SOLVING HARD STABLE MARRIAGE PROBLEMS USING LOGIC-BASED METHODS

by
SELİN EYÜPOĞLU

**Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science**

**Sabancı University
July 2022**

ABSTRACT

SOLVING HARD STABLE MARRIAGE PROBLEMS USING LOGIC-BASED METHODS

SELİN EYÜPOĞLU

Computer Science and Engineering M.Sc. Thesis, July 2022

Thesis Supervisor: Prof. Dr. Esra Erdem

Keywords: stable marriage problems, declarative problem solving, answer set programming, propositional satisfiability, constraint programming

Matching problems have been studied in economics, starting with the seminal paper of Gale and Shapley, which has led to a Nobel Prize in 2012, utilizing game theory methods with the goal of a mechanism design. One of the well-known matching problems is the Stable Marriage Problem (SM). In SM, for a set of n men and n women, we are given the preferences of individuals: for each man, a complete ranking of the women is specified as preferred partners; similarly, for each woman, a complete ranking of the men is specified as preferred partners. The goal is to marry all men and women (i.e., to find n couples) in such a way that marriages are stable: no man and woman in different couples prefer each other to their partners.

We consider a variant of SM, called SMTI, where rankings may be incomplete (i.e., some partners are not acceptable) or may include ties (i.e., some partners are preferred equally). We investigate three hard variants of SMTI, that aim to compute optimal stable matchings with respect to different measures of fairness: Sex-Equal SMTI (maximizes the equality of satisfaction among sexes), Egalitarian SMTI (maximizes the total satisfaction of the preferences of all agents), and Max Cardinality SMTI (minimizes the number of singles).

We introduce a suite of novel declarative methods to solve these hard variants of SMTI, using Answer Set Programming (ASP), Constraint Programming (CP), and Propositional Satisfiability (SAT). We empirically evaluate the scalability of methods over randomly generated instances, as the probability of incompleteness and probability of ties change. For Max Cardinality SMTI, we also compare these methods with the existing approaches based on Integer Linear Programming (ILP) and

Local Search (including Hill-Climbing and Genetic Algorithms). We observe that the declarative methods (ASP, ILP, CP, SAT) are more promising compared to the local search algorithms as the problems get harder with more ties and incompleteness.

ÖZET

MANTIK TEMELLİ YÖNTEMLER KULLANARAK ZOR İSTİKRARLI EVLİLİK PROBLEMLERİNİ ÇÖZME

SELİN EYÜPOĞLU

Bilgisayar Bilimi ve Mühendisliği Yüksek Lisans Tezi, Temmuz 2022

Tez Danışmanı: Prof. Dr. Esra Erdem

Anahtar Kelimeler: istikrarlı evlilik problemi, bildirimsel problem çözme, çözüm kümesi programlama, önermesel gerçeklenebilirlik, kısıt programlama

Bu tez çalışmasında, sıralamaların eksik olabileceği (yani bazı partnerlerin tercih edilmediği) veya denklik içerebileceği (yani bazı partnerlerin eşit tercih edildiği), SMTI (eksik ve denklik içeren sıralamalarla istikrarlı evlilik problemi) adı verilen bir SM varyantını ele alıyoruz. Farklı adalet ölçülerine göre optimum istikrarlı eşleşmeleri hesaplamayı amaçlayan üç SMTI varyantını araştırıyoruz: Cinsiyet Eşitlikçi SMTI (cinsiyetler arasında memnuniyet eşitliğini maksimize eder), Eşitlikçi SMTI (tüm kişilerin tercihlerinin toplam memnuniyetini maksimize eder) ve Maksimum Kardinalite SMTI (çiftlerin sayısını maksimize eder).

Çözüm kümesi programlaması, kısıtlı programlama ve önermesel gerçeklenebilirlik kullanarak SMTI'ın bu zor türevlerini çözmek için yeni bildirimsel yöntemler sunuyoruz. Yöntemlerin rastgele oluşturulmuş örnekler üzerinde ölçeklenebilirliğini deneysel olarak analiz ediyoruz. Maksimum Kardinalite SMTI için, bu yöntemleri aynı zamanda mevcut tamsayılı doğrusal programlama ve yerel arama yaklaşımlarıyla karşılaştırıyoruz. Problemler daha fazla denklik ve eksiklik içerdiğinde, bildirimsel yöntemlerin yerel arama algoritmalarına göre hesaplama açısından diğer yaklaşımlardan genel olarak daha iyi olduğunu gözlemliyoruz.

ACKNOWLEDGEMENTS

First, I would like to thank my family for their endless love, patience and support. Without them, I would not be able to complete my work.

I am very grateful to have Prof. Dr. Esra Erdem as my thesis advisor as she has always been attentive and supportive during my thesis work. Her guidance has been substantially impactful on my work and personal growth. I would like to thank her for having confidence in me.

I would like to thank my friends who have always been so kind and supportive. I am also grateful for the fellow graduate students who have helped me in the preparation of my thesis work. I would like to thank Müge and Aysu for their support, helpful research discussions and the good times we had in the lab.

Last but not least, I am forever indebted to my mom who has been my best friend and my biggest supporter. Her strength, generosity and intellect have inspired me to become the person I am today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Matching problems have been studied in economics, starting with the seminal paper of Gale & Shapley (1962), which has led to a Nobel Prize in 2012, utilizing game theory methods with the goal of a mechanism design. Matching problems are about markets where individuals are matched with individuals, firms, or items, typically across two sides, as in employment (Roth & Sotomayor, 1992) (e.g., who works at which job), kidney donation (e.g., who receives which transplantable organ) (Manlove & O'Malley, 2014; Roth, Sonmez & Utku Unver, 2005), and marriages (Gale & Shapley, 1962; Iwama, Manlove, Miyazaki & Morita, 1999) (e.g., who marries with whom). In each problem, preferences of individuals, firms, or items are given, possibly along with other information (e.g., the quotas of the universities in university entrance) (Alkan & Gale, 2003; Alkan & Moulin, 2003).

We introduce a suite of novel declarative methods to solve these hard variants of SMTI, using Answer Set Programming (ASP) (Brewka, Eiter & Truszczynski, 2016; Lifschitz, 2002; Marek & Truszczyński, 1999; Niemelä, 1999; Simons, Niemelä & Soininen, 2002), Constraint Programming (CP) (Jaffar & Lassez, 1987; Rossi et al., 2006; Van Hentenryck, 1989), and Propositional Satisfiability (SAT) (Biere et al., 2009; Gomes, Kautz, Sabharwal & Selman, 2008).

Our contributions can be summarized as follows:

- We introduce novel methods to solve SMTI and all its hard variants using ASP.
- Based on the CP models of SMTI and Max Cardinality SMTI by Gent & Prosser (2002), we introduce novel methods to solve Egalitarian and Sex-Equal SMTI. We implement all these methods for Google-OR CP-SAT and Choco solvers.
- We adapt the SAT formulation of Drummond, Perrault & Bacchus (2015) for hospital-resident problems with couples to solve SMTI. Based on the updated SAT formulation, we introduce novel methods to solve Max Cardinality SMTI

and Egalitarian SMTI using Partial Weighted MaxSAT.

- We implement these methods using the state-of-the-art solvers: ASP solver Clingo, ASP solver Cmodels with SAT solver ZChaff, SAT solver Lingeling, MaxSAT solver CashWMaxsat, CP solvers Google-OR CP-SAT and Choco.

- We empirically compare several methods to solve Egalitarian and Sex-Equal SMTI using ASP.

- We compare ASP with propositional satisfiability (SAT) (Biere et al., 2009; Gomes et al., 2008) over randomly generated SMTI instances. For a comparison of ASP with SAT, we adapt the ASP implementation of SMTI for Cmodels (Giunchiglia, Lierler & Maratea, 2004) that utilizes the SAT solver ZChaff (Moskewicz, Madigan, Zhao, Zhang & Malik, 2001) to compute solutions. We also use the updated SAT formulation to solve SMTI, and use the SAT solver Lingeling (Biere, 2010) to compute solutions.

- We repeat the experiments conducted by Gent & Prosser (2002) by implementing their CP models for finding a complete stable matching and a largest stable matching. We use a set of randomly generated instances and state-of-the-art CP solvers Choco and Google-OR CP-SAT to compute solutions.

- We empirically compare these methods comprehensively over randomly generated instances of different sizes, in terms of their scalability, as the probabilities of incompleteness and ties change. For this purpose, we implement the random instance generator proposed by Gent & Prosser (2002).

- We also compare these methods with the three existing approaches for Max Cardinality SMTI: Kwanashie & Manlove (2014)'s method based on Integer Linear Programming (ILP) (Kantorovich, 1960), Gelain, Pini, Rossi, Venable & Walsh (2013)'s method based on Hill Climbing (Lin & Kernighan, 1973; Selman & Gomes, 2006), Haas (2021)'s method based on Genetic Algorithms (Holland, 1992). For these comparisons, we use the existing implementations of ILP methods for ILP solver Gurobi and Google-OR MIP solver. We revise the implementation for Gurobi to enhance its performance.

- We extend our studies further by considering sticky-stability (Afacan, Aliogullari & Barlo, 2016) for SMTI, update our ASP method for stickiness and present our experimental results.

Some of these methods are briefly discussed in our short paper (Eyupoglu, Fidan, Gulesen, Izci, Teber, Yilmaz, Alkan & Erdem, 2021).

In the rest of the thesis, first we introduce the definitions of the problems (Section 2.1) then we introduce our solutions using ASP (Section 3), CP (Section 4), SAT (Section 5). After we present the results of our empirical evaluations (Section 6) and discuss related work (Section 8), we conclude (Section 9). We also present our extended studies (Section 7).

2. STABLE MARRIAGE PROBLEMS

One of the well-known matching problems is the Stable Marriage Problem (SM). In SM, for a set of n men and n women, we are given the preferences of individuals: for each man, a complete ranking of the women is specified as preferred partners; similarly, for each woman, a complete ranking of the men is specified as preferred partners. The goal is to marry all men and women (i.e., to find n couples) in such a way that marriages are stable: no man and woman in different couples prefer each other to their partners.

We consider a variant of SM, called SMTI, where rankings may be incomplete (i.e., some partners are not acceptable) or may include ties (i.e., some partners are preferred equally).

Table 2.1 Summary of the complexities of SM problems

Problem	Complexity
SM	P (Gale & Shapley, 1962)
SMI	P (Gusfield & Irving, 1989)
SMT	P (Manlove, 2014, Thm 3.2)
SMTI	P (Irving, 1994)
SMTI (strong)	P (Irving, 1994)
SMTI (weak)	P (Gärdenfors, 1975, Thm 1)
Egalitarian SMI	P (Manlove, 2014, Thm 1.14)
Egalitarian SMT (and thus Egalitarian SMTI)	NP-hard (Manlove, Irving, Iwama, Miyazaki & Morita, 2002, Thm 7)
Maximum Cardinality SMTI	NP-hard (Manlove et al., 2002, Lemma 1)
Sex-Equal SMTI	NP-hard (Kato, 1993, Thm 2.1)

We investigate three hard variants of SMTI (Kato, 1993; Manlove et al., 2002), that aim to compute optimal stable matchings with respect to different measures of fairness: sex-equality (preferences of men and women are considered to be equally important), egalitarian (preferences of every individual are considered to be equally important), maximum cardinality (minimizes the number of singles). The complexities of these variants are presented in Table 2.1.

2.1 SMTI

The Stable Marriage problem with Ties and Incomplete lists (SMTI) is defined by a set M of men, a set W of women, for each man $x \in M$ a partial ordering $\succ_x$ over $W_x \subseteq W$ where incomparability is transitive, and for each woman $y \in W$ a partial ordering $\succ_y$ over $M_y \subseteq M$ where incomparability is transitive.

Let $mrank : M \times W \mapsto \{1, \ldots, |W|\}$ be a partial function such that $mrank(x, y) = r$ represents that a woman y is man x's rth preferred choice with respect to $\succ_x$ and $wrank : W \times M \mapsto \{1, \ldots, |M|\}$ be a partial function such that $wrank(y, x) = r$ represents that a man x is woman y's rth preferred choice with respect to $\succ_y$.

A man x is *acceptable* to a woman y if $wrank(x, y)$ is defined. Similarly, a woman y is acceptable for a man x if $mrank(x, y)$ is defined.

A *matching* for a given SMTI instance is a partial function $\mu : M \mapsto W$. A man x is single if $\mu(x)$ is undefined and a woman y is single if $\mu^{-1}(y)$ is undefined.

A pair (x, y) of a man and a woman is called a *blocking pair* for a matching μ if the following conditions hold:

A1 x and y are acceptable to each other,

A2 x and y are not married to each other (i.e., $\mu(x) \neq y$),

A3 (a) x and y are both single,

(b) $mrank(x, y) < mrank(x, \mu(x))$ and y is single,

(c) $wrank(y, x) < wrank(y, \mu^{-1}(y))$ and x is single or

(d) $mrank(x, y) < mrank(x, \mu(x))$ and $wrank(y, x) < wrank(y, \mu^{-1}(y))$.

A matching for SMTI is called *stable* if it is not blocked by any pair of agents. Note that we consider weakly stable matchings. It is assumed that marriage to an acceptable partner is preferred over being single.

We consider three hard variants of SMTI (Kato, 1993; Manlove et al., 2002), that aim to compute optimal stable matchings with respect to different measures of fairness: sex-equality, egalitarian and maximum cardinality.

2.2 Egalitarian SMTI

Egalitarian SMTI maximizes the total satisfaction of the preferences of all agents. Since the preferred agents have lower rankings, this total satisfaction is maximized when the sum of ranks of all agents is minimized.

Let μ be a matching and $\mathcal{M}$ denote the set of matchings in a given problem. For every man $x \in M$, the satisfaction $c_\mu(x)$ of x's preferences is defined with respect to μ as follows: $c_\mu(x)=R$ if $mrank(x, \mu(x))=R$. Similarly, for every woman $y \in W$, we define the satisfaction $c_\mu(y)=R$ if $wrank(y, \mu^{-1}(y))=R$. Then, the total satisfaction of preferences of all agents is defined as follows: $c(\mu) = \sum_{x \in M \cup W} c_\mu(x)$. Then, a matching $\mu \in \mathcal{M}$ with the minimum $c(\mu)$ is *egalitarian.*

Egalitarian SMTI is NP-Hard (Manlove et al., 2002, Thm 7).

2.3 Sex-Equal SMTI

Sex-Equal SMTI maximizes the equality of satisfaction among sexes. We define the sex equality by the following cost function $c(\mu)=|\sum_{x \in M} c_\mu(x) - \sum_{y \in W} c_\mu(y)|$. Then, a matching $\mu \in \mathcal{M}$ with the minimum $c(\mu)$ is *sex-equal.*

Sex-Equal SMTI is NP-Hard (Kato, 1993, Thm 2.1).

2.4 Max Cardinality SMTI

Max Cardinality SMTI maximizes the number of matched pairs. A matching $\mu \in \mathcal{M}$ is a *maximum cardinality matching* if it is a matching that maximizes $|\mu|$. The number of matched pairs is maximized when the number of singles is minimized.

Max Cardinality SMTI is NP-Hard (Manlove et al., 2002, Lemma 1).

3. SOLVING HARD SMTI PROBLEMS USING ANSWER SET PROGRAMMING

We introduce novel methods for solving SMTI and its hard variants using Answer Set Programming (ASP).

3.1 Answer Set Programming

Answer Set Programming (ASP) (Brewka et al., 2016) is a form of declarative programming which is useful in knowledge-intensive applications. ASP is based on the stable model (answer set) semantics of logic programming Gelfond & Lifschitz (2000). We consider ASP programs that consist of rules of the form

$$Head \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$$

where $n \geq m \geq 0$, *Head* is an atom or $\bot$, and each A_i is an atom. A rule is called a *fact* if $m = n = 0$ and a *(hard) constraint* if *Head* is $\bot$.

Cardinality expressions are special constructs of the form $l\{A_1, \ldots, A_k\}u$ where each A_i is an atom and l and u are nonnegative integers denoting the lower and upper bounds Simons et al. (2002). Programs using these constructs can be viewed as abbreviations for programs that consist of rules of the form. Such an expression describes the subsets of the set $\{A_1, \ldots, A_k\}$ whose cardinalities are at least l and at most u.

Cardinality expressions can be used in heads of rules; then they generate many answer sets whose cardinality is at least l and at most u. For instance, the answer sets for the program $2\{p_1, ..., p_7\}4 \leftarrow$ are subsets of the set $\{p_1, ..., p_7\}$ whose cardinality is at least 2 and at most 4.

Schematic variables A group of rules that follow a pattern can be often described in a compact way using "schematic variables". For instance, the cardinality expression $1\{p_1,\ldots,p_7\}1$ can be represented as $1\{p(i) : \mathit{index}(i)\}1$, along with a definition of $\mathit{index}(i)$ that describes the ranges of variables: $\mathit{index}(1..7)$.

Weighted weak constraints with priorities The ASP programs can be augmented with "weak constraints"—expressions of the following form (Buccafurri, Leone & Rullo, 2000):

$$\stackrel{\sim}{\leftarrow} \mathit{Body}(t_1,...,t_n)[w@p,t_1,...,t_n]$$

Here, $\mathit{Body}(t_1,...,t_n)$ is a formula (as in the body of a rule) with the terms $t_1,...,t_n$. Intuitively, whenever an answer set for a program satisfies $\mathit{Body}(t_1,...,t_n)$, the tuple $\langle t_1,...,t_n\rangle$ contributes a cost of w to the total cost function of priority p. The ASP solver tries to find an answer set with the minimum total cost. For instance, the following weak constraint

$$\stackrel{\sim}{\leftarrow} p(i),p(i+1),\mathit{index}(i),\mathit{index}(i+1)[1@2,i]$$

instructs CLINGO to compute an answer set that does not include both $p(i)$ and $p(i+1)$, if possible. However, if CLINGO cannot find such an answer set, it is allowed to compute an answer set with these atoms $p(i)$ and $p(i+1)$ but with an additional cost of 1 per each such i. Weak constraints are considered by CLINGO according to their priorities.

3.2 Solving SMTI using ASP

Input of an SMTI instance $\langle M,W,\mathit{mrank},\mathit{wrank}\rangle$ is formalized in ASP by a set F_I of facts using atoms of the forms $\mathit{man}(x)$ ("x is a man in M"), $\mathit{woman}(y)$ ("y is a woman in W"), $\mathit{mrank}(x,y,r)$ (i.e., $\mathit{mrank}(x,y)=r$) and $\mathit{wrank}(y,x,r)$ (i.e., $\mathit{wrank}(y,x)=r$). In the ASP formulation P of SMTI, the variables x, $x1$ denote men in M and y, $y1$ denote women in W.

The first pair of rules of the program P characterize a set of individuals of the opposite set for each man and woman who they prefer over being single:

$$\mathit{maccept}(x,y) \leftarrow \mathit{mrank}(x,y,r).$$
$$\mathit{waccept}(y,x) \leftarrow \mathit{wrank}(y,x,r).$$

and the concept of mutual acceptability:

$$acceptable(x,y) \leftarrow maccept(x,y), waccept(y,x).$$

Preferences of man x (i.e., x prefers y to $y1$) and woman y are defined (i.e., y prefers x to $x1$) in terms of rankings.

$$mprefer(x,y,y1) \leftarrow mrank(x,y1,r), mrank(x,y,r1), r > r1.$$
$$wprefer(y,x,x1) \leftarrow wrank(y,x1,r), wrank(y,x,r1), r > r1.$$

We define a matching between men and women where both parties find each other acceptable with the cardinal constraint of 1 for each man.

$$\{marry(x,y) : acceptable(x,y)\}1 \leftarrow man(x).$$

In order to guarantee that a woman is not matched to more than one man, we use the following constraint:

$$\leftarrow \{marry(x,y) : man(x)\} > 1, woman(y).$$

Individuals who stay single under the represented matching by *marry* atoms are described by the *msingle* and *wsingle* atoms.

$$msingle(x) \leftarrow man(x), \{marry(x,y) : woman(y)\}0.$$
$$wsingle(y) \leftarrow woman(y), \{marry(x,y) : man(x)\}0.$$

To establish stability, we refer to conditions A1–A3. The following set of constraints respectively describe and eliminate blocking pairs that are described by (a) – (d) of A3. Conditions A1 and A2 also hold for each constraint.

$$\leftarrow acceptable(x,y), msingle(x), wsingle(y).$$
$$\leftarrow wsingle(y), marry(x,y1), mprefer(x,y,y1), acceptable(x,y).$$
$$\leftarrow msingle(x), marry(x1,y), wprefer(y,x,x1), acceptable(x,y).$$
$$\leftarrow marry(x,y1), marry(x1,y), mprefer(x,y,y1), wprefer(y,x,x1).$$

Given the ASP formulation P whose rules are described above and the ASP description F_I of an SMTI instance I, the ASP solver Clingo generates a stable matching.

3.3 Solving Hard Variants of SMTI using ASP

To solve Sex-Equal SMTI, we add the following weak constraint to optimize sex equality:

$$\overset{\sim}{\leftarrow} t = \#sum\{r1 - r2, x, y : marry(x, y), mrank(x, y, r1), wrank(y, x, r2)\}.\ [|t|@1]$$

To solve Egalitarian SMTI, we add the following weak constraint to minimize the cost function:

$$\overset{\sim}{\leftarrow} marry(x, y), mrank(x, y, r1), wrank(y, x, r2).\ [r1 + r2@1, x, y]$$

To solve Max Cardinality SMTI, we add the following weak constraints to minimize the number of singles:

$$\overset{\sim}{\leftarrow} wsingle(x).\ [1@1, W, x]$$
$$\overset{\sim}{\leftarrow} msingle(y).\ [1@1, M, y]$$

Note that combinations of these optimizations are possible, by giving priorities to weak constraints: instead of specifying just the weight t of the weak constraints, we need to specify also their priorities p by an expression $w@p$.

