An Approach to Robustness in Stable Marriage and Stable Roommates Problems

Begüm Genç
MSC
114220501

NATIONAL UNIVERSITY OF IRELAND, CORK

COLLEGE OF SCIENCE, ENGINEERING AND FOOD SCIENCE

SCHOOL OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

INSIGHT CENTRE FOR DATA ANALYTICS

Thesis submitted for the degree of Doctor of Philosophy

June 2019

Head of Department: Professor Cormac J. Sreenan

Supervisors: Professor Barry O'Sullivan
Dr Mohamed Siala

Contents

List of Figures

List of Tables

I, Begüm Genç, certify that this thesis is my own work and I have not obtained a degree in this university or elsewhere on the basis of the work submitted in this thesis.

Begüm Genç

To those who accompanied me throughout this process with special mention to my mother for always being my inspiration.

Acknowledgements

Firstly, I wish to express my eternal gratitude to my main supervisor Professor Barry O'Sullivan for his time, patience, and unwavering support. I feel incredibly fortunate to have received his guidance and the valuable opportunity he provided for me. He has not only been a great supervisor but also a positive mentor.

I also would like to extend my sincere thanks to my other supervisors Dr. Mohamed Siala, and to Dr. Gilles Simonin for their patience and support. They have been outstanding advisors and great friends to me. Without them, the task of attaining this degree would not be possible. I am also grateful to my examiners Professor James Gleeson and Dr. John Herbert for their evaluation and corrections.

I have thoroughly enjoyed my time at the Insight Centre. I feel privileged to have worked alongside such consummate professionals, many of which I am delighted to call my life-long friends. I sincerely thank all the help provided by the administrative staff Caitriona, Eleanor, and Linda.

Needless to say, my family's support has been invaluable. I especially would like to thank my brother, Burkay, for helping me identify this career path, assisting me the whole time, and playing a key role for me to get this opportunity at Insight. I cannot express enough gratitude my mother, Solmaz, who was a constant source of comfort to me and made sure that I am progressing each day. To my grandmother Saibe, sister-in-law Selen, niece Arya, and nephew Orkun Efe; thank you for always putting a smile on my face. Finally, to my father Mustafa, who gave me his blessing to move to Ireland to further my career. I am grateful to my true friends Neslin, Ozan, and Ozlem for their trips to come see me in Ireland, and bringing a piece of home with them. Last but not least, I wholeheartedly thank Brian, without whose support this section would not be completed.

This dissertation would not have been possible without the financial support of the Science Foundation Ireland Grant No. 12/RC/2289 which is co-funded under the European Regional Development Fund.

Abstract

This dissertation focuses on a novel concept of robustness within the context of matching problems. Our robustness notion for the stable matching framework is motivated by the unforeseen events that may occur after a matching is computed. We define the notion of (a, b)-supermatches as a measure of robustness of a matching. An (a, b)-supermatch characterizes a stable matching such that if any combination of a pairs want to leave the matching, there exists an alternative matching in which those a pairs are assigned new partners, and in order to obtain the new assignment at most b other pairs are broken.

We first formally define the notion of (a, b)-supermatches by using one of the most famous matching problems, namely the Stable Marriage problem (SM), as the platform. We name the problem of finding an (a, b)-supermatch to the SM as the Robust Stable Marriage problem (RSM). Subsequently, we prove that RSM is $\mathcal{NP}$-hard, and the decision problem for the case where $a = 1$ (i.e. deciding if there exists a $(1, b)$-supermatch) is $\mathcal{NP}$-complete. We also develop a constraint programming model and a number of meta-heuristic approaches to find a $(1, b)$-supermatch that minimizes the value of b for the RSM.

Following the results on the RSM, we extend the notion of (a, b)-supermatches to the Stable Roommates problem (SR), namely, the Robust Stable Roommates problem (RSR). We show that the $\mathcal{NP}$-hardness is also valid for the RSR, and we also define a polynomial-time procedure for the RSR to decide if a given stable matching is a $(1, b)$-supermatch. Similarly, we provide a number of meta-heuristic models to solve the optimization problem for finding a $(1, b)$-supermatch that minimizes the value of b. We conclude this dissertation by providing some empirical results on the robustness of different datasets of RSM and RSR instances.

Chapter 1

Introduction

"Change is the only constant."

– Heraclitus, *The Greek Philosopher*

1.1 Motivation

It is essential to build robust systems that can be repaired by only minor changes in case of an unforeseen event [Sus07]. The *Hospitals/Residents problem (HR)* (often referred as the College Admissions problem) is a popular matching problem. An instance of HR is defined by a set of residents $R = \{r_1, \ldots, r_n\}$ and a set of *hospitals* $H = \{h_1, \ldots, h_n\}$. All residents and hospitals express their strictly ordered preferences over the agents of the other set. Each hospital h_i has a positive integer value c_i indicating its capacity. A solution to an HR instance is an assignment between the residents and the hospitals such that each resident is assigned to at most one hospital, none of the hospitals are assigned more residents than their capacities, and the matching is stable. The stability in this context refers to having an assignment that has no blocking pairs. A pair (r_i, h_j) is said to block a matching if:

- A resident r_i is unassigned or prefers h_j to his/her current assignment;
- The hospital h_j is under-subscribed or prefers r_i to at least one of the assigned residents.

Figure 1.1 presents an instance of HR that consists of nine residents, represented $R = \{r_1, r_2, \ldots, r_9\}$, and three hospitals, $H = \{h_1, h_2, h_3\}$, where the

capacities of the hospitals are $c_1 = 2, c_2 = 4, c_3 = 3$, and an example stable matching between the residents and the hospitals. In this figure, the residents r_1, r_2 stay in hospital h_1, r_3, r_4, r_5, r_6 stay in h_2, and r_7, r_8, r_9 stay in h_3.

Assume that the matching between the hospitals and the residents (medical students) given in the Figure 1.1 is stable. After analysing this matching, it is easy to see that all hospitals are working at full-capacity. A huge problem arises if one of the residents can no longer stay in his/her assigned hospital. The reasons behind this request can be for reasons such as: the resident is experiencing a problem with one of the agents serving in the hospital; is not happy with the management of the hospital; or his/her transportation to the place is not possible anymore; etc. In a similar way, the hospital may request him/her to be transferred to another hospital because they are not happy with his/her performance, they do not have sources to supply him/her anymore, etc. We refer to such causes as *unforeseen events*, where the preferences are expressed correctly at the beginning of the assignment, but the matching becomes infeasible as the time progresses.

Given the stable matching in Figure 1.1, assume that, due to an unexpected event, r_3 must be relocated to another hospital. In this case, a relocation of at least one other medical student is required because all the hospitals are full.

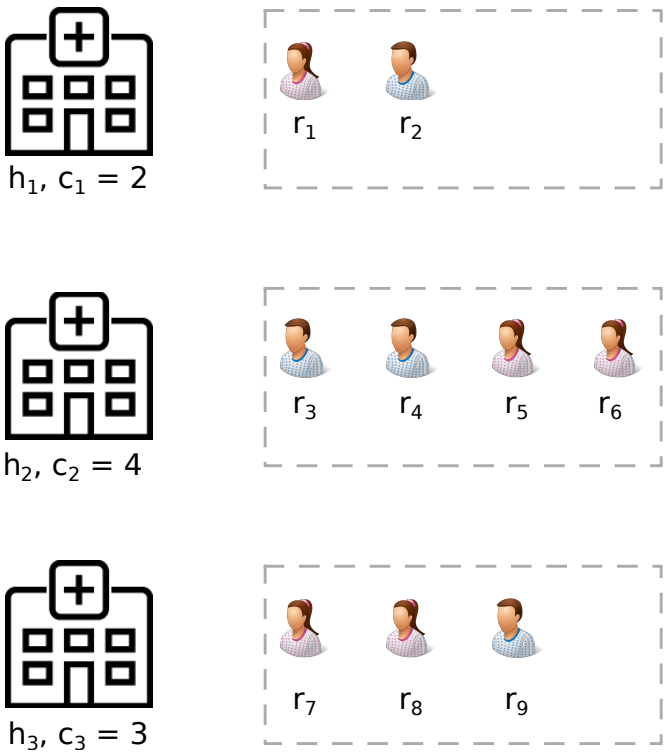


Figure 1.1: An HR instance of 9 residents and 3 hospitals.

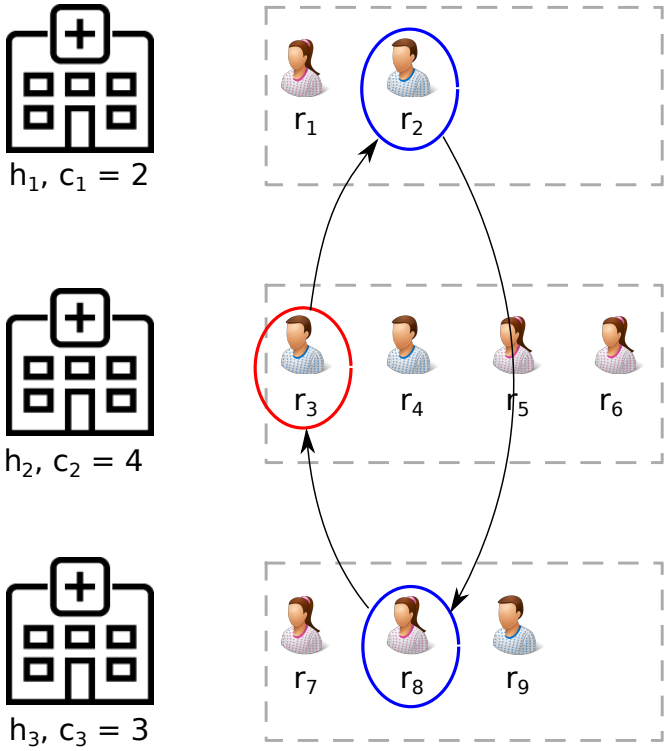


Figure 1.2: A possible relocation on the HR instance illustrated in Figure 1.1.

Figure 1.2 illustrates another possible stable assignment after applying the relocations required to obtain another solution. The relocations include moving the resident r_3 from h_2 to h_1, moving r_2 from h_1 to h_3, and r_8 from h_3 to h_2.

An interesting property of this problem is that, the set of residents that are assigned to hospitals in a stable matching is exactly the same set of residents in any other stable matching of the underlying instance. Additionally, note that, there is a cost associated with the relocations of the residents. Therefore, in the case of unforeseen events, in order to reduce the extra cost, it is desirable to find another assignment with the minimum number of additional changes. We refer to these matchings (solutions) that require only a small number of modifications if some assignments break as *robust solutions*. The main motivation of studying robust solutions is to produce solutions that ideally do not require any modifications in the case of an unforeseen event, or solutions that provide a bound on the cost for the repairs.

This notion is not only important in the context of Hospital/Residents, but also important in the context of other stable matching problems such as Stable Marriage, Ride Sharing, Kidney Exchange, Stable Roommates, etc [Man13]. Some of these problems arise in many real-world settings. For instance, cloud computing can be modelled as a Stable Marriage problem, or peer-to-peer applications

can be modelled as a Stable Roommates problem [DPK13, LMV+07].

For a second motivating example, consider the Paper/Reviewer assignment problem. Some conferences or journals receive hundreds of submissions after announcing the call for papers. In most of the cases, the authors submit the keywords related to their work. The reviewers declare their conflicts and/or express their preferences over the papers, or the topics. After the preferences are provided, the organizers are responsible for assigning the papers to the reviewers. It is possible that after the assignments are completed, a reviewer may ask to change his/her assigned paper due to lack of confidence in the field or the complexity of the paper, etc. In this case, considering that some reviewers may already have started working on their assigned papers, an ideal solution would be the one that can accommodate the change requests by reflecting very limited disturbance on the other people. This ideal solution is a robust matching.

The work presented in this dissertation considers the application of robustness into various matching problems. In Section 1.2, we present an outline of all contributions of this thesis related to the proposed robustness notion.

1.2 Thesis Statement and Contributions

In this dissertation we focus on introducing a novel robustness notion for matching problems. We state the thesis defended in this dissertation and present our approach, by also providing a list of our supporting publications for each point.

Thesis. *Matching problems are widely-studied computational problems that require assigning agents to one another under different optimality criteria. The stability criterion in matching problems is well defined and dominantly used. However, imposing only the stability constraint on matchings is not enough on its own when the dynamism of the real world due to unexpected events is considered. Therefore, the need to consider a notion of robustness in addition to the existing stability constraint emerges. We claim that achieving both stability and robustness is possible. We propose a novel concept of robustness that has not been considered in this field before. By defining robust stable matchings we allow systems to handle unexpected events while making a bounded number of changes, after the matchings have been constructed.*

Discussion. In Chapter 3, we introduce a novel robustness notion for the matching problems called (a, b)-supermatches that uses a fault tolerance framework

and represents matchings that are stable and can handle unexpected events. This notion is inspired by previous work on fault tolerance and robustness notions represented as (a, b)-supermodels in SAT and (a, b)-super solutions in CSP [GPR98, HHW04b]. We introduce the (a, b)-supermatches concept within the context of the Stable Marriage problem (SM) and call the problem of finding a robust solution to an SM instance as the Robust Stable Marriage problem (RSM). The first appearance of our work related to the proposal of the novel concept can be found in the following conference publication [GSOS17c]:

> Begum Genc, Mohamed Siala, Gilles Simonin and Barry O'Sullivan. Robust Stable Marriage. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI 2017*, pages 4925-4926, 2017.

Next, we focus on the complexity of the RSM and prove that although it is easy to find a stable matching, it is not easy to find robust stable ones (i.e. (a, b)-supermatches). We prove in Chapter 3 that finding a $(1, b)$-supermatch to a given Stable Marriage instance is $\mathcal{NP}$-complete. A preliminary version of this research excluding the proofs appears in the following conference publication [GSSO17a]:

> Begum Genc, Mohamed Siala, Gilles Simonin and Barry O'Sullivan. On the Complexity of Robust Stable Marriage. In *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017*, pages 441-448, 2017. doi: 10.1007/978-3-319-71147-8_30.

The proofs of this publication, including some additional examples, is publicly available as a technical report as follows [GSSO17b]:

> Begum Genc, Mohamed Siala, Gilles Simonin, Barry O'Sullivan: On the Complexity of Robust Stable Marriage. Technical report as e-Print on arXiv: 1709.06172 (October 2017). URL: arxiv.org/abs/1709.06172.

Then, we extend this work by providing detailed proofs as well as an independent set formulation for finding a $(1, 1)$-supermatch for any given SM instance. We provide some cases that can be solved in polynomial-time and also show that some cases need not exist at all. The details of the complexity study on RSM can be found in the following journal publication [GSSO19]:

> Begum Genc, Mohamed Siala, Gilles Simonin, Barry O'Sullivan. Complexity Study for the Robust Stable Marriage Problem. In *Theo-*

retical Computer Science, 2019. ISSN 0304-3975, doi: 10.1016/j.tcs.2018.12.017.

In order to show the $\mathcal{NP}$-completeness of the $(1, b)$-supermatches case, we define a polynomial-time procedure to decide if a given stable matching is a $(1, b)$-supermatch. Considering the difficulty of the RSM, we develop four different optimization models to find a $(1, b)$-supermatch that minimizes the b to a given RSM instance. We also provide a performance comparison of these different models on randomly generated SM instances. These findings are presented in Chapter 4. A part of this work including three of the four models appears in the following conference publication [GSOS17a]:

> Begum Genc, Mohamed Siala, Gilles Simonin and Barry O'Sullivan. Finding Robust Solutions to Stable Marriage. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 631-637, 2017. doi: 10.24963/ijcai.2017/88.

A technical report on this work that includes details of the proposed algorithms and examples for the methods is also publicly available [GSOS17b]:

> Begum Genc, Mohamed Siala, Gilles Simonin and Barry O'Sullivan. Finding Robust Solutions to Stable Marriage. Technical report as e-Print on arXiv: 1705.09218 (August 2017). URL: arxiv.org/abs/1705.09218.

We conclude our work by showing that (a, b)-supermatches are not only meaningful in the context of Stable Marriage problem, but can also be applied to other matching problems. In order to show that, we extend the notion of robustness into a different matching problem, namely the Stable Roommates problem (SR), and call the robust version as *Robust Stable Roommates* problem (RSR). We provide a polynomial-time procedure to find $(1, b)$-supermatches for the RSR. We develop two different meta-heuristic approaches to find $(1, b)$-supermatches to RSR and we provide a comparison of them. Subsequently, we perform some experiments that investigate the robustness of different datasets of RSM and RSR. We present these findings in Chapter 5, and the details can be found in the following publication:

> Begum Genc, Mohamed Siala, Gilles Simonin and Barry O'Sullivan. An Approach to Robustness in the Stable Roommates Problem and its Comparison with the Stable Marriage Problem. In *CPAIOR 2019, in press*.

1.3 Overview of the Dissertation

This dissertation consists of four main chapters. We present briefly the structure and the contents of each chapter below.

Chapter 2 presents the necessary background information to understand the concepts and correctly follow the notation used in this dissertation. The main purpose of this chapter is to define all the concepts in detail and also provide examples for the reader to help them understand the concepts before moving to the technical part of the dissertation. We start with an introduction to the mathematical structures widely being used throughout this dissertation such as graphs and partially-ordered sets. Then, we formally introduce what combinatorial optimization is and some of the popular techniques that can be used to model a given combinatorial optimization problem. The discussed techniques are limited to the ones we are using in this dissertation. Following the techniques, we present in detail the two combinatorial optimization problems we focus in this dissertation: the Stable Marriage problem and Stable Roommates problem. We also give a brief introduction to robustness in optimization. Consequently, we present the two robustness models that our work is based on: (a, b)-supermodels in Boolean Satisfiability [GPR98], and (a, b)-super solutions in Constraint Satisfaction Problems [HHW04b]. We conclude this section by reviewing the existing robustness notions within the matching problems.

Chapter 3 focuses on two main points: the formal definition of our novel notion of robustness and the complexity of the Stable Marriage problem when robustness is included. We first define our notion of (a, b)-supermatches and introduce the problem of Robust Stable Marriage (RSM). Then, we present a model using an independent set formulation as a representation for the restricted case of the RSM, i.e. $(1, 1)$-supermatches. We also provide another model, which is a special case of SAT with specific constraints imposed on its clauses, called SAT-SM. We show that SAT-SM is $\mathcal{NP}$-complete by Schaefer's Dichotomy theorem. Then, we use this special SAT formula to show that a restricted case of our proposed problem, $(1, b)$-supermatches is $\mathcal{NP}$-complete. The work in this chapter also includes identification of some polynomial cases as well as some non-existing cases for RSM.

Chapter 4 presents a polynomial-time procedure to decide if a given stable matching is a $(1, b)$-supermatch or not. Consequently, four different models are proposed based on this procedure. The models solve the optimization problem

for RSM, i.e. to find the $(1, b)$-supermatch that minimizes the value of b. These models include: a complete search technique (Constraint Programming), and three meta-heuristics (Genetic Algorithm, Iterated Local Search, and Genetic Local Search). We conclude this section by presenting some experiments to observe the relative performances of the models, and the effects of parameters on the search, time efficiency.

Chapter 5 is mainly an extension of the Chapter 3 and Chapter 4. We define our robustness concept, (a, b)-supermatches in Stable Roommates problem and call the robust version of the problem as Robust Stable Roommates (RSR). The work in this chapter consists of a formal definition of the proposed problem, followed by a polynomial time procedure for deciding if a given stable matching is a $(1, b)$-supermatch. Subsequently, we present two meta-heuristic procedures that are using the polynomial time procedure. Then, we perform experiments to compare the performances of these two models. We also create some interesting datasets for the RSM and the RSR problems. We conclude this chapter by comparing the behaviour of robustness values of the two aforementioned problems.

Lastly, in Chapter 6, we discuss some general concluding remarks, open problems and identify some future directions related to the work presented in this dissertation.

Chapter 2

Background

Abstract. *In this chapter, we present the background and notation required to present our work on robustness in stable matching problems. First, we introduce some mathematical structures, define combinatorial optimization, as well as review a number of techniques for solving those problems. Subsequently, existing robustness notions in the literature, mainly in CP, SAT and matching problems, is discussed.*

2.1 Mathematical Structures

In this section, we introduce two of the mathematical structures: graphs and partially ordered sets.

2.1.1 Graphs

A *graph* is an abstracted structure for modelling and graphically representing the relations between objects. In mathematics, this is a field called *graph theory* [Wil86, Die10, BM11].

Formally, a graph $G = (V, E)$ is composed of a set of *vertices*, denoted by V, and a set of *edges*, denoted by E. Each edge $e \in E$ in a graph is represented by $e = (v_s, v_t)$, where $v_s, v_t \in V$. For each edge $e = (v_i, v_j)$ in the graph G, v_i and v_j are referred as *adjacent* vertices in G. Each adjacent vertex of v is a *neighbour* of v. The set of neighbours defines the *neighbourhood* of v.

A graph can be directed or undirected. If G is a *directed graph*, then its edges are directed, meaning there is an order between its vertices. For a directed edge $e = (v_s, v_t)$, the vertex v_s is called as the *initial vertex* and v_t is called as the *terminal vertex*. In this case, v_s is said to *precede* v_t. Additionally, e is referred as an incoming edge of v_t and an outgoing edge of v_s. On the other hand, if G is an *undirected graph*, the edges are undirected and there does not exist any precedence relation between its vertices. The directed graphs are suitable for representing arbitrary binary relations, whereas undirected graphs are suitable for modelling symmetric binary relations [SD08].

Figure 2.1 illustrates an example directed graph $G_d = (V_d, E_d)$ and an undirected graph $G_u = (V_u, E_u)$, where the graphs are defined by the following sets $V_d = V_u = \{1, 2, 3, 4, 5\}$, $E_d = \{(1, 2), (2, 4), (2, 5), (3, 5), (4, 3), (4, 5)\}$ and $E_u = \{\{1, 2\}, \{1, 3\}, \{1, 5\}, \{2, 4\}, \{2, 5\}, \{3, 5\}\}$.

Given a graph $G = (V, E)$ and a sequence of vertices $v_1, v_2, \ldots, v_n$ such that there exists an edge between each consecutive vertex pair $(v_i, v_{i+1}) \in E, i \in [1, n-1]$ represents a *path* on the graph. A path that starts and ends at the same vertex such that $v_1, v_2, \ldots, v_n$, where $v_1 = v_n$ is called a *cycle*.

The edges (or sometimes the vertices) of a graph can have additional associated information such as a label, weight, value, etc. For example Figure 2.2, illustrates a weighted directed graph with six vertices. The text on each vertex $(v_1, v_2, \ldots, v_6)$ denotes the label (unique identifier) of that vertex. Additionally, there is a weight associated with each edge. Let the weight of each edge $e_{ij} = (v_i, v_j)$ denoted by w_{ij}. For Figure 2.2, the weight of the edge (v_1, v_2) is $w_{12} = 4$, (v_6, v_3) is $w_{63} = 1$, etc.

The *degree* of a vertex v, denoted by $d(v)$, represents the number of vertices adjacent to v. If the graph is directed, then there are two types of degrees: in-degree and out-degree. For each vertex v, the number of incoming edges of

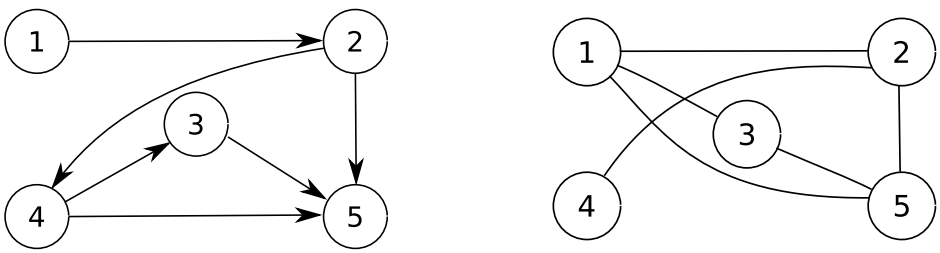


Figure 2.1: An illustration of a directed graph (left) and an undirected graph (right) where both have 5 vertices and 6 edges.

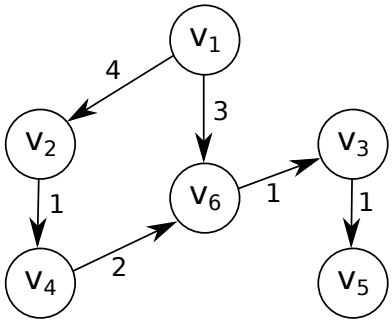


Figure 2.2: A sample weighted directed graph composed of 6 vertices and 6 edges.

v denotes the *in-degree* of v denoted by $d^-(v)$, whereas the number of outgoing edges gives the *out-degree* of it, denoted by $d^+(v)$. For illustrative purposes, the degree of vertex v_3 in Figure 2.1 (right) is calculated as $d(3) = 2$. Moreover, for the directed case in Figure 2.1 (left) in-degree and out-degree of the vertex 4 are $d^-(4) = 1$ and $d^+(4) = 2$, respectively.

Each vertex v in a directed graph $G_d = (V_d, E_d)$ can have a number of predecessors and a number of successors defined by the incident edges of v in the given graph. If there exists a directed edge $e = (v_s, v_t)$, v_s is an *immediate predecessor* of the vertex v_t, and v_t is an *immediate successor* of v_s. Extending this concept, if there exists a path between any two vertices v_s and v_t that starts at v_s and ends at v_t, illustrated as: $v_s, v_i, \ldots, v_t$, then the vertex v_s is a *predecessor* of v_t and v_t is a successor of v_s. If a vertex v does not have any successors in the given graph (i.e. $d^+(v) = 0$), then it is referred as a *sink vertex*; if $d^-(v) = 0$ then it is a *source vertex*. The set of all immediate predecessors of v in G_d is denoted by $N^-(v)$ and the immediate successors by $N^+(v)$. We denote the sets of predecessors and successors as $N_t^-(v)$ and $N_t^+(v)$, respectively. For instance, in Figure 2.2, the immediate predecessors and successors of the vertex v_6 can be found as: $N^-(v_6) = \{v_1, v_4\}, N^+(v_6) = \{v_3\}$. Similarly, the sets of all predecessors and successors of the same vertex are identified as: $N_t^-(v_6) = \{v_1, v_2, v_4\}$, and $N_t^+(v_6) = \{v_3, v_5\}$.

Consider a graph $G = (V, E)$. A graph $G' = (V', E')$ denotes a *sub-graph* of G if $V' \subseteq V$ and $E' \subseteq E$ such that an edge $(v_i, v_j) \in E'$ only if $(v_i, v_j) \in E$. If all the vertices in the sub-graph G' has at least one incident edge in E', then G' defines an *induced sub-graph* of G on the edge set E'. We refer to edge-induced sub-graphs as *induced graph*.

Given a graph $G = (V, E)$, a set of edges $E_{sub} \subset E$ such that removal of those

edges from G results in partitioning the graph into two sub-graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and for each edge $e = (v_i, v_j) \in E_{sub}$, $v_i \in V_1$ and $v_j \in V_2$ then E_{sub} is called as *cut* on G.

Apart from the general definitions, there exist many different graph types. In this work we mainly focus on: weighted graphs, bipartite graphs and cyclic/a-cyclic graphs. We briefly describe the specific properties of these graph types below.

- If the vertices or edges of a graph have associated weights, then it is called a *weighted graph*.
- If the vertices of a graph can be divided into two disjoint sets such that there does not exist any edge between the vertices of the same set, and therefore each edge connects two vertices from different sets, then the graph is a *bipartite graph*.
- If the graph contains at least one cycle, then it is a *cyclic graph*.
- If there are no cycles in the graph, then it is an *acyclic* graph.

2.1.2 Partially Ordered Sets

Partially ordered sets are mathematical structures that are defined over some relations between the objects of the set. Most of the definitions in this section are based on the definitions used by Simovici and Djeraba [SD08]. Formally, a partial order on a set S is a relation $\rho \subseteq S \times S$ that is *reflexive*, *antisymmetric*, and *transitive*. A *partially ordered set* or *poset* $\Pi(S)$ consists of a set S under the relation $\leq$ between all the elements $a, b, c \in S$ has the following properties:

- $a \leq a$ (reflexivity).
- $a \leq b$ and $b \leq a$ implies $a = b$ (anti-symmetry).
- $a \leq b$ and $b \leq c$ implies $a \leq c$ (transitivity).

The inverse of a partial order is also a partial order on the same set. If the number of objects in the set $|S|$ is finite, the poset $(S, \leq)$ is referred as a finite poset.

Let $(S, \leq)$ be a poset. Any two elements x, y of S are *incomparable* if neither $x \leq y$ nor $y \leq x$. If two elements are not incomparable, then they are said to be *comparable*. A chain on a poset is defined in the following way: for any subset

of elements T of S such that for every $x, y \in T$ where $x \neq y$, either $x < y$ or $y < x$ (i.e. x and y are comparable), T represents a *chain*. Any chain $(T, \leq)$ in the poset $(S, \leq)$ such that T is a subset of S, is a maximal chain if there does not exist any other chain in S that is a superset of T. For any subset T of S such that for every $x, y \in T$, x and y are incomparable, then T represents an *anti-chain*.

Finite posets are often represented by *diagrams*. When a poset is drawn by a diagram, each element in the set is represented as a point and each relation is represented as a segment (solid line) and if for two elements b covers a such as $a, b \in S$, $a \leq b$, then a appears below b in the drawing. Any such diagram is also referred as a *Hasse diagram*, named after Helmut Hasse [Bir48]. For instance, given the subset relation $\subseteq$ in sets, the Hasse diagram of a poset $(\Pi(S), \subseteq)$, where $S = \{a, b, c\}$ is given in Figure 2.3 [SD08].

The Hasse diagram can be represented as a graph. Let $(S, \leq)$ be a poset and $G = (V, E)$ be its corresponding graph. In this case, every element i in the set S corresponds to a vertex v_i in V. For each two element $i, j \in S$ such that $i \leq j$, there exists a directed edge between the corresponding vertices as $e = (v_i, v_j)$ in E. Then, the resulting graph G is a directed acyclic graph.

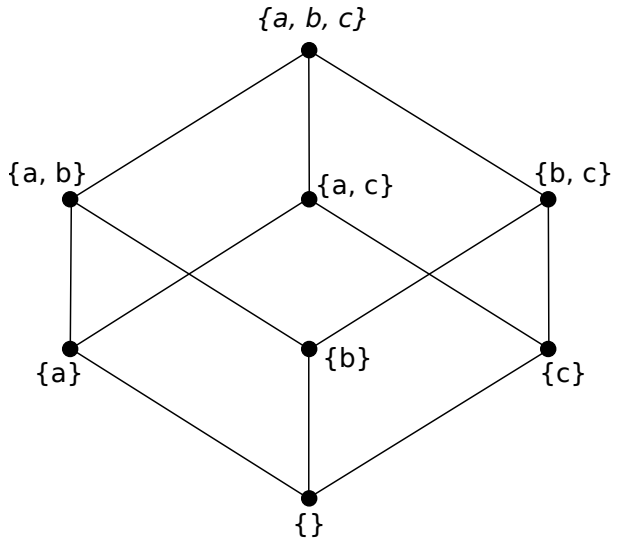


Figure 2.3: Hasse diagram of the poset $(\Pi(S), \subseteq)$, where $S = \{a, b, c\}$.

2.2 Combinatorial Optimization

There are many different classes of the optimization problems with tremendous differences between them [Ped03, Mur11]. Some examples of these classes can be listed as: linear optimization, non-linear optimization, combinatorial optimization, stochastic optimization, etc.

In optimization problems, the aim is to find the "best" solution to the given problem subject to a set of given criteria. *Combinatorial optimization* is a widely studied sub-category of the field of Optimization. Combinatorial optimization has the limitation of a finite, discrete set of feasible solutions [KV06]. In these types of problems, the search space is usually large and not very practical for applying complete search methods. Therefore, in some cases, incomplete search methods are favoured when compared to the complete search methods. Some well-known problems in this category can be formulated as the Job Shop Scheduling problem [YN97], Travelling Salesman problem (TSP) [ABCC07], Bin Packing problem [GJ79], Vehicle Routing problem [TVTV14], Matching Problems [Man13] etc.

As mentioned before, many of the combinatorial optimization problems are computationally hard to solve due to having large search spaces. A complete algorithm is said to *solve* a problem if the procedure always generates a solution for every possible instance of the problem if one exists, or reports that there exists no solution. The main intuition behind solving these problems is to find efficient algorithms that find the best element of some finite set of feasible solutions [KV06]. Graphs are often used to model the combinatorial optimization problems due to their combinatorial foundation.

The Travelling Salesman problem (TSP) is among one of the most famous combinatorial optimization problems. The objective is about finding the shortest tour among a given set of cities such that a salesperson returns to the city from which he starts his tour, by visiting each city exactly once. The underlying structure of the TSP is very suitable to be modelled as a graph. Thus, it is often modelled as a weighted graph with distances being the weights on the edges and strategies are developed based on finding the best path on the graph. Below is the formal definition as a graph model for TSP [MS91].

Definition 1 (TSP) *Given a directed graph $G = (V, E)$, of n vertices and a distance function, $d : V \times V \to \mathbb{R}^+ \cup \{\infty\}$ such that:*

$$d(v_i, v_j) = \begin{cases} D_{i,j} \in \mathbb{R}^+, & \textit{if } (v_i, v_j) \in E, \\ \infty \quad\quad\quad\quad , & \textit{if } (v_i, v_j) \notin E, \end{cases}$$

find a simple cycle of vertices: $v_{i0} \to v_{i1} \to \ldots \to v_{in} = v_{i0}$*, which minimizes*

$$\sum_{k=0}^{n-1} d(v_{ik}, v_{ik+1}).$$

An instance of the TSP can have exponentially many feasible solutions. Trying to compute and compare all possibilities would yield in $n!$ possible solutions, making it a complex problem to solve even for very modest sized instances. Therefore, researchers have worked for almost 60 years on tackling the problem using optimization techniques. For instance, many different meta-heuristic models are studied on the TSP such as ant colony optimization [SD99], local search [VT99], hybrid meta-heuristic algorithms such as genetic local search [FM96], tabu search [Fie94], etc. In the next sections, we look into the details of how a problem is classified as a complex or computationally hard to solve, and also which techniques can be used to solve them.

2.3 Optimization Modelling Languages and Techniques

There exist many different languages for modelling an optimization problem. Depending on the problem and its objective, one model can be superior to the other one. In this section, we explain two of these modelling paradigms: boolean satisfiability and constraint programming. Additionally, the strategies to solve combinatorial optimization problems differ depending on the model and nature of the problem. Two main approaches to be used are complete and incomplete search methods [Pre08]. We discuss in this section about the different search strategies for finding a solution to a given optimization problem.

Complete search methods guarantee finding the optimal solution to a given problem instance or report if none exists. On the other hand, incomplete search strategies do not guarantee that a solution will be found or that they report if none exists to the given combinatorial optimization problem. They may fail to find a solution to a satisfiable problem or fail to find the optimal solution

to an optimization problem. Incomplete search methods are used to find an "acceptable" solution to the given problem if not the exact optimal one [Pre08].

Metaheuristic algorithms define strategies to perform incomplete search [BR03]. They are iterative processes that guide the production of high-quality, near-optimal solutions. Some of the widely used metaheuristics can be listed as: Simulated Annealing, Tabu Search, Ant Colony Optimization (ACO), Iterated Local Search (LS), Genetic Algorithms (GA), etc [GP10]. These algorithms are generally inspired by the natural processes. For instance, the ACO algorithm mimics the behaviour of ants searching for food [DB10]. The pheromones released by the ants during this search leads other ants to certain paths, which eventually results in a valid path to food (i.e. solution). On the other hand, the GA is an evolutionary algorithm that mimics the evolution of a population by combination of genes and exposing mutations [B96].

One of the main challenge for metaheuristics is the risk of getting trapped at a local optimum during the convergence. Therefore the search may fail to reach the global optimum [Voß01]. This problem is often tackled by defining neighbourhoods of solutions and moving from a solution to another neighbour solution. Due to the neighbourhood, there is a trade-off between the localization and the diversification. The metaheuristic approaches apply different techniques to increase the diversity in the search space. As one would expect, a comprehensive survey on metaheuristics points out the fact that there is no final agreement on which metaheuristic model is more suitable for what type of problems, or which metaheuristic is superior to the others [Voß01]. The quality of the models usually depends on the quality of the assumptions made specific to the problem.

In this section, we only focus on the details of two metaheuristic algorithms in detail, which we use in our work. Those algorithms are: Genetic Algorithm and the Local Search. We also provide a Genetic Local Search model, which is a hybrid metaheuristic modelled as a combination of the Genetic Algorithm and the Local Search procedures. These models are discussed in detail in Sections 2.3.3, 2.3.4 and 2.3.5.

2.3.1 Boolean Satisfiability Problem

The *Boolean Satisfiability (SAT)* problem has its roots in mathematical logic, which is the basis of the modern computation. SAT was the first problem proved

$\mathcal{NP}$-complete by Cook [Coo71].

A *Boolean variable* can have only one of the two values: $true$ or $false$. An expression that is defined over Boolean variables and Boolean operators (and, or, xor, etc.) is called a *Boolean expression* or a *propositional logic formula*. It is often referred as "formula" in short. A *clause* refers to an expression that consists only of disjunctions (or conjunctions) of Boolean variables. Each variable in a clause represents a *literal*. Therefore, a literal either represents a positive Boolean variable or a negated Boolean variable. For the sake of giving an example, a disjunctive clause C_i has the form: $C_i = (x_0 \vee x_1 \vee \ldots \vee x_n)$ and a conjunctive clause: $C_j = (x_0 \wedge x_1 \wedge \ldots \wedge x_n)$. If any literal in C_i has the value $true$, then the clause C_i is evaluated to $true$. On the other hand, in order for a clause C_j to be evaluated to $true$, all the literals in C_j must have the value $true$. A clause C is referred as a *Horn clause*, if C is disjunctive with at most one unnegated (positive) literal. Moreover, a C is a *Dual-Horn clause*, if it is disjunctive and it contains at most one negated (negative) literal [Hor51].

There are a number of different forms of formulas. Let $C_i, i \in \{1, \ldots, n\}$, denote a number of conjunctive or disjunctive clauses. We define a number of different formula types:

- *Conjunctive Normal Form (CNF)*: A conjunction of one or more disjunctive clauses as: $(C_1 \wedge C_2 \wedge \ldots \wedge C_n)$.
- *Disjunctive Normal Form (DNF)*: A disjunction of one or more conjunctive clauses as: $(C_1 \vee C_2 \vee \ldots \vee C_n)$.
- *Horn Formula*: If every clause in a CNF contains at most one positive literal.
- *Dual-Horn Formula*: If every clause in a CNF contains at most one negative literal.
- *Affine Formula*: A conjunction of linear equations over a 2-element field, where a linear equation over the 2-element field is defined by Schaefer as an expression of the form $x_1 \oplus x_2 \ldots \oplus x_k = \delta$ where $\oplus$ is the sum modulo 2 operator and δ is 0 or 1 [Sch78].

An *assignment* is a mapping from Boolean variables to $\{true, false\}$. An assignment A is said to satisfy a clause C if and only if there exists a variable x such that C contains x and the assignment of x by A is $true$; or C contains $\neg x$ and the assignment of x by A is $false$. A *Boolean constraint of arity* k is a function

$\phi : \{true, false\}^k \rightarrow \{true, false\}$. Let $(x_1, \ldots x_k)$ be a sequence of Boolean variables and ϕ be a Boolean constraint of arity k. The pair $\langle \phi, (x_1, \ldots x_k) \rangle$ is called a *constraint application*. An assignment A to $(x_1, \ldots x_k)$ satisfies $\langle \phi, (x_1, \ldots x_k) \rangle$ if ϕ evaluates to $true$ on the truth values assigned by A. Let Φ be a set of constraint applications, and A be an assignment to all variables occurring in Φ. A is said to be a *satisfying assignment* of Φ if A satisfies every constraint application in Φ.

In order to distinguish polynomial-time solvable problems from the intractable cases, Schaefer published a remarkable dichotomy theorem [Sch78]. Let $\mathcal{C}$ be a set of Boolean constraints. $SAT(\mathcal{C})$ is defined as the following decision problem: Given a finite set Φ of constraint applications from $\mathcal{C}$, is there a satisfying assignment for Φ? As mentioned before, the Cook's theorem proves that the satisfiability problem, SAT, is $\mathcal{NP}$-complete. Furthermore, Schaefer's findings state that $SAT(\mathcal{C})$ is either in $\mathcal{P}$ or $\mathcal{NP}$-complete. The detailed theorem is given in Theorem 1.

Theorem 1 *Dichotomy Theorem for Satisfiability [DH09, Sch78]. Let $\mathcal{C}$ be a set of Boolean constraints. If $\mathcal{C}$ satisfies at least one of the conditions (a)-(f) below, then $SAT(\mathcal{C})$ is in $\mathcal{P}$. Otherwise, $SAT(\mathcal{C})$ is $\mathcal{NP}$-complete.*

a) Every constraint in $\mathcal{C}$ evaluates to $true$ if all assignments are $true$.

b) Every constraint in $\mathcal{C}$ evaluates to $true$ if all assignments are $false$.

c) Every constraint in $\mathcal{C}$ can be expressed as a Horn formula.

d) Every constraint in $\mathcal{C}$ can be expressed as a dual-Horn formula.

e) Every constraint in $\mathcal{C}$ can be expressed as affine formula.

f) Every constraint in $\mathcal{C}$ is definable by a CNF formula having at most 2 literals in each conjunct.

Although the SAT problem is $\mathcal{NP}$-complete in general, it is obvious that the 2SAT problem, which is a restricted case of SAT where each clause is a disjunction of at most 2 literals, is in $\mathcal{P}$ by Schaefer's Dichotomy Theorem. On the other hand, the version where each clause contains exactly 3 literals, namely the 3SAT problem, is $\mathcal{NP}$-complete.

2.3.2 Constraint Programming

Constraint Programming (CP) is a framework that is used to model problems as a set of decision variables and the relations between them. In this framework, the variables have discrete domains and the relations between variables are denoted by a finite number of *constraints*. We mainly use the notation used in the Handbook of Constraint Programming throughout this section [RvBW06], and also refer to the some recent studies for definitions [RN03, Sia15, CC16, vH01].

A CSP network is defined as a triple $\langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C} \rangle$. In this representation, the set $\boldsymbol{X} = \langle x_1, x_2, \ldots, x_n \rangle$ is an *n-tuple* denoting a finite set of distinct variables, and $\boldsymbol{D} = \langle D_1,\ D_2, \ldots, D_n \rangle$ is the domain of each variable such that $x_i \in D_i$. If the variable x_i is a Boolean variable, then the domain of x_i is denoted by $D_i = \{0, 1\}$. Otherwise, the domain of a variable is represented as a set $D_i = \{d_l, d_{l+1}, \ldots, d_u\}$, where d_l denotes the lower bound (i.e. the minimum value in the domain) and d_u denotes the upper bound, respectively. If $|D_i| = 1$, then the domain is said to be *singleton*.

The third set in the definition, $\boldsymbol{C} = \langle C_1, C_2, \ldots, C_t \rangle$ corresponds to the constraints of the problem. Each constraint C_i is represented as a pair $\langle R_{S_i}, S_i \rangle$, where $S \subset \boldsymbol{X}$ is the *scope* of the constraint and R_S denotes the relation containing all valid assignments of the variables in S. A *solution* to the CSP is an assignment to the n variables in $\boldsymbol{X}$ such that for each constraint C_i, the assignment of the variables in S_i falls into the range R_{S_i}. If there exists at least one such assignment, the CSP instance is said to have a solution, and is *satisfiable*. However, if the set of solutions is empty, the instance is *unsatisfiable*. Informally, the Constraint Satisfaction Problem is to decide whether a given constraint network is satisfiable or not.

As an example, assume we are given a set of two variables $\boldsymbol{X} = \langle x_1, x_2 \rangle$, where each variable has the same finite domain $D = \{1, 2, 3\}$ as $x_1, x_2 \in D$, and the constraint exposed on the system is the *not-equal* constraint (i.e. $x_1 \neq x_2$). Then the following assignments for the variables of the problem: $(x_1, x_2) = \{(1, 1), (2, 2), (3, 3)\}$, do not satisfy the constraint exposed on the network. Therefore, these assignments are not solutions. However, there exists another set of assignments where each of pair in the set makes the constraint satisfied. For instance, each pair in the following set: $(x_1, x_2) = \{(1, 2), (1, 3), (2, 1)\}$ represents a solution for the instance. Hence, having at least one solution, the

given constraint network is satisfiable.

The CSP is considered as a generalization of SAT [Coo71]. If all the variables in CSP are Boolean variables and each clause is considered as a constraint over the related variables, the resulting network represents a SAT instance. Therefore, the complexity result is carried and finding a solution to a given CSP problem is also $\mathcal{NP}$-hard. Moreover, for CSPs, the verification of a given solution is possible to be done in polynomial-time. Thus, deciding if a CSP has a solution is $\mathcal{NP}$-complete.

2.3.2.1 Search Strategies

In this section, we consider some of the systematic search techniques being used in CSPs. In order to find a solution to a CSP, the CSP is split into other smaller CSPs and a *search tree* is constructed from these smaller CSPs. Each node in this tree represents a CSP, where the root of the tree represents the CSP problem that we want to solve. If a CSP is not solved, or failure has not been detected, it is split into the smaller CSPs which represent the successor nodes of the current CSP in the search tree. This process continues until either each CSP is split, failed or solved. The failed or solved CSPs represent the leaf nodes of the tree. The size of this search tree is exponential in the number of the related CSPs variables.

The most straight-forward complete search algorithm can be considered as exhaustively searching all possible assignments to find the best solution. However, if the search space is large or there exist exponentially many number of solutions, enumerating all solutions can be very time consuming if not impossible within the reasonable time limit. Such an exhaustive, systematical search has an exponential-time complexity in the best case, and this is where propagation comes in [vHK06]. Additionally, Constraint Programming is used commonly to perform complete search and it can be made efficient by making use of propagation procedures or backtracking algorithms.

In order to reduce the size of the search space some propagation, filtering and search strategies are being used. Given a constraint C and the notion of consistency, all the values that are not consistent with C are removed from the domains of the variables of C. This procedure is called *domain filtering algorithm*. Then, the effect of the removal of inconsistent values are *propagated* to the other relevant constraints. When all variables contain only the consistent

values in their domains, the CSP is called to be *locally consistent*. Efficient filtering and propagation algorithms speed up the solution phase by reducing the search space.

Backtracking is an intelligent method mainly adapted to Constraint Programming to perform complete search [RvBW06]. Backtracking algorithms behave as performing a depth-first search on the search tree. The intuition is to build up a solution by assigning values to the variables in the problem, branching, and "backtracking" when it is clear that it is impossible to find a solution using the assigned values. The search space is usually modelled as a tree structure, where a tree represents an undirected graph that has only one path between each of its vertices. The backtracking step is used to prune the sub-trees that contain no solutions.

Example. We demonstrate the backtracking algorithm on a well known optimization problem, called *graph colouring*. The graph colouring problem is defined by an undirected graph, and a set of colours each vertex can be associated with such that none of the two adjacent vertices have the same colour. Figure 2.4 represents a sample graph $G = (V, E)$ with 5 vertices. A solution to a 3-colouring of G with the colour set $C = \{blue, red, green\}$ corresponds to each vertex having a colour from C respecting the difference constraint.

Figure 2.5 shows a sample run of the backtracking algorithm and the search tree it constructs when solving G. We denote the colour *red* by R, *blue* by B and *green* by G in the tree to make it more clear for the reader.

The algorithm starts by assigning v_1 the colour red. Then, assume that it assigns v_2 as red. When it assigns v_3 as red, a constraint violation is detected because v_2 and v_3 are adjacent and they cannot have the same colour. The algorithm backtracks and assigns v_3 a different colour, which is green in this case. The solution is found if a successful assignment is found for each vertex. The solution found in Figure 2.5 is when v_1, v_2, v_4 are assigned *red*, v_3 is *green* and v_5 is *blue*.

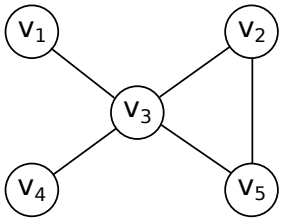


Figure 2.4: A sample undirected graph with 5 vertices.

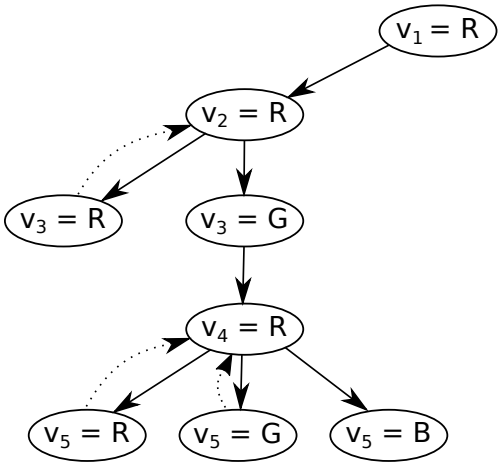


Figure 2.5: The search tree created when backtracking is used to find a solution to the graph colouring instance provided in Figure 2.4.

The simple backtracking algorithm can be significantly improved by making use of a domain filtering algorithm [GI06]. The domain filtering algorithm filters out the inconsistent values in the domains of the unassigned variables. For example, consider the same problem given in Figure 2.4 again. Figure 2.6 illustrates all the steps of using a domain filtering algorithm.

In Figure 2.6, the domain of each variable contains all possible values in the beginning. Assume that, v_1 is assigned R. By the filtering algorithm, R is filtered from the domains of all the adjacent vertices (in this case it is only v_3). Then, on the next step, assume $v_2 = R$. This assignment also filters R from v_3 (already filtered in the previous step) and v_5's domains. Considering the steps defined for Figure 2.5, now the vertex v_3 cannot be assigned to R, because R is already removed from D_3. Therefore, the backtracking step is prevented by the filtering algorithm. As next step, let $v_3 = G$. This assignment eliminates R from D_4 and D_5. This filtering prevents from the other two backtracking steps shown in Figure 2.5 for v_5. After setting $v_4 = R$, a solution to the problem instance is found as all the domains D_i have $|D_i| = 1$.

There also exist some other strategies for making the search algorithms more intelligent. For instance, *nogood learning* in CP learns from the previous search steps and does not to repeat the decisions that results in failure [FD94]. This technique is often referred as conflict learning [LW05]. These strategies are adapted in the solvers.

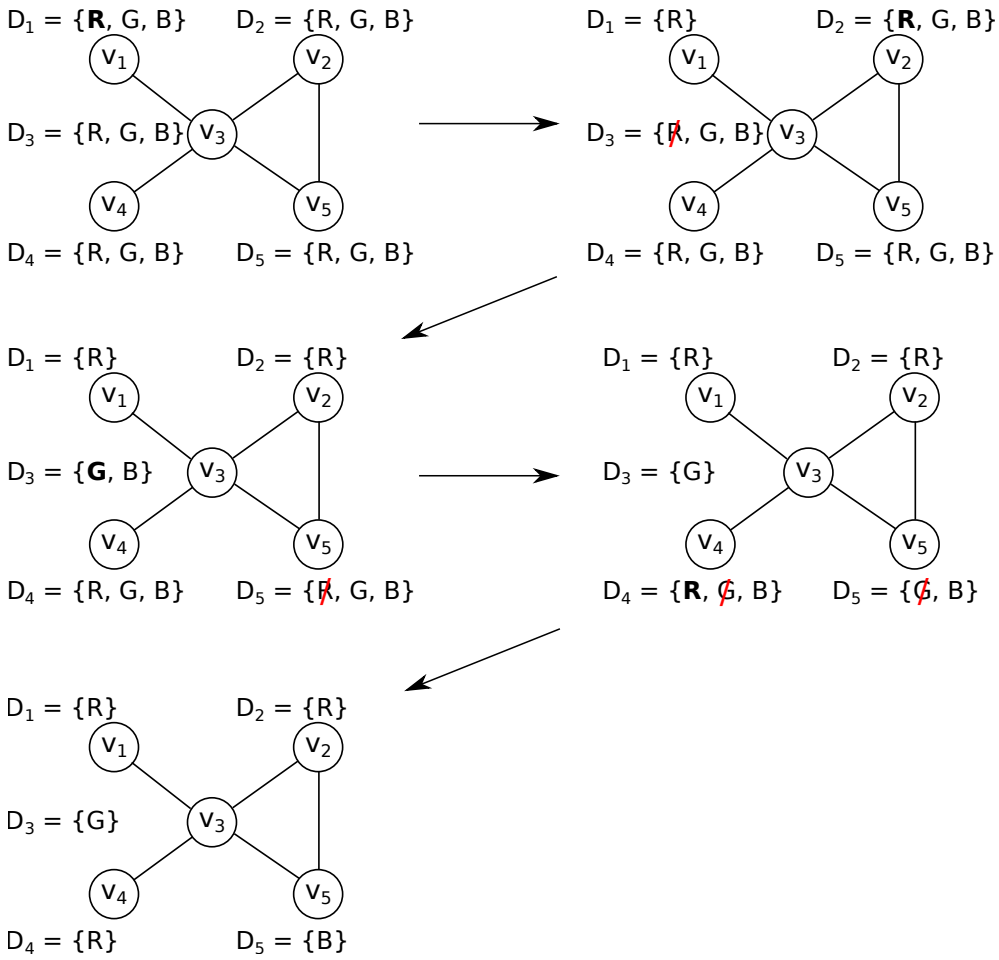


Figure 2.6: The search tree created when a domain filtering algorithm is used to enhance the search for Figure 2.5.

2.3.2.2 Choco Constraint Solver

Choco is a free open-source software that serves as a solver for constraint programming [PFL16]. It uses some filtering algorithms associated with the constraints and also uses some limits on the search. For instance, one can specify a time limit, solution limit, or a limit on the count of backtracks. It allows the use of different search strategies on the tree such as selecting the variable of smallest domain size first, or selecting the smallest domain value first. One can also specify the variables to branch on. Another feature of it is the support of restarts, which means restarting the search starting from a completely new search tree. After a search is terminated, it also provides user with some statistics such as whether the search is completed or not, the number of solutions found, the number of backtracks, the maximum depth of the search tree, and the number of constraints and the variables in the model, etc. In Listing 2.1, we present an example implementation of the graph colouring problem defined in

Figure 2.4 using Choco.

```
import org.chocosolver.solver.Model;
import org.chocosolver.solver.variables.IntVar;

public class GraphColouring {
   public static void main(String[] args) {
      // Initialization of the model
      IntVar[] vars = new IntVar[5];
      Model model = new Model("Graph colouring example");

      // Initialization of the variables
      for (int i = 0; i < numberOfVars; i++)
         vars[i] = model.intVar("var" + i, 0, 2);

      // Constraints
      model.arithm(vars[0], "!=", vars[2]).post();
      model.arithm(vars[2], "!=", vars[3]).post();
      model.arithm(vars[1], "!=", vars[2]).post();
      model.arithm(vars[2], "!=", vars[4]).post();
      model.arithm(vars[1], "!=", vars[4]).post();

      // Solve the problem
      model.getSolver().solve();

      // Print the solution
      for(IntVar v : vars)
         System.out.print(v + ", ");
   }
}
Output: var0 = 0, var1 = 2, var2 = 1, var3 = 0, var4 = 0,
```

Listing 2.1: A Choco implementation of graph colouring problem on a graph given in Figure 2.4.

In Listing 2.1, assume that we represent each color by a number. For instance, $red = 0, green = 1, blue = 2$. Therefore, each of our vertices can be represented by an integer variable (represented by $IntVar$) in Choco with the domains $[0, 2]$. Each vertex v_i in Figure 2.4 is labelled in the Choco model as $var(i - 1)$. For instance, v_1 in the Figure 2.4 corresponds to $var0$ in our Choco model.

Then, we use the edges as the constraints for our model. We create a *not equal* constraint between each pair of vertices that are connected by an edge. For instance, there exists an edge between v_1 and v_3. Therefore, we post a 'not equal' constraint to the model between the corresponding integer variables in Choco (i.e. $vars[0]$ and $vars[2]$). After posting the all 5 edges as constraints to the model, a sample solution to the problem found by Choco is reported in the output as $var0 = 0, var1 = 2, var2 = 1, var3 = 0, var4 = 0$. This solution corresponds to $v_1 = R, v_2 = B, v_3 = G, v_4 = R, v_5 = R$, which can easily be verified to be a correct solution.

2.3.3 Iterated Local Search

The main idea behind *Iterated Local Search (ILS)* is to focus only on a neighbourhood instead of focusing on the full search space [Stü98, LMS03]. To this end, a subset of solutions that are "close" to the current solution is referred as the *neighbourhood*. The search starts from an initial solution and as the name suggests, the solution is attempted to be improved at each iteration until an acceptable solution is found. In order to improve the current solution S, the neighbourhood N_S of S is considered, and the search continues with a solution in N_S that provides a better objective value than S. The perturbation to the "nearby" solution is desired to be not too small and also not too large. Large perturbations lead to almost random states and the algorithm acts stochastically, whereas small perturbations make the algorithm behave in a greedy manner and may not provide an escape from the local optima. In order to make the search more efficient, a history can be kept in order not to search the previously searched local optimas. The outline for ILS is given in Algorithm 1.

The search process is initiated by a random solution s_0. Then, the neighbourhood of the current solution is constructed and the best neighbour s^* is found.

Algorithm 1 Iterated Local Search [Stü98]

```
1: procedure ITERATEDLOCALSEARCH()
2:     s_0 ← generate an initial solution
3:     s* ← LOCALSEARCH(s_0)
4:     while a termination criterion is not met do
5:         s' ← PERTURBATION(s*, history)
6:         s'' ← LOCALSEARCH(s')
7:         s* ← ACCEPTANCECRITERION(s*, s'', history)
```

Then, the best neighbour is modified (s'), and the neighbourhood search of s' is performed. If a better neighbour is found, the current solution is updated by the s'' to be used as the base of the next neighbourhood search.

The iteration provides a convergence to the local optima but does not guarantee to reach the global optimum. Therefore, the ILS procedures can easily get stuck in a local optimum without providing any improvement. In order to prevent this situation, a common strategy is to use *random restarts* [LMS03]. If random restarts are being used, search is restarted a number of times by a randomly created solution and the search proceeds from this point.

Example. Consider the 3-colouring problem instance given in Figure 2.4. Let a neighbour s_n of a solution s to be defined by changing only the colour of a single vertex from s. Additionally, assume a solution is evaluated as a good solution based on the number of vertices in it that satisfy the difference constraint. Assume that a solution s is denoted by a list of colours in order $s = \langle c_1, c_2, c_3, c_4, c_5 \rangle$, where each c_i represents the color of the corresponding vertex v_i. Therefore $c_i \in \{R, G, B\}$.

A sample initially created solution s_0 to this problem is $s_0 = \langle G, G, G, G, G \rangle$. The neighbourhood of s_0 is then corresponding to a set of 10 different solutions as: $\langle R, G, G, G, G \rangle, \langle B, G, G, G, G \rangle, \ldots \langle G, G, G, G, B \rangle$. The best neighbour in this set is one of the solutions that has as many vertices as possible that satisfy the difference constraint. Let this neighbour be $s'' = \langle G, G, R, G, G \rangle$ as it satisfies the constraint by 3 of its vertices: v_1, v_3, v_4. Comparing the current best solution s_0 and s'', one can infer that s'' is a better solution as s_0 does not have any vertices that satisfy the constraint. Therefore, the search proceeds by accepting $s^* = s''$. On the next iteration, a neighbour solution of s^* that is better (such as $s'' = \langle G, B, R, G, G \rangle$) is selected as the base of the neighbour search. The search terminates when a termination criterion is met.

The termination criteria can be a time limit, finding the optimal solution, or finding no improved solutions for a number of iterations.

2.3.4 Genetic Algorithm

Genetic algorithms (GA) are population-based methods inspired by the natural process of evolution [Hol92]. They are compelling robust search and optimiza-

tion tools, which work on the natural concept of evolution, based on natural genetics, and natural selection. They have been used extensively in optimization problems as an alternative to traditional heuristics. Holland introduced the GA and he also showed how to apply the process to various computationally difficult problems [Hol92, B96].

The generic search process consists of five main steps: initialization of the population, evaluation, selection, crossover and mutation [BNKF98]. The initialization starts with finding an initial set of solutions called *population*. Each solution in the population is referred as an *individual*. Each individual is associated with a *fitness value* indicating how good (or fit) it is. The fitness is evaluated with respect to the objective function. The selection process is carried out using the fitness values of individuals. There exist different selection algorithms. One of them is called *roulette wheel selection* [LL12]. Roulette wheel selection gives a more fit individual higher chance to be selected.

There are two main operators that are used to improve the solution quality, provide convergence, and help with tackling the problem of getting stuck at local optima in the population: *crossover* and *mutation*. The population is evolved by applying crossover on different selected individuals or applying mutations on a single selected individual. These operations simulate the real world inheritance of genes from parents to children as well as the mutations. The process terminates either when an acceptable solution is found or a termination criterion is met: such as a pre-determined time limit or not improving the solution for a number of iterations. The process is illustrated in Figure 2.7.

