨V_0 ∪ V_s \ C_j, A(V_0 ∪ V_s \ C_j)⟩
        score_j = w_j + sol. value
Solve KEP on ⟨V_0, A(V_0)⟩ with the modified weights
```

4.4 The Collective Scenario-Based Algorithm

In the experiments from [Awasthi and Sandholm, 2009], both APST1 and APST2 performed considerably better than a non-anticipatory algorithm solving an off-line KEP at each time step. However, both algorithms have a few significant weak points.

First, since there is no closed-form formula for the computed scores, *they require the use of a cycle formulation for the final KEP*. This prevents the use of the compact models from [Dickerson et al., 2013, Pedroso, 2014, Alvelos et al., 2015] and *may lead to scalability issues*. This is particularly true for APST2, which also needs to solve a set of optimization problems for each cycle. Second, *none of the two methods provides an estimation of the expected number of transplants that is accurate up to the sampling error*. The scores computed by APST2 are proportional to the expected value of choosing a single cycle, but the approach disregards non-linear effects that arise when multiple cycles are chosen.

Third, *both APST1 and APST2 sometimes cannot delay exchanges, even if it may have a beneficial effect on the long run*. This is illustrated in Figure 4.3, which shows solutions for the example from Figure 4.2 (node descriptions have been replaced with numbers). We assume that the scenarios represent the only possible future outcomes; dark-shaded nodes are included in the matching for the current time step t_0; light-shaded nodes will be included in a matching at t_1 for at least one scenario, while non-shaded nodes have no chance to be included in a matching. A non-anticipatory algorithm would include at t_0 as many nodes as possible, leading to the solution in Figure 4.3A. APST2 always computes positive scores, and therefore its final KEP will necessarily choose to include the chain $4 \rightarrow 3$. This yields the solution from Figure 4.3B, which is sub-optimal in terms of expected value. In this simple example

APST1 would assign a negative score to the chain $4 \rightarrow 3$ and to the cycle $1 \leftrightarrow 3$, leading to Figure 4.3C and the optimal expected value. However, this depends in general on the (heuristic) penalty value δ, and may fail to happen in a more complex example.

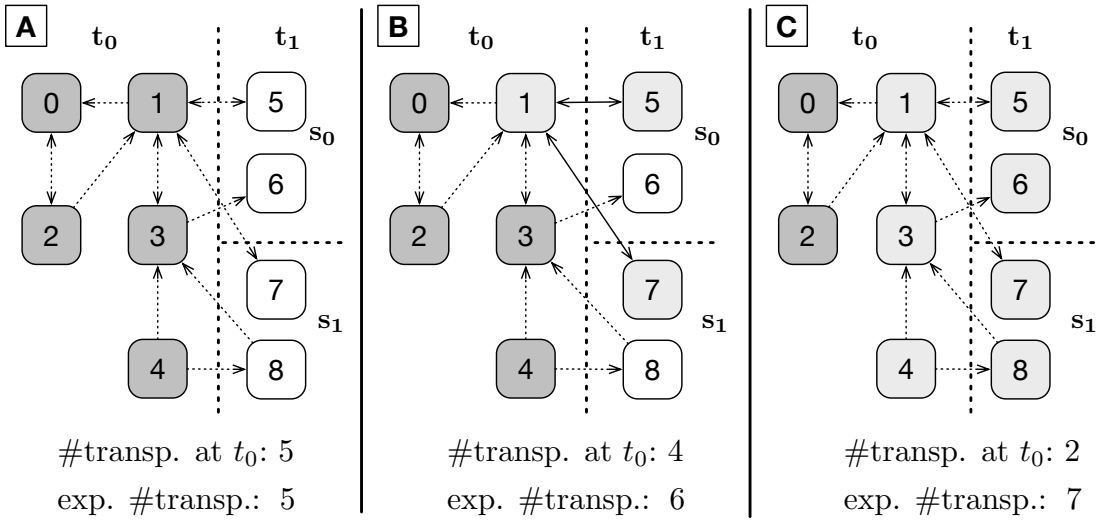


Figure 4.3: A: A non-anticipatory solution. B: The solution returned by APST1; C: the solution with the best expected value (returned by APST2 in this case)

We propose to address all these issues by *solving a single KEP on a graph constructed using all scenarios*, but with modified constraints. The final matching is given by all exchanges in the solution that are defined solely over nodes from the current time step. Indeed, we simply propose to employ the classical Sample Average Approximation scheme for two-stage stochastic programs (see e.g. [Shapiro et al., 2014]), which to the best of our knowledge has never been applied in this context.

Our approach is not tied to a specific KEP formulation, and hence we start by providing an abstract description. Let x be the set of problem variables (both for the current time step and the scenarios). Let F be the feasible set for such decisions. Finally, let $u_{s,i}(x)$ be a function denoting the number of times node i is used (i.e. participates in an exchange), assuming that scenario s occurs. Then our abstract model is given by:

$$\max z = \frac{1}{n} \sum_{s=1}^{n_s} \sum_{i \in V_s} u_{s,i}(x) \tag{4.4}$$

$$\text{s.t. } u_{s,i}(x) \leq 1 \qquad \forall s = 1 \dots n_s, \forall i \in V_0 \cup V_s \tag{4.5}$$

$$x \in F \tag{4.6}$$

which assumes equally likely scenarios. Equation (4.4) is the expected total amount of transplants, while Equation (4.5) states that, considering each scenario individually, the same node cannot participate into two exchanges.

In the case of the cycle formulation, the approach can be instantiated by introducing a variable for each cycle (including those from the scenarios). Specifically, let $\mathcal{C}_0$ be the set of cycles in graph $\langle V_0, A(V_0)\rangle$ and let $\mathcal{C}_s$ be the set of cycles in $\langle V_0 \cup V_s, A(V_0 \cup V_s)\rangle$, excluding the cycles already in $\mathcal{C}_0$. Let $x_{s,j}$ be the variable associated to $C_{s,j}$. Then we have:

$$u_{s,i}(x) = \sum_{\substack{C_{0,j} \in \mathcal{C}_0, \\ i \in C_{0,j}}} x_{0,j} + \sum_{\substack{C_{s,j} \in \mathcal{C}_s, \\ i \in C_{s,j}}} x_{s,j} \tag{4.7}$$

A node is used when scenario s occurs if it participates in a cycle in $\mathcal{C}_0$ or to a cycle in $\mathcal{C}_s$. By substituting in the abstract formulation we obtain:

$$\max z = \sum_{j \in \mathcal{C}_0} w(C_{0,j}) x_{0,j} + \frac{1}{n_s} \sum_{s=1}^{n_s} \sum_{i \in \mathcal{C}_s} w(C_{s,j}) x_{s,j} \tag{4.8}$$

$$\text{s.t.} \sum_{j \in \mathcal{C}_0, i \in C_{0,j}} x_j \leq 1 \qquad \forall i \in V_0 \tag{4.9}$$

$$u_{s,i}(x) = \sum_{\substack{C_{0,j} \in \mathcal{C}_0, \\ i \in C_{0,j}}} x_{0,j} + \sum_{\substack{C_{s,j} \in \mathcal{C}_s, \\ i \in C_{s,j}}} x_{s,j} \qquad \begin{array}{r} \forall s = 1 \dots n_s, \forall i \in V_0 \cup V_s, \\ \exists C_{s,j} \in \mathcal{C}_s : i \in C_{s,j} \end{array} \tag{4.10}$$

$$x_{s,j} \in \{0, 1\} \qquad \forall s = 0 \dots n_s, \forall j \in \mathcal{C}_s \tag{4.11}$$

where terms related exclusively to cycles in the current time step have be collected via factorization. This improves the model performance at the expense of readability. We stress that the approach does not apply exclusively to the cycle formulation, but the cycle formulation provides a good instantiation example due to its simplicity.

4.5 A Sampling-free Model

In this chapter we recognize that, with a few reasonable assumptions, the KEP allows for the construction of a sampling-free probabilistic model of future arrivals and drop-offs. We capture this information via a so-called Abstract Exchange Graph (AEG), whose nodes represent “types" of pairs, rather than individual pairs, and are associated to probability values. An AEG can be easily (and efficiently) obtained from medical or historical data.

By relying on the AEG, we describe how to enrich a given optimization model for the myopic KEP with an anticipatory component, often with very limited impact on its scalability. By doing this, the model becomes capable of taking into account both

future arrivals and drop-offs. For the sake of simplicity, we show how to apply our technique to a specific KEP approach, i.e. the cycle formulation.

In an experimentation on instances obtained via a realistic simulator, we show how sampling-based anticipatory algorithms quickly run into scalability issues, and even fail to provide high quality solutions in case of insufficient sample numbers. Conversely, an AEG-augmented method is consistently able to outperform its myopic counterpart, while having a comparable run-time. While the current experimentation is preliminary and limited in scale, these qualities make our technique one of the few that can hope to address nation-wide programs with thousands of enrolled pairs

4.5.1 Anticipatory Approaches

A few of the main anticipatory algorithms from [Van Hentenryck and Bent, 2009] have been adapted to the KEP in [Awasthi and Sandholm, 2009], but the task was not straightforward and forced the authors to introduce heuristic approximations. These algorithms will be described in our results section, since they are considered in our experimentation. Here, we simply observe that they all rely on scenario sampling, a scenario being a set of nodes that may enter in the next few steps. The main two algorithms from [Awasthi and Sandholm, 2009] require the solution of a multiple (smaller) off-line KEPs, while the method described in Section 4.4 solves a single modified KEP obtained via the Sample Average Approximation [Pagnoncelli et al., 2009].

The only sampling-free anticipatory methods for the online KEP to-date are those from [Dickerson et al., 2012a, Dickerson and Sandholm, 2015]. They are both based on the idea of discounting the value of each cycle with the lost “potential" of the involved nodes. Formally, the new weight of each cycle is given by $w_j - \sum_{i \in C_j} v_i$, where v_i is the estimated potential of $i \in N$. A set of exchanges can then be obtained by solving an off-line KEP with the modified weights. In [Dickerson et al., 2012a] the potentials are estimated via probability considerations, while [Dickerson and Sandholm, 2015] employs a parameter tuning algorithm. Even if the technique proposed here does not rely on sampling, from a mathematical point of view it is more akin to the sampling-based algorithms than to these sampling-free methods. For this reason, a comparison with either [Dickerson et al., 2012a] or [Dickerson and Sandholm, 2015] is missing in this chapter, but we still consider that a priority for future research.

To the best of our knowledge, these are all the methods for the online KEP in the literature. Works [Dickerson et al., 2013, Pedroso, 2014, Alvelos et al., 2015] have considered the effect of potential failures, but not of entering pairs. A simulator for the

online KEP is presented in [Santos et al., 2017], but the authors still rely on periodic execution of an offline approach.

4.5.2 The Abstract Exchange Graph

Here we present our main data structure, i.e. the Abstract Exchange Graph, and its potential applications. Our contributions stem from two simple assumptions, which we state initially in their basic form.

First, all sampling-based anticipatory methods from the literature attempt to account for the arrival of individual pairs/altruistic donors. However, pairs having the same connectivity lead to equivalent sets of cycles: in the absence of an external criterion for favoring certain pairs over others, we have:

Assumption 1. *Nodes having the same connectivity are equivalent.*

Assumption 2. *Arrivals are independent and identically distributed (i.i.d.).*

Population changes occur slowly over time, hence, during regular operation we apply Assumption 2.

Both assumptions are imposed for technical feasibility and can actually be relaxed, and we will show how this can be done at the end of this section.

Formally, the AEG is an annotated directed graph $\langle N, A(N), p\rangle$. Rather than to individual pairs, the nodes correspond to classes ("types") of equivalent pairs. The arcs correspond to "types" of transplants, and can be obtained via the same function A used for the classical KEP. Each node can be associated to a probability value $p_i \in [0, 1]$. Overall, the AEG specifies in a compact fashion both the arrival probabilities of future nodes and the cycles they can form.

4.5.3 Obtaining an AEG

From a mathematical standpoint, the AEG is a simple extension of the compatibility graph used by most KEP approaches. In particular, a "concrete" compatibility graph can be seen as an AEG where $p_i = 1$ for all nodes. This observation allows to easily obtain an AEG from historical data.

Kidney exchange programs usually keep a record of the participating pairs, which can be used to populate a set N. This set can form the basis for a "concrete" graph.

Starting from this graph, an AEG can be obtained by iteratively merging nodes with the same connectivity, as described in Algorithm 3. Once the graph can no longer be contracted in this fashion, all p values are normalized so that they represent frequencies of occurrence, i.e. estimated probabilities.

Algorithm 3: AEG extraction

Require: A graph $\langle N, A(N), p\rangle$, with $p_i = 1\ \forall i \in N$
  loop
    Search for two nodes $i, j \in N$ with the same outgoing/ingoing arcs, i.e.:
    A) $(i,h) \in A(N) \Leftrightarrow (j,h) \in A(N)$ and
    B) $(h,i) \in A(N) \Leftrightarrow (h,i) \in A(N)$
    if such nodes exist **then**
      $N = N \setminus j$ (i.e. remove node j)
      $p_i = p_i + p_j$ (i.e. compute aggregated count)
    else
      set $p_i = p_i / \sum_i p_i$ (normalize counts)
      break loop
  return $\langle N, A(N), p\rangle$

This process always yields a valid AEG. As an example, assume that we start from the compatibility graph in Figure 4.4A: nodes are labeled with the corresponding patient donor pair, and node 0 corresponds to an altruistic donor. Nodes having the same connectivity (i.e. (p_3, d_3) and (p_4, d_4)) are colored the same shade of gray. Figure 4.4B shows the AEG produced by Algorithm 3,where t_0 is associated to the altruistic donor, t_1 to (p_1, d_1) and t_2 to (p_2, d_2), and t_3 is an aggregated node merging (p_3, d_3) and (p_4, d_4), and with the estimated probabilities reported next to each node.

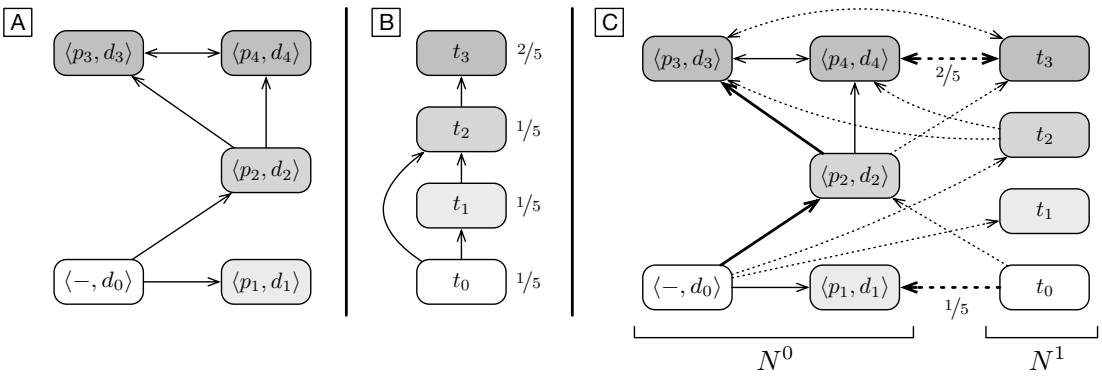


Figure 4.4: A) A "concrete" compatibility graph; B) The corresponding AEG; C) The AEG-base graph used in optimization

An AEG may in principle be quite large, thus undermining scalability. In practice in all our experiments the AEG size was always fairly limited, even if the design of our simulator disregards some real world effects that should *reduce*, rather than increase the number of types.

4.5.4 Using the AEG in Optimization

The AEG can be employed to enrich a model for the off-line KEP with an anticipatory component. We will describe the process using the abstract KEP from Section 4.5.2.

We start by augmenting the problem graph with h instances of all the AEG nodes, so as to represent h future arrival events. Formally, let N^0 be the original set of nodes, N^k be the k-th instance of the AEG nodes, and N be the set of all nodes (i.e. $N = N^0 \cup N^1 \cup \dots N^h$). We then construct the graph:

$$\langle N, A(N^0) \setminus A(N^1) \setminus \dots A(N^h), p^0 \cup p^1 \cup \dots p^h \rangle.$$

Arcs from $A(N^1)$ to $A(N^k)$ are removed since multiple nodes from the same N^k (with $k > 0$) correspond to different possible types *for a single arrival event*. The p_i values associated to nodes in N^0 are all equal to 1, while other p_i values are those from the AEG. An example of such a graph for $h = 1$ is shown in Figure 4.4C, where for the sake of simplicity we assume that N^0 is given by the graph from Figure 4.4A. Dashed arcs connect nodes in N^0 with nodes in N^1.

Exchanges defined exclusively over nodes in N^0 can be interpreted as usual. Exchanges involving even a single node from N^k with $k > 0$ represent *potential future exchanges*, and are associated with a stochastic strategy, similar to those employed in Markov Decision Processes.

Formally, *all* future exchanges (not just the selected ones) are collected in a set $\mathbf{x^F}$. Each of such exchanges is associated to a 0-1 random variable with a Bernoulli distribution. The probability that the variable is equal to 1 captures the odds that 1) the involved nodes arrive; and 2) we choose to perform the exchange: *what we need to decide are these probabilities*. With this approach, we can provide an abstract description of the AEG-based anticipatory KEP:

$$\max z = \sum_{\mathbf{x_i} \in \mathbf{x}} value(\mathbf{x_i}) + \sum_{\mathbf{x_i} \in \mathbf{x^F}} \mathrm{E}\left[value(\mathbf{x_i})\right] \qquad (\mathbf{KEP_A}) \qquad (4.12)$$

$$\text{s.t. } usage_S(\mathbf{x}) + \mathrm{E}\left[usage_S(\mathbf{x^F})\right] \leq \prod_{i \in S} p_i \qquad \forall S \subseteq N \qquad (4.13)$$

$$valid(\mathbf{x_i}) \qquad \forall \mathbf{x_i} \in \mathbf{x} \cup \mathbf{x^F} \qquad (4.14)$$

In the objective function Equation (4.12), we maximize the utility of current exchanges, plus the expected utility of future exchanges. Constraints (4.13) state that the expected number uses of each subset of nodes S cannot exceed the expected number of its

occurrences. For set of nodes in N^0, all p_i values are 1 and the formula boils down to the one in the KEP abstract model. For exchanges that involve future nodes, we take into account the probability that all involved nodes are present. All exchanges should be valid. In principle there is an exponential number of Constraints (4.13): in practice, however, the number is usually polynomial due to restrictions on the exchange size. Moreover, many subset S lead to redundant constraints that can be eliminated. Constraints 4.14 make sure to filter the unnecessary cycles, i.e. considers only the cycles that have at least a node in the respective time window, and filters the cycles with only future nodes.

The thick arcs in Figure 4.4C show the exchanges that would be selected in a (sub-optimal) solution of $\mathrm{KEP_A}$ formulation: the solid ones correspond to (selected) deterministic exchanges, while the dashed ones to stochastic exchanges with a non-zero probability (reported as a label on the arc). The main appeal of the technique is that the abstract formulation $\mathrm{KEP_A}$ is anticipatory, *and yet it does not rely on sampling*. Provided that our assumptions hold, the probabilistic model we employ gives a characterization of future uncertainty that is *limited in accuracy only by our estimates of the p_i values*.