4. SOLVING HARD SMTI PROBLEMS USING CONSTRAINT PROGRAMMING

We present existing and novel methods to solve SMTI and its hard variants using Constraint Programming (CP). We present Gent and Prosser's encoding for finding a complete matching for SMTI (Section 4.2) and the largest matching. After that, we introduce our methods for solving the hard variants of SMTI (Section 4.3).

4.1 Constraint Programming

A Constraint Satisfaction Problem (CSP) is described by a triple $P = \langle X, D, C \rangle$, where $X = \langle x_1, \ldots, x_k \rangle$ is a k-tuple of variables, $D = \langle D_1, \ldots, D_k \rangle$ is a k-tuple of corresponding domains such that $x_i \in D_i$, and $C = \langle C_1, \ldots C_t \rangle$ is a t-tuple of constraints. A constraint C_j is defined by a pair $\langle R_j, S_j \rangle$, such that R_j is a relation on the variables $S_j = \{s_1 \ldots s_m\} \subseteq \{x_1, \ldots x_k\}$. Specifically, $R_j \subseteq D_{s_i} \times \ldots D_{s_m}$. A solution to a CSP $P = \langle X, D, C \rangle$ is an k-tuple $A = \langle a_i, \ldots a_k \rangle$ such that $a_i \in D_i$ and all constraints in C are satisfied. Given a solution $A = \langle a_i, \ldots a_k \rangle$, a constraint $C_j = \langle R_j, S_j \rangle$ is satisfied if R_j holds when A is projected onto S_j (Freuder & Mackworth, 2006).

For optimization, based on an objective function, Constraint Optimization Problems (COPs) are considered. A COP is described by a CSP $P = \langle X, D, C \rangle$ and an objective function $f : D(x_1) \times \ldots D(x_k) \rightarrow \mathbb{Q}$. An optimal solution for a COP is a solution d to P which maximizes or minimizes $f(d)$ (van Hoeve & Katriel, 2006).

4.2 Gent & Prosser's CP Model for SMTI

Gent & Prosser (2002) introduce CP models for SMTI for finding a complete stable matching, if one exists, and finding the largest stable matching. We implement their models for Google OR-Tools using CP-SAT solver.

Firstly, we describe the CP model for SMTI for finding a complete matching. The model uses a variable m_i for each man $x_i \in M$, and w_j for each woman $y_j \in W$. For each variable m_i,

$$domain(m_i) = \{j : y_j \text{ is acceptable to } x_i\}.$$

For each w_j, $domain(w_j)$ is defined similarly.

For every pair (x_i, y_j) of a man $x_i \in M$ and a woman $y_j \in W$ that are acceptable to each other, the CP model includes the following constraints:

$$\neg(w_j = i' \wedge m_i = j) \qquad (x_{i'} \in M, i \neq i', x_{i'} \text{ is acceptable to } y_j),$$

$$\neg(m_i = j' \wedge w_j = i) \qquad (y_{j'} \in W, j \neq j', y_{j'} \text{ is acceptable to } x_i),$$

$$\neg(w_j = i' \wedge m_i = j') \quad (x_{i'} \in M, y_{j'} \in W, mrank(x_i, y_j) < mrank(x_i, y_{j'}), \\ wrank(y_j, x_i) < wrank(y_j, x_{i'})).$$

The first constraint above guarantees that a woman y is matched to exactly one man. The second constraint guarantees that a man x is matched to exactly one woman. The third constraint guarantees that there is no pair $(x, y) \in M \times W$ such that x is matched to a woman y' and y is matched to a man x' where x and y prefer each other to their current partners. In this way, the CP model ensures that there are no blocking pairs, hence the stability of the matching.

4.3 Solving Hard Variants of SMTI using CP

To solve Max Cardinality SMTI, we consider Gent & Prosser's CP model for finding the largest matching for SMTI. In their model, dummy man $dummy_m$ and dummy woman $dummy_w$ are introduced, such that each man x prefers all women on his

preference list to $dummy_w$ and each woman y prefers all men on her preference list to $dummy_m$. Note that an assignment to $dummy_m$ or $dummy_w$ denotes being single.

The CP model for SMTI for finding the largest matching uses a variable m_i for each man $x_i \in M$ and w_j for each woman $y_j \in W$. For each variable m_i,

$$domain(m_i) = \{j : y_j \text{ is acceptable to } x_i\} \cup \{dummy_w\}.$$

For each w_j, $domain(w_j)$ is defined similarly.

The constraints are identical to the constraints of the CP model for finding a complete matching such that M and W include dummy man and dummy woman. To solve Max Cardinality SMTI, we include a minimization statement with the following objective function:

$$|\{x_i \in M | m_i = dummy_w\}|.$$

To solve Egalitarian SMTI, for each pair (x_i, y_j) who find each other acceptable, a Boolean variable b_{ij} is introduced such that b_{ij} is true if and only if (x_i, y_j) are married. Then, the following objective function is minimized for Egalitarian optimization:

$$\sum_{(x_i,y_j)\in M\times W} (mrank(x_i, y_j) + wrank(y_j, x_i)) \times b_{ij}).$$

To solve Sex-Equal SMTI, we utilize b_{ij} variables and minimize the following objective function:

$$|\sum_{(x_i,y_j)\in M\times W} (mrank(x_i, y_j) \times b_{ij} - wrank(y_j, x_i) \times b_{ij})|.$$

5. SOLVING SMTI PROBLEMS USING SAT

We introduce novel methods to solve SMTI and its hard variants using Propositional Satisfiability (SAT). We present Drummond et al.'s formulation SAT-E for solving SMP-C (Section 5.2) and show how we adapt their formulation to solve SMTI. After that, we introduce our methods to solve Max Cardinality and Egalitarian SMTI (Section 5.3).

5.1 SAT

A *Boolean variable* x is a variable that takes a value from the set $\{0,1\}$. A *literal* is either a Boolean variable x (positive literal), or its negation $\neg x$ (negative literal). A *clause* is a disjunction over literals and a CNF formula conjunction over clauses. A truth assignment $I : X \mapsto \{0,1\}$ satisfies a clause $C = (c_1 \vee \cdots \vee c_k)$ if for any c_i, $I(c_i) = 1$ if c_i is a positive literal, or $I(c_i) = 0$ if c_i is a negative literal. A truth assignment satisfies a formula if it satisfies all its clauses. The formula φ is satisfiable if such an assignment exists. Given a formula φ, the SAT problem is to decide whether φ is satisfiable or not.

The MaxSAT problem is an optimisation problem related to SAT. The objective is to maximize the number of satisfied clauses (Li & Manyà, 2009).

The weighted MaxSAT problem is defined by a CNF formula $\varphi = (C_1 \wedge \cdots \wedge C_k)$ and a bounded positive number associated with each clause C_i. The objective is to find a satisfying assignment I that maximizes the sum of weights of satisfied clauses.

Given a CNF formula, the Partial MaxSAT problem allows clauses to be declared relaxable (soft) or non-relaxable (hard). The aim is to find an assignment that satisfies all the hard clauses and the maximum number of soft clauses (Li & Manyà, 2009).

The partial weighted MaxSAT problem is a combination of Partial MaxSAT and weighted MaxSAT that distinguishes between hard and soft clauses. An optimal solution to a partial weighted MaxSAT problem is a truth assignment that satisfies all hard clauses and maximizes the sum of weights of soft clauses.

5.2 Solving SMTI using SAT

Drummond et al. (2015) introduce a SAT formulation SAT-E for solving SMP-C (stable matching with couples). In SMP-C, residents are placed into hospital programs which have quotas. Both residents and hospitals report preferences over each other. Some doctors may form a couple and give their preferences as a couple. The preference lists may be incomplete. However, their definition do not take ties into account. The aim is to find a stable matching such that no doctor-hospital pair form a blocking pair.

Since SM is a restriction of SMP-C, such that there are no couples and the quota of each hospital is 1, their formulation can be adapted to solve SMTI.

A dummy woman $dummy_w$ is introduced and added to men's preference lists as the least preferred woman. Let $n+1$ be the assigned index to $dummy_w$ and let $W' = W \cup \{dummy_w\}$. Let $wAcc_i$ denote the set of women who are acceptable to $x_i \in M$, and let $mAcc_j$ denote the set of men who are acceptable to $y_j \in W$.

The modified SAT formulation uses two sets of variables.

- Man matching variables: $\{mmatch_i[j] | y_j \in W \wedge y_j \text{ is acceptable to } x_i\}$. Note that $mmatch_i[j]$ is true iff woman y_j is matched to x_i and $mmatch_i[n+1]$ is true if x_i is single.
- Woman matching variables:
  $\{wmatch_j[k] | y_j \in W \wedge (1 \leq k \leq max_{\{x_i \in mAcc_j\}} wrank(y_j, x_i))\}$. Note that $wmatch_j[k]$ is true iff a man x_i in such that $0 \leq wrank(y_j, x_i) \leq k$ have been matched to y_j.

Next, we present the clauses used in the modified formulation.

The following clauses are added to ensure that a man is matched to exactly one

woman, including $dummy_w$. For each $x_i \in M$, the following clauses are added:

$$\bigwedge_{y_j, y'_j \in W'} (\neg mmatch_i[j] \vee \neg mmatch_i[j']) \tag{5.1}$$

$$\bigvee_{w \in W'} mmatch_i[j]. \tag{5.2}$$

Next, the following set of formulae are added to ensure consistency of man matching and woman matching variables.

$$wmatch_j[0] \equiv \bigvee_{x_i \in M \text{ s.t. } wrank(y_j, x_i)=1} mmatch_i[j] \tag{5.3}$$

$$\begin{aligned} wmatch_j[k] \equiv \\ (wmatch_j[k-1] \wedge (\bigwedge_{x_i \in M \text{ s.t. } wrank(y_j,x_i)=k-1} \neg mmatch_i[j])) \bigvee \\ (\neg wmatch_j[k-1] \wedge (\bigvee_{x_i \in M \text{ s.t. } wrank(y_j,x_i)=k-1} mmatch_i[j])) \end{aligned} \tag{5.4}$$

Finally, we add formulae to ensure stability. For each man, $x_i \in M$ and for each $y_j \in wAcc_i$ if $|\{x'_i \in M | wrank(y_j, x'_i) = wrank(y_j, x_i)\}| = 1$:

- if $wrank(y_j, x_i) > 1$, the following formula is added

$$\bigwedge_{y_{j'} \in W \text{ s.t. } mrank(x_i, y_{j'}) \leq mrank(x_i, y_j)} \neg mmatch_i[j'] \rightarrow wmatch_j[wrank(y_j, x_i) - 1]. \tag{5.5}$$

- else, the following formula is added

$$\bigwedge_{y_{j'} \in W \text{ s.t. } mrank(x_i, y_{j'}) \leq mrank(x_i, y_j)} \neg mmatch_i[j']. \tag{5.6}$$

else, the following formula is added

$$(5.7) \qquad \bigwedge_{y_{j'} \in W \text{ s.t. } mrank(x_i,y_{j'}) \leq mrank(x_i,y_j)} \neg mmatch_i[j'] \rightarrow wmatch_j[wrank(y_j, x_i)].$$

The formulae (5.5) & (5.6) enforce that if x_i is not matched to any of his weakly preferred women than y_j, then y_j must be matched to one of the men, whom she strictly prefers to x_i. This way, no pair (x_i, y_j) can form a blocking pair. Note that the formula (5.7) is necessary if there is more than one man who is ranked as same as x_i by y_j. It might be the case that y_j is also matched to one of these men, which does not violate stability since a blocking pair requires mutual strict preference.

This formulation is different from SAT-E in the following ways:

- Doctor matching variables of SAT-E are cast as man matching variables, and program matching variables as woman matching variables. Couple matching variables are omitted since there are no couples. The formulae (5.1) & (5.2) correspond to clauses 1.a & 1.b of SAT-E which establish a unique matching. The formulae (5.3) & (5.4) ensure consistency of man matching and woman matching variables.
- To handle ties, each person in the first tie group are considered, instead of the mostly preferred person. Hence, 3.a of SAT-E is adapted as (5.3). Similarly, (5.4) corresponds to 3.b.
- The stability clauses (4) of SAT-E are adapted in (5.5) – (5.7). We distinguish the cases based on the size of the tie group that x_i occurs in the preference list of woman w_j. For the case where it is 1, woman y_j must be matched to a strictly preferred man. If the size is larger than 1, then woman y_j can be matched to a man x'_i whom she ranks the same as x_i as well. This is allowed by using $wmatch_j[wrank(y_j, x_i), 1]$ variables (i.e., woman w_j is matched to a man x'_i whom she ranks $wrank(y_j, x_i)$ or higher), instead of $wmatch_j[wrank(y_j, x_i - 1), 1]$ variables.

The formulation above suffices to find a stable matching.

5.3 Solving Hard Variants of SMTI using SAT

In order to find the largest matching, we extend the formulation by soft clauses. For each man $x_i \in M$, the following soft clause is introduced

$$\neg match_i[n+1],$$

and set its weight to 1. Since the objective is to minimize the number of singles (i.e., maximize the number of married men), the weight for each soft clause which denotes that a x_i is not single, is set 1 in order to count the number of married men. Then, the MaxSAT solution for this formulation is maximal in terms of the number of married men.

To solve Egalitarian SMTI, for each man $x_i \in M$ and woman $w_j \in W$ who are mutually acceptable, the following soft clause is added

$$match_i[j],$$

and its weight is set to $|M| + |W| - (mrank(x_i, y_j) + wrank(y_j, x_i))$. Let $mrank(x_i, y_j) + wrank(y_j, x_i)$, be the total rank of the pair (x_i, y_j). The objective function for Egalitarian SMTI is a minimization of total ranks that matched pairs assign to each other. Since a MaxSAT solver is used, total rank of a pair is subtracted from the maximum value that a total rank can take, which is $|M| + |W|$. In this way, larger weights are assigned to pairs with lower total rank.

Note that if there are no soft clauses that denote single men, the MaxSAT solver tries to match every man to a woman to maximize the total cost of satisfied soft clauses. Hence, we add the following soft clause for each man $x_i \in M$

$$match_i[n+1],$$

and set its weight to $|M| + |W|$. Since being single does not contribute to the objective value of Egalitarian SMTI, the cost of any soft clause that denotes being single is higher than any soft clause that corresponds to a man-woman pair.

6. EXPERIMENTS

We have empirically analyzed the computational performances of declarative methods for SMTI, to better understand their strengths and weaknesses in terms of a variety of measures.

We have compared several ASP approaches to solve Egalitarian and Sex-Equal SMTI (Section 6.1).

We have empirically compared ASP to SAT approaches to solve SMTI (Section 6.2) by using CLINGO, LINGELING, and CMODELS with ZCHAFF.

We have repeated the experiments conducted by Gent & Prosser (2002) for finding a complete stable matching and finding the largest stable matching using Choco and Google OR-Tools CP-SAT solver (Section 6.3).

We have also performed experiments to evaluate our declarative methods and compare them with ILP and local search approaches using a variety of solvers (Section 6.4).

6.1 ASP Experiments

We have investigated two questions in these experiments: 1) the scalability of our ASP-based methods presented above to solve SMTI and its variants, 2) how these methods can be further improved by reformulating some of the constraints.

6.1.1 Experimental Setup

The tests were run on a local machine with Intel Xeon(R) W-2155 3.30GHz CPU and 32GB RAM, which has Ubuntu 18.04.1 as operating system. We have used Clingo version 5.2.2.

We set the time limit for each solver to 2000 seconds, and memory limit to 2 GB.

6.1.2 Benchmarks

To test and analyze the models and our implementations, we need an instance generator. For this, we have implemented the random instance generator proposed by Gent & Prosser (2002).

The random instance generator takes 3 inputs to generate instances: instance size n, (i.e, number of men and women), probability of incompleteness $p1$, and probability of ties $p2$. We have generated a benchmark set, with instance sizes $n = 50$ and $n = 100$, where the value of $p1$ changes in the range of [0.1, 0.8] and the value of $p2$ changes in the range [0.1, 0.9] with 0.1 step. There are 144 combinations and for each combination, we have generated 10 instances and averaged the results.

6.1.3 Results

We present the results for our ASP methods using Clingo for solving SMTI (Table 6.1), Max Cardinality SMTI (Table 6.2), Egalitarian SMTI (Table 6.3) and Sex-Equal SMTI (Table 6.4). In the tables, the numbers in square brackets denote how many of the 10 instances could be solved.

We make the following observations: Our ASP-based methods for SMTI and Max Cardinality SMTI, presented above, are scalable for relatively large instances.

To further improve the computational performance of Egalitarian and Sex-Equal SMTI, we have investigated other ASP methods.

Table 6.1 SMTI with CLINGO: Average CPU-Times (in seconds) for varying $p1$ and $p2$ values.

		$p2$								
Size	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$n = 50$	0.1	5.07	4.99	4.96	4.82	4.74	4.61	4.36	4.03	3.09
	0.2	3.41	3.39	3.33	3.27	3.17	3.04	2.91	2.63	1.88
	0.3	2.11	2.05	2.02	2.01	1.92	1.86	1.73	1.51	1.14
	0.4	1.23	1.2	1.17	1.19	1.12	1.09	1.0	0.92	0.63
	0.5	0.68	0.65	0.65	0.64	0.61	0.6	0.51	0.45	0.34
	0.6	0.35	0.33	0.35	0.32	0.3	0.29	0.26	0.22	0.17
	0.7	0.16	0.15	0.15	0.14	0.14	0.14	0.12	0.11	0.09
	0.8	0.06	0.07	0.06	0.06	0.07	0.06	0.05	0.05	0.03
$n = 100$	0.1	120.5	116.94	122.21	121.05	107.89	108.29	104.07	95.7	81.07
	0.2	80.14	80.63	78.42	79.3	74.38	69.78	68.89	66.07	54.74
	0.3	45.37	45.36	43.66	43.22	40.64	38.17	39.63	34.1	29.4
	0.4	27.52	26.65	26.15	25.84	25.27	23.86	22.75	21.76	17.96
	0.5	14.52	14.06	14.13	13.99	13.93	13.13	13.29	11.21	9.64
	0.6	7.12	7.16	7.2	7.02	6.87	7.53	6.17	5.83	4.63
	0.7	3.12	3.19	3.12	3.12	3.08	3.09	2.8	2.41	1.87
	0.8	0.95	0.96	0.97	0.92	0.93	0.84	0.74	0.64	0.46

Table 6.2 Max Cardinality SMTI with CLINGO: Average CPU-Times (in seconds).

		$p2$								
Size	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$n = 50$	0.1	6.79	6.69	6.57	6.45	6.28	6.26	5.88	5.37	4.19
	0.2	4.78	4.68	4.65	4.54	4.29	4.1	3.93	3.58	2.49
	0.3	2.85	2.78	2.71	2.64	2.5	2.43	2.33	2.05	1.58
	0.4	1.74	1.74	1.69	1.7	1.53	1.52	1.37	1.28	0.93
	0.5	1.04	0.99	1.0	0.98	0.9	0.91	0.75	0.69	0.52
	0.6	0.56	0.52	0.56	0.51	0.44	0.43	0.4	0.35	0.24
	0.7	0.24	0.23	0.23	0.21	0.18	0.19	0.15	0.14	0.11
	0.8	0.07	0.07	0.08	0.09	0.08	0.08*	0.07	0.05	0.04
$n = 100$	0.1	164.37	171.73	158.44	152.13	162.43	148.38	143.9	139.34	119.09
	0.2	114.99	114.34	110.97	106.19	106.82	98.89	93.74	90.68	79.94
	0.3	62.53	61.5	57.85	63.05	59.38	55.7	56.82	48.24	43.39
	0.4	35.2	35.14	34.88	34.39	33.78	32.13	29.58	27.36	22.89
	0.5	18.07	19.05	18.42	17.91	18.09	16.63	16.43	14.31	12.04
	0.6	8.57	8.4	8.34	8.11	8.33	8.58	7.25	6.83	5.39
	0.7	3.44	3.55	3.42	3.42	3.67	3.5	3.22	2.68	2.19
	0.8	1.1	1.07	1.08	1.04	1.08	1.01	0.95	0.81	0.58

* 1 instance reached time limit (over 2000 seconds)

Table 6.3 Egalitarian SMTI with CLINGO: Average CPU-Times (in seconds) for $n = 50$.

	$p2$								
$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	5.45	5.32	5.41	5.58	6.28	29.91	600.68[7]	TO	TO
0.2	3.84	3.87	3.94	4.13	5.66	66.59	403.27[9]	TO	TO
0.3	2.46	2.43	2.51	2.9	4.74	168.82	465.52[7]	TO	TO
0.4	1.47	1.46	1.5	1.5	4.13	22.58	276.27[6]	TO	TO
0.5	0.81	0.77	0.81	0.9	1.25	33.21	497.58[5]	TO	TO
0.6	0.44	0.4	0.44	0.75	1.06	37.59	641.19[6]	TO	TO
0.7	0.17	0.17	0.18	0.41	0.48	6.56	1358.46[3]	TO	TO
0.8	0.07	0.07	0.08	0.12	0.75	9.89[7]	674.07[4]	TO	TO

TO: Timeout (over 2000 seconds).

Table 6.4 Sex-Equal SMTI with CLINGO: Average CPU-Times (in seconds) for $n = 50$.

	$p2$								
$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	TO	TO	TO	TO	TO	TO	TO	TO	TO
0.2	TO	TO	TO	TO	TO	TO	TO	TO	TO
0.3	TO	TO	TO	TO	TO	TO	TO	TO	TO
0.4	1909.7[1]	TO	877.16[2]	TO	TO	TO	TO	TO	TO
0.5	1305.27[2]	592.94[3]	297.8[2]	1658.6[1]	953.01[1]	TO	1182.33[2]	1437.34[3]	TO
0.6	835.24[7]	656.42	1159.53[4]	910.1[4]	1093.83[4]	1048.79[8]	722.22[7]	758.89[3]	TO
0.7	82.61	70.12	240.72	211.46	115.87	245.94	906.8[8]	488.83[4]	896.77[2]
0.8	6.98	9.33	19.6	13.45	17.54	70.72	34.5	211.64	356.36[7]

TO: Timeout (over 2000 seconds).