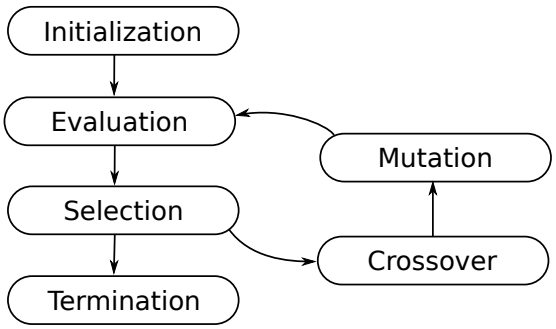


Figure 2.7: The procedure for a generic Genetic algorithm model.

Example. Consider the 3-colouring problem given in Figure 2.4. Let the fitness of an individual be measured by the number of vertices in the graph that satisfy the different colour constraint with respect to the given solution. Assume

that, we have a GA model for this problem, where the population consists of only 3 individuals. Let each individual s denoted by a list of colours in order $s = \langle c_1, c_2, c_3, c_4, c_5 \rangle$, where each c_i represents the color of the corresponding vertex v_i. Therefore $c_i \in \{R, G, B\}$. A population P can be initialized randomly as $P = \langle s_1, s_2, s_3 \rangle$, where $s_1 = \langle R, G, B, G, G \rangle$, $s_2 = \langle G, G, B, B, B \rangle$, and $s_3 = \langle B, B, B, B, B \rangle$. Let us list the individuals in P and their fitness criterion below where c denotes the number of vertices that satisfy the constraint:

$$s_1 = \langle R, G, B, G, G \rangle,\ c = 3,$$
$$s_2 = \langle G, G, B, B, B \rangle,\ c = 2,$$
$$s_3 = \langle B, B, B, B, B \rangle,\ c = 0.$$

When each individual in this population is evaluated for their fitnesses, s_1, s_2 are the fitter than s_3. In the selection phase, it is more likely for the fitter individuals to be selected. Therefore, assume that s_1 and s_2 are selected by roulette wheel selection. Let s_{c1} and s_{c2} be the two products of crossover of s_1 and s_2. A sample crossover operation over the individuals can be defined as swapping the halves of the lists. The operation can be reflected on the individuals as follows:

$$s_1 = \langle R, G, B, G, G \rangle,\ c = 3,$$
$$s_2 = \langle G, G, B, B, B \rangle,\ c = 2,$$
$$s_{c1} = \langle R, G, \boldsymbol{B}, \boldsymbol{B}, \boldsymbol{B} \rangle,\ c = 2,$$
$$s_{c2} = \langle G, G, \boldsymbol{B}, \boldsymbol{G}, \boldsymbol{G} \rangle,\ c = 3.$$

The products s_{c1} and s_{c2} are not the optimal solution. In fact, s_{c1} is a less fit solution when compared to the least fit solution in the population (i.e. s_3). Let the population be refined by eliminating the least fit individuals and adding the fitter ones. This process removes s_3, and adds s_{c2} as $P = \langle s_1, s_2, s_{c2} \rangle$. In detail:

$$s_1 = \langle R, G, B, G, G \rangle,\ c = 3,$$
$$s_2 = \langle G, G, B, B, B \rangle,\ c = 2,$$
$$s_{c2} = \langle G, G, B, G, G \rangle,\ c = 3.$$

On the next step, let s_1 be selected for mutation. A sample mutation operation can be performed as randomly changing the colour of one vertex from the selected individual. Therefore, let s_1 be mutated on the vertex v_5 as $s_1 = \langle R, G, B, G, R \rangle$. The new population P is now updated as:

$$s_1 = \langle R, G, B, G, R \rangle,\ c = 5,$$
$$s_2 = \langle G, G, B, B, B \rangle,\ c = 3,$$
$$s_{c2} = \langle B, G, B, G, G \rangle,\ c = 3.$$

The individual s_1 is now an optimal solution that satisfies the different colours constraint on all of its vertices. The evolution continues until a termination criterion such as finding the optimal solution, having no improvement for a number of iterations, or a time limit is met. If the search cannot find the optimal solution, then the best solution found during the search is returned as the final solution.

2.3.5 Genetic Local Search

Combining different search techniques to enhance the performance of the models is a popular field. The methods for creating hybrids are not only limited to combining methods from different metaheuristics but also combining the metaheuristics with the other optimization methods such as mathematical programming, constraint programming, machine learning etc. [RPB10, Tal15]. We consider combining two metaheuristics: the GA and the LS, which is referred as *genetic local search* in the literature [FM96]. Genetic local search algorithms have been shown to significantly improve the quality of the search and reduce the time for many combinatorial optimization problems such as the Travelling Salesman, graph coloring problems or in the field of bioinformatics [KP94, UAB+91, DH98, MF97, YMLC16]. We use the abbreviation *HB* to refer to the hybrid model.

Here, we use local search in order to increase the diversity in the population after the crossover is applied. The procedure of LS is described in detail in Section 2.3.3, and the GA in Section 2.3.4. We apply the same procedures defined in these previous sections and combine them as described below. After crossover step is applied, two product individuals are generated. We search the neighbourhood each product individual to see if they have a better individual than themselves. If the individual produced has a better neighbour, then it is replaced with its neighbour and the population is updated accordingly. The whole process of HB is illustrated in Figure 2.8. Later, we use this hybrid procedure to solve our proposed optimization problem.

Example. Consider the 3-colouring problem given in Figure 2.4. Let the fitness of an individual be measured by the number of vertices in the graph that satisfy the different colour constraint with respect to the given solution. Assume that, we have a GA model, where the population consists of only 3 individuals.

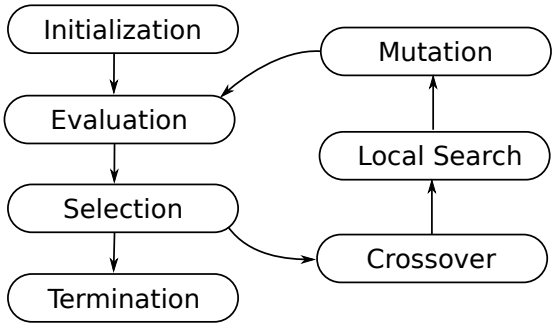


Figure 2.8: The procedure of a generic Genetic Local Search algorithm.

Let each individual s be defined by a list of colours in order $s = \langle c_1, c_2, c_3, c_4, c_5 \rangle$, where each c_i represents the color of the corresponding vertex v_i. Therefore $c_i \in \{R, G, B\}$. Consider the sample population $P = \langle s_1, s_2, s_3 \rangle$, where $s_1 = \langle R, G, B, G, G \rangle, s_2 = \langle G, G, R, B, B \rangle, s_3 = \langle B, B, B, B, B \rangle$ as in the example given in Section 2.3.4. The variable c denotes the number of vertices that satisfy the different colour constraint using the given solution. Below is the list of individuals in the initial population:

$$s_1 = \langle R, G, B, G, G \rangle, c = 3,$$
$$s_2 = \langle G, G, B, B, B \rangle, c = 2,$$
$$s_3 = \langle B, B, B, B, B \rangle, c = 0.$$

Similar to the steps given in the GA example, let s_1 and s_2 crossover and s_{c1}, s_{c2} below denote the products of the crossover:

$$s_{c1} = \langle R, G, \boldsymbol{B}, \boldsymbol{B}, \boldsymbol{B} \rangle, c = 2,$$
$$s_{c2} = \langle G, G, \boldsymbol{B}, \boldsymbol{G}, \boldsymbol{G} \rangle, c = 3.$$

Now the neighbourhood of both product individuals are constructed. We can define the notion of neighbour in the same way that we defined in the example of Section 2.3.3, where a neighbour solution differs from the current one by the colour of exactly one vertex. Let N_{c1} (and N_{c2}) denote the neighbourhood of s_{c1} (respectively s_{c2}). Note that, none of the neighbours of s_c provides an optimal solution. However, the fittest of them (i.e. the best neighbour) is an assignment that has the maximum number of vertices satisfying the difference constraint. An assignment s_{n1} can be found for s_{c1} as containing 4 vertices that satisfy the constraint. Similarly, the best neighbour of s_{c2} can be found as follows:

$$s_{n1} = \langle R, G, \boldsymbol{R}, B, B \rangle, c = 4,$$
$$s_{n2} = \langle G, \boldsymbol{R}, B, G, G \rangle, c = 5.$$

Finding that s_{n1} and s_{n2} are fitter solutions than s_{c1} and s_{c2}, the best neigh-

bours s_{n1} and s_{n2} are considered for refining the population. After eliminating the worst two individuals from the population and adding the better ones, the population P is updated as:

$$s_1 = \langle R, G, B, G, G \rangle, c = 3,$$
$$s_{n1} = \langle R, G, R, B, B \rangle, c = 4,$$
$$s_{n2} = \langle G, R, B, G, G \rangle, c = 5.$$

Considering this population, the solution s_{n2} represents an optimal solution and therefore the search is terminated.

2.3.6 Computational Complexity

The *theory of computation* is a concept in mathematics and computer science based on estimating how efficiently a problem can be solved. The method used to solve a problem is usually referred as an algorithm. In order to solve combinatorial problems, different researchers proposed different approaches and formulations of algorithmic notions. In 1936, Church introduced the idea of "effective calculability" [Chu36]. At the same time, Turing defined the "computability" [Tur36]. Turing stated in his paper: "In a recent paper Alonzo Church has introduced an idea of "effective calculability", which is equivalent to my "computability", but is very differently defined." Then, he showed the equivalence between these two notions. Turing is the one who defined in the best way how a universal machine is able to compute any kind of calculus. Therefore, in modern days, the efficiency of an algorithm is calculated based on Turing's computability studies [Tur36].

Turing explained that any algorithm can be described by a Turing Machine. He defined the *Turing machine* as a machine TM that has a one-dimensional blank tape in motion and a head that is used to print or read symbols on the tape. The finite set of all possible symbols defines an *alphabet* on TM. The tape consists of square-regions and the head is always positioned on one of the regions in TM. The head can only move to the right or to the left of its currently located square. The machine has two actions: write or move. A set of symbols defined over the alphabet defines an *input*. The machine reads the input by one symbol at a time, and either writes a symbol from the alphabet on the tape or the head shifts to the square on the left or right. A TM is said to be *deterministic*, if at most one action can be performed in any state. If the transition function is

changing the state of the machine to a possible set of states, then the machine is referred as a *non-deterministic* TM.

The Turing Machine can be abstracted as having a finite number of states and the machine runs by transitioning between the states with respect to the given input. If the machine terminates after a finite number of transition between the states, the set of symbols written on the tape after termination is called the *output*. A problem is said to be *computable* if there exists a deterministic TM that gives an output for every possible input of that problem. Additionally, a problem is decidable if there exists a deterministic TM that produces a "Yes/No" answer.

In Sections 2.3.6.1 and 2.3.6.2, we present two different complexity measures: complexity of an algorithm and complexity of a problem.

2.3.6.1 Algorithmic Complexity

The main goal in computation is to find an efficient algorithm that solves a problem. The algorithms define formal step-by-step strategies for solving problems [CLRS09]. Each algorithm can be represented by a Turing machine TM. Different algorithms can be compared by their complexities. The complexity of an algorithm is mainly studied in terms of the *time* and *space* required to solve a problem. The *size* of a problem is measured by the amount of input data that is required to describe an instance of the problem (i.e. length of the input). The *time complexity* of an algorithm is described by the number of steps required from the initial state of a TM to termination (i.e. the number moves performed by the head). The *space complexity*, on the other hand, indicates how much memory the system needs to use from start to termination (i.e. the space used on the tape of a TM). The complexities of algorithms proposed for solving the problems can be classified with respect to their order of growth. The *growth function* of an algorithm gives an indication of how the algorithm behaves in terms of running time. We can then compare algorithms by the asymptotic efficiency of their growth functions when the input size is assumed to have no bounds.

The time complexity of an algorithm is often expressed by using the *Big-O notation*, denoted by *O(.)* [Sip06]. Big-O notation expresses an (asymptotic) upper bound on the growth function in terms of the running time or space requirements of a given algorithm. A function $f(n)$ is said to be $O(g(n))$ if and only if

there exists two positive constants $c \in \mathbb{R}^+$ and $n_0 \in \mathbb{Z}^+$ such that [MS91]:

$$\forall n \geq n_0, f(n) \leq c \cdot g(n)$$

In this case, $g(n)$ is said to be an upper bound on $f(n)$. Let TM be a Turing machine and let n denote the size of a problem instance. We say that TM has time complexity $O(f(n))$. Since the Big-O notation represents an upper bound, it is used to indicate a bound for the worst-case running time of the algorithm.

It is possible to characterize algorithms with the input size denoted by n and a positive constant x by using the Big-O notation as [Hoe14]:

- Constant time: $O(1)$
- Logarithmic time: $O(logn)$
- Linear time: $O(n)$
- Polynomial time: $O(n^x)$
- Exponential time: $O(x^n)$

After classifying an algorithm using the Big-O notation, one can infer which algorithm is better in terms of the running time. The hierarchy between these classes can broadly be given as: $O(1) << O(logn) << O(n) << O(n^x) << O(x^n)$.

2.3.6.2 Problem Complexity

Computational complexity is a branch of the theory of computation that focuses on the difficulty of solving a problem [GJ79]. Complexity theory is used to measure the hardness of any kind of problems. It provides us with a methodology and simple tools to prove that some problems are intractable. Within this framework, we can show that a problem is as difficult as any other well-known intractable problems. In this section, we discuss briefly the definitions of some of the complexity classes because we use them later to prove the complexity of our problem. These classes are: $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete, $\mathcal{NP}$-hard and $\#P$-complete. The relationship between these different complexity classes under the assumption $\mathcal{P} \neq \mathcal{NP}$ is illustrated in Figure 2.9.

We start with the definitions of two fundamental computational complexity classes: $\mathcal{P}$ and $\mathcal{NP}$. The class $\mathcal{P}$ is the set of all problems that can be solved by

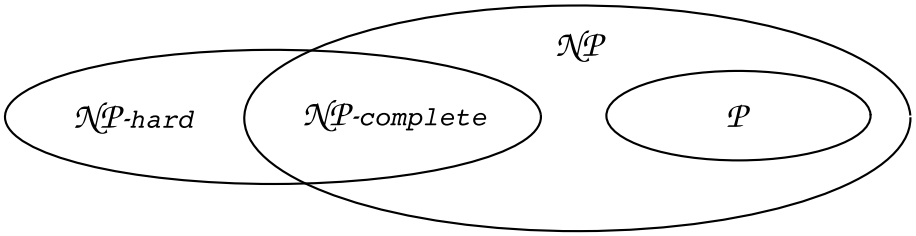


Figure 2.9: Illustration of the complexity classes under the assumption that $\mathcal{P} \neq \mathcal{NP}$.

a deterministic TM that is bounded by time that is a polynomial of the size of the problem. Any such problem in $\mathcal{P}$ is also referred as solvable in "polynomial-time" or "tractable". Thus, there exists an algorithm A for each of these problems that solves any instance of the given problem in $O(n^x)$ where n represents the size of a problem instance, and x is a constant.

Class $\mathcal{NP}$ represents the problems that can be solved by a non-deterministic TM in polynomial-time. In other words, a solution can only be found in polynomial-time if a non-deterministic Turing machine is used. We know how to simulate a non-deterministic machine with a deterministic one. However, it takes exponential time to consider all the possible assignments (2^n). Therefore, $\mathcal{P} \subset \mathcal{NP}$.

The problems in $\mathcal{P}$ are contained by the set $\mathcal{NP}$, however whether $\mathcal{P} = \mathcal{NP}$ is considered to be one of the most important open problems. If one can prove that $\mathcal{P} = \mathcal{NP}$, that means that any problem whose solutions can be verified in polynomial-time is also solvable in polynomial-time.

Another complexity class, namely $\mathcal{NP}$-hard, represents a set of problems that are at least as hard as all the problems in $\mathcal{NP}$ (if not harder). Any problem in $\mathcal{NP}$ can be reduced in polynomial-time to a problem in $\mathcal{NP}$-hard.

Let us briefly explain the difference between an optimization and a decision problem before introducing the rest of the complexity classes of interest. A problem can be modelled as an optimization problem, or a decision problem. An optimization problem is interested in finding a solution to a given problem, whereas the decision problem is interested in generating a "Yes/No" answer for expressing whether there exists a solution to the problem using a set of input values or not [GJ79]. For instance, recall the Travelling Salesman Problem introduced in Definition 1. Let us define the inputs of the problem. There exists a set of cities $C = \{c_1, c_2, \ldots, c_m\}$. A *tour* is defined as an ordering of cities $\langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(m)}$ of C such that:

$$\Big(\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \Big) + d(c_{\pi(m)}, c_{\pi(1)})$$

Definition 2 gives a formal definition for the optimization problem of TSP, whereas Definition 3 represents a formal definition for its decision version.

Definition 2 *Optimization problem for TSP.*
INPUT*: A finite set $C = \{c_1, c_2, \ldots, c_m\}$ of "cities", a "distance" $d(c_i, c_j) \in Z^+$ for each pair of cities $c_i, c_j \in C$, and a bound $B \in Z^+$ (where Z^+ denotes the positive integers)*.
QUESTION*: Find a tour of all the cities in C that minimizes the total length in B.*

Definition 3 *Decision problem for TSP.*
INPUT*: A finite set $C = \{c_1, c_2, \ldots, c_m\}$ of "cities", a "distance" $d(c_i, c_j) \in Z^+$ for each pair of cities $c_i, c_j \in C$, and a bound $B \in Z^+$ (where Z^+ denotes the positive integers)*.
QUESTION*: Is there a tour of all the cities in C having total length no more than B?*

Note that, we use this formal language to define our problems in the remaining sections.

There is another complexity class that is a subset of $\mathcal{NP}$ is called as $\mathcal{NP}$-complete. The problems in $\mathcal{NP}$-complete are both $\mathcal{NP}$ and $\mathcal{NP}$-hard. They represent decision problems whose solutions can be verified in polynomial-time. The first problem to be proven as being in $\mathcal{NP}$-complete is the *Boolean Satisfiability Problem (SAT)* by Cook's theorem [Coo71]. Cook received a Turing award for his work on complexity.

In order to give an insight, the optimization problem for TSP Definition 2 lies in $\mathcal{NP}$, because given a tour P as solution, P can be tested for being a solution or not in polynomial-time. On the other hand, the decision problem of TSP given in Definition 3 is proven to be $\mathcal{NP}$-complete by Karp in 1972 [Kar72]. Following his contributions to the field, Karp also received a Turing award in 1985.

The last class we present is $\#P$-complete. This class contains a set of problems related to counting, whose related decision problem is in $\mathcal{NP}$-complete [Val79]. Similarly, a problem is $\#P$-complete if and only if it is in $\#P$, and all $\#P$-hard problems are reducible to it. The counting problem of the TSP can be formulated as in Definition 4.

Definition 4 *Counting problem for TSP.*
INPUT*: A finite set* $C = \{c_1, c_2, \ldots, c_m\}$ *of "cities", a "distance"* $d(c_i, c_j) \in Z^+$ *for each pair of cities* $c_i, c_j \in C$*, and a bound* $B \in Z^+$ *(where* Z^+ *denotes the positive integers).*
QUESTION*: How many tours of all the cities in* C *have total length no more than* B*?*

2.4 Matching Under Preferences

Matching problems emerge from many large-scale real-world problems and have become the basis of a variety of important applications. These problems, in general, involve assigning agents from one set to another. Typically, each agent has an ordered preference list over the agents of the other set. Similarly, some of the agents may have a capacity constraint to be satisfied, which indicates the maximum number of other agents that should be assigned to it. These types of problems have been widely studied by different research communities such as computer scientists and economists over the years; in fact, the 2012 Nobel Prize for Economics was awarded to Shapley and Roth for their work on stable allocations [201].

The most famous matching problem is the *Stable Marriage (SM)* which first appeared in the literature in 1962 [GS62]. The Stable Marriage problem consists of a set of men and a set of women, where each person has a strictly ordered preference list over the people in the other set. The optimality criterion is to assign each person with a partner such that no two unassigned people prefer each other more than their currently assigned partners. The most straightforward generalizations of the Stable Marriage problem have been studied such as the non-bipartite version *Stable Roommates problem* or the many-one generalization *Hospitals/Residents problem* [GI89]. In addition to these problems, some further stable matching problems such as versions where the preference lists have ties [GI89], assigning pupils to schools [APRS05], student-project allocation [AIM07], kidney exchange programmes [RSÜ04], and house allocation [AS98] problem have been studied under many different optimality criteria such as Pareto optimality, popularity, rank-maximality, etc [Man13]. For a comprehensive survey on different matching problems the reader is referred to the book written by Manlove [Man13].

It is important to note here that matching problems are also studied within the field of algorithmic game theory under different optimality criteria. Game theory is a study of mathematical models that aim to model situations in which multiple participants interact or affect each other's outcomes [NRTV07]. Extending this concept, *algorithmic game theory* is the study of game theory combined with theoretical computer science [Rou10]. One of the most famous concepts in algorithmic game theory is the Nash equilibrium. In a Nash equilibrium, no player can be made better-off by changing their own strategies [Suh97]. The Nash equilibrium focuses on changing strategy for one participant at a time. A specific case, where the participants cannot benefit from jointly changing the strategies of any possible subset of participants, is referred as *strong (Nash) equilibrium* [Suh97]. The game version of the SM is a good example that implements the strong Nash equilibria [NRTV07, DPK13].

In Section 2.4.1, we focus on the details of the Stable Marriage problem and in Section 2.4.2, we discuss details of the Stable Roommates problem. We present a comprehensive explanation for both of the problems, as we need their properties in detail, later.

2.4.1 Stable Marriage Problem

The *Stable Marriage problem (SM)* was formally defined and solved by David Gale and Lloyd Shapley [GS62]. In 1989, Gusfield and Irving wrote a book that focuses on the Stable Marriage problem and provided detailed analysis about the structure of the problem as well as the algorithms being used [GI89]. Unless otherwise stated, we refer to the notation and the information presented in their book.

The SM is a bipartite matching problem with two-sided preferences, where the aim is to find a *stable* matching. Formally, an instance of the Stable Marriage problem takes as input two disjoint sets: a set of men $U = \{m_i, m_{i+1}, \ldots, m_{n_i}\}$ and a set of women $W = \{w_i, w_{i+1}, \ldots, w_{n_i}\}$, where each person has an ordinal preference list over all members of the opposite sex. In the SM, the preference lists are said to be complete if all members of each sex strictly ranks all members of the opposite sex. The *size* of an SM instance is denoted by n and assumed to be $n = |U| = |W|$. Practically, a person can find another person from the other sex *unacceptable*, hence prefers not to give a ranking in his/her preference list. In this case, some preference lists may not be complete. This version of

Table 2.1: Preference lists for men (left) and women (right) for a sample Stable Marriage instance of size 7.

m_0	0 6 5 2 4 1 3
m_1	6 1 4 5 0 2 3
m_2	6 0 3 1 5 4 2
m_3	3 2 0 1 4 6 5
m_4	1 2 0 3 4 5 6
m_5	6 1 0 3 5 4 2
m_6	2 5 0 6 4 3 1

w_0	2 1 6 4 5 3 0
w_1	0 4 3 5 2 6 1
w_2	2 5 0 4 3 1 6
w_3	6 1 2 3 4 0 5
w_4	4 6 0 5 3 1 2
w_5	3 1 2 6 5 4 0
w_6	4 6 2 1 3 0 5

the problem is defined in the literature, without loss of generality, as *Stable Marriage with Incomplete Lists problem (SMI)*.

A man-woman *pair* (m_i, w_j) is *acceptable* if w_j (respectively m_i) appears in the preference list of m_i (respectively w_j). For the SM, a *matching* M is defined as a one-to-one correspondence between the sets U and W such that M consists of a set of acceptable pairs where each man and woman appear in exactly one pair of M. A matching is represented by $M = \{(m_{i0}, w_{j0}), (m_{i1}, w_{j1}), \ldots, (m_{in}, w_{jn})\}$. If $(m_i, w_j) \in M$, we say that w_j (respectively m_i) is the partner of m_i (respectively w_j) and we denote $p_M(m_i) = w_j$ and $p_M(w_j) = m_i$. A pair (m_i, w_j) (sometimes denoted as (i, j)) is said to be *blocking* a matching M if m_i is unassigned or prefers w_j to $M(m_i)$ and w_j is unassigned or prefers m_i to $M(w_j)$. A matching M is called *stable* if there exists no blocking pair in M. Similarly, a pair (m_i, w_j) is said to be *stable* if it appears in some stable matching. Additionally, a pair (m_i, w_j) is called *fixed* if (m_i, w_j) appears in all the stable matchings of the given instance. We also refer to the man m_i and woman w_j that appear in a fixed-pair as fixed. Table 2.1 gives an example of a Stable Marriage instance with 7 men and 7 women, where each line represents the preference list of the relevant person. For clarity, we denote each man m_i with i and each woman w_j with j in the preference lists.

The algorithm proposed by Gale and Shapley, called the GS algorithm, was proven to always find a stable matching to a given SM instance. The GS algorithm finds a stable matching that is either uniquely favouring the men or the women. The case where the men are proposing to the women results in a stable matching that the men are matched with their best possible choice. If the women are proposing, then the GS algorithm results in the stable matching that is the best possible one for the women. Throughout this thesis, when we talk about the operations we use man-oriented language (for instance we say "if a man wants to break-up" or "a man prefers" etc.) to keep consistent and

clear. However, a woman-oriented language can be used interchangeably. This change does not affect the theoretical results.

The idea behind the GS algorithm is straight-forward. The algorithm allows men (or women) to “propose" to women (or men) until every man (or woman) has made a proposal that has been accepted. There are two states during the execution of the algorithm: a person can be free or can be engaged. In the case that the men are proposing, the next free man proposes to the women starting from the most preferred one in his list until he becomes engaged. On the other hand, when a woman receives a proposal, if she is free at the time, she accepts the proposal and becomes engaged with the proposer. However, if she is already engaged with another man, she checks her preference list, refuses the less favourable man and becomes engaged to the more preferred one. If she breaks her current engagement, the man who has been rejected becomes free and the algorithm continues as him proposing to the next most favourable woman on his list. The order of the people proposing is not important as all possible executions of the GS algorithm results in the same stable matching.

Subsequently, Gusfield and Irving slightly enhanced the GS algorithm and called it the Extended Gale-Shapley algorithm. The motivation behind the extended version was to reduce the size of the preference lists. In order to achieve it, they introduced a *deletion* operation for any pair (m_i, w_j) that is not stable. The deletion of pair (m_i, w_j) is performed by deleting the woman w_j from man m_i's list and deleting m_i from w_j's list. The extended algorithm is presented as Algorithm 2.

It was shown by Gale and Shapley that the structure that represents all the stable matchings for an instance forms a *lattice* $\mathscr{M}$. In this lattice, the stable matching in which each man is assigned to their most preferred stable part-

Algorithm 2 Extended Gale-Shapley [GI89]

```
1: procedure EXTENDED-GS()
2:     assign each person to be free;
3:     while some man m is free do
4:         w := first woman on m's list;
5:         if some man p is assigned to w then
6:             assign p to be free;
7:         assign w to m;
8:         for each successor m' of m on w's list do
9:             delete the pair (m', w);
```

ner is called the man-optimal (woman-pessimal) matching and denoted by M_0. Similarly, the woman-optimal (man-pessimal) matching is denoted by M_Z. A stable matching M_i is said to *dominate* another stable matching M_j, denoted by $M_i \preceq M_j$, if every man prefers their partners in M_i to the ones in M_j or indifferent between them. The man-optimal matching M_0 and the woman-optimal matching M_Z represent the minimum and maximum elements of this lattice because M_0 dominates all other stable matchings and M_Z is dominated by all others. In Figure 2.10, we give the lattice of all stable matchings of the SM instance given in Table 2.1.

The size of the lattice of stable matchings can be exponential in the number of the people involved (n), as the number of all stable matchings can be exponential in n [IL86b]. Pittel proves that the average number of stable matchings when the preference lists are similar is $O(nlogn)$ [Pit89]. Although it is known that the number of stable matchings of a given SM instance can be exponential

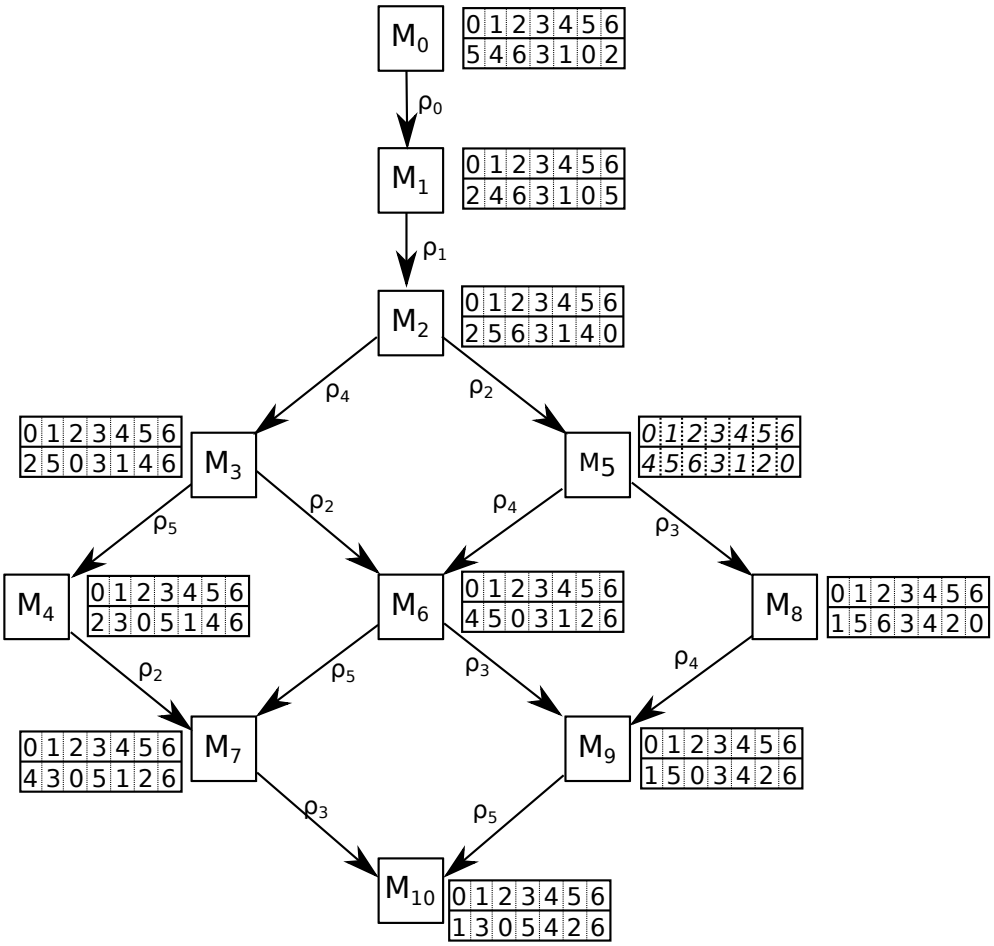


Figure 2.10: The lattice of all stable matchings corresponding to the instance given in Table 2.1.

in n, finding the maximum number of stable matchings an instance can have is an open question proposed by Donald Knuth [Knu76]. Irving and Leather constructed a family of instances that has been shown to contain at least $\Omega(2.28^n)$ stable matchings [IL86b]. On the other hand, the best known asymptotic upper bound is $O(\frac{3}{4}n!)$ [DBS13]. The number of stable matchings is also studied on the three-dimensional version of the SM, and proven by Escamoher and O'Sullivan that it is exponential in the instance size [EO18]. For more details on the number of stable matchings, the reader is referred to the two recent studies [Man13, KGW17].

It is important to note here a remarkable result on the stable pairs from Gusfield's work in Theorem 2.

Theorem 2 (Theorem 1.4.2 [GI89]) *In a Stable Marriage instance that allows unacceptable partners, the men and the women are each partitioned into two sets − those that have partners in all stable matchings and those that have partners in none.*

From this theorem, it is known that the same set of men/women are assigned in all stable matchings. This result is also known as the "Rural Hospitals Theorem" [Man13]. If a person has a partner in a stable matching, then he/she has a partner in all possible stable matchings of the same instance.

Let M be a stable matching of a given SM instance $\mathcal{I}$. Given any man $m_i \in U$ in $\mathcal{I}$, let $s_M(m_i)$ denote the most-preferred woman w_j on m_i's list such that $w_j \in W$ and w_j prefers m_i to $p_M(w_j)$. Also, let $next_M(m_i)$ denote $p_M(s_M(m_i))$. A *rotation* $\rho = (m_0, w_0), (m_1, w_1), \ldots, (m_{l-1}, w_{l-1})$ (where $l \in \mathbb{N}^*$) is an ordered list of pairs that is said to be *exposed* in M such that for each i, $0 \leq i \leq l-1$, $m_{i+1} = next_M(m_i)$, where the addition operation is modulo l. For each pair $(m_i, w_j) \in \rho$, we say m_i (or w_j) is *involved* in ρ or ρ *includes* m_i (or w_j).

It is important to see that a pair cannot be involved in more than one rotation. Corollary 1 shows this property.

Corollary 1 (Corollary 3.2.1 [GI89]) *Every man-woman pair (m, w) is in at most one rotation. Hence there are at most $n(n-1)/2$ rotations in an instance of the Stable Marriage problem of size n.*

The *elimination* of ρ from the matching M that it is exposed in results in another stable matching denoted by M/ρ. The matching M/ρ is defined as: $M/\rho = (M \setminus \{(m_i, w_i) : 0 \leq i \leq l-1\}) \cup \{(m_i, w_{i+1}) : 0 \leq i \leq l-1\}$. In Figure 2.10,

there exists a vertex for each stable matching of the instance and each vertex is contains two vectors for representing the partners. The first vector represents the set of men and the second vector represents the partner of each man in the matching. Each edge $e = (M', M)$ on the lattice is labelled with the rotation ρ, and should be read as: ρ is exposed on M' and $M = M'/\rho$.

There exists a partial order for rotations as shown by Gusfield and Irving [GI89]. A rotation ρ' is said to precede another rotation ρ (denoted by $\rho' \prec \rho$), if ρ' is eliminated in every sequence of eliminations that starts at M_0 and ends at a stable matching in which ρ is exposed. Note that this relation is transitive, that is, $\rho'' \prec \rho' \wedge \rho' \prec \rho \implies \rho'' \prec \rho$. The structure that represents all rotations and their partial order is a directed graph called *rotation poset* denoted by $\Pi = (\mathcal{V}, E)$. Each rotation in Π corresponds to a vertex in $\mathcal{V}$ and there exists a directed edge from ρ' to ρ if ρ' precedes ρ. The number of rotations is bounded by $n(n-1)/2$, and hence, the size of the rotation poset is $O(n^2)$ [GI89].

There are two different edge types in a rotation poset: ***Type 1*** and ***Type 2***. Suppose that a pair (m_i, w_j) is in a rotation ρ. If ρ' is the unique rotation that moves m_i to w_j then $(\rho', \rho) \in E$ and ρ' is called a Type 1 predecessor of ρ. Suppose now that a pair (m_i, w_j) is not involved in any rotations. If ρ moves m_i below w_j, and $\rho' \neq \rho$ is the unique rotation that moves w_j above m_i, then $(\rho', \rho) \in E$ and ρ' is called a Type 2 predecessor of ρ.

If a rotation ρ' precedes another, ρ, we say ρ' is the *predecessor* of ρ. Similarly, ρ is called the *successor* of ρ'. Additionally, given two rotations ρ and ρ', we say that ρ' is an *immediate predecessor* of ρ if $\rho' \prec \rho$ and there is no rotation ρ'' such that $\rho' \prec \rho'' \prec \rho$. Similarly ρ is an *immediate successor* of ρ' if ρ' is an immediate predecessor of ρ. Immediate predecessors of a rotation ρ in a rotation poset are denoted by $N^-(\rho)$ and immediate successors are denoted by $N^+(\rho)$. Later, we shall need transitivity. Therefore, we denote by $N_t^-(\rho)$ (respectively $N_t^+(\rho)$) the predecessors (respectively successors) of a rotation ρ including transitivity. Using the terms defined in Section 2.1.2, two rotations are said to be *comparable* if one precedes the other. Similarly, they are said to be *incomparable* if one does not precede the other.

Gusfield and Irving present an algorithm for the construction of Π, which runs in $O(n^2)$. In order to exploit this structure and to find all rotations, Gusfield and Irving proposed an algorithm based on traversing a maximal chain in the lattice of stable matchings from M_0 to M_Z. All rotations are exploited when traversing this chain. Once a rotation is exposed in the current stable matching

Algorithm 3 Implementation of Algorithm *minimal-differences* [GI89]

```
1: procedure MINIMAL-DIFFERENCES()
2:     find M_0 and M_Z, and create the GS-lists using the Algorithm 2.
3:     i := 0
4:     set up an empty stack
5:     x := 1
6:     while x ≤ n do
7:         if stack empty then
8:             while (p_{M_i}(x) = p_{M_z}(x)) and (x ≤ n) do
9:                 x := x + 1
10:                if x ≤ n then
11:                    push x onto stack
12:        if stack not empty then
13:            m := man on top of stack
14:            m := next_{M_i}(m)
15:            while m not in stack do
16:                push m onto stack
17:                m := next_{M_i}(m)
18:            m' := top of stack
19:            pop stack
20:            set up list ρ_i containing the pair (m', p_{M_i}(m'))
21:            while m ≠ m' do
22:                m' := top of stack
23:                pop stack
24:                add the pair (m', p_{M_i}(m')) to the head of ρ_i
25:            output ρ_i
26:            M_{i+1} := M_i/ρ_i
27:            i := i + 1
28:            update reduced preference lists
```

M, it is eliminated from M. Then the next rotation exposed in M/ρ is found and the algorithm continues searching for rotations until the woman-optimal matching M_Z is obtained. Their algorithm called *minimal-differences* is given in Algorithm 3. Note that, this algorithm runs in $O(n^2)$ time.

All the rotations associated with the instance given in Figure 2.10 respecting the precedence relations are given in Figure 2.11. This figure represents the rotation poset of this instance.

A *closed subset* S is a set of rotations such that for any rotation ρ in S, if there exists a rotation $\rho' \in \mathcal{V}$ that is a predecessor of ρ, then ρ' is also in S. Given a $\rho = (m_0, w_0), (m_1, w_1), \ldots, (m_{l-1}, w_{l-1})$, the set of men $\{m_0, m_1, \ldots, m_{l-1}\}$ in ρ is referred by $x(\rho)$ and the set of women $\{w_0, w_1, \ldots, w_{l-1}\}$ by $y(\rho)$. Additionally,

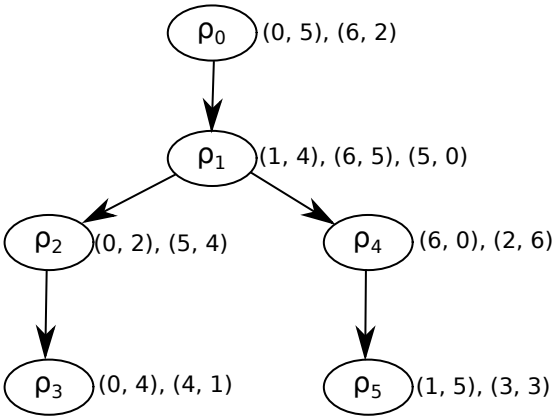


Figure 2.11: Rotation poset of the instance given in Table 2.1.

we denote by $X(S)$ the set of men that are included in at least one of the rotations in S. More formally, $X(S) = \bigcup_{\rho \in S} x(\rho)$. Gusfield and Irving showed the relationship between closed subsets and the stable matchings of an instance, presented in Theorem 3, stating that each closed subset in the rotation poset has a corresponding stable matching.

Theorem 3 (Theorem 2.5.7 [GI89]) *i) There is a one-one correspondence between the closed subsets of* Π *and the stable matchings of* $\mathscr{M}$.

ii) S is the closed subset of rotations of Π *corresponding to a stable matching* M *if and only if* S *is the (unique) set of rotations on every* M_0*-chain in* $\mathscr{M}$ *ending at* M. *Further,* M *can be generated from* M_0 *by eliminating the rotations in their order along any of these paths, and these are the only ways to generate* M *by rotation eliminations starting from* M_0.

iii) If S *and* S' *are the unique sets of rotations corresponding to distinct stable matchings* M *and* M', *then* M *dominates* M' *if and only if* $S \subset S'$.

By way of example, in Figure 2.10, the closed subset S_0 of the man-optimal matching M_0 corresponds to the empty set $S = \emptyset$. Similarly, for the matching M_3, the closed subset $S_3 = \{\rho_0, \rho_1, \rho_4\}$ and $X(S_3) = \{0, 1, 2, 5, 6\}$. For the woman-optimal matching $M_Z = M_{10}$ the closed subset is $S_{10} = \{\rho_0, \rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$ and $X(S_{10}) = \{0, 1, 2, 3, 4, 5, 6\}$.

2.4.2 Stable Roommates Problem

The *Stable Roommates problem (SR)* is a generalization of the Stable Marriage problem, where the sex factor is eliminated. It was proposed by Irving in

1985 [Irv85]. In the following year, Gusfield and Irving included a detailed chapter on SR in their famous book [GI89]. Unless stated otherwise, we refer to Gusfield and Irving's book for definitions in this section.

An instance of SR admits a set of n agents regardless of their gender, where each person expresses a ranking over all other people in the set in order of preference. Most of the definitions for the SM also hold for the SR. Let $P = \{p_1, p_2, \ldots, p_n\}$ denote the set of people in any given SR instance, where $n = 2 \times k, k \geq 0$. A *matching* corresponds to a partition of P into disjoint pairs (or sets of partners). If there does not exist any two persons $\{p_i, p_j\}$ in the set P that are not partners in matching M but they prefer each other to their partners in M, then M is said to be a *stable* matching. If any such pair exists, it is called a *blocking pair*. If a pair $\{p_i, p_j\}$ appears in at least one stable matching, then the pair is called a *stable pair*. If a pair appears in all stable matchings of the underlying instance, then the pair is a *fixed pair*. Any pair that is stable and not fixed is a *non-fixed pair*. Additionally, we refer to each of the persons appearing in a fixed pair as a *fixed person*, and appearing in a non-fixed pair as a *non-fixed person*. It should be noted that, the pairs are denoted by $\{p_i, p_j\}$ in the SR, in contrast to the SM, where a pair is (m_i, w_j). This is because the pairs in SR are unordered, whereas the pairs in SM are ordered by the gender as a man followed by a woman.

Gusfield and Irving show in Lemma 1 that given any SM instance, a corresponding SR instance can be created.

Lemma 1 (Lemma 4.1.1 [GI89]) *Given an instance of the stable marriage problem involving n men and n women, there is an instance (in fact there are many instances) of the stable roommates problem involving those $2n$ persons such that the stable roommates matchings are precisely the stable matchings for the original stable marriage instance.*

The method they use is by padding. For each person p, the idea is to pad all the people of the same gender to the end of the preference list of p in any order. This method removes the gender factor. A sample of how this padding works can be found in Table 2.2. By this conversion, any SM instance has a corresponding SR instance with the exact same stable matchings. Hence, the number of stable matchings in the SR instances also grows exponentially with respect to n.

Table 2.2 shows an instance of SM that admits three men $U = \{m_1, m_2, m_3\}$

Table 2.2: A sample SM instance (left) and its corresponding SR instance (right).

p_i	Preference List of p_i
m_1	2 3 1
m_2	3 1 2
m_3	2 3 1
w_i	Preference List of w_i
w_1	2 3 1
w_2	1 2 3
w_3	1 3 2

p_i	Preference List of p_i
p_1	4 6 5 2 3
p_2	6 4 5 1 3
p_3	5 6 4 1 2
p_4	2 3 1 5 6
p_5	1 2 3 4 6
p_6	1 3 2 4 5

and three women $W = \{w_1, w_2, w_3\}$ on the left, and its corresponding SR instance that admits six people $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ on the right. Observe the relationship between the men/women and the persons in those instances as: $p_1 = m_1, p_2 = m_2, p_3 = m_3, p_4 = w_1, p_5 = w_2, p_6 = w_3$. We used the lexicographic order when applying the padding. We also used i to denote m_i, w_i or p_i in the preference lists for enhancing the readability depending on the type of the problem. There exist two stable matchings for the SM instance: $M_1 = \{(m_1, w_1), (m_2, w_3), (m_3, w_2)\}$ and $M_2 = \{(m_1, w_3), (m_2, w_1), (m_3, w_2)\}$. Note that, there exists one and only one fixed pair (m_3, w_2). For the corresponding SR instance, there also exist exactly two stable matchings as $M_1 = \{\{p_1, p_4\}, \{p_2, p_6\}, \{p_3, p_5\}\}$ and $M_2 = \{\{p_1, p_6\}, \{p_2, p_4\}, \{p_3, p_5\}\}$, where the pair $\{p_3, p_5\}$ is a fixed-pair.

One of the most important differences between an SM and an SR instance is that although any SM instance can be shown to have at least one stable matching, an SR instance may contain no stable matchings at all. Any SR instance that admits at least one stable matching is said to be *solvable* and the latter case is *unsolvable*.

There exists an $O(n^2)$ algorithm to find a stable matching to a given SR instance or report if none exists. The algorithm works in two phases: *Phase 1* and *Phase 2*. Let us define some important concepts before discussing the algorithm. Given the set of people P, the set consisting of each persons preference list is called a *preference table*, denoted by T. The preference tables are modified during the execution of the algorithm; the modified ones correspond to *stable tables*. A preference table T is called *stable* if it satisfies the following properties:

1. $p_i = f_T(p_j)$ if and only if $p_j = l_T(p_i)$;
2. pair $\{p_i, p_j\}$ is absent if and only if p prefers $l_T(p_i)$ to p_j or p_j prefers $l_T(p_j)$

to p_i;

3. no person's list in T is empty.

A pair is said to *belong* to a table T, if the persons in the pair appear in their preference lists in T. A pair is said to be *deleted* from a table T, if they are deleted from each other's preference lists in T. For a given table T and a person p_i, the notations $f_T(p_i)$, $s_T(p_i)$, $l_T(p_i)$ denote, if any, the first, second and last entries in p_i's preference list in T, respectively.

Phase 1 is very similar to the Gale-Shapley algorithm proposed for solving the Stable Marriage problem (see Algorithm 2). It is based on each person proposing to the first available person on their lists. Any person that is currently not holding any proposals hold the first proposal they receive, and become *semi-engaged*. Moreover, if a semi-engaged person receives a proposal from a more preferred person than he/she currently has, he/she becomes semi-engaged to the more preferred person and the previous person becomes free. If a person p becomes semi-engaged to p', all pairs (p', p^*) such that p' prefers p to p^* are deleted from the table. The details of the procedure is presented in Algorithm 4. The table obtained after applying the Phase 1 algorithm is called the *phase-1 table* and is denoted by T_0. If at any step of the algorithm, any of the preference lists become empty, then the instance is immediately reported as unsolvable.

Consider the SR instance involving six people, given in Table 2.3, as running example throughout this section. Applying Algorithm 4 on the SR instance given in Table 2.3 results in the Phase-1 table given in Table 2.4.

In T_0, if each person's list contain exactly one entry, T_0 corresponds to a unique stable match, where each person is matched with the single entry in their preference lists. However, if at least two of the preference lists in T_0 contain more than one entry, the second phase of the SR algorithm is performed. The main idea behind the Phase 2 is to identify *rotations* exposed and eliminating them

Table 2.3: A sample SR instance of six people.

p_i	**Preference List of p_i**
p_1	4 5 2 3 6
p_2	6 1 4 5 3
p_3	2 1 6 5 4
p_4	2 5 6 3 1
p_5	3 4 1 2 6
p_6	1 5 3 2 4

Algorithm 4 Phase 1 of the SR algorithm

```
1: assign each person to be free
2: while some free person x has a nonempty list do
3:     y ← first person on x's list
4:     if some person z is semi-engaged to y then
5:         assign z to be free
6:     assign x to be semi-engaged to y
7:     for each successor x' of x on y'z list do
8:         delete the pair (x', y) from the preference table.
```

Table 2.4: Phase-1 table of the SR instance given in Table 2.3.

p_i	Preference List of p_i
p_1	4 5 2 3
p_2	6 1 4
p_3	1 6 5
p_4	2 5 1
p_5	3 4 1 6
p_6	5 3 2

starting from T_0 until the final table contains one entry at each list or empty. The general algorithm for the Phase 2 is given in Algorithm 5. The overall algorithm for SR takes $O(n^2)$ time.

The *rotations* in SR are represented as ordered lists of pairs and they are defined relative to the tables, and are analogous to the minimal differences described for Stable Marriage problem discussed in Algorithm 3, Page 43. A rotation ρ is denoted as $\rho = (x_0, y_0), (x_1, y_1), \ldots, (x_{r-1}, y_{r-1})$, where all $x_i, y_j \in P$. Each rotation sequence has the property that $y_i = f_T(x_i)$ and $y_{i+1} = s_T(x_i)$ for all i, $0 \leq i \leq r-1$, where i+1 is taken modulo r in table T. In this case, the rotation ρ is said to be *exposed* in table T. The set $\{x_0, \ldots, x_{r-1}\}$ of persons is called the *X-set* of ρ, denoted by $X(\rho)$. Similarly, $\{y_0, \ldots, y_{r-1}\}$ is called the *Y-set*, denoted

Algorithm 5 Phase 2 of the SR algorithm

```
1: T ← T_0
2: while (some list in T has more than one entry) and (no list in T is empty)
   do
3:     find a rotation ρ exposed in T
4:     T ← T/ρ                                   ▷ eliminate ρ
5:     if some list in T is empty then
6:         report instance unsolvable
7:     else
8:         output T                              ▷ T is a stable matching
```

by $Y(\rho)$. The X-set (or Y-set) of a set of rotations refers to the union of X-sets (or Y-sets) of all rotations in the rotation set. For any two rotations ρ_i and ρ_j, the X-sets of ρ_i, ρ_j are disjoint, as are their Y-sets. The positions of x_i and y_i in their respective lists are characterized in Lemma 2.

Lemma 2 (Lemma 4.2.7 [GI89]) *Let $\rho = (x_0, y_0), (x_1, y_1), \ldots, (x_{r-1}, y_{r-1})$ be a rotation exposed in table T. Then, if T/ρ contains no empty lists,*

i) $f_{T/\rho}(x_i) = y_{i+1}$ *for each i,* $0 \leq i \leq r-1$*;*

ii) $l_{T/\rho}(y_i) = x_{i-1}$ *for each i,* $0 \leq i \leq r-1$*;*

iii) $f_{T/\rho}(x) = f_T(x)$ *for each x not in the X-set of* ρ*, and* $l_{T/\rho}(y) = l_T(y)$ *for each y not in the Y-set of* ρ*.*

Given a table T and a rotation ρ exposed in T, *eliminating* ρ from T means that for each pair $(x_i, y_i) \in \rho$, the deletion of y_i from x_i's list and also the deletion of all pairs $\{y_i, z\}$ such that y_i prefers x_{i-1} to z from T. The table after eliminating ρ from T is denoted by T/ρ. The rotation ρ is said to move x_i down from y_i to y_{i+1}, and move y_i up from x_i to x_{i-1} when ρ is eliminated from T and does not produce any empty lists.

There are two types of rotations: *singular* and *non-singular*. A rotation $\rho = (x_0, y_0), (x_1, y_1), \ldots, (x_{r-1}, y_{r-1})$ is called a non-singular rotation if $\bar{\rho} = (y_1, x_0)$, $(y_2, x_1), \ldots, (y_0, x_{r-1})$ is also a rotation. In this case, ρ and $\bar{\rho}$ are called *duals* of each other. If a rotation does not have a dual, then it is a singular rotation.

Gusfield and Irving characterize the pairs with respect to the tables and rotations as presented in Lemma 3.

Lemma 3 (Lemma 4.4.1 [GI89]) *In a solvable roommates instance,*

i) $\{x, y\}$ *is a fixed pair if and only if x's list in the reduced Phase-1 table contains only y and y's contains only x;*

ii) otherwise, $\{x, y\}$ *is a stable pair if and only if the pair* (x, y)*, or the pair* (y, x)*, is in a nonsingular rotation.*

Now, we demonstrate Algorithm 5 on the Phase-1 table presented in Table 2.4. There are two rotations exposed in table T_0. These rotations are $\rho_0 = (1, 4)$, $(6, 5), (5, 3)$ and $\rho_1 = (2, 6), (3, 1)$. Eliminating both ρ_0 and ρ_1 results in the table T' given in Table 2.5. On table T', there is one more rotation exposed: $\rho_2 = (1, 5), (4, 2)$. Note that, eliminating ρ_2 from T' results in reducing all lists to one entry and, therefore, corresponds to the stable matching

Table 2.5: Table T' after elimination of the rotations ρ_0 and ρ_1.

p_i	Preference List of p_i
p_1	5 2
p_2	1 4
p_3	6
p_4	2 5
p_5	4 1
p_6	3

$M = \{\{p_1, p_2\}, \{p_3, p_6\}, \{p_4, p_5\}\}$.

The next step is to identify which rotations are singular, which ones are non-singular and to create duals of the non-singular ones accordingly. The test for deciding the type of a rotation ρ can be done by applying Algorithm 5, starting from T_0, eliminating a set of the exposed rotations without eliminating ρ. If a solution can be found without eliminating ρ, it means that $\bar{\rho}$ exists. This test is a brute force approach and, therefore, needs to be repeated for each individual rotation. Applying this test on the rotations found for the instance given in Table 2.3, we find out that ρ_0 is a singular rotation and ρ_1, ρ_2 are both non-singular rotations. Additionally, the duals of ρ_1 and ρ_2 are found as follows: $\bar{\rho_1} = (1, 2), (6, 3)$ and $\bar{\rho_2} = (2, 1), (5, 4)$. Therefore, our instance contains five rotations in total, $\rho_0, \rho_1, \rho_2, \bar{\rho_1}$, and $\bar{\rho_2}$.

Similar to the definition of rotation posets in SM, the set of both singular and non-singular rotations under $\prec$ defines the *roommates rotation poset*. A rotation π is a *predecessor* of a rotation ρ, denoted by $\pi \prec \rho$, if, whenever T is a table in which ρ is exposed and $T = T_0/Z$, π belongs to the set of rotations Z. We refer to any two rotations ρ, σ in Π such that $\sigma \neq \bar{\rho}$ as *incomparable* if none of them precede the other one, *comparable* otherwise. For two rotations exposed in a table, if the rotations are not duals, then the order of eliminating them does not matter (see Lemma 4).

Lemma 4 (Lemma 4.3.2 [GI89]) *If ρ and σ are rotations exposed in a table T, and $\sigma \neq \bar{\rho}$, then $(T/\rho)/\sigma = (T/\sigma)/\rho$.*

A subset of the rotations in the roommates rotation poset such that the set contains at least one of each dual rotations is called a *complete subset*. Furthermore, for each rotation in this subset, if all predecessors of the rotation is also in the subset, then it is called a *complete closed subset*. The set of only non-singular rotations under $\prec$ also forms a partially ordered set and is called the *reduced rotation poset* denoted by Π.

Gusfield and Irving showed that there exists a one-one correspondence between the complete closed subsets of the reduced rotation poset and the stable matchings of the underlying instance [GI89]. They also provided a characterization for the singular and non-singular rotations, including the precedence relation between them, as presented in Lemma 5. This characterization points out to three properties of rotations. The first one states that given two non-singular rotations, where one of them is the dual of the other one, these two rotations are always incomparable. Second, given two comparable rotations ρ and σ, where ρ precedes σ, their duals are also comparable and $\bar{\sigma}$ precedes $\bar{\rho}$. Finally, the last property states that all rotations that precede a singular rotation are also singular.

Lemma 5 (Lemma 4.3.7 [GI89]) *If ρ, σ are non-singular and π is a singular rotation, then*

i) $\rho \not\prec \bar{\rho}$

ii) $\rho \prec \sigma \Longleftrightarrow \bar{\sigma} \prec \bar{\rho}$

iii) $\tau \prec \pi \Longrightarrow \tau$ *is singular; i.e. a predecessor of a singular rotation is also singular.*

Any stable matching can be obtained by eliminating all singular rotations followed by one of each of the dual rotations from the Phase-1 table T_0. We denote by T_S the table where all singular rotations are eliminated from T_0. The construction of Π can be achieved in $O(n^2)$ time analogously to the construction of the SM rotation poset, once all the non-singular rotations are found. The overall complexity then is $O(n^4)$. This time can be further reduced to $O(n^3 logn)$ by using a more subtle algorithm, but we use the brute force approach.

Recall that we identified each rotation for the SR instance given in Table 2.3. The roommates rotation poset for this instance with rotation set $\{\rho_0, \rho_1, \rho_2, \bar{\rho_1}, \bar{\rho_2}\}$ (left), and also the reduced rotation poset of the same instance with the rotation set $\{\rho_1, \rho_2, \bar{\rho_1}, \bar{\rho_2}\}$ (right) are given in Figure 2.12.

There exist two different complete closed subsets in the reduced rotation graph Π given in Figure 2.12, namely $S_1 = \{\rho_1, \bar{\rho_2}\}$ and $S_2 = \{\bar{\rho_1}, \rho_2\}$. These two complete closed subsets correspond to the two stable matchings of the underlying instance. Applying all the singular rotations and each of the rotations in the given subset starting from T_0 results in those stable matchings, which are $M_1 = \{\{p_1, p_5\}, \{p_2, p_4\}, \{p_3, p_6\}\}$ and $M_2 = \{\{p_1, p_3\}, \{p_2, p_6\}, \{p_4, p_5\}\}$ corre-

sponding to the S_1 and S_2, respectively.

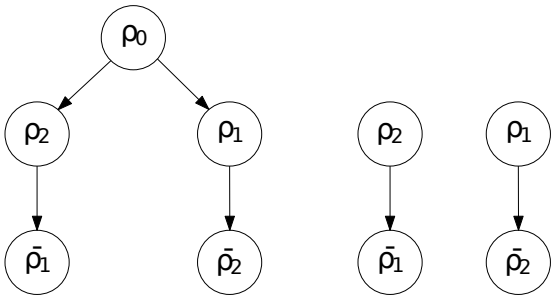


Figure 2.12: The roommates rotation poset (left) and the reduced rotation poset (right) for the instance given in Table 2.3.

2.5 Robust Optimization

Robust Optimization (RO) is a very broad area that dates back to 1970s [Soy73]. We start with a motivational example followed by the general view of the area and then narrow our focus down to a specific field.

We start with motivating the robustness in the context of a flight scheduling problem. It is well-known that the airlines lose huge amounts of money by rescheduling their flights due to unexpected events [SSDS02, Yua09]. Therefore, it is desirable to find schedules such that if an emergency landing must be made, a crew member cannot attend a flight, a flight is cancelled, etc., then an alternative arrangement can be found at a minimum cost or requiring a limited number, if any, additional changes.

Examples of the most active robust optimization fields can be broadly listed as planning, scheduling, logistics, finance, etc [GMT14]. The need for robustness in optimization arises because many problems, especially in the real-world, are usually sensitive to perturbations such as measurement mistakes, errors in data, or they are lacking a clear objective [IS16]. Some recent comprehensive reviews of robust optimization [GMT14, ST16, BBC11] and a textbook [BTGN09] are available in the literature.

In general, in a robust optimization model, uncertain parameters that are derived from noisy, incomplete, or erroneous data are handled as random variables with discrete distributions [XHQC09]. One main issue in RO is that the term *robustness* has many different definitions. These different definitions do not only appear between different fields but also different robustness notions

exist within the same field or problem. Some of these fields that include ambiguous definitions can be listed as: scheduling and timetabling [SBI12], CP models [BS15], machine learning [XCM09], economics [Woo06].

We give some concrete examples on some different robustness notions below. For example, in stochastic constraint programming, a robust solution is one that is consistent with similar decisions made in different scenarios [TMW06]. On the other hand, in robust sensor network design, different levels of uncertainty are defined to evaluate the robustness of a network that is not effected by node failures or some other technical errors [BNR08, KBG08, KBG07]. Another approach to robustness is to find solutions that are robust to changes and can be repaired by minor changes. This notion of robustness motivated by providing small repairs, sometimes also referred as fault-tolerance, was founded by Ginsberg by (a, b)-supermodels in SAT and also extended to CP by Hebrard et al. under the name (a, b)-super solutions [GPR98, HHOW05, HHW04a]. Models that take into account robustness are generally obtained by choosing the parameters of the solution such that the performance of the solution is less influenced by negative effects of the uncertainty [SAFP14].

The robustness notions presented in Boolean satisfiability and constraint programming frameworks are the main inspirations to the robustness notion we introduce in matching problems. Therefore, we focus on the robustness notions being used in these fields.

CP and SAT

Robustness in SAT and CP form one of the branches of the general RO field. The need for robustness in CP emerged due to the need to provide robust solutions that are not brittle to changes in the environment, or uncertainty and noise in data [HS96]. A solution that is not robust may not remain a solution in the case of an error or it may require many modifications if an unexpected event occurs and/or it might be costly to compute a new solution within the required number of maximum changes [RvBW06].

There exist a variety of definitions within the context of CP and SAT [BS15]. We exclude the robust SAT/CSP solvers or their search strategies in this review. Robustness in the context of solvers refers to having ability to solve a variety of instances within a specified time frame [GN07, FH00, MH12]. Our focus is on the ability of an individual solution that is produced to withstand particular

events of change.