4.5.5 Grounding Based on the Cycle Formulation

Our technique is general, but particularly easy to ground on the cycle model. Let $\mathcal{C}$ be the set of all cycles of valid length in the (augmented) graph. Let $\mathcal{C}^0$ be the subset of cycles that involve only nodes in N^0, and let $\mathcal{C}^F = \mathcal{C} \setminus \mathcal{C}^0$ be the set of "future" cycles, i.e. those involving at least one node in N^k with $k > 0$. We can then introduce: 1) a variable $x_j \in \{0, 1\}$ for each $C_j \in \mathcal{C}^0$, such that $x_j = 1$ iff the cycle is chosen in the solution; plus 2) a variable $x_j \in [0, 1]$ for each cycle $C_j \in \mathcal{C}^F$, representing the probability that the involved nodes arrive and the cycle is chosen. We can than state the new formulation (CYF_A) as a Mixed Integer Linear Program :

$$\max z = \sum_{C_j \in \mathcal{C}} w_j x_j \qquad (\mathbf{CYF_A}) \qquad (4.15)$$

$$\text{s.t.} \sum_{C_j \in \mathcal{C}, S \subseteq C_j} x_j \leq \prod_{i \in S} p_i \qquad \forall S \in \mathcal{S} \qquad (4.16)$$

$$x_j \in \{0, 1\} \qquad \forall C_j \in \mathcal{C}^0 \qquad (4.17)$$

$$x_j \in [0, 1] \qquad \forall C_j \in \mathcal{C}^F \qquad (4.18)$$

In 4.15 we are maximising the weight of the cycles Since for cycles in $\mathcal{C}^F$ the value

of x_j naturally represents a probability, summing the variables in Constraints (4.16) is enough to obtain the Constraint 4.16 Constraint 4.17 states that the variables in the current time step must be boolean, and Constraint 4.18 states that cycles involving future nodes can be fractionals.

The family $\mathcal{S}$ contains all subsets of nodes such that lead to non-redundant constraints. Formally, $\mathcal{S}$ contains all $S \subseteq N$ such that:

- the cardinality is not greater than the allowed cycle length, i.e. $|S| \leq L$;
- all nodes S appear together in at least one cycle, i.e. $\exists C_j \in \mathcal{C}$ s.t. $S \subseteq C_j$, since subsets that do not appear in any cycle are irrelevant;
- Finally, if all nodes in S are in N^0, then the cardinality of S is exactly 1, since for subsets entirely in N^0 posting the constraints for each individual node is sufficient.

There are three key facts to observe: first, the $\mathrm{CYF_A}$ model has the same structure of the original cycle formulation, meaning that most techniques employed to increase its scalability (e.g. column generation) should still be applicable. Second, the lack of arcs between nodes in the same N^k acts as a mitigation factor for the number of cycles. Third, all the variables related to the anticipatory components are real-valued, and therefore much easier to handle for the solver. Conversely, all sampling-based approaches require 0-1 variables to handle the cycles appearing in scenarios. The main drawback is the increased number of constraints, whose adverse effects will need to be empirically evaluated.

4.5.6 Handling Drop-offs

Additionally, the AEG-based formulation provides a natural framework for taking into account drop-offs, which may arise as a consequence of pairs leaving the program. For the sake of simplicity, we will initially make the assumption that each pair may drop-off the program between consecutive arrivals with a fixed probability. In other words:

Assumption 3. *Drop-offs for each pair i between arrivals follow a Bernoulli process, with probability r_i.*

A discussion on how to relax the assumption appears at the end of this section.

We can now show how to extend the $\mathrm{CYF_A}$ model to take into account the effect of drop-offs (the same reasoning applies to the abstract $\mathrm{KEP_A}$ formulation). In this case,

it becomes important to understand *when* a given exchange can be performed. Let τ_i be the index of the time step when a node enters the program, i.e. $\tau_i = k$ iff $i \in N^k$. An exchange involving a set of nodes S can be performed only once the last of the involved nodes arrive. We extend our notation so that, for a set $S \subseteq N$, we have $\tau_S = \max\{\tau_i \mid i \in S\}$.

For an exchange to be performed, all involved nodes should arrive *and remain in the program* long enough. Formally, Constraints (4.13) should be rewritten as:

$$\sum_{C_j \in \mathcal{C}, S \subseteq C_j} x_j \leq \prod_{i \in S} r_i^{\tau_S - \tau_i} p_i \qquad \forall S \in \mathcal{S} \tag{4.19}$$

where $r_i^{\tau_S - \tau_i}$ is the probability that node i remains in the program until τ_S.

4.5.7 Transplants versus Survivors

Taking into account drop-off events enables one further extension. In an online setting, the number of transplants is important, but does not take into account that *having a large pool of participants is also a value*: it means more people have survived and may still receive a transplant. We can take this into account by tracking the *expected* number of instances of each node i at time step k via an additional set of variables $q_i^k \geq 0$. Formally we have that:

$$q_i^k = \begin{cases} p_i & \text{if } k = \tau_i \\ r_i(q_i^{k-1} - y_i^k) & \text{if } k > \tau_i \end{cases} \qquad \forall i \in N, \forall k \in \{\tau_i \dots h+1\} \tag{4.20}$$

where y_i^k is the expected number of instances of node i that received a transplant at step k, and there is no need to introduce q_i^k variables for $k < \tau_i$. Equation (4.20) follows directly from Assumption 3. The value of y_i^k is given by:

$$y_i^k = \sum_{\substack{C_j \in \mathcal{C}, i \in C_j, \\ \tau_{C_j} = k}} x_j \qquad \forall i \in N, \forall k \in \{\tau_i \dots h\} \tag{4.21}$$

Incorporating everything in the $\mathrm{CYF_A}$ formulation leads to the following model $(CYF_{A,D})$:

$$\max z = \sum_{i \in N} r_i(q_i^h - y_i^h) + \sum_{C_j \in \mathcal{C}} w_j x_j \qquad (\mathbf{CYF_{A,D}}) \qquad (4.22)$$

$$\text{s.t.} \sum_{C_j \in \mathcal{C}, S \subseteq C_j} x_j \leq \prod_{i \in S} r_i^{\tau_S - \tau_i} p_i \qquad \forall S \in \mathcal{S} \qquad (4.23)$$

$$q_i^{\tau_i} = p_i \qquad \forall i \in N \qquad (4.24)$$

$$q_i^k = r_i(q_i^{k-1} - y_i^k) \qquad \forall i \in N, k \in \{\tau_i + 1 \dots h\} \qquad (4.25)$$

$$y_i^k = \sum_{\substack{C_j \in \mathcal{C}, i \in C_j, \\ \tau_{C_j} = k}} x_j \qquad \forall i \in N, \forall k \in \{\tau_i \dots h\} \qquad (4.26)$$

$$x_j \in \{0, 1\} \qquad \forall C_j \in \mathcal{C}^0 \qquad (4.27)$$

$$x_j \in [0, 1] \qquad \forall C_j \in \mathcal{C}^F \qquad (4.28)$$

$$y_i^k, q_i^k \geq 0 \qquad \forall i \in N, \forall k \in \{\tau_i \dots h\} \qquad (4.29)$$

The goal is to maximize the number of transplants, *plus the expected number of survivors* at the end of the look-ahead horizon h. Performing a transplant is still be more beneficial than waiting, since the q_i^{h+1} values are discounted by at least one factor r_i. Taking into account drop-offs results only in marginal modifications of the original formulation CYF_A, while keeping track of survivors requires more extensive changes. None of the additional variables is subject to integrality constraints, however, which is good for the scalability of the method.

4.5.8 Limitations and Workarounds

Here we discuss some of the limitations of our approach, together with some means for addressing them. At the moment, however, none of these solutions has been experimentally evaluated.

Assumption 1 may be violated if there are additional criteria that differentiate the nodes, e.g. different patient survival probabilities. Violations of this kind are easily accounted for by including such factors in Algorithm 3, when searching for equivalent nodes. The price to pay is an increased number of nodes, which in general may be an issue with AEG-based approaches. A possible mitigation measure would be to use ideas from clustering algorithms to merge in Algorithm 3 nodes that are sufficiently similar, even if they are not exactly equivalent.

A violation of Assumption 2 prevents the definition of a vector p of arrival probabilities. However, our methods still work by replacing the product of probabilities in Constraints (4.13) and (4.23) with a joint arrival probability $P(S)$, i.e:

$$\sum_{C_j \in \mathcal{C}, i \in C_j} x_j \leq \prod_{i \in S} p_i \longleftrightarrow \sum_{C_j \in \mathcal{C}, i \in C_j} x_j \leq P(S) \tag{4.30}$$

Non-stationary drop-off probabilities, i.e. time dependent r_i values, require to rewrite the $r_i^{\tau_S - \tau_i}$ expressions in Constraints (4.23):

$$r_i^{\tau_S - \tau_i} \longleftrightarrow \prod_{k=\tau_i}^{\tau_S} r_i^k \tag{4.31}$$

where r_i^k is the drop-off probability of node i at step k. We focused on maximizing the number of transplants/survivors, but different objective functions may be employed as long as they can be handled in the problem models. In particular, the objectives considered in [Dickerson and Sandholm, 2015] based on fairness and expected organ failures should be manageable without much trouble.

4.6 Minimum Horizon Model

The minimum horizon model approach will purposefully fail to exploit the full accuracy of the AEG for sake of simplicity and computational tractability. The model is not necessarily the most natural one, but it is the easiest to understand. This model targets the online KEP and makes a distinction between the set P of nodes currently in the program (a.k.a. present nodes) and the set F of nodes that may enter the program according to the AEG (a.k.a. future nodes). The model allows for cycles to include up to one node from F. Formally, this means that the model is defined over a graph that includes all nodes, but excludes arcs between pairs of future nodes:

$$G = \langle N \cup F, A(N \cup F),\ A(F) \rangle. \tag{4.32}$$

We will denote as C_P the cycles defined only on present nodes ("present" cycles) and as C_F cycles that include one node from F ("future" cycles). We will denote as C the collection of all cycles, i.e to $C_P \cup C_F$. Each cycle C_j is represented as a sequence of nodes. The model contains one variable x_j per cycle in C. The key insight is that cycles in C_P can be either performed or not performed, and hence are associated to binary variables. Cycles that involve a node in the future, however, represent uncertain choices and are therefore associated with real-valued variables representing the *expected number of times the cycle will be used.*

Note that the expected number of uses for future cycles is a planned decisions, and may

reflect a strategy (besides an uncontrollable outcome). As a consequence, when two cycles both require a node, only one of them can be performed. However, when two cycles both require a future node, we can choose (in probabilistic terms) how we want them to share it, as long as the number of used does not exceed the expected number of arrivals for the node. This is analogous to stochastic policies in Markov Decision Processes, or to stochastic strategies in Game Theoretic approaches.

Then, after a single entry event, each node i from F will be present with its probability p_i, specified by the AEG. If we wait for each h arrivals, classical probability theory (Bernoulli processes) implies that the expected number of arrivals for each future node is given by $p_i h$. These two considerations enable the formulation of a first model:

$$\min z = h \tag{4.33}$$

s.t.

$$\sum_{C_j \in C, i \in C_j} x_j \geq 1, \qquad \forall i \in P \tag{4.34}$$

$$\sum_{C_j \in C_F, i \in C_j} x_j \leq p_i h, \qquad \forall i \in F \tag{4.35}$$

$$x_j \in \{0, 1\}, \qquad \forall i \in P \tag{4.36}$$

$$x_j \geq 0, \qquad \forall i \in F \tag{4.37}$$

Intuitively, we minimise the number of waiting steps h required to be "reasonably sure" that all present nodes receive their transplants. These transplants may come either from present or future cycles. Future cycles cannot exceed the expected number of arrivals for each future node, but this can be arbitrarily increased by waiting for more time steps. Intuitively, the number of waiting steps is roughly correlated to the survival probability of the present node. Much better approximations are of course possible, but as it will become apparent they require more complex models.

The model described in Section 4.5.2 already shows some of the key features of approaches based on the AEG. In particular:

- There is a clear distinction between present/future nodes and present/future cy-

cles;

- Future cycles are associated to real-values (rather than integer) variables, which represent stochastic strategies.

As a consequence, most AEG based models will contain quite a limited number of integer variables, which makes them very appealing from a computational perspective.

4.7 Experiments

In this section we present our experimental evaluation. First we present the experimental setup and the results for the anticipatory algorithms from Section 4.4. Secondly, we continue with the abstract exchange graph from Section 4.5.

4.7.1 Empirical Setup for CSBA

We generated 2 instances, one small and one big. The small instance simulates the pairs arrival to be 5 pairs per month, for over 31 months, that means in total we have 158 pairs of which 11 pairs are altruistic, with 4,086 edges. The big instance simulates the pairs arrival to be 10 pairs per month for over 51 months, which means we have 510 pairs and 25 altruistic pairs, with 15,400 edges. These characteristics were taken from [Awasthi and Sandholm, 2009]. Other characteristics needed are the blood type, the PRA values and the patient and donor age distributions, the percentage of pairs expected to drop out of the pool. We used the following configuration: 51% of the patients were ABO-0, 31% were ABO-A, 13% were ABO-B and 4% were ABO-AB; 47% of the patients had PRA $\leq$ 15% with no defined HLA antibodies; 18% were moderately sensitized, with PRA between 16 and 79%; 36% were highly sensitized with either class I or II PRA $\geq$ 80%. These characteristics are taken from [Saidman et al., 2006] to mimic the real-world population.

We considered all transplants equally worthy, i.e., the weight of each edge in the exchange graph was set to one. In our experiments, the exchange graph changes over time as follows. Initially the exchange starts with no vertices. At each time step a fixed number of vertices enter the graph. Each vertex has a dropout time, which is set according to the reality that 12% of dialysis kidney patients survive 10 years [Dickerson et al., 2012a].

The model that we presented in Section 4.4 is implemented in Python. We use Num-

berjack [Numberjack,] to model the problems using the IBM's CPLEX v12.8 [1] solver. All the tests run on Intel Xeon E5-2640 processors.

We were unable to reproduce the same instances as the one proposed in [Awasthi and Sandholm, 2009], but we've tried to follow the characterisation described as much as possible. One reason to be unable to replicate the results, was, for our understanding, the generation of the scenarios, which are not described in the paper. The second reason is that they've used real instance, which we couldn't find online.

In our experiments, we generated the scenarios as follows: at each time step, for each scenario, we choose a random 4% vertices from the list within lookahead limit, and after we check if these vertices were able to create cycles/chains. Notice that it is possible for some scenarios to not have any cycles/chains. Moreover, as we arrive at the end of the timespan we will not be able to generate scenarios if the lookahead is very long. The realisation randomly choose 6% of the vertices entering each time step.

Each vertex has a time out, corresponding to match the fact that 12% of the dialysis patients survive 10 years, meaning that each month the survival rate is $(1-p)^{120} = 0.12$, where 120 is 12 months $\times$ 10 years . For the small instance, the survival rate will be $(1-p)^{31}$ and for the big instance $(1-p)^{51}$.

4.7.2 Tuning Lookahead and the Number of Scenarios

In the following tables, the instance was run 10 times for each configuration. We are showing the average values along with the standard deviation.

We ran experiments to choose good values for the lookahead and the sampling. The sampling refers to the number of scenarios for each time step, and the lookahead is how many steps into the future the algorithm can see. The size of possible trajectories can be exponential, depending how large the lookahead, so we cannot cover all of them for space reasons. For example in Table 4.1, and Table 4.2, increasing the lookahead too much as well as the sampling does not improve further the number of lives saved.

In Table 4.1 we show the results for the small instance. We notice that our algorithm outperforms the other scenario-based algorithms. In particular, for the configuration $l7\_s5$ our algorithm saves 2.22% more lives than the APST2 and 5.82% more lives than the offline algorithm. We are still 3.48% away from the oracle solution. In Figure

[1] https://www-01.ibm.com/support/docview.wss?uid=swg27050618

Table 4.1: 31 months instance

Lookahead/ Sampling	Our alg	APST2	APST1	Offline	Oracle
l2_s5	50.3 ± 5.84	48.7 ± 6.24	49.5 ± 5.40	44.7± 5.36	57.1 ± 7.38
l2_s10	48.5 ± 4.17	46.5 ± 4.58	49.0 ± 5.09	42.8± 5.79	56.0 ± 5.03
l2_s15	52.8 ± 6.16	49.4 ± 5.29	51.6 ± 5.08	44.5± 3.82	60.1 ± 5.10
l4_s5	49.3 ± 4.10	45.7 ± 4.60	48.1 ± 4.70	43.3± 5.56	54.7 ± 6.00
l4_s10	52.7 ± 5.02	49.1 ± 5.00	50.7 ± 4.79	42.7± 5.83	57.5 ± 3.95
l4_s15	55.5 ± 8.62	**52.3 ± 8.24**	**54.7 ± 8.61**	**47.4± 8.33**	61.1 ± 9.82
l7_s5	**56.4 ± 5.44**	51.7 ± 4.56	52.9 ± 5.31	47.2± 2.99	**61.9 ± 5.90**
l7_s10	54.4 ± 8.94	49.6 ± 7.72	51.4 ± 7.56	45.2± 5.19	59.8 ± 9.07
l7_s15	53.3 ± 7.82	49.3 ± 6.98	50.6 ± 6.97	43.7± 5.44	57.9 ± 8.01

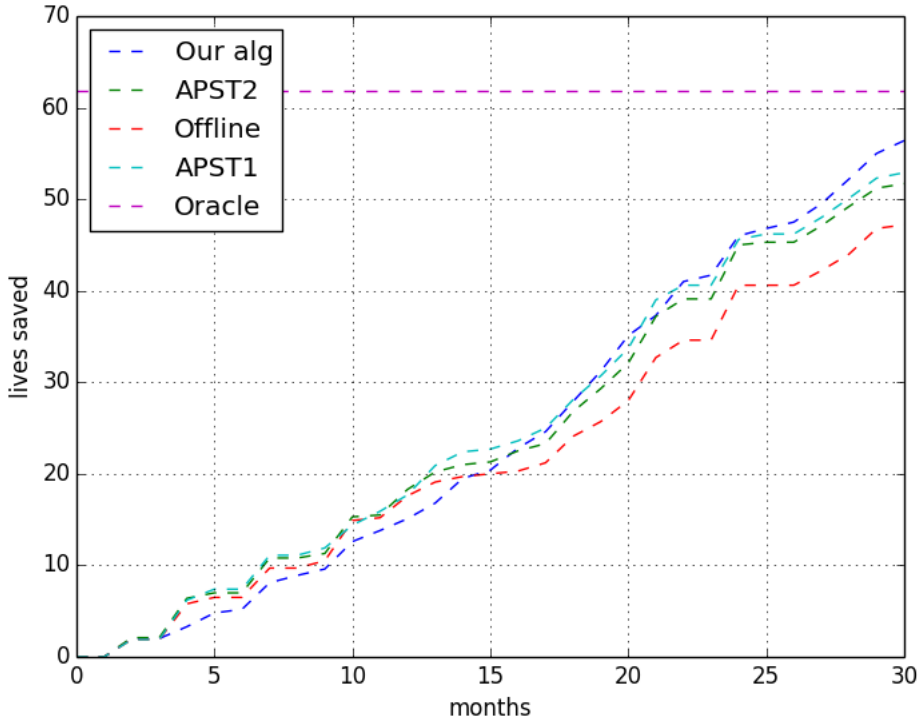


Figure 4.5: Trend of the algorithms for 31 months instance

4.5, we show the evolution of the algorithms and how many lives are saved step-by-step for each algorithm. In particular, we observe for our algorithm, after Month 15 the slope is increasing significantly and surpasses the other algorithms. In Table 4.2 we expose the results of the big instance, 51 months with 510 pairs of which 25 are altruistic donors, and each month 10 new patients enter the program. We notice in configuration $l20\_s20$ that our algorithm saves 3.70% more lives than the APST2, and 2.1% more lives than APST1, and 7.50% more lives than the offline algorithm. The

Oracle saves 3.92% more lives, since it contains the perfect information of the future. Even if the percentages look quit small, there are not that small considering these are human lives.