6.1.4 Other ASP approaches for Egalitarian SMTI

Instead of the straightforward weak constraints used for Egalitarian SMTI below, to minimize $\sum_{x \in M \cup W} c_\mu(x)$:

$$\overset{\sim}{\leftarrow} marry(x,y), mrank(x,y,r1), wrank(y,x,r2). \ [r1+r2@1,x,y]$$

we have considered the following alternative formulations.

- (version 1) computes $\sum_{x \in M} c_\mu(x) + \sum_{x \in W} c_\mu(x)$ utilizing auxiliary atoms of the forms $mancost(x,r)$ and $womancost(y,r)$:

$$\begin{array}{l} mancost(x,r1) \leftarrow marry(x,y), mrank(x,y,r1). \\ womancost(y,r2) \leftarrow marry(x,y), wrank(y,x,r2). \\ \overset{\sim}{\leftarrow} mancost(x,r1). \ [r1@1,M,x] \\ \overset{\sim}{\leftarrow} womancost(y,r2). \ [r2@1,W,y] \end{array}$$

- (version 2) computes $\sum_{x \in M} c_\mu(x) + \sum_{x \in W} c_\mu(x)$ without introducing auxiliary atoms:

$$\begin{array}{l} \overset{\sim}{\leftarrow} marry(x,y), mrank(x,y,r1).[r1@1,M,x] \\ \overset{\sim}{\leftarrow} marry(x,y), wrank(y,x,r2).[r2@1,W,y] \end{array}$$

- (version 3) first utilizes chaining to describe, for every married couple (x,y), the ranges $1..r1$ and $1..r2$ such that $mrank(x,y)$=$r1$ and $wrank(y,x)$=$r2$, and then utilizes these ranges to specify the weak constraints so that Clingo can better benefit from propagation and learning:[1]

$$\begin{array}{l} mcost(x,r1) \leftarrow marry(x,y), mrank(x,y,r1). \\ mcost(x,r1-1) \leftarrow mcost(x,r1), r1 > 1. \\ wcost(y,r2) \leftarrow marry(x,y), wrank(y,x,r2). \\ wcost(y,r2-1) \leftarrow wcost(y,r2), r2 > 1. \\ \overset{\sim}{\leftarrow} mcost(x,c).[1@1,M,x,c] \\ \overset{\sim}{\leftarrow} wcost(y,c).[1@1,W,y,c] \end{array}$$

The results for these three versions are presented in Table 6.5, in comparison with the basic version with the straightforward weak constraints. We have observed that slightly more number of hard instances can be solved with the versions 1 and 2. Meanwhile, we have observed a significant improvement with the version 3 for

[1] This version of the weak constraints is suggested to us by Max Ostrowski at the Potassco list (August 30, 2021).

Table 6.5 Egalitarian SMTI with CLINGO: Average CPU-Times (in seconds) for $n = 50$.

		$p2$								
Program	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
basic	0.1	5.45	5.32	5.41	5.58	6.28	29.91	600.68[7]	TO	TO
	0.2	3.84	3.87	3.94	4.13	5.66	66.59	403.27[9]	TO	TO
	0.3	2.46	2.43	2.51	2.9	4.74	168.82	465.52[7]	TO	TO
	0.4	1.47	1.46	1.5	1.5	4.13	22.58	276.27[6]	TO	TO
	0.5	0.81	0.77	0.81	0.9	1.25	33.21	497.58[5]	TO	TO
	0.6	0.44	0.4	0.44	0.75	1.06	37.59	641.19[6]	TO	TO
	0.7	0.17	0.17	0.18	0.41	0.48	6.56	1358.46[3]	TO	TO
	0.8	0.07	0.07	0.08	0.12	0.75	9.89[7]	674.07[4]	TO	TO
version 1	0.1	6.77	6.57	6.66	6.83	8.74	37.26	599.54[6]	TO	TO
	0.2	4.52	4.41	4.54	4.53	5.91	230.48[9]	585.33[9]	TO	TO
	0.3	2.77	2.76	2.82	3.04	4.9	51.01[9]	417.19[6]	TO	TO
	0.4	1.54	1.6	1.58	1.68	4.17	21.37	264.8[6]	TO	TO
	0.5	0.98	0.97	0.98	1.04	1.31	31.22	391.16[5]	TO	TO
	0.6	0.53	0.48	0.54	0.82	1.01	34.51	526.19[6]	764.65[1]	TO
	0.7	0.24	0.22	0.23	0.44	0.47	6.57	674.72[4]	TO	TO
	0.8	0.07	0.08	0.09	0.12	0.76	201.13[9]	479.74[8]	774.31[1]	TO
version 2	0.1	6.34	6.22	6.39	6.6	7.76	47.38	322.73[7]	1026.42[1]	TO
	0.2	4.17	4.21	4.27	4.54	5.85	50.64	278.07[8]	TO	TO
	0.3	2.59	2.54	2.59	2.92	4.86	44.73[9]	484.22[8]	TO	TO
	0.4	1.55	1.55	1.53	1.61	3.96	19.03	395.45[7]	1586.22[1]	TO
	0.5	0.93	0.9	0.93	1.02	1.25	29.01	886.65[2]	TO	TO
	0.6	0.5	0.47	0.52	0.81	1.0[7]	29.13[9]	348.67[6]	1290.3[2]	TO
	0.7	0.2	0.2	0.21	0.41	0.57[2]	3.96[4]	TO	TO	TO
	0.8	0.07	0.08	0.08	0.12	0.76	17.07[8]	472.26[7]	TO	TO
version 3 (chaining)	0.1	6.62	6.6	6.7	6.85	7.31	11.25	32.79	766.17[6]	TO
	0.2	4.38	4.41	4.52	4.63	5.04	13.48	28.54[9]	335.75[2]	TO
	0.3	2.73	2.73	2.76	2.99	3.69	12.77	104.09	522.89[4]	TO
	0.4	1.63	1.64	1.64	1.71	2.83	6.58	103.73[8]	824.53[4]	TO
	0.5	0.97	0.93	0.97	1.08	1.19	10.4	115.4[8]	1162.78[2]	TO
	0.6	0.52	0.49	0.54	0.7	0.88	10.37	263.36[9]	653.44[4]	TO
	0.7	0.2	0.2	0.22	0.3	0.36	2.77	481.47[9]	1374.47[2]	TO
	0.8	0.08	0.09	0.09	0.13	0.43	163.25	74.75[9]	167.91[2]	758.19[1]

TO: No answer set is computed in 2000 seconds.

harder instances with $p2 = 0.8$ and $p2 = 0.9$. Though, Egalitarian SMTI still seems to be a hard problem for ASP.

We have also investigated how much suboptimal the solutions are computed by CLINGO using anytime search with our chaining-based formulation and a threshold of 10 seconds. We have considered harder instances with more ties (i.e., $p2 > 0.5$). The results are presented at Table 6.6, reporting the average of the suboptimal values computed by CLINGO for 10 instances, the average of the optimal values for these instances, the number of instances for which the suboptimal value equals the optimal value, and the average CPU time for computing the initial solution in seconds. For each $p1$ and $p2$, at least one suboptimal solution is computed for each instance. The computed suboptimal values are observably close to the optimal values. For most of the instances, CLINGO finds the optimal value within 10 seconds but spends approximately 2000 seconds to verify the optimality. These results suggest the use

Table 6.6 Egalitarian SMTI with Anytime CLINGO: Average computed value [optimal value], the number of computed optimal solutions (#Opt), and average CPU time to compute first solution over 10 instances with $n = 50$, using CLINGO with Anytime Search with a time threshold of 10 seconds.

	$p2$=0.6			$p2$=0.7			$p2$=0.8			$p2$=0.9		
$p1$	Computed [Optimal]	#Opt	First (sec)	Computed [Optimal]	#Opt	First (sec)	Computed [Optimal]	#Opt	First (sec)	Computed [Optimal]	#Opt	First (sec)
0.1	410.9[410.1]	8	0.15	363.6[361.0]	6	0.01	320.3[312.3]	2	0.01	261.0[234.1]	0	0.01
0.2	390.1[390.1]	10	0.05	343.9[343.4]	9	0.04	297.9[289.7]	2	0.01	248.6[221.4]	0	0.0
0.3	364.3[364.3]	10	0.06	322.4[322.4]	10	0.02	283.5[278.0]	2	0.01	241.0[215.2]	0	0.0
0.4	346.6[346.6]	10	0.06	298.4[296.7]	7	0.01	259.0[254.3]	0	0.0	225.3[198.3]	1	0.0
0.5	312.6[312.6]	10	0.02	271.5[270.3]	7	0.01	247.0[239.4]	1	0.0	203.4[184.9]	0	0.0
0.6	277.2[277.1]	9	0.02	243.9[242.2]	7	0.01	217.1[211.9]	2	0.0	180.7[165.7]	0	0.0
0.7	236.3[236.3]	10	0.01	213.8[212.1]	5	0.0	183.6[180.4]	3	0.0	152.1[143.9]	0	0.0
0.8	188.1[188.1]	10	0.0	169.6[169.3]	8	0.0	148.1[145.6]	1	0.0	128.0[124.8]	1	0.0

of ASP with anytime search for Egalitarian SMTI.

6.1.5 Other ASP approaches for Sex-Equal SMTI

Based on the promising results of Egalitarian SMTI, instead of the straightforward weak constraints used for Sex-Equal SMTI below, to minimize $|\sum_{x \in M} c_\mu(x) - \sum_{y \in W} c_\mu(y)|$:

$$\overleftarrow{\sim} t = \#sum\{r1 - r2, x, y : marry(x, y), mrank(x, y, r1), wrank(y, x, r2)\}.\ [|t|@1]$$

we have considered the following alternative formulation that utilizes chaining to describe the ranges of costs.

$$\begin{aligned}
&mcost(t) \leftarrow t = \#count\{1, c, m : mcost(m, c)\}.\\
&wcost(t) \leftarrow t = \#count\{1, c, w : wcost(w, c)\}.\\
&cost(|t1 - t2|) \leftarrow mcost(t1), wcost(t2).\\
&cost(t - 1) \leftarrow cost(t), t > 1.\\
&\overleftarrow{\sim} cost(t), t! = 0.[1@1, t]
\end{aligned}$$

The results of our experiments with this chaining-based formulation, presented at Table 6.7, illustrate a significant improvement in computational performance, considering the number of instances that can be solved within the given time threshold. Though, Sex-Equal SMTI still seems to be a hard problem for ASP.

We have also investigated how much suboptimal the solutions are computed by CLINGO using anytime search with our chaining-based formulation and a threshold of 10 seconds. We have considered harder instances with more ties (i.e., $p2 > 0.5$).

Table 6.7 Sex-Equal SMTI with CLINGO: Average CPU-Times (in seconds) for $n = 50$.

		$p2$								
Program	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
basic	0.1	TO	TO	TO	TO	TO	TO	TO	TO	TO
	0.2	TO	TO	TO	TO	TO	TO	TO	TO	TO
	0.3	TO	TO	TO	TO	TO	TO	TO	TO	TO
	0.4	885.96[1]	TO	842.16[1]	1710.85[1]	TO	TO	TO	1522.53[2]	TO
	0.5	999.53[3]	1285.43[4]	796.71[3]	1129.08[1]	1057.15[2]	TO	1248.73[3]	853.61[2]	TO
	0.6	976.46[6]	610.99	859.49[5]	883.93[7]	951.71[7]	1103.2[9]	768.81[7]	506.23[3]	TO
	0.7	175.75	162.89	308.12	309.89	155.91	304.19	704.74[5]	984.09[5]	1432.4[2]
	0.8	9.52	13.24	20.07	17.21	21.17	123.11	54.04	423.64[9]	847.13[8]
chaining	0.1	410.75	283.28	248.9	254.53	261.94	289.79	201.33	450.76[9]	442.77[5]
	0.2	257.84	173.88	172.3	171.38	215.68	186.1	114.89	223.85[9]	531.74[8]
	0.3	153.62	170.87	165.9	176.72	103.1	124.98	176.51	113.81[8]	123.51[3]
	0.4	69.78	67.29	65.9	62.12	123.9	180.76	209.12	163.38[9]	697.26[7]
	0.5	47.83	47.03	45.5	50.09	34.88	37.79	92.22	172.7	277.7[4]
	0.6	23.85	20.89	22.32	24.5	22.58	33.66	31.49	246.33[9]	TO
	0.7	9.37	8.84	8.6	9.34	12.69	16.67	160.07	1053.19[3]	TO
	0.8	2.09	2.11	1.89	2.0	3.79	55.62	134.62	765.52[5]	TO

TO: No answer set is computed in 2000 seconds.

Table 6.8 Sex-Equal SMTI with Anytime CLINGO: Average computed value [optimal value], the number of computed optimal solutions (#Opt) [number of instances for which a suboptimal solution was found], and average CPU time to compute first solution over 10 instances with $n = 50$, using CLINGO with Anytime Search with a time threshold of 10 seconds.

	$p2$=0.6			$p2$=0.7			$p2$=0.8			$p2$=0.9		
$p1$	Computed [Optimal]	#Opt [Solved]	First (sec)	Computed [Optimal]	#Opt [Solved]	First (sec)	Computed [Optimal]	#Opt [Solved]	First (sec)	Computed [Optimal]	#Opt [Solved]	First (sec)
0.1	NS	-	-	NS	-	-	NS	-	-	415.9[0.0]	0[10]	1.32
0.2	NS	-	-	NS	-	-	504.44[0.0]	0[9]	1.37	244.9[0.1]	0[10]	0.91
0.3	414.0[0.1]	0[1]	1.59	422.5[0.1]	0[6]	2.3	313.1[4.7]	0[10]	1.58	231.7[9.6]	0[10]	0.58
0.4	424.5[5.3]	0[4]	2.31	339.56[0.7]	0[9]	1.71	277.6[0.0]	0[10]	1.31	160.3[8.8]	0[10]	0.44
0.5	292.86[0.2]	0[7]	2.92	140.7[1.3]	0[10]	2.36	115.3[2.9]	0[10]	0.72	130.0[3.2]	0[10]	0.26
0.6	108.4[3.6]	0[10]	2.06	59.9[0.2]	1[10]	1.25	77.4[1.2]	0[10]	0.42	92.4[10.2]	0[10]	0.17
0.7	13.4[4.9]	6[10]	1.79	29.3[8.2]	1[10]	0.39	35.4[7.3]	0[10]	0.17	55.1[6.3]	0[10]	0.11
0.8	4.4[2.2]	7[10]	0.29	12.7[5.1]	3[10]	0.13	15.7[3.3]	0[10]	0.08	38.6[5.9]	0[10]	0.05

NS: No answer set is computed in 10 seconds.

The results are presented at Table 6.8, reporting the average of the suboptimal values computed by CLINGO for 10 instances, the average of the optimal values for these instances, the number of instances for which the suboptimal value equals the optimal value, the number of instances for which a suboptimal solution is computed, and the average CPU time for computing the initial solution in seconds. For most of the problems, for each $p1$ and $p2$, at least one suboptimal solution is computed for each instance. However, there are still some instances for which a suboptimal solution cannot be found with anytime search in 10 seconds. Also, the computed suboptimal values are not close to the optimal values. Therefore, unlike our observations with Egalitarian SMTI experiments, these results do not suggest the use of ASP with anytime search for Sex-Equal SMTI.

6.2 SAT Experiments

We have empirically evaluated two SAT approaches over randomly generated SMTI instances. First, we have adapted our ASP implementation of SMTI for CMODELS (Giunchiglia et al., 2004) that utilizes the SAT solver ZCHAFF (Moskewicz et al., 2001). Second, we have used our SAT formulation for SMTI (Section 5) and use LINGELING (Biere, 2010) to compute solutions.

6.2.1 Experimental Setup

The tests were run on a local machine with Intel Xeon(R) W-2155 3.30GHz CPU and 32GB RAM, which has Ubuntu 18.04.1 as operating system. We have used CLINGO version 5.2.2, CMODELS version 3.79 with the SAT solver ZCHAFF 2007.3.12, and SAT-E version released on May 17, 2016 with the SAT solver LINGELING bcj. The algorithm for preprocessing instances and generating clauses for SAT-E is implemented in Python programming language with version 3.6.9. We set the time limit for each solver to 2000 seconds, and memory limit to 2 GB.

6.2.2 Benchmarks

We have used the same benchmark instances as described in Section 6.1.2.

6.2.3 Results and Discussions

For SMTI, we have compared two SAT approaches: SAT (using CMODELS with ZCHAFF), and SAT (using SAT-E with LINGELING). We have also included the ASP (CLINGO) results for further comparisons.

For each solver, for each combination of $p1$ and $p2$, the average CPU times are reported in Table 6.9 for $n = 50$ and $n = 100$. The average number of atoms and clauses introduced by SAT-E and CMODELS are presented in Table 6.10. We have

observed that for all three implementations, the computation time decreases as $p1$ (i.e., probability of incompleteness) increases.

As $p2$ (i.e., probability of ties) increases, the computation time decreases for CLINGO and CMODELS, while it slightly increases for SAT-E. CLINGO and CMODELS are comparable for $p1 \geq 0.5$ but CLINGO is more advantageous than CMODELS due to less consumption of memory for $p1 < 0.5$.

In general, SAT-E is more advantageous than CMODELS, due to smaller theory sizes and a more efficient SAT solver. For instance, for $n = 50$ and $p1 = p2 = 0.1$, SAT-E generates a propositional theory with 4561 atoms and 113267 clauses in average, whereas CMODELS generates a propositional theory with 113425 atoms and 1296263 clauses in average. According to a survey about the SAT solver competitions (Järvisalo, Le Berre, Roussel & Simon, 2012), LINGELING performs significantly better than ZCHAFF.

Table 6.9 SMTI: Average CPU-Times (in seconds) for varying $p1$ and $p2$ values.

		$p2$								
Solver	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$n = 50$	0.1	5.07	4.99	4.96	4.82	4.74	4.61	4.36	4.03	3.09
	0.2	3.41	3.39	3.33	3.27	3.17	3.04	2.91	2.63	1.88
	0.3	2.11	2.05	2.02	2.01	1.92	1.86	1.73	1.51	1.14
CLINGO	0.4	1.23	1.2	1.17	1.19	1.12	1.09	1.0	0.92	0.63
	0.5	0.68	0.65	0.65	0.64	0.61	0.6	0.51	0.45	0.34
	0.6	0.35	0.33	0.35	0.32	0.3	0.29	0.26	0.22	0.17
	0.7	0.16	0.15	0.15	0.14	0.14	0.14	0.12	0.11	0.09
	0.8	0.06	0.07	0.06	0.06	0.07	0.06	0.05	0.05	0.03
	0.1	7.29	7.18	7.18	6.93	6.62	6.44	5.97	5.41	4.38
	0.2	4.92	4.95	4.85	4.7	4.6	4.2	4.04	3.7	2.95
	0.3	3.34	3.31	3.23	3.17	3.03	3.01	2.68	2.41	1.88
CMODELS	0.4	2.02	1.99	1.97	1.98	1.89	1.83	1.64	1.53	1.11
	0.5	1.19	1.12	1.16	1.13	1.06	1.07	0.89	0.81	0.61
	0.6	0.61	0.57	0.6	0.56	0.51	0.5	0.46	0.39	0.28
	0.7	0.26	0.25	0.24	0.23	0.23	0.22	0.19	0.17	0.13
	0.8	0.1	0.1	0.1	0.09	0.1	0.09	0.08	0.08	0.07
	0.1	0.73	0.73	0.71	0.83	0.74	0.78	0.73	0.84	0.7
	0.2	0.52	0.56	0.58	0.57	0.69	0.75	0.69	0.62	0.64
	0.3	0.41	0.45	0.46	0.48	0.53	0.52	0.52	0.53	0.49
SAT-E	0.4	0.3	0.33	0.34	0.39	0.4	0.39	0.43	0.44	0.41
	0.5	0.22	0.23	0.25	0.27	0.29	0.32	0.32	0.33	0.34
	0.6	0.16	0.16	0.18	0.19	0.19	0.22	0.23	0.25	0.27
	0.7	0.1	0.1	0.11	0.12	0.13	0.14	0.14	0.16	0.16
	0.8	0.06	0.06	0.07	0.07	0.08	0.08	0.08	0.09	0.11
$n = 100$	0.1	120.5	116.94	122.21	121.05	107.89	108.29	104.07	95.7	81.07
	0.2	80.14	80.63	78.42	79.3	74.38	69.78	68.89	66.07	54.74
	0.3	45.37	45.36	43.66	43.22	40.64	38.17	39.63	34.1	29.4
CLINGO	0.4	27.52	26.65	26.15	25.84	25.27	23.86	22.75	21.76	17.96
	0.5	14.52	14.06	14.13	13.99	13.93	13.13	13.29	11.21	9.64
	0.6	7.12	7.16	7.2	7.02	6.87	7.53	6.17	5.83	4.63
	0.7	3.12	3.19	3.12	3.12	3.08	3.09	2.8	2.41	1.87
	0.8	0.95	0.96	0.97	0.92	0.93	0.84	0.74	0.64	0.46
	0.1	M	M	M	M	M	M	M	M	M
	0.2	M	M	M	M	M	M	M	M	M
	0.3	M	M	M	M	M	M	M	M	M
CMODELS	0.4	M	M	M	M	M	M	M	M	M
	0.5	23.29	23.33	23.49	23.66	23.91	23.77	21.38	17.03	14.37
	0.6	12.63	12.93	12.97	12.62	11.95	11.72	10.22	8.64	6.71
	0.7	4.62	4.85	4.66	4.73	4.87	4.62	3.88	3.44	2.71
	0.8	1.47	1.42	1.45	1.4	1.4	1.28	1.16	1.01	0.76
	0.1	5.86	5.25	5.94	5.82	6.63	7.5	6.35	7.3	5.15
	0.2	4.08	4.93	5.38	4.74	6.45	5.96	6.69	6.31	5.39
	0.3	3.7	4.12	4.58	3.71	4.67	5.03	4.58	5.21	4.06
SAT-E	0.4	2.82	2.78	2.77	3.3	3.98	4.28	4.43	4.95	3.85
	0.5	1.88	1.97	2.12	2.24	2.86	2.72	3.01	3.26	3.0
	0.6	1.27	1.2	1.42	1.29	1.91	1.97	1.97	2.01	1.42
	0.7	0.58	0.65	0.63	0.72	0.82	0.52	1.1	0.97	0.85
	0.8	0.27	0.28	0.31	0.32	0.36	0.39	0.4	0.41	0.41

M: No answer set is computed due to the memory limit (2 GB).