There is some research in the area that considers robustness as: *"a robust solution has a high probability to remain solution after changes in the environment"* [CWSB14, BS15]. This is the most common use of the term robustness. Another definition that exists defines a solution as *robust* if *"there exists another solution that can be found through a small number of modifications to the current solution"* [GPR98, HHW04b, HHW04a, HO04]. The Handbook of Constraint programming also considers the robustness as *"solutions that are likely to remain solutions even after the change has occurred, or to need only minor repairs"*. However, some research characterize this notion of repairs as fault-tolerance [Heb07] or recoverability [DSOI05].

In Section 2.5.1 and Section 2.5.2, we look into details of some models that use the robustness as "there exists another solution that can be found through a small number of modifications to the current solution".

2.5.1 (a,b)-supermodels

In 1998, Ginsberg, Parkes and Roy introduced (a, b)-supermodels as a measure of robustness in the SAT framework [GPR98, Roy01]. Ginsberg et al. also proved that determining whether a formula has an (a, b)-supermodel is $\mathcal{NP}$-complete [GPR98]. In Ginsberg's work the (a, b)-supermodels are referred as robust solutions. However, later on Roy denotes them by $\gamma-$models, and referring to them as robust solutions as an approach to fault-tolerance in his work [Roy06]. The formal definitions for both of these concepts are given in Definition 5 and Definition 6.

Definition 5 *[GPR98] An (a, b)-supermodel is a model such that if we modify the values taken by the variables in a set of size at most a (breakage), another model can be obtained by modifying the values of the variables in a disjoint set of size at most b (repair).*

Definition 6 *[Roy01] A $\gamma-$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 ith bit of α, there is another bit $j \neq i$ of α which we an negate to get another satisfying assignment.*

Example. We demonstrate (a, b)-supermodels on the following formula that contains three literals x_1, x_2, and x_3:

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3}) \tag{2.1}$$

There exists 6 different assignments for those literals that satisfy the formula. These assignments, referred as $A_i, i \in [1, 6]$, are listed in Table 2.6.

Let us first demonstrate the concept of $(1, b)$-supermodels using the assignment $A_1 = (0, 0, 1)$. If x_1 loses its value, $(1, 0, 1)$ is not a satisfying assignment but changing one other literals value such as x_2, a satisfying assignment $A_6 = (1, 1, 1)$ can be obtained. Similarly, if x_2 loses its value, $(0, 1, 1)$ is a satisfying assignment and no repairs are required. Lastly, if x_3 loses its value, $(0, 0, 0)$ is not a satisfying assignment but by changing one other, such as x_1, $A_4 = (1, 0, 0)$ can be found as a satisfying assignment. Therefore, the assignment A_1 is a $(1, 1)$-supermodel because the loss of a value for any single variable ($a = 1$) may require, in the worst case, one other variable to change its value to obtain another satisfying assignment. On the other hand, by following the same reasoning, we conclude that $A_5 = (1, 1, 0)$ is a $(1, 0)$-supermodel because, for each possible break, each of $(0, 1, 0), (1, 0, 0), (1, 1, 1)$ are satisfying assignments.

Next, we demonstrate $(2, b)$-supermodels on the same model. This time, all size 2 combinations of literals must be checked for a value loss. For A_1, if any of the pairs $\{x_1, x_2\}$, $\{x_1, x_3\}$, or $\{x_2, x_3\}$ lose their values at the same time, no further repairs are required as $(1, 1, 1)$, $(1, 0, 0)$ and $(0, 1, 0)$ are all satisfying assignments. Thus, A_1 is also a $(2, 0)$-supermodel. On the other hand, A_3 is a $(2, 1)$-supermodel because if x_2, x_3 lose their value at the same time, x_1 must be changed to provide a repair. Additionally, if x_1, x_2 lose their value, x_3 must be changed. However, no repairs are required if x_1, x_3 lose their value.

Table 2.6: Satisfying assignments of the sample SAT formula given in Equation 2.1.

$\boldsymbol{A_i}$	$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{x_3}$
A_1	0	0	1
A_2	0	1	0
A_3	0	1	1
A_4	1	0	0
A_5	1	1	0
A_6	1	1	1

2.5.2 (a,b)-super solutions

The (a, b)-super solutions framework is an extended version of (a, b)-supermodels to the constraint programming setting [HHW04b, HHW04a, Heb07, HW05]. Hebrard et al. define (a, b)-super solutions as a generalization of both fault-tolerant solutions in CP and (a, b)-supermodels. An (a, b)-super solution is a solution which if any a variables break, the solution can be repaired by providing repair by changing a maximum of b other variables. They use a notion of distance denoted by $\Delta_A(f, g)$ for two assignments f and g on a set A. Then, they define *repairability* of a solution as:

Definition 7 *A breakage set A is a subset of variables ($A \subset X$). A b-repair of a breakage A for a solution f is a solution g such that $\Delta_A(f, g) = |A|$ and $\Delta(f, g) \leq |A| + b$.*

They also show that the $\mathcal{NP}$-completeness of (a, b)-supermodels lifts to (a, b)-super solutions, and that finding (a, b)-super solutions is also $\mathcal{NP}$-complete. Their main focus, however, is on finding $(1, b)$-super solutions.

As a related term, h-recoverability is defined by Barber and Salido as an $(h, 0)$-super solution (i.e. $a = h$, $b = 0$) [BS15]. They point out the difference being only as the variables to be repaired are consecutive over time for h-recoverability, but in $(h, 0)$-super solutions, they are not consecutive. Hebrard et al. remark that it is rare for problems in general to have solutions where all possible breaks are repairable. Therefore, they look for the *most robust* solution, which maximizes the number of repairable variables by also minimizing the repair costs [Heb07].

Policella, in her thesis, mentions one of the main disadvantages of using (a, b)-super solutions as considering only the number of changes and not their magnitudes [Pol05]. At the same time, Holland and O'Sullivan extend the notion of (a, b)-super solutions to *weighted super solutions (WSS)*, and also use them for finding robust solutions to combinatorial auctions or job shop scheduling [HO04, HO05b, HO05a]. Bofill et al. also work on the robustness of solutions in combinatorial auctions mostly using the (a, b)-supermodels [BBV03]. Holland and O'Sullivan define a solution as robust if there exists an alternative repair solution should some assignments break. Their main contributions are to use a probabilistic approach to model which assignments are more likely to break and also a a notion of repair cost. They define a static and a dynamic version of weighted super solutions as follows.

Definition 8 (Static WSS) *A solution to a CSP is a static weighted super solution, or (α, β)-static WSS, if any set of variables whose probability of losing their current assignments is greater than or equal to α, can be repaired by reassigning other values to these and other variables with a repair cost of at most β.*

Definition 9 (Dynamic WSS) *A solution to a CSP is a dynamic weighted super solution, or (α, β, τ)-dynamic WSS, if any set of variables whose probability of losing their current assignments is greater than or equal to α before time τ, can be repaired by reassigning other values to these and other variables with a repair cost of at most β.*

Considering that the (a, b)-super solutions are defined within the CP networks, the proposed algorithms for finding robust solutions are mainly based on MAC (Maintaining Arc Consistency algorithm) [SF94] by using backtracking methods.

2.5.3 Discussion on (a,b) models

As discussed throughout Section 2.5, there exist many different notions of robustness in RO. Following the definitions given above, let us re-write two commonly used definitions below.

Definition 10 *A solution is robust, if it still remains as a solution in the case of erroneous input, unexpected failures, etc.*

Definition 11 *A solution is fault-tolerant, if it is guaranteed to find another solution by applying small modifications in the case of unexpected events.*

Now, let us partition (a, b)-supermodels and (a, b)-super solutions (we shall refer them as (a, b) models for simplicity) into two, where one of the cases is when $b = 0$ and the other one is when $b \geq 1$. Using the Definitions 10 and 11, one can characterize the (a, b) models as either defining robustness or fault-tolerance (or recoverability). More specifically, the (a, b)-supermodels and $\gamma-$models can be referred as fault-tolerant or recoverable when $b \geq 1$. On the other hand, $(a, 0)$-supermodels provide robust solutions because they do not require any further modification/repairs. In the overall, the models above (referred as (a, b) models) define a general notion of robustness by also using fault-tolerance framework.

2.5.4 Robustness Notions in Matching Problems

Robustness in matching problems is a new and active area. Considering that matching problems is a true interdisciplinary area, the first appearance of robustness dates back to 2011 studied in economics, within the context of Matching Markets by Kojima [Koj11]. Kojima's robustness notion is motivated by the students misreporting their preferences to manipulate the matchings in the central assignment systems such as assignment of students to schools.

These kind of preference manipulations in the literature have been studied under the notion of strategy-proofness. In this context, a *market* is defined by the set of agents and their preferences. A *mechanism* defines a function over the preferences to the set of all matchings. A mechanism is *stable*, if no two agents, one from each side of the market, prefer each other over the partners with whom they are matched [AG15]. It is strategy-proof if no individual student has an incentive to misreport his/her preference [Afa12]. Then, a mechanism is defined by Kojima as *robustly stable* if it is stable, strategy-proof, and also immune to a combined manipulation, where a student first misrepresents his or her preferences and then blocks the matching that is produced by the centralized mechanism. Afacan extended the concept of robust stability proposed by Kojima, to *group robust stability*, where a group of students misrepresent their choices for a manipulation [Afa12].

Later on, some researchers focused on including robustness for stable matchings when uncertainty in the preferences is present. In 2013, Drummond and Boutilier proposed to use minimax regret as a measure of robustness of stable matchings for the Stable Marriage problem [DB13]. Their work is motivated by the incomplete information provided by agents. The use of minimax regret provides robustness by minimizing the worst-case loss.

A recent study from Menon and Larson focuses on finding "good" solutions (i.e. a matching with the least number of blocking pairs in expectation) and refer to a good solution as a robust one [ML18]. They mostly focus on providing algorithms to find such matchings.

Aziz et al. considered different models to study this problem and mostly focused on the complexity of the problem [ABG+16]. They define a *stability probability* as the probability of a matching being stable. Then, they model a number of problems related to stability probability using the uncertainty and provide rich complexity results, showing some of those problems are $\mathcal{NP}$-hard, $\mathcal{NP}$-

complete or $\#P$-complete, also leaving some open problems.

One of the most recent robustness notions has been proposed by Jacobovic on the Stable Marriage problem, where the main idea is to use a probability model and a social cost function to measure robustness [Jac16]. They are motivated by some agents leaving the stable matching after it has been constructed. For instance, a student changes his/her decision to go to the college that he/she has been assigned to. They propose their version as *perturbation robust stable matching*, by introducing a probabilistic model that finds the possibility of agents to leave, or the probabilities are assumed to be known. They focus on the case in which only one agent is allowed to leave the stable matching at a time and show that $v - perturbation$ robust stable matching is solvable in polynomial-time by reducing the problem to a max-flow problem.

Mai and Vazirani also work on a robust version of SM [MV18]. Their robustness notion is defined over the errors in the input. These errors are defined by *swaps* or *shifts*. A swap occurs in the preference lists of a person, where the positions of two adjacent people in the list are swapped. A shift occurs if a person wants to move up a person's position in his/her list and shift all the remaining people accordingly. They provide a polynomial-time solution based on finding "nearby" instances by using the lattice structures. The problem they solve is to finding a stable matching for a given Stable Marriage instance that maximizes the probability of being stable for the nearby instance obtained after introduction of an error. Their work is also allowing only one error at a time.

Mai and Vazirani discuss our work on robustness in matching problems as a misnomer in their work [MV18]. They suggest that "fault-tolerance" is a more appropriate term for the robustness notion that we proposed. However, we use the robustness definition that appears in the Handbook of Constraint Programming, defined at the beginning of Section 2.5. As we discussed in Section 2.5, (a, b)-supermatches (the formal definition is given in Definition 12 in Chapter 3, Page 64) define a notion of robustness for matching problems.

Note that, most of the papers investigated in this section have been published as e-prints. To the best of our knowledge, there do not exist any formally peer-reviewed versions for the associated publications [ML18, ABG+16, Jac16, MV18] at the time this thesis being written. Although the two research papers; one of them by Jacobovic, and the other one by Mai and Vazirani contain overlapping aspects with our research, our notion of robustness is completely different as can be seen in Chapter 3.

2.6 Chapter Summary

This chapter reviews the definitions and notation relevant to this dissertation. Two mathematical structures: graphs and partially ordered sets have been introduced as they are being widely used in the technical chapters. A formal definition for optimization problems has been given and a detailed information of the field including complexity, modelling and search techniques have been presented. Subsequently, formal definitions of the two matching problems, namely the Stable Marriage and the Stable Roommates problems have been presented in detail. The chapter is concluded by discussing the Robust Optimization by mainly focusing on the different robustness notions in optimization problems. We also provide a discussion of these notions in the context of (a, b)-supermodels and (a, b)-super solutions. Subsequent chapters will introduce new notation and definitions that are specific to the work presented in this dissertation at the beginning of the relevant chapters.

Chapter 3

Robust Stable Marriage

Abstract. *In this chapter, we propose a notion of robustness for the stable matching problems called (a, b)-supermatches. We first formulate the problem on the Stable Marriage framework. We refer to the problem of finding (a, b)-supermatches for Stable Marriage problem as the Robust Stable Marriage problem (RSM). Then, we present a formulation, based on finding independent sets, for finding the $(1, 1)$-supermatches of the underlying instance. We prove the $\mathcal{NP}$-completeness of deciding if there exists a $(1, 1)$-supermatch to a given Stable Marriage instance by showing a reduction based on a special case of SAT, which we also prove is $\mathcal{NP}$-complete. We conclude the chapter by identifying some cases of RSM that are in $\mathcal{P}$ and $\mathcal{NP}$-hard.*

3.1 Introduction

The *Stable Marriage problem (SM)* is one of the most popular variants of the stable matching problems with an active history of more than fifty years [Man13]. The SM provides an interesting and rich framework to study new concepts mostly because all SM instances have at least one stable matching and their structural properties are well studied.

In Section 1.1, we provided motivation for the need of a notion of robustness for the Hospital/Residents (HR) problem, a generalization of the SM. If men (or women) in SM are abstracted as the residents, and women (or men) as the hospitals, then with the restriction of having the capacities of each hospital

set to one, we obtain a Hospital/Residents instance. In that example, we mentioned about the robustness as a relocation cost for a number of pairs. In the context of SM, the robustness investigates the number of break-up requests of man-woman pairs by considering the cost of finding new partners to the broken pairs.

3.2 Notation and Definitions

In this section, we characterize and formally define the notion of robustness we propose for stable matching problems and also formulate our notation. We use the Stable Marriage problem as the framework to define our notion.

Informally, an (a, b)-*supermatch* is a stable matching such that if any a men in the matching break-up from their current partners, it is possible to find another stable matching by changing the partners of those a men and also changing the currently assigned partners of at most b other men. Note that, the term "men" can be replaced by "women", or "pairs" in the definition of (a, b)-supermatches without loss of meaning. Although we use a man-oriented language in this work (for example, we say that the "man" wishes to break-up), there is no sexist meanings to it. Also note that, each man or woman in a stable matching refers to a unique pair. Therefore, the break-up of a man/woman can also be considered as losing the pair in which the specified man/woman is involved.

Let us first define the terminology to be used throughout this dissertation. Given a stable matching M and a set of pairs $\Psi \in M$, when searching for another stable matching M', where none of the pairs in Ψ are in M', we say that the pairs in Ψ wish to *break-up*. We also say that any pair $(m_i, w_j) \in \Psi$ is a *breakage*, or m_i *wishes* to break-up, interchangeably. Subsequently, the men in $\Psi \in M$ are said to *lose* their partners to obtain M'.

Let M_i, M_j and M_k be three different stable matchings. We measure the ***distance*** between two stable matchings, denoted by $d(M_i, M_j)$, as the number of men that have different partners between M_i and M_j. Considering the discussion above, this distance can also be computed as the number of pairs in M_i that are not in M_j (or vice versa). Note that, this notion is analogous with the Hamming distance. Then, we can infer that M_j is closer to M_i than M_k if $d(M_i, M_j) < d(M_i, M_k)$. We say that a ***repair matching*** for the breakage of Ψ in M is a stable matching M' that minimizes the value of $d(M, M^*)$ taken over

every other stable matching M^* such that $M^* \cap \Psi = \emptyset$. The ***repair cost*** in this context is the value $d(M, M') - |\Psi|$.

The work presented in this chapter is mostly based on the rotations and structural properties of the rotation posets of the SM. Recall briefly a few concepts. A *rotation* $\rho = ((m_{k_0}, w_{k_0}), (m_{k_1}, w_{k_1}), \ldots, (m_{k_{l-1}}, w_{k_{l-1}}))$, where $l \in \mathbb{N}^*$, is an ordered list of pairs in a stable matching M such that changing the partner of each man m_{k_i} to the partner of the next man $m_{k_{i+1}}$ (the operation $+1$ is modulo l) in the list ρ leads to a stable matching denoted by M/ρ. The latter is said to be obtained after eliminating ρ from M. In this case, we say that (m_{l_i}, w_{l_i}) is ***eliminated*** by ρ, whereas $(m_{l_i}, w_{l_{i+1}})$ is ***produced*** by ρ, and that ρ is *exposed* in M. All stable matchings of an instance form a lattice, and there exists a partial order for the rotations. If a pair appears in at least two stable matchings, it is a non-fixed pair; otherwise a fixed-pair.

A set of rotations is called a *closed subset* S, if for each rotation $\rho \in S$, all predecessors of ρ are also in S. Then, let **L(**S**)** be the set of rotations that are the sink nodes of the graph induced by the rotations in a given closed subset S. Similarly, let **N(**S**)** be the set of the rotations that are not included in S that either have no incoming edges or have all their predecessors in S.

Consider the rotation poset Π presented in Figure 2.11 (Page 44) and a closed subset $S = \{\rho_0, \rho_1, \rho_2\}$ on it. These sets for S are identified as **L(**S**)** = $\{\rho_2\}$ and **N(**S**)** = $\{\rho_3, \rho_4\}$. Formally, for each $\rho \in$ **N(**S**)** either $d_{in}(\rho) = 0$ or for all $\rho' \prec \rho$, $\rho' \in S$. This can be illustrated as having a cut in a given graph Π, where the cut divides Π into two sub-graphs, namely Π_1 and Π_2. If there are any comparable nodes between Π_1 and Π_2, let Π_1 be the part that contains the preceding rotations. Eventually, Π_1 corresponds to the closed subset S, **L(**S**)** corresponds to the sink nodes of Π_1. Additionally, **N(**S**)** corresponds to the source nodes of Π_2. We refer to the rotations in **L(**S**)** as the ***sink rotations of*** S and rotations in **N(**S**)** as the ***neighbour rotations of*** S. Figure 3.1 illustrates a cut on the Π in Figure 2.11, and the two sub-graphs, where the sets of interests are highlighted.

Observe that, each rotation in the rotation poset contains a minimum of two pairs. We introduce a function $\boldsymbol{R}^*(\boldsymbol{V})$ that takes a set of rotations denoted by V as input, and returns only the rotations that are of size 2 (i.e. has 2 pairs) in this set.

Additionally, in order to avoid repetition, we use a notation for stable match-

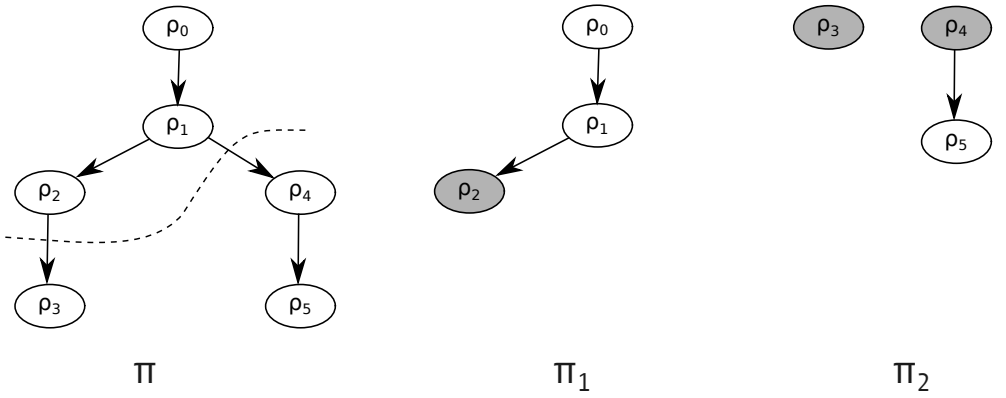


Figure 3.1: A closed subset $S = \{\rho_0, \rho_1, \rho_2\}$ in Π, the sub-graphs Π_1, Π_2 after the cut, and the sets **L(**S**)** $= \{\rho_2\}$, and **N(**S**)** $= \{\rho_3, \rho_4\}$ highlighted in Π_1 and Π_2, respectively.

ings and their corresponding closed subsets such that if a stable matching is identified using some superscripts or subscripts, then its corresponding closed subset contains them as well (i.e. the closed subset of M_i^j is denoted by S_i^j).

3.3 (a,b)-supermatches

In this section, we first formally define the problem of finding an (a, b)-supermatch, and then discuss in practice, how one can be found. We give in Definition 12 a formal definition for (a, b)-supermatches.

Definition 12 ((a,b)-supermatch) *Given an SM instance $\mathcal{I}$, a stable matching M of $\mathcal{I}$ is said to be an (a, b)-supermatch if for any set $\Psi \subseteq M$ of non-fixed stable pairs, where $|\Psi| = a$, there exists a stable matching M' such that $M' \cap \Psi = \emptyset$ and $d(M, M') \leq b + a$, where $a, b \in \mathbb{N}$.*

We refer to the problem of deciding, and eventually finding, if there exists an (a, b)-supermatch to a given Stable Marriage instance as the ***Robust Stable Marriage problem (RSM)***.

Recall that, due to the Rural Hospitals Theorem (see Theorem 2, Page 41), the same set of agents are assigned in all stable matchings. Therefore, within the context of RSM, if any a men who are matched in M lose their partners, we cannot expect to find another stable matching without those a men becoming matched to a partner again.

Let us give an insight into the identification of (a, b)-supermatches. Table 3.1 is a list of all the stable matchings given in Figure 2.10, Page 40.

Table 3.1: The list of all stable matchings given in Figure 2.10.

Stable Matching	**Pairs**
M_0	$\{(0,5),(1,4),(2,6),(3,3),(4,1),(5,0),(6,2)\}$
$M_1 = M_0/\rho_0$	$\{(0,2),(1,4),(2,6),(3,3),(4,1),(5,0),(6,5)\}$
$M_2 = M_1/\rho_1$	$\{(0,2),(1,5),(2,6),(3,3),(4,1),(5,4),(6,0)\}$
$M_3 = M_2/\rho_4$	$\{(0,2),(1,5),(2,0),(3,3),(4,1),(5,4),(6,6)\}$
$M_4 = M_3/\rho_5$	$\{(0,2),(1,3),(2,0),(3,5),(4,1),(5,4),(6,6)\}$
$M_5 = M_2/\rho_2$	$\{(0,4),(1,5),(2,6),(3,3),(4,1),(5,2),(6,0)\}$
$M_6 = M_5/\rho_4 = M_3/\rho_2$	$\{(0,4),(1,5),(2,0),(3,3),(4,1),(5,2),(6,6)\}$
$M_7 = M_4/\rho_2 = M_6/\rho_5$	$\{(0,4),(1,3),(2,0),(3,5),(4,1),(5,2),(6,6)\}$
$M_8 = M_5/\rho_3$	$\{(0,1),(1,5),(2,6),(3,3),(4,4),(5,2),(6,0)\}$
$M_9 = M_6/\rho_3 = M_8/\rho_4$	$\{(0,1),(1,5),(2,0),(3,3),(4,4),(5,2),(6,6)\}$
$M_{10} = M_7/\rho_3 = M_9/\rho_5$	$\{(0,1),(1,3),(2,0),(3,5),(4,4),(5,2),(6,6)\}$

We give the basic intuition behind determining the robustness of a stable matching in the context of a $(1,b)$-supermatch on a stable matching in this list, namely M_2. In order to find the value of b for M_2, where $a = 1$, one can simply compute the distance from M_2 to every other stable matching M of the underlying instance (listed in Table 3.1) to find the closest stable matching M' in case of the breakage of each pair $(m_i, w_j) \in M_2$. Then, the stable matching M' that minimizes the distance $d(M_2, M')$ is reported as the repair stable matching for the breakage of (m_i, w_j).

- $(m_0, w_2) \rightarrow M' = M_5$, $d(M_2, M_5) = 2$
- $(m_1, w_5) \rightarrow M' = M_1$, $d(M_2, M_1) = 3$
- $(m_2, w_6) \rightarrow M' = M_3$, $d(M_2, M_3) = 2$
- $(m_3, w_3) \rightarrow M' = M_4$, $d(M_2, M_4) = 4$
- $(m_4, w_1) \rightarrow M' = M_8$, $d(M_2, M_8) = 3$
- $(m_5, w_4) \rightarrow M' = M_5$, $d(M_2, M_5) = 2$
- $(m_6, w_0) \rightarrow M' = M_3$, $d(M_2, M_3) = 2$

Considering that $a = 1$, the b values are computed by subtracting 1 from all the computed distance values (see Definition 12). Because, the value of b represents the maximum value of the additional number of break-ups required to obtain another stable matching. Then, one can infer that if any of the men from the following set $\{m_0, m_2, m_5, m_6\}$ wishes to break-up, a repair matching at a repair cost of 1 can be found. On the other hand, for m_1 and m_4 the cost is 2 and for m_3 the cost is 3. Hence, M_2 is a $(1,3)$-supermatch can be read as: "if any of

the men loses his current partner in M_2, an alternative stable matching can be found by changing at most three other men's partners". Similarly, the reader can verify that stable matching M_6 is a $(1,1)$-supermatch by following the same procedure. It is left as an exercise to the reader.

Furthermore, in order to check if a stable matching M is a $(2,b)$-supermatch, all combinations of two different pairs $(m_i, w_j), (m_k, w_l) \in M$ must be considered for break-up. The M is compared with all the other stable matchings M', where $(m_i, w_j), (m_k, w_l) \notin M'$. Any such M' that minimizes the distance with M represents the repair stable matching for the break-up of those two men. Then, the maximum repair cost for all combinations of two pairs indicates the robustness of M.

An important remark is that there is a one-to-one correspondence between the stable matchings in $\mathcal{I}$ and the sets of incomparable rotations in $\mathcal{V}$ as shown in Proposition 1. Note that, a closed subset is defined by adding all predecessors of each node in the subset to the subset. Equivalently, if all rotations that precede some other rotations in S are removed from S, the resulting set corresponds to a set of incomparable nodes, namely the sink rotations as **L(**S**)**.

Proposition 1 *There is a one-to-one correspondence between the incomparable rotations and the stable matchings in the underlying instance.*

Proof. $\Rightarrow$ Let V_I denote a set of incomparable rotations in the rotation poset of the underlying instance. By adding all the predecessors of the rotations in set V_I, we obtain a closed subset. The latter defines a stable matching.

$\Leftarrow$ Let M be a stable matching and let S be its closed subset. The set of sink rotations of S corresponds (by definition) to a set of incomparable rotations. □

We do not study $(1,0)$-supermatches because, in the case of the SM these do not exist as shown in Proposition 2. Therefore, the first interesting general case is when $a = 1$ and $b = 1$. In the rest of this chapter, we work on the $(1,1)$-supermatches.

Proposition 2 $(1,0)$*-supermatches does not exist.*

Proof. The proof is quite straightforward by Theorem 2, Page 41. Given a stable matching M, if a pair $(m_i, w_j) \in M$ is to break-up, the agents m_i and w_j must be matched to other people. This means that at least one other couple has to break-up to provide a repair to each break-up. □

3.4 (1,1)-supermatches

Finding a $(1,1)$-supermatch is a special case of the general case of finding (a,b)-supermatches, since it allows breaking-up of only one pair and guarantees a repair stable matching by breaking-up at most one other pair. In this section, we first discuss how to characterize a $(1,1)$-supermatch and then discuss a model for finding if there exists one in a given SM instance.

Given a stable matching M and its corresponding closed subset S, in order to decide if M is a $(1,1)$-supermatch, all possible breakages for each pair must be considered. We refer to each pair by the man involved in that pair, i.e. (m_i, w_j) is referred by m_i. The idea is that, the breakage of a man m_i can only be repaired at a cost of 1 if one of the following holds:

i) If there exists a sink rotation ρ of S, where $|\rho| = 2$ and ρ involves m_i;

ii) If there exists a neighbour rotation ρ of S, where $|\rho| = 2$ and ρ involves m_i.

Note that the sink rotations of a closed subset S can be removed from the S without requiring that any additional rotations are removed (the successors) to obtain another closed subset. Because, by definition, none of the successors of the sink rotation are in the S. Similarly, the neighbour rotations of an S can be added to S without requiring that any other rotations (the predecessors) are added to obtain another closed subset. Because, by definition, all predecessors of a neighbour rotation are already in the S. Recall that, given a set of rotations V, the function $X(V)$ returns the set of all men involved in the rotations in V. In addition, the function $R^*(V)$ returns all the rotations $\rho \in V$, where $|\rho| = 2$. Proposition 3 presents a characterization for the $(1,1)$-supermatches by using their corresponding closed subsets.

Proposition 3 *Let M be a stable matching and S be its closed subset. M is a $(1,1)$-supermatch if and only if every non-fixed men of the instance are included in a neighbour or sink rotation of S of size 2, i.e. $|X(R^*(\mathbf{L}(S) \cup \mathbf{N}(S)))| = n$, where n is the number of non-fixed men.*

Proof. $\Rightarrow$ Suppose for contradiction that $|X(R^*(\mathbf{L}(S) \cup \mathbf{N}(S)))| \neq n$. Take a (non-fixed) man m that is not in $X(R^*(\mathbf{L}(S) \cup \mathbf{N}(S)))$. In order to repair M when m breaks-up with his partner, one needs to remove, or add, a rotation of size at least 3 or a set of rotations where at least 2 other men are modified. Therefore, by contradiction, M cannot be a $(1,1)$-supermatch.

$\Leftarrow$ If $|X(R^*(\mathbf{L}(S) \cup \mathbf{N}(S)))| = n$, then there exists a repair for the breakage of every man by either adding or removing a single rotation of size 2 to/from S. Therefore, M corresponds to a $(1,1)$-supermatch. □

3.4.1 A Model Using Independent Sets

We describe a representation for the problem of finding $(1,1)$-supermatches to a given SM instance. Our representation is based on finding independent sets with additional constraints. Independent sets have been used previously to find stable matchings in the Stable Roommates problem (SR) [GI89]. For SR, Gusfield and Irving show that each maximal independent set on an undirected graph derived from its rotation poset is in one-to-one correspondence with the stable matchings of the underlying SR instance. However, for the Robust Stable Marriage problem, we show that the additional robustness constraints change the structure of the problem, and therefore the approach is different.

An *independent set* I in an undirected graph G is defined as a set of vertices such that no two vertices from the set I share an edge in G. Consider a Stable Marriage instance $\mathcal{I}$ and let $\Pi = (\mathcal{V}, E)$ be its rotation poset. We use $\Pi_u = (\mathcal{V}, E_u)$ to denote the undirected representation of the rotation poset Π with transitivity. It is important to observe that, every independent set defined in Π_u consists of incomparable rotations. Because Π_u contains transitive edges, meaning that if a rotation ρ' is preceding another rotation ρ, there exists an edge $(\rho', \rho) \in E_u$, and therefore the rotations ρ' and ρ together cannot be members of any independent sets of Π_u. Using Proposition 1 and the observation, it is correct to state that there exists a one-to-one correspondence between the independent sets of Π_u and the stable matchings of the underlying instance.

Given a set of nodes I in $\mathcal{V}$, let P_I be the set of vertices that corresponds to rotations that precede a rotation from I. We use $\Pi^I_{sub} = (\mathcal{V}^I_{sub}, E_{sub})$ to denote the subgraph induced by Π where the set of vertices is $\mathcal{V}^I_{sub} = \mathcal{V} \setminus (P_I \cup I)$. We also define a function $source(G)$ that takes a directed graph G as input and returns the vertices in G that do not have any predecessors. Theorem 4 gives a formulation of the problem of finding $(1,1)$-supermatches. It is immediate from the Propositions 1 and 3.

Theorem 4 *There exists a one-to-one correspondence between the $(1,1)$-supermatches and the set of the independent sets $\mathcal{S}$ that satisfies $I \in \mathcal{S}$ if and only if the rotations of size 2 in the independent set I together with the source rotations*

in Π^I_{sub} cover all the non-fixed men (i.e. $|X(R^*(I \cup source(\Pi^I_{sub})))| = n$), where n is the number of non-fixed men.

Proof. By construction. □

Example. For illustration, consider the case of the Stable Marriage instance of Table 2.1 (Page 38) and its rotation poset Π given in Table 2.11 (Page 44). The version of Π, where the transitive edges are included (denoted by Π_u) is illustrated in Figure 3.2.

A sample independent set on Π_u can be given as $I_i = \{\rho_2, \rho_4\}$ as they do not share any edge. The rotations of size 2 in this set are $R^*(I_i) = I_i$ as all the rotations in I_i are of size 2. The sub-graph $\Pi^{I_i}_{sub}$ contains only the vertices ρ_3 and ρ_5 and an empty set of edges. The source nodes of $\Pi^{I_i}_{sub}$ are therefore $\{\rho_3, \rho_5\}$ and they are both of size 2. The set of men included in these rotations all together $\{\rho_2, \rho_4, \rho_3, \rho_5\}$ corresponds to $X(R^*(I_i \cup source(\Pi^{I_i}_{sub}))) = \{0, 1, 2, 3, 4, 5, 6\}$. It can be verified that the size of this set is 7, i.e. the number of non-fixed men n in the problem. Hence, we say that there is a repair for the breakage of each man at a cost of 1. Therefore, according to Theorem 4, the independent set I_i corresponds to a $(1, 1)$-supermatch.

Then, the closed subset S_i that corresponds to I_i is obtained by starting from $S_i = I_i$, and adding all the predecessors P_{I_i} of the rotations in I_i to S_i. Considering that $P_I = \{\rho_0, \rho_1\}$, the corresponding closed subset S_i yields in $S_i = \{\rho_0, \rho_1, \rho_2, \rho_4\}$. The closed subset S_i, and the corresponding stable matching M_i can be verified from the Table 3.1 as M_6. It can be used as the proof for

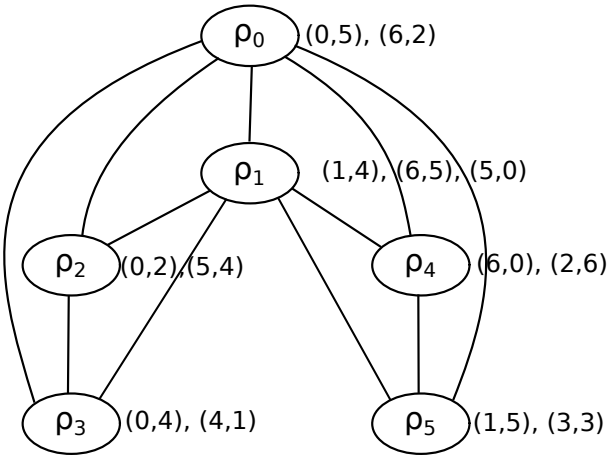


Figure 3.2: Undirected graph representation with transitive edges included of the rotation poset given in Figure 2.11.

the example left to the reader that M_6 is a $(1,1)$-supermatch.

A second illustration on the same graph is with the independent set $I_j = \{\rho_0\}$, corresponding to stable matching M_1 in Table 3.1. The rotation set $R^*(I_j) = I_j$, and $\Pi_{sub}^{I_j}$ consists of the rotations $\{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$. Since ρ_1 is the only source node of $\Pi_{sub}^{I_j}$ with size 3, we have $R^*(I_j \cup (source(\Pi_{sub}^{I_j})) = \{\rho_0\}$. Then, $X(R^*(I_j \cup (source(\Pi_{sub}^{I_j}))) = \{0, 6\}$ and the size of this set is 2. Due to $2 < n$, where $n = 7$, being true, the stable matching corresponding to the closed subset $\{\rho_0\}$ (i.e. M_1) is not a $(1,1)$-supermatch.

Although we provide a model for identifying the $(1,1)$-supermatches, we do not have a polynomial-time method to decide if a $(1,1)$-supermatch exists for a given instance using this model. This model can lead to an elegant proof for the complexity of RSM. However, in order to prove the complexity, we make use of a special SAT formulation as discussed in Section 3.5.

3.5 Complexity of Finding (1,1)-supermatches

This section corresponds to the most technical part of this dissertation. We provide formal definitions for the RSM and some of its sub-problems. Then, we define a special case of SAT, that is shown to be $\mathcal{NP}$-complete by Schaefer's Dichotomy Theorem. Then we use this formulation to prove that our problem, RSM, is $\mathcal{NP}$-hard. We categorize some sub-problems as $\mathcal{NP}$-complete, $\mathcal{NP}$-hard or polynomial-time solvable as well as identifying some open problems.

We start by defining the general case of the RSM in Definition 13.

Definition 13 (π_{ab}) *Decision problem for (a,b)-supermatch.*
INPUT*: $a, b \in \mathbb{N}$, and a Stable Marriage instance $\mathcal{I}$.*
QUESTION*: Is there an (a,b)-supermatch for $\mathcal{I}$?*

The main purpose of this section is to decide to which complexity class π_{ab} belongs. We work on the complexity of the π_{ab} by breaking it into its sub-cases such as the decision problem for $(1,b)$-supermatches, $(1,1)$-supermatches, etc. Next, we show the $\mathcal{NP}$-completeness of deciding if there exists a $(1,1)$-supermatch for a given Stable Marriage instance. In order to achieve this, in Section 3.5.1, we begin by defining a specific family of SM instances. The motivation for studying this family is that finding $(1,1)$-supermatches using the rotation posets of the underlying instances from this family has very specific con-

straints that can easily be modelled. Therefore, considering these constraints, we introduce in Section 3.5.2 a specific Boolean Satisfiability (SAT) problem with many specific rules capturing these constraints. This SAT problem is actually a SAT formulation of the problem of finding a $(1, 1)$-supermatch on this restricted family of SM instances. After defining the SAT formulation, in Section 3.5.3 we show using the Schaefer's Dichotomy Theorem (defined in Section 2.3.1) that this SAT formulation is $\mathcal{NP}$-complete. Subsequently, we state our complexity result for RSM by showing the equivalence between this special SAT formulation and the problem of deciding if there exists a $(1, 1)$-supermatch for the restricted family of instances. We also provide examples for the reader to understand the steps in the rest of the section better.

3.5.1 A Specific Problem Family F

We characterize a specific, restricted family F of Stable Marriage instances with four properties, where $\Pi_{\mathrm{F}} = (\mathcal{V}_{\mathrm{F}}, E_{\mathrm{F}})$ represents their generic rotation poset. Before introducing the properties, recall that, given a pair (m_i, w_j) in a rotation ρ, if ρ' is the unique rotation that moves m_i to w_j then there exists an edge (ρ', ρ) in the poset, and ρ' is called a Type 1 predecessor of ρ. We call the edge (ρ', ρ) as a Type 1 edge. The following are the four properties of family F:

Prop. (1) Each rotation $\rho_i \in \mathcal{V}_F$, contains *exactly* 2 pairs $\rho_i = ((m_{i1}, w_{i1}), (m_{i2}, w_{i2}))$.

Prop. (2) Each rotation $\rho_i \in \mathcal{V}_F$, has *at most* 2 immediate predecessors and at most 2 immediate successors.

Prop. (3) Each edge $e_i \in E_F$, is a *Type 1* edge.

Prop. (4) For each man $m_i, i \in [1, n]$, m_i is involved in *at least* 2 rotations.

Figure 3.3 illustrates some cases respecting these properties. In this figure, we denote each pair (m_i, w_j) by (i, j) to make it easier to read. The different cases in the figure emphasize Property (2), where a rotation has exactly: one predecessor and one successor (A,B); one predecessor and two successors, which is also similar to having one successor and two predecessors (C); two predecessors and two successors (D). It can be verified that Properties (1) and (3) are satisfied in all of the cases. However, Property (4) is not satisfied as this is not a complete example. Observe that, the ordering of the pairs is not important as there exist only two pairs in each rotation.

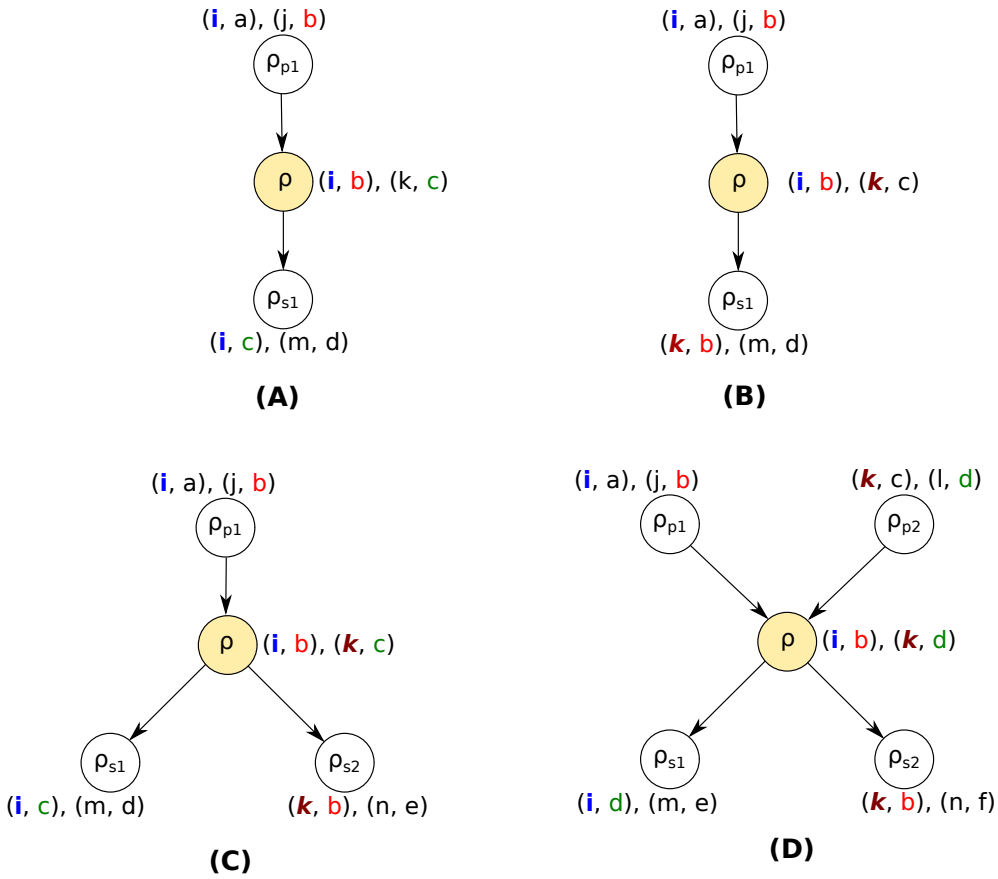


Figure 3.3: Illustrations of different cases for the men and women included in the rotations in Π of SM instances in F.

We would like to emphasize the difference between Case A and Case B in Figure 3.3. Due to Property (3), any two rotations that have an edge between them contain a man and a woman in common. When the case is generalized to three rotations $\rho_{p1} - \rho - \rho_{s1}$, it should be noted that those three rotations either contain the same man m_i but contain different women, w_b between ρ_{p1} and ρ, but w_c between ρ and ρ_{s1}, as in Case A; or the same woman w_b in all three rotations but different men m_i between ρ_{p1} and ρ, but m_k between ρ and ρ_{s1}, as in the Case B.

This family of instances is very restricted as it does not allow any of the rotations to contain more than 2 pairs. Additionally, due to the bound on the number of successors and predecessors of a rotation, the rotation posets of these instances have very specific structures. Later in Section 4.7.2, we identify a set of instances that is in F. The number of stable matchings for each instance in this set is exponential with respect to the size n.

Lemma 6 is a characterization for all the SM instances and can easily be observed on the family F.

Lemma 6 *For every two different paths P_1 and P_2 defined on the rotation poset Π_F of an instance of family* F*, where both start at rotation ρ_s, end at ρ_t, and the pair $(m_e, w_f) \in \rho_s$, if all rotations on P_1 (respectively P_2) contain m_e, at least one of the rotations on P_2 (respectively P_1) does not contain w_f.*

Proof. Suppose for contradiction that man m_e is involved in all rotations on P_1, and w_f is involved in all rotations on P_2. This scenario is likely to occur on F as Property (3) indicates that all edges are of Type 1 (i.e. a rotation and its immediate successor involves the same man, or woman), which is also easy to observe in Figure 3.3, where any two rotations connected by an edge in the rotation poset always contain a man and a woman in common. In this case, because of the supposition, namely that m_e and w_f are carried on to the rotation ρ_t, the pair (m_e, w_f) is reproduced. In other words, exposing ρ_t on a stable matching produces the pair (m_e, w_f). However, this pair is already eliminated by ρ_s, meaning this couple is already produced. The supposition contradicts the fact that exposing rotations on stable matchings causes men to be matched with their less preferred partners, and if a couple is eliminated once, it cannot be produced again. □

In Table 3.2 we provide a sample SM instance $\mathcal{I}$ from family F. The instance contains 6 men and 6 women. Figure 3.4 illustrates the rotation poset of this instance. It can be easily verified that Properties (1), (2), (3), and (4) apply to the rotation poset.

Note that in Section 3.5.2, we use this instance as a look-up example. We give a complete example in the sense that we first introduce a sample SM instance, then we show how to construct this instance by starting from a special SAT formulation, without knowing anything about the SM instance. Let us first define some sub-problems of $\boldsymbol{\pi_{ab}}$ below.

Table 3.2: An SM instance from family F of 6 men and 6 women.

m_1	1 2 6 3 4 5
m_2	2 1 4 3 5 6
m_3	3 4 1 2 5 6
m_4	4 3 5 1 2 6
m_5	5 3 2 1 4 6
m_6	6 2 3 1 4 5

w_1	3 2 1 4 5 6
w_2	5 6 1 2 3 4
w_3	6 5 4 3 1 2
w_4	2 3 4 1 5 6
w_5	4 5 1 2 3 6
w_6	1 6 2 3 4 5

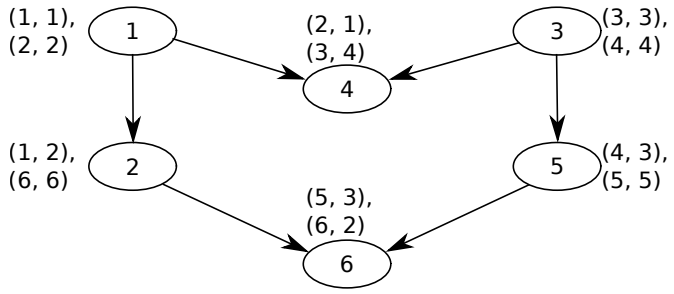


Figure 3.4: Final version of the rotation poset constructed from the sample in Table 3.3.

Definition 14 (π_{1b}) *A particular case of π_{ab}. Decision problem for $(1, b)$-supermatch.*
INPUT*: $b \in \mathbb{N}$, and a Stable Marriage instance $\mathcal{I}$.*
QUESTION*: Is there a $(1, b)$-supermatch for $\mathcal{I}$?*

Definition 15 (π_{11}) *A particular case of π_{1b}. Decision problem for $(1, 1)$-supermatch.*
INPUT*: A Stable Marriage instance $\mathcal{I}$.*
QUESTION*: Is there a $(1, 1)$-supermatch for $\mathcal{I}$?*

Definition 16 (π_{11}^{F}) *A particular case of π_{11}. Decision problem for $(1, 1)$-supermatch for problem family* F.
INPUT*: A Stable Marriage instance $\mathcal{I} \in$* F.
QUESTION*: Is there a $(1, 1)$-supermatch for $\mathcal{I}$?*

Our aim is to first prove the complexity of π_{1b} as it is a more specific case of π_{ab}. We construct the proof for showing the problem π_{1b} is $\mathcal{NP}$-complete by first showing that its restricted version π_{11} is $\mathcal{NP}$-complete. In order to do this, we work on the specific family of Stable Marriage instances F, proving $\mathcal{NP}$-completeness for π_{11}^{F}. Then we generalize the results.

3.5.2 The Definition of SAT-SM

In this section, we define a special, restricted case of SAT that takes additional inputs. As motivated before, this SAT formulation is inspired by the specific family F defined earlier in Section 3.5.1. Please note that this definition is long as we include examples in the definition to make it more understandable. The formulation requires some rules and conditions to be applied on the input of the problem. The intuition behind imposing these rules and conditions is to reflect the specified properties (Prop. (1), (2), (3), and (4)) of the family F. This

special SAT formulation is actually equivalent to finding $(1, 1)$-supermatches to any instance in F, which we prove later in Section 3.5.3. However, for the moment, we would like to give some insight to the reader about why we need all the conditions and the rules in the definition of this formulation to make the definition easier to follow.

All the Boolean and integer variables in the input of the problem are inspired by the rotations. First, we give an abstraction of the use of these variables. The set of integers corresponds to the set of rotations in the rotation poset of a Stable Marriage instance $\mathcal{I}$. Assume that a matching M (S is M's corresponding closed subset) is a $(1, 1)$-supermatch to $\mathcal{I}$. The Boolean variables then correspond to the status of the rotations with respect to S. More specifically, for each rotation identified by a unique integer e, each Boolean variable y_e represents if the corresponding rotation e is a sink rotation in S, each s_e represents if e is included in S, p_e represents if all parents of e are in S but $e \notin S$ (i.e. a neighbour rotation).

Second, the conditions on the lists reflect the properties of F. **Rule 1** captures the property of the problem defined in Lemma 6. Recall that, our motivation is to show that an SM instance can be constructed from this SAT formulation. Considering this, if **Rule 1** is not satisfied, then previously broken pairs might be re-matched in the constructed SM instance, which is not allowed in the SM [GI89].

Finally, the conditions on the clauses model the problem $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$. In what is presented below, the purpose of the clauses in Ⓐ ensure each man is included either in a sink rotation in S or all of its parents are in S. Clauses in Ⓑ ensure the result corresponds to a closed subset. Clauses in Ⓒ define how a rotation can be identified as a sink rotation. Additionally, clauses in Ⓓ define how a rotation can be identified as having its parents in S but not itself.

The reason for including all these clauses and rules should become more clear to the reader when we are constructing a Stable Marriage instance from an instance of SAT-SM later in the proof of Theorem 6.

We now begin our definition for the special case of SAT, namely SAT-SM.

Input: SAT-SM takes as input the following:

- A set of integers: $\boldsymbol{\chi} = [1, |\boldsymbol{\chi}|]$,
- A set of n lists: $l_1, l_2, \ldots, l_n$, where each list is an ordered list of integers

of $\boldsymbol{\chi}$,

- A set of disjoint Boolean variables: $Y = \{y_e \mid e \in \boldsymbol{\chi}\}$,
- Another set of disjoint Boolean variables: $S = \{s_e \mid e \in \boldsymbol{\chi}\}$,
- Another set of disjoint Boolean variables: $P = \{p_e \mid e \in \boldsymbol{\chi}\}\}$.

Conditions on the Lists: The lists $l_1, \ldots, l_n$ are subject to the following constraints.

- Each list $\forall a \in [1, n]$, $l_a = (\chi^a_1, \ldots, \chi^a_{k_{l_a}})$, where $k_{l_a} = |l_a| \geq 2$.
- Each element of $\boldsymbol{\chi}$ appears in exactly two different lists.

Example. For illustration, the set $\boldsymbol{\chi}$ represents the indexes of rotations and a list l_a represents the index of each rotation containing the man m_a. The order in l_a specifies the path in the rotation poset from the first rotation to the last one for a man m_a. The requirement for having each index in two different lists is related to Property (1) of F. In addition to those two conditions, we have the following rule over the lists:

[Rule 1] For any χ^m_i and χ^m_j from the same list l_m where $m \in [1, n]$ and $j > i$, there does not exist any sequence S that starts at χ^m_i and ends at χ^m_j constructed by iterating the two consecutive rules below:

α) given $\chi^a_e \in S$, the next element in S is χ^a_{e+1}, where $e + 1 \leq k_{l_a}$.

β) given $\chi^a_e \in S$, the next element in S is χ^b_f, where $\chi^a_e = \chi^b_f, a \neq b \in [1, n]$, and $1 \leq f \leq k_{l_b}$.

Example. Before continuing to the conditions on the clauses, we would like to present an example where **Rule 1** is not satisfied. Consider three lists defined over $\boldsymbol{\chi} = \{1, 2, 3\}$ as: $l_1 = (1, 3), l_2 = (1, 2), l_3 = (2, 3)$. We use the properties α and β to construct a sequence that is not allowed by the rule. Let the pair $\langle i, l_k \rangle$ denote the element i in list l_k, and we use $\rightarrow_p$ to show the property p that is applied to move from one pair to the other one in the sequence. For elements 1 and 3 in from the same list l_1, consider the sequence $S = \langle 1, l_1 \rangle \rightarrow_\beta \langle 1, l_2 \rangle \rightarrow_\alpha \langle 2, l_2 \rangle \rightarrow_\beta \langle 2, l_3 \rangle \rightarrow_\alpha \langle 3, l_3 \rangle \rightarrow_\beta \langle 3, l_1 \rangle$. This sequence starts and ends at the same list l_1, where 1 appears before 3 in l_1. The existence of this sequence causes the given three lists to be excluded from the SAT-SM.

Conditions on the clauses. The CNF that defines SAT-SM is a conjunction of four groups of clauses: Ⓐ, Ⓑ, Ⓒ and Ⓓ. The groups are subject to the following

conditions:

Ⓐ: For any list $l_a = (\chi^a_1, \ldots, \chi^a_{k_{l_a}})$, we have a disjunction between the Y-elements and the P-elements as $\bigvee_{i=1}^{k_{l_a}}(y_{\chi^a_i} \vee p_{\chi^a_i})$.

$$\text{Ⓐ is defined by } \bigwedge_{a=1}^{n}\left(\bigvee_{i=1}^{k_{l_a}}(y_{\chi^a_i} \vee p_{\chi^a_i})\right). \tag{3.1}$$

Ⓑ: For any list $l_a = (\chi^a_1, \ldots, \chi^a_{k_{l_a}})$, we have a conjunction of disjunctions between two S-elements with consecutive indexes as $\bigwedge_{i=1}^{k_{l_a}-1}(s_{\chi^a_i} \vee \neg s_{\chi^a_{i+1}})$.

$$\text{Ⓑ is defined by } \bigwedge_{a=1}^{n}\bigwedge_{i=1}^{k_{l_a}-1}(s_{\chi^a_i} \vee \neg s_{\chi^a_{i+1}}). \tag{3.2}$$

Ⓒ: This group of clauses is split in two. For any list $l_a = (\chi^a_1, \ldots, \chi^a_{k_{l_a}})$, the first sub-group C_1 contains all the clauses defined by the logic formula $\bigwedge_{i=1}^{k_{l_a}-1} y_{\chi^a_i} \rightarrow (s_{\chi^a_i} \wedge \neg s_{\chi^a_{i+1}})$. In CNF notation it leads to $\bigwedge_{i=1}^{k_{l_a}-1}(\neg y_{\chi^a_i} \vee s_{\chi^a_i}) \wedge (\neg y_{\chi^a_i} \vee \neg s_{\chi^a_{i+1}})$. Note that, C_1 also covers the special case, when $i = k_{l_a}$.

$$C_1 \text{ is defined by } \bigwedge_{a=1}^{n}\left(\bigwedge_{i=1}^{k_{l_a}}(\neg y_{\chi^a_i} \vee s_{\chi^a_i}) \wedge \bigwedge_{i=1}^{k_{l_a}-1}(\neg y_{\chi^a_i} \vee \neg s_{\chi^a_{i+1}})\right). \tag{3.3}$$

The second sub-group C_2 has three specific cases according to the position of elements in the ordered lists. As fixed above, each element of $\boldsymbol{\chi}$ appears in exactly two different lists. Thus, for any $e \in \boldsymbol{\chi}$, there exists two lists l_a and l_b such that $\chi^a_i = \chi^b_j = e$, where $i \in [1, k_{l_a}]$ and $j \in [1, k_{l_b}]$. For each couple of elements of $\boldsymbol{\chi}$ denoted by (χ^a_i, χ^b_j) that are equal to the same value e, we define a clause with these elements and the next elements in their lists respecting the ordering: $s_{\chi^a_i} \rightarrow (y_{\chi^a_i} \vee s_{\chi^a_{i+1}} \vee s_{\chi^b_{j+1}})$. With a CNF notation it leads to: $(\neg s_{\chi^a_i} \vee y_{\chi^a_i} \vee s_{\chi^a_{i+1}} \vee s_{\chi^b_{j+1}})$.

We add the two specific cases where χ^a_i or χ^b_j, or both are the last elements of their ordered lists. The complete formula for the set of clauses C_2 for each two

element (χ_i^a, χ_j^b) $s.t.$ $\chi_i^a = \chi_j^b$ is:

$$C_2 \quad \left[\begin{array}{ll} \bigwedge\limits_{i \neq k_{l_a},\ j \neq k_{l_b}} & \neg \boldsymbol{s}_{\chi_i^a} \vee \boldsymbol{y}_{\chi_i^a} \vee \boldsymbol{s}_{\chi_{i+1}^a} \vee \boldsymbol{s}_{\chi_{j+1}^b} \\ \bigwedge\limits_{i \neq k_{l_a},\ j = k_{l_b}} & \neg \boldsymbol{s}_{\chi_i^a} \vee \boldsymbol{y}_{\chi_i^a} \vee \boldsymbol{s}_{\chi_{i+1}^a} \\ \bigwedge\limits_{i = k_{l_a},\ j = k_{l_b}} & \neg \boldsymbol{s}_{\chi_{k_{l_a}}^a} \vee \boldsymbol{y}_{\chi_{k_{l_a}}^a}. \end{array} \right. \tag{3.4}$$

Ⓓ: Similar to C_2, for each couple of elements of $\boldsymbol{\chi}$ denoted by (χ_i^a, χ_j^b) equal to the same value e, there exists a clause with these elements and the previous elements in their lists respecting the ordering: $p_{\chi_i^a} \leftrightarrow (\neg s_{\chi_i^a} \wedge s_{\chi_{i-1}^a} \wedge s_{\chi_{j-1}^b})$. In CNF notation, it leads to: $(\neg p_{\chi_i^a} \vee \neg s_{\chi_i^a}) \wedge (\neg p_{\chi_i^a} \vee s_{\chi_{i-1}^a}) \wedge (\neg p_{\chi_i^a} \vee s_{\chi_{j-1}^b}) \wedge (s_{\chi_i^a} \vee \neg s_{\chi_{i-1}^a} \vee \neg s_{\chi_{j-1}^b} \vee p_{\chi_i^a})$.

By generalizing the formula for any couple, and by adding the two cases where $\chi_i^{l_a}$, or $\chi_j^{l_b}$, or both are the first elements of their respective lists, the complete formula Ⓓ for each two element (χ_i^a, χ_j^b) $s.t.$ $\chi_i^a = \chi_j^b = e$ is described by:

$$\text{Ⓓ} \quad \left[\begin{array}{ll} \bigwedge\limits_{i \neq 1,\ j \neq 1} & (\neg \boldsymbol{p}_{\chi_i^a} \vee \boldsymbol{s}_{\chi_{i-1}^a}) \wedge (\neg \boldsymbol{p}_{\chi_j^b} \vee \boldsymbol{s}_{\chi_{j-1}^b}) \wedge \\ & \quad (\boldsymbol{s}_{\chi_i^a} \vee \neg \boldsymbol{s}_{\chi_{i-1}^a} \vee \neg \boldsymbol{s}_{\chi_{j-1}^b} \vee \boldsymbol{p}_{\chi_i^a}) \\ \bigwedge\limits_{i = 1,\ j \neq 1} & (\neg \boldsymbol{p}_{\chi_j^b} \vee \boldsymbol{s}_{\chi_{j-1}^b}) \wedge (\boldsymbol{s}_{\chi_i^a} \vee \neg \boldsymbol{s}_{\chi_{j-1}^a} \vee \boldsymbol{p}_{\chi_i^a}) \\ \bigwedge\limits_{i = 1,\ j = 1} & (\boldsymbol{s}_{\chi_1^a} \vee \boldsymbol{p}_{\chi_1^a}) \\ \wedge & (\neg \boldsymbol{p}_e \vee \neg \boldsymbol{s}_e). \end{array} \right. \tag{3.5}$$

To conclude the definition:

***The full CNF formula of* SAT-SM:** Ⓐ $\wedge$ Ⓑ $\wedge$ C_1 $\wedge$ C_2 $\wedge$ Ⓓ.

We introduce a sample SAT-SM instance as an example to demonstrate the methods described in the rest of this section. Table 3.3 introduces a SAT-SM instance consisting of 6 lists and a set of 6 integers $\boldsymbol{\chi} = \{1, 2, 3, 4, 5, 6\}$. Note that, these lists are constructed using the set $\boldsymbol{\chi}$ by following the *conditions on the lists* described on Page 76 defined for SAT-SM and **Rule 1**.

Each list can be seen as corresponding to a man involved in the constructed

Table 3.3: An instance of SAT-SM of 6 lists and 6 integers.