Table 4.2: 51 months instance

Lookahead/ Sampling	Our alg	APST2	APST1	Offline	Oracle
l5_s10	142.2 ± 10.43	133.8 ± 11.12	140.2 ± 12.89	117.6 ± 9.00	172.5 ± 12.53
l5_s20	151.6 ± 10.79	**144.6 ± 9.39**	149.2 ± 11.91	**125.5 ± 10.36**	**186.3 ± 12.87**
l5_s50	151.1 ± 12.16	142.8 ± 9.48	**150.1 ± 12.61**	123.1 ± 7.67	180.4 ± 12.43
l10_s10	149.5 ± 11.79	137.7 ± 8.48	146.3 ± 8.48	122.1 ± 8.53	177.5 ± 11.30
l10_s20	152.8 ± 10.54	138.2 ± 8.84	143.2 ± 7.50	119.2 ± 8.55	182.4 ± 11.77
l10_s50	158.4 ± 11.56	139.8 ± 7.13	148.0 ± 10.26	123.9 ± 9.62	184.6 ± 11.58
l20_s10	154.9 ± 8.63	139.1 ± 7.87	146.9 ± 7.36	124.2 ± 7.08	172.6 ± 8.53
l20_s20	**160.6 ± 9.23**	141.7 ± 9.66	149.7 ± 10.62	122.3 ± 5.34	180.6 ± 11.76
l20_s50	-	-	-	-	-

In Figure 4.6, we show the evolution of the algorithms for the big instance, step by step. In this case, we observe, that our algorithm delays some decisions at the beginning, and after Month 32 the lives saved increase.

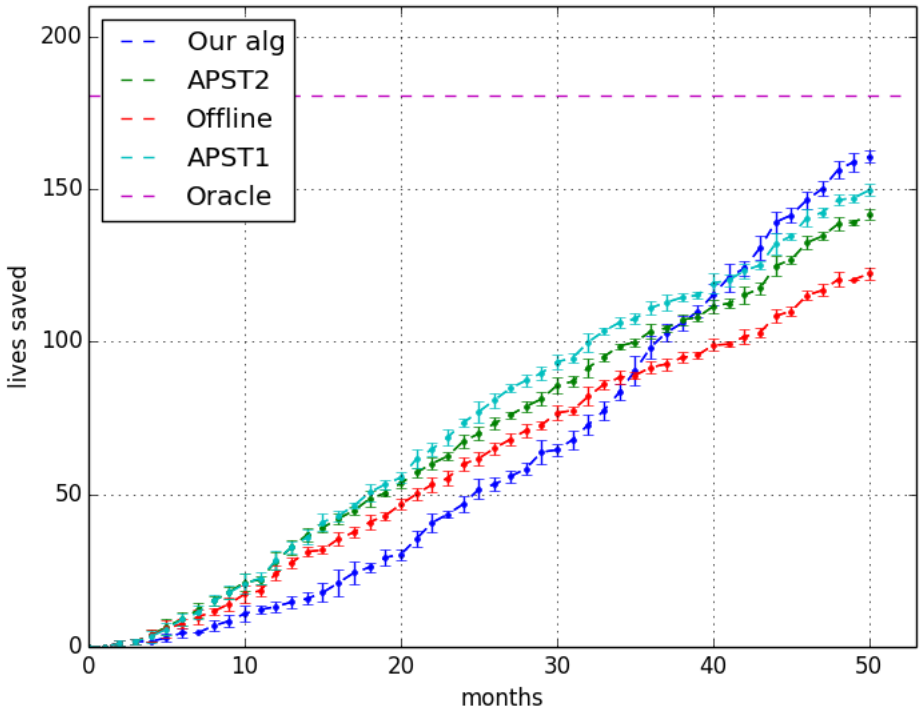


Figure 4.6: Trend of the algorithms 51 months instance

4.7.3 Tuning the Batch Size

In real life, the data that we have to deal with can be very large and the horizon of events can be very long. We introduce the notion of batch size, as a parameter that defines how many patients the algorithm can consider as an atomic event. Normally, the batch size is used to the establish how often the algorithm is executed. In our case for the small instance, we have five vertices entering per month and the batch size is five, so the algorithm runs every month. The death rate is adjusted to match the real-world event that 12% of dialysis patients survive 10 years.

Table 4.3: Batch configuration 31 months instance with fix drops out

Batch	Our alg	APST2	APST1	Offline	Oracle
3	27.1 ± 4.70	25.2 ± 4.64	26.1 ± 4.61	19.1 ± 4.30	29.9 ± 5.02
5	$\mathbf{56.4 \pm 5.33}$	$\mathbf{51.7 \pm 4.56}$	$\mathbf{52.9 \pm 5.26}$	$\mathbf{47.2 \pm 2.99}$	$\mathbf{61.9 \pm 5.78}$
10	53.6 ± 5.85	48.0 ± 3.40	50.2 ± 4.64	44.3 ± 3.68	58.4 ± 5.98
15	50.6 ± 4.10	46.9 ± 5.55	51.1 ± 3.70	41.5 ± 4.05	53.6 ± 3.44
17	52.9 ± 3.23	49.7 ± 3.57	52.1 ± 4.34	43.9 ± 3.50	56.0 ± 3.40
20	49.2 ± 9.63	46.7 ± 7.59	48.8 ± 8.35	42.7 ± 6.70	51.1 ± 9.75

We ran experiments to determine the appropriate batch size for both instances. Increasing too much the batch size does not help saving more lives, as we can observe in Table 4.3 and 4.4.

Table 4.4: Batch configuration for 51 months instance

Batch	Our alg	APST2	APST1	Offline	Oracle
5	155.9 ± 7.35	144.9 ± 8.10	150.6 ± 9.40	125.7 ± 7.10	185.0 ± 8.42
10	160.6 ± 9.23	141.7 ± 9.66	149.7 ± 10.62	122.3 ± 5.34	180.6 ± 11.76
12	146.9 ± 5.97	130.8 ± 6.66	135.5 ± 6.29	112.6 ± 6.94	165.1 ± 6.29
18	145.6 ± 9.18	$127.4 \pm 14.$	130.6 ± 13.73	108.3 ± 7.12	156.3 ± 10.08

4.7.4 Empiricial Evaluation for the AEG

From a mathematical standpoint the approach described in Section 4.5 is closest to the algorithms from [Awasthi and Sandholm, 2009] and the model described in Section 4.4, which we therefore chose for our comparison, together with a myopic approach and an oracle (used respectively as baseline and optimistic bound).

Conversely, our method is more distantly related to the ones from [Dickerson et al., 2012a, Dickerson and Sandholm, 2015], despite being also sampling-free and with the same practical use cases. Such methods rely on adjusting (e.g. via parameter tuning) the

weights of an off-line KEP to take into account the impact of current decisions on the future. The AEG is arguably more accurate from a formal point of view, and does not require a computationally expensive fitting step. However, the approach from [Dickerson and Sandholm, 2015] can use an off-line KEP formulation with virtually no modification, and provides a bit more flexibility in terms of the supported problem objectives. An empirical comparison of the two approaches is planned as part of future research.

All the considered algorithms treat a scenario as a set of nodes that may enter in the next h steps (no drop-offs are considered the scenarios). The two main algorithms from [Awasthi and Sandholm, 2009] estimate the expected impact of choosing a cycle, and then solve a KEP using such impacts as weights. The first is referred to as APST1, it can be considered an adaptation of the REGRETS method from [Van Hentenryck and Bent, 2009], and computes cycle scores by solving an offline KEP *for each scenario*. The second is referred to as APST2, it shares shares ideas with the EXPECTATION method from [Van Hentenryck and Bent, 2009], and computes scores by solving an offline KEP *for each cycle and scenario*. For further details, the reader may refer to [Awasthi and Sandholm, 2009] or [Chisca et al., 2018].

The CSBA algorithm described in Chapter 4 solves a single modified KEP on a graph constructed using all scenarios, obtained by using the Sample Average Approximation [Pagnoncelli et al., 2009]. The final matching is given by all exchanges in the solution that are defined solely over nodes from the current time step (i.e. that do not include nodes entering in any scenario). In the objective, exchanges related to the current time step are summed exactly, whereas future exchanges are considered in expectation.

4.7.5 Methods and Instances

Some anticipatory algorithms for the KEP may occasionally suggest to delay an transplant, if they estimate that such actions is going to be beneficial in the long term. This may happen even if the delayed exchange could be performed in the current time interval without conflicting with other selected exchanges. For a human decision maker, this kind of behavior is hard to justify, especially when personal health is at stake.

Luckily, in the cycle formulation such a behavior can be prevented or at least discouraged via a simple pre-processing technique. Namely, we can remove from the graph all cycles that include no node currently in the program: as a result, delaying an exchange that can be performed immediately becomes far less likely. In our experimentation, we have applied this technique to all methods that may benefit from that, in particular the

AEG-based approach and the CSBA algorithm. Neither APST1 nor APST2 have such need since, they eventually solve a KEP including only the current nodes.

4.7.5.1 Instances

We've generated two main instances using real-world probabilities: each instance is a (concrete) exchange graph which represents a population. At each time step (corresponding to one month), arrival events are modeled by sampling from the graph a number of nodes (referred to as batch size). Unlike in other works reported in the literature about the online KEP, pairs are sampled *with reinsertion*: this means that the graphs represent stable or slowly varying populations, and captures much better what happens in the real world compared to sampling without reinsertion. Our scenarios are sampled from the same graphs.

Specifically, we generated a small instance with 160 pairs (plus 11 altruistic donors), used for experiments running over 31 months with batch size 5, and a larger one with 1029 pairs (plus 29 altruistic donors), for experiments over 12 months with batch size 15 and 20. For each setup, the generated pairs have blood types, PRA values, and ages following the distributions from [Saidman et al., 2006]. Tissue compatibility has been approximated by suppressing a fraction of the arcs, chosen uniformly at random according to statistics reported in the same paper: this process disregards patterns that may arise in the real world and may therefore lead to larger number of types. The AEG graph was generated as described in Section 4.5.2: in our settings, we detected 22 node types for the small instance and 26 for the big instance, i.e. very small numbers compared to the size of the original graphs. The death rate is adjusted to match the reality that 12% of kidney patients survive 10 years [2].

We implemented all the algorithms in Python, using Numberjack [Numberjack,] as a modeling front-end and CPLEX as a back-end. As we mentioned, we also include in our comparison an oracle that solves a single offline KEP including all pairs entering the program (and disregarding drop-off dates), which provides an optimistic bound on the performance of any online algorithm. We consider all transplants equally worthy.

[2]United States Renal Data System (USRDS), 2007: http://www.usrds.org/

Table 4.5: 31-months setup sample algorithms (batch 5)

Lookahead/ Samples	Pairs	Altr.	CSBA		APST2		APST1	
			Lives	WaitL	Lives	WaitL	Lives	WaitL
h2_s5	149	11	61.0±4.08	64.5	60.6±4.32	64.7	62.0±3.52	63.4
h2_s10	149	11	60.6±4.59	65.1	**60.7±4.63**	64.5	61.6±4.29	63.8
h2_s15	149	11	60.6±4.50	65.0	60.5±4.70	64.6	62.2±4.39	63.2
h3_s5	149	11	61.2±4.46	64.3	60.5±4.45	64.8	62.1±3.84	63.2
h3_s10	149	11	60.8±4.36	65.0	60.7±4.73	64.5	62.8±3.82	62.4
h3_s15	149	11	60.3±4.52	65.4	60.3±4.70	64.7	61.7±3.91	63.5
h4_s5	149	11	60.7±4.44	64.7	60.5±4.45	64.8	62.2±3.70	62.8
h4_s10	149	11	**61.1±4.35**	64.5	60.7±4.73	64.6	62.0±3.52	63.1
h4_s15	149	11	60.8±4.75	64.8	60.6±4.74	64.6	**63.2±4.08**	62.2

Table 4.6: 31-months setup AEG (batch 5)

Lookahead	Pairs	Altr.	AEG		Myopic		Oracle
			Lives	WaitL	Lives	WaitL	Lives
h2	149	11	**76.1 ± 3.87**	83.9	53.5 ±2.75	66.5	88.3 ±7.46
h3	149	11	75.0 ± 3.43	85.0	53.5 ±2.75	66.5	88.3 ±7.46
h4	149	11	75.6 ± 3.43	84.4	53.5 ±2.75	66.5	88.3 ±7.46

4.7.6 Results

We report the results for the sampling-based algorithms (i.e. CSBA, APST1, and APST2) in one set of tables, while the results for the (sampling-free) AEG, myopic, and oracle methods are in a second set of tables. Every table row corresponds to a different algorithm and each row to a different configuration. Morever

In particular, Table 4.5, 4.7, and 4.9 show the results of the scenario sampling algorithms: each row is labeled as $\{hx_1\_sx_2\}$, where $x_1 \in \{2, 3, 4\}$ is the look-ahead horizon and $x_2 \in \{5, 10, 15\}$ is the number of scenarios. Table 4.6, 4.8, and 4.10 show instead the results of the sampling-free algorithms. Each row is labeled in this case as $\{hx_1\}$, where $x_1 = \{2, 3, 4\}$ is the look-ahead horizon. Each algorithm, except for the oracle, has two columns: one called “Lives" for the number of transplants, and one called “WaitL" and for the number of pairs still in the waiting list. Each cell reports an average over 10 runs, and for the “Lives" column we also report the standard deviation. The solver timelimit is set to 600 seconds for each run, therefore for each time step each algorithm has 600/Months setup overall. If an algorithm requires to solve multiple KEPs (say k) in a single time step, then the timelimit for each attempt is set

to 600/(Months * k) seconds. A “dash" in a cell means that the algorithm was not able to find a solution for all time steps.

Table 4.7: 12-months setup sample algorithms (batch 15)

Lookahead/ Samples	Pairs	Altr.	CSBA		APST2		APST1	
			Lives	WaitL	Lives	WaitL	Lives	WaitL
h2_s5	171	9	82.8±6.00	67.8	82.8±7.11	67.8	82.4±7.29	67.1
h2_s10	171	9	**83.2±6.20**	68.0	**83.5±7.06**	66.9	**84.6±7.92**	66.0
h2_s15	171	9	83.0±6.68	67.8	-	-	86.0±7.51	65.0
h3_s5	171	9	82.9±6.59	68.1	82.5± 7.06	65.9	82.7±8.21	67.4
h3_s10	171	9	82.3±6.20	68.3	-	-	83.3±7.07	67.5
h3_s15	171	9	82.6±7.65	67.6	-	-	84.0±6.31	67.5
h4_s5	171	9	82.8±6.66	67.9	-	-	79.7±8.64	70.3
h4_s10	171	9	82.4±6.17	68.3	-	-	83.7±4.65	68.5
h4_s15	171	9	82.6±7.75	68.0	-	-	81.6±8.24	69.1

Table 4.8: 12-months setup AEG (batch 15)

Lookahead	Pairs	Altr.	AEG		Myopic		Oracle
			Lives	WaitL	Lives	WaitL	Lives
h2	171	9	**94.8 ± 8.59**	85.2	67.9 ± 5.49	71.0	113.4± 2.67
h3	171	9	94.5 ± 8.34	85.5	67.9 ± 5.49	71.0	113.4± 2.67
h4	171	9	94.8 ± 8.78	85.2	67.9 ± 5.49	71.0	113.4± 2.67

Table 4.9: 12-months setup sample algorithms (batch 20)

Lookahead/Samples	Pairs	Altr.	CSBA		APST1	
			Lives	WaitL	Lives	WaitL
h2_s5	228	12	106.7 ± 9.68	92.6	107.2 ± 8.83	91.9
h2_s10	228	12	106.8 ± 9.43	92.9	108.2 ± 7.98	90.6
h2_s15	228	12	107.6 ± 10.16	92.1	**108.4 ± 8.80**	90.2
h3_s5	228	12	106.8 ± 9.90	92.4	103.9 ± 8.46	94.0
h3_s10	228	12	**107.7 ± 10.77**	91.7	106.7 ± 9.21	91.4
h3_s15	228	12	106.7 ± 10.24	93.0	-	-
h4_s5	228	12	106.3 ± 10.04	92.6	103.8±9.27	93.3
h4_s10	228	12	107.4 ± 8.77	92.6	103.83±7.41	95
h4_s15	228	12	106.0 ± 8.91	93.1	-	-

Table 4.10: 12-months setup AEG (batch 20)

Lookahead	Pairs	Altr.	AEG		Myopic		Oracle
			Lives	WaitL	Lives	WaitL	Lives
h2	228	12	121.6 ± 9.29	118.4	92.5 ± 8.05	92.3	137.4 ± 13.12
h3	228	12	**122.2 ± 9.85**	117.8	92.5 ± 8.05	92.3	137.4 ± 13.12
h4	228	12	121.8 ± 9.41	118.2	92.5 ± 8.05	92.3	137.4 ± 13.12

4.7.6.1 Effect of the Batch Size

In Tables 4.5 and 4.7, 4.9, we show the results of a long run setup with small batch (5) size vs a short run setup with larger batch size (15, 20). As it can be seen in Table 4.5, $APST1$ outperforms the other scenario-sampling algorithms, but increasing the batch size to 20 is enough for the method to hit the time limit, due to the large number of KEPs that the algorithms needs to solve. The $APST2$ method runs out of time even with batch size 10, since it needs to loop over all scenarios and also all cycles; the algorithm did not scale for batch size 20. Both the CSBA and AEG algorithms manage to complete all experiments within the time limit.