Table 6.10 SMTI: Average program size for varying $p1$ and $p2$ values with $n = 50$

		$p2$								
Solver	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
#atoms	0.1	113425	112634	112320	110987	111095	109762	108012	104989	96243
	0.2	89963	90285	90230	89101	88298	87244	85941	83127	7602
	0.3	71035	70487	70567	70404	69397	69143	67309	64401	57927
CMODELS	0.4	53118	52825	51973	52700	51603	51357	49781	48818	43102
	0.5	38439	37674	37980	37733	37144	37668	34827	33615	30541
	0.6	26444	25382	26313	25192	24580	24654	23674	22664	19953
	0.7	15863	15638	15549	15338	15134	15239	14381	14004	12168
	0.8	8078	8315	8229	7929	8014	7647	7400	7004	6373
	0.1	4561	4549	4546	4522	4527	4499	4470	4412	4209
	0.2	4035	4047	4048	4029	4012	3997	3975	3904	3750
	0.3	3557	3549	3556	3554	3532	3540	3493	3436	3264
SAT-E	0.4	3043	3037	3018	3037	3016	3013	2976	2971	2795
	0.5	2551	2527	2544	2542	2525	2552	2466	2430	2323
	0.6	2075	2031	2078	2037	2014	2030	1998	1977	1855
	0.7	1552	1545	1542	1540	1536	1542	1517	1507	1424
	0.8	1037	1062	1059	1038	1050	1036	1023	1005	956
#clauses	0.1	1296263	1279570	1276494	1254057	1255637	1229722	1191007	1129910	962283
	0.2	937668	937568	935593	919152	905123	887740	864279	813669	681242
	0.3	670565	659872	660876	659103	641055	637263	609766	562206	464503
CMODELS	0.4	443675	439157	428319	435531	421185	417162	395655	377376	297961
	0.5	280194	271381	274901	270723	263916	268156	237253	221306	181749
	0.6	164589	154443	162863	152339	147149	146310	137174	124952	98137
	0.7	79703	77568	77378	75205	73489	73914	67096	62831	48854
	0.8	31065	31959	31535	29783	29984	27705	26257	23537	19860
	0.1	113267	112769	112840	111994	112858	112572	112637	113063	112663
	0.2	89763	90366	90651	89949	89769	89690	89944	89656	90452
	0.3	70816	70513	70890	71112	70659	71194	70580	70288	69436
SAT-E	0.4	52875	52774	52180	53241	52605	53017	52538	53488	52607
	0.5	38173	37552	38096	38113	37914	38974	36965	37190	37651
	0.6	26151	25201	26327	25417	25072	25611	25230	25403	25022
	0.7	15534	15400	15420	15377	15379	15761	15404	15747	15043
	0.8	7722	8026	8013	7790	7990	7824	7782	7784	7740

6.3 CP Experiments

We have repeated the experiments conducted by Gent & Prosser (2002) by implementing their CP models. Similarly to their experiments, we present our results in terms of search effort and solubility. We have used a set of randomly generated instances of size $n = 10$ and $n = 50$ to empirically compare two state-of-the-art solvers: Choco and Google-OR CP-SAT.

6.3.1 Experimental Setup

We have run our implementations of the CP models in Choco and OR-Tools.

As suggested by Gent and Prosser, in addition to scalability in timings, we also analyze the search effort relative to constrainedness κ. In order to calculate constrainedness κ, for each pair $(x_i, y_j) \in M \times W$, and each case that satisfies the condition of the constraint, we add $\log(1 - \frac{1}{|domain(m_i)| \times |domain(w_j)|})$. Finally, we divide this sum by $\Sigma_{x_i \in M} \log(|domain(m_i)|) + \Sigma_{y_j \in W} \log(|domain(w_j)|)$.

In order to evaluate the search effort, we report the number of branches visited during search, as reported by the CP-SAT solver of OR-Tools. Gent and Prosser report the average number of search nodes explored during search, as reported by Choco solver, in order to evaluate search effort. We use the same metric for our experiments of Choco. We also report solubility, which is the rate of instances where CP-SAT solver or Choco found a complete matching.

The tests were run on a local machine with Intel Xeon(R) W-2155 3.30GHz CPU and 32GB RAM, which has Ubuntu 18.04.1 as operating system. The algorithms are implemented in Python programming language with version 3.6.9, and Java programming language with version 11.0.15. We also use OR-Tools version 9.0.9048 and Choco solver version 4.10.9. We set the time limit to 2000 seconds.

6.3.2 Benchmarks

To test and analyze the models and our implementations, we have used the random instance generator proposed by Gent & Prosser (2002). The random instance generator takes 3 inputs to generate instances: instance size n, (i.e, number of men and women), probability of incompleteness $p1$, and probability of ties $p2$. We have generated a benchmark set, with instance sizes $n = 10$ and $n = 50$, where the value of $p1$ change in the range of [0.1, 0.8] with 0.1 step and the value of $p2$ changes in the range [0.1, 1.0] with 0.01 step. For each combination of $p1$ and $p2$, we have generated 100 instances and averaged the results.

Note that the experimental setup of Gent and Prosser considers the same $p1$ and $p2$ values with $n = 10$, where 50 instances were randomly generated for each combination of $p1$ and $p2$ values.

Table 6.11 Average constrainedness of benchmark instances.

		$p2$								
Size	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$n = 10$	0.1	0.8	0.77	0.75	0.71	0.67	0.62	0.57	0.5	0.44
	0.2	0.78	0.76	0.73	0.7	0.65	0.61	0.56	0.51	0.46
	0.3	0.77	0.74	0.72	0.69	0.65	0.6	0.56	0.51	0.47
	0.4	0.76	0.74	0.71	0.68	0.65	0.61	0.57	0.53	0.5
	0.5	0.77	0.74	0.72	0.69	0.66	0.63	0.59	0.57	0.54
	0.6	0.8	0.78	0.75	0.73	0.7	0.68	0.65	0.63	0.61
	0.7	0.87	0.85	0.81	0.82	0.79	0.76	0.74	0.73	0.7
	0.8	1.01	0.99	0.97	0.95	0.94	0.93	0.89	0.9	0.91
$n = 50$	0.1	1.66	1.64	1.62	1.59	1.55	1.49	1.41	1.25	0.91
	0.2	1.54	1.52	1.5	1.47	1.43	1.38	1.29	1.13	0.81
	0.3	1.42	1.4	1.38	1.35	1.31	1.26	1.17	1.02	0.71
	0.4	1.3	1.28	1.26	1.23	1.19	1.13	1.05	0.9	0.63
	0.5	1.18	1.16	1.14	1.11	1.07	1.01	0.92	0.78	0.55
	0.6	1.06	1.04	1.01	0.98	0.94	0.88	0.8	0.68	0.49
	0.7	0.94	0.92	0.89	0.86	0.82	0.76	0.69	0.58	0.45
	0.8	0.83	0.81	0.78	0.75	0.71	0.66	0.59	0.52	0.44

Table 6.11 presents the average constrainedness of benchmarks instances with $n = 10$ and $n = 50$ used in our experiments.

We have run our implementations of the models in Choco and OR-Tools on two sets of benchmark instances, with $n = 10$ and $n = 50$. For $n = 10$, we present the results of both solvers. For $n = 50$, Choco resulted in timeout for each 1 instance out of 3, hence we only present results of OR-Tools.

6.3.3 Results: Existence of a complete stable matching

We have implemented the CP model of Gent & Prosser (2002) for finding a complete stable matching and repeated their experiments with Choco and Google-OR CP-SAT solvers.

6.3.3.1 Search effort plotted against the probability of ties

Fig. 6.1(a) shows the average cost of the decision problem, in terms of search nodes, plotted against $p2$ values for different values of $p1$, for $n = 10$. Fig. 6.1(b) shows the average number of visited branches by OR-Tools, for $n = 10$. Fig. 6.1(c) shows the same values as Fig. 6.1(b) for $n = 50$.

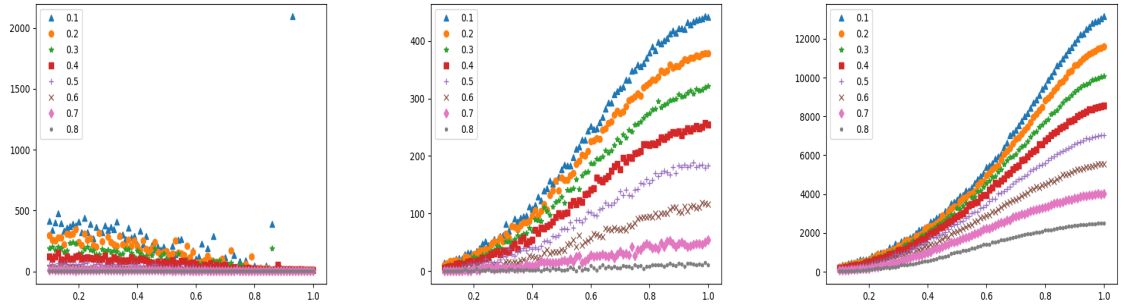
Fig. 6.1(d) shows the average cost of the decision problem, in terms of search nodes, as reported by Gent and Prosser (Gent & Prosser, 2002, Figure 5) for $n = 10$. Tables 6.12 and 6.13 provide further details; the numbers in square brackets denote the number of instances (out of 100) for which a complete matching was found.

We have observed a similar increasing trend in terms of search effort with increasing $p2$ values for different values of $p1$. We have observed that the results of OR-Tools for $n = 10$ show a similar trend as Gent and Prosser's results. However, there is an inconsistency between the results of our experiments with Choco and the results of Gent and Prosser, in terms of search effort.

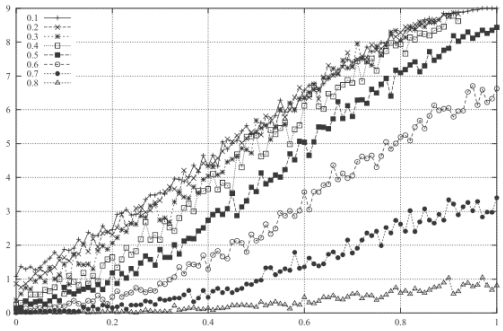
6.3.3.2 Search effort plotted against the constrainedness of instances

Fig. 6.2(a) shows the average number of visited search nodes by Choco, plotted against κ values for different values of $p1$, for $n = 10$. Fig. 6.2(b) shows the average number of visited branches for $n = 10$. Fig. 6.2(c) shows the same values as Fig. 6.2(b), for $n = 50$. Fig. 6.2(d) shows the average cost of the decision problem, in terms of search nodes, as reported by Gent and Prosser (Gent & Prosser, 2002, Figure 6) for $n = 10$.

A clear distinction of κ values between different $p1$ values can be observed in our results for $n = 50$. Unlike Gent and Prosser's results for $n = 10$, our results for



(a) (b) (c)



(d)

Figure 6.1 Search effort plotted against $p2$: (a) the average number of visited search nodes, by the Choco solver ($n = 10$) (b) the average number of visited branches, by the CP-SAT solver of OR-Tools ($n = 10$) (c) the average number of visited branches, by the CP-SAT solver of OR-Tools ($n = 50$), (d) the average number of visited search nodes by Choco (Gent & Prosser, 2002, Figure 5) ($n = 10$).

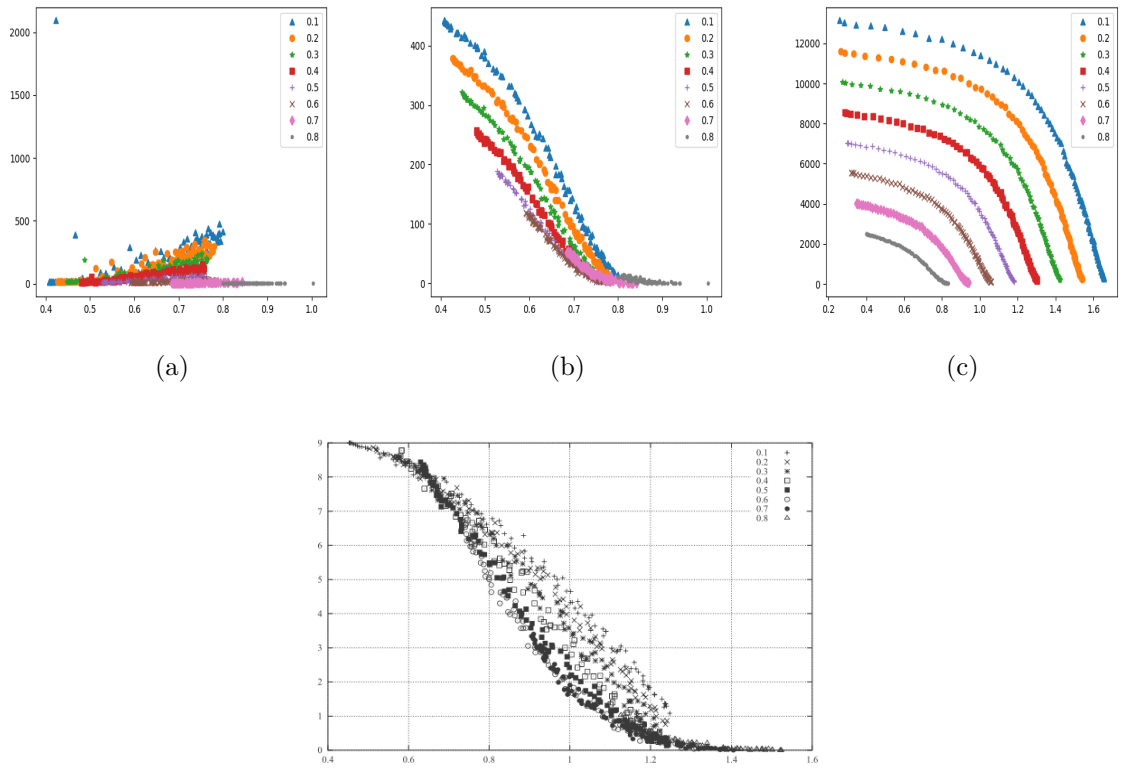
Table 6.12 Finding a complete matching: Search effort for $n = 10$: the average number of visited search nodes by the Choco solver, and the average number of visited branches by the CP-SAT solver of OR-Tools.

Solver		$p2$								
	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Choco	0.1	413.23[99]	409.62[99]	341.9	222.62[99]	111.96	100.86	114.46	47.51	18.4
	0.2	299.75[99]	237.83[94]	282.62[98]	207.04[97]	164.37	75.1	59.46	24.92	16.6
	0.3	188.63[87]	149.57[90]	180.2[91]	174.04[98]	100.74[98]	48.54	31.37	24.98	21.67
	0.4	120.41[70]	128.02[84]	83.27[89]	92.16[96]	70.4[94]	59.87[98]	57.08	22.59[99]	14.62
	0.5	51.24[50]	61.71[62]	42.54[70]	50.9[79]	31.42[86]	27.28[95]	18.09[99]	11.94[98]	11.6
	0.6	27.5[32]	31.17[42]	23.06[53]	21.83[60]	17.32[79]	13.46[82]	10.37[84]	8.69[91]	8.34[97]
	0.7	8.13[15]	8.62[21]	12.4[35]	8.0[32]	8.7[47]	6.88[57]	6.45[71]	5.2[69]	5.19[86]
	0.8	7.0[2]	2.44[9]	3.44[9]	1.73[11]	2.77[26]	3.26[27]	2.93[43]	2.74[34]	2.97[37]
CP-SAT	0.1	14.57[99]	29.47[99]	62.89	108.31[99]	177.16	252.2	317.52	380.03	421.98
	0.2	11.66[99]	29.45[94]	51.59[98]	97.0[97]	162.87	226.26	274.98	327.51	356.21
	0.3	7.26[87]	26.2[90]	37.95[91]	69.72[98]	129.08[98]	187.78	224.78	274.28	306.77
	0.4	6.19[70]	13.13[84]	34.91[89]	54.11[96]	96.64[94]	141.24[98]	176.98	219.63[99]	242.93
	0.5	5.88[50]	12.5[62]	27.13[70]	38.53[79]	72.8[86]	101.32[95]	138.65[99]	153.53[98]	183.92
	0.6	2.12[32]	4.45[42]	12.06[53]	24.9[60]	37.15[79]	57.35[82]	71.4[84]	95.34[91]	102.68[97]
	0.7	0.0[15]	5.62[21]	5.4[35]	9.88[32]	17.04[47]	27.65[57]	35.18[71]	44.72[69]	47.36[86]
	0.8	0.0[2]	0.89[9]	2.44[9]	1.45[11]	1.81[26]	9.41[27]	6.42[43]	8.38[34]	13.73[37]

Table 6.13 Finding a complete matching: Search effort of CP-SAT solver for $n = 50$: the average number of visited branches.

	$p2$								
$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	282.94	673.99	1379.08	2297.86	3708.81	5411.25	7308.47	9585.94	11788.42
0.2	246.82	573.05	1345.21	2297.99	3375.27	4954.66	6854.59	8815.36	10638.22
0.3	212.68	586.08	1152.72	1987.9	3280.52	4559.18	6055.07	7750.3	9263.53
0.4	188.2[99]	497.95[99]	1080.83	1831.62	2868.41	3964.97	5391.01	6763.09	7859.13
0.5	127.87[97]	362.18	909.39	1525.26	2476.64	3449.64	4642.15	5602.56	6550.43
0.6	99.51[88]	305.07[91]	698.35	1298.04	2057.37	2902.45	3700.05	4501.28	5192.23
0.7	69.1[61]	168.76[87]	522.19[93]	963.38	1559.91	2224.32	2838.33	3320.42	3799.49
0.8	24.55[20]	103.51[39]	284.96[69]	532.7[82]	1034.6[94]	1418.31	1811.75	2145.96[99]	2395.38

$n = 50$ show a steep decline in terms of search effort with increasing κ values. Gent and Prosser's observations hold for the results of OR-Tools for $n = 10$ and $n = 50$. In comparison to results of OR-Tools for $n = 10$, the role of incompleteness on the search effort is more clear for $n = 50$. We have observed inconsistency in terms of the number of search nodes between our experiments and Gent and Prosser's.



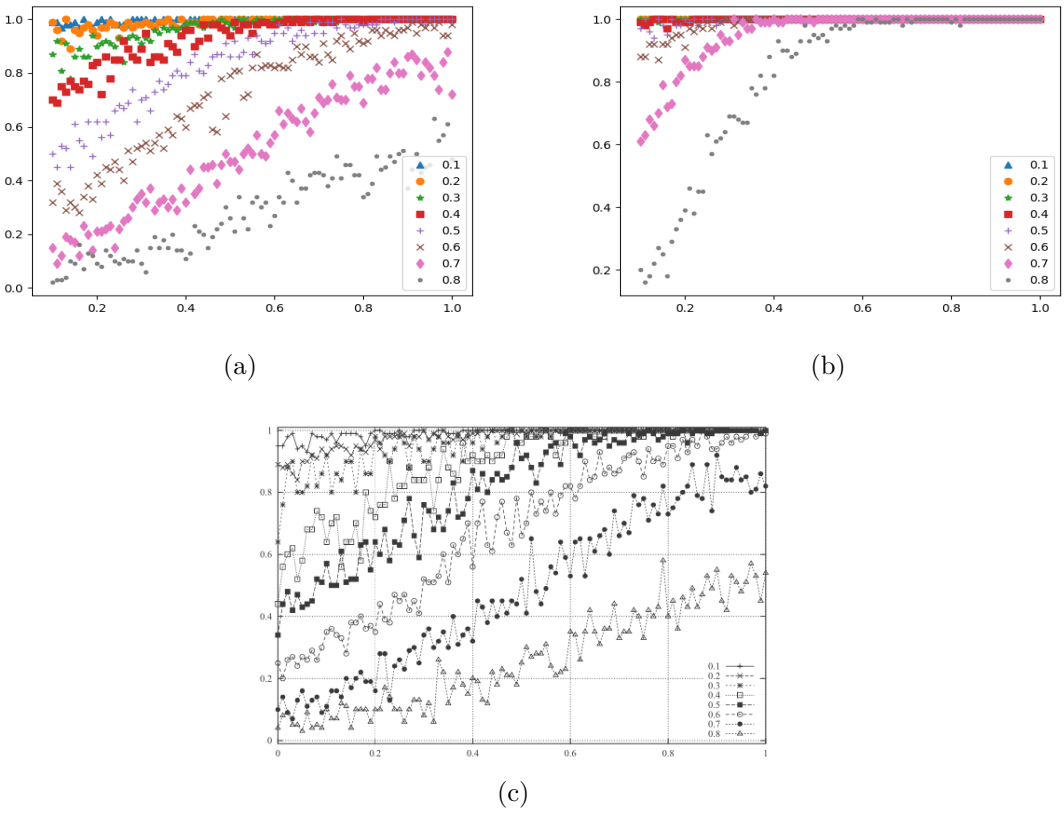
(a) (b) (c)

(d)

Figure 6.2 Search effort plotted against κ: (a) the average number of visited search nodes, by the Choco solver ($n = 10$) (b) the average number of visited branches, by the CP-SAT solver of OR-Tools ($n = 10$) (c) the average number of visited branches, by the CP-SAT solver of OR-Tools ($n = 50$), (d) the average number of visited search nodes by Choco (Gent & Prosser, 2002, Figure 6) ($n = 10$).