List	Elements
l_1	(1, 2)
l_2	(1, 4)
l_3	(3, 4)
l_4	(3, 5)
l_5	(5, 6)
l_6	(2, 6)

Table 3.4: Clauses of the SAT-SM instance given in Table 3.3.

list	Ⓐ	Ⓑ	C_1
l_1	$(y_1 \vee y_2 \vee p_1 \vee p_2)$	$(s_1 \vee \neg s_2)$	$(\neg y_1 \vee s_1) \wedge (\neg y_2 \vee s_2) \wedge (\neg y_1 \vee \neg s_2)$
l_2	$(y_1 \vee y_4 \vee p_1 \vee p_4)$	$(s_1 \vee \neg s_4)$	$(\neg y_1 \vee s_1) \wedge (\neg y_4 \vee s_4) \wedge (\neg y_1 \vee \neg s_4)$
l_3	$(y_3 \vee y_4 \vee p_3 \vee p_{4)}$	$(s_3 \vee \neg s_4)$	$(\neg y_3 \vee s_3) \wedge (\neg y_4 \vee s_4) \wedge (\neg y_3 \vee \neg s_4)$
l_4	$(y_3 \vee y_5 \vee p_3 \vee p_5)$	$(s_3 \vee \neg s_5)$	$(\neg y_3 \vee s_3) \wedge (\neg y_5 \vee s_5) \wedge (\neg y_3 \vee \neg s_5)$
l_5	$(y_5 \vee y_6 \vee p_5 \vee p_6)$	$(s_5 \vee \neg s_6)$	$(\neg y_5 \vee s_5) \wedge (\neg y_6 \vee s_6) \wedge (\neg y_5 \vee \neg s_6)$
l_6	$(y_2 \vee y_6 \vee p_2 \vee p_6)$	$(s_2 \vee \neg s_6)$	$(\neg y_2 \vee s_2) \wedge (\neg y_6 \vee s_6) \wedge (\neg y_2 \vee \neg s_6)$
χ	C_2	Ⓓ	
1	$(\neg s_1 \vee y_1 \vee s_2 \vee s_4)$	$(\neg p_1 \vee \neg s_1) \wedge (s_1 \vee p_1)$	
2	$(\neg s_2 \vee y_2 \vee s_6)$	$(\neg p_2 \vee \neg s_2) \wedge (\neg p_2 \vee s_1) \wedge (s_2 \vee \neg s_1 \vee p_2)$	
3	$(\neg s_3 \vee y_3 \vee s_4 \vee s_5)$	$(\neg p_3 \vee \neg s_3) \wedge (s_3 \vee p_3)$	
4	$(\neg s_4 \vee y_4)$	$(\neg p_4 \vee \neg s_4) \wedge (\neg p_4 \vee s_1) \wedge (\neg p_4 \vee s_3) \wedge$ $(s_4 \vee \neg s_1 \vee \neg s_3 \vee s_4)$	
5	$(\neg s_5 \vee y_5 \vee s_6)$	$(\neg p_5 \vee \neg s_5) \wedge (\neg p_5 \vee s_3) \wedge (s_5 \vee \neg s_3 \vee p_5)$	
6	$(\neg s_6 \vee y_6)$	$(\neg p_6 \vee \neg s_6) \wedge (\neg p_6 \vee s_5) \wedge (\neg p_6 \vee s_2) \wedge$ $(s_6 \vee \neg s_5 \vee \neg s_2 \vee p_6)$	

Stable Marriage instance $\mathcal{I}$. For instance, l_1 corresponds to m_1, l_2 corresponds to m_2, etc. Then, each element can be seen as a rotation in the rotation poset of $\mathcal{I}$, i.e. 1 corresponds to ρ_1, 2 corresponds to ρ_2, and so on. Then, this table describes which rotations involve which men. For example, m_1 is included in rotations ρ_1, ρ_2, and none of the other rotations contain m_1. Recall the rotation poset of the SM instance given in Figure 3.4. Observe that the lists capture the men in the rotations in this poset.

Then, using the lists given in Table 3.3, clauses of the model can easily be derived by following the *conditions on the clauses* described on Page 76. The complete list of clauses are detailed in Table 3.4. Note that, although it is not specified on the table for making it easy to read, the SAT-SM instance is a conjunction of all the clauses in each group (i.e. Ⓐ $\wedge$ Ⓑ $\wedge$ C_1 $\wedge$ C_2 $\wedge$ Ⓓ). Also note that, this example instance is one of the smallest that can be created by respecting the rules of the SAT-SM.

3.5.3 The Complexity of SAT-SM

We show some properties of the SAT-SM in this section to be used in our theorem.

Lemma 7 *There always exist clauses of minimum length 4 that are defined over positive literals in* Ⓐ.

Proof. For any list of ordered elements $l_a \in \{l_1, l_2, \ldots, l_n\}$, the length of each list is defined as $k_{l_a} \geq 2$ in SAT-SM, which results in Ⓐ having n clauses that have at least 4 positive literals in each. □

Lemma 8 *There always exist clauses of length 2 that are defined over two negative literals in* Ⓒ.

Proof. The clauses in Ⓒ consists of C_1 and C_2. The clauses in C_1 are defined over the list of ordered elements. For any two consecutive elements in a list $l_a \in \{l_1, l_2, \ldots, l_n\}$, there exists one clause in C_1: $\bigwedge_{i=1}^{k_{l_a}-1}(\neg y_{\chi_i^a} \vee \neg s_{\chi_{i+1}^a})$. By definition, the minimum length of an ordered list l_a is $k_{l_a} = 2$ and therefore the formula contains at least n clauses defined over two negative literals. □

Lemma 9 *Any clause defined over only positive literals of size at least two is not affine.*

Proof. Any clause C of the given form with k positive literals has $2^k - 1$ valid assignments. The cardinality of an affine relation is always a power of 2 [Sch78]. Thus, C is not affine. □

The SAT-SM problem is the question of finding an assignment of the Boolean variables that satisfies the above CNF formula.

Theorem 5 *The* SAT-SM *problem is $\mathcal{NP}$-complete.*

Proof. We use Schaefer's Dichotomy Theorem (see Theorem 1, Page 18) to prove that SAT-SM is $\mathcal{NP}$-complete [Sch78]. Schaefer identifies six cases, where if any one of them is valid the SAT problem is solved in polynomial time. Any SAT formula that does not satisfy any of those six is $\mathcal{NP}$-complete.

It is easy to see the properties *b*, *c*, and *f* in Schaefer's Dichotomy Theorem do not apply to SAT-SM due to Lemma 7. Similarly, properties *a* and *d* are not satisfiable because of Lemma 8. The clauses in Ⓐ are defined as clauses over positive literals and it is known that they always exist by Lemma 7. By applying Lemma 9 on the clauses in Ⓐ, we infer that property *e* is not applicable either.

Hence, SAT-SM is $\mathcal{NP}$-complete. □

Having proved that SAT-SM is $\mathcal{NP}$-complete, we now present in Theorem 6 the main result of this section, which is the $\mathcal{NP}$-completeness of $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$.

Theorem 6 *The decision problem $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$ is $\mathcal{NP}$-complete.*

Proof. The witness is known to be polynomial-time decidable [GSOS17a]. Therefore, $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$ is in $\mathcal{NP}$. We discuss this polynomial-time witness in Section 4.1 in detail. We show that $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$ is $\mathcal{NP}$-complete by presenting a polynomial reduction from the SAT-SM problem to $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$. Note that this proof is long. We first discuss how to construct an SM instance from the given SAT-SM instance, and then conclude the proof by showing the equivalency between SAT-SM and $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$. Therefore, let us divide this section into sub-sections for the construction of the SM instance, and showing the equivalency.

Construction of the rotation poset Π_{F}. From an instance $\mathcal{I}_{SSM}$ of SAT-SM, we construct in polynomial-time an instance $\mathcal{I}$ of $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$. This means the construction of the rotation poset $\Pi_{\mathrm{F}} = (\mathcal{V}_{\mathrm{F}}, E_{\mathrm{F}})$ with all stable pairs in the rotations, and the preference lists. We begin by constructing the set of rotations $\mathcal{V}_{\mathrm{F}}$ and then proceed by deciding which man is a part of which stable pair in which rotation. First, for each integer in the input, i.e. $\forall e \in \boldsymbol{\chi}$, we have a corresponding rotation ρ_e. Initially, each rotation contains two "empty" pairs.

Second, for each integer i appears in a list l_a, we add the i as the man in the first empty pair to rotation ρ_a. More formally, $\forall l_a, a \in [1, n], \forall \chi^a_i \in [1, k_{l_a}]$, we insert m_a as the man to the first empty pair in rotation $\rho_{\chi^a_i}$. Since $k_{l_a} \geq 2$ from Lemma 7, each man of $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$ is involved in at least two rotations (satisfying Property (4) of the family F).

Illustration. For the SAT-SM formula presented in Table 3.3, the rotations are constructed as: $\rho_1 = ((1,-),(2,-))$, $\rho_2 = ((1,-),(6,-))$, $\rho_3 = ((3,-),(4,-))$, $\rho_4 = ((2,-),(3,-))$, $\rho_5 = ((4,-),(5,-))$, and $\rho_6 = ((5,-),(6,-))$. Note that, we use $-$ instead of the women in the pairs as we do not know which woman is involved in which rotation, yet.

As each χ^a_i appears in exactly two different lists l_a and l_b, each rotation is guaranteed to contain exactly two pairs involving different men m_a, m_b (satisfying Property (1) of F), and to possess at most two immediate predecessors and two immediate successors in Π_{F} (satisfying Property (2) of F).

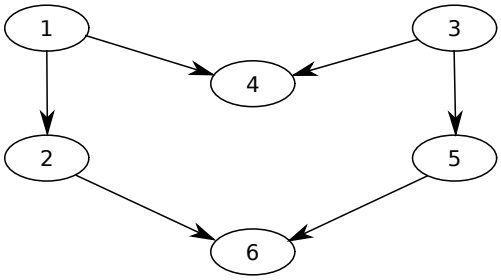


Figure 3.5: Initial version of the rotation poset constructed from the sample in Table 3.3.

For the construction of the set of edges E_{F}, for each couple of elements of $\boldsymbol{\chi}$ denoted by (χ_i^a, χ_{i+1}^a), $a \in [1,n], \forall i \in [1, k_{l_a} - 1]$, we add an edge from $\rho_{\chi_i^a}$ to $\rho_{\chi_{i+1}^a}$. The shape of the rotation poset of the example SAT-SM given in Table 3.3 after the vertices and edges are created is shown in Figure 3.5. Recall that, the labelling of the vertices (or rotations) come from the integers in $\boldsymbol{\chi}$.

It is important to observe that this construction yields each edge in E_{F} representing a Type 1 relationship (satisfying Property (3) and (4) of F). This can easily be seen as each edge links two rotations, where exactly one of the men is involved in both rotations. Now, in order to complete the rotation poset Π_{F}, the women involved in rotations must also be added. The following procedure is used to complete the rotations in the rotation poset:

1. For each element $\chi_1^a \in \boldsymbol{\chi}$, with $a \in [1,n]$, let $\rho_{\chi_1^a}$ be the rotation that involves man m_a. In this case, the partner of m_a in $\rho_{\chi_1^a}$ is completed by inserting woman w_a, so that the resulting rotation contains the stable pair $(m_a, w_a) \in \rho_{\chi_1^a}$.

2. We perform a breadth-first search on the rotation poset from the completed rotations. For each complete rotation $\rho = ((m_i, w_b), (m_k, w_d)) \in \mathcal{V}_{\mathrm{F}}$, if the immediate successor of ρ involves m_i (resp. m_k), let ρ_{s1} (resp. ρ_{s2}) be the immediate successor of ρ that modifies m_i (resp. m_k). If ρ_{s1} exists, then we insert the woman w_d in ρ_{s1} as the partner of man m_i. In the same manner, if ρ_{s2} exists, we insert the woman w_b in ρ_{s2} as the partner of man m_k. The procedure creates at most two stable pairs (m_i, w_d) and (m_k, w_b). We can observe that the completed rotations follow the same behaviour as the rotation poset illustrated in Figure 3.3 (D), Page 72.

Remark. We now show that none of the constructed paths in the rotation poset cause a pair that was eliminated before by a rotation to be re-matched. **Rule 1** imposed on the SAT-SM ensures that there does not exist more than one path

between any two rotations. Therefore, by imposing **Rule 1**, we can conclude that Lemma 6 is satisfied and none of the pairs that is eliminated by a preceding rotation is re-matched.

Throughout the construction we showed that all the properties required to have a valid rotation poset from the family F are satisfied. Using this process we are adding an equal number of women and men into the rotation poset. We can observe that after adding all the women in Figure 3.5 (Page 82) by using the procedure defined above, we end up with the rotation poset illustrated at the very beginning of this section in Figure 3.4 (Page 74).

Construction of the SM instance $\mathcal{I}$ from $\Pi_{\mathbf{F}}$. Having constructed the rotation poset, the last step to obtain an instance $\mathcal{I}$ of $\boldsymbol{\pi}^{\mathrm{F}}_{\mathbf{11}}$ by constructing the preference lists. Recall that $\Pi_{\mathrm{F}} = (\mathcal{V}_{\mathrm{F}}, E_{\mathrm{F}})$ is the rotation poset that we are building. By using the rotation poset Π_{F} created above, we can construct incomplete preference lists for the men and women. Gusfield et al. have previously defined a procedure to show that every finite poset corresponds to a Stable Marriage instance [GILS87]. They describe a method to create the preference lists using the rotation poset. We use a similar approach for creating the lists as detailed below:

- Apply topological sort on $\mathcal{V}_{\mathrm{F}}$.
- For each man $m_i \in [1, n]$, insert woman w_i as the most preferred to m_i's preference list.
- For each woman $w_i \in [1, n]$, insert man m_i as the least preferred to w_i's preference list.
- For each rotation $\rho \in \mathcal{V}_{\mathrm{F}}$ in the ordered set, for each pair (m_i, w_j) produced by ρ, insert w_j to the man m_i's list in decreasing order of preference ranking. Similarly, place m_i to w_j's list in increasing order of preference ranking.

Lemma 6 on our rotation poset clearly imposes that each preference list contains each member of the opposite sex at most once. To finish, one can observe that the instance obtained $\mathcal{I}$ respects the Stable Marriage requirements, and the specific properties from problem family F.

Illustration. Applying the procedure above, the preference lists of the men and women are found as shown in Table 3.5. The integers in the preference list of

Table 3.5: The incomplete preference lists derived from the rotation poset in Figure 3.4.

m_1	1 2 6
m_2	2 1 4
m_3	3 4 1
m_4	4 3 5
m_5	5 3 2
m_6	6 2 3

w_1	3 2 1
w_2	5 6 1 2
w_3	6 5 4 3
w_4	2 3 4
w_5	4 5
w_6	1 6

a man m_i denote the index of women. For instance, man m_1 prefers w_1 over w_2 and his least preferred partner is w_6. It is also similar for the preference lists of the women. Observe that, the lists obtained after the procedure correspond to the preference lists provided in Table 3.2, where the non-stable pairs are removed.

Equivalency between SAT-SM and $\boldsymbol{\pi_{11}^F}$. Having seen the construction of an SM instance $\mathcal{I}$ of $\boldsymbol{\pi_{11}^F}$ from a given instance $\mathcal{I}_{SSM}$ of SAT-SM we now present the equivalence between the two decision problems: $\boldsymbol{\pi_{11}^F}$ and SAT-SM.

$\Rightarrow$ Suppose that there exists a solution to an instance $\mathcal{I}$ of the decision problem $\boldsymbol{\pi}_{\mathbf{11}}^{\mathrm{F}}$. Then we have a $(1,1)$-supermatch and its corresponding closed subset S. As defined at the beginning of this chapter, **L(**S**)** is the set of sink vertices of the graph induced by the rotations in S, **N(**S**)** the set of vertices such that all their predecessors (if any) are in S but not themselves. From these two sets, we can assign all the literals in $\mathcal{I}_{SSM}$ as follows:

- For each rotation $\rho_i \in$ **L(**S**)**, set $y_i = true$. Otherwise, set $y_i = false$.
- For each rotation $\rho_i \in S$, set $s_i = true$. Otherwise, set $s_i = false$.
- For each rotation $\rho_i \in$ **N(**S**)**, set $p_i = true$. Otherwise, set $p_i = false$.

If S represents a $(1,1)$-supermatch, that means by removing only one rotation present in **L(**S**)** or by only adding one rotation from **N(**S**)**, any pair of the corresponding stable matching can be repaired with no additional modifications. Thus every men must be contained in a sink or a neighbour vertex. This leads to having for each man one of the literals assigned to true in his list in SAT-SM. Therefore every clause in Ⓐ in Equation 3.1 are satisfied.

For the clauses in Ⓑ in Equation 3.2, for any man's list the clauses are forcing each s_i literal to be true if the next one s_{i+1} is. From the definition of a closed

subset, from any sink vertex of S, all the preceding rotations (integers in the lists) must be in S. And thus every clause in Ⓑ is satisfied.

As the clauses in Ⓒ altogether capture the definition of being a sink vertex of the graph induced by the rotations in S, they are all satisfied by **L(**S**)**. At last, for the clauses in Ⓓ, it is also easy to see that any rotation being in **N(**S**)** is equivalent to not being in the solution and having predecessors in. Thus all the clauses are satisfied. Thus we can conclude that this assignment satisfies the SAT formula of $\mathcal{I}_{SSM}$.

$\Leftarrow$ Suppose that there exists a solution to an instance $\mathcal{I}_{SSM}$ of the decision problem SAT-SM. Thus we have a valid assignment to satisfy the SAT formula of $\mathcal{I}_{SSM}$. We construct a closed subset S to solve $\mathcal{I}$. As previously, we use the sets **L(**S**)** and **N(**S**)**, then for each literal y_i assigned to true, we put the rotation ρ_i in **L(**S**)**. We do the same for p_i and s_i as above.

The clauses in Ⓑ enforce the belonging to S of all rotations preceding any element of S, thus the elements in S form a closed subset. To obtain a $(1,1)$-supermatch, we have to be sure we can repair any couple by removing only one rotation present in **L(**S**)** or by only adding one rotation from **N(**S**)**. The clauses in Ⓒ enforce the rotations in **L(**S**)** to be without successors in S. And in the same way the clauses in Ⓓ enforce the rotations in **N(**S**)** to not be in S but have their predecessors in the solution.

Now we just have to check that all the men are contained in at least one rotation from **L(**S**)** $\cup$ **N(**S**)**. From the clauses from Ⓐ, we know that at least one y_e or p_e for any man m_i is assigned to true. Thus from this closed subset S, we can repair any couple (m_i, w_j) using one modification by removing/adding the rotation having m_i. Since there exists a $1-1$ equivalence between a stable matching and the closed subset in the rotation poset, we have a $(1,1)$-supermatch.

Illustration. In terms of example, a satisfying assignment for the look-up example SAT-SM instance $\mathcal{I}_{SSM}$ in Table 3.3 is presented in Table 3.6. One should read Table 3.6 as: $S = \{\rho_1, \rho_2, \rho_3\}$. It can be observed in Figure 3.4 that ρ_2 and ρ_3 are the sink rotations of S. Additionally, ρ_4 and ρ_5 are not in S, but all their predecessors are in, i.e. they are neighbour rotations. The S corresponds to one

Table 3.6: Solution transformation from $\mathcal{I}_{SSM}$ to $\mathcal{I}$.

true	**false**
$s_1, s_2, s_3, y_2, y_3, p_4, p_5$	$s_4, s_5, s_6, y_1, y_4, y_5, y_6, p_1, p_2, p_3, p_6$

of the $(1,1)$-supermatches to the SM instance $\mathcal{I}$ constructed from $\mathcal{I}_{SSM}$. □

Corollary 2 *Both decision problems $\boldsymbol{\pi_{11}}$ and $\boldsymbol{\pi_{1b}}$ are $\mathcal{NP}$-complete.*

Note that, we cannot generalize our results to the general case $\boldsymbol{\pi_{ab}}$. Because there is no known polynomial-time witness for (a,b)-supermatches for $a > 1$.

3.6 Threshold and Polynomial Cases

We present a family of instances for which a $(1,1)$-supermatch can be found in polynomial time. Then, we show that $(2,0)$-supermatches do not exist. Last, we discuss the existence of $(a,0)$-supermatches.

3.6.1 Polynomial Cases

In order to show a polynomial-time solvable SM family for $(1,1)$-supermatches, we first introduce a labelling for rotations in the rotation poset. We assume that the label of each source vertex of the rotation poset is 1, and for every other rotation, the rotations label is calculated by the number of edges to its furthest predecessor summed by the predecessor's label. Note that the furthest predecessor is always a source vertex due to having a poset, and therefore not having any cycles.

We often encounter with these instances when trying to create SM instances that have many stable matchings. Thus, these instances are interesting in the sense that some "large" SM instances often contain a $(1,1)$-supermatch and finding one is easy despite the size of the problem.

Formally, the label of $\rho \in \mathcal{V}$, denoted by $l(\rho)$, is defined recursively as: $l(\rho) = 1$ if ρ is a source node, and $l(\rho) = 1 + \max_{\rho' \in N^-(\rho)}(l(\rho'))$ otherwise. The set of vertices that have the same label l defines a *level*, referred as *Level l*. The purpose of the labelling process is to be able to find some levels in the rotation poset such that every level corresponds to a unique stable matching.

Lemma 10 *All rotations that have the same label are incomparable.*

Proof. The proof is derived from the definition of the labelling function. If one rotation precedes another, i.e. they are comparable, the successors label must

be larger than the label of the predecessor. Therefore, if two rotations have the same label then they are incomparable. □

Using the intuition of the levels, we define a family of SM instances called F_w. Each SM instance in this family is identified by having a Level l such that the set of all rotations of size 2 in Level l combined with the rotations in the Level $l+1$ involve all the non-fixed men of the underlying instance. Then, we show that all SM instances in this family have a $(1,1)$-supermatch and we can find this $(1,1)$-supermatch by a polynomial-time algorithm. In Definition 17, we give a formal definition of F_w.

Definition 17 *F_w is a family of Stable Marriage instances such that each instance has a stable matching M, where the M and its corresponding closed subset S have the two following properties:*
1) All the sink rotations of size 2 in S have the same label;
2) The rotations of size 2 contained in the union of the sink rotations and neighbour rotations of S, cover all the non-fixed men (i.e. $|X(R^*(\mathbf{L}(S) \cup \mathbf{N}(S)))| = n$, where n denotes the number of non-fixed men).

The first observation on this family is given in Lemma 11.

Lemma 11 *All instances of F_w admit a $(1,1)$-supermatch.*

Proof. Proposition 3 (Page 67) and Property 2 from Definition 17 lead directly to this result. □

Lemma 12 *There exists a $(1,1)$-supermatch M in any instance $\mathcal{I}$ of F_w such that in M's closed subset S all the sink nodes of size 2 have the same label l and all the nodes in Level l are in S.*

Proof. Let M be such a $(1,1)$-supermatch for $\mathcal{I}$. Let L be the set of sink rotations of S. Assume there exists a set of rotations L^*, where $L \subseteq L^*$ such that all the rotations in L^* have the label l. Then, the closed subset S^* defined with $L \cup L^*$ as sink rotations corresponds to a $(1,1)$-supermatch. The proof is quite straightforward. We know from Proposition 3 that $|X(R^*(\mathbf{L}(S) \cup \mathbf{N}(S)))| = n$ (where n is the number of non-fixed men). Moreover, we have $R^*(\mathbf{L}(S^*) \cup \mathbf{N}(S^*))$ is a superset of $R^*(\mathbf{L}(S) \cup \mathbf{N}(S))$. Hence, $|X(R^*(\mathbf{L}(S^*) \cup \mathbf{N}(S^*)))| = n$ and by Proposition 3, the stable matching corresponding to S^* is a $(1,1)$-supermatch.

In short, if a subset of rotations in a level corresponds to a $(1,1)$-supermatch with the given properties, the closed subset that covers all the rotations in that

level also corresponds to a $(1,1)$-supermatch. Thus, in order to find a $(1,1)$-supermatch for $\mathcal{I}$, it is sufficient to look for the closed subset whose set of sink rotations contains all the rotations in a level. □

Definition 18 (π_{11}^w) *A particular case of π_{11} for problem family F_w.*
INPUT*: A Stable Marriage instance $\mathcal{I} \in \mathrm{F}_w$.*
OUTPUT*: Find a $(1,1)$-supermatch for $\mathcal{I}$?*

We now state our tractability result.

Theorem 7 *π_{11}^w is solvable in $O(|\mathcal{V}| + |E|)$ time.*

Proof. Consider an instance $\mathcal{I}$ in F_w. The principle of our polynomial-time procedure for any instance of F_w is to first identify the different levels, then to look for the closed subset whose set of sink rotations contains all the rotations in every level to see if it corresponds to a $(1,1)$-supermatch (until one is found). The levels in the rotation poset can be identified by applying topological sort on the rotation poset first and then by calculating the labels of rotations in the sorted list. Topological sorting results in a list of rotations such that if rotation ρ' is preceding rotation ρ in Π, then ρ' appears before ρ in the sorted list. The running time of topological sort is $O(|\mathcal{V}| + |E|)$.

Let N_l denote the set of all rotations at Level l, and N_{l+1} denote the ones at level $l+1$. Then, for any Level l, if the union set of all rotations of size 2 in Level l and Level $l+1$ contains all the non-fixed men, i.e. $|X(R^*(N_l \cup N_{l+1}))| = n$, then l corresponds to a $(1,1)$-supermatch. The corresponding stable matching is constructed by adding all predecessors of all the rotations at Level l, including the rotations themselves into the closed subset. Note that, the last Level l corresponds to a $(1,1)$-supermatch only if $|X(R^*(N_l))| = n$.

Let $m < |V|$ be the number of levels in the poset. The check at a level to see if it contains all the men can be done in constant time, and construction of the corresponding stable matching is $O(|\mathcal{V}|)$, which is computed only once at the end. Therefore, the running time of the overall algorithm is $O(|\mathcal{V}| + |E|)$ due to the Topological sorting.

Also, observe that the number of stable matchings this algorithm creates is bounded by the number of levels, where the number of levels is equal to the maximum number of edges in a path between a source vertex and a sink vertex. □

Note that, although the algorithm described in the proof of Theorem 7 is com-

Table 3.7: Preference lists for men (left) and women (right) for a Stable Marriage instance $\mathcal{I}_{\mathrm{F}}$ of size 8.

m_1	8 6 2 5 1 4 7 3
m_2	5 7 8 2 1 4 3 6
m_3	2 1 7 5 6 8 3 4
m_4	4 3 2 6 7 5 8 1
m_5	4 1 6 5 2 7 8 3
m_6	5 1 8 4 3 2 7 6
m_7	7 2 3 8 4 5 6 1
m_8	3 5 7 4 2 1 6 8

w_1	4 8 7 6 1 5 2 3
w_2	4 2 7 1 5 3 8 6
w_3	2 1 5 3 7 6 4 8
w_4	8 4 3 6 5 7 2 1
w_5	4 2 6 7 8 1 5 3
w_6	7 4 1 3 8 5 6 2
w_7	5 4 1 3 6 7 8 2
w_8	3 8 5 1 4 2 6 7

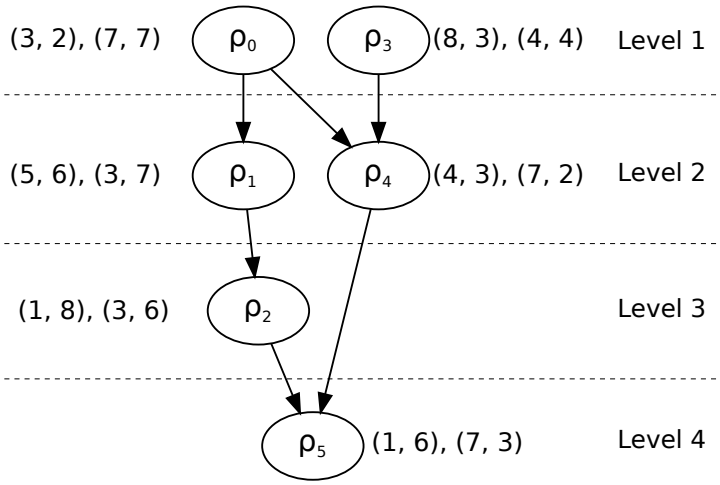


Figure 3.6: Rotation poset of the instance $\mathcal{I}_{\mathrm{F}}$ given in Table 3.7.

plete for the family F_w, it does not guarantee a solution to instances that are not in F_w. In Table 3.7 we give an example of a Stable Marriage instance $\mathcal{I}_{\mathrm{F}}$ of size 8, that is not in F_w. Figure 3.6 is the rotation poset that represents this instance, respectively. Note that, $\mathcal{I}_{\mathrm{F}}$ has a $(1,1)$-supermatch. For any such $\mathcal{I}_{\mathrm{F}}$, using the notion of levels of a rotation poset fails to find a $(1,1)$-supermatch.

Recall that, if a man has the same partner in all stable matchings of the underlying instance, then the man is said to be fixed. However, if he has at least one alternative partner, then the man is non-fixed. The instance $\mathcal{I}_{\mathrm{F}}$ contains two fixed men (m_2 and m_6) and 12 stable matchings in total. Only one stable matching, namely $M = \{(1,8),(2,5),(3,6),(4,3),(5,7),\ (6,1),(7,2),(8,4)\}$, is a $(1,1)$-supermatch. The closed subset corresponding to M is $S = \{\rho_0, \rho_1, \rho_3\}$. Let N_i denote a level identified by the algorithm described earlier as follows: $N_1 = \{\rho_0, \rho_3\}$, $N_2 = \{\rho_1, \rho_4\}$, $N_3 = \{\rho_2\}$, and $N_4 = \{\rho_5\}$. Additionally, the corresponding closed subsets for each level is as follows: $S_{N_1} = \{\rho_0, \rho_3\}$, $S_{N_2} = \{\rho_0, \rho_3, \rho_1, \rho_4\}$, $S_{N_3} = \{\rho_0, \rho_1, \rho_2\}$, $S_{N_4} = \{\rho_0, \rho_3, \rho_1, \rho_4, \rho_2, \rho_5\}$. We

can calculate the $(1, b)$-robustness of each stable matching corresponding to these levels. The stable matching of the sets S_{N_1}, S_{N_2}, and S_{N_3} are all $(1, 2)$-supermatches, whereas the stable matching of S_{N_4} is a $(1, 3)$-supermatch.

3.6.2 Finding an (a,0)-supermatch

We consider the case of finding an $(a, 0)$-supermatch and show that $(2, 0)$-supermatches do not exist for the Stable Marriage instances. Let $\boldsymbol{\pi_{a0}}$ denote the problem of finding an $(a, 0)$-supermatch.

Definition 19 ($\boldsymbol{\pi_{a0}}$) *A particular case of $\boldsymbol{\pi_{ab}}$. Decision problem for $(a, 0)$-supermatch.*
INPUT*: A Stable Marriage instance $\mathcal{I}$.*
QUESTION*: Is there an $(a, 0)$-supermatch for $\mathcal{I}$?*

Firstly note that $(1, 0)$-supermatches do not exist because in order to find new partners for a couple, at least one other couple must break up. We next show that $(2, 0)$-supermatches need not exist in general and then discuss the generic method for solving $\boldsymbol{\pi_{a0}}$.

Theorem 8 *Given any Stable Marriage instance where the number of non-fixed men n is at least 3, there do not exist any $(2, 0)$-supermatches.*

Proof. Suppose that a $(2, 0)$-supermatch M exists and let S be its closed subset. We argue that the only way to repair M (if two couples decide to break up) is to obtain a closed subset by either adding a rotation of size 2 to S or to remove a rotation of size 2 from S. Let m_1, m_2, m_3 be three distinct non-fixed men (recall that $n > 2$). For every pair of men $\{m_1, m_2\}, \{m_2, m_3\}, \{m_1, m_3\}$, there exists a rotation ρ of size 2 that involves both men such that $\rho \in$ **L(**S**)** or $\rho \in$ **N(**S**)**. Therefore, there necessarily exists a man $m \in \{m_1, m_2, m_3\}$ that is involved in two rotations ρ_1 and ρ_2 that are both in **L(**S**)** or both in **N(**S**)**. Since ρ_1 and ρ_2 involve m, then they are comparable. This contradicts the fact that **L(**S**)** (respectively **N(**S**)**) contains rotations that are incomparable. Therefore, $(2, 0)$-supermatches need not exist in general. □

We make the following observation regarding the general case of $(a, 0)$-supermatches. Let n denote the number of non-fixed men in a stable marriage instance $\mathcal{I}$ such that $2 < a \leq n$. Also, for a given stable matching M and a number of pairs in M to break-up, the repair matching denotes the closest stable matching to M, where the pairs that wish to break-up do not exist in the

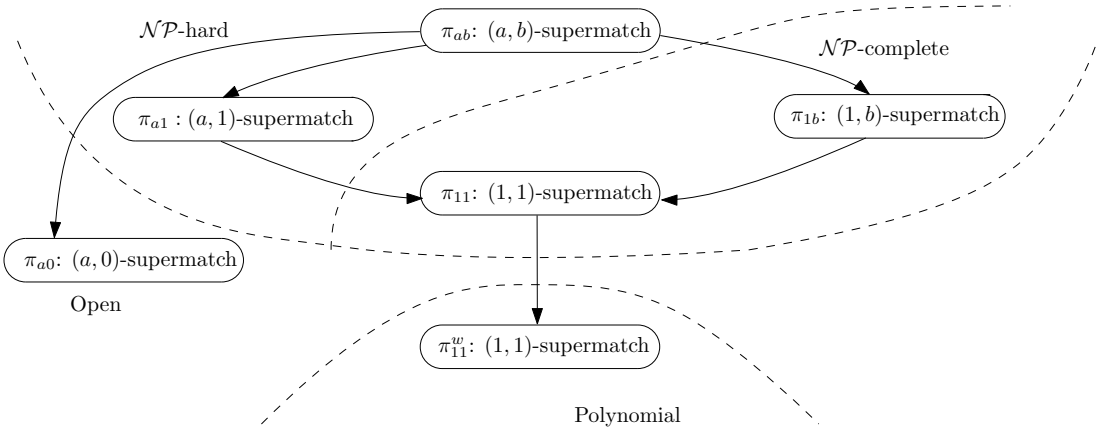


Figure 3.7: Illustration of the complexity hierarchy between the different cases of RSM.

repair matching. Suppose that M is a $(a,0)$-supermatch for $\mathcal{I}$. This means that it is possible to find a repair stable matching to M for a breakage involving every combination of non-fixed men of size a. Considering Theorem 8, we have the intuition that $(a,0)$-supermatches need not exist in general. However, if they exist, we suspect they exist in instances that have many number of stable matchings.

Conclusion. Figure 3.7 summarizes our findings in this chapter by also emphasizing the hierarchy between different cases of finding an (a,b)-supermatch. We do not know about the complexity class of finding $(a,0)$-supermatches, yet. However, we suspect they do not exist. If one can find a polynomial-time witness for (a,b)-supermatches or $(a,1)$-supermatches, these two problems can also be shown as $\mathcal{NP}$-complete. In fact, we suspect that the complexity of finding (a,b)-supermatches is Σ_3^P-complete as discussed in Chapter 6 in greater detail.

3.7 Chapter Summary

In this chapter, we introduced a novel robustness notion called (a,b)-supermatches for the matching problems under ordinal preferences. We studied in depth the decision problem of a sub-case of RSM, i.e. the $(1,1)$-supermatches. We first provided a characterization for $(1,1)$-supermatches. Then, we provided a model based on finding the independent sets on the rotation posets of the underlying SM instances.

In order to prove the $\mathcal{NP}$-completeness of deciding if there exists a $(1,1)$-supermatch to a given SM instance, we first defined a special SAT formulation. By using Schaefer's Dichotomy Theorem, we proved that this special formula is $\mathcal{NP}$-complete. Then, we showed equivalency between the two problems. We also showed that the $\mathcal{NP}$-completeness lifts up to $(1,b)$-supermatches. Currently, there is no polynomial-time witnesses for (a,b)-supermatches. Similarly, we cannot yet generalize our findings on the $\mathcal{NP}$-completeness to the $(a,1)$-supermatches. Therefore, we cannot generalize the complexity result. However, as another sub-problem, we showed that $(2,0)$-supermatches do not exist for the SM. However, there is no generalization of this result to the $(a,0)$-case, yet.

Chapter 4

Methods for Finding (1,b)-supermatches in RSM

Abstract. We define a polynomial-time procedure to decide if a given stable matching is a $(1, b)$-supermatch using the rotation poset of the underlying Stable Marriage instance. Then, we propose one complete, and three meta-heuristic models that are based on this polynomial-time procedure for solving the optimization problem of finding a $(1, b)$-supermatch that minimizes the value of b. These four models are: CP, local search, genetic algorithm, and a hybrid of both the local search and the genetic algorithm procedures. Subsequently, we perform experiments on two different datasets. One of them consists of uniformly random SM instances, and the other one consists of SM instances that contain many stable matchings (referred as MANY). We first report the performance comparison of the four models on the random RSM instances. Finally, we report how these models perform on MANY.

4.1 Notation and Definitions

We present a polynomial-time algorithm based on rotations to verify if a given stable matching is a $(1, b)$-supermatch. We use some of the notation defined for RSM in Section 3.2, and also introduce below a few others for this chapter. Notice that, for an SM instance that contains at least two different stable matchings, each stable matching M of the instance has a value b such that M

is a $(1, b)$-supermatch. We define the ***most robust stable matching*** of a given SM instance as a $(1, b)$-supermatch that has the minimum b value among all the $(1, b)$-supermatches of the underlying instance. We suppose that M is a given stable matching, S its corresponding closed subset, and (m_i, w_j) is the non-fixed pair in M that wishes to break-up. In order to avoid repetition, we use a notation for stable matchings and their corresponding closed subsets such that if a stable matching is identified using some superscripts or subscripts, then its corresponding closed subset contains them as well (i.e. the closed subset of M_i^j is denoted by S_i^j).

Recall the definitions of ***elimination*** and ***production rotations*** introduced in Section 3.2. Our work in this section is mostly based on these two special rotations. Given a rotation ρ exposed on a stable matching M and a pair $(m_i, w_j) \in \rho$, the stable matching M/ρ is said to be obtained after eliminating ρ from M. In this case, we say that the pair $(m_k, w_k), 0 \leq k \leq |\rho|$, is *eliminated* by ρ and (m_k, w_{k+1}) is *produced* by ρ. Suppose that there is a stable matching M, and rotation $\rho = (m_0, w_0), (m_1, w_1), \ldots, (m_{l-1}, w_{l-1})$ is exposed on the M. Then ρ is the elimination rotation for the pairs: $(m_0, w_0), (m_1, w_1), \ldots, (m_{l-1}, w_{l-1})$, and it is the production rotation for the pairs: $(m_0, w_1), (m_1, w_2), \ldots, (m_{l-1}, w_0)$.

For each pair (m_i, w_j) that appears in some stable matching M, the pair $(m_i, w_j) \notin M_0$, and $M \neq M_0$, there exists a unique production rotation $\boldsymbol{\rho_{p_{i,j}}}$ that produces (m_i, w_j). Similarly, if $(m_i, w_j) \notin M_Z$ and $M \neq M_Z$, then there exists a unique elimination rotation $\boldsymbol{\rho_{e_{i,j}}}$ that eliminates (m_i, w_j). Recall that it is always the case that M strictly dominates M/ρ and the addition operations $(+1)$ on rotations are modulo $|\rho|$.

We identify four different cases below to make it easier to show the existence of the production and elimination rotations. These four different cases are listed based on the existence of a stable pair (m_i, w_j) in the man-optimal and the woman-optimal stable matchings (M_0 and M_Z, respectively) as follows:

(1) $(m_i, w_j) \in M_0$ and $(m_i, w_j) \in M_Z$, or;

(2) $(m_i, w_j) \in M_0$ and $(m_i, w_j) \notin M_Z$, or;

(3) $(m_i, w_j) \notin M_0$ and $(m_i, w_j) \in M_Z$, or;

(4) $(m_i, w_j) \notin M_0$ and $(m_i, w_j) \notin M_Z$.

Case (1) refers to the case in which (m_i, w_j) is a fixed pair. For Case (2), the w_j is the most preferred stable partner for m_i, but m_i also has alternative, less

preferred, partner(s) than w_j. For Case (3), w_j is the least preferred stable partner for m_i and he has better alternative(s). The last case, Case (4), applies when m_i has more alternatives than w_j, where he can be partners with a more preferred woman (women) or less preferred ones.

If there is a stable matching M , and it includes the pair $(m_i, w_j) \in M$, then we can observe the production and elimination rotations of that pair as:

- If Case (1), then there do not exist a production or elimination rotation for the (m_i, w_j);
- If Case (2), then there exists a rotation that eliminates the (m_i, w_j);
- If Case (3), then there exists a rotation that produces the (m_i, w_j);
- If Case (4), then there exists a rotation that produces the (m_i, w_j) and there exists another one that eliminates it.

Suppose that a pair $(m_i, w_j) \in M$ wishes to break-up. Let M' denote the closest stable matching to M that does not include (m_i, w_j). Recall that the stable matching M' is said to be the repair matching for M for pair (m_i, w_j). The M' can either be a stable matching that dominates M in the lattice of stable matchings, it can be a stable matching that is dominated by M, or it can be incomparable to M. As an illustration, in the case of $M_0 \preceq M' \preceq M$, it means that there exists a rotation ρ that is in the closed subset of M and not in the closed subset of M' (i.e. $\rho \in S$ and $\rho \notin S'$). This rotation is the rotation that produces the pair (m_i, w_j) in M. However, if $M \preceq M' \preceq M_Z$, then there exists a rotation ρ that eliminates the pair in M. Hence, when it is included in a closed subset, we see a different partner for m_i than w_j. In this case, $\rho \notin S$ and $\rho \in S'$.[1]

Considering the above-mentioned cases based on precedence, we identify two different sets for each stable matching M. For any pair $(m_i, w_j) \in M$, if the production rotation $\rho_{p_{i,j}}$ exists, then there exists a set of stable matchings S_u, where each of them dominates M and does not include the (m_i, w_j). Similarly, if there exists the elimination rotation $\rho_{e_{i,j}}$, then there exists a set of stable matchings S_d, where each of the stable matchings in S_d is dominated by M and none of them include the pair (m_i, w_j).

[1] We show later in this section that an incomparable stable matching can never be the closest stable matching within the defined context. Therefore, we do not discuss this case for the moment.

4.2 Methodology for verifying a (1,b)-supermatch

In this section, our motivation is to find a procedure that verifies if a given stable matching is a $(1, b)$-supermatch, or not. Considering that there may be exponential number of stable matchings to the instance, enumerating all stable matchings may be impractical for some cases. We present here a procedure that decides if a given stable matching is a $(1, b)$-supermatch in polynomial-time.

The intuition of this procedure is due to the lattice structure of all stable matchings. In order to find if a stable matching M is a $(1, b)$-supermatch, one needs to find all the closest stable matchings to M for the break-up of each pair. We discuss below a procedure that shows how to find the closest stable matching M' to M given a pair in $(m_i, w_j) \in M$ and $(m_i, w_j) \notin M'$. We first show that the stable matching M' is either $M' \preceq M$ or $M \preceq M'$ in the lattice. We begin by showing how to construct both of these two stable matchings, then prove that one of them is the closest to M.

Recall that each man in a stable matching refers to a unique pair. We identify for each pair $(m_i, w_j) \in M$, for m_i, two stable matchings $\boldsymbol{M}^{*i}_{\text{UP}}$ and $\boldsymbol{M}^{*i}_{\text{DOWN}}$. The M^{*i}_{UP} represents a stable matching that dominates M and does not include the pair (m_i, w_j). Similarly, M^{*i}_{DOWN} represents a stable matching that is dominated by M and does not include the pair (m_i, w_j). It is important to note that M^{*i}_{UP} or M^{*i}_{DOWN} need not exist. For instance, all of the dominating stable matchings of M include the pair, it means that the pair is included in the man-optimal stable matching M_0 (Case (1) and Case (2)). In other words, w_j is the best possible partner for m_i. Therefore, an M^{*i}_{UP} does not exist. Similarly, if there exist no dominated stable matchings that does not include the unwanted pair, it means that w_j is the least preferred partner of m_i (Case (1) and Case (3)). Therefore, there does not exist an M^{*i}_{DOWN}.

Let us define the closed subsets of the above-mentioned two stable matchings M^{*i}_{UP} and M^{*i}_{DOWN}, where m_i wishes to break-up in our current stable matching M. We name their closed subsets as S^{*i}_{UP} and S^{*i}_{DOWN}, respectively. Note that given S_u as the set of all the dominating stable matchings of M, and S_d as the dominated ones: $M^{*i}_{\text{UP}} \in S_u$ and $M^{*i}_{\text{DOWN}} \in S_d$.

In Equation 4.1, we define the S^{*i}_{UP} as a closed subset that includes all the rotations in the corresponding closed subset S of M, except the rotation that produced (m_i, w_j), i.e. $\rho_{p_{i,j}}$, and all the successors of $\rho_{p_{i,j}}$ in S. More formally,

if $(m_i, w_j) \notin M_0$, we define a specific set of rotations S^{*i}_{UP} as follows:[2]

$$S^{*i}_{\text{UP}} = S \setminus (\{\rho_{p_{i,j}}\} \cup (N^+_t(\rho_{p_{i,j}}) \cap S)). \tag{4.1}$$

Subsequently, in Equation 4.2, we define S^{*i}_{DOWN} as a closed subset that contains all the rotations in S together with the rotation that eliminates (m_i, w_j), i.e. $\rho_{e_{i,j}}$. Additionally, we need all the predecessors of the $\rho_{e_{i,j}}$ that are not in S to make sure S^{*i}_{DOWN} is a closed subset. Formally, if $(m_i, w_j) \notin M_Z$, we define a specific set of rotations S^{*i}_{DOWN} as follows:

$$S^{*i}_{\text{DOWN}} = S \cup \{\rho_{e_{i,j}}\} \cup (N^-_t(\rho_{e_{i,j}}) \setminus S). \tag{4.2}$$

In Figure 4.1 we illustrate for a closed subset S and a pair (m_i, w_j) that wishes to break-up in M: the two sets S^{*i}_{UP}, S^{*i}_{DOWN}, and also the closed subset (S_k) of an incomparable matching M_k to be used later in this section.

Observe that the set S^{*i}_{UP} is obtained by removing the production rotation that produced the partner of $m_i \in M$ and all of its successors that are in S (i.e. $\rho_{p_{i,j}}$

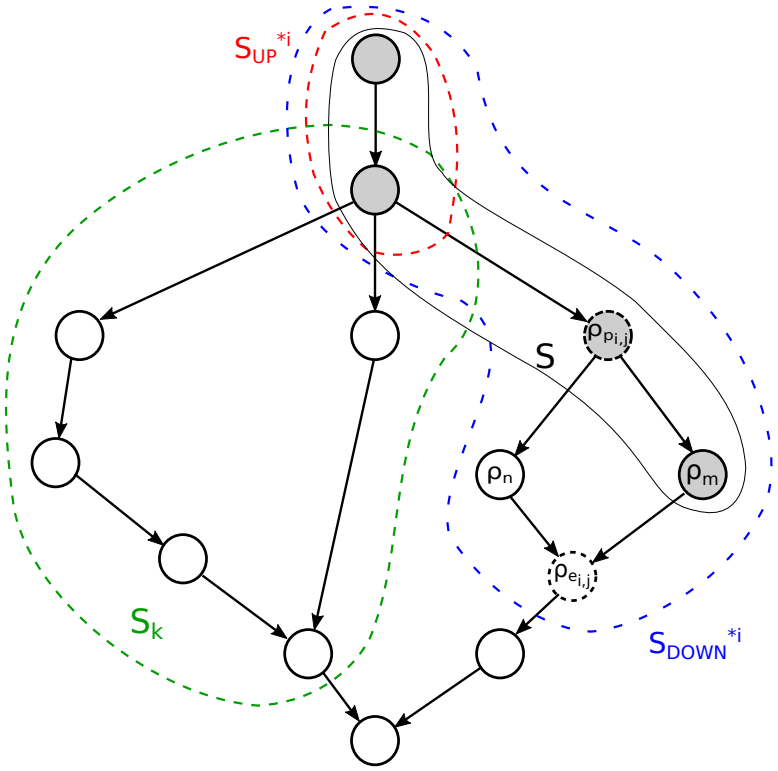


Figure 4.1: A set of closed subsets illustrated on a sample rotation poset.

[2]The brackets () in Equations 4.1 and 4.2 are used to emphasise priority between the operators.

and ρ_m). On the other hand, the set S^{*i}_{DOWN} is obtained by adding the elimination rotation that eliminates the partner of m_i in M and also all its predecessors that are not in S (i.e. $\rho_{e_{i,j}}$ and ρ_n). Observe that S^{*i}_{UP} and S^{*i}_{DOWN} are in fact, closed subsets since S is a closed subset.

For illustration, consider the stable matching M_5 in Figure 2.10 (Page 40), and its closed subset $S_5 = \{\rho_0, \rho_1, \rho_2\}$, which can be verified in Table 3.1 (Page 65). Table 4.1 shows for the break-up of each man (m_i, w_j) the closed subsets S^{*i}_{UP} and S^{*i}_{DOWN}, if one exists. If the closed subset does not exist, we denote it by the sign "-".

Table 4.2 shows the stable matchings corresponding to the closed subsets in Table 4.1 and the distances between the current stable matching M_5 to each one of them. The distances are denoted as d^i_{up} and d^i_{down} in the table, where for each man m_i, $d^i_{up} = d(M_5, M^{*i}_{\text{UP}})$ and $d^i_{down} = d(M_5, M^{*i}_{\text{DOWN}})$, respectively. If M^{*i}_{UP} does not exist for a man m_i, then d^i_{up} is set to ∞ (the same value is used when M^{*m}_{DOWN} does not exist). Lastly, $b_i = min(d^i_{up}, d^i_{down}) - 1$ represents the repair cost of each man. The reason for subtraction of 1 is because we are considering only $(1, b)$-supermatches, hence $a = 1$ (see Definition 12, Page 64).

Table 4.1: The closed subsets S^{*i}_{UP} and S^{*i}_{DOWN} for M_5.

(m_i, w_j)	$\rho_{p_{i,j}}$	$\rho_{e_{i,j}}$	S^{*i}_{UP}	S^{*i}_{DOWN}
(m_0, w_4)	ρ_2	ρ_3	$\{\rho_0, \rho_1\}$	$\{\rho_0, \rho_1, \rho_2, \rho_3\}$
(m_1, w_5)	ρ_1	ρ_5	$\{\rho_0\}$	$\{\rho_0, \rho_1, \rho_2, \rho_4, \rho_5\}$
(m_2, w_6)	-	ρ_4	-	$\{\rho_0, \rho_1, \rho_2, \rho_4\}$
(m_3, w_3)	-	ρ_5	-	$\{\rho_0, \rho_1, \rho_2, \rho_4, \rho_5\}$
(m_4, w_1)	-	ρ_3	-	$\{\rho_0, \rho_1, \rho_2, \rho_3\}$
(m_5, w_2)	ρ_2	-	$\{\rho_0, \rho_1\}$	-
(m_6, w_0)	ρ_1	ρ_4	$\{\rho_0\}$	$\{\rho_0, \rho_1, \rho_2, \rho_4\}$

Table 4.2: The repair stable matchings M^{*i}_{UP} and M^{*i}_{DOWN} for each man in M_5 following the Table 4.1 and the distances between M_5 and the repair stable matchings.

M^{*i}_{UP}	M^{*i}_{DOWN}	d^i_{up}	d^i_{down}	b_i
M_2	M_8	2	2	1
M_1	M_7	4	4	3
-	M_6	∞	2	1
-	M_7	∞	4	3
-	M_8	∞	2	1
M_2	-	2	∞	1
M_1	M_6	4	2	1

Table 4.3: The robustness values of all stable matchings for the sample given in Table 2.1.

M_k	S_k	b
M_0	$\{\}$	5
M_1	$\{\rho_0\}$	4
M_2	$\{\rho_0, \rho_1\}$	3
M_3	$\{\rho_0, \rho_1, \rho_4\}$	2
M_4	$\{\rho_0, \rho_1, \rho_4, \rho_5\}$	3
M_5	$\{\rho_0, \rho_1, \rho_2\}$	3
M_6	$\{\rho_0, \rho_1, \rho_2, \rho_4\}$	1
M_7	$\{\rho_0, \rho_1, \rho_2, \rho_4, \rho_5\}$	3
M_8	$\{\rho_0, \rho_1, \rho_2, \rho_3\}$	3
M_9	$\{\rho_0, \rho_1, \rho_2, \rho_3, \rho_4\}$	2
M_{10}	$\{\rho_0, \rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$	3

The robustness of a stable matching is characterized by the repair cost of the non-fixed man that has the worst repair cost $b = \sum_{i \in \{1 \ldots n\}} max(b_i)$. For instance, for the given example, M_5 is characterized as a $(1, 3)$-supermatch as men m_1 and m_3 require at least 3 other men to change their partners. Moreover, Table 4.3 lists the $(1, b)$-robustness values for each stable matching for the SM instance in Figure 2.10 (Page 40). The most robust stable matching in this example is M_6 since it has the smallest value for b. Similarly, M_0 is the least robust stable matching.

We now show the correctness of our method. Recall that $X(\rho)$ denotes the set of men involved in the rotation ρ. Lemma 13 gives a main characterization for the incomparable rotations.

Lemma 13 *Given two incomparable rotations ρ and ρ', the set of men in ρ and ρ' are disjoint (i.e. $X(\{\rho\}) \cap X(\{\rho'\}) = \emptyset$).*

Proof. By definition of incomparability, if two rotations are incomparable, it means that they modify a set of men who do not require modifications from the other first. Therefore the sets of men are distinct. □

In Lemma 14 we characterize the distance relation between comparable stable matchings. We show that if two stable matchings are close in the lattice in terms of the length of the shortest path that connect them, they are also closer in terms of our distance function.

Lemma 14 *Given three stable matchings M_x, M_y and M_z such that $M_x \preceq M_y \preceq M_z$, then M_z (or M_x) is closer to M_y than M_x (or M_z) i.e. $d(M_y, M_z) \leq d(M_x, M_z)$*

and $d(M_x, M_y) \leq d(M_x, M_z)$.

Proof. Using the properties of domination and the closed subsets in Theorem 3 (Page 44), we can infer $S_x \subset S_y \subset S_z$.

Assume to the contrary that $d(M_y, M_z) > d(M_x, M_z)$. This situation occurs only if a set of pairs that are present in M_x are eliminated to obtain M_y and then re-matched with the same partners they had in M_x to get M_z. However, this contradicts Corollary 1 (Page 41). For similar reasons, $d(M_x, M_y) < d(M_x, M_z)$. □

Remark. Observe for Lemma 14 that the distance of a stable matching to two different stable matchings can be the same, i.e. $d(M_x, M_y) = d(M_x, M_z)$. This case occurs if the rotation set in the difference sets $S_y \setminus S_x$, $S_z \setminus S_x$, and $S_z \setminus S_y$ include the same set of men. We demonstrate this case on a Stable Marriage instance $\mathcal{I}$ of size 8 given in Manlove's book, Page 91 [Man13]. Figure 4.2 illustrates the lattice of the stable matchings of $\mathcal{I}$.

Consider the three stable matchings $M_8 \preceq M_{15} \preceq M_{21}$ in this instance. The stable matchings can be identified by starting with the man-optimal matching (denoted by M_1 on the figure) and exposing each rotation one by one by respecting their order until the desired stable matching is reached. The rotations on the path from M_1 to a stable matching M_i identifies the corresponding closed subset of M_i (denoted by S_i). The pairs involved in these three matchings are as follows:

$$M_8 = \{(1,1),(2,3),(3,4),(4,8),(5,2),(6,5),(7,6),(8,7)\}$$
$$M_{15} = \{(1,5),(2,4),(3,3),(4,6),(5,8),(6,7),(7,2),(8,1)\}$$
$$M_{21} = \{(1,7),(2,8),(3,2),(4,1),(5,6),(6,4),(7,3),(8,5)\}$$

The closed subsets corresponding to these three stable matchings are:

$$S_8 = \{\rho_1, \rho_2, \rho_3\}$$
$$S_{15} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6\}$$
$$S_{21} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8, \rho_9, \rho_{10}\}$$

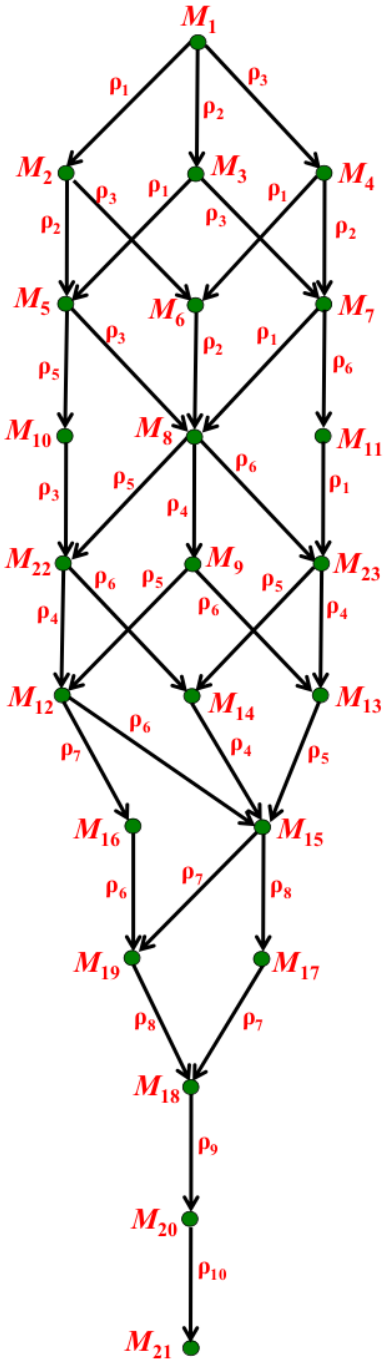


Figure 4.2: A sample Stable Marriage instance of 8 men and 8 women from Manlove [Man13].

The pairs involved in the rotations are presented below:

$$\rho_1 = (1,3),(2,1), \qquad \rho_6 = (4,8),(7,6),(5,2),$$
$$\rho_2 = (3,7),(5,4),(8,2), \qquad \rho_7 = (3,3),(8,1),$$
$$\rho_3 = (4,5),(7,8),(6,6), \qquad \rho_8 = (2,4),(5,8),(6,7),$$
$$\rho_4 = (1,1),(6,5),(8,7), \qquad \rho_9 = (1,5),(5,7),(8,3),$$
$$\rho_5 = (2,3),(3,4), \qquad \rho_{10} = (3,1),(7,2),(5,3)(4,6).$$

In this case, the difference sets are:

$$S_{15} \setminus S_8 = \{\rho 4, \rho 5, \rho 6\}$$
$$S_{21} \setminus S_{15} = \{\rho 7, \rho 8, \rho 9, \rho 10\}$$
$$S_{21} \setminus S_8 = \{\rho 4, \rho 5, \rho 6, \rho 7, \rho 8, \rho 9, \rho 10\}$$

Due to the fact that $X(S_{15} \setminus S_8) = X(S_{21} \setminus S_{15}) = X(S_{21} \setminus S_8) = \{1,2,3,4,5,6,7,8\}$, all 8 men have different partners in all three stable matchings. In addition, the distances of all of them are equal to each other $d(M_8, M_{15}) = d(M_{15}, M_{21}) = d(M_8, M_{21}) = 8$.

Having defined two stable matchings M^{*i}_{UP} and M^{*i}_{DOWN} above, we now show that one of them is the closest stable matching to M when m_i wishes to break-up. As the first step, in Lemma 15 we show that M^{*i}_{UP} is the closest stable matchings to M in the lattice when compared with all other dominating stable matchings in S_u. Subsequently, in Lemma 16 we show that M^{*i}_{DOWN} is the closest to M when compared with all other dominated stable matchings in S_d. The reader is referred to the illustration given in Figure 4.1 (Page 97), as it demonstrates the sets used in Lemma 15, 16, and 17.

Lemma 15 *If there exists a stable matching M_x that: (a) does not contain (m_i, w_j), (b) dominates M, and (c) different from M^{*i}_{UP}, then M_x dominates M^{*i}_{UP}.*

Proof. $M^{*i}_{\text{UP}} \preceq M$ by definition. Suppose by contradiction that there exists an M_x such that $(m_i, w_j) \notin M_x$ and $M^{*i}_{\text{UP}} \preceq M_x \preceq M$. It implies that $S^{*i}_{\text{UP}} \subset S_x \subset S$. In this case, the difference set $(S_x \setminus S^{*i}_{\text{UP}})$ contains the rotation that produced the pair and also all its successors is S. Formally, $(S_x \setminus S^{*i}_{\text{UP}}) \subset \Big\{\{\rho_{p_{i,j}}\} \cup \{N^+_t(\rho_{p_{i,j}}) \cap S\}\Big\}$. Adding any rotation from this set to S_x results in a contradiction by either adding (m_i, w_j) to the matching, thereby not breaking that couple, or because

the resulting set is not a closed subset. □

Example. An example M_x as discussed in Lemma 15 could be illustrated in Figure 4.1 as a S_k that contains only the sink vertex of the rotation poset.