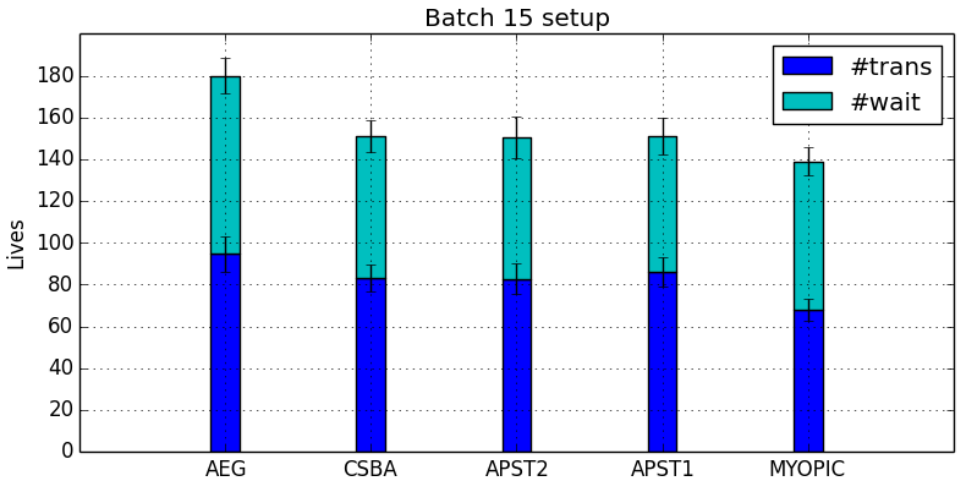


Figure 4.7: Comparison of #transplants and #waiting list of the algorithms 15 batch setup.

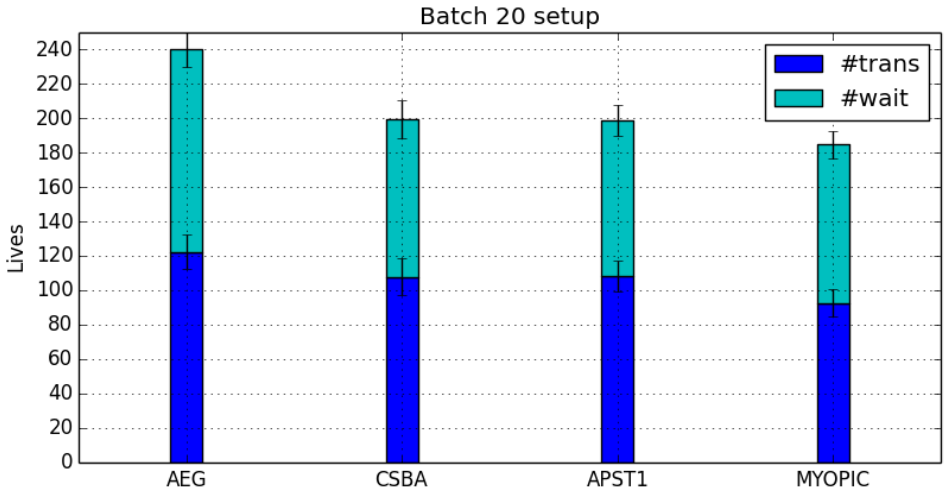


Figure 4.8: Comparison of #transplants and #waiting list of the algorithms 20 batch setup.

4.7.6.2 Effect of Look-ahead and Number of Scenarios

Increasing the look-ahead and the number of scenarios does not have a consistent effect on the algorithms, which reach the best results for different configurations (in bold in the tables). We notice that the $CSBA$ algorithm does not outperform the other scenario-based algorithms, probably due to the fact that we are sampling pairs with reinsertion.

4.7.6.3 Trends

The AEG model improves the number of transplants over the myopic algorithm by a factor 26.9 % in Table 4.7 and by a factor 29.7% in Table 4.9. These results are in-line with the improvements obtained by the sampling-free method from [Dickerson and Sandholm, 2015], and interestingly they are reached *even if the AEG method in fact optimise the number of survivors*.

In Figure 4.7 and Figure 4.8 we compare the results for the best configuration: our method consistently (and significantly) outperforms all sampling-based algorithms, in terms of both the number of transplants and that of survivors (transplants plus waiting list). Figure 4.9 and Figure 4.10 show the evolution of the cumulative (average) number of transplants for each algorithm, which is initially very similar for all methods, until month 3-4 where the AEG models start to outpace the competitors.

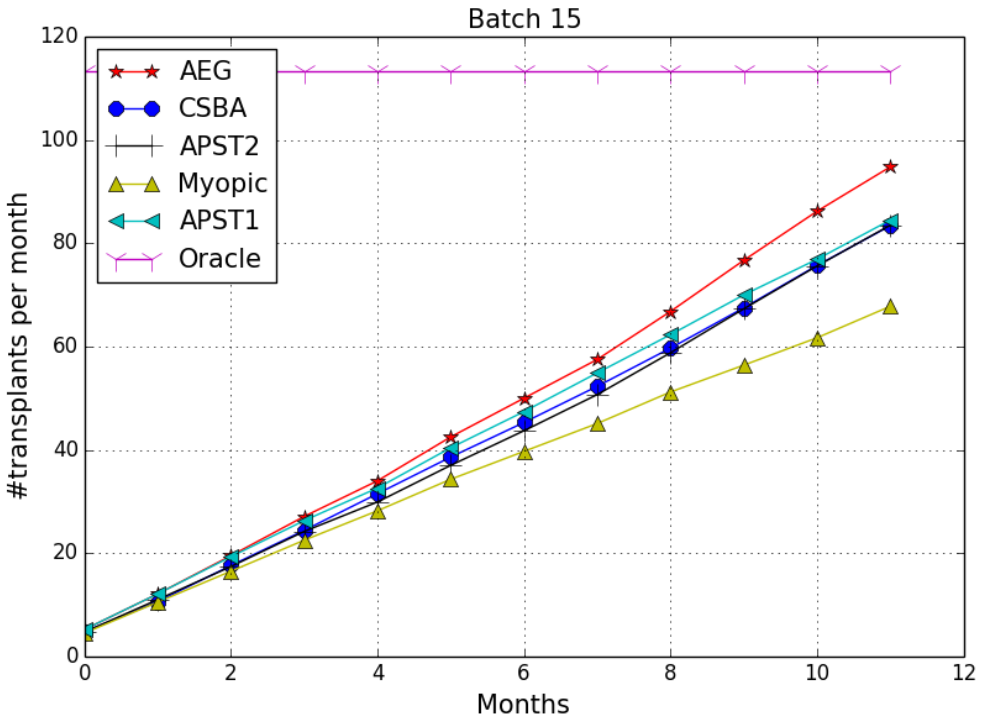


Figure 4.9: Trend of the algorithms batch 15 setup.

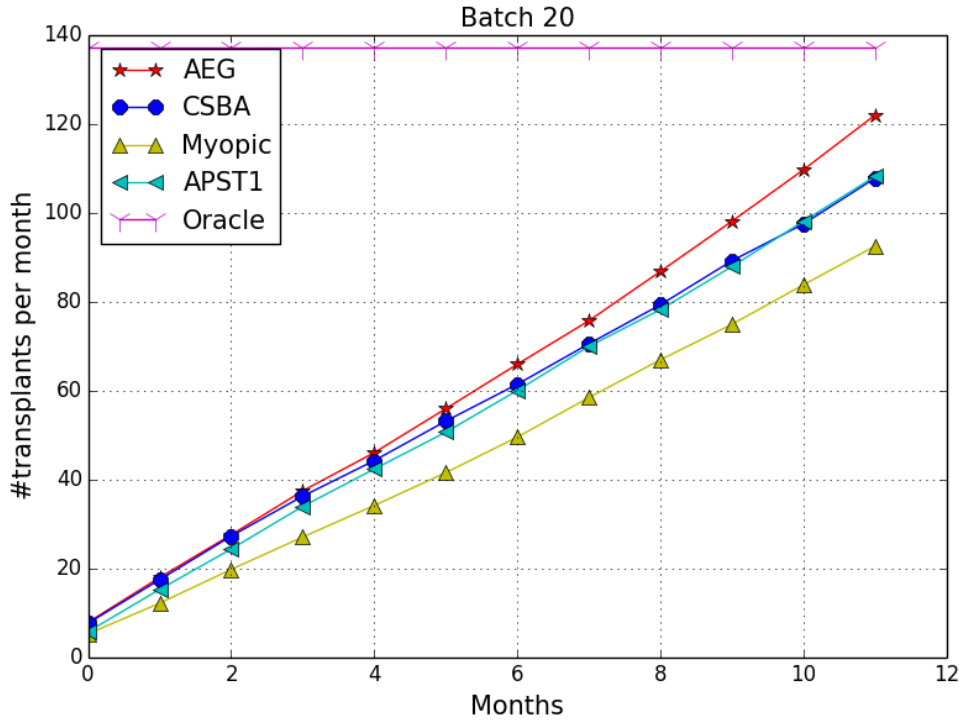


Figure 4.10: Trend of the algorithms batch 20 setup.

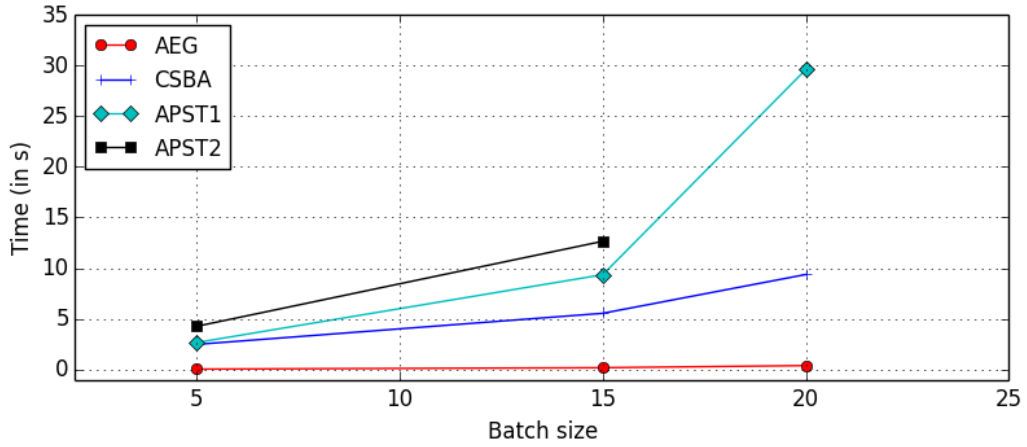


Figure 4.11: Times of the algorithms

4.7.6.4 Solution Times

In Figure 4.11 we show the solution times of the anticipatory algorithms for the best configuration of 5, 10 and 15 batch size. Each value is measured in seconds and represents the average of 10 runs. The $APST2$ algorithm has only 2 points since it does not scale further than 10 batch size, and the solution time for algorithm $APST1$ grows very fast with the batch size. Meanwhile the AEG model remains very scalable even increasing the batch size.

4.8 Chapter Summary

We showed how a correct estimation of the expected value for the online KEP can save significantly more lives. Moreover, we show how Sample Average Approximation can improve the solutions. We also notice that by delaying some decisions we obtain good results.

Informally speaking, our main contribution is an anticipatory model for the KEP based on probabilistic, rather than statistical, considerations, and therefore to forgo sampling. Additionally, our AEG-based technique allows us to take into account the effect of drop-offs. When grounded on the cycle formulation, our method does not require the introduction of additional *integer* variables. Overall, the resulting approach manages to outperform significantly both sampling-based anticipatory algorithms and a myopic approach, while having a scalability similar to the latter.

Plans for future research include an experimental comparison with other sampling-free methods, and tackling large-scale, real world, instances. This will likely require the use of column generation, which intuitively could prove particularly effective, given that most of our problem variables are not subject to integrality constraints.

Chapter 5

Super Solutions Made Easy: Robust Kidney Exchange Programs

"The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge."
Stephen Hawking

5.1 Introduction

When optimising under uncertainty, it is desirable that solutions are robust to unexpected disruptions and changes. A possible formalization of robustness is given by super solutions. In this chapter we propose a method exploiting Logic Based Benders Decomposition to find super solutions to an optimization problem, for generic disruptions. The master deals with the original problem, while subproblems try to find repair solutions for each possible disruption.

Dealing with uncertainty in optimization is increasingly recognized as a key necessity for tackling real-world problems [Powell, 2016]. Thanks to improved computational power and to advances in the solution techniques, approaches for stochastic optimization are becoming more viable and effective. Most of these methods rely on some form of probabilistic or statistical model, which requires reliable knowledge from the domain experts or substantial amounts of data.

Sometimes, however, such probabilistic information is not available or cannot be used (e.g. due to fairness concerns), or it is desirable to be resilient against *any* of a given set

of disruptions. In such cases, super solutions provide a fully combinatorial approach to robust decision making [Hebrard et al., 2004a, Holland and O'Sullivan, 2005b]. An assignment x of decision variables is a super solution if any disruption causing a change in a bounded number of variables can be countered by changing those and a bounded number of other variables. One such example is given by kidney exchange programs which provide broader access to transplants, by allowing incompatible patient-donor pairs to exchange donors. Centralised kidney exchange programs exist in many countries, including the US, the Netherlands and the UK [Manlove and O'Malley, 2014], and require that an optimisation problem is solved at regular intervals (the Kidney Exchange Problem – KEP) to find the best matching. Planned transplants may fail for a number of reasons, making robust solutions for the KEP highly desirable. Ideally, for any transplant lost, one wishes to have some ready alternative, making super solutions particularly appealing in this context.

Finding super solutions is challenging, and in the literature it is done either via reformulated models [Weigel et al., 1998] or via specialized Constraint Programming algorithms [Hebrard et al., 2004b, Holland and O'Sullivan, 2005b]. Unfortunately, the former approach often has poor scalability, while the latter requires low-level modifications to constraint solvers, and is therefore difficult to implement and maintain. These factors have historically limited the practical applicability of the approach.

In this chapter, we introduce a new method to build super solutions for constraint satisfaction or optimization problems, inspired by Logic-Based Benders Decomposition (LBBD). The approach broadens the scope of super solutions to include disruptions in a wide range of forms, it is more scalable than a reformulation, and easier to manage than specialized algorithms. The method is also an anytime algorithm, meaning that robust solutions for *some* disruptions are available in case of an early stop, and it can run in parallel. We use the KEP as a case study, and compare our methods against a reformulation-based approach. The chapter is structured as follows: in Section 5.2 we cover the necessary background and briefly survey related work, in Section 5.3 we describe our method, and in Section 5.4 we ground it on the KEP (and discuss an alternative, reformulation-based, model). We provide experimental results in Section 5.6 and concluding remarks in Section 5.7.

5.2 Background and Related Work

In this section we recall the main definitions and approaches related to super solutions, and Logic-Based Benders Decomposition (LBBD).

5.2.1 Online Optimisation

Online anticipatory algorithms combines online algorithms (from computer science) and stochastic programming (from operations research), and assume the availability of a distribution of future events or an approximation thereof. They take decisions during operations by solving deterministic optimisation problems that represent possible realisations of the future. By exploiting insights into the problem structure, online anticipatory algorithms address the time-critical nature of decisions, which allows for only a few optimisations at decision time or between decisions.

In the online kidney exchange problem, new pairs can appear and existing ones can expire at each time step. The objective is to choose a collection of cycles with maximum weight at each time step. Let $\mathcal{D}_t$ denote the graph at time t, and let P be the distribution according to which the graph changes over time. Let $\mathcal{C}(\mathcal{D})$ be the set of all cycles in $\mathcal{D}$ no longer than L. We introduce a $0/1$ variable x_c for each cycle $c \in \mathcal{C}(\mathcal{D})$. Let $\mathcal{C} = \bigcup_{t=0}^{T} \mathcal{C}(\mathcal{D}_t)$ and $V = \bigcup_{t=0}^{T} V_t$. Let $\mathcal{S}$ is the set of scenarios. The online problem is

$$\max_{c \in \mathcal{C}(\mathcal{D}_0)} E_P \quad [... \max_{c \in \mathcal{C}(\mathcal{D}_T)} \sum w_c x_c] \tag{5.1}$$

$$\sum_{\mathcal{C}_j \in \mathcal{C}: i \in \mathcal{C}_j} x_j \leq 1, \qquad \forall i \in V \tag{5.2}$$

$$x_c \in \{0, 1\} \qquad \forall c \in \mathcal{C} \tag{5.3}$$

We will show how online anticipatory algorithms can be used in the kidney exchange problem and to demonstrate their benefits.

Most of the work in kidney exchange has focused on prefixed run optimization on a static pool. However, this does not reflect the reality, as candidates arrive and depart from the pool. Gershkov and Moldovanu [Gershkov and Moldovanu, 2009] and Said [Said, 2012] introduce optimal dynamic mechanisms when agents arrive over time under Poisson processes in different environments under different objectives.

In [Ünver, 2010] the author deducted efficient dynamic methods for barter exchanges such that the total exchange surplus is maximised. However, the model does not accurately reflect real-world kidney exchange as the results are based on the assumption that only blood type compatibility is considered, ignoring the importance of the tissue type compatibility.

In [Dickerson et al., 2012a] the authors presented a method that learns potentials of elements, i.e., of vertices or edges in a offline graph, and then use these potentials to guide myopic matching. The potential represents an estimate of how much that element can contribute to the objective in the future. In [Dickerson and Sandholm, 2015] the authors present a "FUTURE MATCH" framework, which takes as input from human experts an overarching objective, and automatically learns a matching strategy to achieve its goal. Even in this setting, the authors use the cycle/edge "potentials".

5.2.2 Super Solutions

The concept of super solution was introduced by [Hebrard et al., 2004b] in Constraint Programming building upon earlier work on super models in SAT [Ginsberg et al., 1998]. Formally, an assignment of the decision variables x is an (a, b) super solution if, in case of a loss of value for any set of at most a variables, it is possible to construct an alternative solution by changing those a variables and at most b other variables. More flexibility can be added (as shown in [Hebrard et al., 2004b]) by specifying which variables are subject to breaks (the so called *break-set*), and which values and variables can be used for the repair.

The idea was generalised in [Holland and O'Sullivan, 2005b] to (a, b, c) super solutions. The c parameter constrains the cost of the robust and repair solutions to be at most c units (or equivalently a factor c) from the cost c^* of an optimal, non-robust solution. After c^* is known, finding an (a, b, c) super solution is equivalent to finding an (a, b) super solution for the original problem with the cost bounding constraint. Also in [Holland and O'Sullivan, 2005b], they describe the (α, β)-weighted super solutions, that uses probabilistic failures to determine the sets of variables that require repairs, as well as the costs of establishing such repairs.

Super solutions are described in [Hebrard et al., 2004b, Hebrard et al., 2004a, Holland and O'Sullivan, 2005b, Holland and O'Sullivan, 2004] via specialised Constraint Programming algorithms. Those employ local consistency as a sufficient condition to check robustness, and to prune a branch of the search tree when such a check fails. When all variables are assigned, the check turns also into a necessary condition,

thus ensuring soundness.