6.3.3.3 Solubility plotted against the probability of ties

Fig. 6.3(a) shows the average solubility plotted against $p2$ values for different values of $p1$, for $n = 10$. Both implementations resulted in same answers to the corresponding decision problems, hence we present the results by a single plot. Fig. 6.3(b) shows the results of OR-Tools, for $n = 50$. Fig. 6.3(c) shows the same plot as reported by Gent and Prosser (Gent & Prosser, 2002, Figure 3) for $n = 10$. We have observed high solubility for higher $p2$ values in both plots. Unlike the results for $n = 10$, we have observed overall higher solubility in our experiments. We have observed that the role of incompleteness decreases as ties increases for $n = 50$.

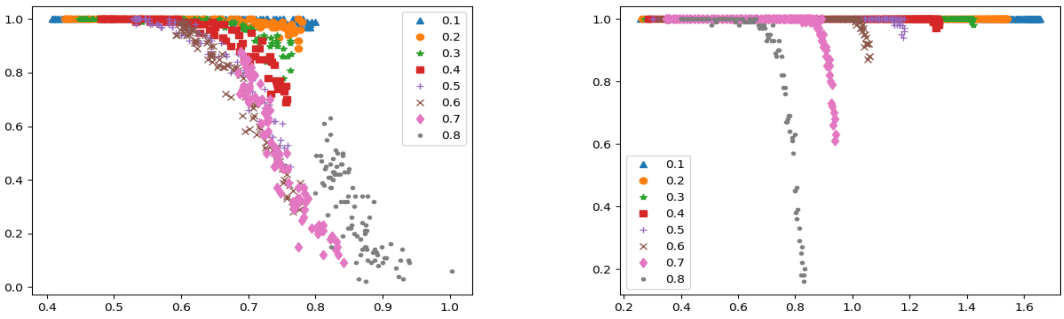


(a) (b)

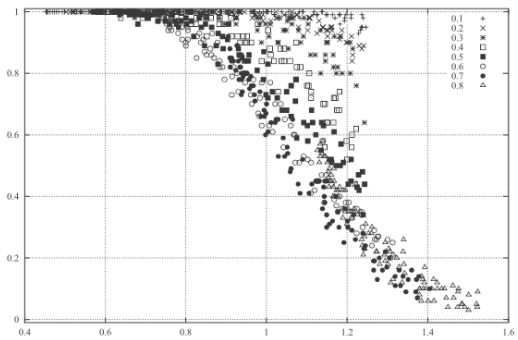
(c)

Figure 6.3 Solubility plotted against $p2$: (a) solubility ($n = 10$), (b) solubility ($n = 50$), (c) solubility (Gent & Prosser, 2002, Figure 3) ($n = 10$).

Fig. 6.4(a) shows the average solubility returned by OR-Tools and Choco implementations plotted against κ values for different values of $p1$, for $n = 10$. Fig. 6.4(b) shows the results of OR-Tools, for $n = 50$. Fig. 6.4(c) shows the same plot as reported by Gent and Prosser (Gent & Prosser, 2002, Figure 4) for $n = 10$. Unlike the results for $n = 10$, we have not observed a declining trend in solubility with increasing κ values. We have observed that the role of incompleteness decreases as constrainedness increases for $n = 50$.



(a) (b)



(c)

Figure 6.4 Solubility plotted against κ: (a) solubility ($n = 10$), (b) solubility ($n = 50$), (b) solubility (Gent & Prosser, 2002, Figure 4) ($n = 10$).

6.3.4 Results: Finding a largest stable matching

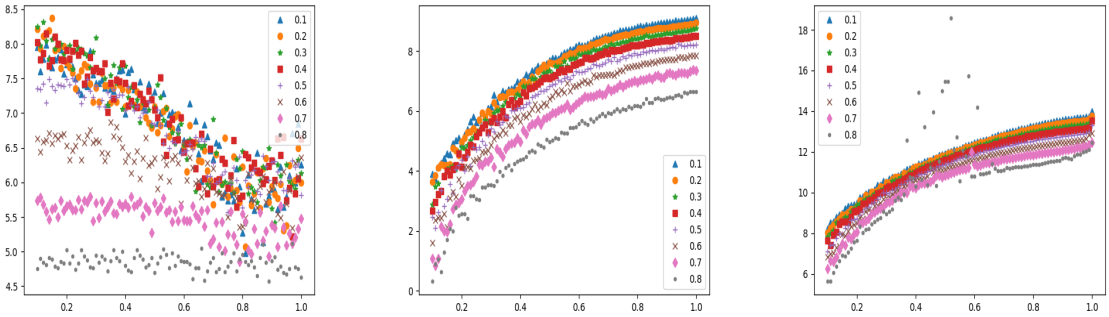
We have implemented the CP model of Gent & Prosser (2002) for finding a largest stable matching and repeated their experiments with Choco and Google-OR CP-SAT solvers.

Fig. 6.5(a) shows the average number of visited search nodes by Choco solver in log scale to find the largest stable matching, plotted against $p2$ values for different $p1$ values, for $n = 10$. Fig. 6.5(b) shows the average number of visited branches by OR-Tools in log scale, for $n = 10$. Fig. 6.5(c) shows the same values as Fig. 6.5(b), for $n = 50$. Tables 6.14 and 6.15 provide further details.

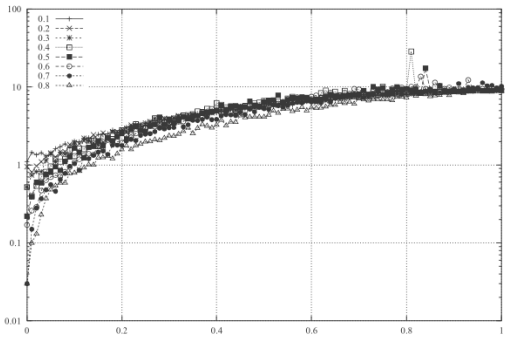
Fig. 6.5(d) shows the average number of search nodes in log scale to find the largest stable matching, as reported by Gent and Prosser (Gent & Prosser, 2002, Figure 9) for $n = 10$. However, there is an inconsistency between the results of our experiments with Choco and the results of Gent and Prosser, in terms of search effort.

Table 6.14 Max Cardinality SMTI: Search effort for $n = 10$: the average number of visited search nodes by the Choco solver, and the average number of visited branches by the CP-SAT solver of OR-Tools.

		$p2$								
Solver	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Choco	0.1	7.96	7.71	7.44	7.0	7.31	6.61	6.33	5.27	5.9
	0.2	8.22	7.73	7.36	7.1	7.1	6.53	5.68	6.22	5.94
	0.3	8.25	7.94	8.1	7.68	7.08	6.54	6.92	6.45	5.62
	0.4	8.02	7.91	7.65	7.43	7.05	6.44	5.94	5.8	6.32
	0.5	7.35	7.49	7.39	7.04	6.83	6.44	6.28	5.64	6.29
	0.6	6.64	6.53	6.61	6.31	6.34	6.57	6.2	5.37	5.91
	0.7	5.74	5.66	5.7	5.7	5.7	5.46	5.19	5.54	5.38
	0.8	4.76	5.02	4.94	4.8	4.7	4.85	4.92	4.96	4.87
CP-SAT	0.1	3.9	4.94	6.14	6.96	7.69	8.22	8.54	8.82	8.98
	0.2	3.66	4.97	5.91	6.85	7.62	8.09	8.37	8.63	8.78
	0.3	2.89	4.8	5.6	6.53	7.35	7.92	8.18	8.44	8.61
	0.4	2.66	4.13	5.64	6.34	7.09	7.63	7.93	8.2	8.38
	0.5	2.46	4.37	5.43	6.11	6.83	7.31	7.71	7.88	8.1
	0.6	1.6	3.52	4.87	5.67	6.41	6.89	7.24	7.52	7.71
	0.7	1.08	3.07	4.46	4.99	5.83	6.32	6.75	6.96	7.24
	0.8	0.31	2.58	3.54	4.37	4.97	5.57	5.95	6.17	6.43



(a) (b) (c)



(d)

Figure 6.5 Search effort plotted against $p2$: (a) the average number of visited nodes by the Choco solver, in log scale ($n = 10$) (b) the average number of visited branches by the CP-SAT solver of OR-Tools, in log scale ($n = 10$) (c) the average number of visited branches by the CP-SAT solver of OR-Tools, in log scale ($n = 50$), (d) the average number of visited search nodes by Choco, in log scale (Gent & Prosser, 2002, Figure 9) ($n = 10$).

Table 6.15 Max Cardinality SMTI: Search effort of CP-SAT solver for $n = 50$: the average number of visited branches.

	$p2$								
$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	8.21	9.5	10.54	11.28	11.93	12.48	12.9	13.27	13.57
0.2	8.02	9.33	10.48	11.26	11.8	12.34	12.81	13.16	13.43
0.3	7.82	9.36	10.29	11.07	11.77	12.23	12.63	12.98	13.24
0.4	7.66	9.11	10.22	10.97	11.59	12.06	12.49	12.79	13.01
0.5	7.12	8.74	10.0	10.72	11.4	11.85	12.27	12.53	12.76
0.6	6.86	8.5	9.73	10.61	11.16	11.63	11.97	12.24	12.46
0.7	6.26	8.07	9.5	10.22	10.84	11.37	11.64	11.86	12.08
0.8	5.63	7.67	8.96	10.29	15.47	10.8	11.12	11.38*	11.68

* 1 instance reached time limit (over 2000 seconds)

6.4 Empirical Comparisons: ASP, CP, ILP, SAT and Local Search

We evaluate our ASP, CP and SAT methods by using a variety of solvers. We also add comparisons to ILP and local search methods which are discussed in our short paper (Eyupoglu et al., 2021). We have revised the Gurobi implementation for our experiments.

6.4.1 Experimental Setup

To test and analyze the models and implementations, we have run our implementations with benchmark instances.

The tests were run on a local machine with Intel Xeon(R) W-2155 3.30GHz CPU and 32GB RAM, which has Ubuntu 18.04.1 as operating system. The algorithms are implemented in Python programming language with version 3.6.9. Additionally, we have used Gurobi version 9.1.1, OR-Tools version 8.1.848, Clingo version 5.2.2, and SAT-E version released on May 17, 2016 with the MaxSAT solver CashWMaxsat (Lei, Cai, Wang, Peng, Geng, Wan, Deng & Lu, 2021).

The time limit is set to 2000 seconds for each solver, and memory limit is set to 4 GB.

6.4.2 Benchmarks

We have used the same benchmark instances as described in Section 6.1.2.

6.4.3 Results

The ILP based method of Kwanashie and Manlove (2014) is adapted by utilizing Google-OR Tools CP-SAT solver to solve Max Cardinality, Egalitarian and Sex-Equal SMTI, for which the empirical results are denoted by OR-CP(KM). The

results for the implementation of the ILP model are denoted by OR-MIP(KM) for the OR-Tools MIP solver, and Gurobi(KM) for Gurobi.

The results for the implementation of Gent & Prosser (2002)'s CP model are denoted by OR-CP(GP). For each solver, for each combination of $p1$ and $p2$, the average CPU times for solving Max Cardinality SMTI are reported in Table 6.16 for $n = 50$, and in Table 6.19 for $n = 100$. The average CPU times for the implementation of the local search algorithm (LTIU) and genetic algorithm (GA) are presented for Max Cardinality SMTI, in Table 6.17 for $n = 50$, and Table 6.20 for $n = 100$. We have also investigated the suboptimality of the solutions computed by LTIU and GA. We have considered instances for which LTIU and GA have overall higher time consumption ($p2 < 0.5$). The results are presented at Table 6.18 for $n = 50$ and Table 6.21 for $n = 100$, reporting the average of the suboptimal values computed for 10 instances, the average of the optimal values for these instances, and the number of instances for which the suboptimal value equals the optimal value.

The average CPU times for $n = 50$ are presented in Table 6.22 for solving Egalitarian SMTI, and Table 6.23 for solving Sex-Equal SMTI. In the tables, the numbers in square brackets denote how many of the 10 instances could be solved.

The search effort results for solving Max Cardinality SMTI are presented in Table 6.24, in terms of average number of choices made by Clingo, average number of branches visited by CP-SAT solver, and average number of propagations made by CashWMaxsat during search, in log scale. The search effort results for solving Egalitarian SMTI and Sex-Equal SMTI are presented in Table 6.25 and Table 6.26 using the same metrics, respectively. The results for Clingo are obtained by using the chaining-based formulations for Egalitarian SMTI and Sex-Equal SMTI.

6.4.4 Discussions

For Max Cardinality SMTI, for a pair of $p1$ and $p2$ values, average CPU times to solve instances increase with the value of n for each solver. This is due to the increase in the number of constraints to be satisfied in ILP approaches (Gurobi and OR-Tools solvers), larger space of matchings to search in the local search approaches (LTIU and GA), and the larger program size for ASP (Clingo). For a pair of n and $p2$ values, as $p1$ increases, the number of blocking pairs most likely decreases, and thus the average CPU time usually increases for local search methods (due to larger

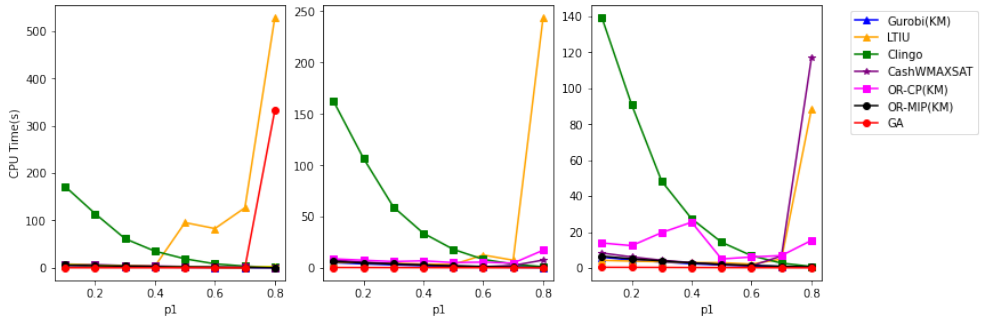


Figure 6.6 CPU Time(s) for solving Max Cardinality SMTI $p2 = 0.2$, 0.5, and 0.8 with $n = 100$

space of matchings). Meanwhile, the time consumption for other methods usually decreases as the preference lists get shorter since there are less number of constraints / rules / clauses. However, it is unexpected to see that for $n = 100$ and $p1 = 0.8$, CashWMaxsat has relatively higher time consumption, especially for $p2 > 0.6$.

For a pair of n and $p1$ values, as $p2$ increases, the number of ties increases, and thus the average CPU time usually decreases for the local search methods (due to larger possibility of stable matchings, and, in addition, due to more variety in the initial population for GA). Meanwhile, the number of blocking pairs most likely increases, and thus the average CPU time usually increases for ILP and CP methods (due to larger constraints). The rules / constraints in ASP do not get larger, but the number of stability constraints increases, and thus the average CPU time usually decreases.

For the experimental results for $n = 50$, we have observed that the average CPU times are comparable for Clingo and CashWMaxsat. It is interesting to observe that CashWMaxsat outperforms Clingo and OR-CP(GP) for $p1 = 0.1$. OR-CP(GP) is also inferior to MIP, ASP and SAT approaches in terms of scalability for Max Cardinality SMTI. The memory usage and time consumption of OR-CP(GP) becomes significantly worse than other methods when n increases to 100. For $n = 50$, OR-CP(GP) and CashWMaxsat are more advantageous for instances with shorter preference lists and less number of ties. Moreover, for $n = 100$, CashWMaxsat is more efficient than LTIU for $p1 = 0.7$ and $p1 = 0.8$, and GA for $p1 = 0.8$ when $p2 \leq 0.6$.

For local search approaches, we have observed that LTIU and GA have similar results in terms of suboptimality. Both approaches find optimal matchings for almost all instances where there is a complete matching, whereas suboptimality decreases with increasing $p1$ values.

Figure 6.6 provides a comparison of the different approaches with respect to the CPU

time for $n = 100$. It can be observed that, for most instances, these approaches are comparable to each other, and that the local search methods and CashWMaxsat take more time for larger values of $p1$ (when the preference lists are more incomplete). We can also observe that, for smaller values of $p1 < 0.7$, GA is more efficient, whereas, for larger values of $p1 \geq 0.7$, Clingo is more efficient. It is also notable that Gurobi(KM) and OR-MIP(KM) outperform other approaches.

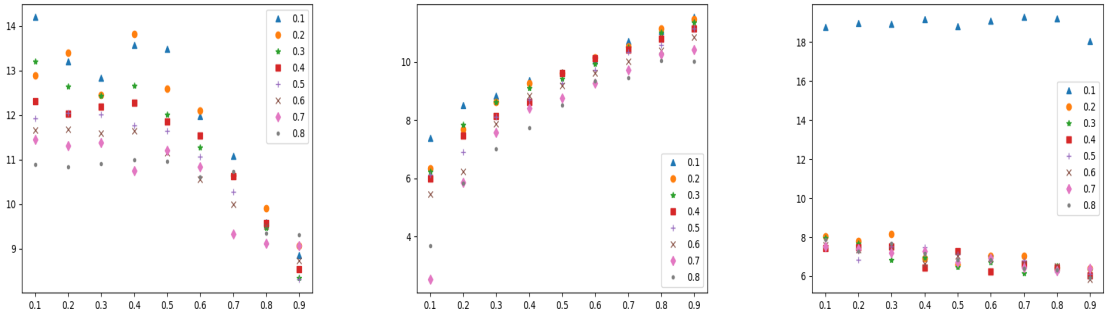
For Sex-Equal SMTI and Egalitarian SMTI, we have observed that the average CPU times are larger for the CP and ASP approaches when compared with the Max Cardinality SMTI experimental results (Table 6.16). Note that there is a larger number of instances that could not be solved with these approaches within the given time threshold. The ILP approaches, on the other hand, perform better.

For Egalitarian SMTI, observe that OR-CP(GP) outperforms Clingo and OR-CP(KM) for $p2 > 0.5$, and CashWMaxsat for $p2 > 0.6$. For Sex-Equal SMTI, OR-CP(GP) clearly outperforms Clingo. It also outperforms OR-CP(KM) for $p2 = 0.9$. It can be concluded that OR-CP(GP) is more efficient than Clingo and OR-CP(KM), for instances with shorter preference lists that contain higher number of ties.

It is interesting to observe that Clingo has better results for Max Cardinality SMTI, compared to Egalitarian SMTI and Sex-Equal SMTI. This may be related to the objective functions: Egalitarian SMTI and Sex-Equal SMTI aims to minimize the sum of costs that are calculated by addition or subtraction of some nonnegative numbers whereas Max Cardinality SMTI aims to minimize the sum of 1's. The large CPU times for the ASP approach for Sex-Equal SMTI could also be due to the use of aggregates in weak constraints.

Figure 6.7 provides a comparison of ASP, CP and SAT approaches for solving Max Cardinality SMTI, in terms of search effort for $n = 50$. It is interesting to observe that for a fixed value of $p1$, the search effort for CashWMaxsat slightly decreases for larger values of $p2$ while CPU time increases. We also observe that for larger values of $p2$, search effort decreases for Clingo while it increases for OR-Tools' CP-SAT solver. Our observation for the CP approach is consistent with Gent and Prosser's.

For Egalitarian SMTI, we observe that for a fixed $p1$ value and increasing $p2$ values, the search effort for OR-CP(GP) grows faster than Clingo and CashWMaxsat. However, OR-CP(GP) finds the optimal solution for each instance within the time threshold, hence is more efficient than Clingo and CashWMaxsat. For Sex-Equal SMTI, we also observe that for a fixed $p1$ value and increasing $p2$ values, the



(a) (b) (c)

Figure 6.7 Search effort for solving Max Cardinality SMTI plotted against $p2$: (a) the average number of choices by Clingo, in log scale ($n = 50$) (b) the average number of visited branches by the CP-SAT solver of OR-Tools, in log scale ($n = 50$) (c) the average number of propagations by CashWMaxsat, in log scale ($n = 50$).

search effort for OR-CP(GP) grows faster than Clingo.

Table 6.16 Max Cardinality SMTI: Average CPU-Times (in seconds) for $n = 50$.