Lemma 16 *If there exists an M_x that: (a) does not contain (m_i, w_j), (b) is dominated by M, (c) and different from M^{*i}_{DOWN}, then M^{*i}_{DOWN} dominates M_x.*

Proof. Similar to the proof above, suppose that there exists an M_x such that $(m_i, w_j) \notin M_x$ and $M \preceq M_x \preceq M^{*i}_{\text{DOWN}}$. We have $S \subset S_x \subset S^{*i}_{\text{DOWN}}$. It implies that the difference set $(S_x \setminus S)$ contains the elimination rotation of the pair and its all preceding rotations that are not in S. More formally, $(S_x \setminus S) \subset \Big\{\{\rho_{e_{i,j}}\} \cup \{N_t^-(\rho_{e_{i,j}}) \setminus S\}\Big\}$. In order to add $\rho_{e_{i,j}}$ all other rotations must be added to form a closed subset. If all rotations are added, $S = S^{*i}_{\text{DOWN}}$ which results in a contradiction. □

Example. An example M_x as discussed in Lemma 16 could be illustrated in Figure 4.1 as a S_k that contains all the rotations in S^{*i}_{DOWN} and the immediate successor of $\rho_{e_{i,j}}$.

Finally, we show in Lemma 17 that an incomparable stable matching to M cannot be closer to M than either M^{*i}_{UP} or M^{*i}_{DOWN}. Note that, an incomparable stable matching M_k is already illustrated in Figure 4.1.

Lemma 17 *For any stable matching M_k incomparable with M such that M_k does not contain the pair (m_i, w_j), M^{*i}_{UP} is closer to M than M_k.*

Proof. Let S_k be the closed subset corresponding to M_k, and S be that corresponding to M. First, we consider the case in which $S_k \cap S = \emptyset$. If the closed subsets have no rotations in common the rotations in these sets are incomparable. Using Lemma 13, $X(S_k) \cap X(S) = \emptyset$. Therefore, $d(M_k, M) = |X(S_k)| + |X(S)|$, whereas $d(M^{*i}_{\text{UP}}, M) \leq |X(S)|$.

Second, we consider the case in which $S_k \cap S \neq \emptyset$. Let M_c be the closest dominating stable matching of both M_k and M^{*i}_{UP}, along with S_c as its corresponding closed subset. Using Lemma 14 we know that $d(M^{*i}_{\text{UP}}, M) \leq d(M_c, M)$, where $d(M_c, M) = |X(S \setminus S_c)|$.

Using Lemma 13 we know that $X(S_k \setminus S_c) \cap X(S \setminus S_c) = \emptyset$. Therefore, $d(M_k, M) = |X(S_k \setminus S_c)| + |X(S \setminus S_c)|$. By substituting the formula above, $d(M_k, M) \geq |X(S_k \setminus S_c)| + d(M^{*i}_{\text{UP}}, M)$. Using the fact that $|X(S_k \setminus S_c)| > 0$ from the definition of M_k, we can conclude that $d(M_k, M) > d(M^{*i}_{\text{UP}}, M)$. □

Theorem 9 is a direct consequence of Lemmas 15, 16, and 17. It concludes that the repair stable matching for M for pair (m_i, w_j) is one of the stable matchings we have identified.

Theorem 9 *The closest stable matching of a stable matching M given the unwanted pair (m_i, w_j) is either M^{*i}_{UP} or M^{*i}_{DOWN}.*

Proof. The proof is immediate from Lemmas 15, 16, and 17. □

Lemma 18 identifies the stable matching in any SM instance that is brittle, i.e. it has the highest value of b.

Lemma 18 *For any SM instance, either M_0 or M_Z is the $(1, b)$-supermatch that has the largest b among all stable matchings of the underlying instance.*

Proof. Suppose for contradiction that there exists a stable matching M, where $M \notin \{M_0, M_Z\}$, and M has a larger b value among those three stable matchings when their $(1, b)$-robustness is computed. Assume that this large value of b of M is due to the distance for M to a repair matching M'. We know that the man-optimal stable matching M_0 dominates all other stable matchings, and the woman-optimal one M_Z is dominated by all others (i.e. $M_0 \preceq M' \preceq M_Z$). By Lemma 14, for any stable matching M' that is $M_0 \preceq M' \preceq M$ or $M \preceq M' \preceq M_Z$, M' is closer to M than the man/woman optimal matchings. This means that, if $M_0 \preceq M' \preceq M$, then $d(M, M')leqd(M, M_0)$. Similarly, if $M \preceq M' \preceq M_Z$, then $d(M, M')leqd(M, M_Z)$. Hence, the supposition contradicts. □

4.2.1 Complexity

We show that checking if a stable matching is a $(1, b)$-supermatch can be performed in total, in $O(n \times |\mathcal{V}|)$ time after a $O(n^2 + |\mathcal{V}|^2)$ preprocessing step. We explain the reasoning below.

The pre-processing step consists of building: the rotation poset (in $O(n^2)$ time), the transitive predecessor and successor lists $N_t^-(\rho)$, $N_t^+(\rho)$ for each rotation ρ (in $O(|\mathcal{V}|^2)$ time), and the identification of $\rho_{e_{i,j}}$ and $\rho_{p_{i,j}}$ for each pair (m_i, w_j) whenever applicable (in $O(n^2)$ time). The construction of the rotation poset takes $O(n^2)$ as discussed in Section 2.4.1 (Page 42). The construction of the lists $N_t^-(\rho)$, $N_t^+(\rho)$ for each rotation is performed by identifying all the rotations in the rotation poset, where the number of rotations is $O(|\mathcal{V}|)$. Therefore, this identification takes $O(|\mathcal{V}|^2)$. The identification of the elimination and pro-

duction rotations for each pair is performed by searching for all the rotations in the rotation poset. There are exactly $n/2$ pairs and $|\mathcal{V}|$ rotations yielding in $O(n + |\mathcal{V}|)$. Combining all the three orders, we obtain a pre-processing time of $O(n^2 + |\mathcal{V}|^2)$. Note that, the number of rotations $|\mathcal{V}|$ is bounded by $n(n-1)/2$.

Next, we compute S^{*i}_{UP} and S^{*i}_{DOWN} for each man. Note that S^{*i}_{UP} and S^{*i}_{DOWN} can be constructed in $O(|\mathcal{V}|)$ time (by definition of S^{*i}_{UP} and S^{*i}_{DOWN}) for each man m_i (see Equation 4.1 and Equation 4.2). Note that, computing the distance $d(M^{*i}_{\text{UP}}, M)$ is equal to the number of men participating in the rotations that are eliminated from S to obtain S^{*i}_{UP}. The difference set, $S \setminus S^{*i}_{\text{UP}}$ obtains, in the worst case, all the rotations in the poset, i.e. $O(|\mathcal{V}|)$. The same applies to calculating the distance $d(M^{*i}_{\text{DOWN}}, M)$. Last, if $b < d(M^{*i}_{\text{UP}}, M) - 1$ and $b < d(M^{*i}_{\text{DOWN}}, M) - 1$, we know that it is impossible to repair M when m_i needs to change his partner with at most b other changes. Otherwise, M is a $(1, b)$-supermatch. Hence, the defined procedure takes $O(n \times |\mathcal{V}|)$, which can be equal to $O(n^3)$, in the worst case.

As a final remark, we obtain a closed subset S as the solution to the procedure defined. Hence, there also exists an additional cost to convert the solution S into its corresponding stable matching M. In order to do that, starting from M_0, all the rotations in S starting from the sink rotation of S by respecting the precedence order, must be eliminated. The resulting matching M', is obtained by eliminating $|\rho|$ pairs for each rotation ρ in S.

4.3 Constraint Programming Model

Constraint programming (CP) is one of the most powerful techniques for solving combinatorial search problems by expressing the relations between decision variables as constraints [RvBW06]. In this section we give a CP formulation for finding the most robust stable matching, i.e. the $(1, b)$-supermatch with the minimum b. The idea is to formulate the Stable Marriage problem using rotations, then extend that formulation in order to compute the two values $d(M^{*i}_{\text{UP}}, M)$ and $d(M^{*i}_{\text{DOWN}}, M)$ (where M is the solution) for each man m_i so that the value of b is always greater or equal to one of them. Using rotations to model stable matching problems has been used in [GI89] on Page 194, and also in [Fed92, FIM07, SO17].

4.3.1 Variables

We first introduce all the variables to be used in the model. Let the set of rotations be $\mathcal{V} = \{\rho_1, \rho_2, \ldots, \rho_{|\mathcal{V}|}\}$. The list below shows the notation we use in our CP model.

M :	The solution stable matching.
S:	The closed subset corresponding to M.
SP:	The set of all non-fixed, stable pairs.
NM:	The set of all men that appears in at least one of the pairs in SP, also referred as the non-fixed men.
R_i:	The set of rotations in which man m_i is involved.
S^{*i}_{UP}:	The closed subset corresponding to the closest dominating stable matching M^{*i}_{UP} when m_i in M wishes to break-up.
S^{*i}_{DOWN}:	The closed subset corresponding to the closest dominated stable matching M^{*i}_{DOWN} for m_i in M wishes to break-up.
$\rho_{p_{i,j}}$:	For each pair $(m_i, w_j) \notin M_0$, $\rho_{p_{i,j}}$ denotes the unique identifier for the rotation that produces (m_i, w_j).
$\rho_{e_{i,j}}$:	For each $(m_i, w_j) \notin M_Z$, $\rho_{e_{i,j}}$ denotes the unique identifier for the rotation that eliminates the pair (m_i, w_j).
w_i^0:	The unique identifier for the man m_i's partner in M_0.
w_i^Z:	The unique identifier for the man m_i's partner in M_Z.

Notice that $p_{i,j}$ and $e_{i,j}$ are integer values that point to the index of the rotation that they denote, i.e. $p_{i,j}, e_{i,j} \in \{1, \ldots, |\mathcal{V}|\}$. Similarly, for a pair $(m_i, w_j) \in M_0$, $w_i^0 = j$ and $(m_i, w_j) \in M_Z$, $w_i^Z = j$.

In our model, we assume that an $O(n^2)$ pre-processing step is performed to compute M_0, M_Z, SP, NM, the rotation poset, $\rho_{e_{i,j}}$, and $\rho_{p_{i,j}}$ for every pair (m_i, w_j) whenever applicable by using the methods described in Section 2.4.1 (Page 37) and Section 4.1 (Page 93). Additionally, the lists of predecessors and successors of each rotation ρ including transitivity ($N_t^-(\rho)$, $N_t^+(\rho)$) are computed. This step takes additional $O(|\mathcal{V}|^2)$ time. The running times are discussed in detail in Section 4.2.1.

Now, we define the variables to be used in our model.

$x_{i,j}$: A boolean variable $x_{i,j}$ for each pair $(m_i, w_j) \in SP$ to indicate if (m_i, w_j) is part of the solution matching M. It represents the decision variable in the model.

b: An integer variable b with the initial domain $[1, |NM| - 1]$. The lower bound for the domain is set to 1, because in order to obtain another stable matching at least 1 other man needs to have a different partner. Additionally, the upper bound is set to $|NM| - 1$ because in the worst case every other person needs to change their partners to accommodate m_i's wish to break-up. This variable represents the objective.

α_i: A boolean variable for each $m_i \in NM$ to indicate if $(m_i, w_{w_i^0})$ is in the solution M.

β_i: A boolean variable for each $m_i \in NM$ to indicate if $(m_i, w_{w_i^Z})$ is in the solution M.

s_v: A boolean variable s_v for each rotation $\rho_v, v \in \{1, \ldots, |\mathcal{V}|\}$ to indicate if ρ_v is in S.

$s^i_{up_v}$: A boolean variable for each $m_i \in NM, \rho_v \in \mathcal{V}$ to indicate if ρ_v is in the difference set $S \setminus S^{*i}_{\text{UP}}$.

$s^i_{down_v}$: A boolean variable for each $m_i \in NM, \rho_v \in \mathcal{V}$ to indicate if ρ_v is in the difference set $S^{*i}_{\text{DOWN}} \setminus S$.

y^i_l: A boolean variable for each man $m_l \in NM$ and $m_l \in NM$ to indicate in order to find the repair S^{*i}_{UP} when m_i breaks up with his partner, if the partner of m_l needs to be changed or not.

z^i_l: A boolean variable for each man $m_l \in NM$ and $m_l \in NM$ to indicate in order to find the repair S^{*i}_{DOWN} when m_i breaks up with his partner, if the partner of m_l needs to be changed or not.

4.3.2 Constraints

Using the terminology described above, we construct the CP model step by step. In order to make it clear for the reader, we separate different logical blocks into different subsections and use parentheses in equations to emphasize priority

between the operators.

SM formulation. We first express that a pair is a part of a solution if and only if it is produced by a rotation (if it is not a part of M_0) and not eliminated by another (if it is not a part of M_Z). More specifically, the first set of constraints ensures that if a pair (m_i, w_j) is in M, the rotation that produces (m_i, w_j) is in S and the rotation that eliminates is not in S. Hence, Constraint 4.3 is required to keep track of the pairs that are in the solution.

$$\begin{aligned} &\forall (m_i, w_j) \in SP\text{:} \\ &x_{i,j} \leftrightarrow \neg s_{e_{i,j}} && \text{, if } (m_i, w_j) \in M_0, \\ &x_{i,j} \leftrightarrow s_{p_{i,j}} && \text{, if } (m_i, w_j) \in M_Z, \\ &x_{i,j} \leftrightarrow (s_{p_{i,j}} \wedge \neg s_{e_{i,j}}) && \text{, otherwise.} \end{aligned} \tag{4.3}$$

A set of rotations corresponds to the closed subset of the solution if and only if all the parents of each rotation in the set are also in the set, and the rotations produce the pairs in the solution. Constraint 4.4 ensures that the solution is a stable matching by ensuring that S is a closed subset.

$$\begin{aligned} &\forall \rho_v \in \mathcal{V}, \forall \rho_{v'} \in N^-(\rho_v)\text{:} \\ &\quad s_v \rightarrow s_{v'}. \end{aligned} \tag{4.4}$$

Having ensured that the result is always a stable matching, and keeping track of the pairs, the next step is to find the closest stable matchings S^{*i}_{UP} and S^{*i}_{DOWN}. In order to find these stable matchings, we focus on the differences between the solution M and them. In other words, we construct the difference sets $S \setminus S^{*i}_{\text{UP}}$ and $S^{*i}_{\text{DOWN}} \setminus S$.

Building the difference set $S \setminus S^{*i}_{\text{UP}}$. The following set of constraints (Constraint 4.5 and Constraint 4.6) are required to build the difference set $S \setminus S^{*i}_{\text{UP}}$. These constraints are specifically for keeping track of the pairs included in the difference sets. Constraint 4.5 handles the special case when man m_i is matched to his partner in M_0.

$$\begin{aligned} &\forall m_i \in NM\text{:} \\ &\alpha_i \leftrightarrow x_{i,w_i^0}. \end{aligned} \tag{4.5}$$

Constraint 4.6 is used to indicate that the pair (m_i, w_j) is a part of the solution

(i.e. the repair stable matching) if and only if its production rotation is included in the difference set $S \setminus S^{*i}_{\text{UP}}$.

$$\forall m_i \in NM, \forall (m_i, w_j) \in SP, \text{ where } j \neq w_i^0\text{:}$$
$$x_{i,j} \leftrightarrow s^i_{up_{p_{i,j}}}. \tag{4.6}$$

The set of constraints given between Constraint 4.7 to Constraint 4.9 are for keeping track of the rotations in $S \setminus S^{*i}_{\text{UP}}$. Constraint 4.7 is used to ensure if the current partner of a man is not produced by a rotation, then there does not exist a dominating repair stable matching and neither its closed subset S^{*i}_{UP}.

$$\forall m_i \in NM, \forall \rho_v \in \mathcal{V}\text{:}$$
$$\alpha_i \rightarrow \neg s^i_{up_v}. \tag{4.7}$$

Constraint 4.8 ensures that if there exists a rotation ρ in the closed subset S of the given matching and it is in the difference set (i.e. it needs to be removed from the S to construct the S^{*i}_{UP}), then all of ρ's successors in S must also be in the difference set (i.e. the successors must be removed, respectively).

$$\forall m_i \in NM, \forall \rho_v \in \mathcal{V}, \forall \rho_{v'} \in N^+(\rho_v)\text{:}$$
$$s^i_{up_v} \wedge s_{v'} \rightarrow s^i_{up_{v'}}. \tag{4.8}$$

Then, in order to construct the difference set for the dominating stable matching fully, we specify the constraints from the other direction, i.e. if a rotation ρ is in the difference set, it must be in the closed subset S of the current matching M, and either the pairs it produces are in M or at least one other successor of ρ is in the difference set. There are two cases to distinguish as specified in Constraint 4.9.

$$\forall m_i \in NM, \forall \rho_v \in \mathcal{V}\text{:}$$
$$s^i_{up_v} \rightarrow \Big(s_v \wedge (x_{i,j} \vee \bigvee_{\rho_{v*} \in N^-(\rho_v)} s^i_{up_{v*}})\Big) \quad \text{,if } \rho_v \text{ produces } (m_i, w_j), \tag{4.9}$$
$$s^i_{up_v} \rightarrow \Big(s_v \wedge (\bigvee_{\rho_{v*} \in N^-(\rho_v)} s^i_{up_{v*}})\Big) \quad \text{,otherwise.}$$

Building the difference set $S^{*i}_{\text{DOWN}} \setminus S$. Following the same intuition as above, we use the following set of constraints to represent the difference set $S^{*i}_{\text{DOWN}} \setminus S$. We are not adding detailed explanations for each case as their logic is the same as the cases above for constructing of the difference set $S \setminus S^{*i}_{\text{UP}}$ used for construction of the closest dominating stable matching.

$$\forall m_i \in NM : \beta_i \leftrightarrow x_{i,w_i^Z}. \tag{4.10}$$

$$\forall m_i \in NM, \forall (m_i, w_j) \in SP, \text{ where } j \neq w_i^Z : x_{i,j} \leftrightarrow s^i_{down_{e_{i,j}}}. \tag{4.11}$$

$$\forall m_i \in NM, \forall \rho_v \in \mathcal{V} : \beta_i \rightarrow \neg s^i_{down_v}. \tag{4.12}$$

$$\forall m_i \in NM, \forall \rho_v \in \mathcal{V}, \forall \rho_{v'} \in N^-(\rho_v) : s^i_{down_v} \wedge \neg s_{v'} \rightarrow s^i_{down_{v'}}. \tag{4.13}$$

$$\begin{aligned} &\forall m_i \in NM, \forall \rho_v \in \mathcal{V} : \\ &s^i_{down_v} \rightarrow \Big(\neg s_v \wedge (x_{i,j} \vee \bigvee_{\rho_{v*} \in N^+(\rho_v)} s^i_{down_{v*}})\Big) \quad \text{,if } \rho_v \text{ eliminates } (m_i, w_j), \\ &s^i_{down_v} \rightarrow \Big(\neg s_v \wedge (\bigvee_{\rho_{v*} \in N^+(\rho_v)} s^i_{down_{v*}})\Big) \quad \text{,otherwise.} \end{aligned} \tag{4.14}$$

Counting the repair cost. Next, we define a number of constraints to count exactly how many men must change their partners in order to provide a repair to M using the difference sets for S^{*i}_{UP} and S^{*i}_{DOWN} (i.e. $S \setminus S^{*i}_{\text{UP}}$ and $S^{*i}_{\text{DOWN}} \setminus S$).

Constraints 4.15 and 4.16 are required to keep track if man m_l must change his partner in M to obtain the S^{*i}_{UP} (or S^{*i}_{DOWN}) upon the break-up of man m_i. The idea is that, if m_l appears in any of the rotations in the difference set, it means his partner is changed to provide the repair stable matching. Constraint 4.15 is the case for obtaining S^{*i}_{UP}, whereas Constraint 4.16 is for the S^{*i}_{DOWN}.

$$\begin{aligned} &\forall m_l \neq m_i \in NM\text{:} \\ &y^i_l \leftrightarrow \bigvee_{\rho_v \in R_l} s^i_{up_v}, \text{ and} \end{aligned} \tag{4.15}$$

$$z^i_l \leftrightarrow \bigvee_{\rho_v \in R_l} s^i_{down_v}. \tag{4.16}$$

After finding out which men are involved in the difference sets, the next step is to count the number of them. Constraint 4.17 is for counting the number of additional men (excluding the man that wishes to break-up m_i) that are changing their partners in the difference set to S^{*i}_{UP}. Similarly, Constraint 4.18 is for S^{*i}_{DOWN}. Note that, we also specify the case that if the man m_i has never been produced by a rotation, there does not exist an S^{*i}_{UP}, therefore the number of men that must be modified is ∞ (or n can be used to denote that it is even

larger than the worst possible repair cost). The same applies to the other set, the closest dominated stable matching.

$$\forall m_l \neq m_i \in NM\text{:}$$
$$\alpha_i \rightarrow d^i_{up} = n \text{ and } \neg\alpha_i \rightarrow d^i_{up} = \sum y^i_l, \text{ and} \tag{4.17}$$
$$\beta_i \rightarrow d^i_{down} = n \text{ and } \neg\beta_i \rightarrow d^i_{down} = \sum z^i_l. \tag{4.18}$$

Finally, we constrain the value of b to indicate that b holds the maximum value of the required repairs when the repair stable matchings of each man's break-up is considered.

$$\forall m_i \in NM\text{:}$$
$$\Big(\neg\alpha_i \rightarrow (b \geq d^i_{up})\Big) \vee \Big(\neg\beta_i \rightarrow (b \geq d^i_{down})\Big) \tag{4.19}$$

To conclude the model, objective is to $minimise\ b$. An example of a CP model of an instance can be found in Appendix A.

Note that, the initial upper bound for b can easily be lowered by using some preprocessing. One such strategy is to use a greedy approach in which we traverse a chain in the lattice of stable matchings starting from M_0 to M_Z. For instance, Algorithm 3 (Page 43) can be used to find this maximal chain. Then, one can check each stable matching on this chain for a $(1, b)$-supermatch and compute the b values. The minimum b value on this chain then sets an upper bound on the variable b. However, note that this preprocessing step can be very costly because it computes $(1, b)$-values for each stable matching, where the number of stable matchings on this chain can be $n(n-1)/2$. For a faster but less effective upper bound adjustment, consider Lemma 18 (Page 104). We know that either M_0 or M_Z sets the highest value for b. We already know of the existence of M_0 and the M_Z after applying the Gale-Shapley algorithm (detailed in Algorithm 2, Page 39). Computing the b value of these two $(1, b)$-supermatches provides us information about the worst case b among all the stable matchings. If the $(1, b)$-robustness of the M_0 and M_Z are different from each other, then the one that has a lower b value sets the upper bound for the b value in the CP model.

4.4 Genetic Algorithm Approach

Genetic Algorithms (GA) are being used extensively in optimization problems as an alternative to traditional heuristics [Hol92]. Our approach is based on the

traditional GA, where evolutions are realised after the initialisation of a random set of solutions. The process terminates either when an acceptable solution is found or the search loop reaches a pre-determined limit, i.e. maximum iteration count, time limit, etc.

The search process begins from an initial population of solutions. Each solution in this set represents an *individual*. Each individual in the population is associated with a *fitness value* indicating how good it is with respect to the objective. The quality of the population is improved through two main operators: crossover and mutation [BNKF98]. The process terminates either when an acceptable solution is found or the search loop reaches a pre-determined limit. The genetic algorithm that we propose to find the most robust stable matching is described below with a look-up example demonstrating one iteration of the algorithm.

4.4.1 Initialization

The purpose of the initialization step is to generate a number of random stable matchings for constructing the initial population set, denoted by $\boldsymbol{P}$. Recall that each closed subset in the rotation poset corresponds to a stable matching. The random stable matching generation is performed by selecting a random rotation ρ from the rotation poset $\Pi = (\mathcal{V}, E)$ of the given SM instance, and initializing a subset of rotations S by adding ρ. Then, all predecessors of ρ are added to S to make sure S is a closed subset and subsequently the corresponding stable matching M is found as outlined in Algorithm 6.

Line 6 in Algorithm 6 uses a method CREATESM(S) which converts the given closed subset S to its corresponding stable matching M by exposing all the

Algorithm 6 Random stable matching creation

1: **procedure** CREATERANDOMSM(Π)
   Input: The rotation poset $\Pi = (\mathcal{V}, E)$
   Output: A random stable matching M
2: $\rho \leftarrow$ SELECTRANDOM($\mathcal{V}$)
3: $S \leftarrow \{\rho\}$
4: **for** $\rho_p \in$ ALLPREDECESSORS(ρ) **do**
5: $S \leftarrow S \cup \{\rho_p\}$
6: $M \leftarrow$ CREATESM(S)
   return M

Algorithm 7 Initialization of the population

```
1: procedure INITIALIZE(n, Π)
       Input: A positive integer n, rotation poset Π = (V, E)
       Output: A set of random stable matchings P
2:     P ← ∅
3:     i ← 0
4:     while i < |P| do
5:         M_i ← CREATERANDOMSM(Π)
6:         P ← P ∪ {M_i}
7:         i ← i + 1
       return P
```

rotations in S, respecting their precedence order starting from M_0. For the initial population, $|\boldsymbol{P}|$ such random stable matchings are created and added to the population set $\boldsymbol{P}$ as outlined in Algorithm 7.

Example. Let us illustrate this step on the Stable Marriage instance given in Table 2.1 (Page 38). Assume that the population size $|\boldsymbol{P}|$ is set to 3. In order to generate the three initial stable matchings, three random rotations are selected from the rotation poset, namely: ρ_0, ρ_1, ρ_5 (find the rotation poset of the SM instance in Figure 2.11, Page 44). Since ρ_0 has no predecessors, the stable matching that $\{\rho_0\}$ corresponds is obtained by exposing only ρ_0 on M_0, resulting in M_1 (find the lattice of all stable matchings in Figure 2.10, Page 40).

Subsequently, the closed subset obtained by adding all predecessors of ρ_1 corresponds to M_2, where $S_2 = \{\rho_0, \rho_1\}$ and the closed subset for ρ_5 corresponds to M_4, where $S_4 = \{\rho_0, \rho_1, \rho_4, \rho_5\}$. Thus, the population consists of: $\boldsymbol{P} = \langle M_1, M_2, M_4 \rangle$.

4.4.2 Evaluation

For each stable matching M_i, we denote by b_i its robustness value for representing a $(1, b)$-supermatch. At the evaluation step, we compute the value b_i of each $M_i \in \boldsymbol{P}$. Then, a fitness value v_i, normalised to the interval $[0, 1]$, is assigned to each such M_i. Given two stable matchings from the population $M_i, M_j \in \boldsymbol{P}$, if $b_i < b_j$, then M_i is said to be more fit than M_j. On the other hand, if $v_i < v_j$, then M_j is said to be more fit than M_i as it has a greater fitness value.

Let b_{max} denote the maximum b value in the population. First, we get the complements of the numbers with b_{max} to reflect the relation between b_i and

v_i, that is a stable matching with a larger v_i is a more fit solution. Moreover, a small constant c_o is added to each v_i to make sure even the least fit stable matching M_i, i.e. $b_i = b_{max}$, is still a solution that can take role in the evolution of the population (see Equation 4.20). Then, the normalization is applied as shown in Equation 4.21.

$$v_i = b_{max} + c_o - b_i. \tag{4.20}$$

$$v_i = \frac{v_i}{\sum_{M_i \in \boldsymbol{P}} v_i}. \tag{4.21}$$

These steps are detailed in Algorithm 8. The first for loop finds the maximum b value. The second one finds the v_i for each M_i by applying Equation 4.20, and also the total sum for the normalization step. Then, the last loop normalizes the values as shown in Equation 4.21.

Example. The b values for the stable matchings given in the look-up example, where $\boldsymbol{P} = \langle M_1, M_2, M_4 \rangle$ are calculated as $b_1 = 4$, $b_2 = 3$, and $b_4 = 3$, by the polynomial procedure defined in Section 4.1. Given these values, $b_{max} = 4$. For the sake of the example, let us fix c_0 to 0.5. Then, the fitness values are calculated as :

$$v_1 = 4 + 0.5 - 4 = 0.5,$$

$$v_2 = 4 + 0.5 - 3 = 1.5,$$

$$v_4 = 4 + 0.5 - 3 = 1.5.$$

Algorithm 8 Evaluation of the fitness values

```
1:  procedure EVALUATION(P)
        Input: Current population P
        Output: none
2:      b_max ← 0
3:      for M_i ∈ P do
4:          if b_i > b_max then
5:              b_max ← b_i
6:      sum_v ← 0
7:      for M_i ∈ P do
8:          v_i ← (b_max + c_0 − b_i)
9:          sum_v ← (sum_v + v_i)
10:     for M_i ∈ P do
11:         v_i ← (v_i / sum_v)
```

Then, the sum of the fitness values is $sum_v = v_1 + v_2 + v_4 = 3.5$. By normalizing the fitness values, we obtain the following values: $v_1 = 0.143, v_2 = 0.429, v_4 = 0.429$ indicating that M_2 and M_4 are the fittest stable matchings in $\boldsymbol{P}$.

4.4.3 Evolution

We use roulette wheel selection to select a random stable matching in our procedure [LL12]. Algorithm 9 provides details on the roulette wheel selection procedure we use. First, a random number $r \in [0, 1]$ is generated. Then, the fitness values of the stable matchings in the population are visited. At each step, a sum of the fitness values seen so far are recorded in a variable named sum_c . This procedure continues until a stable matching M_i is found according to the criteria at Line 5 in Algorithm 9. This criteria ensures that if a stable matching has a higher fitness value, then it has a greater chance to be selected.

The evolution step is then continued by applying crossover and mutation on selected stable matchings. We will present the details of the crossover and mutation operators below. First, two stable matchings are selected by the roulette wheel selection. Let these matchings named M_1 and M_2. If these matchings are different from each other, the crossover procedure is applied. After applying crossover on M_1, M_2 and obtaining the resulting stable matchings M_{c1}, M_{c2}, the population is refined.

Subsequently, between Lines 8 and 12 of Algorithm 10, a stable matching M_m is selected for mutation. If M_m is different from the fittest stable matching M_{fit}, and a random mutation probability is satisfied (i.e. a random number $rand \in [0, 1]$ is generated and checked if it is less than a fixed value $rand < p_m$) then M_m is mutated. The mutation step may result in producing either better or worse solutions. We do not apply a refinement procedure after mutation,

Algorithm 9 Roulette Wheel Selection

```
1: procedure SELECTION(P)
       Input: Current population P
       Output: A stable matching M_i ∈ P
2:     r ← RANDOMDOUBLE(0, 1)
3:     sum_c ← 0
4:     for M_i ∈ P do
5:         if (sum_c ≤ r) ∧ (r < sum_c + v_i) then return M_i
6:         sum_c ← sum_c + v_i
```

Algorithm 10 Evolution of the population

```
1: procedure EVOLUTION(P)
       Input: Current population P
       Output: none
2:     M_1 ← SELECTION(P)
3:     M_2 ← SELECTION(P)
4:     if M_1 ≠ M_2 then
5:         (M_c1, M_c2) ← CROSSOVER(M_1, M_2)
6:         REFINE(M_c1, M_c2)
7:         EVALUATION(P)
8:     M_fit ← GETFITTEST(P)
9:     M_m ← SELECTION(P)
10:    rand ← RANDOM(0, 1)
11:    if M_m ≠ M_fit and rand < p_m then
12:        MUTATION(M_m)
```

Algorithm 11 Refinement of the population

```
1: procedure REFINE(M_c1, M_c2)
       Input: Current population P and two stable matchings M_c1, M_c2
       Output: none
2:     P ← P ∪ {M_c1, M_c2}
3:     (M_l1, M_l2) ← LEASTFITTWOSMS(P)
4:     P ← P \ {M_l1, M_l2}
```

instead directly apply mutation on M_m.

The refinement procedure is detailed in Algorithm 11. In the refinement procedure, the given M_{c1}, M_{c2} are immediately added to the population $\boldsymbol{P}$. Then, the worst two solutions, in other words the two least fit stable matchings M_{l1} and M_{l2} are removed from the population.

Next, we go into the details of the crossover and mutation operations.

Crossover. The procedure for the crossover is detailed in Algorithm 12. Given two stable matchings M_1, M_2 and their corresponding closed subsets S_1 and S_2, one random rotation is selected from each closed subset. Let ρ_1 and ρ_2 denote the randomly selected rotations from each set. The method works as follows: if S_2 already contains the rotation ρ_1, then S_{c2} is constructed by adding all the rotations in S_2 but excluding the ρ_1 and all its successor rotations that are in S_2. We denote this removal process by REMOVE(S_2, ρ_1). However, if $\rho_1 \notin S_2$, then ρ_1 and all its predecessors that are not included in S_2 are added to S_{c2} with the rotations that are already in S_2 to form the new closed subset S_{c2}. We denote

Algorithm 12 Crossover procedure

```
1: procedure CROSSOVER(M_1, M_2)
       Input: Two stable matchings M_1, M_2 ∈ P
       Output: Two stable matchings M_c1, M_c2
2:     ρ_1 ← RANDOM(S_1)
3:     ρ_2 ← RANDOM(S_2)
4:     if ρ_1 ∈ S_2 then
5:         S_c2 ← REMOVE(S_2, ρ_1)
6:     else
7:         S_c2 ← ADD(S_2, ρ_1)
8:     if ρ_2 ∈ S_1 then
9:         S_c1 ← REMOVE(S_1, ρ_2)
10:    else
11:        S_c1 ← ADD(S_1, ρ_2)
12:    M_c1 ← CREATESM(S_c1)
13:    M_c2 ← CREATESM(S_c2)
       return (M_c1, M_c2)
```

the addition process by the method ADD(S_2, ρ_1). This new closed subset S_{c2} corresponds to one of the stable matchings produced by the crossover, namely M_{c2}. Similarly, the same process is repeated for S_1 by constructing S_{c1}, and another stable matching M_{c1} is obtained.

Example. Recall that, the current population for the example is $\boldsymbol{P} = \langle M_1, M_2, M_4 \rangle$, where their closed subsets are $S_1 = \{\rho_0\}, S_2 = \{\rho_0, \rho_1\}$, and $S_4 = \{\rho_0, \rho_1, \rho_4, \rho_5\}$. As an example, assume that M_1 and M_4 are selected for crossover by the roulette wheel selection. One random rotation is selected from each set e.g.: $\rho_0 \in S_1$ and $\rho_4 \in S_4$. Then, one of the produced stable matchings is M_3 since $\rho_4 \notin S_1$ and $\{\rho_0\} \cup \{\rho_4, \rho_1\} = S_3$. The second one is M_0 by removing all the ρ_0, ρ_1, ρ_4 and ρ_5 from S_4. After adding new stable matchings, the population becomes $\boldsymbol{P} = \langle M_1, M_2, M_4, M_0, M_3 \rangle$. Then the population is refined by removing the two least fit stable matchings, which are in this case M_0 because $b_0 = 5$ and M_1 because $b_1 = 4$. Hence, the refined population after crossover is $\boldsymbol{P} = \langle M_2, M_4, M_3 \rangle$.

Mutation. Let M be the stable matching selected for the mutation and S denote its corresponding closed subset. The procedure starts by selecting a random rotation ρ from the rotation poset $\mathcal{V}$. Then, similar to the crossover procedure above, if $\rho \notin S$, then ρ and all its predecessors required to form a closed subset are added to S. However, if $\rho \in S$, then ρ and all its successors in S are

Algorithm 13 Mutation procedure

```
1: procedure MUTATION(M)
       Input: A stable matching M ∈ P
       Output: none
2:     ρ ← RANDOM(V)
3:     if ρ ∈ S then
4:         S ← REMOVE(S, ρ)
5:     else
6:         S ← ADD(S, ρ)
```

removed from S. Algorithm 13 outlines the formal procedure.

Example. In order to demonstrate mutation on the example population, assume that M_3, where $S_3 = \{\rho_0, \rho_1, \rho_4\}$, is selected for mutation and ρ_2 is the randomly selected rotation from the rotation poset. Since $\rho_2 \notin S_3$, ρ_2 and all its predecessors are added to S_3, resulting in $\{\rho_0, \rho_1, \rho_4\} \cup \{\rho_2\} = S_6$. The new stable matching is M_6, which is added to the population and the original stable matching used for mutation, M_3, is deleted. The final population is: $\boldsymbol{P} = \langle M_6, M_2, M_4 \rangle$.

General Procedure. The procedure for the overall GA procedure is given in Algorithm 14. The evolution step is repeated until either a solution is found or the termination criteria are met. In our case, we have three termination criteria: a time limit lim_{time}, an iteration limit to cut-off the process if there

Algorithm 14 General procedure of genetic algorithm

```
1: procedure GENETICALGORITHM(Π)
       Input: The rotation poset Π = (V, E)
       Output: The fittest stable matching M_fit
2:     P ← INITIALIZE(n, Π)
3:     EVALUATION(P)
4:     M_fit ← null
5:     b_fit ← 0
6:     cnt ← 0
7:     while time < lim_time or b_fit != opt do
8:         EVOLUTION(P)
9:         EVALUATION(P)
10:        if M_fit is updated then
11:            cnt ← 0
12:        else
13:            cnt ← cnt + 1
14:        if cnt = lim_iter then return M_fit
       return M_fit
```

is no improved solution for the specified number of iterations lim_{iter}, or if the optimal b is found as $b_{fit} = opt = 1$. For the cut-off termination criterion, we keep a counter (cnt) to count the number of iterations with no improvement, i.e. more fit solution than the current fittest solution. This counter is increased at each unimproved iteration until it meets lim_{iter}. If a more fit solution is generated during the process, the counter is reset to $cnt = 0$.

Complexity. The overall procedure is based on the polynomial-time method described in the Section 4.1. Recall that this procedure takes $O(n \times |\mathcal{V}|)$ time after an $O(n^2 + |\mathcal{V}|^2)$ preprocessing step. In order to speed up the process of calculating $(1, b)$-robustness values from a practical point of view, in our experiments an extra data structure is used to memorize the fitness value of each generated stable matching. Note that, the search space may have exponential number of stable matchings. This look-up table structure saves on time to recompute robustness of a stable matching. In both crossover and mutation phases of GA, the worst-case time complexity for one call is bounded by $O(|\mathcal{V}|)$, which is also lifted to the evolution step.

The evaluation phase evaluates the fitnesses of all the individuals, stable matchings, in the population, where the evaluation depends on the computation of the b. Note that, the values of b are not computed for each stable matching in the population at each iteration due to the look-up table. However, the fitnesses of all $|\boldsymbol{P}|$ individuals are computed. Therefore, the evaluation step may take $O(n \times |\mathcal{V}| \times |\boldsymbol{P}|)$ steps in the worst case.

In summary, the general complexity is linear with respect to the number of rotations and $O(k \times n \times |\mathcal{V}| \times |\boldsymbol{P}|)$, where k denotes the iteration count and n denotes the number of non-fixed men, and the maximum value of $|\mathcal{V}|$ is $n(n-1)/2$.

4.5 Local Search Approach

In our local search model, we use the well-known iterated local search with restarts approach [Stü98, LMS03]. The first step of the algorithm is to find a random solution to start with. A random stable matching M is created by using the same initialization method detailed in Algorithm 6 in Section 4.4 for the GA procedure. Briefly, this procedure is based on selecting a random rotation from the rotation poset, and adding all its predecessors to form a closed subset.

Then, a set of neighbour stable matchings is created using the properties of the rotation poset.

4.5.1 Neighbourhood

A *neighbour* in this context is defined as a stable matching, whose corresponding closed subset differs only by one rotation from the closed subset S of the matching M. Recall that **L(**S**)** denotes the set of sink rotations S. Removing one rotation $\rho_i \in$ **L(**S**)** from S corresponds to a different dominating neighbour of M. Similarly, **N(**S**)** is previously defined as denoting the set of rotations that are not included in S and for each $\rho \in$ **N(**S**)** either $d_{in}(\rho) = 0$ or all of their predecessors are in S. In the same manner, adding one rotation from **N(**S**)** to S corresponds to a dominated neighbour of M.

Example. Let us first demonstrate these two sets on an example. Figure 4.3 illustrates an example rotation poset, where the closed subset is $S = \{\rho_0, \rho_1, \rho_3, \rho_4, \rho_5, \rho_7\}$. The set of sink rotations in this set is identified as **L(**S**)** $= \{\rho_1, \rho_4, \rho_7\}$.

Similarly, the set of neighbour rotations is **N(**S**)** $= \{\rho_2, \rho_6\}$ since all their predecessors are already included in S. For the stable matching that corresponds to S, there exist 5 different neighbours. These five neighbours are listed by using their closed subsets as: $S_1 = \{\rho_0, \rho_3, \rho_4, \rho_5, \rho_7\}$, $S_2 = \{\rho_0, \rho_1, \rho_3, \rho_5, \rho_7\}$, $S_3 = \{\rho_0, \rho_1, \rho_3, \rho_4, \rho_5\}$, $S_4 = \{\rho_0, \rho_2, \rho_1, \rho_3, \rho_4, \rho_5, \rho_7\}$, and $S_5 = \{\rho_0, \rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7\}$.

Algorithm 15 expands on the procedure for identifying the set of neighbour

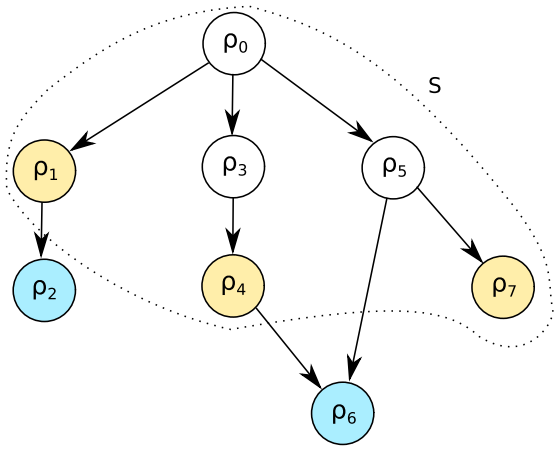


Figure 4.3: Illustration of the sets **L(**S**)** $= \{\rho_1, \rho_4, \rho_7\}$ and **N(**S**)** $= \{\rho_2, \rho_6\}$ on a sample rotation poset for a given closed subset S.

Algorithm 15 Identification of the neighbour set

```
1:  procedure FINDNEIGHBOURS(M, Π)
        Input: A stable matching M and the rotation poset Π = (V, E)
        Output: A set of stable matchings N
2:      N ← ∅
3:      for ρ ∈ V do
4:          if ρ ∉ S then
5:              cnt ← 0
6:              for e ∈ INCOMINGEDGES(ρ) do
7:                  if e.source ∉ S then
8:                      cnt ← cnt + 1
9:                      break
10:             if cnt = 0 then
11:                 M_n ← CREATESM(S ∪ {ρ})
12:                 N ← N ∪ {M_n}
13:     for ρ ∈ S do
14:         cnt ← 0
15:         for e ∈ OUTGOINGEDGES(ρ) do
16:             if e.target ∈ S then
17:                 cnt ← cnt + 1
18:         if cnt = 0 then
19:             M_n ← CREATESM(S \ {ρ})
20:             N ← N ∪ {M_n}
        return N
```

stable matchings $\boldsymbol{N}$ of a given stable matching M. Lines 3 to 12 give detail on the procedure of finding the set of neighbour rotations $\mathbf{N}(S)$. A variable called cnt is used to count the number of incoming edges of ρ, where the source node is not in S. If this counter variable has a value of $cnt = 0$, it means that it is a neighbour rotation.

Similarly, Lines 13 to 20 define the procedure for identifying the sink rotations $\mathbf{L}(S)$. Each neighbour stable matching is denoted by M_n. Again, a variable called cnt is used to count the number of outgoing edges of ρ whose target node is in S. If $cnt = 0$, it means that all of the successor of ρ are in S, which identifies ρ as a sink rotation.

4.5.2 Search

The general procedure of our local search model is detailed in Algorithm 16. Let M_c denote the current stable matching. We sometimes refer to this sta-

Algorithm 16 General procedure for local search

```
1: procedure LocalSearch(Π)
       Input: The rotation poset Π = (V, E)
       Output: The (1, b)-supermatch M_best that minimizes b
2:     M_c ← CreateRandomSM(Π)
3:     M_best ← M_c
4:     cnt ← 0
5:     while time < lim_time do
6:         cnt ← 0
7:         iter ← 0
8:         while iter < lim_desc do
9:             N ← FindNeighbours(M_c, Π)
10:            M_n ← Best(N)
11:            if b_n < b_c then
12:                M_c ← M_n
13:                if b_c < b_best then
14:                    M_best ← M_c
15:                    cnt ← 0
16:            cnt ← cnt + 1
17:            iter ← iter + 1
18:            if cnt = lim_iter or b_best = opt then return M_best
19:        M_c ← CreateRandomSM(Π)
       return M_best
```

ble matching as the parent stable matching. The procedure first identifies the neighbours of the current stable matching. At each iteration, if a neighbour M_n has lower b than the parent stable matching M_c (i.e. $b_n < b_c$), in other words, it is a more robust solution, the search continues by using the M_n as the parent stable matching for the next iteration. A variable M_{best} is used to keep track of the best solution found so far in the search process.

We denote the best stable matching in set $\boldsymbol{N}$ by M_n and the function $\text{BEST}(\boldsymbol{N})$ is used to find the stable matching with the lowest value of b in set $\boldsymbol{N}$. Also a counter cnt is used to count the number of iterations with no improved solutions. The cut-off limit is bounded by a variable denoted by lim_{iter} to stop the search if there is no improvement for the specified number of iterations.

There is also an iteration limit lim_{desc} that indicates the depth of search for the neighbours of the current stable matching. In other words, it is the number of iterations that descend from a randomly created stable matching. In this context, we define the depth as the number of successor stable matchings created starting from a random stable matching. For instance, in Line 2 in Algorithm 16, an initial random stable matching M_c is created. Assuming that the depth is set

to 2, i.e. $limit_{desc} = 2$, first $\boldsymbol{N}$ of M_c is created. Then, on the second iteration, the best neighbour M_n's neighbourhood is explored. After the second iteration, instead of searching the best neighbour of the M_n's best neighbour, the parent matching is set to another random stable matching. The search continues in this fashion until a termination criteria is met.

As one of our termination criterion given in Line 18, if the stable matching M_{best} has the optimal robustness, where $b_{best} = opt = 1$, then the search is terminated and M_{best} is returned as the answer as it is the optimal solution.

Example. Let us illustrate the neighbourhood search on the SM instance given in Table 2.1 (Page 38). Assume that the search starts with the randomly generated stable matching, M_5, where $S_5 = \{\rho_0, \rho_1, \rho_2\}$. The sets **L(**$S_5$**)** and **N(**$S_5$**)** are identified as follows: **L(**S_5**)** $= \{\rho_2\}$, **N(**S_5**)** $= \{\rho_3, \rho_4\}$. Therefore, the current stable matching M_5 has three neighbours as follows:

$$S_5 \setminus \{\rho_2\} = \{\rho_0, \rho_1\} = S_2,$$

$$S_5 \cup \{\rho_3\} = \{\rho_0, \rho_1, \rho_2, \rho_3\} = S_8,$$

$$S_5 \cup \{\rho_4\} = \{\rho_0, \rho_1, \rho_2, \rho_4\} = S_6.$$

Then, the neighbourhood is $\boldsymbol{N} = \{M_2, M_6, M_8\}$. Next step is to calculate the robustness of each stable matching in $\boldsymbol{N}$. Using the method described in Section 4.1, the b values of all the stable matchings are calculated as follows: $b_5 = 3, b_2 = 3, b_8 = 3, b_6 = 1$. Since M_6 is the most robust stable matching in $\boldsymbol{N}$ and more robust than the current stable matching M_5, the neighbourhood search for the next iteration descends from M_6. Additionally, no that, the M_6 is the optimal solution. Therefore, it is returned as the solution.

Complexity. The complexity of the LS procedure depends on the computation of the b values. Finding neighbours is based on identification of the sink rotations of M_c, where there can be at most $|\mathcal{V}|$ sink rotations. Then, there is a constant cost for removing each sink rotation, or adding each neighbour rotation. The best neighbour is identified after computing b values of $|\boldsymbol{N}|$ stable matchings. As we did in the GA, we use a look-up table to remember the b values of already computed stable matchings. The overall procedure takes $O(k \times n \times |\mathcal{V}| \times |\boldsymbol{N}|)$, where k is the number of iterations and n is the number of non-fixed men.

4.6 Genetic Local Search (Hybrid) Approach

In our hybrid model (HB), we use a genetic local search hybrid. The HB model we propose is quite straightforward. The structure of the GA model stays the same and the neighbourhood search of the LS algorithm is applied on the products of the crossover operation to enhance the quality of the produced stable matchings.

The outline of the HB model differs from the GA model only in the evolution phase (refer to Algorithm 10, Page 116). Algorithm 17 shows the evolution method that adapts the LS model's neighbourhood search used in the HB. The Lines between 5 and 9 denotes the hybridization. The local search is adapted in these lines by finding the neighbourhood of the products of the crossover operations and selecting the best neighbour in the neighbourhood. If there is no better neighbour of M_{c1}, then the M_{c1} remains unchanged (similarly, M_{c2}).

The details of the methods FINDNEIGHBOURS and BEST are discussed in Section 4.5 for the GA and is used as exactly the same for the HB model. The details of the remaining functions are also explained in Section 4.4.3.

Complexity. At each iteration of the algorithm, the robustness values of all the neighbours of both M_{c1} and M_{c2} as well as the robustness of all the population

Algorithm 17 Evolution of the population

1: **procedure** EVOLUTION(Π)
   Input: The rotation poset $\Pi = (\mathcal{V}, E)$
   Output: The $(1, b)$-supermatch M_{best} that minimizes b
2: $M_1 \leftarrow$ SELECTION($\boldsymbol{P}$)
3: $M_2 \leftarrow$ SELECTION($\boldsymbol{P}$)
4: **if** $M_1 \neq M_2$ **then**
5: $(M_{c1}, M_{c2}) \leftarrow$ CROSSOVER(M_1, M_2)
6: $\boldsymbol{N} \leftarrow$FINDNEIGHBOURS($M_{c1}, \Pi$)
7: $M_{c1} \leftarrow$ BEST($\boldsymbol{N}$)
8: $\boldsymbol{N} \leftarrow$FINDNEIGHBOURS($M_{c2}, \Pi$)
9: $M_{c2} \leftarrow$ BEST($\boldsymbol{N}$)
10: REFINE(M_{c1}, M_{c2})
11: EVALUATION($\boldsymbol{P}$)
12: $M_{fit} \leftarrow$ GETFITTEST($\boldsymbol{P}$)
13: $M_m \leftarrow$ SELECTION($\boldsymbol{P}$)
14: $rand \leftarrow$ RANDOM$(0, 1)$
15: **if** $M_m \neq M_{fit}$ and $rand < p_m$ **then**
16: MUTATION(M_m)

in the evaluation step is calculated. Note that, if the value of b of a stable matchings was computed before, we fetch the value from a look-up table to avoid the re-computation. If we denote by $\boldsymbol{N'}$ all the neighbours of M_{c1} and M_{c2} at a single iteration, the overall complexity of the algorithm is changed as $O(k \times n \times |\mathcal{V}| \times |\boldsymbol{N'} + \boldsymbol{P}|)$.

4.7 Experiments

We have four models developed to find the most robust stable matching of a given SM instance. These are namely a Constraint Programming model (CP), an iterative local search approach (LS), a genetic algorithm approach (GA), and a hybrid of the LS and the GA (HB). In the next section, we compare their performances on different datasets.

We use Java 8 to perform our experiments [GJS+14]. The CP model is implemented using Choco 4.0.1 [PFL16] and the two meta-heuristics are implemented in Java by using the JGraphT library [SKM+15]. All experiments were performed on Dell M600s with 2.66 Ghz processors under Linux. The plots were prepared using Gnuplot [WKm15]. For each test, we ran three tests using different randomization seeds and reported the average of them. The time limit lim_{time} of each run is limited to 20 mins unless otherwise stated. We first use a dataset that consists of randomly created instances. We further compare the models on a dataset that consists of instances that contain (possibly) many number of stable matchings.

Unless otherwise stated, the parameter combinations for the models used during the tests are as given below. In our tests, we ran the CP model by using geometric restarts policy that adjusts a cut-off value and has shown to be an effective strategy [W+99]. In order to use our models, we performed some parameter tuning tests on randomly generated SM instances $n \in \{100 \times k \mid k \in \{1, \ldots, 10\}\}$, where n is the number of men/women, using different parameter combinations. We do not share the results of these tests as they are not very interesting, and the behaviour of the graphs are predictable. We list below the selected parameter combination for each model.

We used the greedy heuristic described in Section 4.3 to restrict the domain of b. We fixed $lim_{iter} = 10000$ and $lim_{desc} = 50$ for the LS model. Allowing a large cut-off limit lim_{iter} allows the model the perform the search for longer. This gives

the model a slightly increased chance of not getting stuck at a local minima. Additionally, a large the depth for the neighbour search reduces the number of random restarts. Our GA parameters are fixed as: $|\boldsymbol{P}| = 60$, $lim_{iter} = 10000$, and $p_m = 0.8$. A large population allows to explore more different stable matchings. The mutation probability is set to a high value to increase the randomization a little, and help the algorithm not getting stuck at local minimas. Using a similar reasoning for the HB model, we fixed $lim_{iter} = 10000$, $|\boldsymbol{P}| = 60$, and $p_m = 0.7$. We later discuss this model using two different population sizes $|\boldsymbol{P}| = 60$ and $|\boldsymbol{P}| = 10$. We slightly decreased the mutation probability when compared to the p_m in GA, as the neighbourhood search also helps with the diversity of the procedure.

4.7.1 Random Instances

We generated two datasets of randomly generated SM instances to compare the performances of CP, LS, GA, and the HB model. We refer to the first set as the DATA-S and the second one as the DATA-L. The first dataset DATA-S consists of 50 uniformly random SM instances for each size $n \in \{50 \times k \mid k \in \{1, \ldots, 6\}\}$, where n denotes the number of men/women. Note that, DATA-S contains 300 instances. The second set DATA-L contains 50 instances for each size $n \in \{100 \times k \mid k \in \{4, ..., 20\}\}$.

The details of both of the generated datasets is presented in Table 4.4. The columns in this table report averages values for each size: size of the instance i.e. the number of men/women (**n**), number of non-fixed men (**nf**), number of rotations in the rotation poset ($|\boldsymbol{\mathcal{V}}|$). By looking at these generated instances, we can observe that there are many men that have at least two stable partners

Table 4.4: Details on the instances in DATA-S and DATA-L.

n	**nf**	$\|\boldsymbol{\mathcal{V}}\|$	**n**	**nf**	$\|\boldsymbol{\mathcal{V}}\|$
50	33.40	11.36	700	651.32	120.88
100	75.28	21.40	800	742.60	127.28
150	121.44	33.64	900	849.20	143.36
200	166.28	40.88	1000	948.24	158.00
250	219.52	53.92	1100	1045.20	172.20
300	257.80	58.20	1200	1140.08	178.88
400	356.36	77.12	1300	1241.72	194.16
500	455.56	91.20	1400	1340.00	202.76
600	554.36	104.44			

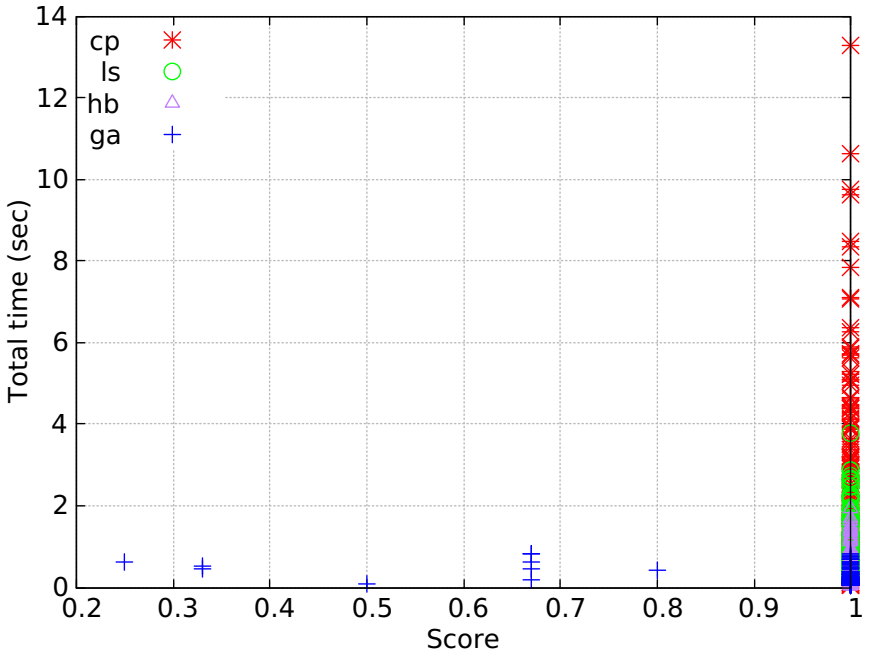


Figure 4.4: Search efficiency on the instances in DATA-S.

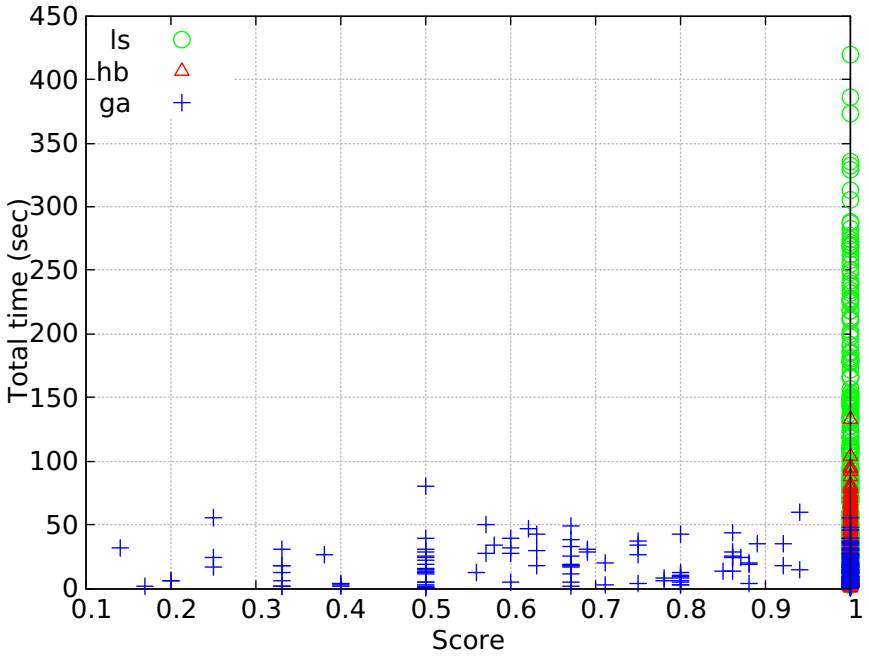


Figure 4.5: Search efficiency on the instances in DATA-L.

(i.e. non-fixed). We can also observe that the rotation posets are not very large in terms of the number of rotations. Also note that, the average number of rotations in the rotation posets increases as the size n increases.

In Figures 4.4 and 4.5 we plot the normalized objective value of the best solution found by the search model $h \in \{\mathrm{CP}, \mathrm{GA}, \mathrm{LS}, \mathrm{HB}\}$ (x-axis) after a given

time (y-axis). Recall that, we repeated our tests for each model by using three different randomization seeds.

Let $h(I)$ be the objective value of the best solution found using model h on instance I and $lb(I)$ (resp. $ub(I)$) the lowest (resp. highest) objective value found by any model on I. The formula below gives a normalized score in the interval $[0, 1]$:

$$score(h, I) = \frac{ub(I) - h(I) + 1}{ub(I) - lb(I) + 1}$$

This score function has previously been used by Hebrard and Siala [HS17]. The value of $score(h, I)$ is equal to 1 if h has found the best solution for this instance among all models. It decreases as $h(I)$ gets further from the optimal objective value, and is equal to 0 if and only if h did not find any solution for I. Notice that 1 is added both to the numerator and the denominator to avoid division by 0 in case the upper bound and the lower bound are equal. The plots show how far the models are from finding the optimal solution and also how long they take to find a solution. If the model h finds the lowest b value among all other models, then the score evaluates to 1, meaning that the model h is the best one. Note that the CP model runs out of memory for large instances. Therefore, we do not plot it in Figure 4.5.

The outcome from both figures is clear. In the first plot, the best solutions are found by CP, LS and HB. All the solutions are found as optimal by the CP model. However, CP takes much longer time when compared to the meta-heuristic models. Our GA model does not seem to be well suited for this problem. It fails to find the best solutions in many instances. Additionally, it is the only model that fails to find the optimal solutions. It can be observed in Figure 4.4 that GA takes the least time when compared to the other models. This is due to GA being stuck at a local minima, failing to produce different stable matchings. The procedure of the GA is then terminated due to the lim_{iter} criterion. Our remark on the GA model is that, a different crossover or mutation operation can help the GA perform better.

The LS and HB are very efficient both in the quality of the solutions they provide, and also the total time. However, our HB model outperforms LS in terms of the time efficiency. Observing the poor performance of GA and the good performance of HB, we can conclude that neighbourhood search adds GA the

required diversity and effectively helps it getting stuck at the local minimas. The results in Figure 4.5 are again in favour of the hybrid algorithm. Both LS and HB models always find the best solutions as all the instances have a score equal to 1. The GA model again fails to find the best solutions in many instances, and terminates early. We present this information in Figure 4.6 and Figure 4.7 running times of each model. We did not run the CP model for sizes $n > 300$ due to space limitations.