In [Roy, 2006] the authors study the complexity of finding σ-models looking at the theoretical complexity of tractable instances of satisfiability (SAT) . There are other papers on fault tolerant solutions using sat encoding such as [Bofill et al., 2013] improves the work of [Ginsberg et al., 1998] using weighted Partial MaxSAT formulation. Their approach strengthens the complexity to $\mathcal{O}(n^a)$ for each handled variable change, where n is the number of variables, and a is the number of breaks, instead of $\mathcal{O}(n^{a+b})$ of [Ginsberg et al., 1998], where b is the number of repairs. However, the method we will present is more general and can deal with non-binary variables, or disruptions that may affect multiple variables at the same time. In this chapter we use based line formulation that has the same asymptotic space complexity, i.e. $\mathcal{O}(n^a)$ for handling each variable change. Robust solutions for combinatorial auctions are studied in [Holland and O'Sullivan, 2005a, Holland and O'Sullivan, 2007]. The work of [Demirović et al., 2018] introduces recoverable team formation (RTF) in the coalition context.

5.2.3 Logic-Based Benders Decomposition (LBBD)

Our method relies on LBBD [Hooker and Ottosson, 2003], an iterative method that breaks down a decision making task into a *master problem* (MP) and a *subproblem* (SP), in charge of different subsets of variables. At each iteration, the solution produced by the master is fed to the subproblem, which tries to extend it into a full assignment. If this is not feasible, a procedure generates one or more constraints (*cuts*) to be added in the master formulation. These cuts invalidate a set of master solutions, always including the current one. If the problem cost does not depend on the subproblem variables, then once all subproblems are feasible, the current master solution is also optimal. In the opposite case, cuts should be generated even for feasible subproblems, until the master becomes infeasible.

5.3 Super Solutions via Benders Decomposition

In this chapter, we define a method to obtain super solutions for problems in the generic form, i.e., the target problem (MP):

$$\min z = f(x) \qquad \textbf{(MP)} \tag{5.4}$$

$$\text{s.t. } x \in X \tag{5.5}$$

where x is a vector of n decision variables, $f(x)$ is a cost function, and the X set corresponds to the feasible decision space, which can be expressed by any means (constraints, equations, logical clauses, etc.).

5.3.1 Generic Disruptions

Our method supports and it is best described in terms of generic disruptions. These are characterized via a triplet $\langle D, E, P \rangle$, where $D(\hat{x})$ denotes the set of all possible disruptions for a given solution $\hat{x}$, by associating each of them with an index d. For example, in classical $(1, b)$ super solutions:

$$D(\hat{x}) = \{1, \ldots, n\}. \tag{5.6}$$

In other words, in this case there is one potential disruption per variable. In general, *a disruption is always uniquely identified by a* $(d, \hat{x})$ *pair.* P is a set of predicates $P_{d,\hat{x}}(x)$, each corresponding to the preconditions of disruption $(d, \hat{x})$. For example, in $(1, b)$ super solutions:

$$P_{d,\hat{x}}(x) = [\![x_d = \hat{x}_d]\!] \tag{5.7}$$

where $[\![\cdot]\!]$ denotes the truth value of a predicate. Informally, the precondition represents a condition that should hold for disruption $(d, \hat{x})$ to happen. E is a set of predicates $E_{d,\hat{x}}(x)$, each corresponding to the effects of disruption $(d, \hat{x})$. For example, in $(1, b)$ super solutions:

$$E_{d,\hat{x}}(x) = [\![x_d \neq \hat{x}_d]\!]\,. \tag{5.8}$$

As another example, often only a particular set of variables in the solution may be subject to change and these are said to be members of the break-set. For each variable in the break-set, a repair-set is required that comprises the set of variables whose values may change to provide another solution. For this case we can write Equations (5.6),

(5.7), and (5.8) as follows:

$$D(\hat{x}) = \mathcal{B} \tag{5.9}$$

$$P_{d,\hat{x}}(x) = [\![x_d = \hat{x}_d]\!] \tag{5.10}$$

$$E_{d,\hat{x}}(x) = [\![x_d \neq \hat{x}_d]\!] \wedge \bigwedge_{i \notin \mathcal{R}_d} [\![x_i = \hat{x}_i]\!] \tag{5.11}$$

where $\mathcal{B}$ is the break-set and $\mathcal{R}_d$ is the repair set associated to the disruption. The precondition predicates are as in the previous case. Concerning the effect predicates, a disruption of the variables of the break-set prevents the variable x_d to take the value it had in the $\hat{x}$ solution; moreover, since the disruption can be handled by changing only the variables in the repair set, all other variables are forced to maintain their values.

The framework can handle more general cases, and in particular disruptions with domain-specific preconditions and domain-specific effects. In general, we have the following:

Remark 5.1. *If $P_{d,\hat{x}}(x)$ holds for solution x, then $E_{d,\hat{x}}(x)$ holds in any repair solution for disruption $(d, \hat{x})$, where d denotes a single possible disruption.*

The result holds by construction, since as long as the disruption identified by the pair $(d, \hat{x})$ applies, all of its repair solutions should take its effects into account. As in all super solution approaches, we assume that repairs can change at most b variables. The c parameter from (a, b, c) super solutions can be supported as described in Section 5.2.

Intuitively, any variable is subject to change, disruption d prevents the d-th variable from taking the value it had in the $\hat{x}$ solution, and its precondition is that variable x_d actually takes such value.

For $(a, \cdot)$ super solutions, let $\{I\}_d$ be the succession of all subsets of up to a variables indices. Then:

$$D(x) = [1..|I|] \tag{5.12}$$

$$E_{d,\hat{x}}(x) = \left[\!\!\left[\bigwedge_{i \in I_d} x_i \neq \hat{x}_i \right]\!\!\right] \tag{5.13}$$

$$P_d(x) = \top \tag{5.14}$$

where $|I|$ is the total number of elements in $\{I\}$.

In general D enumerates the potential disruptions for a solution $\hat{x}$, while P explains why they apply: together, they generalize the break-sets from [Hebrard et al., 2004b].

The E formulas specify what each disruption actually does.

5.3.2 Decomposition Scheme

We exploit the basic properties of super solutions to obtain an LBBD scheme, and more importantly cut generation procedures, which can be applied to any target problem. The method will inherit some classical benefits of LBBD (such as the ability to use different solvers for the master and the subproblem), and provide some unique benefits.

We task our master problem to find a solution, and the subproblem to identify how to repair every disruption. This choice provides two advantages: 1) since disruptions in super solutions are unrelated, we can handle each of them *in a separate subproblem*. In turn, 2) having multiple subproblems allows us to *determine which disruptions should be handled*: this is precisely what makes our method capable of dealing with generic disruptions.

Formally, *the master is the same as the target problem*, i.e. $\mathbf{MP}$. Given a master solution $\hat{x}$ and a disruption index d, the *basic form of one subproblem* is:

$$E_{d,\hat{x}}(y) \qquad (\mathbf{SP_0}) \tag{5.15}$$

$$y \in X \tag{5.16}$$

where the y variables correspond 1-1 to the master variables and are used to define the repair solution. Hence, the subproblem is the same as the original problem, without the cost function and with the addition of the disruption effects. Since the subproblem has no impact on the cost of the master solution, if at any point all subproblems are feasible, the current $\hat{x}$ represents an optimal robust solution.

Algorithm 4: BD4SS

Require: Triplet $\langle D, E, P \rangle$
  repeat
    solve master problem to find $\hat{x}$
    if master infeasible **then break**
    $robust = \top$
    for $d \in D(\hat{x})$ **do**
      if the subproblem for $(\hat{x}, d)$ is infeasible **then**
        $robust = \bot$
        generate cuts for $(\hat{x}, d)$
        break
  until $robust$

Pseudo-code for a basic version of this process is provided in Algorithm 4, whose name stands for "Bender Decomposition For Super Solutions". Note that, while stopping every iteration at the first infeasible subproblem is enough for convergence, processing all potential disruptions may lead to more cuts and be beneficial in the long run.

5.3.3 Basic Cuts

A *trivial cut* can be obtained by observing that there are two options for dealing with a non-repairable disruption: 1) making sure that it cannot occur; 2) changing some assignments in the current solution. Hence, a valid cut for a disruption $(d, \hat{x})$ which lead to an infesible subproblem is:

$$\neg P_{d,\hat{x}}(x) \vee [\![H(x, \hat{x}) \geq 1]\!], \tag{5.17}$$

where H is the Hamming distance, i.e.:

$$H(x', x'') = \sum_{i=1}^{n} |x'_i \neq x''_i|. \tag{5.18}$$

Equation (5.17) states that either the precondition for d should be violated, or at least one change should be made w.r.t. $\hat{x}$.

An improved cut can be obtained by modifying the subproblem so that its objective is to minimize the distance between the repair solution and the current master solution, i.e.:

$$\min \beta = H(y, \hat{x}) \qquad (\mathbf{SP}) \tag{5.19}$$

$$\text{s.t. } E_{d,\hat{x}}(y) \tag{5.20}$$

$$y \in X \tag{5.21}$$

If the optimal subproblem cost β^* is not greater than b, then a repair solution exists. This counts as a "feasible" result for the purpose of Algorithm 4. However, if $\beta^* > b$, then we know that any repair solution will need to change at least β^* variables. Based on this observation, we can derive a tightened version of Equation (5.17), which serves as our *basic cut*:

$$\neg P_{d,\hat{x}}(x) \vee [\![H(x, \hat{x}) \geq \beta^* - b]\!], \tag{5.22}$$

i.e. any new master solution should either violate the precondition $P_{d,\hat{x}}(x)$, or $\beta^* - b$ is the minimum number of variables that needs to be changed.

Proof. In case of violation of the precondition for $(d, \hat{x})$, then the cut is correct. Otherwise, $P_{d,\hat{x}}(x)$ holds and finding a repair solution is necessary. Due to Remark 5.1, we have that:

$$P_{d,\hat{x}}(x) \Rightarrow E_{d,\hat{x}}(y). \tag{5.23}$$

Hence, the same $E_{d,\hat{x}}(y)$ will be part of any possible subproblem for the disruption. This means that *all such subproblems will share the same feasible space*. Let such feasible space be $X_{d,\hat{x}}$. We know that:

$$\min\{H(y, \hat{x}) : y \in X_{d,\hat{x}}\} \geq \beta^*. \tag{5.24}$$

Then, due to triangular inequality in the Hamming distance:

$$\min\{H(y, x) + H(x, \hat{x}) : y \in X_{d,\hat{x}}\} \geq \beta^*. \tag{5.25}$$

We wish for x to be repairable, hence the maximum value for $H(y, x)$ is b. From this, we obtain:

$$\min\{b + H(x, \hat{x}) : y \in X_{d,\hat{x}}\} \geq \beta^*. \tag{5.26}$$

And therefore:

$$\min\{H(x, \hat{x}) : y \in X_{d,\hat{x}}\} \geq \beta^* - b, \tag{5.27}$$

which proves correctness.

5.3.4 Strengthened Cuts

Our basic cuts can be strengthened by proving that a *subset* of assignments in the master solution is enough to prevent repairability. Generic cut strengthening procedure for Bender Decomposition exist, but they are computationally very expensive. Here, we leverage the structure of super solutions to obtain insights into which variables should be included in the cut. In particular, given a set of indices I, let H_I be the Hamming distance restricted to I, i.e.:

$$H_I(x', x'') = \sum_{i \in I} |x'_i \neq x''_i|. \tag{5.28}$$

Given an infeasible subproblem, we will try to identify a subset of indices I such that the minimum H_I distance from the master solution $\hat{x}$ is still greater than b. We do this

heuristically and iteratively, by starting from the variables that are different from $\hat{x}$ in the solution of $\mathbf{SP}$, and then by solving the modified subproblem:

$$\min H_I(y, \hat{x}) + \frac{1}{n} H_{\overline{I}}(y, \hat{x}) \qquad (\mathbf{SP_I}) \tag{5.29}$$

$$\text{s.t. } E_{d,\hat{x}}(y) \tag{5.30}$$

$$y \in X, \tag{5.31}$$

where $\overline{I}$ is the set of variable indices that are not in I. The cost function is a lexicographic composition, whose main goal is to minimize the distance $H_I(y, \hat{x})$. Let y^* be the optimal solution of this subproblem. If $H_I(y^*, \hat{x}) > b$, we can generate the *strengthened cut*:

$$\neg P_{d,\hat{x}}(x) \vee [\![H_I(x, \hat{x}) \geq \beta_I^* - b]\!], \tag{5.32}$$

were $\beta_I^* = H_I(y^*, \hat{x})$. We can then expand the I set by including all variables that take different values in y^* and $\hat{x}$. The term $1/n H_{\overline{I}}(y, \hat{x})$ exists to reduce the number of variables that are added to I in this fashion at each step.

The pseudo-code for the procedure is given in Algorithm 5, and adds two optimizations: first, it generates cuts only when the value of $H_I(y^*, \hat{x})$ increases: since the I set grows monotonically, subsequent iterations with the same H_I distance result in redundant cuts. Second, the process is stopped as soon as β_I^* becomes equal to β^*, which is guaranteed to happen when all variable indices are included in the I set.

Algorithm 5: STRENGTHEN

Require: A disruption $(d, \hat{x})$
  solve $\mathbf{SP}$ to find y^* and β^*
  $I = \{i \in [1..n] : y_i^* \neq \hat{x}_i\}$
  $\beta_I^* = 0$
  repeat
    Solve $\mathbf{SP_I}$ to find y^*
    if $H_I(y^*, \hat{x}) > \max(b, \beta_I^*)$ **then**
      $\beta_I^* = H_I(y^*, \hat{x})$
      Generate the cut from Equation (5.32)
  until $\beta_I^* = \beta^*$

5.3.5 Combined Cuts

A third cut generation method can be defined via a linear combination of pairs of existing cuts. In particular, let us suppose we have:

$$\neg P_{d_k,\hat{x}^k}(x) \vee [\![H_{I^k}(x,\hat{x}^k) \geq \beta^*_{I^k} - b]\!] \tag{5.33}$$
$$\neg P_{d_h,\hat{x}^h}(x) \vee [\![H_{I^k}(x,\hat{x}^h) \geq \beta^*_{I^h} - b]\!]\,. \tag{5.34}$$

The cuts are built respectively for disruptions $(d_k,\hat{x}^k)$ and $(d_h,\hat{x}^h)$, and are expressed in the form of Equation (5.32), which subsumes that of Equation (5.22). First, we observe that any inequality concerning a H_I distance holds true if we expand the set I. Hence, we can write:

$$\neg P_{d_k,\hat{x}^k}(x) \vee [\![H_{I^{kh}}(x,\hat{x}^k) \geq \beta^*_{I^k} - b]\!] \tag{5.35}$$
$$\neg P_{d_h,\hat{x}^h}(x) \vee [\![H_{I^{kh}}(x,\hat{x}^h) \geq \beta^*_{I^h} - b]\!]\,. \tag{5.36}$$

where both Hamming distances are defined on the same set $I^{kh} = I^k \cup I^h$. Due to triangular inequality, we have that:

$$H_{I^{kh}}(x,\hat{x}^k) + H_{I^{kh}}(x,\hat{x}^h) \geq H_{I^{kh}}(\hat{x}^k,\hat{x}^h). \tag{5.37}$$

Therefore, the two cuts can always be merged to yield a valid *combined cut*:

$$\neg P_{d_k,\hat{x}^k}(x) \vee \neg P_{d_h,\hat{x}^h}(x) \vee [\![H_{I^{kh}}(x,\hat{x}^k) + H_{I^{kh}}(x,\hat{x}^h) \geq H_{I^{kh}}(\hat{x}^k,\hat{x}^h)]\!]\,, \tag{5.38}$$

which is non-redundant if $H_{I^{kh}}(\hat{x}^k,\hat{x}^h) > \beta^*_{I^k} - \beta^*_{I^h} - 2b$. A simple procedure for generating all non-redundant combined cuts is provided in Algorithm 6, where n_c is assumed to be the number of basic or strengthened cuts generated so far.

Algorithm 6: COMBINE

Require: $C = \{(d_k,\hat{x}^k,I^k,\beta^*_{I^k}) : \forall k \in [1..n_c]\}$
  for $k \in [1..n_c - 1]$ **do**
    for $h \in [k+1, n_c]$ **do**
      if $H_{I^{kh}}(\hat{x}^k,\hat{x}^h) > \beta^*_{I^k} - \beta^*_{I^h}$ -2b **then**
        Generate the cut from Equation (5.38)

5.3.6 Remarks

The master and the subproblem for our decomposition correspond very closely to the original target problem in terms of both structure and size. This makes them easy to implement (in contrast to specialized search methods), and typically not harder to solve than the target problem (in contrast to reformulated models).

Finding super solutions, however, remains very challenging, as it should be expected of any method providing strong robustness guarantees. Each iteration of the method requires to solve multiple, typically NP-hard problems. Moreover, as the number of iterations grows, the master tends to become more difficult due to the accumulated cuts.

The use of procedural code for many sections of our methods provides a substantial degree of flexibility: in particular, it allows our method to deal with generic disruptions. The available flexibility could be further exploited: for example, one could conceive a hybrid chance-constraint/super solution method to provide robustness against disruptions with a given total probability of occurrence.

Another advantage of the approach is that, in case of early stops, one can still access a master solution that is robust against *some* disruptions. This can be achieved via reformulated models, but not via specialised search algorithms (at least not trivially).

Note however that our process is not designed to provide such intermediate solutions as a main output, and there is no guarantee that more iterations will correspond to a larger number of rapairable disruptions.

5.4 A Case Study on the KEP

We now proceed to grounding our method on the Kidney Exchange Problem, using the cycle formulation from Section 5.2 as a basis. In this case the Master Problem is the original KEP model, and the sub-problem is the repair. When the subproblem is not feasible it will generate the respective cut/cuts to be added to the master. Therefore if the subproblem is feasible it means we found the robust solution.

For sake of simplicity, we will be interested in finding $(1, b, c)$ super solutions that are

repairable w.r.t. the loss of *cycles*. Hence we have:

$$D(\hat{x}) = \{d \in [1..n_c] : \hat{x}_d = 1\} \tag{5.39}$$

$$P_{d,\hat{x}}(x) = [\![x_d = 1]\!] \tag{5.40}$$

$$E_{d,\hat{x}}(x) = [\![x_d = 0]\!] \tag{5.41}$$

where n_c is the number of cycles, and a disruption index d corresponds to a cycle. Each cut is in the form:

$$\neg P_{d,\hat{x}}(x) \vee [\![H_I(x, \hat{x}) \geq \beta_I^* - b]\!] \tag{5.42}$$

corresponds to:

$$[\![x_d = 0]\!] \vee \left[\!\!\left[\sum_{\substack{i \in [1..n] \\ \hat{x}_i = 0}} x_i + \sum_{\substack{i \in [1..n] \\ \hat{x}_i = 1}} (1 - x_i) \geq \beta_I^* - b \right]\!\!\right] \tag{5.43}$$

and can be translated into the mathematical constraint:

$$\sum_{\substack{i \in [1..n] \\ \hat{x}_i = 0}} x_i + \sum_{\substack{i \in [1..n] \\ \hat{x}_i = 1}} (1 - x_i) \geq (\beta_I^* - b)(1 - x_d). \tag{5.44}$$

The right hand side is non trivial (i.e. larger than 0) iff x_d is equal to 0, i.e. if the precondition for the disruption is violated. This (little) information is sufficient to apply all the techniques described in Section 5.3 to the KEP.