Solver	$p1$	$p2$ 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Clingo	0.1	6.79	6.69	6.57	6.45	6.28	6.26	5.88	5.37	4.19
	0.2	4.78	4.68	4.65	4.54	4.29	4.1	3.93	3.58	2.49
	0.3	2.85	2.78	2.71	2.64	2.5	2.43	2.33	2.05	1.58
	0.4	1.74	1.74	1.69	1.7	1.53	1.52	1.37	1.28	0.93
	0.5	1.04	0.99	1.0	0.98	0.9	0.91	0.75	0.69	0.52
	0.6	0.56	0.52	0.56	0.51	0.44	0.43	0.4	0.35	0.24
	0.7	0.24	0.23	0.23	0.21	0.18	0.19	0.15	0.14	0.11
	0.8	0.07	0.07	0.08	0.09	0.08	0.08*	0.07	0.05	0.04
Gurobi(KM)	0.1	0.44	0.47	0.48	0.5	0.49	0.52	0.58	0.66	0.59
	0.2	0.34	0.35	0.37	0.39	0.4	0.44	0.46	0.54	0.51
	0.3	0.27	0.29	0.32	0.37	0.33	0.36	0.38	0.41	0.39
	0.4	0.21	0.22	0.22	0.24	0.26	0.28	0.29	0.33	0.32
	0.5	0.17	0.16	0.17	0.18	0.19	0.21	0.21	0.23	0.24
	0.6	0.12	0.12	0.14	0.14	0.14	0.15	0.16	0.17	0.16
	0.7	0.09	0.09	0.1	0.1	0.11	0.11	0.12	0.12	0.11
	0.8	0.07	0.07	0.07	0.08	0.08	0.08	0.08	0.09	0.09
OR-MIP(KM)	0.1	0.69	0.71	0.72	0.72	0.73	0.74	0.77	0.8	0.84
	0.2	0.55	0.57	0.58	0.6	0.6	0.61	0.62	0.65	0.7
	0.3	0.44	0.46	0.48	0.48	0.49	0.5	0.52	0.53	0.56
	0.4	0.35	0.36	0.36	0.38	0.39	0.39	0.4	0.43	0.45
	0.5	0.27	0.27	0.29	0.29	0.3	0.31	0.31	0.33	0.36
	0.6	0.21	0.2	0.22	0.23	0.23	0.24	0.24	0.25	0.27
	0.7	0.15	0.15	0.16	0.16	0.17	0.18	0.18	0.19	0.19
	0.8	0.11	0.11	0.11	0.12	0.12	0.13	0.13	0.14	0.14
OR-CP(KM)	0.1	0.9	0.93	0.91	1.03	1.05	1.07	1.11	1.14	1.32
	0.2	0.74	0.73	0.75	0.77	0.81	0.85	1.03	1.0	0.99
	0.3	0.59	0.6	0.62	0.64	0.65	0.73	0.81	0.75	0.76
	0.4	0.47	0.47	0.48	0.5	0.51	0.6	1.61	0.75	0.6
	0.5	0.38	0.37	0.38	0.41	0.44	0.46	0.53	0.54	0.53
	0.6	0.3	0.29	0.31	0.33	0.34	0.41	0.52	0.81	0.4
	0.7	0.24	0.23	0.24	0.26	0.3	0.4	0.48	0.58	0.31
	0.8	0.18	0.18	0.19	0.19	0.21	0.88	0.29	0.26	0.39
OR-CP(GP)	0.1	17.37	17.06	17.14	16.91	17.14	17.03	17.23	16.59	14.51
	0.2	12.31	12.42	12.4	12.29	12.09	12.43	12.03	12.07	10.34
	0.3	8.73	8.64	8.66	8.68	8.44	8.79	8.49	7.9	6.86
	0.4	5.72	5.64	5.53	5.72	5.55	5.56	5.4	5.3	4.38
	0.5	3.53	3.46	3.54	3.5	3.45	3.55	3.18	3.06	2.69
	0.6	2.04	1.93	2.06	1.98	1.9	1.92	1.85	1.72	1.44
	0.7	0.95	0.93	0.96	0.95	0.94	0.98	0.91	0.88	0.72
	0.8	0.35	0.38	0.38	0.36	0.38	0.36	0.36	0.33	0.3
CashWMaxsat	0.1	0.72	0.72	0.72	0.73	0.71	0.73	0.75	0.73	0.72
	0.2	6.28	6.86	7.08	8.66	9.72	13.68	19.38	45.62	64.06
	0.3	3.75	4.59	5.52	6.0	6.87	8.99	19.35	41.91	43.58
	0.4	3.16	3.37	3.27	4.38	4.69	5.93	10.73	20.53	33.49
	0.5	2.24	2.28	2.29	2.81	2.86	4.54	5.34	19.42	29.0
	0.6	1.29	1.37	1.68	2.29	2.21	2.58	3.53	4.68	13.1
	0.7	0.73	0.86	0.94	0.95	1.08	1.6	1.81	2.62	5.88
	0.8	0.29	0.41	0.41	0.56	1.26	2.16	6.87	1.29	2.2

* 1 instance reached time limit (over 2000 seconds)

Table 6.17 Max Cardinality SMTI: Average CPU-Times (in seconds) for LTIU and GA with $n = 50$.

		$p2$								
Solver	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
LTIU	0.1	0.46	0.51	0.4	0.47	0.4	0.39	0.36	0.33	0.24
	0.2	0.36	0.4	0.36	0.41	0.36	0.33	0.36	0.28	0.22
	0.3	0.36	0.39	0.37	0.33	0.3	0.31	0.28	0.26	0.21
	0.4	43.33	0.33	0.28	0.9	0.29	0.25	0.26	0.22	0.18
	0.5	15.57	0.27	2.45	0.27	0.34	0.24	0.5	0.38	0.15
	0.6	16.71	29.4	6.27	12.75	0.54	0.27	0.2	0.19	0.31
	0.7	47.9	43.58	82.46	14.97	29.92	397	5.64	14	1.08
	0.8	114.67	94.89	85.28	93.76	77.39	54.16	65.11	37.17	17.27
GA	0.1	0.05	0.07	0.09	0.1	0.11	0.11	0.11	0.1	0.1
	0.2	0.05	0.06	0.08	0.09	0.1	0.1	0.1	0.09	0.09
	0.3	0.04	0.06	0.07	0.08	0.09	0.09	0.09	0.09	0.08
	0.4	18.69	0.05	0.06	0.07	0.07	0.08	0.08	0.07	0.07
	0.5	9.01	0.04	0.05	0.06	0.06	0.06	0.06	0.06	0.06
	0.6	8.88	17.34	0.04	8.94	0.05	0.05	0.05	0.05	0.05
	0.7	23.74	14.98	38.57	7.37	7.33	7.68	0.04	0.04	0.04
	0.8	62.31	43.37	42.71	41.75	29.28	23.16	21.96	0.05	4.87

Table 6.18 Max Cardinality SMTI: Average computed value [optimal value], and the number of computed optimal solutions (#Opt) over 10 instances with $n = 50$, for LTIU and GA.

		$p2$=0.1		$p2$=0.2		$p2$=0.3		$p2$=0.4	
Solver	$p1$	Computed [Optimal]	#Opt	Computed [Optimal]	#Opt	Computed [Optimal]	#Opt	Computed [Optimal]	#Opt
LTIU	0.1	50[50]	10	50[50]	10	50[50]	10	50[50]	10
	0.2	50[50]	10	50[50]	10	50[50]	10	50[50]	10
	0.3	50[50]	10	50[50]	10	50[50]	10	50[50]	10
	0.4	49.6[49.8]	8	50[50]	10	50[50]	10	50[50]	10
	0.5	49.8[49.9]	9	50[50]	10	50[50]	10	50[50]	10
	0.6	49.8[49.9]	9	49.6[49.8]	8	50[50]	10	49.8[49.9]	9
	0.7	49.4[49.7]	7	49.4[49.8]	7	48.6[49.6]	5	49.8[50.0]	9
	0.8	46.2[48.1]	0	47.8[49.2]	2	48.2[49.1]	3	48.0[49.2]	2
GA	0.1	50[50]	10	50[50]	10	50[50]	10	50[50]	10
	0.2	50[50]	10	50[50]	10	50[50]	10	50[50]	10
	0.3	50[50]	10	50[50]	10	50[50]	10	50[50]	10
	0.4	49.6[49.8]	8	50[50]	10	50[50]	10	50[50]	10
	0.5	49.8[49.9]	9	50[50]	10	50[50]	10	50[50]	10
	0.6	49.8[49.9]	9	49.6[49.8]	8	50[50]	10	49.8[49.9]	9
	0.7	49.2[49.7]	7	49.6[49.8]	8	48.6[49.6]	5	49.8[50.0]	9
	0.8	45.8[48.1]	0	48.2[49.2]	3	47.8[49.1]	3	48.0[49.2]	3

Table 6.19 Max Cardinality SMTI: Average CPU-Times (in seconds) for $n = 100$.

Solver		$p2$								
	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Clingo	0.1	164.37	171.73	158.44	152.13	162.43	148.38	143.9	139.34	119.09
	0.2	114.99	114.34	110.97	106.19	106.82	98.89	93.74	90.68	79.94
	0.3	62.53	61.5	57.85	63.05	59.38	55.7	56.82	48.24	43.39
	0.4	35.2	35.14	34.88	34.39	33.78	32.13	29.58	27.36	22.89
	0.5	18.07	19.05	18.42	17.91	18.09	16.63	16.43	14.31	12.04
	0.6	8.57	8.4	8.34	8.11	8.33	8.58	7.25	6.83	5.39
	0.7	3.44	3.55	3.42	3.42	3.67	3.5	3.22	2.68	2.19
	0.8	1.1	1.07	1.08	1.04	1.08	1.01	0.95	0.81	0.58
Gurobi(KM)	0.1	4.66	4.76	4.84	4.99	5.17	5.37	5.74	6.01	6.17
	0.2	3.51	3.58	3.66	3.91	3.97	4.25	4.46	4.38	4.53
	0.3	2.53	2.6	2.7	2.86	3.0	3.07	3.22	3.35	3.48
	0.4	1.79	1.84	1.92	1.95	2.12	2.19	2.23	2.36	2.4
	0.5	1.25	1.25	1.34	1.37	1.49	1.58	1.66	1.65	1.71
	0.6	0.81	0.83	0.88	0.95	1.02	1.08	1.21	1.2	1.19
	0.7	0.53	0.55	0.56	0.6	0.66	0.71	0.77	0.79	0.76
	0.8	0.35	0.35	0.38	0.39	0.41	0.45	0.46	0.48	0.45
OR-MIP(KM)	0.1	5.91	5.98	6.02	6.17	6.2	6.22	6.31	6.56	6.75
	0.2	4.68	4.75	4.77	4.88	4.87	4.88	4.98	5.09	5.37
	0.3	3.59	3.65	3.73	3.77	3.84	3.77	3.86	3.94	4.05
	0.4	2.64	2.7	2.72	2.76	2.81	2.74	2.74	2.85	3.0
	0.5	1.86	1.88	1.92	1.94	1.99	1.98	2.01	2.03	2.18
	0.6	1.27	1.31	1.35	1.39	1.38	1.4	1.41	1.45	1.53
	0.7	0.83	0.86	0.88	0.92	0.94	0.94	0.95	0.94	1.0
	0.8	0.54	0.55	0.58	0.59	0.62	0.62	0.61	0.62	0.64
OR-CP(KM)	0.1	7.63	7.98	8.16	8.41	8.81	9.77	13.72	13.91	26.17
	0.2	6.57	6.67	6.83	7.59	7.47	7.13	8.89	12.4	18.25
	0.3	4.66	4.81	5.65	5.81	6.21	12.48	9.86	19.66	20.61
	0.4	3.54	3.73	4.01	4.27	6.88	6.1	31.63	25.45	13.72
	0.5	2.6	2.67	3.16	3.29	5.16	8.3	6.18	4.99	10.15
	0.6	1.86	1.92	2.1	3.06	5.61	7.67	9.82	6.13	6.37
	0.7	1.27	1.33	1.55	3.48	4.65	13.28	6.64	6.88	3.37
	0.8	0.89	0.9	1.11	1.59	17.22	72.66	195.72	15.32	1.6
OR-CP(GP)	0.1	M	M	M	M	M	M	M	M	M
	0.2	M	M	M	M	M	M	M	M	M
	0.3	M	M	M	M	M	M	M	M	M
	0.4	M	M	M	M	M	M	M	M	M
	0.5	M	M	M	M	M	M	M	M	M
	0.6	24.77	25.69	25.93	25.87	26.35	27.69	27.33	26.13	21.24
	0.7	11.18	11.52	11.14	11.35	11.53	11.35	11.34	10.42	9.15
	0.8	3.95	3.9	3.99	3.91	3.92	3.82	3.7	3.56	2.97
CashWMaxsat	0.1	7.49	7.51	7.63	7.42	7.73	7.92	8.09	8.33	8.29
	0.2	5.47	5.52	5.52	5.62	5.66	5.7	5.96	6.14	6.14
	0.3	3.83	3.98	3.95	3.93	3.99	4.01	4.13	4.28	4.2
	0.4	2.63	2.71	2.7	2.67	2.79	2.74	2.82	2.96	2.84
	0.5	1.75	1.74	1.78	1.86	1.85	1.79	1.87	1.87	1.87
	0.6	1.05	1.09	1.1	1.09	1.14	1.16	1.12	1.17	1.12
	0.7	0.59	0.6	0.59	0.62	2.35	0.65	0.64	6.69	0.59
	0.8	2.98	3.24	4.19	6.9	7.63	69.63	75.36	116.65	113.46

M: Memory limit reached (over 4 GB)

Table 6.20 Max Cardinality SMTI: Average CPU-Times (in seconds) for LTIU and GA with $n = 100$.

		$p2$								
Solver	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
LTIU	0.1	5.69	5.46	5.83	5.29	5.65	4.84	5.01	4.25	3.69
	0.2	4.90	4.94	5.2	4.96	5.09	4.6	4.55	3.8	3.16
	0.3	4.45	4.55	4.36	4.42	4.47	4.17	4.04	3.48	2.93
	0.4	3.86	3.83	3.93	3.69	3.78	3.78	3.3	3.02	2.9
	0.5	3.5	95.63	3.42	3.3	3.29	2.97	2.88	2.96	2.37
	0.6	2.94	83.01	6.05	2.6	12.54	2.66	2.56	2.15	1.72
	0.7	141.2	126.92	80.2	16.23	7.5	3.88	4.79	4.92	2.09
	0.8	420.05	529.24	235.42	245.69	244.11	136.01	111.51	88.17	17.62
GA	0.1	0.2	0.28	0.33	0.38	0.41	0.42	0.41	0.38	0.34
	0.2	0.18	0.25	0.3	0.34	0.36	0.37	0.37	0.35	0.31
	0.3	0.16	0.22	0.27	0.3	0.32	0.33	0.32	0.31	0.27
	0.4	0.14	0.19	0.23	0.26	0.28	0.28	0.28	0.26	0.24
	0.5	0.12	0.16	0.19	0.22	0.23	0.24	0.24	0.22	0.2
	0.6	0.1	0.13	0.16	0.17	0.19	0.2	0.19	0.18	0.17
	0.7	56.72	0.1	51.88	0.13	0.14	0.15	0.15	0.14	0.13
	0.8	288.81	333.06	123.44	81.91	0.1	0.1	0.1	0.1	0.09

Table 6.21 Max Cardinality SMTI: Average computed value [optimal value], and the number of computed optimal solutions (#Opt) over 10 instances with $n = 100$, for LTIU and GA.

		$p2$=0.1		$p2$=0.2		$p2$=0.3		$p2$=0.4	
Solver	$p1$	Computed [Optimal]	#Opt	Computed [Optimal]	#Opt	Computed [Optimal]	#Opt	Computed [Optimal]	#Opt
LTIU	0.1	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.2	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.3	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.4	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.5	100[100]	10	99.8[100]	9	100[100]	10	100[100]	10
	0.6	100[100]	10	99.8[100]	9	100[100]	10	100[100]	10
	0.7	99.6[99.8]	9	99.6[100]	8	99.8[100]	9	100[100]	10
	0.8	98[98]	10	97.6[99]	4	99[99.8]	7	99.2[100]	6
GA	0.1	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.2	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.3	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.4	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.5	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.6	100[100]	10	100[100]	10	100[100]	10	100[100]	10
	0.7	99.8[99.8]	10	100[100]	10	99.8[100]	9	100[100]	10
	0.8	98[98]	10	97.4[99]	5	99[99.8]	7	99.6[100]	8

Table 6.22 Egalitarian SMTI: Average CPU-Times (in seconds) for $n = 50$.

Solver		$p2$								
	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Clingo	0.1	6.62	6.6	6.7	6.85	7.31	11.25	32.79	766.17[6]	TO
	0.2	4.38	4.41	4.52	4.63	5.04	13.48	28.54[9]	335.75[2]	TO
	0.3	2.73	2.73	2.76	2.99	3.69	12.77	104.09	522.89[4]	TO
	0.4	1.63	1.64	1.64	1.71	2.83	6.58	103.73[8]	824.53[4]	TO
	0.5	0.97	0.93	0.97	1.08	1.19	10.4	115.4[8]	1162.78[2]	TO
	0.6	0.52	0.49	0.54	0.7	0.88	10.37	263.36[9]	653.44[4]	TO
	0.7	0.2	0.2	0.22	0.3	0.36	2.77	481.47[9]	1374.47[2]	TO
	0.8	0.08	0.09	0.09	0.13	0.43	163.25	74.75[9]	167.91[2]	758.19[1]
Gurobi(KM)	0.1	0.42	0.44	0.46	0.46	0.48	0.5	0.52	0.59	0.66
	0.2	0.34	0.35	0.36	0.39	0.4	0.42	0.49	0.57	0.65
	0.3	0.27	0.29	0.29	0.32	0.34	0.35	0.39	0.48	0.85
	0.4	0.21	0.22	0.22	0.24	0.27	0.28	0.37	0.43	0.48
	0.5	0.17	0.16	0.18	0.19	0.19	0.23	0.25	0.32	0.59
	0.6	0.12	0.12	0.14	0.15	0.15	0.18	0.2	0.27	0.32
	0.7	0.09	0.1	0.1	0.11	0.11	0.12	0.15	0.17	0.31
	0.8	0.07	0.08	0.09	0.09	0.09	0.1	0.1	0.12	0.16
OR-MIP(KM)	0.1	0.7	0.72	0.73	0.75	0.76	0.79	0.83	0.99	1.3
	0.2	0.56	0.58	0.59	0.61	0.62	0.67	0.7	1.01	1.28
	0.3	0.45	0.47	0.48	0.51	0.52	0.53	0.63	0.98	1.9
	0.4	0.35	0.36	0.37	0.39	0.44	0.47	0.66	0.86	1.14
	0.5	0.28	0.27	0.29	0.32	0.32	0.37	0.45	0.75	1.28
	0.6	0.21	0.21	0.22	0.24	0.25	0.35	0.37	0.58	0.93
	0.7	0.15	0.15	0.16	0.18	0.19	0.2	0.28	0.42	0.63
	0.8	0.11	0.11	0.12	0.12	0.14	0.16	0.17	0.24	0.39
OR-CP(KM)	0.1	1.01	1.04	1.07	1.38	2.13	96.34	417.34[5]	TO	TO
	0.2	1.43	1.51	1.19	1.85	3.71	196.42	391.73[6]	TO	TO
	0.3	1.1	1.19	1.33	1.78	11.76	60.11[8]	561.85[4]	TO	TO
	0.4	0.9	1.0	1.01	1.19	9.06	108.64	629.32[4]	TO	TO
	0.5	0.79	0.77	0.81	1.14	1.97	147.54	213.31[3]	TO	TO
	0.6	0.6	0.61	0.65	1.84	2.25	131.32	82.9[1]	TO	TO
	0.7	0.45	0.45	0.49	1.19	1.21	26.87	951.98[1]	TO	TO
	0.8	0.35	0.36	0.36	0.44	1.48	55.2[8]	477.63[3]	TO	TO
OR-CP(GP)	0.1	17.89	19.14	19.85	20.22	20.88	21.35	22.36	26.17	25.5
	0.2	12.32	12.36	12.42	12.3	12.25	12.72	13.4	17.21	15.08
	0.3	8.8	8.7	8.81	8.86	8.68	8.78	8.91	8.59	8.0
	0.4	5.78	5.78	5.66	5.81	5.75	5.8	5.9	6.16	5.04
	0.5	3.62	3.52	3.65	3.64	3.62	3.77	3.46	3.69	3.46
	0.6	2.13	2.02	2.2	2.1	2.04	2.14	2.11	2.06	1.88
	0.7	1.03	1.02	1.05	1.06	1.07	1.1	1.11	1.12	1.04
	0.8	0.41	0.43	0.44	0.43	0.45	0.45	0.46	0.45	0.47
CashWMaxsat	0.1	0.86	1.22	3.16	35.82	585.88[9]	680.27[3]	TO	TO	TO
	0.2	5.62	6.04	7.04	7.92	9.63	11.96	13.45	31.42	39.33
	0.3	3.85	5.29	6.15	6.89	7.44	8.16	12.28	26.84	37.98
	0.4	3.19	3.33	3.56	5.05	6.08	7.42	13.27	19.42	26.41
	0.5	2.03	2.19	2.35	3.13	2.76	4.32	4.93	13.97	24.36
	0.6	1.37	1.37	1.72	2.12	1.9	3.55	4.29	6.62	16.06
	0.7	0.72	0.9	0.97	0.99	1.25	1.5	2.39	6.25	10.57
	0.8	0.28	0.41	0.37	0.47	0.62	0.72	0.95	1.53	3.14

TO: Timeout (over 2000 seconds)

Table 6.23 Sex-Equal SMTI: Average CPU-Times (in seconds) for $n = 50$.

Solver		$p2$								
	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Clingo	0.1	410.75	283.28	248.9	254.53	261.94	289.79	201.33	450.76[9]	442.77[5]
	0.2	257.84	173.88	172.3	171.38	215.68	186.1	114.89	223.85[9]	531.74[8]
	0.3	153.62	170.87	165.9	176.72	103.1	124.98	176.51	113.81[8]	123.51[3]
	0.4	69.78	67.29	65.9	62.12	123.9	180.76	209.12	163.38[9]	697.26[7]
	0.5	47.83	47.03	45.5	50.09	34.88	37.79	92.22	172.7	277.7[4]
	0.6	23.85	20.89	22.32	24.5	22.58	33.66	31.49	246.33[9]	TO
	0.7	9.37	8.84	8.6	9.34	12.69	16.67	160.07	1053.19[3]	TO
	0.8	2.09	2.11	1.89	2.0	3.79	55.62	134.62	940.11[5]	TO
Gurobi(KM)	0.1	0.48	0.56	0.55	0.6	0.64	0.71	0.81	0.8	0.77
	0.2	0.39	0.44	0.48	0.63	0.55	0.74	0.76	0.93	0.71
	0.3	0.31	0.35	0.38	0.46	0.53	0.52	0.64	0.6	0.58
	0.4	0.25	0.27	0.28	0.32	0.37	0.38	0.44	0.47	0.46
	0.5	0.2	0.21	0.25	0.26	0.3	0.37	0.29	0.35	0.35
	0.6	0.16	0.16	0.18	0.21	0.2	0.23	0.25	0.24	0.23
	0.7	0.12	0.13	0.14	0.15	0.15	0.17	0.17	0.18	0.17
	0.8	0.1	0.1	0.1	0.11	0.12	0.12	0.12	0.13	0.13
OR-MIP(KM)	0.1	0.84	0.84	0.86	0.87	0.93	0.98	1.08	1.36	1.95
	0.2	0.67	0.69	0.71	0.74	0.74	0.87	0.91	1.51	2.36
	0.3	0.54	0.55	0.56	0.58	0.63	0.67	0.7	1.01	1.21
	0.4	0.45	0.44	0.44	0.48	0.51	0.52	0.64	0.94	1.35
	0.5	0.35	0.36	0.37	0.38	0.38	0.51	0.51	0.75	1.06
	0.6	0.27	0.27	0.29	0.32	0.32	0.41	0.52	0.46	0.47
	0.7	0.21	0.21	0.22	0.23	0.24	0.26	0.31	0.42	0.34
	0.8	0.16	0.17	0.18	0.18	0.19	0.21	0.22	0.22	0.32
OR-CP(KM)	0.1	1.94	2.01	1.81	2.15	2.22	2.47	2.6	2.13	11.04
	0.2	1.57	1.6	1.32	1.01	0.96	1.23	1.15	1.73	46.04[9]
	0.3	0.7	0.76	0.78	0.83	0.9	0.96	1.14	4.07	48.42[4]
	0.4	1.04	1.08	1.11	1.13	1.33	1.47	2.07	3.88	474.78[5]
	0.5	0.41	0.41	0.45	0.47	0.5	0.56	1.6	9.06	434.35[5]
	0.6	0.33	0.33	0.36	0.4	0.4	0.7	1.01	19.24	76.53[1]
	0.7	0.26	0.27	0.28	0.3	0.32	0.4	8.62	82.19[6]	163.76[6]
	0.8	0.23	0.23	0.24	0.25	0.29	1.75	1.64	9.4	80.41[8]
OR-CP(GP)	0.1	23.94	24.07	24.89	24.18	23.58	23.2	23.48	22.81	23.97
	0.2	14.57	14.9	15.1	15.39	15.18	16.15	16.2	18.3	19.19
	0.3	10.34	10.32	9.81	9.85	9.69	9.81	9.76	9.51	8.77
	0.4	6.88	6.93	7.02	6.93	6.79	7.09	6.59	6.9	6.23
	0.5	4.31	4.26	4.43	4.45	4.57	4.58	4.11	4.38	4.48
	0.6	2.52	2.44	2.61	2.53	2.5	2.61	2.74	2.47	2.4
	0.7	1.3	1.31	1.36	1.42	1.23	1.3	1.35	1.42	1.16
	0.8	0.48	0.52	0.54	0.55	0.59	0.59	0.58	0.61	0.6

TO: Timeout (over 2000 seconds)

Table 6.24 Max Cardinality SMTI: Search effort for $n = 50$.