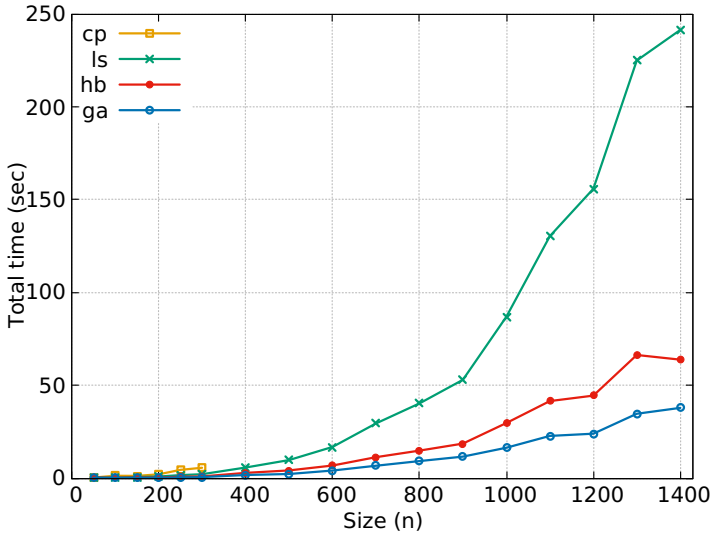


Figure 4.6: Average total time spent by each model on all instances (i.e. DATA-S merged with DATA-L).

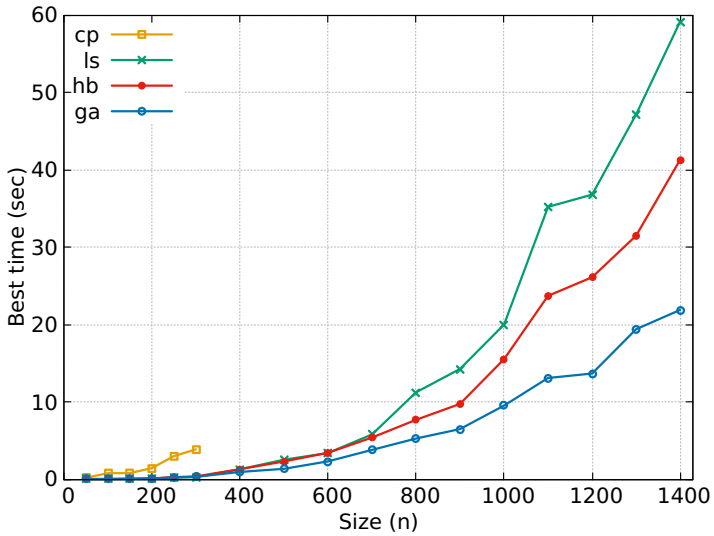


Figure 4.7: Average time spent to find the best solution by each model on all instances (i.e. DATA-S merged with DATA-L).

Figure 4.6 and Figure 4.7 report the average values of the total time and the best time each model takes for search for each size n, respectively. The best time is measured as the time spent from the beginning of the search until the solution is found. The total time, on the other hand, is measured as the time between the beginning of the search and the termination.

In both of these figures, we can see the same trend. Our GA model finds a solution at the early stages of the algorithm and terminates early as it cannot improve the solution it finds. For the small instances, when $50 \leq n \leq 300$, we can observe that the CP model takes more time to find the best solution when compared to the other models, and it also takes longer to terminate. We know from Figures 4.4 and 4.5 that both LS and HB find the same values of average minimum b. Consequently, in Figure 4.6 and Figure 4.7, we can observe that HB finds the solution quicker than LS, and also terminates earlier. Our HB model takes longer time than the GA model, and less than LS. Additionally, it finds the optimal solution. This demonstrates that the hybrid approach enhances the performance of the meta-heuristic models on random instances.

Additionally, Figure 4.8 plots the average number of different stable matchings created by each model until a solution is found. We can observe that the GA procedure is not able to explore many different stable matchings, when the values are compared to the other two models. The LS model explores many different stable matchings. However, the HB converges faster than the LS, when both total time and the number of stable matchings are considered.

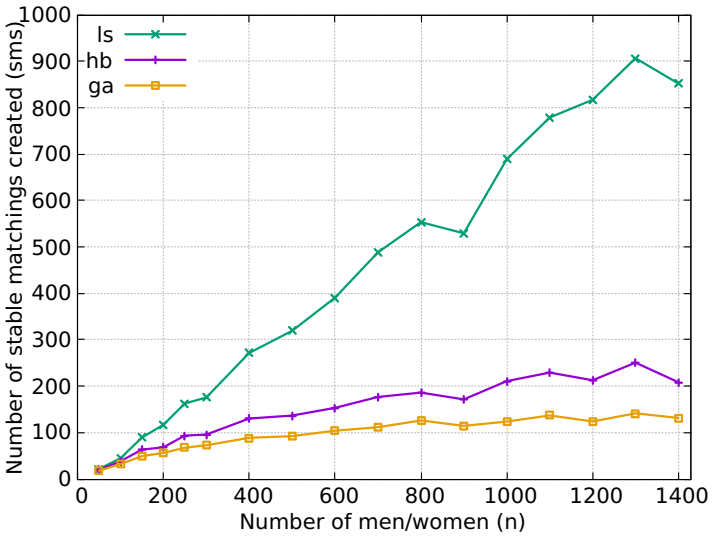


Figure 4.8: Average number of different stable matchings found by each model on all instances (i.e. DATA-S merged with DATA-L).

We do not provide a plot for the average minimum b values found by each model. Because, their values are very similar. Therefore, the lines overlap. However, we provide Table 4.5 in order to show how much they differ in terms of finding a good solution. Note that, this table provides a different metric than the comparisons given in Figure 4.4 and Figure 4.5. In those two figures, we take into account the different seeds used for each model and we report the minimum and maximum values found during different runs. However, in Table 4.5, we report the average minimum value b found by all three different runs. The columns represent in order the average value of number of men/-women in the instance $(\mathbf{n})$, the average minimum value b found by the model: GA (**GA-b**), LS (**LS-b**), HB (**HB-b**), and CP (**CP-b**). We coloured the cells that contain the best values to see the difference easier. Table 4.5 shows us that although HB can find better results when individual runs are considered, on the average, LS finds more robust solutions (i.e. lower values of b).

We can observe that when working on small uniformly random SM instances, LS and HB are very competitive. There is no clear advantage on one another. Although HB performs slightly better in terms of time efficiency, both models can be preferred. For larger random instances, especially after $n > 800$, the HB model has a clear advantage in terms of the total time spent over the LS. However, the b values found by HB are not as consistent as the LS. If individual

Table 4.5: The average minimum b values found by each model on all instances (i.e. DATA-S merged with DATA-L).

n	GA - b	LS - b	HB - b	CP - b
50	19.31	19.24	19.24	19.24
100	48.92	48.84	48.84	48.84
150	83.72	83.60	83.60	83.60
200	114.79	114.52	114.52	114.52
250	161.03	160.52	160.52	160.52
300	189.11	188.56	188.56	188.56
400	267.83	267.16	267.16	-
500	349.25	347.96	347.96	-
600	431.43	430.20	430.20	-
700	511.11	510.00	510.00	-
800	582.71	581.12	581.15	-
900	683.92	682.24	682.25	-
1000	769.05	766.68	766.69	-
1100	857.91	855.96	856.09	-
1200	931.21	928.08	928.25	-
1300	1028.47	1025.28	1025.48	-
1400	1110.55	1106.72	1106.733	

runs are considered, HB is able to find better values of b; but if the average is reported, then LS provides lower values. Note that, the difference for the average values is very small and can be neglected in some cases (i.e. $difference \leq 0.1$).

4.7.2 Large Instances (MANY)

After observing the competitive results of the HB and the LS models, we compared their performance on a dataset of SM instances, where the instances contain many stable matchings. We refer to this dataset as MANY. We often say that the instances in this dataset are "large", where the term *large* for this dataset stands for the number of stable matchings of an instance and not the number of men/women involved in the instance.

MANY. In order to generate instances that contain many stable matchings, we first generated 100 SM instances for each size $n = \{16, 32, 64, 128\}$ using the family described by Irving and Leather [IL86a]. Then, we slightly modified the generated instances, similar to the technique used by Siala and O'Sullivan [SO17]. Our dataset MANY consists of these modified instances.

First let us introduce this family of instances described by Irving and Leather. Irving and Leather prove that any instance in the original family contains at least 2^{n-1} stable matchings for an instance of size $n = 2^i$. They define the family over two matrices for the preferences of each gender, mp_n, mp'_n for the mens' preferences and wp_n, wp'_n for the womens'. The preference lists of these large instances are obtained recursively by appending the following matrices until the desired instance size is found:

$$mp_{2n} = \begin{bmatrix} mp_n & mp'_n \\ mp'_n & mp_n \end{bmatrix}, wp_{2n} = \begin{bmatrix} wp_n & wp'_n \\ wp'_n & wp_n \end{bmatrix}$$

For our experiments, we slightly modify each instance of this family by first randomly selecting two random men m_i, m_j. Then, we modify m_i's preference list by swapping the positions of two randomly selected women within the list. We repeat the same for m_j. We also modify the preference lists of two random women in the same way. As a result, for each instance in this dataset, we modify the original preference lists of men by changing 4 women's positions. Similarly, we modify the original women's preference table by changing 4 men's

positions. In other words, the original preference set between the original and the modified instances have a Hamming Distance of 8.

Example. We give in Table 4.6 an example of an original instance. Subsequently, in Table 4.7, we provide our modified version of this instance. The different positions in both men's and women's preference lists are marked as bold to make it easier to see. Observe that, the preference lists of men m_2 and m_8 are modified. Additionally, the preference lists of women w_2 and w_6 are modified by making small changes.

We also plot the rotation posets corresponding these two instances in order to provide an insight on the structures. Figure 4.9 illustrates the rotation posets of the instances presented in Table 4.6 (left) and Table 4.7 (right). We can observe that the technique described above may significantly decrease the number of rotations in the rotation poset for some cases. However, as we show later, these instances yield in a large number of stable matchings.

Before discussing the results on these instances, observe that, the original instances have the properties of the specific family of SM instances discussed in

Table 4.6: An SM instance of size 8 that belongs to the original family described by Irving and Leather [IL86a].

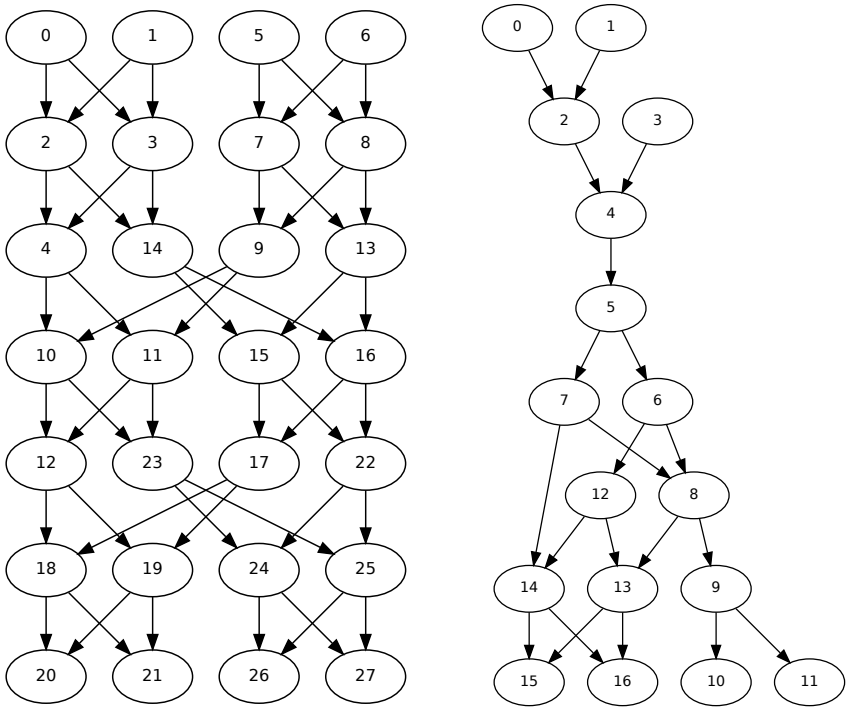
	Preference lists of men							
m_1	1	2	3	4	5	6	7	8
m_2	2	1	4	3	6	5	8	7
m_3	3	4	1	2	7	8	5	6
m_4	4	3	2	1	8	7	6	5
m_5	5	6	7	8	1	2	3	4
m_6	6	5	8	7	2	1	4	3
m_7	7	8	5	6	3	4	1	2
m_8	8	7	6	5	4	3	2	1

	Preference lists of women							
w_1	8	7	6	5	4	3	2	1
w_2	7	8	5	6	3	4	1	2
w_3	6	5	8	7	2	1	4	3
w_4	5	6	7	8	1	2	3	4
w_5	4	3	2	1	8	7	6	5
w_6	3	4	1	2	7	8	5	6
w_7	2	1	4	3	6	5	8	7
w_8	1	2	3	4	5	6	7	8

Table 4.7: An SM instance of size 8 that belongs to our benchmark MANY obtained by the original instance given in Table 4.6.

	Preference lists of men							
m_1	1	2	3	4	5	6	7	8
m_2	2	**8**	4	3	6	5	**1**	7
m_3	3	4	1	2	7	8	5	6
m_4	4	3	2	1	8	7	6	5
m_5	5	6	7	8	1	2	3	4
m_6	6	5	8	7	2	1	4	3
m_7	7	8	5	6	3	4	1	2
m_8	8	7	**5**	**6**	4	3	2	1

	Preference lists of women							
w_1	8	7	6	5	4	3	2	1
w_2	7	8	5	**4**	3	**6**	1	2
w_3	6	5	8	7	2	1	4	3
w_4	5	6	7	8	1	2	3	4
w_5	4	3	2	1	8	7	6	5
w_6	3	4	1	2	**5**	8	**7**	6
w_7	2	1	4	3	6	5	8	7
w_8	1	2	3	4	5	6	7	8



(a) Instance given in Table 4.6. (b) Instance given in Table 4.7.

Figure 4.9: Rotation posets corresponding to the large instances.

Definition 17 (Page 87). An interesting observation is that, although the original instances have exponentially many number of stable matchings, a $(1,1)$-supermatch to a given instance can easily be found in polynomial-time using the procedure discussed in the proof of Theorem 7 (Page 88).

For the sake of an example, consider the fact that there are 8 men/women in the original instance. We know that a rotation contains at least two pairs, and incomparable rotations contain different set of men (see Lemma 13, Page 99). Each rotation in this poset is incomparable with at least 3 rotations, where all 4 rotations are in the same level. Therefore, all rotations in this poset contain exactly 2 pairs. The first level in this rotation poset can be identified as consisting of $\rho_0, \rho_1, \rho_5, \rho_6$. By Theorem 7, this level corresponds to a $(1,1)$-supermatch.

We now present the results that we obtained on the robustness of these modified instances. Table 4.8 reports for each size the average value of: number of all men/women (**n**), the number of non-fixed men(**nf**), the number of rotations in the rotation poset ($|\mathcal{V}|$). Additionally, it reports the average minimum b found by the model LS, HB where population size $|\boldsymbol{P}| = 10$, and HB where population size $|\boldsymbol{P}| = 60$ (**b**); followed by the total time spent in minutes for each of the three models (**t (min)**).

The first observation on this table is about the number of non-fixed men (nf). It shows that by slightly modifying the original large instances, we can obtain SM instances in which almost everyone have at least one alternative partner. The reason nf having values 15.99 and 31.99 for sizes $n = 16$ and $n = 32$ is because one of the instances among 100 instances in each set in MANY both produce 1 fixed-man. On the other hand, when the instance size is 64 or 128, all men have at least two stable partners.

Next, this dataset shows that the robustness of instances that have many stable matchings is very high (i.e the value of b is low). For each instance size, our best model for that size is able to find solutions whose average b values evaluate to a b value that is $opt = 1 < b < 2$. For instance, for size $n = 16$, the LS model finds solutions such that for the breakage of any man on the solution, on the average, 1.12 other men need to break-up from their current partners. Similarly, for size $n = 128$, HB models finds that the solution is guaranteed to be repaired by only 1.02 additional men's break-up.

As one can observe from Table 4.8, we ran the HB model by using different sizes of population. The reason to this should become more clear later in this section, when we discuss the running times of the models. In Table 4.8, we can observe that reducing the number of individuals in the population of HB (10 vs. 60) causes the algorithm to find slightly worse solutions. For instance, for size $n = 64$, the average minimum b is found as 1.74 by a population of size 10, and 1.28 by a population of size 60. This is due to having an increased chance in getting stuck at local minima for a smaller population. On the other hand, LS finds competitive values for b for size $16 \leq n \leq 64$. However, as we can see for $n = 128$, LS finds solutions that are far away from the optimal solution.

We provide in Table 4.9 more details on the performances of these models. This table consists of two sections presented one below the other due to space limitation. Recall that, MANY contains 100 instances for each size n. The columns in the first half report for each size the value of: average number of all men (**n**),

Table 4.8: Summary of the results on large instances for RSM.

instance			**LS**		**HB, $\|P\| = 10$**		**HB, $\|P\| = 60$**	
n	**nf**	**$\|\mathcal{V}\|$**	**b**	**t (min)**	**b**	**t (min)**	**b**	**t (min)**
16	15.99	100.43	1.12	0.003	1.21	0.003	1.1	0.004
32	31.99	447.26	1.03	0.054	1.30	0.024	1.04	0.029
64	64	1889.95	1.685	3.158	1.74	0.824	1.28	0.916
128	128	7788.02	14.055	8.367	1.02	13.989	1.01	17.609

average of the number of different stable matchings created until the search terminates (**sm**), the maximum number of stable matchings created for a single instance among all 100 instances (**max-sm**), the time spent until the solution is found ($\boldsymbol{t_{best}}$ in minutes). Then, the columns in the second half report for each size the average value of: the number of instances where the value of b is found as 1 i.e. optimal (**opt**), the number of instances that terminated by a time-out i.e. reaching $lim_{time} = 20$ mins (**t/o**, the number of instances that terminated after not improving the solution for $lim_{iter} = 5000$ iterations (**no-im**).

Table 4.9 shows that HB outperforms the LS method on large instances. If we look at the time spent during LS, we can infer that a solution is found at the early stages of the algorithm (t_{best}) and the algorithm kept running until the cut-off limit or the time-limit is met. This is due to having large number of neighbours and also getting stuck at local minima. We can observe this by looking at the number of stable matchings produced in all methods. When $n = 128$, LS struggles finding the neighbours and can only explore 48 different stable matchings in total. However, HB models are able to explore more diverse stable matchings and hence find better solutions. We can observe that when $n = 128$ for LS model, on the average 32.5% of the instances terminate due to the time limit. However, nearly all the HB models either terminate because they could not find any improved solutions or they already found the optimal.

The results can be summed up as the LS model performs very well for the smaller instances in MANY. However, it struggles to complete the search for larger sizes. The HB model seems to perform well for the instances in MANY. The running time of the HB model can be improved by using a smaller population. But there is a trade-off between the total running time and the b values found.

Table 4.9: More details of the results on large instances for RSM.

	LS			**HB,** $\boldsymbol{\|P\| = 10}$			**HB,** $\boldsymbol{\|P\| = 60}$		
n	**sm**	**max-sm**	$\boldsymbol{t_{best}}$	**sm**	**max-sm**	$\boldsymbol{t_{best}}$	**sm**	**max-sm**	$\boldsymbol{t_{best}}$
16	88.61	672	0.001	34.76	215	0	90.75	822	0.001
32	352.1	7221	0.038	74.38	998	0.008	125.7	2720	0.016
64	677.1	7257	1.939	126.3	1857	0.323	155.8	2453	0.629
128	48.53	583	1.993	98.29	299	13.88	138.9	191	17.61
n	**opt**	**t/o**	**no-im**	**opt**	**t/o**	**no-im**	**opt**	**t/o**	**no-im**
16	93	0	7	90	0	10	93	0	7
32	99	0	1	93.5	0	6.5	98.5	0	1.5
64	92	8	0	95	0	5	98.5	1.5	0
128	67.5	32.5	0	99	1	0	99.5	0.5	0

4.8 Chapter Summary

We presented a polynomial-time procedure using the rotation poset to verify if a given stable matching is a $(1, b)$-supermatch. Subsequently, we developed four different models to find the most robust solution: a Constraint Programming, a local search, a genetic algorithm, and a hybrid model, all based on the proposed polynomial-time procedure. Then, we compared the four models and concluded that our hybrid procedure and the local search outperforms the genetic algorithm on random RSM instances. For small random instances, we verified that the local search and the hybrid model provide similar results to the constraint programming model. We compared the performances of local search and the hybrid models further on a dataset whose instances contain many stable matchings. We found out that although the LS model is successful for the smaller sizes in this dataset, HB performs better for larger sizes.

Chapter 5

Robust Stable Roommates

Abstract. We investigate the robustness concept further on a generalized version of the SM, namely the Stable Roommates problem (SR). We name the robust version of the problem as Robust Stable Roommates problem (RSR). We define a polynomial-time procedure based on the reduced rotation poset of the underlying SR instance to decide if a stable matching is a $(1, b)$-supermatch. Then, learning from the proposed models for RSM, we propose two meta-heuristic approaches for finding the most robust solution to an RSR instance: a local search procedure and a hybrid (genetic local search) procedure. We conclude this section by providing a comparison of the two models and an overview of the robustness of different RSM and RSR instances.

5.1 Introduction

The *Stable Roommates problem (SR)* consists of a set of $n = 2 \times k, k \in \mathbb{N}^+$ agents, where each agent has a preference list in which he/she ranks all other agents in strict order of preference. The aim is to find a matching that is stable. The basics of the problem is presented in Section 2.4.2 (Page 44).

We define the *Robust Stable Roommates problem (RSR)* analogous to the RSM (see Definition 12, Page 64). We refer the problem of finding an (a, b)-supermatch to a given SR instance as the *Robust Stable Roommates problem (RSR)*. In the RSM, given a stable matching M and a pair $(m_i, w_j) \in M$, when searching

for another stable matching M', where $(m_i, w_j) \notin M'$, we say that m_i wants to break-up. Similarly, in the RSR, given a stable matching M and a pair $\{p_i, p_j\} \in M$, when searching for another stable matching M', where $\{p_i, p_j\} \notin M'$, we say that $\{p_i, p_j\}$ wants to *leave* the M.

A stable matching of an RSR instance is called an (a, b)-*supermatch* if any a pairs do not want to be partners anymore (i.e. leave the stable matching), it is possible to find another stable matching by changing the partners of the people involved in those a pairs and at most b other pairs. Observe that, the definitions of the (a, b)-supermatches in the RSM (given in Definition 12, Page 64) and the RSR (see Definition 20) are very similar to each other. In both RSM and RSR, the value of a and b denote the number of pairs.

Definition 20 ((a,b)-supermatch) *Given an SR instance $\mathcal{I}$, and two integers $a, b \in \mathbb{N}$, a stable matching M of $\mathcal{I}$ is said to be an (a, b)-supermatch if for any set $\Psi \subseteq M$ of non-fixed stable pairs, where $|\Psi| = a$, there exists a stable matching M' such that $M' \cap \Psi = \emptyset$ and $d(M, M') \leq b + a$.*

Let us illustrate the notations and concepts introduced in this section on an SR instance of 10 people provided by Gusfield and Irving (can be found in Page 171 in [GI89]). Table 5.1 presents the preference table for this instance. Note that, to aid readability we sometimes denote the person by its index number (i.e. p_i as i, or pair $(\{p_i, p_j\})$ as $\{i, j\}$) etc.). This instance contains 7 stable matchings (later we list them in Table 5.6). Consider one of its stable matchings, namely $M_1 = \{\{p_1, p_3\}, \{p_2, p_4\}, \{p_5, p_7\}, \{p_6, p_8\}, \{p_9, p_{10}\}\}$ to illustrate an (a, b)-supermatch. In order M_1 to be a $(2, 0)$-supermatch, there must exist a stable matching in the underlying instance if any two pairs leave M_1. In other words, for the leave of $\{p_1, p_3\}$ and $\{p_2, p_4\}$, there exists a stable matching M'

Table 5.1: The preference table of an SR instance of size 10.

p_i	Preference list of p_i
1	8 2 9 3 6 4 5 7 10
2	4 3 8 9 5 1 10 6 7
3	5 6 8 2 1 7 10 4 9
4	10 7 9 3 1 6 2 5 8
5	7 4 10 8 2 6 3 1 9
6	2 8 7 3 4 10 1 5 9
7	2 1 8 3 5 10 4 6 9
8	10 4 2 5 6 7 1 3 9
9	6 7 2 5 10 3 4 8 1
10	3 1 6 5 2 9 8 4 7

where $M' \cap M_1 = \{\{p_5, p_7\}, \{p_6, p_8\}, \{p_9, p_{10}\}\}$. Similarly, an M^* for the leave of $\{p_1, p_3\}$ and $\{p_5, p_7\}$, where $M^* \cap M_1 = \{\{p_2, p_4\}, \{p_6, p_8\}, \{p_9, p_{10}\}\}$; an M'' where $M'' \cap M_1 = \{\{p_2, p_4\}, \{p_5, p_7\}, \{p_9, p_{10}\}\}$, for the leave of $\{p_1, p_3\}, \{p_6, p_8\}$, etc. This must hold for each pair combination of size 2.

On the other hand, in order $M_1 = \{\{p_1, p_3\}, \{p_2, p_4\}, \{p_5, p_7\}, \{p_6, p_8\}, \{p_9, p_{10}\}\}$ to be a $(2, 1)$-supermatch, for the leave of each combination of size 2, there must be a stable matching M' where the people involved in those 2 pair have different partners. Additionally, in any such M', either all other pairs remain the same as in M_1, or there is one more pair where the two people forming the pair also have different partners in M'. As an example, if $\{p_1, p_3\}$ and $\{p_2, p_4\}$ together want to leave M_1, then there is a stable matching M', where $\{\{p_1, p_3\}, \{p_2, p_4\}\} \cap M' = \emptyset$, and:

- $M' \cap M_1 = \{\{p_5, p_7\}, \{p_6, p_8\}, \{p_9, p_{10}\}\}$, or
- $M' \cap M_1 = \{\{p_6, p_8\}, \{p_9, p_{10}\}\}$, or
- $M' \cap M_1 = \{\{p_5, p_7\}, \{p_9, p_{10}\}\}$, or
- $M' \cap M_1 = \{\{p_5, p_7\}, \{p_6, p_8\}\}$.

Note that, there must be a stable matching M' as illustrated above for the leave of all 10 different pair combinations of size 2.

5.2 Notation and Definition

We define some terminology for the RSR similar to the ones in the RSM. We measure the distance between two stable matchings M and M' in an RSR instance by the number of pairs that exist in one, but not the other one, and denote it by $d(M, M')$. Let M be a stable matching and $\Psi \subseteq M$ be a set of non-fixed pairs to leave M. A *repair matching* is defined analogously to the RSM. A repair matching represents a stable matching, where for the leaves of Ψ from M is a stable matching M' that minimizes the value of $d(M, M^*)$ taken over every other stable matching M^* such that $M^* \cap \Psi = \emptyset$. The *repair cost* in this context is the value $d(M, M') - |\Psi|$. Additionally, in order to avoid repetition, as we did for the RSM, we use a notation for stable matchings and their corresponding closed subsets such that if a stable matching is identified using some superscripts or subscripts, then its corresponding closed subset contains them as well (i.e. the closed subset of M_i^j is denoted by S_i^j).

Recall the basics of SR. Each stable matching has a corresponding unique complete closed subset. A complete closed subset S represents a set of rotations in the reduced rotation poset such that for each $\rho \in S$, ρ is a non-singular rotation and all the predecessors of ρ are also in S. There exists a one-to-one correspondence between the complete closed subsets of the reduced rotation poset and the stable matchings of the underlying instance. The preference table obtained after applying the first phase of the SR algorithm is called T_0. We denote by $\boldsymbol{T_S}$ the preference table obtained after eliminating all singular rotations starting from T_0. Any stable matching can be obtained by eliminating one of each dual rotations starting from T_S. We shall sometimes abuse the notation and shorten the term preference table to table.

We define elimination and production of pairs for RSR with respect to the preference tables. A rotation ρ is said to ***eliminate*** $\{p_i, p_j\}$ if there exists a table T such that ρ is exposed in T and when ρ is eliminated from T, the resulting table does not contain $\{p_i, p_j\}$. More formally, ρ eliminates $\{p_i, p_j\}$ if ρ is exposed in T, $\{p_i, p_j\} \in T$, and $\{p_i, p_j\} \notin T/\rho$. On the other hand, a rotation ρ is said to ***produce*** $\{p_i, p_j\}$ if there exists a table T such that:

- ρ is exposed in T, and
- Preference lists of p_i or p_j contains more than 1 person (i.e. $|L_T(i)| > 1$, $|L_T(j)| > 1$), and
- Preference lists of p_i contains only p_j in the table obtained after eliminating ρ from T (i.e. $|L_{T/\rho}(i)| = 1$ and $L_{T/\rho}(i) = \{p_j\}$), and
- Preference lists of p_j contains only p_i in the table obtained after eliminating ρ from T (i.e. $|L_{T/\rho}(j)| = 1$ and $L_{T/\rho}(j) = \{p_i\}$).

We use the term ***flipping ρ*** from S as the process of removing $\rho \in S$ from S and adding its dual $\bar{\rho}$ to S. We define a set of sink rotations and a set of neighbour rotations analogous to the definitions in RSM. Given a complete closed subset S, the set **L(**S**)** denote the set of rotations that are the sink vertices of the graph induced by the rotations in S and the **N(**S**)** denote the set of the rotations that are not included in S and for each $\rho \in$ **N(**S**)** either $d_{in}(\rho) = 0$ or for all $\rho' \prec \rho$, $\rho' \in S$. We also use $\boldsymbol{L_T(i)}$ to denote the list of the person p_i in a stable table T.

In the rest of this section we use an instance $\mathcal{I}$ that has more than one solution. Because, some SR instances do not have a solution or they only have a single solution. Therefore, these instances do not provide any information on the

robustness. In fact, they are not robust with respect to our robustness definition. Therefore, we work on instances that contain at least two stable matchings. We also use $\{p_i, p_j\}$ to denote a non-fixed pair of $\mathcal{I}$ that is stable. Recall that, by Lemma 3 (Page 49), $\{p_i, p_j\}$ is a stable non-fixed pair if and only if (p_i, p_j) or (p_j, p_i) is in a non-singular rotation.

The intractability result of the RSM is lifted to the RSR as the SR is a generalization of the SM.

Theorem 10 *RSR is $\mathcal{NP}$-hard.*

Proof. The proof is straightforward as it is possible to create an SR instance from any given SM instance with the exact same stable matchings in polynomial-time by padding every other person of the same sex to the preference list of each person (see Lemma 1, Page 45). Every (a, b)-supermatch in the SM instance is also an (a, b)-supermatch in the corresponding SR instance and vice versa. Hence, RSR is $\mathcal{NP}$-hard because RSM is $\mathcal{NP}$-hard. □

5.3 Verification of (1,b)-supermatches

Although the outlines of the approaches for finding a $(1, b)$-supermatch for RSR and RSM are similar, there is one major difference to be considered. The RSM instances contain at most one rotation that produces a non-fixed stable pair and at most one other that eliminates it. In this section, we show that it is different for the RSR. We prove in the rest of the section that the RSR has two cases: a pair can be produced by a unique rotation and eliminated by another one, or it can be eliminated by two different rotations and produced by two others. Throughout this section, we denote by ρ_e the elimination rotation, and by ρ_p the production rotation. If there is more than one rotation that eliminates/produces the pair, we use $\rho_{p1}, \rho_{p2}, \rho_{e1}, \rho_{e2}$, etc. to identify them.

We begin by proving the existence and identification of these rotations for each pair. First, let us identify some cases to aid readability of this section. For any non-fixed stable pair $\{p_i, p_j\}$, we identify two cases by the position of the person in the other one's preference list in preference table T_S as follows:

Case 1: There are two sub-cases:

- $f_{T_S}(i) = p_j$ and $l_{T_S}(j) = p_i$, or

– $l_{T_S}(i) = p_j$ and $f_{T_S}(j) = p_i$.

Case 2: Otherwise.

Case 1 is a special case that applies if one of the persons in the pair is the other one's most preferred person in T_S (respectively, the other one is the least preferred person in T_S). Note that, in both of the cases, the two persons have more than 1 preferences in their preference lists, i.e. $L_{T_S}(i) > 1$ and $L_{T_S}(j) > 1$. Because the pairs are non-fixed. Later on, we refer to these cases for identifying scenarios.

Example. Let us illustrate the notation and concepts introduced in this section on the example instance given in Table 5.1. We first apply the first phase of the SR algorithm (see Algorithm 4, Page 48) on Table 5.1. We denote by T_0 the table obtained at this stage. Then, we remove all singular rotations from T_0 (see Algorithm 5, Page 48) and obtain T_S. Table 5.2 presents the T_S of the instance presented in Table 5.1. For this instance, we can identify the following potential stable pairs. We also show to which case the identified pairs belong.

Set of pairs that are identified as Case 1: $\{\{p_1, p_3\}, \{p_2, p_4\}, \{p_3, p_5\}, \{p_4, p_9\}, \{p_5, p_7\}, \{p_6, p_8\}, \{p_7, p_1\}, \{p_8, p_{10}\}, \{p_9, p_2\}, \{p_{10}, p_6\}\}$.

Set of pairs that are identified as Case 2: $\{\{p_1, p_4\}, \{p_2, p_3\}, \{p_2, p_8\}, \{p_3, p_6\}, \{p_3, p_2\}, \{p_4, p_1\}, \{p_4, p_6\}, \{p_5, p_{10}\}, \{p_5, p_8\}\}$.

Table 5.2: The table T_S for the SR instance of size 10 presented in Table 5.1.

p_i	Preference list of p_i
1	3 4 7
2	4 3 8 9
3	5 6 2 1
4	9 1 6 2
5	7 10 8 3
6	8 3 4 10
7	1 5
8	10 2 5 6
9	2 10 4
10	6 5 9 8

5.3.1 Identification of Elimination and Production Rotations

In this section, we show the existence of the elimination and production rotations for each pair in a given RSR instance. We identify these rotations now, and use later when describing our methodology for verifying if a given stable matching is a $(1, b)$-supermatch in Section 5.3.2.

First, in Lemma 19, we show how to identify the elimination rotation(s) for a given pair regardless of its case. Note that, the elimination rotations for RSR are defined similar to the ones in RSM, except the pairs in RSM are ordered but the ones in RSR are unordered.

Lemma 19 *A non-fixed stable pair $\{p_i, p_j\}$ is eliminated by a rotation ρ if and only if $(p_i, p_j) \in \rho$ or $(p_j, p_i) \in \rho$.*

Proof. $\Rightarrow$ Let the rotation be defined as $\rho = (x_0, y_0), (x_1, y_1) \ldots, (x_{|\rho|-1}, y_{|\rho|-1})$. Let us remind a few concepts first. We know that every stable pair $\{p_i, p_j\}$ in the SR is included in a non-singular rotation ρ as either $(p_i, p_j) \in \rho$ or $(p_j, p_i) \in \rho$ (by Lemma 3, Page 49). Recall that, eliminating ρ from a table also results in deleting all pairs $\{y_m, z\}$ such that y_m prefers x_{m-1} to z in the table (see Section 2.4.2, Page 49). We first show that our non-fixed stable pair $\{p_i, p_j\}$ cannot be one of such $\{y_m, z\}$. In other words, we show that a rotation that does not include the pair cannot eliminate it.

Suppose for contradiction that a non-fixed stable pair $(p_i, p_j) \notin \rho$ but ρ eliminates it by the deletion of a pair $(x_m, y_m) \in \rho$, where $m \in [0, |\rho| - 1]$. From the definition of a complete closed subset, all the stable matchings contain either the non-singular rotation $\rho = \ldots, (x_{m-1}, y_{m-1}), (x_m, y_m), (x_{m+1}, y_{m+1}), \ldots$ or its dual $\bar{\rho} = \ldots, (y_m, x_{m-1}), (y_{m+1}, x_m), \ldots$, (note that $+1$ operation is modulo $|\rho|$) in their corresponding complete closed subsets, denoted by S. Consider the scenario plotted in Table 5.3. This is an illustration (not complete) of the consequence of deleting pairs in the rotation.

Table 5.3: An illustration of a table T.

p	Preference list of p
$\ldots$	$\ldots$
x_m	$\ldots, y_m, (z), y_{m+1} \ldots$
$\ldots$	$\ldots$
y_m	$\ldots, x_{m-1}, (z), x_m, \ldots$
$\ldots$	$\ldots$

For instance, eliminating ρ moves x_m from y_m to y_{m+1} by deleting $\{y_m, z\}$. In a similar way, eliminating $\bar{\rho}$ moves y_m from x_{m-1} to x_m by deleting such $\{x_m, z\}$. The pairs eliminated in this process cannot be exposed in any stable tables later on. Therefore, they cannot be a part of a rotation, meaning the pair is not stable. This contradicts the fact that pair $\{p_i, p_j\}$ is a non-fixed *stable* pair. Therefore, $\{p_i, p_j\}$ cannot be any such $\{x_m, z\}$ or $\{y_m, z\}$ The proof is the same for (p_j, p_i).

$\Leftarrow$ From the definition of eliminating a rotation ρ from T , where $(p_i, p_j) \in \rho$, the elimination results in the deletion of p_j from p_i's list. Similarly, if $(p_j, p_i) \in \rho$ then it results in the deletion of p_i from p_j's list .

Hence, for any non-fixed stable pair $\{p_i, p_j\}$ to be eliminated, there exists a non-singular rotation ρ and the pair appears in this rotation as either $(p_i, p_j) \in \rho$ or $(p_j, p_i) \in \rho$. □

Subsequently, Lemma 20 identifies the production rotation(s) for each pair.

Lemma 20 *If a non-fixed stable pair $\{p_i, p_j\}$ is eliminated by ρ_e, then $\{p_i, p_j\}$ is produced by its dual, $\rho_p = \bar{\rho_e}$.*

Proof. A rotation is said to produce $\{p_i, p_j\}$ if eliminating it from a preference table T reduces $L_{T/\rho}(i)$ to a single entry, namely to p_j and $L_{T/\rho}(j)$ to p_i. We prove the existence of the production rotations over the two cases (Case 1 and Case 2) identified in Section 5.3.

We have two sub-cases in Case 1. First case is when p_j is p_i's most preferred person in T_S, i.e. $f_{T_S}(i) = p_j, l_{T_S}(j) = p_i$. In order to reduce p_i's list to only p_j, we need a rotation that moves p_i from his/her second best choice up to the first choice. We refer to this operation as ***limiting p_i from the right***. Similarly, to reduce the p_j's list to p_i, we need a rotation that moves p_j from his/her second least-preferred person to the least preferred person. We refer to this operation as ***limiting p_j from the left***. Referring back to Table 5.3 for notation, the production rotation ρ_p of the pair $\{p_i, p_j\} = (x_m, y_m)$ must contain the pair $(y_{m+1}, x_m) \in \rho_p$ to limit x_m from the right. Additionally, it must contain (y_m, x_{m-1}) to limit y_m from the left. To illustrate, the production rotation has the shape: $\rho_p = \ldots, (y_m, x_{m-1}), (y_{m+1}, x_m), \ldots$. Note that, each ordered pair can only appear in exactly one rotation. Observe that, the dual of ρ_p contains the pair (x_m, y_m), from definition of the dual. By Lemma 19, we know that the rotation that contains (x_m, y_m) is the elimination rotation of the pair $\{p_i, p_j\}$. Therefore, $\rho_p = \bar{\rho_e}$. The proof for the second sub-case is similar, where $(y_m, x_m) \in \rho_e$.

For a pair $\{p_i, p_j\}$ in Case 2, each person has both more and less preferred people in their lists. Therefore, in order to produce a pair, their lists must be limited from both the left and right. Let ρ_{p1} denote the rotation that limits p_i from the left and p_j from the right, and ρ_{p2} denote the rotation that limits p_i from the right and p_j from the left, respectively. Let the preference lists for the pair $\{p_i, p_j\}$ denoted by $L_{T_S}(i) = [\ldots, y_{m-1}, y_m, y_{m+1}]$ and $L_{T_S}(j) = [\ldots, x_{m-1}, x_m, x_{m+1}]$ where $\{p_i, p_j\} = (x_m, y_m)$. The pair (x_m, y_{m-1}) must be in ρ_{p1} to limit p_i from the left and (x_{m+1}, y_m) be in ρ_{p1} to limit p_j from the right. Additionally, the pair (y_{m+1}, x_m) must be in ρ_{p2} to limit p_i from right and (y_m, x_{m-1}) to limit p_j from left. Note that, the dual of ρ_{p1} contains (y_m, x_m), the dual of ρ_{p2} contains (x_m, y_m), from the definition of a dual rotation. By Lemma 19, we know these rotations are elimination rotations of the pair $\{p_i, p_j\}$.

Note that the two rotations ρ_{p1} and ρ_{p2} do not require one of them to be eliminated from the table first; they are incomparable. Therefore, depending on the order of elimination of the rotations, both of them are identified as production rotations. □

Having identified the production rotation for a pair, Proposition 4 gives a characterisation on production rotation(s) of a pair $\{p_i, p_j\}$ and the complete closed subsets of the stable matchings that $\{p_i, p_j\}$ belong.

Proposition 4 *For each non-fixed stable pair $\{p_i, p_j\}$ in a stable matching M, the corresponding complete closed subset of M contains all production rotations of $\{p_i, p_j\}$.*

Proof. The proof is immediate from the proof of Lemma 20 as eliminating the production rotations starting from preference table T_S guarantees to reduce the entries in final table (solution) T to $L_T(i) = p_j$ and $L_T(j) = p_i$. □

We sum up the findings above for the non-fixed stable pairs. If a pair is described by Case 1, then there exists only one elimination rotation for this pair and only one production rotation as the dual of the elimination one. Because, the preference list needs to be limited from only one direction (limiting from the left or from the right depending on the sub-case). However, for the pairs described by Case 2, there exist two elimination rotations for this pair, and also two other production rotations. Note that, for the pairs of Case 2, including one production rotation in the complete closed subset and not the other one, results in producing another partner for that pair. If the second production rotation is

not added, its dual must be added to the complete closed subset. Therefore, the pair produced depends on the dual rotation. Subsequently, Theorem 11 is an immediate result of Lemma 19 and Lemma 20.

Theorem 11 *For a non-fixed stable pair $\{p_i, p_j\}$, if:*

> ***Case 1:*** *There exists a unique elimination rotation ρ_e, where $(p_i, p_j) \in \rho_e$ or $(p_j, p_i) \in \rho_e$, and a unique production rotation ρ_p, where $\rho_p = \bar{\rho_e}$.*
>
> ***Case 2:*** *There exist two different elimination rotations ρ_{e1}, ρ_{e2}, where $(p_i, p_j) \in \rho_{e1}, (p_j, p_i) \in \rho_{e2}$ and two rotations $\rho_{p1} = \bar{\rho_{e1}}, \rho_{p2} = \bar{\rho_{e2}}$ that produce the pair.*

Proof. Immediate from Lemma 19 and Lemma 20. □

Example. Recall the look-up example presented in Table 5.1, and its T_S table presented in Table 5.2. This instance contains 10 non-singular rotations as presented in Table 5.4.

For a given pair covered by Case 1, e.g. $\{p_6, p_8\}$, there exists a unique elimination rotation $\rho_e = \bar{\rho_7}$. Its production rotation is also unique and is the dual of the elimination rotation $\rho_p = \rho_7$. Similarly, for $\{p_5, p_3\}$, there exists a unique elimination rotation $\rho_e = \rho_4$. Its production rotation is also unique and is the dual of the elimination rotation $\rho_p = \bar{\rho_4}$. As a final example, for a pair covered by Case 2, e.g. $\{p_1, p_4\}$, there exist two different elimination rotations, $\rho_{e1} = \bar{\rho_3}$ and $\rho_{e2} = \bar{\rho_6}$, whereby including any of them in a closed subset results in a stable matching, where the persons involved in the pair have different partners than each other. Moreover, there exist two different rotations, $\rho_{p1} = \rho_3$ and $\rho_{p2} = \rho_6$, which when both included in a complete closed subset, the corresponding stable

Table 5.4: The list of non-singular rotations of the instance given in Table 5.2.

ρ_3	(1, 3), (2, 4)
$\bar{\rho_3}$	(4, 1), (3, 2)
ρ_4	(3, 5), (10, 6)
$\bar{\rho_4}$	(6, 3), (5, 10)
ρ_5	(8, 10), (9, 2)
$\bar{\rho_5}$	(2, 8), (10, 9)
ρ_6	(4, 9), (7, 1), (10, 5)
$\bar{\rho_6}$	(9, 10), (1, 4), (5, 7)
ρ_7	(3, 6), (8, 2)
$\bar{\rho_7}$	(2, 3), (6, 8)

matching contains the pair. Also, observe that, the pair $\{p_5, p_8\}$ is not a stable pair as none of the non-singular rotations contain it (see Lemma 3, Page 49).

5.3.2 Methodology

In the previous section, we identified for each pair the rotation(s) that produce and eliminate them. In this section, we show how to use the identified rotations to obtain the closest stable matching M' to a given stable matching M if one of the pairs wants to leave the stable matching at a time. Let S_P denote the set of all the complete closed subsets for the underlying SR instance. Lemma 21 gives a characterization for the complete closed subsets.

Lemma 21 *Let $S \in S_P$. For each sink rotation ρ of S, the set $S \setminus \{\rho\} \cup \{\bar{\rho}\} \in S_P$.*

Proof. From the definition of closed subset, all predecessors $\rho' \in N^-(\rho)$ are also in S. Since ρ is a sink rotation, successors $\rho^* \in N^+(\rho)$ are not in S. Additionally, from the definition of the complete closed subset, duals of all ρ^* must be in the complete closed subset (i.e. $\bar{\rho^*} \in S$) and $\bar{\rho}$ is not in S. Using Lemma 5 (Page 51), we know that $\bar{\rho^*} \prec \bar{\rho}$. Hence, all predecessors of $\bar{\rho}$ are already in S, making $\bar{\rho}$ a neighbour rotation and results in $S \setminus \{\rho\} \cup \{\bar{\rho}\} \in S_P$. □

The distance between two stable matchings $d(M, M')$ is previously defined in Section 5.2 as the number of different pairs between M and M'. Observe that the distance can be calculated by also using their corresponding complete closed subsets, instead of the stable matchings. If $S \setminus S' = \{\rho\}$, it means $\rho \in S$ and $\bar{\rho} \in S'$. We know that, $X(\{\rho\}) = Y(\{\bar{\rho}\})$ and $Y(\{\rho\}) = X(\{\bar{\rho}\})$. Therefore, between M and M', only the people in ρ (or $\bar{\rho}$) have different partners. This can also be generalised to a set of rotations. Hence, the distance can also be denoted as $d(S, S') = |X(S \setminus S') \cup Y(S \setminus S')|/2$. Note that $d(S', S) = d(S, S')$.

Lemma 22 identifies the closest stable matching to a stable matching M, when a rotation from its corresponding complete closed subset is to be removed.

Lemma 22 *Given a stable matching M and its corresponding complete closed subset S, if $\rho \in S$ is a rotation to remove from S, the closest stable matching M' to M such that $\rho \notin S'$ is found by the formula:*

$$C(S, \rho) = S' = (S \setminus (\{\rho\} \cup N^+(\rho))) \cup \{\bar{\rho}\} \cup \bigcup_{\rho^* \in N^+(\rho)} \bar{\rho^*}. \tag{5.1}$$

Proof. The fact that the set S' is a complete closed subset follows from Lemma 5 (Page 51) and Lemma 21 as flipping a sink rotation of S yields in another complete closed subset. However, if ρ is not a sink rotation in S, we must flip all the successors of ρ to obtain a complete closed subset.

Let M^* denote the stable matching after flipping $\rho \in S$. Then $d(M, M^*) = d(S, S^*) = |X(\{\rho\}) \cup Y(\{\rho\})|/2$. Now, let M^* denote the stable matching after flipping both $\rho, \sigma \in S$. Then, the two stable matchings differ only by the partners of all people that are involved in ρ and σ. Note that, we calculate the distance over the number of pairs in RSR. Therefore, the distance is: $d(M, M^*) = |X(\{\rho\}) \cup Y(\{\rho\}) \cup X(\{\sigma\}) \cup Y(\{\sigma\})|/2$. Observe that, flipping more rotations can only increase the distance between two stable matchings. In Formula 5.1, the required number of flips is the minimum, and therefore, the function $C(S, \rho)$ returns the closest stable matching to M when $\rho \in S$ to be removed from S. □

Finally, Theorem 12 concludes how to find the closest stable matching M' to M if $\{p_i, p_j\} \in M$ wants to leave the M.

Theorem 12 *Given a stable matching M and a pair $\{p_i, p_j\}$ to leave M, the closest stable matching M' to M is identified by its corresponding S' by using the Formula 5.1 as follows:*

1. *If Case 1, then $S' = C(S, \rho_p)$.*
2. *If Case 2, let M_1 and M_2 be the two stable matchings s.t. $S_1 = C(S, \rho_{p1})$ and $S_2 = C(S, \rho_{p2})$. Then $S' = S_1$ if $d(M, M_1) < d(M, M_2)$, otherwise $S' = M_2$.*

Proof. The proof is immediate from Theorem 11 and Lemma 22. □

In order to verify if a given stable matching M is a $(1, b)$-supermatch, all the closest stable matchings to the given stable matching are found under the assumption that each pair wants to leave the stable matching, one at a time. For each such pair, its production rotation is identified and then Theorem 12 is applied to find the closest stable matching. Among all the closest stable matchings, the stable matching M' that results in the maximum distance to M defines the robustness of M, i.e. $b = d(M, M') - 1$, where 1 denotes the pair that wants to leave.

Example. Figure 5.1 illustrates the reduced rotation poset of our running example presented in Table 5.1.

Table 5.5 presents a list of all the complete closed subsets of the reduced rotation poset given in Figure 5.1. Additionally, Table 5.6 presents a list of all the stable matchings corresponding to the complete closed subsets given in Table 5.5.

On this example, we demonstrate how to find the $(1, b)$-robustness value of the stable matching M_6. We choose M_6 because it contains a number of pairs of both Case 1 and Case 2. We start by identifying in M_6 in Table 5.7 the production

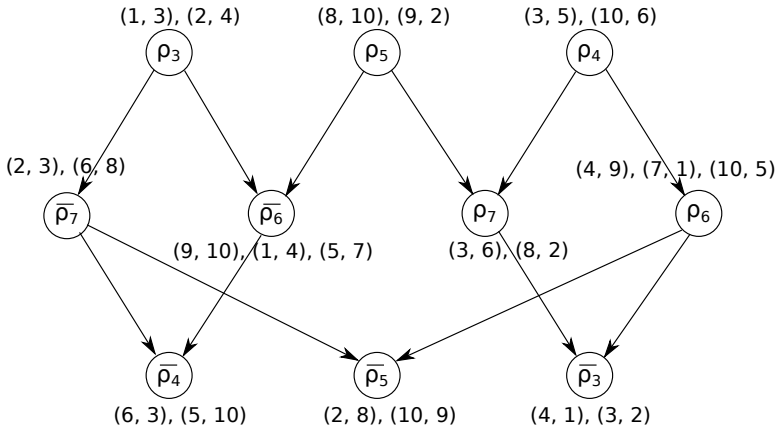


Figure 5.1: Reduced rotation poset of the rotations given in Table 5.4.

Table 5.5: A list of all the seven complete closed subsets of the poset given in Figure 5.1.

$$
\begin{aligned}
S_1 &= \{\bar{\rho_3}, \rho_4, \rho_5, \rho_6, \rho_7\} \\
S_2 &= \{\rho_3, \bar{\rho_4}, \rho_5, \bar{\rho_6}, \bar{\rho_7}\} \\
S_3 &= \{\rho_3, \rho_4, \bar{\rho_5}, \rho_6, \bar{\rho_7}\} \\
S_4 &= \{\rho_3, \rho_4, \rho_5, \rho_6, \rho_7\} \\
S_5 &= \{\rho_3, \rho_4, \rho_5, \rho_6, \bar{\rho_7}\} \\
S_6 &= \{\rho_3, \rho_4, \rho_5, \bar{\rho_6}, \rho_7\} \\
S_7 &= \{\rho_3, \rho_4, \rho_5, \bar{\rho_6}, \bar{\rho_7}\}
\end{aligned}
$$

Table 5.6: A list of all the stable matchings corresponding to the complete closed subsets given in Table 5.5.

$$
\begin{aligned}
M_1 &= \{\{p_1, p_3\}, \{p_2, p_4\}, \{p_5, p_7\}, \{p_6, p_8\}, \{p_9, p_{10}\}\} \\
M_2 &= \{\{p_1, p_7\}, \{p_2, p_8\}, \{p_3, p_5\}, \{p_4, p_9\}, \{p_6, p_{10}\}\} \\
M_3 &= \{\{p_1, p_4\}, \{p_2, p_9\}, \{p_3, p_6\}, \{p_5, p_7\}, \{p_8, p_{10}\}\} \\
M_4 &= \{\{p_1, p_4\}, \{p_2, p_3\}, \{p_5, p_7\}, \{p_6, p_8\}, \{p_9, p_{10}\}\} \\
M_5 &= \{\{p_1, p_4\}, \{p_2, p_8\}, \{p_3, p_6\}, \{p_5, p_7\}, \{p_9, p_{10}\}\} \\
M_6 &= \{\{p_1, p_7\}, \{p_2, p_3\}, \{p_4, p_9\}, \{p_5, p_{10}\}, \{p_6, p_8\}\} \\
M_7 &= \{\{p_1, p_7\}, \{p_2, p_8\}, \{p_3, p_6\}, \{p_4, p_9\}, \{p_5, p_{10}\}\}
\end{aligned}
$$

Table 5.7: All production and elimination rotations for each pair in M_6.

$\{p_i, p_j\}$	Case	Production (ρ_p)	Elimination (ρ_e)
$\{p_1, p_7\}$	1	$\rho_p = \bar{\rho}_6$	$\rho_e = \rho_6$
$\{p_2, p_3\}$	2	$\rho_{p1} = \rho_7$, $\rho_{p2} = \rho_3$	$\rho_{e1} = \bar{\rho}_7$, $\rho_{e2} = \bar{\rho}_3$
$\{p_4, p_9\}$	1	$\rho_p = \bar{\rho}_6$	$\rho_e = \rho_6$
$\{p_5, p_{10}\}$	2	$\rho_{p1} = \rho_4$, $\rho_{p2} = \bar{\rho}_6$	$\rho_{e1} = \bar{\rho}_4$, $\rho_{e2} = \rho_6$
$\{p_6, p_8\}$	1	$\rho_p = \rho_7$	$\rho_e = \bar{\rho}_7$

Table 5.8: All production and elimination rotations for each pair in M_6.

$\{p_i, p_j\}$	$C(S, \rho)$	S	$d(M, M')$	S'	b
$\{p_1, p_7\}$	$C(S, \bar{\rho}_6) = \{\rho_3, \rho_4, \rho_5, \rho_6, \rho_7\}$	S_4	4	S_4	3
$\{p_2, p_3\}$	$C(S, \rho_7) = \{\rho_3, \rho_4, \rho_5, \bar{\rho}_6, \bar{\rho}_7\}$	S_7	2	S_7	1
	$C(S, \rho_3) = \{\bar{\rho}_3, \rho_4, \rho_5, \rho_6, \rho_7\}$	S_1	4		
$\{p_4, p_9\}$	$C(S, \bar{\rho}_6) = \{\rho_3, \rho_4, \rho_5, \rho_6, \rho_7\}$	S_4	4	S_4	3
$\{p_5, p_{10}\}$	$C(S, \rho_4) = \{\rho_3, \bar{\rho}_4, \rho_5, \bar{\rho}_6, \bar{\rho}_7\}$	S_2	3	S_2	2
	$C(S, \rho_3) = \{\bar{\rho}_3, \rho_4, \rho_5, \rho_6, \rho_7\}$	S_1	4		
$\{p_6, p_8\}$	$C(S, \bar{\rho}_7) = \{\rho_3, \rho_4, \rho_5, \bar{\rho}_6, \bar{\rho}_7\}$	S_7	2	S_7	1

and the elimination rotations for each pair, including their cases.

After the identification of the production rotations are completed, we flip the production rotations by applying the formula given in the Equation 5.1 to find the closest stable matching in the case of each pair's removal. The complete closed subsets found after the formula for each pair is presented in Table 5.8. Note that, if the pair is covered by Case 2 and there are two different potentially closest stable matchings, the minimum distance one is selected as the closest.

Computing the partial b values (i.e. the repair cost for each pair removal), we conclude that, in the worst case, the maximum cost of repairing the leave of a pair for M_6 is 3. Therefore, M_6 is a $(1, 3)$-supermatch.

5.3.3 Complexity

The production and elimination rotations of each pair can be identified in a pre-processing step. We show that checking if a stable matching is a $(1, b)$-supermatch can be performed in $O(n \times |\mathcal{V}|)$ time after the $O(n^4)$ preprocessing step for an instance where $n = 2 \times k$ people are involved. We look into the details of these computations below.

First, we look at the details of the pre-processing step. Given an SR instance, the identification of the rotations and building the reduced rotation poset takes

$O(n^4)$ by our implementation, but can be reduced to $O(n^3 logn)$ (Section 2.4.2, Page 51). The identification of all the predecessors and successors of each rotation ρ takes $O(|\mathcal{V}|^2)$ time as we search the rotation poset for each rotation. Identifying elimination and production rotations for each pair $\{p_i, p_j\}$ whenever applicable is $O(n \times |\mathcal{V}|)$. Because there are $n/2$ pairs, and the number of rotations in a reduced rotation poset, $|\mathcal{V}|$, is $O(|n^2|)$. Thus, the overall pre-processing step takes $O(n^4)$ time.

The main algorithm is to compute for each pair in M, the closest stable matching M' by using the methodology presented in by Theorem 12. Finding the distance between two stable matchings is $O(n)$ as we find the different rotations between two complete closed subsets and count the people involved in them. Flipping a rotation takes constant time. The worst case of finding the closest stable matching is to flip all the non-singular rotations in S, where the number of all non-singular rotations is $|\mathcal{V}|/2$. Therefore, this computation takes $O(n \times |\mathcal{V}|)$ time.

We also discuss below some additional costs arise when converting an M to its corresponding S or vice-versa. Given a stable matching M, its corresponding complete closed subset S is found by finding and adding the production rotation(s) of each pair and their predecessors into set S, starting from an empty set $S = \{\}$. Considering that the predecessors and production rotations are found in the pre-processing step, they can be retrieved quickly by using an additional structure, e.g. a hash table. Adding all these rotations in the set then takes $O(|\mathcal{V}|)$. On the other hand, in order to find the M corresponding to an S, all the rotations in S are eliminated from T_S by respecting their precedence order. The order between the rotations in S can be found by applying a sorting algorithm, where the sorting algorithm can be implemented in $O(|\mathcal{V}| \times log|\mathcal{V}|)$. Note that, when compared to the operations in RSM, the RSR has a higher cost.

5.4 Models for Finding (1,b)-supermatches

In this section, we describe two meta-heuristic models that we use to find a $(1, b)$-supermatch to a given RSR instance: a local search approach (LS) and a hybrid of genetic algorithm and local search (HB). Our models are inspired from the models used for RSM defined in Chapter 4, and are based on the methodology described in Section 5.3.2. Our hybrid model and the local search

model are extensions of the RSM models discussed in Section 4.5 (Page 119), and Section 4.6 (Page 124). We did not use the genetic algorithm model for the RSR tests as it does not yield good results and is outperformed by the other two models. A CP model could be tailored for the RSR similar to the one explained in Section 4.3. However, we did not provide any such complete methods. Our experience is that, unless the RSR instance yields in a small rotation poset, the model is expected to run even slower than the RSM instances of similar size. In the remainder of this section, we describe the two models tailored to the RSR and then provide a comparison of them.

5.4.1 Local Search Approach

The local search (LS) approach developed for finding a $(1, b)$-supermatch to a given RSR instance is similar to the local search approach developed for RSM in Section 4.5. Considering the structural similarities between the RSM and the RSR, we tailored the LS model of RSM as it was shown that the LS model outperforms the genetic algorithm and produces near optimal solutions for the RSM, especially for the small instances.

We first describe the general procedure, discuss in detail how to create a random stable matching and how the neighbourhood of a stable matching is constructed. Recall that, in the LS model, there exists a neighbourhood $\boldsymbol{N}$ for the current solution. The algorithm works by searching the $\boldsymbol{N}$ of the current solution, finding the best neighbour M_n in the neighbourhood, and then descending the search by checking the neighbourhood of the current best solution. The search is restarted from a randomly created stable matching at every few iterations to avoid getting stuck at a local minima. The search continues until a termination criterion is met. In our model, we have three termination criteria:

1. Cut-off limit lim_{iter}, which indicates improvement of the best solution. If the algorithm cannot find a better solution than the best one found so far M_{best} for lim_{iter} iterations, then the M_{best} is returned as the result. The counting of the number of iterations to compare with lim_{iter} is achieved by keeping track of an additional iteration counter that resets each time a better solution is found.