Since we are dealing with $(1, b, c)$ super solutions, we need to include an additional cost-bounding constraint both in the master and in the subproblems:

$$\sum_{i \in \mathcal{C}} w_i x_i \geq c^* - c \tag{5.45}$$

where c^* is the cost of an optimal non-robust solution. Intuitively, in a robust solution we do not want to loose too many lives, which makes sense from a medical perspective.

It is not particularly difficult to obtain $\langle D, E, P \rangle$ triplets and cut expressions for other disruptions, including the loss of multiple nodes, edges, or cycles.

For sake of example, robustness w.r.t. the loss of a *node* (rather than a cycle) could be

formulated as:

$$D(\hat{x}) = \left\{ d \in [1..n] : \sum_{c \in \mathcal{C}, d \in C_c} \hat{x}_c = 1 \right\} \tag{5.46}$$

$$E_{d,\hat{x}}(x) = \left[\!\!\left[\sum_{c \in \mathcal{C}, d \in C_c} \hat{x}_c = 0 \right]\!\!\right] \tag{5.47}$$

$$P_{d,\hat{x}}(x) = \left[\!\!\left[\sum_{c \in \mathcal{C}, d \in C_c} \hat{x}_c = 1 \right]\!\!\right] \tag{5.48}$$

where n is the number of nodes. In this case, a disruption is associated to each node d that is used by at least one cycle, and its effect is forbidding the selection of all cycles connected to node d. Similar expression can be obtained for other disruptions, including the loss of multiple nodes, or edges. All such cases are not covered in our experimentation.

5.4.1 Model Reformulation

As a term of comparison, we obtained a model reformulation by generalizing the approach from [Weigel et al., 1998]. In particular, we generate a model with a "main" vector of n decision variables x, and a vector of n repair variables y_i for each x_i. Overall, the model has $n \times (n+1)$ variables, and corresponds to the problem of finding an $(1, b)$ super solution and all its repair solutions.

We generate a CSP with $n \times (n+1)$ variables, where every variable in the original CSP appears twice. Furthermore we need to add a disequality constraint between each variable and its duplicate. , i.e., $X_j = d_i \implies X_j^{'} \neq d_i$ and add the constraint between X_i and $X_j^{'}$ if there was a constraint between X_i and X_j for example. The disadvantage of this approach is that we represent the whole CSP twice. Therefore the fault tolerant

formulations ($\mathbf{FTF_{kep}}$) adapted to the Kidney Exchange problem is the following:

$$\max z = \sum_{i \in \mathcal{C}} w_i x_i \qquad (\mathbf{FTF_{KEP}}) \qquad (5.49)$$

$$\text{s.t.} \sum_{i \in \mathcal{C}: k \in C_i} x_i \leq 1 \qquad \forall k \in V \qquad (5.50)$$

$$\sum_{j \in \mathcal{C}: k \in C_j} y_{ij} \leq 1 \qquad \forall i \in \mathcal{C}, \forall k \in V \qquad (5.51)$$

$$y_{ii} = 1 - x_i \qquad \forall i \in \mathcal{C} \qquad (5.52)$$

$$x_i \in \{0, 1\} \qquad \forall i \in \mathcal{C} \qquad (5.53)$$

$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in \mathcal{C} \qquad (5.54)$$

where Equation (5.50) and (5.51) ensure that neither the super solution nor any repair solution can use a node twice. Equation (5.52) corresponds to the effects of the disruption. We then need to add constraints to limit the Hamming distance between the super solution and the repair solutions. In particular, for each $i \in \mathcal{C}$ we have:

$$\sum_{j \in \mathcal{C}} y_{ij}(1 - x_j) + \sum_{j \in \mathcal{C}} (1 - y_{ij}) x_j \leq b + |\mathcal{C}|(1 - x_i)$$

which is a quadratic constraint, with a right-hand side term that is non-trival iff $x_i = 1$, i.e. if cycle i is used. Finally, we need one last set of constraints to handle the cost bound:

$$\sum_{j \in \mathcal{C}} w_j y_{ij} \geq c^* - c \qquad \forall i \in \mathcal{C} \qquad (5.55)$$

Overall, the reformulated model is not only considerably larger, but also non-linear (although a big-M based linearization is of course possible).

5.4.1.1 Space Complexity

Given *n* the number of variables and *c* the number of constraints for the original problem, therefore for the reformulation approach the number of variables is $n \times (n + 1)$, and the number of constraints is$(n + 1) \times c$. The domains and constraints relation do not change. The space complexity therefore is in $\mathcal{O}(n^2)$. The reformulation approach works for small instances, but definitely will not scale.

5.5 Greedy model

Finding a solution of an algorithmic match does not mean that all the exchanges will go to an actual transplant. Some may fail due to last minute tests, the patient withdraws from the system, or the conditions get worse for the patient. When a pair pulls out from a solution, one simple way is to rerun the program again, excluding the failure that caused the damage.

In [Dickerson et al., 2013, Pedroso, 2014, Klimentova et al., 2016] study the problem of failure aware of transplants in the kidney exchange problem. [Dickerson et al., 2013] proposes to move away from the deterministic clearing model used by kidney exchanges into a probabilistic model where the input includes failure probabilities on possible planned transplants, and the output is a transplant plan with maximum expected value, then they develop a custom (branch-and-price based) integer program solver specifically for the probabilistic clearing problem. [Pedroso, 2014] and [Klimentova et al., 2016] proposes a method for computing the expectation for the length of a maximum set of vertex-disjoint cycles in a digraph where vertices and/or arcs are subject to failure with a known probability. Moreover, they are adjusting the weights of the cycles or including some failure probabilities. Those approaches tend to favour more reliable exchanges, rather than to define repair actions in case of failures. In this section, we are showing a greedy model, i.e. we consider to rerun the program when a pair/edge causes a failure.

Algorithm 7: Kidney node failure

Solve KEP on graph $\langle V, A(V)\rangle$
let S_{opt} be an optimal solution for a given instance, if exists
S_p = all pairs in S_{opt}
for all $p \in S_{pairs}$ **do**
    let c_p be all cycles that contain p
    $x \leftarrow x \setminus x_{c_p}$
    solution_rematch $\leftarrow$ *rematch* $(1, b, c)$
for all $\forall sol \in solution\_rematch$ **do**
    $b \leftarrow$ max $Outs$
    $c \leftarrow$ max $Outs$ - max Ins
return $(1, b, c)$

With this in mind, we will explore two possible failures:

- edge failure, when the compatible edge between two pairs is broken, therefore all the cycles that contain the particular the edge will be removed from the compatibility graph;

- node failure, when a pair pulls out from the system, therefore all the cycles containing that pair will be removed from the compatibility graph.

For each pair in the optimal solution we identify the variables that belong to. The break set is the set of variable that may break if the respective pair may break, and the repair set is the set of variables that can be used as repair. Breaking a pair may require more than one variable to change. If a pair pulls out from the program, all the cycles that he is taking part in will break as well. For each pair in the optimal solution, we break the variables that contain the pair and we resolve the problem, using the cycle formulation. Below we describe the algorithm for the node failure and the edge failure procedure.

Algorithm 7 describes the pseudo-code of the node failure. In line 1, first it identifies the optimal solution and in the following lines (2-6) identifies the break set and the cycles that belongs to, i.e. the nodes that might fail, therefore we will only rerun the matching program only for those nodes. Specifically, line 6 will remove the all cycles that the node is part of, and in line 7 call the $rematch(1, b, c)$ function.

Algorithm 8 solves a new KEP on the restricted graph and detects the nodes that exit solution, and the new nodes in the solution. Finally the Algorithm 7 will return the tuple $(1, b, c)$, as the format described in the previous sections.

Algorithm 8: *rematch* $(1, b, c)$

1: Solve KEP on graph $\langle V \setminus S_p, A(V \setminus S_p)\rangle$
2: $S_{rematch}$ = new optimal solution, if exists
3: **if** solution **then**
4: S_{new} = all pairs in $S_{rematch}$
5: Outs = $S_p \setminus S_{rematch}$
6: Ins = $S_{rematch} \setminus S_p$
7: return ($S_{rematch}$, Outs, Ins)

Algorithm 9 describes the pseudo-code of the edge-failure procedure, which is very similar to Algorithm 7, with the difference that Algorithm 8 will rerun the restricted graph as $\langle V, A(V \setminus V_r)\rangle$. Both algorithms describe a greedy way to deal with last minute rearrangements, as many hospitals prefer to rerun the matching program. We do encourage to use

Algorithm 9: Kidney edge repair

```
1: Solve KEP on graph ⟨V, A(V)⟩
2: let S_opt be an optimal solution for a given instance, if exists
3: V_r = all edges in S_opt
4: for all l ∈ V_r do
5:     let c_p be all cycles that contain l
6:     x ← x \ x_{c_p}
7:     solution_rematch ← rematch (1, b, c)
8: for all ∀sol ∈ solution_rematch do
9:     b ← max Outs
10:    c ← max Outs - max Ins
11: return (1, b, c)
```

5.6 Experiments

In this section we present our empirical experiments, which were designed with two main goals:

- comparing the effectiveness of our method against one with a similar applicability and flexibility, i.e. the reformulation from Section 5.4;
- testing the effectiveness of different cut generation procedures.

Last, we will present the results for the greedy algorithm.

Instances We use the benchmarks from [Mattei and Walsh, 2013], which were obtained via the state-of-the-art donor pool generation method described in [Saidman et al., 2006]. We consider all transplants equally worthy, hence the weight of each cycle corresponds to its number of exchanges. The maximum length of a cycle is three.

5.6.1 Choosing the *(a, b, c)* Parameters

As mentioned in Section 5.4, we focus on finding $(1, b, c)$ super solutions, where disruptions correspond to (single) selected cycles becoming non-viable. Therefore, we have $a = 1$, and we are left with two main parameters to tune, i.e. b and c.

The b parameter refers to how many variables can be changed in a repair solution, excluding (in classical super solution approaches) the variables having lost their value. In our formulation, this requires to exclude such variables in the computation of the

Hamming distance. For most problems, finding a repair solution by changing just one variable (i.e. $b = 1$) is a difficult task, and the KEP is no exception. Interestingly, however, we found that setting $b = 2$ was enough for the solution of the cycle formulation to be robust, for most of the instances in our benchmarks. Therefore, we chose to limit our experimentation to $b = 1$.

The use of the cost-bounding parameter c is in this case mandatory. In fact, any KEP solution remains feasible if a cycle is removed: hence, without some cost bound, the loss of a cycle could be countered by doing nothing at all. In general, if k is the largest size of an allowed exchange, then any solution is trivially (a, b) robust, unless we require not losing more than $ak-1$ units from the non-robust optimum (i.e. $c \leq ak-1$). In our case, there are therefore only two reasonable values for c, i.e. 1 and 2. We chose to keep $c = 2$ in our experimentation.

5.6.2 Results

We performed experiments over groups of 10 instances with number of nodes $n_{pairs} \in \{16, 32, 64, 128\}$. We solved them via the reformulation-based model, and via our Benders Decomposition approach, using different cut generation procedures, and always stopping the process after $5 \times n_{pairs}$ master iterations.

We investigated approaches that stop each iteration of Algorithm 4 after the first infeasible subproblem (with the _1st suffix), and approaches that process all disruptions (with the _all suffix). The _all-suffix approaches will generate more cuts: this may reduce the number of iterations, but require more time and makes the master more difficult to solve. We considered the following combination of the cuts from Section 5.3: only basic cuts, identifiable from the cut1 prefix; basic cuts + combined cuts, identifiable by cut2; strengthened cuts, identifiable by cut3; and strengthened cuts + combined cuts, identifiable by cut4. Strengthened cuts are expensive to produce, but may reduce the number of iterations. Combined cuts are cheap to obtain, but their large number may make the master more difficult to solve.

The results of the experimentation are reported in Table 5.1, Table 5.2, Figure 5.1, and Figure 5.2. The table reports, for each approach and each size: 1) the fraction of instances for which a robust solution was found; 2) the fraction of instances for which robustness was proved infeasible; and 3) the fraction of open instances, for which the iteration limit was reached. Overall, the reformulation scales very poorly, leading to consistent memory problems for sizes larger than 32. Using combined cuts does not have a large impact on the number of open instances but it does have a hug impact

Table 5.1: Solutions of the Reformulation and Benders decomposition

#Pairs	Reform	Benders decomposition											
		cut1_1st			cut2_1st			cut3_1st			cut4_1st		
	%sol	%sol	nosol	open	%sol	nosol	open	%sol	nosol	open	%sol	nosol	open
16	50	100	0	0	100	0	0	100	20	0	100	0	0
32	10	80	0	20	80	0	20	80	20	0	80	20	0
64	-	60	0	40	60	0	40	70	30	0	70	30	0
128	-	90	0	10	90	0	10	90	0	10	90	0	10

Table 5.2: Solutions of Benders decomposition

#Pairs	Benders decomposition											
	cut1_all			cut2_all			cut3_all			cut4_all		
	%sol	nosol	open	%sol	nosol	open	%sol	nosol	open	%sol	nosol	open
16	100	0	0	100	0	0	100	0	0	100	0	0
32	80	0	20	70	0	30	80	20	0	80	20	0
64	60	0	40	60	30	10	70	30	0	70	30	0
128	90	0	10				100	0	0			

increasing the size of the instances, whereas strengthened cuts are very effective. Processing all disruption at each iteration seems to be slightly detrimental.

In Figure 5.1 we show the average solution time of all approaches in logarithmic scale (including the non-robust cycle formulation, referred to as optimal). The time for the reformulation grows very quickly with the number of pairs, and the series has only two data points. The approaches using strengthened cuts tend to be the most effective, while the time for methods using combined cuts are consistently higher than those who do not generate them.

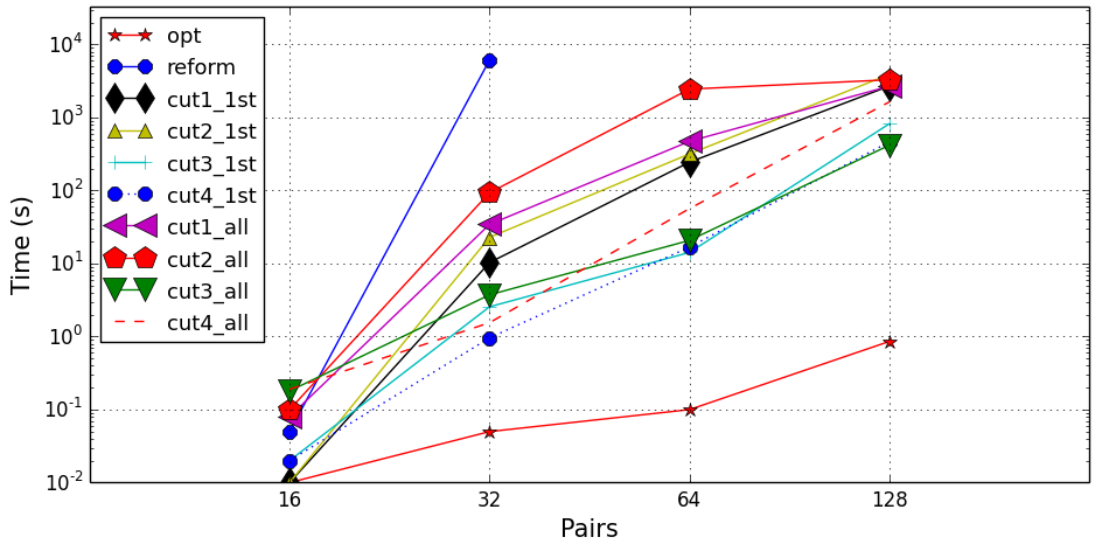


Figure 5.1: Resolution time with b=1 log scale

In Figure 5.2 we plot the average number of master iterations for the Benders Decomposition, again in logarithmic scale. Strengthened cuts are the clear winner. The _all-suffix approaches do sometimes iterate less than the corresponding _1st-suffix ones, although this is not always the case. The reason is that adding more cuts may force the master to produce different sequences of solutions, and therefore have a non-monotonic effect on the number of iterations. Combined cuts seem to have a more consistent effect, although not a dramatic one: this suggests that a more difficult master is the main reason for their increased solution time, and that controlling the number of generated cuts per iteration may make them more useful.

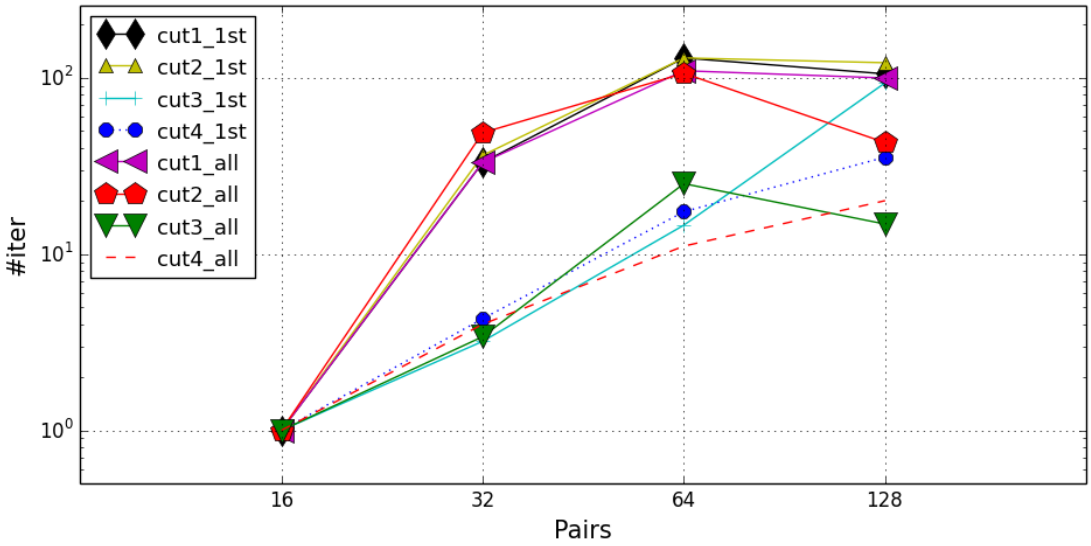


Figure 5.2: Number of master's iterations with b=1 log scale

5.6.3 Results for the Greedy Model

The instances are taken from [Dickerson et al., 2012b], which uses a state of the art donor pool generation method, described in [Saidman et al., 2006]. In this section we present only the node failure as it has a bigger impact on the graph then the edge failure. For the edge failure we found that it was easier to rerun the program than the node failure. In Table 5.3 we present the average results over 10 instances per class, and these results are computed using the classical cycle formulation. The results presented in Table 5.4 are solved using the model of [Manlove and O'Malley, 2014], which is currently running in the UK. Again, we show the averages over 10 instances per class. For the sake of image resolution we only show classes of 16 pairs and 32 pairs, with the respective altruistic number.