Solver	$p1$	$p2$ 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Clingo	0.1	14.2	13.2	12.83	13.57	13.48	11.98	11.09	9.59	8.86
	0.2	12.89	13.4	12.45	13.82	12.6	12.1	10.64	9.91	9.07
	0.3	13.2	12.64	12.44	12.67	12.01	11.27	10.61	9.46	8.34
	0.4	12.31	12.03	12.19	12.28	11.86	11.53	10.63	9.57	8.54
	0.5	11.93	12.05	12.01	11.76	11.64	11.07	10.27	9.1	8.3
	0.6	11.66	11.68	11.59	11.65	11.15	10.55	9.99	9.5	8.72
	0.7	11.45	11.31	11.38	10.75	11.2	10.83	9.33	9.11	9.07
	0.8	10.89	10.84	10.91	11.0	10.96	10.76*	10.73	9.34	9.3
OR-CP(GP)	0.1	7.39	8.53	8.85	9.37	9.63	10.06	10.72	11.07	11.54
	0.2	6.34	7.69	8.63	9.28	9.62	10.15	10.53	11.15	11.47
	0.3	6.23	7.84	8.63	9.1	9.42	9.95	10.5	11.02	11.35
	0.4	5.98	7.47	8.15	8.61	9.63	10.13	10.43	10.8	11.14
	0.5	6.13	6.9	8.13	8.71	9.28	9.72	10.35	10.6	11.16
	0.6	5.44	6.23	7.87	8.85	9.18	9.62	10.02	10.39	10.86
	0.7	2.51	5.85	7.58	8.41	8.75	9.27	9.74	10.26	10.43
	0.8	3.69	5.83	7.01	7.73	8.51	9.34	9.46	10.05	10.01
CashWMaxsat	0.1	18.8	18.98	18.95	19.17	18.82	19.11	19.28	19.22	18.04
	0.2	8.05	7.79	8.18	6.91	6.59	7.04	7.05	6.42	6.39
	0.3	7.96	7.7	6.83	6.94	6.48	6.72	6.14	6.52	6.31
	0.4	7.43	7.48	7.54	6.44	7.27	6.24	6.66	6.43	6.04
	0.5	7.51	6.84	7.53	7.47	7.13	6.8	6.79	6.21	6.11
	0.6	7.61	7.35	7.32	6.66	7.06	6.94	6.58	6.51	5.82
	0.7	7.53	7.38	7.21	7.26	6.7	6.94	6.46	6.27	6.4
	0.8	7.9	7.28	7.68	7.16	6.83	6.85	6.4	6.3	5.94

* 1 instance reached time limit (over 2000 seconds)

Table 6.25 Egalitarian SMTI: Search effort for $n = 50$.

Solver	$p1$	$p2$ 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Clingo	0.1	13.29	13.52	13.71	14.13	14.66	16.53	18.83	23.63[6]	TO
	0.2	12.99	13.11	13.44	13.88	14.57	17.65	18.96[9]	22.75[2]	TO
	0.3	12.77	13.05	13.27	13.84	15.0	18.1	21.04	23.36[4]	TO
	0.4	12.44	12.61	12.82	13.13	15.36	17.09	21.45[8]	24.02[4]	TO
	0.5	12.19	12.3	12.57	13.33	13.88	17.99	21.88[8]	25.04[2]	TO
	0.6	11.87	11.96	12.24	13.68	14.36	18.26	22.81[9]	24.45[4]	TO
	0.7	11.53	11.56	11.93	13.08	13.59	16.62	23.84[9]	26.02[2]	TO
	0.8	10.84	11.03	11.28	12.23	14.13	22.44	21.74[9]	23.21[2]	25.92[1]
OR-CP(GP)	0.1	7.42	8.84	9.2	10.43	12.38	13.32	12.9	10.93	14.43
	0.2	6.36	7.75	8.87	10.84	11.2	12.35	12.7	12.95	11.33
	0.3	6.1	8.06	8.87	10.5	10.34	10.94	13.27	10.69	11.29
	0.4	5.63	7.68	8.4	8.96	9.61	10.01	10.14	11.3	11.04
	0.5	6.26	6.56	8.14	8.96	9.37	9.91	9.91	10.42	12.61
	0.6	5.52	6.21	8.11	8.98	9.25	9.42	10.48	10.37	10.82
	0.7	1.85	5.98	7.52	8.59	8.59	9.46	9.58	10.05	10.49
	0.8	3.58	6.03	7.21	7.81	8.79	9.02	9.23	9.6	10.02
CashWMaxsat	0.1	M	M	M	M	M	M	M	M	M
	0.2	12.04	12.1	12.13	12.12	12.23	12.28	12.39	12.66	13.01
	0.3	11.85	11.9	11.99	12.12	12.02	12.21	12.31	12.43	12.81
	0.4	11.64	11.68	11.79	11.82	11.87	12.01	12.15	12.37	12.77
	0.5	11.43	11.51	11.42	11.56	11.57	11.85	12.0	12.15	12.62
	0.6	11.15	11.1	11.39	11.4	11.45	11.6	11.77	12.06	12.5
	0.7	10.93	11.0	11.02	11.11	11.18	11.37	11.62	11.84	12.21
	0.8	10.38	10.55	10.81	10.75	11.03	11.12	11.32	11.6	11.82

M: Memory limit reached (over 4 GB)
TO: Timeout (over 2000 seconds)

Table 6.26 Sex-Equal SMTI: Search effort for $n = 50$.

Solver	$p1$	$p2$ 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Clingo	0.1	13.38	13.57	13.96	14.38	15.12	16.05	16.15	17.89[9]	17.65[5]
	0.2	13.04	13.14	13.72	14.37	15.39	15.57	15.78	16.8[9]	18.38[8]
	0.3	12.76	13.16	13.43	14.07	14.62	15.94	16.99	16.8[8]	17.11[3]
	0.4	12.5	12.7	13.01	13.05	15.24	16.04	16.85	16.57[9]	18.88[7]
	0.5	12.22	12.31	12.67	13.63	13.8	14.68	16.53	17.55	18.32[4]
	0.6	11.89	11.99	12.33	13.66	14.26	15.64	15.66	18.61[9]	TO
	0.7	11.55	11.54	12.04	13.19	14.88	15.33	19.06	21.63[3]	TO
	0.8	10.94	11.39	11.75	12.59	14.46	18.76	19.91	22.56[5]	TO
OR-CP(GP)	0.1	7.55	8.58	9.11	9.57	10.18	10.8	11.74	12.41	12.06
	0.2	6.64	7.76	8.69	9.72	10.44	12.0	12.62	13.97	14.15
	0.3	6.15	7.75	8.5	9.64	10.15	10.89	11.28	11.81	11.19
	0.4	5.93	7.39	8.16	9.07	9.85	10.22	10.35	10.94	11.17
	0.5	5.98	6.48	7.78	8.74	9.08	10.18	10.08	10.58	10.93
	0.6	5.76	6.2	7.86	8.69	8.87	9.61	10.21	10.18	12.24
	0.7	3.56	5.92	7.83	8.14	8.55	9.27	9.64	10.12	10.45
	0.8	4.49	6.07	7.13	7.65	8.29	9.07	9.27	9.66	9.9

TO: Timeout (over 2000 seconds)

7. FURTHER STUDIES ON SMTI

We extend our studies further by considering an alternative stability concept for SM. For this purpose, we study the notion of sticky-stability proposed by Afacan et al. (2016) (Section 7): we define sticky-SMTI (Section 7.1), propose an ASP method for solving sticky-SMTI (Section 7.2) and present our experimental results (Section 7.3).

Sticky-stability is introduced in paper Sticky Matching in School Choice (Afacan et al., 2016) to accommodate appeal costs in school placements. Their work focuses on the appeals that solely arise from priority violations. The appeal costs are incorporated in the traditional matching framework under the assumption that a rational parent would appeal to a violation of his/her child's priority at a school, given that appealing is more advantageous than the cost.

In their framework, students' (or parents') answers to the following question are collected: "What is the least rank difference between school pairs where you would be willing to appeal in the case a priority violation if your appeal would definitely be granted?". Based on the student's answer, his/her stickiness degree is defined as the given least rank difference minus one. Given preferences of students, their stickiness degrees, schools' priorities and capacities, a matching (placement of students into schools) is *sticky-stable* if

- no student prefers being unassigned to his/her assignment,
- any student's more preferred school has full capacity, and
- there is no student-school pair such that the rank difference between his/her assignment and the school is greater than his/her stickiness degree, and s/he has a higher priority than another student who is assigned to that school.

Note that Gale and Shapley's stability coincides with sticky-stability when each student's stickiness degree is zero.

7.1 Sticky-Stable SMTI

Afacan et al. (2016) introduce a weakening of stability that results in improved efficiency and no appeals against priority violations. Motivated by these results, we investigate the efficiency improvements in hard variants of SMTI. For that purpose, we define sticky-stable SMTI problem.

We allow men and women to personalize the level of preference violations that they would comply with, by allowing them to report their stickiness degrees. We slightly modify the stickiness degree definition to fit our definitions better. Each agent is required to specify their stickiness degree alongside their preference list. The stickiness degree of a man(woman) is equal to one less than his(her) answer to the following question: "What is the least rank difference between women(men) where you would be willing to appeal in the case a priority violation if your appeal would definitely be granted?". Let $sticky : (M \cup W) \mapsto \mathbb{N}$ be a function which maps every agent to their stickiness degree. Then, sticky-SMTI problem with sticky stability is characterized by the tuple $\langle M, W, mrank, wrank, sticky \rangle$.

A matching μ is *sticky-blocked* by a pair $(x, y) \in M \times W$, if the following hold:

S1 x and y are acceptable to each other,

S2 x and y are not married to each other (i.e., $\mu(x) \neq y$),

S3 (a) x and y are both single,

(b) $mrank(x, \mu(x)) - mrank(x, y) > sticky(x)$ and y is single,

(c) $wrank(y, \mu^{-1}(y)) - wrank(y, x) > sticky(y)$ and x is single or

(d) $mrank(x, \mu(x)) - mrank(x, y) > sticky(x)$ and $wrank(y, \mu^{-1}(y)) - wrank(y, x) > sticky(y)$.

A matching is called *sticky-stable* if it is not sticky-blocked by any pair of agents.

7.2 Solving Sticky-SMTI using ASP

The input of a sticky-SMTI instance $\langle M, W, mrank, wrank, sticky \rangle$ is formalized in ASP by extending the set of facts F_I, as described in Section 3, by atoms of the form $sticky(x, k)$ ("stickiness degree of agent x is k, i.e., $sticky(x) = k$").

We revise our ASP formulation by modifying the set of rules that describe preferences of men and women to incorporate sticky-stability. We define preferences of man x (i.e., x prefers y to $y1$) and woman y (i.e., y prefers x to $x1$) in terms of rankings and their stickiness degree.

$$mprefer(x, y, y1) \leftarrow mrank(x, y1, r), mrank(x, y, r1), sticky(x, k), r - r1 > k.$$
$$wprefer(y, x, x1) \leftarrow wrank(y, x1, r), wrank(y, x, r1), sticky(y, k), r - r1 > k.$$

7.3 Experimental Results

We conduct experiments using our ASP formulations and Clingo to investigate the effect of sticky-stability for solving hard variants of SMTI. To evaluate the sticky stability under optimization, we compare the optimization values for hard variants with traditional stability. We evaluate these comparisons by answering the following question "How much does stickiness improve optimality?". We also compare average CPU times to answer the question "How much does stickiness improve computational performance?".

The weak constraint introduced in Section 3.3 is utilized to solve Max Cardinality sticky-SMTI. The straight-forward and chaining-based weak constraints presented in Section 6.1.4 and Section 6.1.5 are utilized to solve Egalitarian sticky-SMTI and Sex-Equal sticky-SMTI, respectively. The benchmark instances are extended by assigning a stickiness degree of 1 to each man, and 0 to each woman.

In Table 7.1, the average number of singles for traditional stability are presented. The instances with incompleteness between 0.1 and 0.5 are omitted since the number of singles is mostly 0. The minimum number of singles decreases to 0 for each benchmark instance when stickiness is introduced.

Table 7.1 Max Cardinality SMTI: Average number of singles for $n = 50$.

	$p2$								
$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.6	0.2	0.4	0	0.2	0	0	0	0	0
0.7	0.6	0.4	0.8	0	0.2	0	0	0	0
0.8	3.8	1.6	1.8	1.6	0.6	0.2	0	0	0

The average CPU times for solving Max Cardinality sticky-SMTI are reported in Table 7.2. The results are comparable to the results for traditional stability. We observe that stickiness does not provide a significant improvement in computational performance. Similarly to instances with $n = 50$, the minimum number of singles decreases to 0 for each benchmark instance with stickiness. We conclude that for Max Cardinality, there is a notable improvement in optimality, but not in computational performance.

Table 7.2 Max Cardinality optimization: Average CPU-Times (in seconds).

		$p2$								
Size	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
	0.1	6.79	6.69	6.57	6.45	6.28	6.26	5.88	5.37	4.19
	0.2	4.78	4.68	4.65	4.54	4.29	4.1	3.93	3.58	2.49
	0.3	2.85	2.78	2.71	2.64	2.5	2.43	2.33	2.05	1.58
SMTI	0.4	1.74	1.74	1.69	1.7	1.53	1.52	1.37	1.28	0.93
($n = 50$)	0.5	1.04	0.99	1.0	0.98	0.9	0.91	0.75	0.69	0.52
	0.6	0.56	0.52	0.56	0.51	0.44	0.43	0.4	0.35	0.24
	0.7	0.24	0.23	0.23	0.21	0.18	0.19	0.15	0.14	0.11
	0.8	0.07	0.07	0.08	0.09	0.08	0.08*	0.07	0.05	0.04
	0.1	5.22	5.03	5.04	4.8	4.79	4.47	4.1	3.5	2.1
	0.2	3.56	3.55	3.54	3.04	2.98	2.86	2.53	2.17	1.31
	0.3	2.21	2.16	2.12	1.99	1.84	1.79	1.64	1.32	0.79
sticky-SMTI	0.4	1.32	1.28	1.23	1.23	1.17	1.2	0.96	0.77	0.38
($n = 50$)	0.5	0.77	0.82	0.74	0.67	0.64	0.61	0.46	0.36	0.22
	0.6	0.39	0.34	0.36	0.33	0.3	0.29	0.24	0.18	0.11
	0.7	0.15	0.15	0.15	0.13	0.12	0.11	0.1	0.09	0.07
	0.8	0.08	0.06	0.06	0.06	0.06	0.05	0.04	0.04	0.03
	0.1	164.37	171.73	158.44	152.13	162.43	148.38	143.9	139.34	119.09
	0.2	114.99	114.34	110.97	106.19	106.82	98.89	93.74	90.68	79.94
	0.3	62.53	61.5	57.85	63.05	59.38	55.7	56.82	48.24	43.39
SMTI	0.4	35.2	35.14	34.88	34.39	33.78	32.13	29.58	27.36	22.89
($n = 100$)	0.5	18.07	19.05	18.42	17.91	18.09	16.63	16.43	14.31	12.04
	0.6	8.57	8.4	8.34	8.11	8.33	8.58	7.25	6.83	5.39
	0.7	3.44	3.55	3.42	3.42	3.67	3.5	3.22	2.68	2.19
	0.8	1.1	1.07	1.08	1.04	1.08	1.01	0.95	0.81	0.58
	0.1	92.04	91.52	90.17	89.94	88.54	85.99	80.62	74.38	56.44
	0.2	62.69	62.62	62.77	62.42	61.92	58.78	55.34	50.5	32.92
	0.3	35.19	36.28	35.5	34.94	33.5	31.38	30.34	27.34	19.89
sticky-SMTI	0.4	26.18	25.85	24.72	24.19	23.82	22.94	21.22	16.91[9]	11.31
($n = 100$)	0.5	14.11	16.43	13.75	13.21	13.03	12.21	11.95	10.31	6.8
	0.6	8.45	12.2	7.17	6.75	6.32	6.16	5.44	4.76	2.89
	0.7	3.33	3.53	2.98	2.79	2.76	2.76	2.26	1.79	1.09
	0.8	0.99	0.92	0.91	0.81	0.77	0.7	0.59	0.47	0.28

We present the results for Egalitarian sticky-SMTI and Sex-Equal sticky-SMTI using straight-forward and chaining-based formulations in Table 7.3 and Table 7.4, respectively. We observe that the performance notably deteriorates in terms of time consumption, when compared to traditional stability. This is possibly caused by larger space of matchings due to sticky-stability. It is also interesting to observe that chaining provides better improvement for Sex-Equal sticky-SMTI than Egalitarian sticky-SMTI.

In Table 7.5 and Table 7.6, we present average optimal values computed by Clingo using chaining-based formulations for Egalitarian and Sex-Equal optimization, respectively. We observe that there is a slight improvement in optimal values for Egalitarian optimization with stickiness. Note that Clingo cannot find optimal solutions for any of the instances with $p2 \geq 0.4$ for Egalitarian sticky-SMTI, hence our comparison is restricted. For Sex-Equal optimization, we observe that there is a significant improvement in optimal values. For almost each instance, the optimal value decreases to 0 when stickiness is introduced.

We conclude that for Egalitarian optimization, the computational performance worsens while optimality slightly improves when stickiness is introduced. However, for Sex-Equal optimization, stickiness significantly improves optimality while computational performance slightly worsens.