2. Depth limit lim_{desc}, which indicates the depth of the neighbourhood search starting from an initial stable matching. We define the depth of the search as the number of successor neighbour stable matchings created,

starting from an initial random stable matching. Therefore, lim_{desc} can be illustrated as a limit on the number of stable matchings that descends from a randomly created stable matching. We make use of an additional iteration counter to keep track of the depth. This counter is restarted each time a random stable matching is created.

3. Optimal opt, which indicates if the algorithm has already found the optimal solution (i.e. $b = 1$).

The procedure starts by creating a random stable matching M_c. The algorithm for creating M_c is given in Algorithm 18. In this algorithm, first, an empty complete closed subset S_c is created. Next, all the non-singular rotations of the underlying instance $\mathcal{V}$ are stored in a list A, indicating that all the rotations in A are safe to add to the S_c. "Safe" in this context means that adding $\rho \in A$ does not violate the rules for being a complete closed subset for S_c.

The procedure consists of repeating a random rotation ρ_r selection from the set of available rotations A, adding the ρ_r to S_c by also adding all of its predecessors ρ_p until S_c contains one of each dual rotation. When a rotation ρ is added to S_c, then both ρ and its dual $\bar{\rho}$ are removed from the available set A. This removal ensures that there exists only one rotation in S_c from each dual pair (i.e. ρ, $\bar{\rho}$).

After creating a random stable matching M_c identified by its complete closed subset S_c, the $\boldsymbol{N}$ of M_c is found by checking all the sink rotations in S_c. By using Lemma 21, we know that flipping any sink rotation in M_c creates another stable matching M_n, which we refer as a ***neighbour*** of the M_c. The set of neighbours for a stable matching defines its ***neighbourhood***.

Algorithm 18 Random stable matching creation

```
1:  procedure CREATERANDOMSM(), return: a complete closed subset
2:      S_c ← {}
3:      A ← V
4:      while |S_c| < |V|/2 do
5:          ρ_r ← select a random rotation in A
6:          S_c ← S_c ∪ {ρ_r}
7:          A ← A \ {ρ_r, ρ̄_r}
8:          for ρ' ∈ N⁻(ρ_r) do
9:              if ρ' ∉ S_c then
10:                 S_c ← S_c ∪ {ρ'}
11:                 A ← A \ {ρ', ρ̄'}
        return S_c
```

Example. Figure 5.2 illustrates an example reduced rotation poset of an RSR instance that involves 10 non-singular rotations in total. These non-singular rotations are identified as: $\rho_0, \rho_1, \rho_2, \rho_3, \rho_4$, and their duals. Let a complete closed subset on this poset be $S_c = \{\rho_0, \rho_1, \rho_2, \bar{\rho_3}, \bar{\rho_4}\}$, as highlighted in the figure. The sink rotations of S_c can be identified as $\mathbf{L}(S_c) = \{\rho_0, \rho_2, \bar{\rho_3}\}$. Flipping these rotations, one at a time, corresponds to a different neighbour M_{ni} of M_c as shown below:

1. Flip of ρ_0 corresponds to $S_{n1} = \{\bar{\rho_0}, \rho_1, \rho_2, \bar{\rho_3}, \bar{\rho_4}\}$;
2. Flip of ρ_2 corresponds to $S_{n2} = \{\rho_0, \rho_1, \bar{\rho_2}, \bar{\rho_3}, \bar{\rho_4}\}$;
3. Flip of $\bar{\rho_3}$ corresponds to $S_{n3} = \{\rho_0, \rho_1, \rho_2, \rho_3, \bar{\rho_4}\}$.

The reader can verify that all three sets S_{n1}, S_{n2}, S_{n3} are complete closed subsets. Then, we say that the neighbourhood of M_c contains three neighbour stable matchings as follows: $\boldsymbol{N} = \langle M_{n1}, M_{n2}, M_{n3} \rangle$.

Recall that, the LS method proposed for the RSM defines the neighbours over both sink rotations and the neighbour rotations. However, in the RSR, we only use the sink rotations. This is due to neighbour rotations corresponding to the duals of the sink rotations. Flipping a sink rotation or a neighbour rotation results in the same neighbour stable matching.

The general procedure of the LS approach is the same as the one developed for the RSM. The details of the overall procedure can be found in Algorithm 16 (Page 122). They only differ in terms of a random stable matching creation and the definition of the neighbourhood, as discussed above.

The complexity of the LS procedure depends on the computation of the b values (see Section 5.3.3, Page 151). Finding the set of neighbours ($\boldsymbol{N}$) is based on the identification of the sink rotations of the current stable matching identified by its complete closed subset S_c, where there can be at most $|\mathcal{V}|/2$ sink rotations. Then, there is a constant cost for flipping each sink rotation, i.e. achieved simply

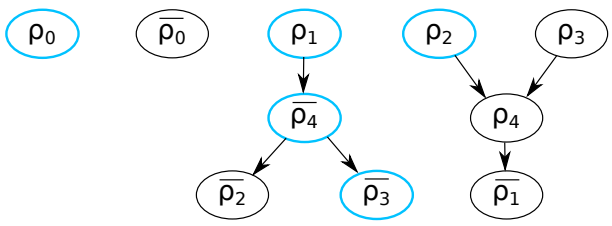


Figure 5.2: The reduced rotation poset of an RSR instance that contain 10 non-singular rotations.

by adding its dual rotation. The best neighbour is identified after computing b values of $|\boldsymbol{N}|$ stable matchings. This procedure takes $O(k \times n \times |\mathcal{V}| \times |\boldsymbol{N}|)$, where k is the number of iterations, and n is the number of non-fixed pairs.

5.4.2 Hybrid Approach

The hybrid approach we define for the RSR is a tailored version of the hybrid algorithm proposed for the RSM (see Section 4.6, Page 124). In order to avoid repetition, we are not disclosing all the details of the HB procedure. The outline of the GA procedure we use in this work is the same as the model described for the RSM detailed in Section 4.4 (Page 111). We only slightly modify some of the functions when compared to the HB model proposed for RSM. These are: definition of the neighbourhood, crossover technique, and the mutation technique. We briefly discuss the changes below.

First, the overall procedure of the HB algorithm tailored for the RSR is structured as follows:

1. Initialize a population by randomly created stable matchings by using Algorithm 18.
2. Evolve the population (randomly select individuals from the population, apply crossover, search for neighbours of the products of crossover, apply mutation).
3. Repeat the evolution until some termination criterion is met (having no improved solutions for a number of iterations lim_{iter}, exceeding the time-limit lim_{time}, or finding the optimal solution $opt = 1$).

As can be seen from the procedure, the only local search enhancement to the GA algorithm is the search for the neighbours of the stable matchings after crossover. Let M_{c1}, M_{c2} be the two stable matchings produced by the crossover. We update M_{c1} by its best neighbour after the neighbour search (the same applies to the M_{c2}). Creating a random stable matching and finding the neighbours of a stable matching are already discussed in Section 5.4.1.

In our model, only the crossover and mutation operations are different than the original GA model defined for the RSM. Instead of defining the crossover by adding rotations to the closed subset or removing them as we did for the RSM, we *flip* them for the RSR. Considering Lemma 22, we define the crossover

procedure for two stable matchings M_1, M_2 as follows. First, we find a random rotation $\rho_1 \in S_1$, and a random rotation $\rho_2 \in S_2$. If the ρ_1 is not in S_2, it means $\bar{\rho_1} \in S_2$ due to the completeness property of the closed subsets in SR. Therefore, we flip $\bar{\rho_1}$ in S_2 and the duals of all of its predecessors $\rho' \in N^-(\rho)$ if ρ' is not included in S_2. We repeat the same procedure for the other stable matching M_2 as well.

Moreover, for the mutation operation, we select a random rotation ρ from the reduced rotation poset of the underlying instance and also a stable matching M by the roulette wheel selection. If $\rho \in S$, we flip ρ and all the required predecessors. If its dual $\bar{\rho} \in S$, then we flip $\bar{\rho}$ and the predecessors.

Note that, the complexity of the method is the same with discussed in Section 4.6 and is $O(k \times n \times |\mathcal{V}| \times |\boldsymbol{N} + \boldsymbol{P}|)$, where k is the number of iterations, n is the number non-fixed pairs, $|\mathcal{V}|$ is the number of rotations in the reduced rotation poset, $|\boldsymbol{N}|$ is the size of neighbourhood, and $|\boldsymbol{P}|$ is the size of the population.

5.5 Experiments

Our objective for these experiments is to find which model is better for solving the RSR instances, and also to observe how the robustness values differ between the RSM and RSR instances that are created by similar methods. In this section, we first compare the performance of the two proposed models: LS and HB on random RSR instances. Then, we provide a comparison of the robustness of different datasets of RSM and RSR instances.

The code is implemented in Java. All experiments are performed on Dell M600s with 2.66 Ghz processors under Linux, using three different randomization seeds. We fixed the time limit as $lim_{time} = 20$ mins, the limit for the number of iterations with no improvement in the solution $lim_{iter} = 5000$, the number of stable matchings that descend from a random stable matching $lim_{desc} = 50$ for each instance. We then report the average of the three runs, unless otherwise stated. We use the population size for the HB as 30 and the mutation probability as 0.7. The reason that we use smaller values for our HB model for the RSR than the HB model for the RSM is mainly due to the size of the problem. We observed in the experiments on dataset MANY that the HB model can be made

faster in the exchange of a small value loss in the value of b [1]. Considering that we work with some large RSR instances, and the overall procedure for the RSR is slower than the RSM, we work with a relatively small population.

5.5.1 A Comparison of Models

In this section, we compare the performances of the HB and the LS models proposed for the RSR. We do not perform the comparison on uniformly random RSR instances. Because, the mean and the variance of the number of stable matchings in random SR instances has been shown to be asymptotic to $e^{1/2}$ and $(\pi n/4e)^{1/2}$ [Pit93]. This value corresponds to very small numbers even for large instance sizes. These numbers can be observed related to our experiments on the dataset called RANDOM later in Section 5.5.2.1.

In order to compare the performances of HB and LS models, we look for RSR instances that are likely to contain many stable matchings to gain more insight on their performances. For this purpose, we first create a dataset of purely random SM instances considering that each SM instance contains at least one stable matching and also they each have a corresponding SR instance (see Lemma 1, Page 45). We observed in Figure 4.8 that the uniformly random SM instances contain many different stable matchings. The values reported in the figure represents a subset of the stable matchings visited by the models. Hence, this conversion tackles the problem of random SR instances having a small number of stable matchings, while preserving the randomness.

Our dataset consists of 30 uniformly random SM instances of each size $n \in \{100 \times k \mid k \in \{1, \ldots, 10\}\}$, where n denotes the number of men/women. Note that, the resulting (i.e. converted) SR instances have size $2 \times n$. On the other hand, RSM instances actually also have $n + n = 2 \times n$ people. Thus, when we report the numbers in terms of pairs: i.e. number of pairs in RSR, which is n, number of men/women which also equals to n, we obtain a fair comparison.

In Figure 5.3 we plot the normalised objective value of the best solution found by the search model $h \in \{\mathrm{LS}, \mathrm{HB}\}$ (x-axis) and the total time (y-axis). The scoring technique used here is presented in detail in Section 4.7.1 (Page 128). Let $h(I)$ be the objective value of the best solution found using model h on instance I and $lb(I)$ (resp. $ub(I)$) the lowest (resp. highest) objective value

[1] Our datasets are publicly available at: github.com/begumgenc/rsmData

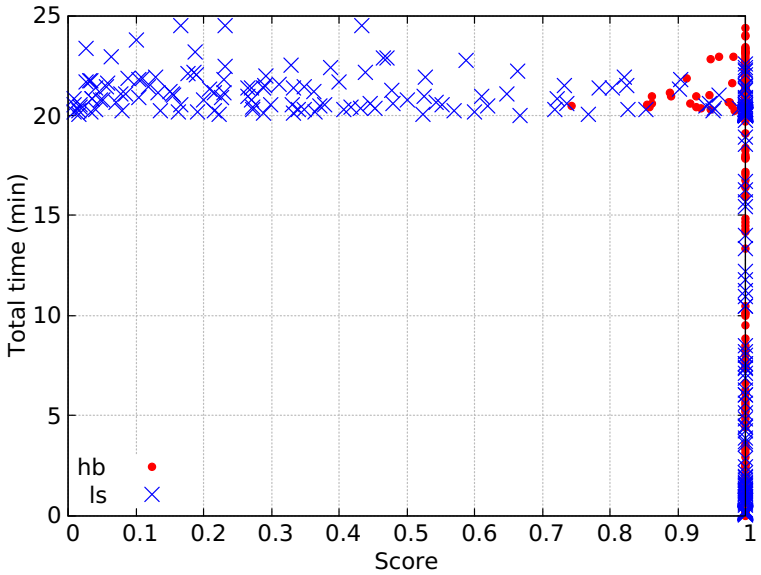


Figure 5.3: Performance comparison of LS and HB models.

found by any model on I. Recall that, the value of $score(h, I)$ is equal to 1 if h has found the best solution for this instance among all models, decreases as $h(I)$ gets further from the optimal objective value, and is equal to 0 if and only if h did not find any solution for I. Each point on this graph represents an instance solved using the related model (HB or LS). Thus, there are 30×10 points for each model.

Note that the running time of some instances in Figure 5.3 exceed the lim_{time}, which is set to 20 minutes. This is due to the cost of creating a stable matching. The instances that exceed the time limit are the ones that have size $n \geq 800$. We see the proof of this in another graph later. We do not interrupt the construction of a stable matching during search and also its calculation for the b value. This operation is costly when there are many pairs, and hence, the time limit is exceeded.

Table 5.9 reports in the rows the average number of different stable matchings created by HB and LS models (**sm-hb** and **sm-ls**, respectively), average minimum values of b found by them (**b-hb** and **b-ls**, respectively) as well as some information about the dataset (i.e. the total number of pairs (**n**), the average values of: non-fixed pairs (**np**), number of rotations in the reduced rotation poset ($|\mathcal{V}|$)).

Both HB and LS models produce very competitive results w.r.t. the average values of minimum b they find when $200 \leq n \leq 800$. LS sometimes finds slightly better results. However, we observe that for instances that have $n \geq 1000$, the

Table 5.9: An overview of performances of HB and LS models on random RSR instances.

n	**200**	**400**	**600**	**800**	**1000**	**1200**	**1400**	**1600**	**1800**	**2000**
np	76.8	166.4	264.4	359.4	454.9	547.7	651	747.8	852.9	948
$\|\mathcal{V}\|$	48.2	83.5	119.7	154.8	181.9	205	239.3	264.8	296.6	321.8
sm-ls	61.1	133.4	222.8	228.2	94.7	47.3	21.7	13.3	6.8	5
sm-hb	45.6	70.5	94.3	115.2	84.9	47.8	21.8	13.3	6.9	4.9
b-ls	49.93	116.90	193.7	268.6	360.7	461.2	592.8	693.7	789.6	910.7
b-hb	49.96	116.91	193.7	268.8	350.9	426.3	529.8	620.5	732.9	834.6

HB produces much better results (i.e. finds lower values for b). However, HB has this advantage only due to the randomization in the initial population creation. When we look at the larger instances in more detail, the most robust solution found in the initial population has almost always better robustness values than the ones found by the LS. Recall that, we use an initial population of size 30 for HB model for the RSR. Even if the same stable matching is created for a few times, recall that we do not re-compute the b value of an already discovered stable matching. Therefore, it does not effect the total time significantly.

We observe in Table 5.9 that the HB and the LS models are both exploring similar number of different stable matchings for larger instances. For instance, for $n = 2000$, the LS model searches for neighbours of a randomly created stable matching and finds 5 neighbours on the average. Similarly, the HB model also creates (on the average) 4.9 different stable matchings. However, the stable matchings found by the HB are more random in the sense that the LS explores only the neighbours of a candidate solution but the HB creates random stable matchings. Hence, the randomization lets HB to find better robustness values. This is the reason for larger-sized LS instances in Figure 5.3 resulting in low scores when compared to the HB.

Figure 5.4 provides more detailed comparison between the LS and the HB. This figure compares the average minimum b value found by the two models for each instance in the set. There are different plots for each size. Each instance has an associated instance ID. In the sub-figures, we can observe a comparison of the performances of the HB and the LS models for each instance in the dataset.

The main observation from the Figure 5.4 is that both the HB and the LS models find similar b values for small instances. However, for instances where $n > 1000$, the HB model finds lower b values than the LS. Figure 5.5 has a similar structure with the Figure 5.4, but it plots the total time used by the each model for each

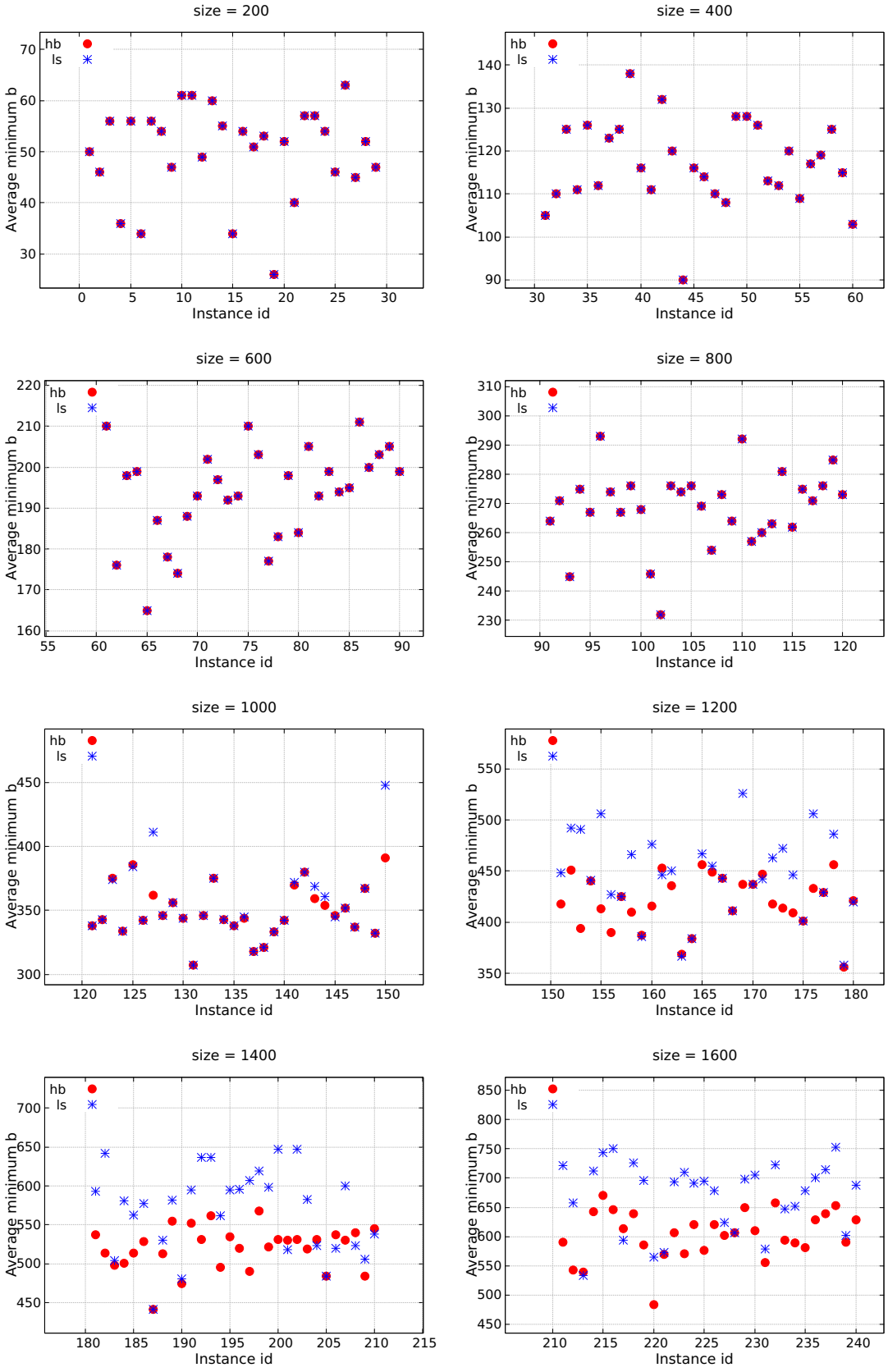


Figure 5.4: Robustness values found by LS and HB models (continued on the next page).

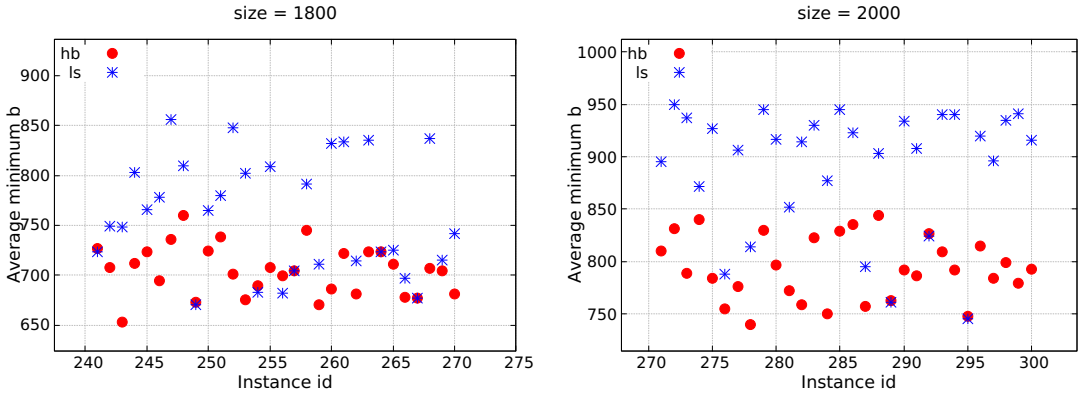


Figure 5.4: Robustness values found by LS and HB models (continued from previous page).

instance. We observe in Figure 5.5 that for each instance that has size $200 \leq n \leq 600$, both models complete the search (i.e. no time-out). Additionally, they find similar b values (in Figure 5.4).

In Figure 5.5, we observe that even though the time limit is set to 20 minutes, most of the instances exceed the time limit when $n > 1000$. This is because, we do not enforce the search to stop while creating a stable matching. As can be observed from the figures, as the size increases, creating one stable matching can take up to nearly 5 minutes (when $n = 2000$). This means that, in the provided time frame (≈ 25 minutes), the HB model only creates 4-5 random stable matchings for the initial population. Thus, can not perform any of the search steps, but only works on creating a small set of random stable matchings.

Figure 5.6 plots the comparison of the average total time spent by HB and LS for all instances of the same size. The average values in Figure 5.6 are similar to the ones obtained from Figure 5.5. In Figure 5.5, each point represents an instance. However, in Figure 5.6, each point represents the average value of all 30 instances of each size.

In summary, we can conclude that for small instances, both HB and LS perform well in terms of finding solutions with low b values. If the time is essential, HB model can be preferred over LS as it converges faster. On the other hand, HB is able to find better solutions for larger instances. However, it is also deceiving in the sense that it performs as a random search due to the initialization of the random population.

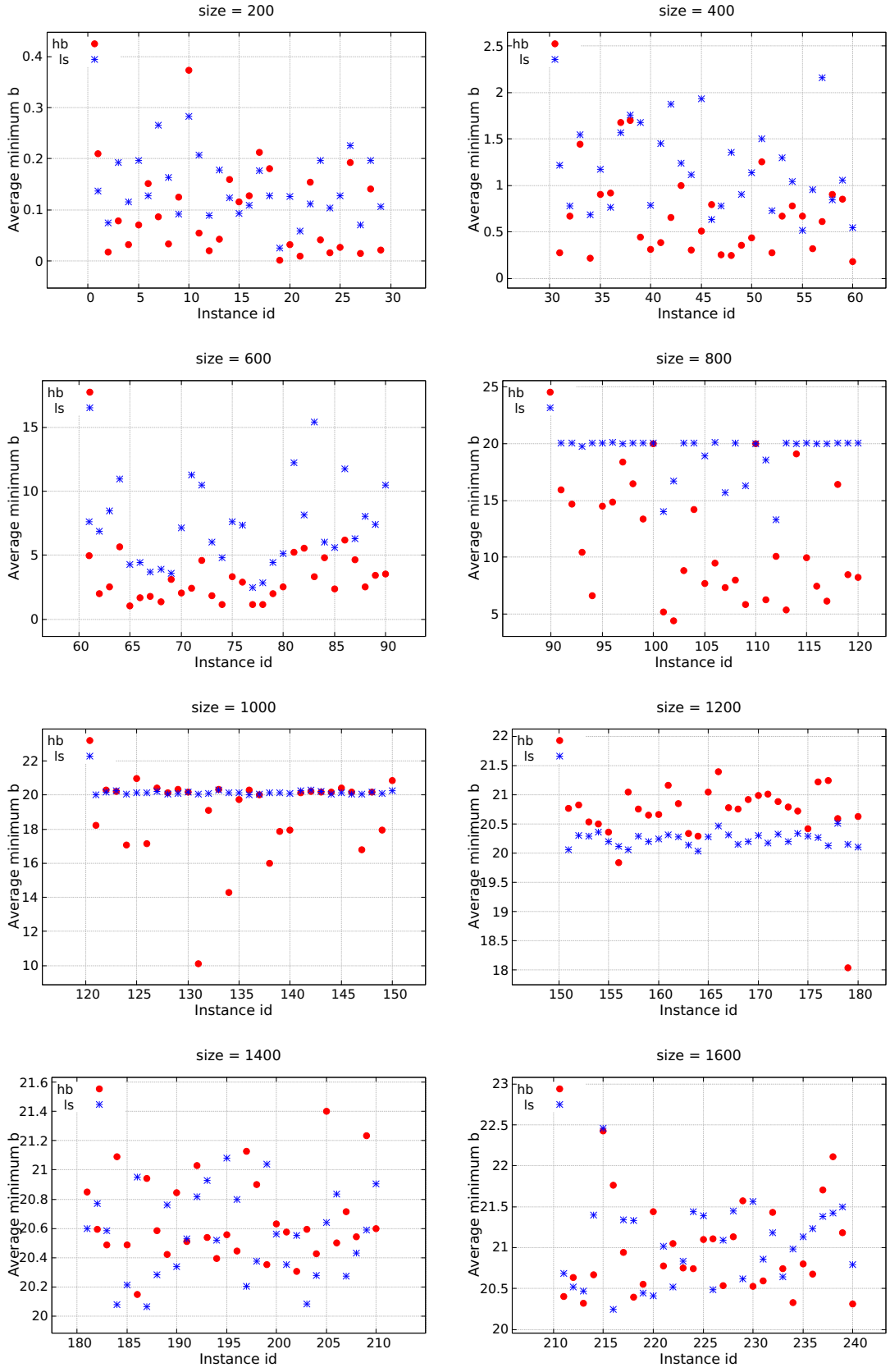


Figure 5.5: Total time spent by LS and HB models (continued on the next page).

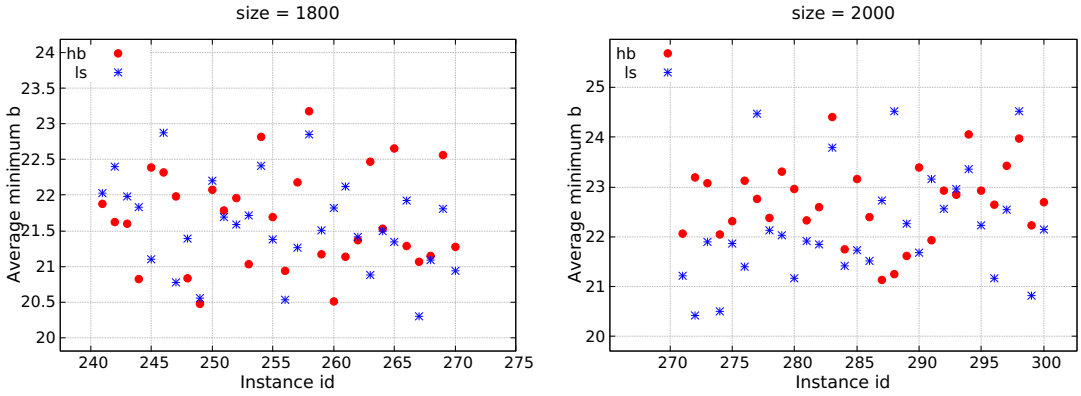


Figure 5.5: Total time spent by LS and HB models (continued from previous page).

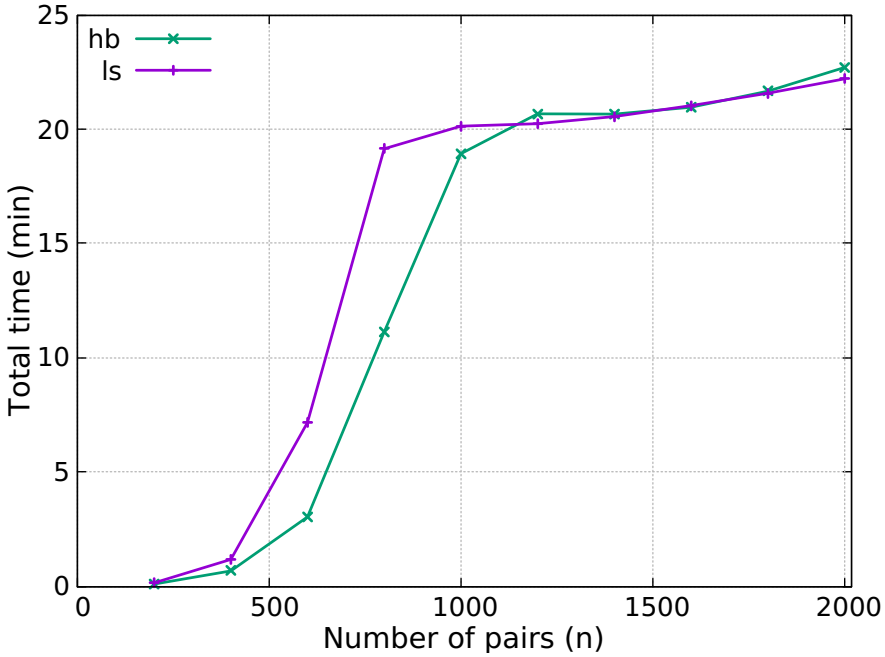


Figure 5.6: Total time spent by the LS and the HB models.

5.5.2 Robustness of RSM vs RSR

In this section, our objective is to investigate the robustness of the RSM and the RSR instances that have some similarities in between. We use different models (the LS or HB) to solve the datasets depending on their structures. In order to perform our experiments, we created the following three datasets that consist of both RSM and RSR instances:

RANDOM: Instances that have randomly created preference lists.

SAME: Instances that have their preference lists created from the same

master list.

MOD: Instances whose preference lists generated by a combination of techniques used for RANDOM and SAME.

In the following sections, we first describe how each dataset is created. We discuss in detail the techniques used to generate the instances for each problem. Subsequently, we perform tests and present results by also providing a comparison when applicable.

5.5.2.1 Experiments on RANDOM

Description. This dataset consists of 30 randomly created SM and 30 randomly created SR instances for each size $n \in \{100 \times k \mid k \in \{1, \ldots, 10\}\}$. Note that, n denotes the number of men or women in an SM instance and each SR instance contains $2 \times n$ people. Noticing that both problems contain n pairs, we refer to n as the number of pairs when presenting the results. We ensure that all the SR instances in the dataset have at least two stable matchings. We set the time limit as $lim_{time} = 20$ mins.

Results. In this section, we present a comparison of robustness values of RSM and RSR instances in the dataset RANDOM. In these experiments, we use the LS model for the RSM instances as the LS performs well in small scaled random RSM instances as shown in Figure 4.4 (Page 127). We also use LS for the RSR as the random SR instances contain only a small number of stable matchings and the LS for RSR works well for small instances(discussed in Section 5.5.1). Note that, the "small" here stands for the average number of non-fixed pairs, not the total number of the pairs that are involved in the problem instance.

Table 5.10 and Table 5.11 present a summary of the robustness of the random RSM and the RSR instances. The columns report in order for each size the average value of: the total number of pairs in the instance (**n**), the number of rotations in the rotation poset (RSM) or the reduced rotation poset (RSR) ($|\mathcal{V}|$), the number of stable matchings found during the search (**sm**), the number of non-fixed pairs (**np**), the value of b of the best solution (**b**), the ratio $\frac{b}{np}$ (**ratio**), and the time spent until finding the best solution in seconds ($\boldsymbol{t_{best}}$).

None of the instances in RANDOM exceeds the time limit lim_{time}. A solution to the random RSR instances can be found quickly due to the huge number of the

Table 5.10: Results on uniformly random instances for RSM.

n	$\lvert\mathcal{V}\rvert$	sm	np	b	ratio	t_{best}
100	22.02	47.9	75.12	48.27	0.64	0.02
200	41.59	116.9	166.19	115.34	0.69	0.10
300	60.22	182.4	263.94	193.08	0.73	0.37
400	74.51	244.1	356.58	265.98	0.74	0.77
500	91.47	322.5	456.00	350.16	0.76	2.18
600	103.82	394.9	551.10	425.51	0.77	3.67
700	117.08	449.6	646.69	505.61	0.78	5.89
800	131.81	527.6	749.98	595.64	0.79	9.09
900	146.34	585.5	848.32	679.82	0.80	14.60
1000	156.00	632.4	943.23	758.82	0.80	21.16

Table 5.11: Results on uniformly random instances for RSR.

n	$\lvert\mathcal{V}\rvert$	sm	np	b	ratio	t_{best}
100	3.91	3.78	17.91	5.31	0.3	0.003
200	3.87	3.94	26.76	8.52	0.32	0.003
300	4.36	4.56	35.53	11.22	0.32	0.017
400	4.71	5.92	37.64	10.93	0.29	0.048
500	4.29	4.81	37.62	11.70	0.31	0.066
600	4.16	4.48	42.44	14.47	0.34	0.130
700	4.58	5.50	48.71	16.02	0.33	0.312
800	4.93	5.99	55.02	17.39	0.32	0.498
900	4.82	7.07	57.64	18.50	0.32	0.662
1000	4.60	5.19	55.16	18.36	0.33	0.557

fixed pairs compared to the random RSM instances of similar size. However, as observed in Figure 5.7, the general procedure for RSR takes longer as the RSR operations are more costly.

Additionally, observe from the tables that the random RSM instances contain many more stable matchings than the random RSR instances of similar sizes. Recall that, the value of sm denotes only the number of a subset of the stable matchings found during the search. However, we can confirm the RSR instances not containing many stable matchings by looking at the number of rotations in their rotation posets. Note that, for the RSR instances, when 1000 pairs are included, the corresponding rotation posets, on the average, contain $|\mathcal{V}| \approx 5$ rotations.

This is mainly caused by the large numbers of fixed-pairs in the random RSR instances. For instance, the average number of non-fixed pairs in the RSM instances of size 100 is $np = 75.12$. However, we observe in the RSR instances of even 1000 pairs that there are only 55.16 non-fixed pairs on the average,

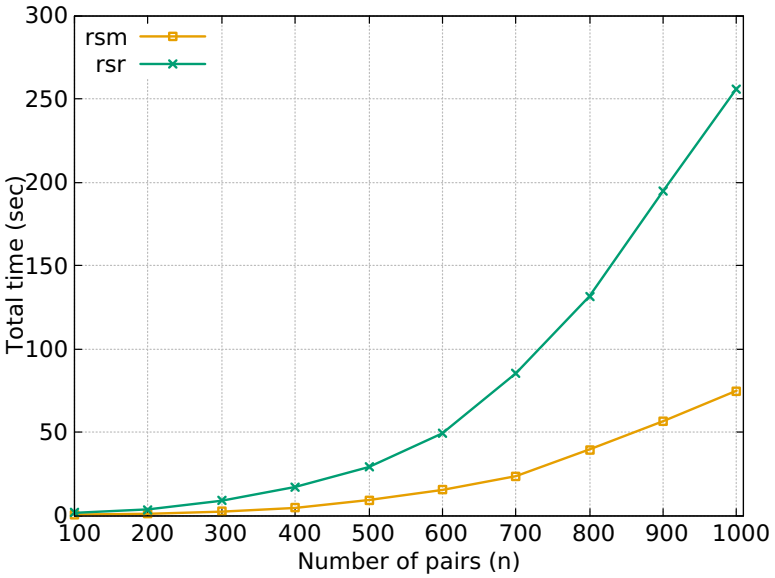


Figure 5.7: Total time spent during search for the RSM and RSR instances in RANDOM.

which is less than the smallest sized RSM instances that we tested. Note that, we measure the robustness ratio over the non-fixed pairs of the instances. It is desirable to obtain a smaller value for the ratio to indicate a better robustness for the instance. Because a smaller ratio indicates that a smaller proportion of the people that have alternative partners need to change their partners for a repair. Observe that, the ratio of the RSR instances is lower when compared to RSM. The ratio shows that the breakage of the pairs in the RSR instances are less costly to be repaired. Thus, we conclude that purely random RSR instances require a smaller proportion of the people to change their partners in the case of a breakage, when compared to the RSM.

5.5.2.2 Experiments on MANY

We discussed in Section 4.7.2 (Page 132) a dataset for SM instances called MANY that contains many number of stable matchings in each instance.

Recall that, each SM instance has a corresponding SR instance. Hence, it is correct to state that each SM instance in MANY has a corresponding SR instance that contains the exact same stable matchings with the exact same robustness values. Therefore, we know that by converting the SM instances in MANY into their corresponding SR instances, we obtain SR instances that contain many number of stable matchings.

We did not repeat the robustness tests on the corresponding SR instances as the search models proposed for the RSR is much slower when compared to the RSM instances of similar sizes. The operations for RSR is already more costly as discussed in Section 5.3.3. Additionally, the rotation posets of the corresponding SR instances in MANY are twice as big than the SM ones. This problem results in many timed-out instances when performing the experiments on RSR. However, we know that the findings of the RSM on MANY apply to the corresponding RSR instances. We mention about this conversion, because any interested reader can use the SM instances in MANY to create SR instances that contain many stable matchings. In addition to the existence of this family, we also know that the RSM instances in MANY are very robust. Due to the correspondence, the robustness results are lifted to the RSR.

5.5.2.3 Experiments on SAME

Description. In this section, we are mainly interested in answering the following two questions: "How are the instances affected by small modifications on the preference lists?" and "What is the effect of slightly modifying the RSM or the RSR instances on their robustness values?". In this section, we create a dataset celled SAME to answer these questions.

Using the same master preference list to create HR, SM and SR instances has already been studied by different researchers as we briefly discuss below. The main idea is to use one (or two) master list(s) to derive the people's preference lists. Irving et. al. study variants of the SM including ties and incomplete lists, using master lists [IMS08]. Escamocher and O'Sullivan use master lists in 3D Stable Marriage problem to generate a set of instances and show the exact number of stable matchings in those instances [EO18]. Additionally, O'Malley studies the HR problem when the preferences are derived from a single master list [O'M07]. He also studies SR using master lists when the problem has ties and incomplete lists.

We define below our procedure for how to use master lists to generate SAME. For the SM instances, we first create a master list denoted by l_m for men, and a different master list for women denoted by l_w. These lists contain all members of the opposite sex in arbitrary order. Then, we make use of a probability value $pr \in [0, 1]$ to control the alteration of the master list. The process starts with generating the preference lists for each man m_i by copying each person in l_m

into the list of m_i. For the first person's preference list, we do not change the master list. For the remaining $n-1$ lists, we change the lists as follows: For each woman $w \in l_m$, we first generate a random number $r_n \in [0,1]$. If $r_n < pr$, then we swap w with another randomly selected woman from the master list l_m. However, we do not make this selection completely random. If swapping a pair results in the agents being placed in their original positions in the master list, we do not perform the swap. The same procedure is repeated for the lists created from the master list l_w for women.

Example. Table 5.12 presents three different SM instances of size 6 generated from the same master preference list by using a probability value $pr = 0.2$. Note that, the master preference list $l_m = [1,6,2,5,4,3]$ for the men and $l_w = [5,3,4,6,1,2]$ for the women.

In order to create the SR instances in this dataset, we repeat the same procedure. We generate only one master list l that includes every person of the instance. Then, we copy the master list into each person's preference list by removing themselves from their list. We then swap the positions of some people in each list as defined for the SM case. Table 5.13 presents six different SR instances, consisting of 6 people in total, generated using two different master preference lists. The probability value is set as $pr = 0.2$. The three instances in the first row are obtained by using the same random master preference list

Table 5.12: Three different SM instances of size 6, created from the same master preference list, where the master list for men $l_m = [1,6,2,\ 5,4,3]$ and $l_w = [5,3,4,6,1,2]$ for women.

	Instance-1	**Instance-2**	**Instance-3**
m_i	**Preference list of** m_i	**Preference list of** m_i	**Preference list of** m_i
1	1 6 2 5 4 3	1 6 2 5 4 3	1 6 2 5 4 3
2	1 5 2 6 4 3	5 6 2 3 1 4	1 2 6 5 4 3
3	1 2 6 5 4 3	5 6 2 1 4 3	1 6 3 5 2 4
4	1 4 2 5 6 3	6 1 4 2 5 3	2 6 1 5 4 3
5	1 6 2 5 4 3	1 6 2 5 4 3	1 5 6 4 2 3
6	6 1 5 4 2 3	1 6 2 5 4 3	1 6 2 5 4 3
w_j	**Preference list of** w_i	**Preference list of** w_i	**Preference list of** w_i
1	5 3 4 6 1 2	5 3 4 6 1 2	5 3 4 6 1 2
2	1 3 2 6 5 4	2 3 4 6 1 5	2 3 4 1 5 6
3	6 3 4 5 2 1	5 3 2 1 6 4	6 3 4 5 1 2
4	5 3 4 6 1 2	5 3 4 6 1 2	3 6 4 5 1 2
5	5 2 4 3 1 6	5 3 4 6 2 1	5 3 4 6 1 2
6	5 3 4 6 1 2	1 3 5 6 4 2	5 3 4 1 2 6

Table 5.13: Six different SR instances created from two different master lists l_1, l_2, where $l_1 = [1, 6, 2, 5, 4, 3]$ and $l_2 = [6, 2, 5, 1, 3, 4]$.

	Instance-1	**Instance-2**	**Instance-3**
$l_1 = [1, 6, 2, 5, 4, 3]$			
p_i	**Preference list of** p_i	**Preference list of** p_i	**Preference list of** p_i
1	6 3 5 2 4	4 5 2 3 6	6 2 5 4 3
2	6 5 1 4 3	4 6 5 1 3	1 6 5 4 3
3	1 6 2 5 4	2 6 1 5 4	1 2 6 5 4
4	6 5 2 3 1	1 3 2 5 6	1 6 2 3 5
5	6 1 2 4 3	1 6 2 4 3	6 3 2 4 1
6	1 5 2 4 3	1 2 4 5 3	1 2 5 4 3
$l_2 = [6, 2, 5, 1, 3, 4]$			
p_j	**Preference list of** p_i	**Preference list of** p_i	**Preference list of** p_i
1	6 2 5 3 4	6 2 4 3 5	2 5 6 3 4
2	1 5 6 3 4	4 1 5 3 6	6 5 1 3 4
3	6 2 4 1 5	6 2 1 4 5	6 1 5 4 2
4	5 2 6 1 3	5 6 2 1 3	6 2 5 1 3
5	6 1 2 3 4	6 2 1 3 4	6 1 2 3 4
6	2 4 3 1 5	5 2 3 1 4	2 5 1 3 4

$l_1 = [1, 6, 2, 5, 4, 3]$. The instances in the second row are generated from the same master list $l_2 = [6, 2, 5, 1, 3, 4]$.

To sum up, SAME contains 30 instances from each size $n \in \{100, 200, 300, 400\}$ (n denotes the number of pairs). The probability values are $pr \in \{0.1 \times k \mid k \in \{1, \ldots, 9\}\}$. In our dataset, the 30 instances for each size is obtained by using 5 different master lists and creating 6 different instances from each master list. This is to guarantee using instances that are created from the same master list, but also to use different master lists for diversity. In total, we have 30 instances for each (size, probability) pair for each problem, i.e. the RSM and the RSR.

Results. For these tests, we used the LS model for both RSM and RSR. Because, the proposed LS models perform well for small instances for both problems. See Table 4.5 (Page 131) for the RSM and Table 5.9 (Page 160) for the RSR.

Recall that, we use master lists to obtain RSM and RSR instances in SAME. The modifications on the master lists are controlled by a probability value $pr \in [0, 1]$. Figure 5.8 shows how the average minimum b changes depending on the value of pr for RSM. We infer from these results that when the lists are very similar

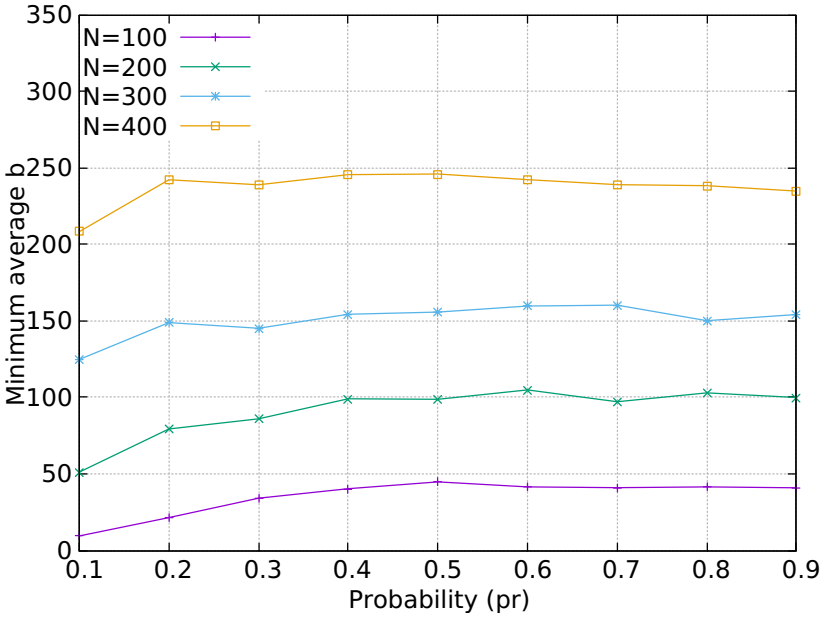


Figure 5.8: Robustness of the SM instances created from the same master list.

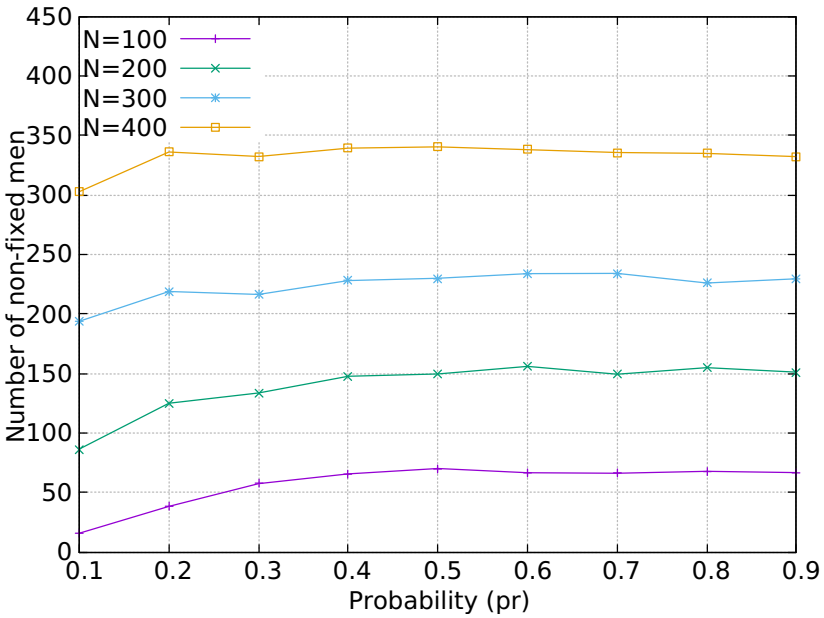


Figure 5.9: The relation between the number of non-fixed men and the probability of modification for the SM instances created from the same master list.

(i.e. $pr < 0.2$), the values of b for the instances are likely to be low, i.e. the repairs costs for breakages are low.

Figure 5.9 shows the relation between the probability pr and the number of non-fixed men in the instances. We can observe from this figure that if all the men/women have very similar preference lists (i.e. $pr < 0.2$), there are more fixed pairs when compared to more random lists. Therefore, we can infer that

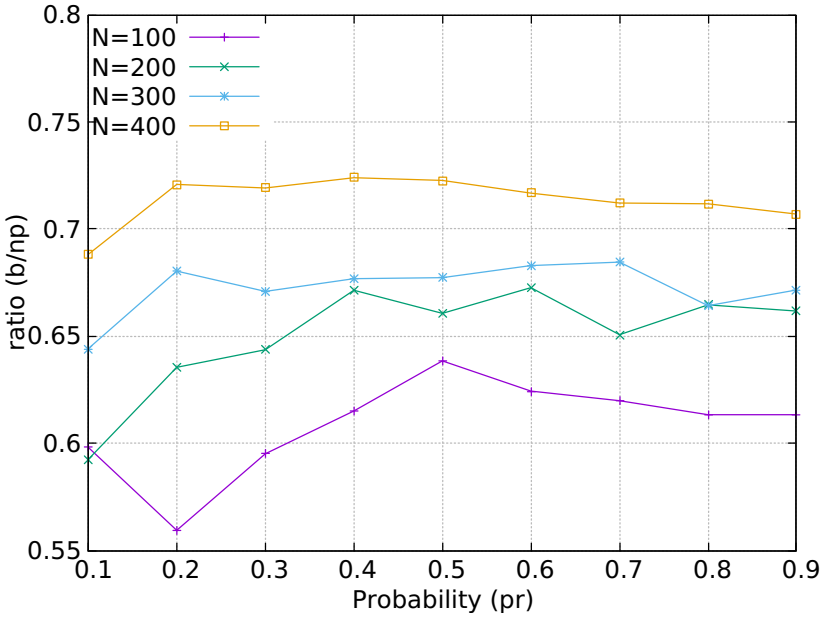


Figure 5.10: Relation between the ratio (b/np) and the probability (pr) of the SM instances created from the same master list.

the robustness values on these instances are low in Figure 5.8, because there does not exist many alternative partners. An alternative analysis is to look at the ratio between the average minimum b values and the number of non-fixed pairs (np). Figure 5.10 plots how the ratio changes with respect to the value of the alteration probability. Note that, a smaller value of the ratio is desirable, as it means that the breakages require less repair cost. This plot shows that the probability does not have a significant effect on the ratio. However, when there are more people involved, the repair costs increase.

The results of the RSR using master lists are not interesting as one may expect. Creating random SR instances and modifying them to obtain other instances make SR very brittle. We found out that out of every 6 instances created from the same master list, on the average 3.7 of them are either unsolvable or has a single solution. This behaviour makes it almost impossible to observe the effect of probability on robustness. Therefore, we do not discuss in detail the results of using same master lists for robustness in random RSR instances.

5.5.2.4 Experiments on MOD

Description. The results obtained from the experiments on SAME bring along the question: "How brittle are the RSR instances against small modifications?". In order to answer this question, we generated a dataset called MOD. In this

dataset, we do not work with uniformly random SR instances as they do not contain many stable matchings. Our aim is to observe what happens if we apply small modifications on the SR instances that are rich in stable matchings.

Our results on the dataset RANDOM in Section 5.5.2.1 show that the random SM instances have many more stable matchings when compared to the random SR instances of similar size. Hence, we make use of random SM instances. The main idea for MOD is to create some random SM instances and then convert them to their corresponding SR instances. By this conversion, we know that the instances in those two sets (i.e. SM instances vs. SR instances) have the exact same stable matchings, and hence, the exact same robustness values. By making small modifications on the corresponding SM and SR instances, we can observe how brittle the robustness values for the RSM and the RSR are.

In this dataset, we first create a random SM dataset for each size $n \in \{100 \times k \mid k \in \{1, \ldots, 10\}\}$ by generating 30 instances from each size. Let *sm_random* denote this dataset. Then, let *sm_mod* consist of slightly modified versions of the instances in *sm_random*. Moreover, let *sr_conv* denote the corresponding SR instances of the SM instances in *sm_random*. Subsequently, let *sr_mod* consist of modified versions of the SR instances in *sr_conv*.

The modification step is similar to the modification process described in SAME. Let us first define a *master instance* in a similar way to a master list. A master instance denotes an instance and consists of a set of preference lists. In this dataset, we use a probability value equal to 0.1 to perform swaps as modification of the lists. The set *sm_mod* is created by using the instances in *sm_random* as master instances. For each instance $\mathcal{I}$ in *sm_random*, the procedure is to copy each preference list in $\mathcal{I}$ to the newly generated instance $\mathcal{I}$' in *sm_mod*. Then, for each person in each preference list, using a random probability value $pr \in [0, 1]$, if $pr < 0.1$ we perform a swap with another person in the same list. We follow the same procedure to generate *sr_mod* from *sr_conv*.

Let us illustrate some instances from each dataset. Recall the SM-SR conversion by the example presented in Table 2.2 (Page 46). In this table, we are given an SM instance $\mathcal{I}_{sm}$ of size 3, and its corresponding SR instance $\mathcal{I}_{sr}$ of size 6. In Table 5.14, we use these two previously shown instances, and also show the modified versions of them. In Table 5.14, we use $\mathcal{I}_{sm} \in$ *sm_random* to denote the original instance, and $\mathcal{I}_{sm-m} \in$ *sm_mod* to denote its modified version. Similarly, we denote the instance converted from $\mathcal{I}_{sm}$ by $\mathcal{I}_{sr} \in$ *sr_conv*. Finally, we denote by $\mathcal{I}_{sr-m} \in$ *sr_mod*, the modified version of $\mathcal{I}_{sr}$.

Table 5.14: An SM instance and the three other instances generated from it given as an overview of the instances in MOD.

	$\mathcal{I}_{sm}$	$\mathcal{I}_{sm-m}$
m_i	**Preference list of** m_i**,** w_i	
m_1	2 3 1	1 3 2
m_2	3 1 2	3 1 2
m_3	2 3 1	2 1 3
w_1	2 3 1	2 3 1
w_2	1 2 3	2 1 3
w_3	1 3 2	1 3 2

	$\mathcal{I}_{sr}$	$\mathcal{I}_{sr-m}$
p_i	**Preference list of** p_i	
p_1	4 6 5 2 3	4 6 5 3 2
p_2	6 4 5 1 3	6 4 5 1 3
p_3	5 6 4 1 2	5 6 4 2 1
p_4	2 3 1 5 6	2 3 1 5 6
p_5	1 2 3 4 6	1 2 6 4 3
p_6	1 3 2 4 5	5 3 2 4 1

Note that, an SM instance of size n corresponds to an SR instance of size $2 \times n$, where both instances have n pairs. Also observe that, the robustness and number of fixed-pairs in both *sm_random* and *sr_conv* are equal. Additionally, the reduced rotation posets of the instances in *sr_conv* are double the size of the rotation posets of their corresponding instances in *sm_random*. In summary, the dataset MOD contains a number of instances as follows:

- *sm_random*: 30 SM instances for each size $n \in \{100 \times k \mid k \in \{1, \ldots, 10\}\}$,
- *sm_mod*: 30 SM instances for each size $n \in \{100 \times k \mid k \in \{1, \ldots, 10\}\}$,
- *sr_conv*: 30 SR instances for each size $n \in \{200 \times k \mid k \in \{1, \ldots, 10\}\}$,
- *sr_mod*: 30 SR instances for each size $n \in \{200 \times k \mid k \in \{1, \ldots, 10\}\}$.

Results. We know that LS performs well on small instances for the RSM instancs, when $size < 1000$ (see Table 4.5, Page 131). Although LS model performs well for the small RSR instances, HB model finds better results (i.e. smaller values for b) when $n > 1200$ (see Table 5.9, Page 160). Therefore, we use the LS model to perform the experiments on the RSM instances, and the HB model for the RSR instances in this dataset.

Recall that, MOD is created by first using a number of SM instances, and then converting them to SR instances. The conversion guarantees that the two corresponding SM and SR instances contain the exact same stable matchings. Later, we slightly modify each master SM and SR instance.

In Table 5.15 the average number of the non-fixed pairs for each size n for each dataset is presented. The results indicate that RSR instances that are rich in stable matchings are not as brittle as the purely random ones. We can observe that small modifications on the instances do not lead to major changes.

Table 5.15: An overview of the instances in MOD with respect to the average number of non-fixed pairs in each set.

n	sm-random	sm-mod	sr-conv	sr-mod
100	76.80	76.67	76.80	79.87
200	166.37	166.17	166.37	156.87
300	264.43	263.57	264.43	276.20
400	359.4	359.73	359.4	357.17
500	454.87	454.07	454.87	446.27
600	547.67	547.10	547.67	549.37
700	651.00	650.60	651.00	656.33
800	747.83	747.33	747.83	739.53
900	852.97	853.70	852.97	853.60
1000	948.03	947.30	948.03	927.63

Although the RSR instances seem to be affected more by the modification, the effect of more modification is also a factor here. In other words, we should also consider the fact that, for the RSR instances, for each preference list, the chances of swapping two pairs is 2 times greater than the modification of a list in RSM. Because, the preference lists in RSR are nearly double the size of the ones in RSM. And we consider applying a swap for each person in the preference list. Hence, the RSR instances can be modified slightly more.

Figure 5.11 illustrates the relation between the number of pairs and the robustness (i.e. the average minimum b values) of the original RSM (*sm_random*), RSR (*sr_conv*), and their modified instances (*sm_mod* and *sr_mod*). First outcome of this figure is that, the HB model for RSR performs worse than the LS model on RSM for larger instances, where $n > 600$. Because, the robustness values for *sm_random* and *sr_conv* must be equal as the instances in *sr_conv* are the corresponding SR instances of *sm_random*. However, the observation is that *sm-random-ls* has lower values than *sr-conv-hb* for sizes $n > 600$. Hence, we can say that for the RSR instances that have $n \leq 600$ pairs, and that are rich in stable matchings, the HB model performs very well. The reader should note that, we can not verify that the LS model for the RSM produces the optimal solutions as our CP model cannot be used for the large instances. However, under the assumption that LS performs well on the RSM instances, we can say that the HB model for the RSR performs well for the instances that have $n \leq 600$ pairs.

Second observation on this dataset is that, the RSR instances that contain many stable matchings are not as brittle as the ones observed in SAME. The robustness values we obtain from *sm_mod* is nearly the same with *sm_random*. We

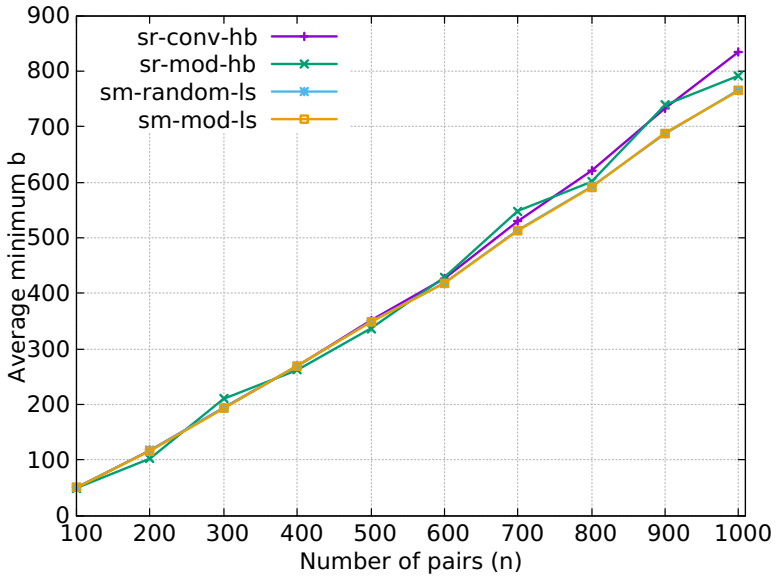


Figure 5.11: Robustness of the instances in MOD.

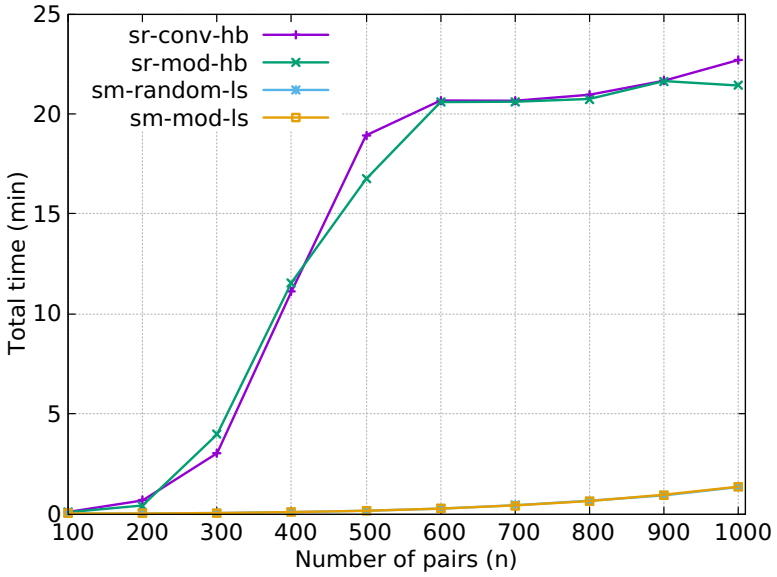


Figure 5.12: The average total time of the instances in MOD.

observe some difference between the robustness values found for larger sizes. For instance, for $n = 1000$, *sm_mod* has on the average ≈ 947 non-fixed pairs. On the other hand, *sr_mod* has on the average ≈ 928 non-fixed pairs (see Table 5.15). The expectation is to see the robustness value of *sr_mod* below the *sm_mod* for $n = 1000$. However, we observe a larger value. We observe that for smaller sizes, all four datasets produce nearly the same results.