Table 5.3: Summary of instances using the cycle formulation model.

Nodes	Objective	Time	Cycles	Chains	Arcs
16p+0 altr	4.3	0.024	1.8	0	50
16p+1 altr	8.3	0.236	2.3	1	89.7
16p+2 altr	9.5	0.439	2	2	106
32p+0 altr	17.1	0.866	6.8	0	267
32p+1 altr	17.2	10.45	5.4	1	296.2
32p+3 altr	20.9	112.693	5.2	3	384.2
32p+4 altr	22.8	382.380	5.1	4	435.3

Between the Table 5.3 and Table 5.4, we observe similar results, therefore we choose

for the Algorithm 7 to use the classical KEP cycle formulation. We notice that increasing the number of edges the number of cycles and chains increase, and the running time increases as well.

5.6.3.1 Results of the Rematching Model

Table 5.4: Summary of instances using the model used in UK.

Nodes	Objective	Time	Cycles	Chains	Arcs
16p+0 altr	4.3	0.040	1.8	0	50
16p+1 altr	8.3	0.234	2.3	1	89.7
16p+2 altr	9.29	0.333	2	2	106
32p+0 altr	17.1	0.269	7.1	0	267
32p+1 altr	17.2	1.304	5.8	1	296.2
32p+3 altr	20.9	3.579	5.9	3	384.2
32p+4 altr	22.88	5.227	5.75	4	435.3

Every plot has 10 instances per class. The "optimal" bars represent the value of any optimal solution, the "max outs" bars shows the maximum number of pairs that are not in the new solutions. The "lost transplants" bars represent the maximum number of pairs that will not take part in any other solution. For example, for the second instance in Figure 5.3, the Algorithm 7 will return the tuple $(1, 3, 2)$, which means that if one pair pulls out from the optimal solution, three other pairs will exit the program, and in the worst case we will loose two pairs.

We have tried different pairs number, i.e. 16, 32 pairs, with no altruistic, 1,2,3,4 altruistic donors. For each pair in the solution, we choose to rerun the algorithm excluding the cycles the pair is part of. We calculate for each run the maximum number of pairs that exit the solution, and the average of the transplants that are lost as they can not be rematched in any other solution. Grey bars represent the number of transplants in the optimal solution.

The Figures 5.5 and 5.6 show the trend of the greedy rematched algorithm. We conclude by saying that the algorithm present in the previous section is by far preferred, instead of the greedy algorithm. Many hospitals prefer to go ahead with the solution computed and excluding the cycle which is broken and add the other pairs back in the waiting list, instead of rerunning the algorithm.

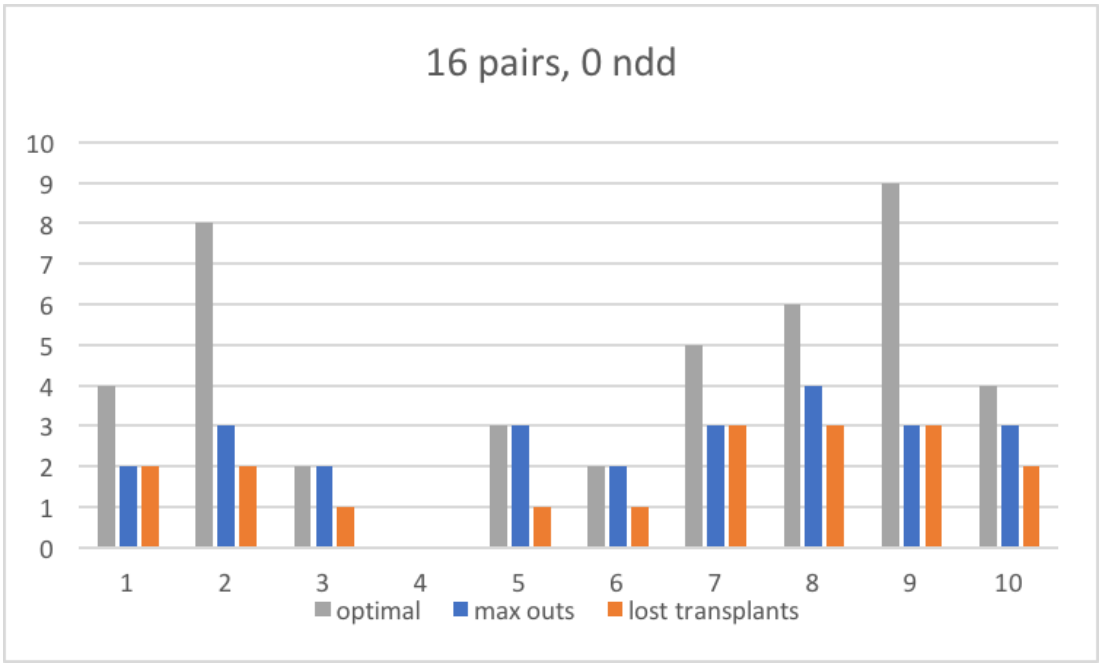


Figure 5.3: Illustration of a kidney exchange rematch with 16 pairs.

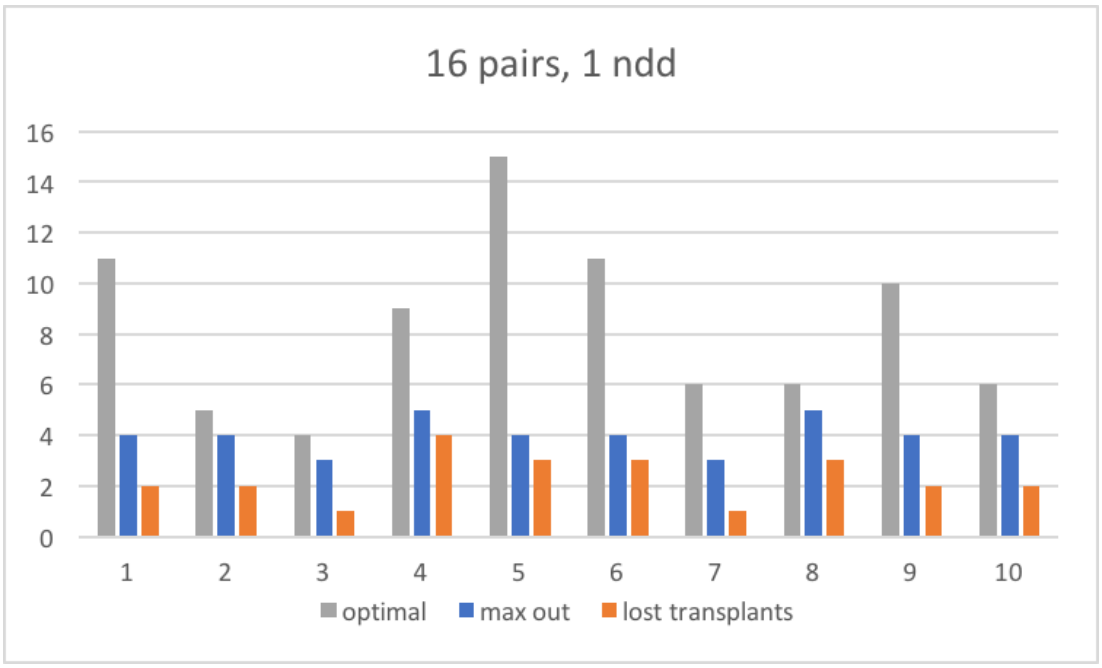


Figure 5.4: Illustration of a kidney exchange rematch with 16 pairs and 1 altruistic donor.

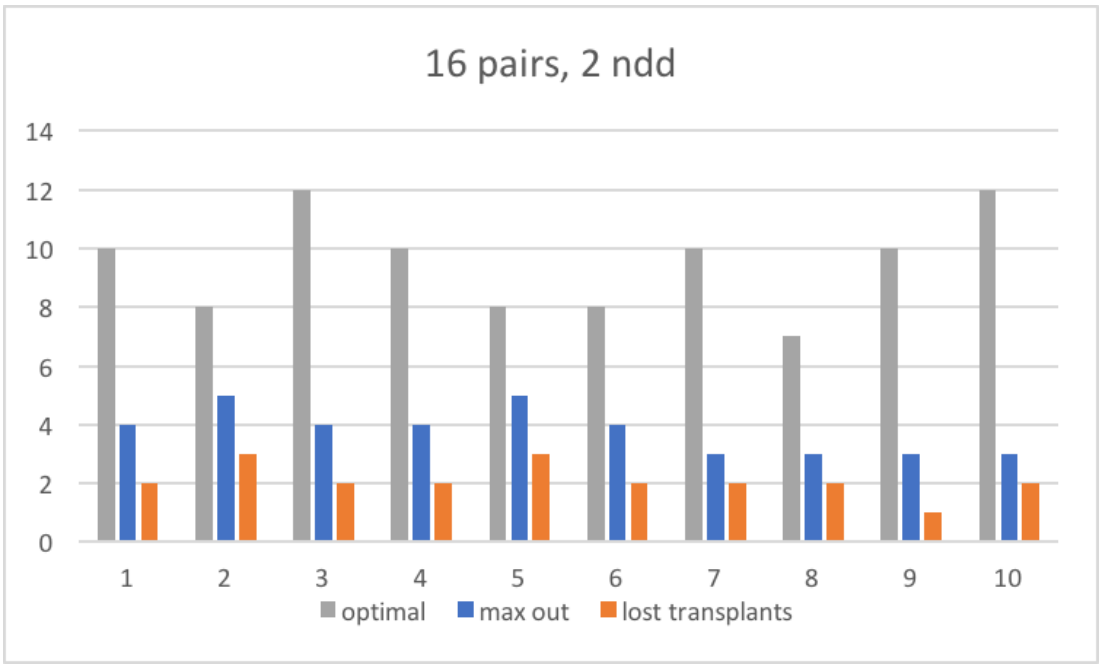


Figure 5.5: Illustration of a kidney exchange rematch with 16 pairs and 2 altruistic donor.

Figure 5.6: Illustration of a kidney exchange rematch with 32 pairs and 0 altruistic donor.

5.7 Chapter Summary

We have defined a Benders Decomposition approach to obtain super solutions for a satisfaction or optimization problem, together with a number of general cut generation procedures. Our method can deal with generic disruptions, dramatically outperforms reformulation-based approaches in terms of scalability, and is more flexible and much easier to apply compared to specialized algorithms from the literature. We believe our contributions are a significant step towards making super solution widely applicable to practical problem.

The method is not without drawbacks and, in our current research, many improvement directions are still open: these include cuts defined using multiple subproblems, principled approaches to limit cut generation, improvements to the anytime features of the approach, and hybridization with probabilistic models.

Chapter 6

Conclusions and Future work

"My goal is simple. It is a complete understanding of the universe, why it is as it is and why it exists at all." Stephen Hawking

6.1 Conclusions

This thesis outlined a number of algorithms for two matching problems under preferences, i.e. the popular matching problem and the kidney exchange problem, centered for solving combinatorial and optimisation problems.

In **Chapter 3** we described how to encode a popular matching using the constraint programming approach. We also showed the time complexity of finding an optimal popular matching where ties are allowed in the preference lists. We showed that the optimal popular matching with ties in the preference list can be reached in polynomial time, and gave two models to show this, i.e. the minimum weight perfect matching and a flow circulation.

Moreover, we considered the case where additional copies can be added for the instances that does not admit a popular matching, i.e. the the popular matching problem with copies. We introduced new graph properties for this extended case, considering more than two copies, and the experiments showed essentially the key aspects of a good branching strategy, as well as the efficiency of our dominance rules.

The kidney exchange problem is divided in two major parts. The first part described and solved dynamic kidney exchanges, where pairs arrive and departure over time. The second part focused on the offline version of the problem which search for a robust matching managing uncertainty, i.e. multiple cycle/edge/pairs disruptions. In particular:

A. In **Chapter 4**, first, we presented an online anticipatory algorithm scenario sampling using a single KEP constructed using all scenarios. We simply proposed to employ the classical Sample Average Approximation scheme for two-stage stochastic programs and showed that our approach performs better than other scenario based algorithms. Second, we proposed a sampling-free probabilistic model, Abstract Exchange Graph (AEG), overcoming the scalability issues of the scenario based algorithms.

B. In **Chapter 5**, we formally defined the notion of super solution to the kidney exchange problem. Therefore, we used Logic-based Benders Decomposition to boost the search for a robust solution. We applied this method to the kidney exchange problem with great results. Using valid cuts, clears a large portion of the search space. Therefore, investigating other cuts can improve prunning the search space for large instances.

6.2 Future Work

There is much space for improvements and optimisation around the matching problems under preferences. We present some future research directions and a list of open questions that arose from the research performed as part of this thesis.

6.2.1 Extended Popular Matchings

For a given bipartite matching problem there may be many different matchings of a certain type, e.g. Pareto optimal, rank-maximal etc that can be interesting to encode them in a constraint programming setting. As a first step, given a bipartite matching problem and an optimality criterion matching, it would be useful to find efficient algorithms to determine whether the problem instance admits a unique matching, and if not to find such matching.

There is a wide range of possibilities for future study for the popular matchings with copies. A future work for this problem would be to minimise the cost when an extra copy is added. From a practical point of view, it would be nice to understand the behaviour in real-life situations of the methods described in **Chapter 3**, i.e. how "good" are the popular matchings likely to be for a given instance of House allocation or Hospital/residents derived from some practical application.

6.2.2 Extended Kidney Exchanges

In practice, kidney exchanges are subject to a variety of types of uncertainty, like the uncertainty over the possible exchanges or long-term future layout of the pool. The super-solution model described in **Chapter 5** can be extended to a dynamic setting, where we seek to find robust solutions in the pool that evolves over time.

Further comparison with other failure-aware models from [Dickerson et al., 2013, Klimentova et al., 2016] can also be performed. Nevertheless, machine learning techniques for learning dynamic matching policies or learning failure rates [Ashlagi et al., 2013], which can be incorporated in the program and comparison with work from [Dickerson and Sandholm, 2015] is almost mandatory. Incentivize donor participation in kidney exchange, would have great impact on the state of organ donation [Stoler et al., 2017].

Open questions:

- **Fairness in health care.** Create more general models of fairness in organ exchange, i.e. in [Hooker and Williams, 2012] the authors give a general method for combining equity and economic efficiency in a single mathematical programming model. It would be interesting applying this to the kidney exchange problem. One concrete first step would be to apply the work of [Hooker and Williams, 2012] to model the equity trade-off in joint kidney-liver exchange.
- **Multiple organ exchanges.** A possible evolution of the program could be to include also the compatible pairs [Santos et al., 2017] in the system. Nevertheless multiple organ exchanges could create more exchanges, i.e. work in [Dickerson and Sandholm, 2013] considers kidneys and liver exchanges.
- **Other anticipatory algorithms.** A comparison with other sampling-free methods from [Dickerson et al., 2012a] or [Dickerson and Sandholm, 2015] can be considered a priority for future research.

- **Window time constraints** Studying the kidney exchange in a more detailed setting, i.e., considering the time validity of a deceased donor, or the distance between the patient and the donor hospital.
- **Clearing kidney markets on a nationwide scale.** Different methods to clear these markets has been studied considering either column generation or constraint generation [Abraham et al., 2007a, Dickerson et al., 2012b, Plaut et al., 2016]. Further extensions of the techniques presented in **Chapter 4** and **Chapter 5** can be performed to model more scalable algorithms.

Bibliography

[car, 2019] (2019). CARMS canadian resident matching service. https://www.carms.ca/. Accessed: 2019-09-30.

[Abdulkadiroğlu et al., 2005] Abdulkadiroğlu, A., Pathak, P. A., Roth, A. E., and Sönmez, T. (2005). The boston public school match. *American Economic Review*, 95(2):368–371.

[Abraham et al., 2006] Abraham, D., Chen, N., Kumar, V., and Mirrokni, V. S. (2006). Assignment problems in rental markets. In *WINE,Greece,2006, Proceedings*, pages 198–213.

[Abraham et al., 2007a] Abraham, D. J., Blum, A., and Sandholm, T. (2007a). Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, pages 295–304.

[Abraham et al., 2004] Abraham, D. J., Cechlárová, K., Manlove, D. F., and Mehlhorn, K. (2004). Pareto optimality in house allocation problems. In *International Symposium on Algorithms and Computation*, pages 3–15. Springer.

[Abraham et al., 2007b] Abraham, D. J., Irving, R. W., Kavitha, T., and Mehlhorn, K. (2007b). Popular matchings. *SIAM J. Comput.*, 37(4):1030–1045.

[Acharyya et al., 2014] Acharyya, R., Chakraborty, S., and Jha, N. (2014). Counting popular matchings in house allocation problems. In *Computer Science - Theory and Applications - 9th International Computer Science Symposium in Russia, CSR 2014, Moscow, Russia, June 7-11, 2014. Proceedings*, pages 39–51.

[Alvelos et al., 2015] Alvelos, F., Klimentova, X., Rais, A., and Viana, A. (2015). A compact formulation for maximizing the expected number of transplants in kidney exchange programs. In *Journal of Physics: Conference Series*, volume 616, pages 1–6. IOP Publishing.

[Anderson et al., 2015] Anderson, R., Ashlagi, I., Gamarnik, D., and Roth, A. E. (2015). Finding long chains in kidney exchange using the traveling salesman problem. *Proceedings of the National Academy of Sciences*, 112(3):663–668.

[Ashlagi et al., 2011] Ashlagi, I., Gilchrist, D. S., Roth, A. E., and Rees, M. A. (2011). Nonsimultaneous chains and dominos in kidney-paired donation—revisited. *American Journal of transplantation*, 11(5):984–994.

[Ashlagi et al., 2013] Ashlagi, I., Jaillet, P., and Manshadi, V. H. (2013). Kidney exchange in dynamic sparse heterogenous pools. *arXiv preprint arXiv:1301.3509*.

[Ashlagi and Roth, 2012] Ashlagi, I. and Roth, A. E. (2012). New challenges in multihospital kidney exchange. *American Economic Review*, 102(3):354–59.

[Awasthi and Sandholm, 2009] Awasthi, P. and Sandholm, T. (2009). Online stochastic optimization in the large: Application to kidney exchange. In *Proc. of IJCAI*, volume 9, pages 405–411.

[Barnhart et al., 1998] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

[Beldiceanu et al., 2007] Beldiceanu, N., Carlsson, M., Demassey, S., and Petit, T. (2007). Global constraint catalogue: Past, present and future. *Constraints*, 12(1):21–62.

[Benini et al., 2011] Benini, L., Lombardi, M., Milano, M., and Ruggiero, M. (2011). Optimal resource allocation and scheduling for the cell be platform. *Annals of Operations Research*, 184(1):51–77.

[Berge, 1957] Berge, C. (1957). Two Theorems in Graph Theory. *Proceedings of the National Academy of Science*, 43:842–844.

[Biró, 2008] Biró, P. (2008). Student admissions in hungary as gale and shapley envisaged. *University of Glasgow Technical Report TR-2008-291*.

[Bofill et al., 2013] Bofill, M., Busquets, D., Muñoz, V., and Villaret, M. (2013). Reformulation based maxsat robustness. *Constraints*, 18(2):202–235.

[Chen and Snmez, 2002] Chen, Y. and Snmez, T. (2002). Improving efficiency of on-campus housing: An experimental study. *American Economic Review*, 92(5):1669–1686.

[Chisca et al., 2018] Chisca, D. S., Lombardi, M., Milano, M., and O'Sullivan, B. (2018). From offline to online kidney exchange optimization. In *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI, 5-7 November, Volos, Greece.*, pages 587–591.

[Chisca et al., 2019a] Chisca, D. S., Lombardi, M., Milano, M., and O'Sullivan, B. (2019a). Logic-based benders decomposition for super solutions: An application to the kidney exchange problem. In *Proceedings of Principles and Practice of Constraint Programming - 25th International Conference, CP, Stamford, CT, USA, September 30 - October 4*, pages 108–125.

[Chisca et al., 2019b] Chisca, D. S., Lombardi, M., Milano, M., and O'Sullivan, B. (2019b). A sampling-free anticipatory algorithm for the kidney exchange problem. In *Proceedings of Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR, Thessaloniki, Greece, June 4-7*, pages 146–162.

[Chisca et al., 2016] Chisca, D. S., Siala, M., Simonin, G., and O'Sullivan, B. (2016). A cp-based approach for popular matching. In *Proceedings of the Thirtieth AAAI, 2016, Phoenix, Arizona, USA.*, pages 4202–4203.

[Chisca et al., 2017] Chisca, D. S., Siala, M., Simonin, G., and O'Sullivan, B. (2017). New models for two variants of popular matching. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, Boston, MA, USA, November 6-8*, pages 752–759.

[Constantino et al., 2013] Constantino, M., Klimentova, X., Viana, A., and Rais, A. (2013). New insights on integer-programming models for the kidney exchange problem. *European Journal of Operational Research*, 231(1):57–68.

[Demirović et al., 2018] Demirović, E., Schwind, N., Okimoto, T., and Inoue, K. (2018). Recoverable team formation: Building teams resilient to change. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1362–1370. International Foundation for Autonomous Agents and Multiagent Systems.

[Dickerson et al., 2016] Dickerson, J. P., Manlove, D. F., Plaut, B., Sandholm, T., and Trimble, J. (2016). Position-indexed formulations for kidney exchange. In *Proc. of EC*, pages 25–42. ACM.

[Dickerson et al., 2012a] Dickerson, J. P., Procaccia, A. D., and Sandholm, T. (2012a). Dynamic matching via weighted myopia with application to kidney exchange. In

Proc. of AAAI, volume 2012, pages 98–100.

[Dickerson et al., 2012b] Dickerson, J. P., Procaccia, A. D., and Sandholm, T. (2012b). Optimizing kidney exchange with transplant chains: Theory and reality. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 711–718.

[Dickerson et al., 2013] Dickerson, J. P., Procaccia, A. D., and Sandholm, T. (2013). Failure-aware kidney exchange. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 323–340.

[Dickerson and Sandholm, 2013] Dickerson, J. P. and Sandholm, T. (2013). Liver and multi-organ exchange. *American Journal of Transplantation*, 13:272–273.

[Dickerson and Sandholm, 2015] Dickerson, J. P. and Sandholm, T. (2015). Futurematch: Combining human value judgments and machine learning to match in dynamic environments. In *AAAI*, pages 622–628.

[Edmonds and Karp, 1972] Edmonds, J. and Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264.

[Fredman and Tarjan, 1987] Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615.

[Gale and Shapley, 1962] Gale, D. and Shapley, L. S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, 69:9–15.

[Gardenfors, 1975] Gardenfors, P. (1975). Match making: Assignments based on bilateral preferences. *Systems Research and Behavioral Science*, 20:166–173.

[Garg et al., 2010] Garg, N., Kavitha, T., Kumar, A., Mehlhorn, K., and Mestre, J. (2010). Assigning papers to referees. *Algorithmica*, 58(1):119–136.

[Gent et al., 2001] Gent, I. P., Irving, R. W., Manlove, D., Prosser, P., and Smith, B. M. (2001). A constraint programming approach to the stable marriage problem. In *Proceedings CP 2001, Paphos, Cyprus, December*, pages 225–239.

[Gershkov and Moldovanu, 2009] Gershkov, A. and Moldovanu, B. (2009). Dynamic revenue maximization with heterogeneous objects: A mechanism design approach. *American economic Journal: microeconomics*, 1(2):168–98.

[Ginsberg et al., 1998] Ginsberg, M. L., Parkes, A. J., and Roy, A. (1998). Supermodels and robustness. In *AAAI/IAAI*, pages 334–339.

[Glorie et al., 2014] Glorie, K. M., van de Klundert, J. J., and Wagelmans, A. P. (2014). Kidney exchange with long chains: An efficient pricing algorithm for clearing barter exchanges with branch-and-price. *Manufacturing & Service Operations Management*, 16(4):498–512.

[Guillaume and Latapy, 2004] Guillaume, J.-L. and Latapy, M. (2004). Bipartite structure of all complex networks. *Information Processing Letters*, 90(5):215 – 221.

[Gusfield and Irving, 1989] Gusfield, D. and Irving, R. W. (1989). *The Stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press.

[Hebrard, 2008] Hebrard, E. (2008). Mistral, a Constraint Satisfaction Library. In *Proceedings of the CP-08 Third International CSP Solvers Competition*, pages 31–40.

[Hebrard et al., 2004a] Hebrard, E., Hnich, B., and Walsh, T. (2004a). Robust solutions for constraint satisfaction and optimization. In *ECAI*, volume 16, page 186.

[Hebrard et al., 2004b] Hebrard, E., Hnich, B., and Walsh, T. (2004b). Super solutions in constraint programming. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 157–172. Springer.

[Hebrard et al., 2010] Hebrard, E., O'Mahony, E., and O'Sullivan, B. (2010). Constraint programming and combinatorial optimisation in numberjack. In *CPAIOR 2010, Bologna, Italy, 2010. Proceedings*, pages 181–185.

[Hebrard and Walsh, 2005] Hebrard, E. and Walsh, T. (2005). Improved algorithm for finding (a, b)-super solutions. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005*, pages 848–855.

[Holland and O'Sullivan, 2004] Holland, A. and O'Sullivan, B. (2004). Super solutions for combinatorial auctions. In *Recent Advances in Constraints, Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004, Lausanne, Switzerland, June 23-25, 2004, Revised Selected and Invited Papers*, pages 187–200.

[Holland and O'Sullivan, 2005a] Holland, A. and O'Sullivan, B. (2005a). Robust solutions for combinatorial auctions. In *Proceedings 6th ACM Conference on Electronic Commerce (EC-2005), Vancouver, BC, Canada, June 5-8, 2005*, pages 183–192.

[Holland and O'Sullivan, 2005b] Holland, A. and O'Sullivan, B. (2005b). Weighted super solutions for constraint programs. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 378–383.

[Holland and O'Sullivan, 2007] Holland, A. and O'Sullivan, B. (2007). Truthful risk-managed combinatorial auctions. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1315–1320, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Hooker, 2016] Hooker, J. (2016). Logic-based benders decomposition tutorial. In *http://public.tepper.cmu.edu/jnh/bendersTutorial2016.pdf*.

[Hooker, 2007] Hooker, J. N. (2007). Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3):588–602.

[Hooker and Ottosson, 2003] Hooker, J. N. and Ottosson, G. (2003). Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60.

[Hooker and Williams, 2012] Hooker, J. N. and Williams, H. P. (2012). Combining equity and utilitarianism in a mathematical programming model. *Management Science*, 58(9):1682–1693.

[Huang and Kavitha, 2013] Huang, C. and Kavitha, T. (2013). Popular matchings in the stable marriage problem. *Inf. Comput.*, 222:180–194.

[Huang et al., 2011] Huang, C.-C., Kavitha, T., Michail, D., and Nasre, M. (2011). Bounded unpopularity matchings. *Algorithmica*, 61(3):738–757.

[Irving, 1985] Irving, R. W. (1985). An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6(4):577–595.

[Irving, 2007a] Irving, R. W. (2007a). The cycle roommates problem: a hard case of kidney exchange. *Information Processing Letters*, 103(1):1–4.

[Irving, 2007b] Irving, R. W. (2007b). The cycle roommates problem: a hard case of kidney exchange. *Information Processing Letters*, 103(1):1 – 4.

[Irving et al., 2006] Irving, R. W., Kavitha, T., Mehlhorn, K., Michail, D., and Paluch, K. E. (2006). Rank-maximal matchings. *ACM Transactions on Algorithms*, 2(4):602–610.

[Irving et al., 2008] Irving, R. W., Manlove, D. F., and Scott, S. (2008). The stable marriage problem with master preference lists. *Discrete Applied Mathematics*, 156(15):2959 – 2977.

[Kavitha et al., 2011] Kavitha, T., Mestre, J., and Nasre, M. (2011). Popular mixed matchings. *Theoretical Computer Science*, 412(24):2679–2690.

[Kavitha and Nasre, 2009] Kavitha, T. and Nasre, M. (2009). Optimal popular matchings. *Discrete Applied Mathematics*, 157(14):3181–3186.

[Kavitha and Nasre, 2011] Kavitha, T. and Nasre, M. (2011). Popular matchings with variable item copies. *Theor. Comput. Sci.*, 412(12-14):1263–1274.

[Kavitha et al., 2014] Kavitha, T., Nasre, M., and Nimbhorkar, P. (2014). Popularity at minimum cost. *J. Comb. Optim.*, 27(3):574–596.

[Klimentova et al., 2014] Klimentova, X., Alvelos, F., and Viana, A. (2014). A new branch-and-price approach for the kidney exchange problem. In *Proc. of ICCSA*, pages 237–252. Springer.

[Klimentova et al., 2016] Klimentova, X., Pedroso, J. P., and Viana, A. (2016). Maximising expectation of the number of transplants in kidney exchange programmes. *Computers & Operations Research*, 73:1–11.

[Knuth, 1976] Knuth, D. (1976). Mariages stables. 1976. *Les Presses de l'Universite de Montreal, Montreal*.

[Kuhn and Yaw, 1955] Kuhn, H. W. and Yaw, B. (1955). The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97.

[Kujansuu et al., 1999] Kujansuu, E., Lindberg, T., and Mäkinen, E. (1999). The stable roommates problem and chess tournament pairings. *Divulgaciones Matemáticas*, 7(1):19–28.

[Kwon et al., 1999] Kwon, O. J., Kwak, J. Y., Lee, K. S., Kang, C. M., and Park, H. Y. (1999). Exchange-donor program in renal transplantation: A single center experience. *Journal of the Korean Surgical Society*, 57(6):789–796.

[Lovász and Plummer, 1986] Lovász, L. and Plummer, M. (1986). *Matching Theory (Annals of Discrete Math Volume 29)*. North-Holland, Amsterdam.

[Mak-Hau, 2015] Mak-Hau, V. (2015). On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. *Journal of Combinatorial Optimization*, pages 1–25.

[Manlove, 2008] Manlove, D. (2008). Hospitals/residents problem. In *Encyclopedia of Algorithms*.

[Manlove et al., 2007] Manlove, D., O'Malley, G., Prosser, P., and Unsworth, C. (2007). A constraint programming approach to the hospitals / residents problem. In *Proceedings of CPAIOR, 4th International Conference, 2007, Brussels, Belgium*, pages 155–170.

[Manlove and Sng, 2006] Manlove, D. and Sng, C. T. S. (2006). Popular matchings in the capacitated house allocation problem. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 492–503.

[Manlove, 2005] Manlove, D. F. (2005). Modelling and solving the stable marriage problem using constraint programming. In *In Proceedings of the 5th Workshop on Modelling and Solving Problems with Constraints, held at IJCAI '05*, pages 10–17.

[Manlove, 2013] Manlove, D. F. (2013). *Algorithmics of matching under preferences*. World Scientific.

[Manlove and O'Malley, 2014] Manlove, D. F. and O'Malley, G. (2014). Paired and altruistic kidney donation in the UK: algorithms and experimentation. *ACM Journal of Experimental Algorithmics*, 19(1):271–282.

[Manlove et al., 2006] Manlove, D. F., Sng, C. T. S., Manlove, D. F., and Sng, C. T. S. (2006). Uk popular matchings in the capacitated house allocation problem.

[Martello and Toth, 1990] Martello, S. and Toth, P. (1990). An exact algorithm for large unbounded knapsack problems. *Operations research letters*, 9(1):15–20.

[Matsui and Hamaguchi, 2016] Matsui, T. and Hamaguchi, T. (2016). Characterizing a set of popular matchings defined by preference lists with ties. *CoRR*, abs/1601.03458.

[Mattei and Walsh, 2013] Mattei, N. and Walsh, T. (2013). Preflib: A library of preference data HTTP://PREFLIB.ORG. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT)*, Lecture Notes in Artificial Intelligence. Springer.

[McCutchen, 2008] McCutchen, R. M. (2008). The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In *Latin American Symposium on Theoretical Informatics*, pages 593–604. Springer.

[McDermid and Irving, 2009] McDermid, E. and Irving, R. W. (2009). Popular matchings: Structure and algorithms. In *Computing and Combinatorics, 15th Annual International Conference, COCOON 2009, Niagara Falls, NY, USA, 2009, Proceedings*, pages 506–515.

[Mestre, 2006] Mestre, J. (2006). Weighted popular matchings. In *International Colloquium on Automata, Languages, and Programming*, pages 715–726. Springer.

[NRMP, 2019] NRMP (2019). NRMP National Resident Matching Program. Accessed: 2019-09-30.

[Numberjack,] Numberjack, V. . https://github.com/eomahony/numberjack.

[Pagnoncelli et al., 2009] Pagnoncelli, B., Ahmed, S., and Shapiro, A. (2009). Sample average approximation method for chance constrained programming: theory and applications. *Journal of optimization theory and applications*, 142(2):399–416.

[Park et al., 1999] Park, K., Moon, J. I., Kim, S. I., and Kim, Y. S. (1999). Exchange donor program in kidney transplantation1. *Transplantation*, 67(2):336–338.

[Pedroso, 2014] Pedroso, J. P. (2014). Maximizing expectation on vertex-disjoint cycle packing. In *International Conference on Computational Science and Its Applications*, pages 32–46. Springer.

[Plaut et al., 2016] Plaut, B., Dickerson, J. P., and Sandholm, T. (2016). Fast optimal clearing of capped-chain barter exchanges. In *Proc. of AAAI*, pages 601–607.

[Powell, 2016] Powell, W. B. (2016). A unified framework for optimization under uncertainty. In *Optimization Challenges in Complex, Networked and Risky Systems*, pages 45–83. INFORMS.

[Prosser, 2014] Prosser, P. (2014). Stable roommates and constraint programming. In *CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings*, pages 15–28.

[Régin, 1996] Régin, J. (1996). Generalized Arc Consistency for Global Cardinality Constraint. In *AAAI96*, pages 209–215.

[Régin, 1994] Régin, J.-C. (1994). A filtering algorithm for constraints of difference in csps. In *AAAI*, volume 94, pages 362–367.

[Rossi et al., 2006] Rossi, F., Van Beek, P., and Walsh, T. (2006). *Handbook of constraint programming*. Elsevier.

[Roth, 2008] Roth, A. E. (2008). What have we learned from market design? *Innovations: Technology, Governance, Globalization*, 3(1):119–147.

[Roth et al., 2004] Roth, A. E., Sönmez, T., and Ünver, M. U. (2004). Kidney exchange. *The Quarterly Journal of Economics*, 119(2):457–488.

[Roth et al., 2005] Roth, A. E., Sönmez, T., and Ünver, M. U. (2005). Pairwise kidney exchange. *Journal of Economic theory*, 125(2):151–188.

[Roth et al., 2007] Roth, A. E., Sönmez, T., and Ünver, M. U. (2007). Efficient kidney exchange: Coincidence of wants in markets with compatibility-based preferences. *American Economic Review*, 97(3):828–851.

[Roy, 2001] Roy, A. (2001). *Symmetry Breaking and Fault Tolerance in Boolean Satisfiability*. PhD thesis. University of Oregon.

[Roy, 2006] Roy, A. (2006). Fault tolerant boolean satisfiability. *Journal of Artificial Intelligence Research*, 25:503–527.

[Said, 2012] Said, M. (2012). Auctions with dynamic populations: Efficiency and revenue maximization. *Journal of Economic Theory*, 147(6):2419–2438.

[Saidman et al., 2006] Saidman, S. L., Roth, A. E., Sönmez, T., Ünver, M. U., and Delmonico, F. L. (2006). Increasing the opportunity of live kidney donation by matching for two-and three-way exchanges. *Transplantation*, 81(5):773–782.

[Santos et al., 2017] Santos, N., Tubertini, P., Viana, A., and Pedroso, J. P. (2017). Kidney exchange simulation and optimization. *Journal of the Operational Research Society*, 68(12):1521–1532.

[Shapiro et al., 2014] Shapiro, A., Dentcheva, D., and Ruszczynski, A. (2014). *Lectures on Stochastic Programming: Modeling and Theory, Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[Sng and Manlove, 2010] Sng, C. T. and Manlove, D. F. (2010). Popular matchings in the weighted capacitated house allocation problem. *Journal of Discrete Algorithms*, 8(2):102 – 116.

[Stoler et al., 2017] Stoler, A., Kessler, J. B., Ashkenazi, T., Roth, A. E., and Lavee, J. (2017). Incentivizing organ donor registrations with organ allocation priority. *Health Economics*, 26(4):500–510.

[Thiel et al., 2001] Thiel, G., Vogelbach, P., Gürke, L., Gasser, T., Lehmann, K., Voegele, T., Kiss, A., and Kirste, G. (2001). Crossover renal transplantation: hurdles to be cleared! In *Transplantation proceedings*, volume 1, pages 811–816.

[Ünver, 2010] Ünver, M. U. (2010). Dynamic kidney exchange. *The Review of Economic Studies*, 77(1):372–414.

[Van Hentenryck and Bent, 2009] Van Hentenryck, P. and Bent, R. (2009). *Online stochastic combinatorial optimization*. The MIT Press.

[van Hoeve and Katriel, 2006] van Hoeve, W. and Katriel, I. (2006). Global constraints. handbook of constraint programming.

[Wallis et al., 2011] Wallis, C. B., Samy, K. P., Roth, A. E., and Rees, M. A. (2011). Kidney paired donation.

[Weigel et al., 1998] Weigel, R., Bliek, C., and Faltings, B. V. (1998). On reformulation of constraint satisfaction problems (extended version). *Artificial Intelligence. Citeseer*.