In order to understand the poor performance of the HB model on the larger RSR instances, we look at the times spent by the models. In Figure 5.12, we observe that HB model is terminated by the time limit for sizes $n > 600$. Note

that, this is an expected result from Figure 5.5 (Page 163). This termination also explains the poor performance of the HB for the larger sizes. On the other hand, the LS method on the RSM instances runs without time-out.

The main outcomes of the experiments on MOD can be summarized as:

- The RSM and the RSR instances that contain similar number of pairs are equally brittle.
- The HB model proposed for the RSR instances finds b values as good as the LS model proposed for the RSM instances of same size.
- Using the HB model for the RSR instances that have $n > 600$ pairs is not practical due to the time-limit.

5.6 Chapter Summary

In this section, we extended the idea of using (a, b)-supermatches as a notion of robustness to the Stable Roommates problem. We discussed in detail the structural similarities and differences between the rotation posets of Stable Marriage problem and Stable Roommates problem. Then, we proposed a polynomial-time procedure using the reduced rotation poset of the underlying SR instance for deciding if a given stable matching is a $(1, b)$-supermatch. Finally, we adapted a local search algorithm and a hybrid algorithm that uses the polynomial-time procedure to find a $(1, b)$-supermatch to a given SR instance. Our findings on the comparison of the two models indicate that the hybrid algorithm for the RSR performs well on the large instances. However, this is mainly achieved by taking advantage of the initialization of a random population. On a comparison of the RSM and the RSR, we identified a family of SR instances that are rich in stable matchings and very robust (i.e. when $a = 1$, they have very low b values). We observed that the uniformly random RSR instances do not contain many stable matchings, but the breakages on their non-fixed pairs can be repaired at a relatively low cost. The RSR instances created from the same master lists are very brittle against changes. On the other hand, the RSM instances are very consistent for small modifications. However, if the RSR instances that contain many stable matchings are not as brittle, and in fact, they have the same consistency with the RSM instances. A final, important remark is that we observed that the HB model proposed for the RSR is able to find good solutions for the RSR instances that have $n \leq 600$ pairs.

Chapter 6

Conclusion and Future Work

6.1 Thesis Defence

Thesis. *Matching problems are widely-studied computational problems that require assigning agents to one another under different optimality criteria. The stability criterion in matching problems is well defined and dominantly used. However, imposing only the stability constraint on matchings is not enough on its own when the dynamism of the real world due to unexpected events is considered. Therefore, the need to consider a notion of robustness in addition to the existing stability constraint emerges. We claim that achieving both stability and robustness is possible. We propose a novel concept of robustness that has not been considered in this field before. By defining robust stable matchings we allow systems to handle unexpected events while making a bounded number of changes, after the matchings have been constructed.*

Defence. We proposed a novel robustness notion for stable matching problems to tackle the problem of dynamism of the real world due to unexpected events. Our proposed notion is defined in terms of matchings that are stable and we introduced, in addition, a degree of robustness. We named any such matching an (a, b)-*supermatch*.

In Chapter 3 we formally defined our robustness notion on a famous matching problem called the Stable Marriage problem. Informally, an (a, b)-supermatch denotes a matching that is stable such that if any of the a pairs in the matching break their assignments, another stable matching is guaranteed to be found by changing the partners of those a pairs and also the partners of at most b other

pairs. Subsequently, we defined the problem of finding an (a, b)-supermatch for any given Stable Marriage instance as the Robust Stable Marriage problem. Considering this notion, the most robust solution of a Stable Marriage instance is a $(1, b)$-supermatch that has the minimum b value among all the $(1, b)$-supermatches of the underlying instance.

Next, we studied the complexity of the proposed problem and showed that the problem of finding the most robust stable matching is $\mathcal{NP}$-hard in Chapter 3. In order to prove it, we first worked on a restricted case, where $a = 1, b = 1$. We proposed a special case of SAT that is $\mathcal{NP}$-complete by using Schaefer's Dichotomy theorem. We also identified a specific family of instances of SM. Subsequently, we showed the equivalence between the SAT formula and the case of deciding if there exists a $(1, 1)$-supermatch to an RSM instance from the specific family. Then, we generalized the $\mathcal{NP}$-completeness result to the $(1, b)$ case. In order to prove the $\mathcal{NP}$-completeness of the $(1, b)$ case, we defined a procedure as polynomial-time witness for $(1, b)$-supermatches in Chapter 4. Given a stable matching, this procedure finds in polynomial-time the minimum number of pairs that are required to be assigned new partners in case of one pair's breakage. We defined this procedure based on the structural properties of the rotation posets of the RSM instances.

We developed a CP model to solve RSM for finding a $(1, b)$-supermatch that minimizes the value of b. After proving that the RSM is difficult to solve, we developed three other models that are meta-heuristic approaches. We used the previously defined polynomial-time procedure as the basis of all four models. The CP model is a complete method that guarantees to find the most robust solution if given enough time. We compared the performances of the meta-heuristic models with the CP model on randomly generated RSM instances to see how successful the meta-heuristic models are. We observed that the local search and a genetic-local search hybrid have the best performance. Then, we extended our experiments to a family of instances of RSM that are rich in stable matchings.

In Chapter 5, we applied our novel concept of robustness combined with the stability into another matching problem, namely the Stable Roommates problem. We named the robust version as the Robust Stable Roommates problem. We performed similar steps as we did for the RSM case: we proved that the RSR is $\mathcal{NP}$-hard, we defined a polynomial-time witness for the verification of a $(1, b)$-supermatch, and also developed some meta-heuristic models based on

the polynomial-time procedure. We also tailored the two meta-heuristic models that are found to be performing well on the RSM to the RSR. We analysed the $(1, b)$-supermatches further of a number of different datasets for both the RSM and the RSR.

The overall results showed that introducing robustness combined with stability to the matching problems need not be an easy problem. Different models can be developed to find robust solutions. Our findings show us that some stable matchings are more robust than the others. Therefore, one can significantly benefit from using the most robust stable matching. The main benefit of using the most robust matching is the guarantee that a stable matching can be repaired with a bounded cost if a given number of pairs break. Additionally, we showed the robustness notion is not only useful for the Stable Marriage problem, but it can be applied to different matching problems.

6.2 Future Work

We identify some future research directions and also discuss some questions that arise from the notion of (a, b)-supermatches. As the concept is new in the context of the matching problems, there are many interesting different directions that can be identified. We begin in Section 6.2.1 with identifying some directions for the theoretical aspect of the problem. Subsequently, in Section 6.2.2, we discuss how the models can be improved. We conclude this dissertation by identifying some applications, where the RSM and the RSR can be useful in Section 6.2.3.

6.2.1 Complexity

The theoretical results on the complexity of Robust Stable Marriage problem are obtained by breaking the (a, b)-supermatches into sub-cases as presented in Table 6.1. For a different illustration see Figure 3.7 (Page 91). In this table, we marked the results that we know by ✓or ✗ depending on if the problem belongs to that class or not. There are some problems or cases that we do not know their complexity. We marked those problems with a $?$.

We proved in this dissertation that the decision problems for both $(1, 1)$-supermatches and the $(1, b)$-supermatches are $\mathcal{NP}$-complete. However, we cannot

Table 6.1: The overall complexity results.

Problem	$\mathcal{NP}$**-hard**	$\mathcal{NP}$**-complete**	$\mathcal{P}$
$(1,1)$-supermatch ($\boldsymbol{\pi_{11}}$)	✓	✓	✗
$(1,1)$-supermatch from family $\mathrm{F}_w(\boldsymbol{\pi_{11}^w})$	✗	✗	✓
$(1,b)$-supermatch ($\boldsymbol{\pi_{1b}}$)	✓	✓	✗
(a,b)-supermatch ($\boldsymbol{\pi_{ab}}$)	✓	?	✗
$(a,0)$-supermatch ($\boldsymbol{\pi_{a0}}$)	?	?	?
$(a,1)$-supermatch ($\boldsymbol{\pi_{a1}}$)	✓	?	✗

generalize the $\mathcal{NP}$-completeness result to the case of (a,b)-supermatches, as there is no known polynomial-time witness for the general case as of now.

There is recent study on *Recoverable Team Formation problem (RTF)* [DSOI18]. Demirović et al. studied a very similar notion to our robustness concept. They introduce recoverability into the Team Formation problem. The *Team Formation Problem (TF)* is defined as given a set of agents, selecting a team of agents with minimum cost such that a certain set of skills is covered. They introduce the recoverability to this problem as an additional cost to cope with the unexpected events after the teams have been formed. Their notion of recoverability is very similar to our notion of robustness for the matching problems. They focus on the case of finding another team if any of the agents become unavailable after a team has been formed. In a sense, they could use an (a,b) model notion in which a represents the number of agents that become unavailable, and b represents the repair cost. They show in their work that RTF is Σ_3^P-complete.

Considering the similarities between the RTF and the RSM, we have a strong belief that the general case of finding (a,b)-supermatches is also Σ_3^P-complete. Therefore, we believe it is challenging but an interesting direction to further study the complexity of the general case of finding (a,b)-supermatches.

In our work, we showed that the $(2,0)$-supermatches need not exist for the Robust Stable Marriage problem. We discussed the generalization of this case to the $(a,0)$-supermatches, but left the proof of their existence or non-existence as an open problem. It serves an interesting direction to study the complexity of this, especially for different types of matching problems.

We also leave open the question of how to find $(a,1)$-supermatches, which considers the case of guaranteeing a stable matching at a repair cost of 1 in case any set of a pairs lose their assignments. This case is $\mathcal{NP}$-hard as it is a more general case of the $(1,1)$-supermatches. However, $(a,1)$-supermatches suffer from the break-up of many number of different combinations of men.

We believe that studying the $(a, 1)$-supermatches is an easier step that should be tackled before the complexity of the (a, b)-supermatches. Because, studying the combinations may reveal more structural properties about the rotation posets, and also give a light to $(a, 0)$ and (a, b)-supermatches. Additionally, mining different family of instances that can be solved in polynomial-time can also be interesting.

6.2.2 Improvements on the Current Models

We have provided one exact method for solving the optimization problem for the RSM. Additionally, we provided five meta-heuristic models in total, where three of them are proposed for the RSM and two of them are tailored from the existing models for the RSR. In order to show the complexity of the $(1, 1)$-supermatches, we also provided a SAT formulation and also an independent set formulation of the problem for the RSM.

We performed our experiments presented in this thesis by using the Java programming language. As an improvement, the code can be prepared more efficiently and the total time spent for finding a solution can be reduced. In addition, we implemented our CP model using the Choco constraint library. Although the solver Choco 4 is a powerful one, there are faster solvers available [SFS+14]. Additionally, our model can be improved by applying some search strategies such as the symmetry breaking methods in order to reduce the search [GPP06].

Our meta-heuristic models can be implemented by using strategies. For instance, different crossover and mutation operations can be implemented to help the GA tackle the problem of getting stuck at the local minima. This may also help the hybrid model to perform better. Note that, all our work is based on the rotations and the rotation posets. Other models that are not based on the rotations can be interesting to study.

It would be interesting to model the problem by using other methods such as integer programming, or different meta-heuristic approaches. Moreover, the independent set formulation can lead to an elegant $\mathcal{NP}$-hardness proof for RSM.

6.2.3 Variations and Applications

In this dissertation, we studied two variants of the matching problems: Stable Marriage and Stable Roommates problems due to their structural resemblance. Note that, the SR is a generalization of the SM. We have a strong belief that our polynomial time verification procedure can be adapted to the matching problems that have similar structural properties by only a applying minor modifications on the procedure. Our robustness notion can be applied to many different matching problems. We identify below some problems that we think are most useful to apply the robustness concept.

One of the examples that the (a, b)-supermatches notion could be important is the Kidney-Exchange problem [RSÜ04]. In this problem setting, there exists a set of patients that have kidney failure, and another set of donors. Unfortunately, not each donor is compatible with each patient. If any unexpected events occur after the assignment process such as one of the donors refuse to donate his kidney to his assigned patient for some reason, the assignment model can be repaired by changing some of the assignments. Considering that the time can be critical within the kidney-exchange framework to assess a new assignment, a robust initial solution would be desirable.

Another problem we identify as an interesting application is the paper-reviewer assignments. In some conferences, the organization receives many paper submissions, but they have a limited number of reviewers. It is desirable to find assignments between the papers and the reviewers in a way that all the reviewers get papers that are most relevant to their expertise. Additionally, all the papers must be reviewed by a number of reviewers, while making sure the balance between the reviewers is fair. However, this system can have many unexpected events. For the sake of an example, a reviewer may notice a conflict of interest with one of the papers that is assigned to him/her. Ideally, the reviewer should immediately express the conflict after noticing it. An alternative problem is that, the reviewer may become unavailable at some point. Therefore, an assignment that does not require any additional changes ($b = 0$) or minimum number of additional changes is essential.

An application especially for the two problems identified above, the RSM, or the RSR could be interesting to see the practicality of the solutions from a user point of view. In addition to studying these problem variants, applications of the SM and SR could be considered. For instance, the basic wireless resource manage-

ment problem can be modelled as a matching problem, where the agents are resources and users [GSB+15]. The robustness concept in this setting would be beneficial as a resource can unexpectedly become unavailable at any time during the execution. Additionally, an SR model is used to model P2P networks [LMV+07]. If one has data available related to these applications, an evaluation of them to get an insight on their robustness values could be performed by using the models provided in this dissertation. It would be insightful to see the robustness of real instances.

It is also important to note that, we only studied the basic version of the SM and the SR. Our studied version covers the case of agents having incomplete lists. However, the problem can become more interesting when ties are allowed i.e., to allow people express their preferences by allowing to express indifference between some agents.

Appendix A

CP Model

Below is an example of the CP model presented in Section 4.3 (Page 105), for the Stable Marriage instance provided in Table 2.1 (Page 38) as an example. Find below the full CP model of this SM instance. We begin with identifying the set of stable pairs of this instance: $SP = \{(m_0, w_5), (m_0, w_2), (m_0, w_4), (m_0, w_1), (m_1, w_4), (m_1, w_5), (m_1, w_3), (m_2, w_6), (m_2, w_0), (m_3, w_3), (m_3, w_5), (m_4, w_1), (m_4, w_4), (m_5, w_0), (m_5, w_4), (m_6, w_2), (m_6, w_5), (m_6, w_0), (m_6, w_6)\}$. There exists no fixed-pairs.

The constraints ensuring if a pair is a part of the solution by checking the existence of the rotations that produce and eliminate each pair (see Constraint 4.3):

$$x_{0,5} \leftrightarrow \neg s_0, \quad x_{0,2} \leftrightarrow s_0 \wedge \neg s_2, \quad x_{0,4} \leftrightarrow s_2 \wedge \neg s_3, \quad x_{0,1} \leftrightarrow s_3,$$
$$x_{1,4} \leftrightarrow \neg s_1, \quad x_{1,5} \leftrightarrow s_1 \wedge \neg s_5, \quad x_{1,3} \leftrightarrow s_5, \quad x_{2,6} \leftrightarrow \neg s_4,$$
$$x_{2,0} \leftrightarrow s_4, \quad x_{3,3} \leftrightarrow \neg s_5, \quad x_{3,5} \leftrightarrow s_5, \quad x_{4,1} \leftrightarrow \neg s_3,$$
$$x_{4,4} \leftrightarrow s_3, \quad x_{5,0} \leftrightarrow \neg s_1, \quad x_{5,4} \leftrightarrow s_1 \wedge \neg s_2, \quad x_{5,2} \leftrightarrow s_2,$$
$$x_{6,2} \leftrightarrow \neg s_0, \quad x_{6,5} \leftrightarrow s_0 \wedge \neg s_1, \quad x_{6,0} \leftrightarrow s_1 \wedge \neg s_4, \quad x_{6,6} \leftrightarrow s_4.$$

The constraints to make sure the solution forms a closed subset by requiring all predecessors of each rotation (see Constraint 4.4):

$$s_1 \rightarrow s_0, \quad s_2 \rightarrow s_1, \quad s_3 \rightarrow s_2, \quad s_4 \rightarrow s_1, \quad s_5 \rightarrow s_4.$$

The constraints that handle the case if a man is matched to his best possible partner (see Constraint 4.5):

$$\alpha_0 \leftrightarrow x_{0,5}, \quad \alpha_1 \leftrightarrow x_{1,4}, \quad \alpha_2 \leftrightarrow x_{2,6}, \quad \alpha_3 \leftrightarrow x_{3,3},$$
$$\alpha_4 \leftrightarrow x_{4,1}, \quad \alpha_5 \leftrightarrow x_{5,0}, \quad \alpha_6 \leftrightarrow x_{6,2}.$$

The constraints to state if a pair is no longer wanted, the rotation that produces the couple must be removed from the solution closed subset (see Constraint 4.6):

$$x_{0,2} \leftrightarrow s^0_{up_0}, \quad x_{0,4} \leftrightarrow s^0_{up_2}, \quad x_{0,1} \leftrightarrow s^0_{up_3}, \quad x_{1,5} \leftrightarrow s^1_{up_1},$$
$$x_{1,3} \leftrightarrow s^1_{up_5}, \quad x_{2,0} \leftrightarrow s^2_{up_4}, \quad x_{3,5} \leftrightarrow s^3_{up_5}, \quad x_{4,4} \leftrightarrow s^4_{up_3},$$
$$x_{5,4} \leftrightarrow s^5_{up_1}, \quad x_{5,2} \leftrightarrow s^5_{up_2}, \quad x_{6,5} \leftrightarrow s^6_{up_0}, \quad x_{6,0} \leftrightarrow s^6_{up_1},$$
$$x_{6,6} \leftrightarrow s^6_{up_4}.$$

The constraints that ensure there does not exist a repair matching S^{*i}_{UP} for man m_i if he is matched to his best possible partner (see Constraint 4.7):

$$\alpha_0 \rightarrow \neg s^0_{up_0} \wedge \neg s^0_{up_1} \wedge \neg s^0_{up_2} \wedge \neg s^0_{up_3} \wedge \neg s^0_{up_4} \wedge \neg s^0_{up_5},$$
$$\alpha_1 \rightarrow \neg s^1_{up_0} \wedge \neg s^1_{up_1} \wedge \neg s^1_{up_2} \wedge \neg s^1_{up_3} \wedge \neg s^1_{up_4} \wedge \neg s^1_{up_5},$$
$$\alpha_2 \rightarrow \neg s^2_{up_0} \wedge \neg s^2_{up_1} \wedge \neg s^2_{up_2} \wedge \neg s^2_{up_3} \wedge \neg s^2_{up_4} \wedge \neg s^2_{up_5},$$
$$\alpha_3 \rightarrow \neg s^3_{up_0} \wedge \neg s^3_{up_1} \wedge \neg s^3_{up_2} \wedge \neg s^3_{up_3} \wedge \neg s^3_{up_4} \wedge \neg s^3_{up_5},$$
$$\alpha_4 \rightarrow \neg s^4_{up_0} \wedge \neg s^4_{up_1} \wedge \neg s^4_{up_2} \wedge \neg s^4_{up_3} \wedge \neg s^4_{up_4} \wedge \neg s^4_{up_5},$$
$$\alpha_5 \rightarrow \neg s^5_{up_0} \wedge \neg s^5_{up_1} \wedge \neg s^5_{up_2} \wedge \neg s^5_{up_3} \wedge \neg s^5_{up_4} \wedge \neg s^5_{up_5},$$
$$\alpha_6 \rightarrow \neg s^6_{up_0} \wedge \neg s^6_{up_1} \wedge \neg s^6_{up_2} \wedge \neg s^6_{up_3} \wedge \neg s^6_{up_4} \wedge \neg s^6_{up_5}.$$

The constraints that require if a rotation is in the difference set $S \setminus S^{*i}_{\text{UP}}$, then all successors of that rotation that are a part of solution are also in the difference set (see Constraint 4.8). These constraints are listed as below.

For $i = 0$:

$$s^0_{up_0} \wedge s_1 \rightarrow s^0_{up_1}, \quad s^0_{up_0} \wedge s_2 \rightarrow s^0_{up_2}, \quad s^0_{up_0} \wedge s_3 \rightarrow s^0_{up_3},$$
$$s^0_{up_0} \wedge s_4 \rightarrow s^0_{up_4}, \quad s^0_{up_0} \wedge s_5 \rightarrow s^0_{up_5}, \quad s^0_{up_1} \wedge s_2 \rightarrow s^0_{up_2},$$
$$s^0_{up_1} \wedge s_3 \rightarrow s^0_{up_3}, \quad s^0_{up_1} \wedge s_4 \rightarrow s^0_{up_4}, \quad s^0_{up_1} \wedge s_5 \rightarrow s^0_{up_5},$$
$$s^0_{up_2} \wedge s_3 \rightarrow s^0_{up_3}, \quad s^0_{up_4} \wedge s_5 \rightarrow s^0_{up_5}.$$

For $i = 1$:

$$s^1_{up_0} \wedge s_1 \rightarrow s^1_{up_1}, \quad s^1_{up_0} \wedge s_2 \rightarrow s^1_{up_2}, \quad s^1_{up_0} \wedge s_3 \rightarrow s^1_{up_3},$$
$$s^1_{up_0} \wedge s_4 \rightarrow s^1_{up_4}, \quad s^1_{up_0} \wedge s_5 \rightarrow s^1_{up_5}, \quad s^1_{up_1} \wedge s_2 \rightarrow s^1_{up_2},$$
$$s^1_{up_1} \wedge s_3 \rightarrow s^1_{up_3}, \quad s^1_{up_1} \wedge s_4 \rightarrow s^1_{up_4}, \quad s^1_{up_1} \wedge s_5 \rightarrow s^1_{up_5},$$
$$s^1_{up_2} \wedge s_3 \rightarrow s^1_{up_3}, \quad s^1_{up_4} \wedge s_5 \rightarrow s^1_{up_5}.$$

For $i = 2$:

$$
\begin{array}{lll}
s^2_{up_0} \wedge s_1 \rightarrow s^2_{up_1}, & s^2_{up_0} \wedge s_2 \rightarrow s^2_{up_2}, & s^2_{up_0} \wedge s_3 \rightarrow s^2_{up_3}, \\
s^2_{up_0} \wedge s_4 \rightarrow s^2_{up_4}, & s^2_{up_0} \wedge s_5 \rightarrow s^2_{up_5}, & s^2_{up_1} \wedge s_2 \rightarrow s^2_{up_2}, \\
s^2_{up_1} \wedge s_3 \rightarrow s^2_{up_3}, & s^2_{up_1} \wedge s_4 \rightarrow s^2_{up_4}, & s^2_{up_1} \wedge s_5 \rightarrow s^2_{up_5}, \\
s^2_{up_2} \wedge s_3 \rightarrow s^2_{up_3}, & s^2_{up_4} \wedge s_5 \rightarrow s^2_{up_5}. &
\end{array}
$$

For $i = 3$:

$$
\begin{array}{lll}
s^3_{up_0} \wedge s_1 \rightarrow s^3_{up_1}, & s^3_{up_0} \wedge s_2 \rightarrow s^3_{up_2}, & s^3_{up_0} \wedge s_3 \rightarrow s^3_{up_3}, \\
s^3_{up_0} \wedge s_4 \rightarrow s^3_{up_4}, & s^3_{up_0} \wedge s_5 \rightarrow s^3_{up_5}, & s^3_{up_1} \wedge s_2 \rightarrow s^3_{up_2}, \\
s^3_{up_1} \wedge s_3 \rightarrow s^3_{up_3}, & s^3_{up_1} \wedge s_4 \rightarrow s^3_{up_4}, & s^3_{up_1} \wedge s_5 \rightarrow s^3_{up_5}, \\
s^3_{up_2} \wedge s_3 \rightarrow s^3_{up_3}, & s^3_{up_4} \wedge s_5 \rightarrow s^3_{up_5}. &
\end{array}
$$

For $i = 4$:

$$
\begin{array}{lll}
s^4_{up_0} \wedge s_1 \rightarrow s^4_{up_1}, & s^4_{up_0} \wedge s_2 \rightarrow s^4_{up_2}, & s^4_{up_0} \wedge s_3 \rightarrow s^4_{up_3}, \\
s^4_{up_0} \wedge s_4 \rightarrow s^4_{up_4}, & s^4_{up_0} \wedge s_5 \rightarrow s^4_{up_5}, & s^4_{up_1} \wedge s_2 \rightarrow s^4_{up_2}, \\
s^4_{up_1} \wedge s_3 \rightarrow s^4_{up_3}, & s^4_{up_1} \wedge s_4 \rightarrow s^4_{up_4}, & s^4_{up_1} \wedge s_5 \rightarrow s^4_{up_5}, \\
s^4_{up_2} \wedge s_3 \rightarrow s^4_{up_3}, & s^4_{up_4} \wedge s_5 \rightarrow s^4_{up_5}. &
\end{array}
$$

For $i = 5$:

$$
\begin{array}{lll}
s^5_{up_0} \wedge s_1 \rightarrow s^5_{up_1}, & s^5_{up_0} \wedge s_2 \rightarrow s^5_{up_2}, & s^5_{up_0} \wedge s_3 \rightarrow s^5_{up_3}, \\
s^5_{up_0} \wedge s_4 \rightarrow s^5_{up_4}, & s^5_{up_0} \wedge s_5 \rightarrow s^5_{up_5}, & s^5_{up_1} \wedge s_2 \rightarrow s^5_{up_2}, \\
s^5_{up_1} \wedge s_3 \rightarrow s^5_{up_3}, & s^5_{up_1} \wedge s_4 \rightarrow s^5_{up_4}, & s^5_{up_1} \wedge s_5 \rightarrow s^5_{up_5}, \\
s^5_{up_2} \wedge s_3 \rightarrow s^5_{up_3}, & s^5_{up_4} \wedge s_5 \rightarrow s^5_{up_5}. &
\end{array}
$$

For $i = 6$:

$$
\begin{array}{lll}
s^6_{up_0} \wedge s_1 \rightarrow s^6_{up_1}, & s^6_{up_0} \wedge s_2 \rightarrow s^6_{up_2}, & s^6_{up_0} \wedge s_3 \rightarrow s^6_{up_3}, \\
s^6_{up_0} \wedge s_4 \rightarrow s^6_{up_4}, & s^6_{up_0} \wedge s_5 \rightarrow s^6_{up_5}, & s^6_{up_1} \wedge s_2 \rightarrow s^6_{up_2}, \\
s^6_{up_1} \wedge s_3 \rightarrow s^6_{up_3}, & s^6_{up_1} \wedge s_4 \rightarrow s^6_{up_4}, & s^6_{up_1} \wedge s_5 \rightarrow s^6_{up_5}, \\
s^6_{up_2} \wedge s_3 \rightarrow s^6_{up_3}, & s^6_{up_4} \wedge s_5 \rightarrow s^6_{up_5}. &
\end{array}
$$

The constraints that ensure if a rotation ρ is in the difference set $S \setminus S^{*i}_{\text{UP}}$, then ρ must be in the closed subset of the current solution, and it is either the rotation that produced the pair that is a part of the solution or there exists a predecessor of ρ that produces it (see Constraint 4.9).

For $v = \rho_0$:

$$s^0_{up_0} \to s_0 \wedge x_{0,2}, \quad s^1_{up_0} \to s_0, \quad s^2_{up_0} \to s_0,$$
$$s^3_{up_0} \to s_0, \quad s^4_{up_0} \to s_0, \quad s^5_{up_0} \to s_0,$$
$$s^6_{up_0} \to s_0 \wedge x_{6,5}.$$

For $v = \rho_1$:

$$s^0_{up_1} \to s_1 \wedge s^0_{up_0}, \quad s^1_{up_1} \to s_1 \wedge (x_{1,5} \vee s^1_{up_0}), \quad s^2_{up_1} \to s_1 \wedge s^2_{up_0},$$
$$s^3_{up_1} \to s_1 \wedge s^3_{up_0}, \quad s^4_{up_1} \to s_1 \wedge s^4_{up_0}, \quad s^5_{up_1} \to s_1 \wedge (x_{5,4} \vee s^5_{up_0}),$$
$$s^6_{up_1} \to s_1 \wedge (x_{6,0} \vee s^6_{up_0}).$$

For $v = \rho_2$:

$$s^0_{up_2} \to s_2 \wedge (x_{0,4} \vee s^0_{up_0} \vee s^0_{up_1}), \quad s^1_{up_2} \to s_2 \wedge (s^1_{up_0} \vee s^1_{up_1}),$$
$$s^2_{up_2} \to s_2 \wedge (s^2_{up_0} \vee s^2_{up_1}), \quad s^3_{up_2} \to s_2 \wedge (s^3_{up_0} \vee s^3_{up_1}),$$
$$s^4_{up_2} \to s_2 \wedge (s^4_{up_0} \vee s^4_{up_1}), \quad s^5_{up_2} \to s_2 \wedge (x_{5,4} \vee s^5_{up_0} \vee s^5_{up_1}),$$
$$s^6_{up_2} \to s_2 \wedge (s^6_{up_0} \vee s^6_{up_1}).$$

For $v = \rho_3$:

$$s^0_{up_3} \to s_3 \wedge (x_{0,1} \vee s^0_{up_0} \vee s^0_{up_1} \vee s^0_{up_2}), \quad s^1_{up_3} \to s_3 \wedge (s^1_{up_0} \vee s^1_{up_1} \vee s^1_{up_2}),$$
$$s^2_{up_3} \to s_3 \wedge (s^2_{up_0} \vee s^2_{up_1} \vee s^2_{up_2}), \quad s^3_{up_3} \to s_3 \wedge (s^3_{up_0} \vee s^3_{up_1} \vee s^3_{up_2}),$$
$$s^4_{up_3} \to s_3 \wedge (x_{4,4} \vee s^4_{up_0} \vee s^4_{up_1} \vee s^4_{up_2}), \quad s^5_{up_3} \to s_3 \wedge (s^5_{up_0} \vee s^5_{up_1} \vee s^5_{up_2}),$$
$$s^6_{up_3} \to s_3 \wedge (s^6_{up_0} \vee s^6_{up_1} \vee s^6_{up_2}).$$

For $v = \rho_4$:

$$s^0_{up_4} \to s_4 \wedge (s^0_{up_0} \vee s^0_{up_1}), \quad s^1_{up_4} \to s_4 \wedge (s^1_{up_0} \vee s^1_{up_1}),$$
$$s^2_{up_4} \to s_4 \wedge (x_{2,0} \vee s^2_{up_0} \vee s^2_{up_1}), \quad s^3_{up_4} \to s_4 \wedge (s^3_{up_0} \vee s^3_{up_1}),$$
$$s^4_{up_4} \to s_4 \wedge (s^4_{up_0} \vee s^4_{up_1}), \quad s^5_{up_4} \to s_4 \wedge (s^5_{up_0} \vee s^5_{up_1}),$$
$$s^6_{up_4} \to s_4 \wedge (x_{6,6} \vee s^6_{up_0} \vee s^6_{up_1}).$$

For $v = \rho_5$:

$$s^0_{up_5} \to s_5 \wedge (s^0_{up_0} \vee s^0_{up_1} \vee s^0_{up_4}), \quad s^1_{up_5} \to s_5 \wedge (x_{1,3} \vee s^1_{up_0} \vee s^1_{up_1} \vee s^1_{up_4}),$$
$$s^2_{up_5} \to s_5 \wedge (s^2_{up_0} \vee s^2_{up_1} \vee s^2_{up_4}), \quad s^3_{up_5} \to s_5 \wedge (x_{3,5} \vee s^3_{up_0} \vee s^3_{up_1} \vee s^3_{up_4}),$$
$$s^4_{up_5} \to s_5 \wedge (s^4_{up_0} \vee s^4_{up_1} \vee s^4_{up_4}), \quad s^5_{up_5} \to s_5 \wedge (s^5_{up_0} \vee s^5_{up_1} \vee s^5_{up_4}),$$
$$s^6_{up_5} \to s_5 \wedge (s^6_{up_0} \vee s^6_{up_1} \vee s^6_{up_4}).$$

The constraints for construction of the difference set $S^{*i}_{\text{DOWN}} \setminus S$ can also be generated similar to the difference set constraints above; see Constraint 4.10 (Page 110) to Constraint 4.14 (Page 110). We do not include those constraints

here. The following constraints are for keeping track of which other men are required to change their partners in order to repair man m_i for difference set $S \setminus S^{*i}_{\text{UP}}$; see Constraint 4.15 (Page 110).

For $i = 0$:

$$y^0_1 \leftrightarrow s^0_{up_1} \vee s^0_{up_5}, \quad y^0_2 \leftrightarrow s^0_{up_4}, \quad y^0_3 \leftrightarrow s^0_{up_5},$$
$$y^0_4 \leftrightarrow s^0_{up_3}, \quad y^0_5 \leftrightarrow s^0_{up_1} \vee s^0_{up_2}, \quad y^0_6 \leftrightarrow s^0_{up_0} \vee s^0_{up_1} \vee s^0_{up_4}.$$

For $i = 1$:

$$y^1_0 \leftrightarrow s^1_{up_0} \vee s^1_{up_2} \vee s^1_{up_3}, \quad y^1_2 \leftrightarrow s^1_{up_4}, \quad y^1_3 \leftrightarrow s^1_{up_5},$$
$$y^1_4 \leftrightarrow s^1_{up_3}, \quad y^1_5 \leftrightarrow s^1_{up_1} \vee s^1_{up_2}, \quad y^1_6 \leftrightarrow s^1_{up_0} \vee s^1_{up_1} \vee s^1_{up_4}.$$

For $i = 2$:

$$y^2_0 \leftrightarrow s^2_{up_0} \vee s^2_{up_2} \vee s^2_{up_3}, \quad y^2_1 \leftrightarrow s^2_{up_1} \vee s^2_{up_5}, \quad y^2_3 \leftrightarrow s^2_{up_5},$$
$$y^2_4 \leftrightarrow s^2_{up_3}, \quad y^2_5 \leftrightarrow s^2_{up_1} \vee s^2_{up_2}, \quad y^2_6 \leftrightarrow s^2_{up_0} \vee s^2_{up_1} \vee s^2_{up_4}.$$

For $i = 3$:

$$y^3_0 \leftrightarrow s^3_{up_0} \vee s^3_{up_2} \vee s^3_{up_3}, \quad y^3_1 \leftrightarrow s^3_{up_1} \vee s^3_{up_5}, \quad y^3_2 \leftrightarrow s^3_{up_4},$$
$$y^3_4 \leftrightarrow s^3_{up_3}, \quad y^3_5 \leftrightarrow s^3_{up_1} \vee s^3_{up_2}, \quad y^3_6 \leftrightarrow s^3_{up_0} \vee s^3_{up_1} \vee s^3_{up_4}.$$

For $i = 4$:

$$y^4_0 \leftrightarrow s^4_{up_0} \vee s^4_{up_2} \vee s^4_{up_3}, \quad y^4_1 \leftrightarrow s^4_{up_1} \vee s^4_{up_5}, \quad y^4_2 \leftrightarrow s^4_{up_4},$$
$$y^4_3 \leftrightarrow s^4_{up_5}, \quad y^4_5 \leftrightarrow s^4_{up_1} \vee s^4_{up_2}, \quad y^4_6 \leftrightarrow s^4_{up_0} \vee s^4_{up_1} \vee s^4_{up_4}.$$

For $i = 5$:

$$y^5_0 \leftrightarrow s^5_{up_0} \vee s^5_{up_2} \vee s^5_{up_3}, \quad y^5_1 \leftrightarrow s^5_{up_1} \vee s^5_{up_5}, \quad y^5_2 \leftrightarrow s^5_{up_4},$$
$$y^5_3 \leftrightarrow s^5_{up_5}, \quad y^5_4 \leftrightarrow s^5_{up_3}, \quad y^5_6 \leftrightarrow s^5_{up_0} \vee s^5_{up_1} \vee s^5_{up_4}.$$

For $i = 6$:

$$y^6_0 \leftrightarrow s^6_{up_0} \vee s^6_{up_2} \vee s^6_{up_3}, \quad y^6_1 \leftrightarrow s^6_{up_1} \vee s^6_{up_5}, \quad y^6_2 \leftrightarrow s^6_{up_4},$$
$$y^6_3 \leftrightarrow s^6_{up_5}, \quad y^6_4 \leftrightarrow s^6_{up_3}, \quad y^6_5 \leftrightarrow s^6_{up_1} \vee s^6_{up_2}.$$

For difference set $S^{*i}_{\text{DOWN}} \setminus S$, the constraints are generated similarly, except the Boolean variables y are replaced with z and s_{up} are replaced with s_{down}; see Constraint 4.16 (Page 110). We do not include them here.

$$\alpha_0 = \text{true} \rightarrow d^0_{up} = n, \text{else} \rightarrow d^0_{up} = y^0_1 + y^0_2 + y^0_3 + y^0_4 + y^0_5 + y^0_6,$$
$$\alpha_1 = \text{true} \rightarrow d^1_{up} = n, \text{else} \rightarrow d^1_{up} = y^1_0 + y^1_2 + y^1_3 + y^1_4 + y^1_5 + y^1_6,$$
$$\alpha_2 = \text{true} \rightarrow d^2_{up} = n, \text{else} \rightarrow d^2_{up} = y^2_0 + y^2_1 + y^2_3 + y^2_4 + y^2_5 + y^2_6,$$
$$\alpha_3 = \text{true} \rightarrow d^3_{up} = n, \text{else} \rightarrow d^3_{up} = y^3_0 + y^3_1 + y^3_2 + y^3_4 + y^3_5 + y^3_6,$$
$$\alpha_4 = \text{true} \rightarrow d^4_{up} = n, \text{else} \rightarrow d^4_{up} = y^4_0 + y^4_1 + y^4_2 + y^4_3 + y^4_5 + y^4_6,$$
$$\alpha_5 = \text{true} \rightarrow d^5_{up} = n, \text{else} \rightarrow d^5_{up} = y^5_0 + y^5_1 + y^5_2 + y^5_3 + y^5_4 + y^5_6,$$
$$\alpha_6 = \text{true} \rightarrow d^6_{up} = n, \text{else} \rightarrow d^6_{up} = y^6_0 + y^6_1 + y^6_2 + y^6_3 + y^6_4 + y^6_5.$$

The constraints for counting how many other men are required to change their partners in order to provide a repair stable matching that dominates the current matching are listed below; see Constraint 4.17 (Page 111).

Similarly, the number of men that need to change their partners to obtain the the dominated stable matching is calculated as follows; see Constraint 4.18 (Page 111).

$$\beta_0 = \text{true} \rightarrow d^0_{down} = n, \text{else} \rightarrow d^0_{down} = z^0_1 + z^0_2 + z^0_3 + z^0_4 + z^0_5 + z^0_6,$$
$$\beta_1 = \text{true} \rightarrow d^1_{down} = n, \text{else} \rightarrow d^1_{down} = z^1_0 + z^1_2 + z^1_3 + z^1_4 + z^1_5 + z^1_6,$$
$$\beta_2 = \text{true} \rightarrow d^2_{down} = n, \text{else} \rightarrow d^2_{down} = z^2_0 + z^2_1 + z^2_3 + z^2_4 + z^2_5 + z^2_6,$$
$$\beta_3 = \text{true} \rightarrow d^3_{down} = n, \text{else} \rightarrow d^3_{down} = z^3_0 + z^3_1 + z^3_2 + z^3_4 + z^3_5 + z^3_6,$$
$$\beta_4 = \text{true} \rightarrow d^4_{down} = n, \text{else} \rightarrow d^4_{down} = z^4_0 + z^4_1 + z^4_2 + z^4_3 + z^4_5 + z^4_6,$$
$$\beta_5 = \text{true} \rightarrow d^5_{down} = n, \text{else} \rightarrow d^5_{down} = z^5_0 + z^5_1 + z^5_2 + z^5_3 + z^5_4 + z^5_6,$$
$$\beta_6 = \text{true} \rightarrow d^6_{down} = n, \text{else} \rightarrow d^6_{down} = z^6_0 + z^6_1 + z^6_2 + z^6_3 + z^6_4 + z^6_5.$$

Finally, the constraint for measuring the robustness of the solution concludes the model; see Constraint 4.19 (Page 111).

$$\Big(\neg(\alpha_0) \rightarrow (b \geq d^0_{up})\Big) \vee \Big(\neg(\beta_0) \rightarrow (b \geq d^0_{down})\Big),$$
$$\Big(\neg(\alpha_1) \rightarrow (b \geq d^1_{up})\Big) \vee \Big(\neg(\beta_1) \rightarrow (b \geq d^1_{down})\Big),$$
$$\Big(\neg(\alpha_2) \rightarrow (b \geq d^2_{up})\Big) \vee \Big(\neg(\beta_2) \rightarrow (b \geq d^2_{down})\Big),$$
$$\Big(\neg(\alpha_3) \rightarrow (b \geq d^3_{up})\Big) \vee \Big(\neg(\beta_3) \rightarrow (b \geq d^3_{down})\Big),$$
$$\Big(\neg(\alpha_4) \rightarrow (b \geq d^4_{up})\Big) \vee \Big(\neg(\beta_4) \rightarrow (b \geq d^4_{down})\Big),$$
$$\Big(\neg(\alpha_5) \rightarrow (b \geq d^5_{up})\Big) \vee \Big(\neg(\beta_5) \rightarrow (b \geq d^5_{down})\Big),$$
$$\Big(\neg(\alpha_6) \rightarrow (b \geq d^6_{up})\Big) \vee \Big(\neg(\beta_6) \rightarrow (b \geq d^6_{down})\Big).$$

References

[201] Nobel Media AB 2019. Prize announcement. nobelprize.org. https://www.nobelprize.org/prizes/economic-sciences/2012/prize-announcement/. Online; accessed 26 February 2019.

[ABCC07] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[ABG+16] Haris Aziz, Péter Biró, Serge Gaspers, Ronald de Haan, Nicholas Mattei, and Baharak Rastegari. Stable matching with uncertain linear preferences. *CoRR*, abs/1607.02917, 2016.

[Afa12] Mustafa Oguz Afacan. Group robust stability in matching markets. *Games and Economic Behavior*, 74(1):394–398, 2012.

[AG15] Itai Ashlagi and Yannai A. Gonczarowski. Dating strategies are not obvious. *CoRR*, abs/1511.00452, 2015.

[AIM07] David J. Abraham, Robert W. Irving, and David F. Manlove. Two algorithms for the student-project allocation problem. *Journal of Discrete Algorithms*, 5(1):73–90, 2007.

[APRS05] Atila Abdulkadiroğlu, Parag A. Pathak, Alvin E. Roth, and Tayfun Sönmez. The boston public school match. *American Economic Review*, 95(2):368–371, 2005.

[AS98] Atila Abdulkadiroğlu and Tayfun Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–701, 1998.

[B96] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[BBC11] Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

[BBV03] Miquel Bofill, Dıdac Busquets, and Mateu Villaret. Auction robustness through satisfiability modulo theories. In *Workshop on Agreement Technologies (WAT 2009)*, 2003.

[Bir48] G. Birkhoff. *Lattice Theory*. Colloquium publications. American Mathematical Society, 1948.

[BM11] Adrian Bondy and U.S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer London, 2011.

[BNKF98] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, January 1998.

[BNR08] Mani Bhushan, Sridharakumar Narasimhan, and Raghunathan Rengaswamy. Robust sensor network design for fault diagnosis. *Computers & Chemical Engineering*, 32(4):1067 – 1084, 2008.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM COMPUTING SURVEYS*, pages 268–308, 2003.

[BS15] Federico Barber and Miguel A. Salido. Robustness, stability, recoverability, and reliability in constraint satisfaction problems. *Knowledge Information Systems*, 44(3):719–734, September 2015.

[BTGN09] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.

[CC16] Clément Carbonnel and Martin C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints*, 21(2):115–144, Apr 2016.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[CWSB14] Laura Climent, Richard J. Wallace, Miguel A. Salido, and Federico Barber. Robustness and stability in constraint programming under dynamism and uncertainty. *J. Artif. Intell. Res.*, 49:49–78, 2014.

[DB10] Marco Dorigo and Mauro Birattari. *Ant Colony Optimization*, pages 36–39. Springer US, Boston, MA, 2010.

[DB13] Joanna Drummond and Craig Boutilier. Elicitation and approximately stable matching with partial preferences. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 97–105. AAAI Press, 2013.

[DBS13] Ewa Drgas-Burchardt and Zbigniew Switalski. A number of stable matchings in models of the gale–shapley type. *Discrete Applied Mathematics*, 161(18):2932 – 2936, 2013.

[DH98] Raphaël Dorne and Jin-Kao Hao. A new genetic local search algorithm for graph coloring. In Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature — PPSN V*, pages 745–754, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[DH09] Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, pages 403–424. 2009.

[Die10] Reinhard Diestel. *Graph Theory (5th Edition)*. Graduate texts in mathematics. Springer, 2010.

[DPK13] Jaspal Singh Dhillon, Suresh Purini, and Sanidhya Kashyap. Virtual machine coscheduling: A game theoretic approach. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Util-*

ity and Cloud Computing, pages 227–234. IEEE Computer Society, 2013.

[DSOI05] Emir Demirović, Nicolas Schwind, Tenda Okimoto, and Katsumi Inoue. Recoverable team formation: Building teams resilient to change. Berlin, Heidelberg, 2005.

[DSOI18] Emir Demirović, Nicolas Schwind, Tenda Okimoto, and Katsumi Inoue. Recoverable team formation: Building teams resilient to change. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pages 1362–1370, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[EO18] Guillaume Escamocher and Barry O'Sullivan. Three-dimensional matching instances are rich in stable matchings. In Willem-Jan van Hoeve, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 182–197, Cham, 2018. Springer International Publishing.

[FD94] Daniel Frost and Rina Dechter. Dead-end driven learning. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 294–300, 1994.

[Fed92] Tomás Feder. A new fixed point approach for stable networks and stable marriages. *J. Comput. Syst. Sci.*, 45(2):233–284, 1992.

[FH00] Antonio J. Fernández and Patricia M. Hill. A comparative study of eight constraint programming languages over the boolean and finite domains. *Constraints*, 5(3):275–301, Jul 2000.

[Fie94] C-N Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, 51(3):243–267, 1994.

[FIM07] Tamás Fleiner, Robert W. Irving, and David Manlove. Efficient algorithms for generalized stable marriage and roommates problems. *Theor. Comput. Sci.*, 381(1-3):162–176, 2007.

[FM96] Bernd Freisleben and Peter Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman prob-

lems. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 616–621. IEEE, 1996.

[GI89] Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, USA, 1989.

[GI06] Fabrizio Grandoni and Giuseppe F. Italiano. Algorithms and constraint programming. In *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, pages 2–14, 2006.

[GILS87] Dan Gusfield, Robert Irving, Paul Leather, and Michael Saks. Every finite distributive lattice is a set of stable matchings for a small stable marriage instance. *Journal of Combinatorial Theory, Series A*, 44(2):304 – 309, 1987.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GJS+14] James Gosling, Bill Joy, Guy L. Steele, Gilad Bracha, and Alex Buckley. *The Java Language Specification, Java SE 8 Edition*. Addison-Wesley Professional, 1st edition, 2014.

[GMT14] Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471 – 483, 2014.

[GN07] Eugene Goldberg and Yakov Novikov. Berkmin: A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12):1549 – 1561, 2007. SAT 2001, the Fourth International Symposium on the Theory and Applications of Satisfiability Testing.

[GP10] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[GPP06] Ian P. Gent, Karen E. Petrie, and Jean-François Puget. Symmetry in constraint programming. In *Foundations of Artificial Intelligence*, volume 2, pages 329–376. Elsevier, 2006.

[GPR98] Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy. Supermodels and robustness. In *In AAAI/IAAI*, pages 334–339, 1998.

[GS62] David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[GSB+15] Yunan Gu, Walid Saad, Mehdi Bennis, Merouane Debbah, and Zhu Han. Matching theory for future wireless networks: Fundamentals and applications. *IEEE Communications Magazine*, 53(5):52–59, 2015.

[GSOS17a] Begum Genc, Mohamed Siala, Barry O'Sullivan, and Gilles Simonin. Finding robust solutions to stable marriage. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 631–637, 2017.

[GSOS17b] Begum Genc, Mohamed Siala, Barry O'Sullivan, and Gilles Simonin. Finding robust solutions to stable marriage. *CoRR*, abs/1705.09218, 2017.

[GSOS17c] Begum Genc, Mohamed Siala, Barry O'Sullivan, and Gilles Simonin. Robust stable marriage. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 4925–4926, 2017.

[GSSO17a] Begum Genc, Mohamed Siala, Gilles Simonin, and Barry O'Sullivan. On the complexity of robust stable marriage. In *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part II*, pages 441–448, 2017.

[GSSO17b] Begum Genc, Mohamed Siala, Gilles Simonin, and Barry O'Sullivan. On the complexity of robust stable marriage. *CoRR*, abs/1709.06172, 2017.

[GSSO19] Begum Genc, Mohamed Siala, Gilles Simonin, and Barry O'Sullivan. Complexity study for the robust stable marriage problem. *Theoretical Computer Science*, 775:76 – 92, 2019.

[Heb07] Emmanuel Hebrard. *Robust solutions for constraint satisfaction and optimisation under uncertainty*. PhD thesis, University of New South Wales, 2007.

[HHOW05] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In *AAAI*, volume 5, pages 372–377, 2005.

[HHW04a] Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. Robust solutions for constraint satisfaction and optimization. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 186–190, 2004.

[HHW04b] Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. Super solutions in constraint programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, First International Conference, CPAIOR 2004, Nice, France, April 20-22, 2004, Proceedings*, pages 157–172, 2004.

[HO04] Alan Holland and Barry O'Sullivan. 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, 2004.

[HO05a] Alan Holland and Barry O'Sullivan. 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, 2005.

[HO05b] Alan Holland and Barry O'Sullivan. 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, 2005.

[Hoe14] Benoit Hoessen. *Solving the Boolean satisfiability problem using the parallel paradigm*. PhD thesis, 2014. Thèse de doctorat dirigée par Audemard, Gilles Informatique Artois 2014.

[Hol92] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[Hor51] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

[HS96] Pascal Van Hentenryck and Vijay Saraswat. Strategic directions in constraint programming. *ACM Comput. Surv.*, 28(4):701–726, December 1996.

[HS17] Emmanuel Hebrard and Mohamed Siala. Explanation-based weighted degree. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 167–175. Springer, 2017.

[HW05] Emmanuel Hebrard and Toby Walsh. 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, Proceedings*, page 848, 2005.

[IL86a] Robert W. Irving and Paul Leather. The complexity of counting stable marriages. *SIAM J. Comput.*, 15(3):655–667, 1986.

[IL86b] Robert W. Irving and Paul Leather. The complexity of counting stable marriages. *SIAM J. Comput.*, 15(3):655–667, August 1986.

[IMS08] Robert W. Irving, David F. Manlove, and Sandy Scott. The stable marriage problem with master preference lists. *Discrete Applied Mathematics*, 156(15):2959 – 2977, 2008.

[Irv85] Robert W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6(4):577 – 595, 1985.

[IS16] Jonas Ide and Anita Schöbel. Robustness for uncertain multiobjective optimization: A survey and analysis of different concepts. *OR Spectr.*, 38(1):235–271, January 2016.

[Jac16] Royi Jacobovic. Perturbation robust stable matching. *CoRR*, abs/1612.08118, 2016.

[Kar72] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[KBG07] Prakash R. Kotecha, Mani Bhushan, and Ravindra D. Gudi. Constraint programming based robust sensor network design. *Industrial & Engineering Chemistry Research*, 46(18):5985–5999, 2007.

[KBG08] Prakash R. Kotecha, Mani Bhushan, and Ravindra D. Gudi. Design of robust, reliable sensor networks using constraint programming. *Computers & Chemical Engineering*, 32(9):2030 – 2049, 2008. Networked and Complex Systems S.I.

[KGW17] Anna R. Karlin, Shayan Oveis Gharan, and Robbie Weber. A simply exponential upper bound on the maximum number of stable matchings. *CoRR*, abs/1711.01032, 2017.

[Knu76] Donald E. Knuth. *Mariages stables et leurs relations avec d'autres problèmes combinatoires: introduction à l'analyse mathématique des algorithmes*. Collection de la Chaire Aisenstadt. Presses de l'Université de Montréal, 1976.

[Koj11] Fuhito Kojima. Robust stability in matching markets. *Theoretical Economics*, 6(2):257–267, 2011.

[KP94] Antoon Kolen and Erwin Pesch. Genetic local search in combinatorial optimization. *Discrete Applied Mathematics*, 48(3):273 – 284, 1994.

[KV06] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer-Verlag Berlin Heidelberg, 2006.

[LL12] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193 – 2196, 2012.

[LMS03] Helena R. Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[LMV+07] Dmitry Lebedev, Fabien Mathieu, Laurent Viennot, Anh-Tuan Gai, Julien Reynier, and Fabien De Montgolfier. On using matching theory to understand p2p network design. In *INOC 2007, International Network Optimization Conference*, 2007.

[LW05] Hui Li and Brian Williams. Generalized conflict learning for hybrid discrete/linear optimization. In *International Conference on Principles and Practice of Constraint Programming*, pages 415–429. Springer, 2005.

[Man13] David Manlove. *Algorithmics Of Matching Under Preferences*. Theoretical computer science. World Scientific Publishing, 2013.

[MF97] Peter Merz and Bernd Freisleben. Genetic local search for the tsp: new results. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pages 159–164, April 1997.

[MH12] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In Nicolas Beldiceanu, Narendra Jussien, and Éric Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 228–243, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[ML18] Vijay Menon and Kate Larson. Robust and approximately stable marriages under partial information. *CoRR*, abs/1804.09156, 2018.

[MS91] Bernard M.E. Moret and Henry D. Shapiro. *Algorithms from P to NP: Design & efficiency*. Algorithms from P to NP. Benjamin/Cummings, 1991.

[Mur11] K.G. Murty. *Optimization for Decision Making: Linear and Quadratic Models*. International Series in Operations Research & Management Science. Springer US, 2011.

[MV18] Tung Mai and Vijay V. Vazirani. Finding stable matchings that are robust to errors in the input. *CoRR*, abs/1804.00553, 2018.

[NRTV07] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[O'M07] Gregg O'Malley. *Algorithmic aspects of stable matching problems*. PhD thesis, University of Glasgow, 2007.

[Ped03] P. Pedregal. *Introduction to Optimization*. Texts in Applied Mathematics. Springer New York, 2003.

[PFL16] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[Pit89] Boris Pittel. The average number of stable matchings. *SIAM J. Discret. Math.*, 2(4):530–549, November 1989.

[Pit93] Boris Pittel. The" stable roommates" problem with random preferences. *The Annals of Probability*, pages 1441–1477, 1993.

[Pol05] Nicola Policella. Scheduling with uncertainty: A proactive approach using partial order schedules. *AI Commun.*, 18(2):165–167, 2005.

[Pre08] Steven David Prestwich. The relation between complete and incomplete search. In *Hybrid Metaheuristics, An Emerging Approach to Optimization*, pages 63–83. 2008.

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[Rou10] Tim Roughgarden. Algorithmic game theory. *Commun. ACM*, 53(7):78–86, July 2010.

[Roy01] Amitabha Roy. *Symmetry Breaking and Fault Tolerance in Boolean Satisfiability*. PhD thesis, Department of Computer and Information Science. University of Oregon, 2001. AAI3024528.

[Roy06] Amitabha Roy. Fault tolerant boolean satisfiability. *J. Artif. Intell. Res.*, 25:503–527, 2006.

[RPB10] Günther R. Raidl, Jakob Puchinger, and Christian Blum. *Metaheuristic Hybrids*, pages 469–496. Springer US, Boston, MA, 2010.

[RSÜ04] Alvin E Roth, Tayfun Sönmez, and M Utku Ünver. Kidney exchange. *The Quarterly Journal of Economics*, 119(2):457–488, 2004.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[SAFP14] Shaul Salomon, Gideon Avigad, Peter J Fleming, and Robin C Purshouse. Active robust optimization: Enhancing robustness to uncertain environments. *IEEE transactions on cybernetics*, 44(11):2221–2231, 2014.

[SBI12] Miguel A. Salido, Federico Barber, and Laura Paola Ingolotti. Robustness for a single railway line: Analytical and simulation methods. *Expert Syst. Appl.*, 39(18):13305–13327, 2012.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[SD99] Thomas Stützle and Marco Dorigo. Aco algorithms for the traveling salesman problem. *Evolutionary algorithms in engineering and computer science*, pages 163–183, 1999.

[SD08] Dan A. Simovici and Chabane Djeraba. *Mathematical Tools for Data Mining - Set Theory, Partial Orders, Combinatorics*, page 79. Advanced Information and Knowledge Processing. Springer, 2008.

[SF94] Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, editor, *Principles and Practice of Constraint Programming*, pages 10–20, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[SFS+14] Peter J Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008–2013. *AI Magazine*, 35(2):55–60, 2014.

[Sia15] Mohamed Siala. *Search, propagation, and learning in sequencing and scheduling problems*. Theses, INSA de Toulouse, May 2015.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation, Second Edition*, volume 2. Thomson Course Technology Boston, 2006.

[SKM+15] John Sichi, Joris Kinable, Dimitrios Michail, Barak Naveh, and Contributors. Jgrapht - graph algorithms and data structures in java (version 0.9.1). http://www.jgrapht.org/, 2015.

[SO17] Mohamed Siala and Barry O'Sullivan. Rotation-based formulation for stable matching. In *Principles and Practice of Constraint Pro-*

gramming - 23rd International Conference, CP 2017, Melbourne, Australia, August 28 - September 1, 2017, Proceedings, 2017.

[Soy73] A. L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

[SSDS02] Goran Stojkovič, François Soumis, Jacques Desrosiers, and Marius M. Solomon. An optimization model for a real-time flight scheduling problem. *Transportation Research Part A: Policy and Practice*, 36(9):779 – 788, 2002.

[ST16] Seçil Sözüer and Aurélie C. Thiele. The state of robust optimization. In *Robustness Analysis in Decision Aiding, Optimization, and Analytics*, pages 89–112. Springer International Publishing, Cham, 2016.

[Stü98] Thomas Stützle. *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. PhD thesis, TU Darmstadt, 1998.

[Suh97] Sang-Chul Suh. Double implementation in nash and strong nash equilibria. *Social Choice and Welfare*, 14(3):439–447, Jun 1997.

[Sus07] Gerald Jay Sussman. Building robust systems an essay. Massachusetts Institute of Technology, January 2007.

[Tal15] El-Ghazali Talbi. Hybrid metaheuristics for multi-objective optimization. *Journal of Algorithms & Computational Technology*, 9(1):41–63, 2015.

[TMW06] S. Armagan Tarim, Suresh Manandhar, and Toby Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, Jan 2006.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[TVTV14] Paolo Toth, Daniele Vigo, Paolo Toth, and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2014.

[UAB+91] Nico L. J. Ulder, Emile H. L. Aarts, Hans-Jürgen Bandelt, Peter J. M. van Laarhoven, and Erwin Pesch. Genetic local search algorithms for the traveling salesman problem. In Hans-Paul Schwefel and Reinhard Männer, editors, *Parallel Problem Solving from Nature*, pages 109–116, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

[vH01] Willem Jan van Hoeve. The alldifferent constraint: A survey. *CoRR*, cs.PL/0105015, 2001.

[vHK06] Willem-Jan van Hoeve and Irit Katriel. Chapter 6 - global constraints. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 169 – 208. Elsevier, 2006.

[Voß01] Stefan Voß. Meta-heuristics: The state of the art. In Alexander Nareyek, editor, *Local Search for Planning and Scheduling*, pages 1–23, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[VT99] Christos Voudouris and Edward Tsang. Guided local search and its application to the traveling salesman problem. *European journal of operational research*, 113(2):469–499, 1999.

[W+99] Toby Walsh et al. Search in a small world. In *IJCAI*, volume 99, pages 1172–1177. Citeseer, 1999.

[Wil86] Robin J. Wilson. *Introduction to Graph Theory*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[WKm15] Thomas Williams, Colin Kelley, and many others. Gnuplot 5.0: an interactive plotting program. http://www.gnuplot.info/, June 2015.

[Woo06] Jim Woodward. Some varieties of robustness. *Journal of Economic Methodology*, 13(2):219–240, 2006.

[XCM09] Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. *J. Mach. Learn. Res.*, 10:1485–1510, December 2009.

[XHQC09] Y. Xu, G.H. Huang, X.S. Qin, and M.F. Cao. Srccp: A stochastic robust chance-constrained programming model for municipal solid waste management under uncertainty. *Resources, Conservation and Recycling*, 53(6):352 – 363, 2009.

[YMLC16] Cheng-Hong Yang, Sin-Hua Moi, Yu-Da Lin, and Li-Yeh Chuang. Genetic algorithm combined with a local search method for identifying susceptibility genes. *Journal of Artificial Intelligence and Soft Computing Research*, 6(3):203 – 212, 2016.

[YN97] Takeshi Yamada and Ryohei Nakano. Job shop scheduling. *IEE control Engineering series*, pages 134–134, 1997.

[Yua09] Duojia Yuan. *Flight Delay-cost Simulation Analysis and Schedule Optimization: A Simulation Methodology to Estimate Flight Delay and Delay Propagation*. VDM Publishing, 2009.