Table 7.3 Egalitarian optimization: Average CPU-Times (in seconds) for $n = 50$.

version		$p2$								
	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
basic (sticky-SMTI)	0.1	355.25[8]	1236.92[5]	133.79[1]	TO	TO	TO	TO	TO	TO
	0.2	73.34	172.33[7]	870.15[3]	TO	TO	TO	TO	TO	TO
	0.3	81.51	227.91[6]	TO	TO	TO	TO	TO	TO	TO
	0.4	52.89	455.18[5]	TO	TO	TO	TO	TO	TO	TO
	0.5	38.43[8]	528.8[7]	1176.89[1]	TO	TO	TO	TO	TO	TO
	0.6	221.48[9]	600.6	TO	TO	TO	TO	TO	TO	TO
	0.7	88.69	844.21[5]	TO	TO	TO	TO	TO	TO	TO
	0.8	34.8	497.03[5]	695.11[1]	TO	TO	TO	TO	TO	TO
chaining (sticky-SMTI)	0.1	256.31	870.76[8]	301.61[1]	TO	TO	TO	TO	TO	TO
	0.2	95.29	409.01	1094.85[4]	TO	TO	TO	TO	TO	TO
	0.3	111.12	239.43[6]	1632.77[1]	TO	TO	TO	TO	TO	TO
	0.4	116.94	991.32[9]	TO	TO	TO	TO	TO	TO	TO
	0.5	64.47[8]	412.37[6]	669.64[1]	TO	TO	TO	TO	TO	TO
	0.6	313.35	753.16[9]	TO	TO	TO	TO	TO	TO	TO
	0.7	131.38	763.34[6]	1428.69[1]	TO	TO	TO	TO	TO	TO
	0.8	21.78	759.43[7]	597.72[1]	TO	TO	TO	TO	TO	TO
chaining (SMTI)	0.1	6.62	6.6	6.7	6.85	7.31	11.25	32.79	766.17[6]	TO
	0.2	4.38	4.41	4.52	4.63	5.04	13.48	28.54[9]	335.75[2]	TO
	0.3	2.73	2.73	2.76	2.99	3.69	12.77	104.09	522.89[4]	TO
	0.4	1.63	1.64	1.64	1.71	2.83	6.58	103.73[8]	824.53[4]	TO
	0.5	0.97	0.93	0.97	1.08	1.19	10.4	115.4[8]	1162.78[2]	TO
	0.6	0.52	0.49	0.54	0.7	0.88	10.37	263.36[9]	653.44[4]	TO
	0.7	0.2	0.2	0.22	0.3	0.36	2.77	481.47[9]	1374.47[2]	TO
	0.8	0.08	0.09	0.09	0.13	0.43	163.25	74.75[9]	167.91[2]	758.19[1]

TO: Timeout (over 2000 seconds)

Table 7.4 Sex-Equal optimization: Average CPU-Times (in seconds) for $n = 50$.

version		$p2$								
	$p1$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
basic (sticky-SMTI)	0.1	TO	TO	TO	TO	TO	TO	TO	TO	TO
	0.2	TO	TO	TO	TO	TO	TO	TO	TO	TO
	0.3	TO	TO	TO	TO	TO	TO	TO	TO	TO
	0.4	TO	TO	TO	TO	TO	TO	TO	TO	TO
	0.5	TO	TO	772.78[1]	TO	TO	TO	TO	TO	TO
	0.6	1248.89[4]	1587.36[3]	1219.22[2]	854.95[4]	663.89[5]	460.92[1]	189.56[2]	224.88[2]	TO
	0.7	640.57	737.86[9]	562.21[9]	544.95[8]	468.17[7]	183.51[6]	734.49[2]	1038.07[7]	354.71[3]
	0.8	46.36	37.31	59.58	152.56	41.83	487.05[9]	201.71	242.83	154.56[9]
chaining (sticky-SMTI)	0.1	1235.66[9]	1101.75[9]	902.55[9]	714.64[8]	526.75[9]	265.78[7]	257.97[7]	TO	417.28[1]
	0.2	596.75	775.73	553.13	458.14[9]	496.73[8]	269.84	205.59[9]	831.66[5]	1442.84[4]
	0.3	628.05	399.2	321.53	219.84	241.37	314.82[9]	246.77[6]	163.38[4]	875.59[1]
	0.4	229.63	193.25[9]	228.8	167.61	116.58	153.53[8]	144.8[8]	341.76[4]	1355.42[1]
	0.5	147.18	95.74	169.34	81.46	76.31	120.21	137.22[8]	337.76[9]	867.91[1]
	0.6	61.87	68.46	64.73	50.95	58.66	84.41	495.0[9]	420.49[8]	TO
	0.7	25.36	45.21	29.47	31.16	103.26	38.62	193.94[9]	216.74[5]	TO
	0.8	4.95	9.94	10.74	12.62	43.37	50.49	67.14[9]	244.69[4]	1393.38[1]
chaining (SMTI)	0.1	410.75	283.28	248.9	254.53	261.94	289.79	201.33	450.76[9]	442.77[5]
	0.2	257.84	173.88	172.3	171.38	215.68	186.1	114.89	223.85[9]	531.74[8]
	0.3	153.62	170.87	165.9	176.72	103.1	124.98	176.51	113.81[8]	123.51[3]
	0.4	69.78	67.29	65.9	62.12	123.9	180.76	209.12	163.38[9]	697.26[7]
	0.5	47.83	47.03	45.5	50.09	34.88	37.79	92.22	172.7	277.7[4]
	0.6	23.85	20.89	22.32	24.5	22.58	33.66	31.49	246.33[9]	TO
	0.7	9.37	8.84	8.6	9.34	12.69	16.67	160.07	1053.19[3]	TO
	0.8	2.09	2.11	1.89	2.0	3.79	55.62	134.62	940.11[5]	TO

TO: Timeout (over 2000 seconds)

Table 7.5 Egalitarian optimization: Average optimal values for Egalitarian sticky-SMTI (Average optimal values for Egalitarian SMTI) for $n = 50$.

	$p2$		
$p1$	0.1	0.2	0.3
0.1	591.8(629.2)	573.12[8](610.9)	509.0[1](543.9)
0.2	538.8(565.2)	508.5(542.3)	501.5[4](509.6)
0.3	515.1(534.7)	497.83[6](533.3)	464.0[1](472.4)
0.4	470.8(500.5)	446.22[9](474.2)	TO(455.3)
0.5	412.5[8](463.3)	391.5[6](434.2)	347.0[1](410.4)
0.6	373.2(407.5)	350.0[9](380.9)	TO(360.9)
0.7	308.2(345.0)	288.5[6](331.7)	302.0[1](310.5)
0.8	226.9(259.0)	224.29[7](255.7)	213.0[1](228.7)

TO: Timeout (over 2000 seconds)

Table 7.6 Sex-Equal optimization: Average optimal values for Sex-Equal sticky-SMTI (Average optimal values for Sex-Equal SMTI) for $n = 50$.

	$p2$					
$p1$	0.1	0.2	0.3	0.4	0.5	0.6
0.1	0.11[9](11.7)	0.0[9](3.9)	0.0[9](2.6)	0.0[8](0.6)	0.0[9](1.6)	0.0[7](0.2)
0.2	0.0(13.6)	0.0(15.2)	0.0(2.1)	0.0[9](1.1)	0.0[8](0.6)	0.0(0.1)
0.3	0.0(9.9)	0.0(6.9)	0.0(1.1)	0.0(0.3)	0.0(0.1)	0.0[9](0.1)
0.4	0.0(38.3)	0.0[9](2.6)	0.0(4.6)	0.0(10.2)	0.0(5.2)	0.0[8](5.3)
0.5	0.0(28.1)	0.0(10.6)	0.0(9.0)	0.0(3.3)	0.0(1.1)	0.0(0.2)
0.6	0.0(16.3)	0.0(16.4)	0.0(10.2)	0.0(1.9)	0.0(1.8)	0.0(3.6)
0.7	0.0(40.2)	0.0(18.2)	0.0(4.1)	0.0(1.5)	0.0(12.1)	0.0(4.9)
0.8	0.0(18.6)	0.0(8.5)	0.0(4.2)	0.0(3.2)	0.0(10.6)	0.0(2.2)

8. RELATED WORK

Numerous ASP, LP, CP and SAT models have been introduced to solve stable matching problems.

8.1 ASP Methods for Stable Matching Problems

Clercq, Schockaert, Cock & Nowe (2016) introduced an ASP method for SMTI that is flexible enough to compute stable matchings for the three-dimensional variant, which is called 3D-SMTI. Their method is also demonstrably flexible enough to select matchings based on Sex-Equal, Egalitarian, Minimum Regret and Max Cardinality optimization. Amendola (2018) investigated an ASP method for SM that is derived from its abstract argumentation modeling. They introduce an incoherent ASP program to solve Stable Roommates Problem by applying the same approach. Erdem, Fidan, Manlove & Prosser (2020) introduced an ASP method to compute stable matchings for Stable Roommate Problems with Ties and Incomplete lists (SRTI). The authors also introduced weak constraints to solve Almost, Egalitarian and Rank Maximal SRTI. Additionally, the authors provided an empirical analysis for comparing their method to Amendola's ASP method.

8.2 CP Methods for Stable Marriage Problems

Gent, Irving, Manlove, Prosser & Smith (2001) presented two CP encodings to solve Stable Marriage Problem (SM) and Stable Marriage Problem with Incomplete lists (SMI). For the first encoding, reduced domains are obtained upon enforcing

AC which correspond to GS-lists that are obtained from Extended Gale-Shapley algorithm (EGS). Second is a Boolean encoding based on which a SAT encoding was introduced by Gent, Prosser, Smith & Walsh (2002) for SMTI. They also conducted experiments, using random instance generator of Gent & Prosser (2002) to evaluate computing a complete matching for SMTI, if one exists. In Gent & Prosser (2002), authors extend their first encoding to provide a CP model for finding a complete matching for SMTI, and finding the largest or smallest stable matching. They provided the first empirical study of the complete solutions on the related decision and optimisation problem by using Choco solver.

8.3 SAT Methods for Stable Matching Problems

Drummond et al. (2015) presented a SAT and an IP encoding (SAT-E and IP-E) for solving SMP-C, where SAT-E is a novel approach. In their study, SMP-C (stable matching with couples) is cast as the problem of matching residents to hospital programs. In SMP-C, both residents (hospitals) express their preferences over hospitals (residents) whom they accept being matched with, in terms of preference lists. There may be some couples of residents each of which provides a joint preference list over hospitals. The aim is to find a stable matching in which no resident-hospital pair has motive to appeal. An empirical study is presented where both of which are state-of-the-art solvers Lingeling (Biere, 2010) and CPLEX are used to solve SAT-E and IP-E, respectively. They also implemented two existing deferred acceptance (DA) style algorithms and provide an empirical analysis to compare them with SAT and IP approaches. They showed that SAT is an effective method that outperforms IP and DA approaches.

8.4 Stability

Due to significant efficiency losses caused by traditional stability, researchers have proposed several ways of weakening stability such as “pseudo-stability” (Abdulkadiroglu, 2011), “individual trade stability” (Papai, 2013), “justness” (Morrill, 2015), and ”sticky-stability” (Afacan et al., 2016). Sticky-stability was introduced to ac-

commodate appeal costs in school placements. Authors introduce two sticky-stable mechanisms in school choice domain which are more efficient than DA. Although being prone to misreporting under complete information case, the mechanisms show advantageous strategic qualities for obtaining sticky-stable matchings. Moreover, such mechanisms yield more efficient and appeal-free outcomes.

9. CONCLUSIONS

We have conducted an empirical study to compare a variety of methods to solve hard variants of SMTI: Max Cardinality, Sex-Equal and Egalitarian SMTI. We have introduced a novel Answer Set Programming (ASP) formulation that utilizes weak constraints, which is implemented for CLINGO. We have implemented the CP models proposed by Gent & Prosser (2002) for Google OR-Tools and Choco solver, and replicate their experiments for finding a complete stable matching and finding a largest stable matching. We used their method for finding a largest stable matching to solve Max Cardinality SMTI and adapted it for solving other hard variants. We also adapted the SAT formulation of Drummond et al. (2015) to solve SMTI along with Max Cardinality and Egalitarian SMTI. We have empirically compared these declarative methods over randomly generated instances. We also compared them to existing ILP and local search approaches. For this purpose, we have used an existing Integer Linear Programming model of Max Cardinality SMTI which is adapted it for solving other hard variants and implemented for Gurobi and Google-OR Tools (MIP and CP). We have used the implementations of two different existing local search algorithms to solve Max Cardinality SMTI.

We have also performed experiments to compare Answer Set Programming with Propositional Satisfiability, over SMTI instances. For the latter, we have utilized CMODELS with the SAT solver ZCHAFF, and SAT-E implementation with the SAT solver LINGELING.

There are several important discussions. First of all, modeling is an important step in all these problem-solving methods. For that reason, we have utilized the existing and empirically evaluated models in the literature, for SAT, ILP and local search. We have come up with our own ASP formulation for SMTI variants after trying different formulations and considering elaboration tolerance, as the existing ASP formulations of the stable marriage problems used as benchmarks in the ASP competitions address SMT problem (a tractable variant of SM (Irving, 1994)).

In our experiments that we replicate from Gent & Prosser (2002), we observe simi-

lar results in terms of search effort and solubility for OR-Tools while our results for Choco solver are inconsistent with Gent & Prosser's. In our comparison of methods for optimization variants, it interesting to observe that the declarative methods (ASP, SAT, ILP, CP) are more promising than local search algorithms as the problems get harder with more ties and incompleteness. Our experimental results also indicate that while ASP, SAT and CP approaches are mostly comparable, ILP approaches are generally more efficient. It is also notable that the CP model of Gent & Prosser (2002) does not scale well for Max Cardinality SMTI, in terms of memory consumption.

Additionally, we have described a variation of SMTI which is based on sticky-stability (Afacan et al., 2016), called sticky-SMTI. We have presented the experimental results for solving Max Cardinality, Egalitarian, and Sex-Equal sticky-SMTI. We have observed a significant improvement in optimality for Max Cardinality and Sex-Equal optimization while the CPU times are comparable with traditional stability.

Based on our experiences with the declarative programming paradigms, the difficulty of adapting existing models for SMTI and its hard variants was similar for ILP and CP. Adapting the SAT formulation was challenging due to edge cases that stem from ties in preference lists. It was easier to implement existing models in ILP and CP using the state-of-the-art solvers. Although ASP is a substantially different paradigm, inventing a straight-forward model for SMTI was relatively easy given a basic familiarity with the syntax and semantics. Moreover, observing answer sets makes debugging much easier compared to others.

For future work, there are interesting problems regarding our methodology and stable matching problems. For instance, solving Sex-Equal SMTI using SAT is a future work for us. We are also interested in extending our formulations to solve other stable matching problems such as Stable Roommate Problem with Ties and Incomplete lists (SRTI), and to solve other optimization variants of SMTI such as Minimum Regret SMTI and Rank Maximal SMTI. We believe that an extended empirical study that compares these methods for solving diverse stable matching problems will be valuable to better understand the strengths of declarative methods.

BIBLIOGRAPHY

Abdulkadiroglu, A. (2011). Generalized matching for school choice. Working Paper, Duke University.

Afacan, M. O., Aliogullari, Z. H., & Barlo, M. (2016). Sticky matching in school choice. *Economic Theory*, *64*(3), 509–538.

Alkan, A. & Gale, D. (2003). Stable schedule matching under revealed preference. *Journal of Economic Theory*, *112*(2), 289–306.

Alkan, A. & Moulin, H. (2003). *Mathematical theories of allocation of discrete resources: equilibria, matchings, mechanisms*. Elsevier.

Amendola, G. (2018). Solving the stable roommates problem using incoherent answer set programs. In *Proceedings of the RiCeRcA Workshop co-located with the 17th International Conference of the Italian Association for Artificial Intelligence*, volume 2272 of *CEUR Workshop Proceedings*.

Biere, A. (2010). Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT race 2010. Technical report, FMV Technical Reports 10/1.

Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.). (2009). *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Brewka, G., Eiter, T., & Truszczynski, M. (2016). Answer set programming: An introduction to the special issue. *AI Magazine*, *37*(3), 5–6.

Buccafurri, F., Leone, N., & Rullo, P. (2000). Enhancing disjunctive datalog by constraints. *IEEE Trans. Knowl. Data Eng.*, *12*(5), 845–860.

Clercq, S. D., Schockaert, S., Cock, M. D., & Nowe, A. (2016). Solving stable matching problems using answer set programming. *Theory Pract. Log. Program.*, *16*(3), 247–268.

Drummond, J., Perrault, A., & Bacchus, F. (2015). SAT is an effective and complete method for solving stable matching problems with couples. In *Proc. of IJCAI*, (pp. 518–525).

Erdem, E., Fidan, M., Manlove, D. F., & Prosser, P. (2020). A general framework for stable roommates problems using answer set programming. *Theory Pract. Log. Program.*, *20*(6), 911–925.

Eyupoglu, S., Fidan, M., Gulesen, Y., Izci, I., Teber, B., Yilmaz, B., Alkan, A., & Erdem, E. (2021). Proceedings 37th international conference on logic programming (technical communications): Stable marriage problems with ties and incomplete preferences: An empirical study. *Electronic Proceedings in Theoretical Computer Science*, *345*, 189–190.

Freuder, E. C. & Mackworth, A. K. (2006). Constraint satisfaction: An emerging paradigm. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence* (pp. 13–27). Elsevier.

Gale, D. & Shapley, L. S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, *69*(1), 9–15.

Gärdenfors, P. (1975). Match making: Assignments based on bilateral preferences. *Behavioral Science*, *20*(3), 166–173.

Gelain, M., Pini, M. S., Rossi, F., Venable, K. B., & Walsh, T. (2013). Local search approaches in stable matching problems. *Algorithms*, *6*(4), 591–617.

Gelfond, M. & Lifschitz, V. (2000). The stable model semantics for logic programming. *Logic Programming*, *2*, 10.

Gent, I. P., Irving, R. W., Manlove, D. F., Prosser, P., & Smith, B. M. (2001). A constraint programming approach to the stable marriage problem. In *Proc. of CP'01*, (pp. 225–239).

Gent, I. P. & Prosser, P. (2002). An empirical study of the stable marriage problem with ties and incomplete lists. In *Proc. of ECAI*, (pp. 141–145).

Gent, I. P., Prosser, P., Smith, B., & Walsh, T. (2002). SAT encodings of the stable marriage problem with ties and incomplete lists. In *Proc. of SAT*, (pp. 133–140).

Giunchiglia, E., Lierler, Y., & Maratea, M. (2004). SAT-based answer set programming. In *Proc. of AAAI*, (pp. 61–66).

Gomes, C. P., Kautz, H. A., Sabharwal, A., & Selman, B. (2008). Satisfiability solvers. In F. van Harmelen, V. Lifschitz, & B. W. Porter (Eds.), *Handbook of Knowledge Representation*, volume 3 (pp. 89–134). Elsevier.

Gusfield, D. & Irving, R. W. (1989). *The Stable Marriage Problem: Structure and Algorithms*. MIT Press.

Haas, C. (2021). Two-sided matching with indifferences: Using heuristics to improve properties of stable matchings. *Computational Economics*, *57*(4), 1115–1148.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press.

Irving, R. W. (1994). Stable marriage and indifference. *Discrete Applied Mathematics*, *48*(3), 261–272.

Iwama, K., Manlove, D. F., Miyazaki, S., & Morita, Y. (1999). Stable marriage with incomplete lists and ties. In Wiedermann, J., van Emde Boas, P., & Nielsen, M. (Eds.), *Automata, Languages and Programming*, volume 1644, (pp. 443–452). Springer.

Jaffar, J. & Lassez, J.-L. (1987). Constraint logic programming. In *Proc. of POPL*, (pp. 111–119).

Järvisalo, M., Le Berre, D., Roussel, O., & Simon, L. (2012). The international sat solver competitions. *AI Magazine*, *33*(1), 89–92.

Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management Science*, *6*(4), 366–422.

Kato, A. (1993). Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics*, *10*(1), 1–19.

Kwanashie, A. & Manlove, D. F. (2014). An integer programming approach to the hospitals/residents problem with ties. In *Proc. of Operations Research*, (pp. 263–269).

Lei, Z., Cai, S., Wang, D., Peng, Y., Geng, F., Wan, D., Deng, Y., & Lu, P. (2021). Cashwmaxsat: Solver description. In *MaxSAT Evaluation 2021*.

Li, C. M. & Manyà, F. (2009). Maxsat, hard and soft constraints. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications* (pp. 613–631). IOS Press.

Lifschitz, V. (2002). Answer set programming and plan generation. *Artificial Intelligence*, *138*(1-2), 39–54.

Lin, S. & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, *21*(2), 498–516.

Manlove, D. (2014). Algorithmics of matching under preferences. In *Bull. EATCS*. World Scientific Publishing Co. Pte. Ltd.

Manlove, D. F., Irving, R. W., Iwama, K., Miyazaki, S., & Morita, Y. (2002). Hard variants of stable marriage. *Theoretical Computer Science*, *276*(1-2), 261–279.

Manlove, D. F. & O'Malley, G. (2014). Paired and altruistic kidney donation in the UK: algorithms and experimentation. *ACM J. Exp. Algorithmics*, *19*(1).

Marek, V. W. & Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm* (pp. 375–398). Springer.

Morrill, T. (2015). Making just school assignments. *Games and Economic Behavior*, *92*, 18–27.

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: engineering an efficient sat solver. In *Proc. of IEEE DAC*, (pp. 530–535).

Niemelä, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, *25*(3), 241–273.

Papai, S. (2013). Matching with minimal priority rights. Working Paper, Concordia University.

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier.

Roth, A. E., Sonmez, T., & Utku Unver, M. (2005). Pairwise kidney exchange. *Journal of Economic Theory*, *125*(2), 151–188.

Roth, A. E. & Sotomayor, M. (1992). Two-sided matching. *Handbook of game theory with economic applications*, *1*, 485–541.

Selman, B. & Gomes, C. P. (2006). Hill-climbing search. In *Encyclopedia of Cognitive Science*. American Cancer Society.

Simons, P., Niemelä, I., & Soininen, T. (2002). Extending and implementing the stable model semantics. *Artif. Intell.*, *138*(1), 181–234.

Van Hentenryck, P. (1989). *Constraint Satisfaction in Logic Programming*. MIT Press.

van Hoeve, W. & Katriel, I. (2006). Global constraints. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence* (pp. 169–208). Elsevier.

APPENDIX A

ASP formulations

In Figure A.1, we present the ASP formulation presented to Clingo for solving SMTI, which is introduced in Section 3.2.

In addition to the ASP formulation presented in Figure A.1, corresponding straightforward weak constraints are presented to Clingo, for solving Max Cardinality SMTI (Fig. A.2), Egalitarian SMTI (Fig. A.3) and Sex-Equal SMTI (Fig. A.7). These weak constraints are introduced in Section 3.3.

We also present alternative versions of weak constraints presented to Clingo for solving Egalitarian SMTI in Figure A.4 (version 1), Figure A.5 (version 2), and Figure A.6 (chaining), and Sex-Equal SMTI in Figure A.8 (chaining) which are introduced in Section 6.1.

Implementations of Gent and Prosser's CP model

In Figure A.9, we present our Python implementation of Gent and Prosser's CP model presented to OR-Tools' CP-SAT solver, for finding a stable matching. We also present our implementations presented to CP-SAT solver, in addition to the implementation for finding a stable matching, for solving Max Cardinality SMTI, Egalitarian SMTI, and Sex-Equal SMTI in Figure A.10, Figure A.11 and Figure A.12, respectively.

```
% man M prefers W to being single
maccept(M, W) :- man(M), woman(W), mrank(M, W, R).

% woman W prefers man M to being single
waccept(W, M) :- man(M), woman(W), wrank(W, M, R).

% acceptability
acceptable(M, W) :- maccept(M, W), waccept(W, M).

% generate matching
{marry(M, W) : woman(W), acceptable(M, W)}1 :- man(M).
:- {marry(M, W) : man(M)} > 1, woman(W).

% singles
msingle(M) :- man(M), {marry(M, W) : woman(W)}0.
wsingle(W) :- woman(W), {marry(M, W) : man(M)}0.

% man M prefers woman W1 to woman W2.
mprefer(M, W1, W2) :- mrank(M, W1, R1), mrank(M, W2, R2), R1 < R2.

% woman W prefers man M1 to man M2.
wprefer(W, M1, M2) :- wrank(W, M1, R1), wrank(W, M2, R2), R1 < R2.

% M1 and W2 form a blocking pair.
:- man(M1; M2), woman(W1; W2), marry(M1, W1), marry(M2, W2),
                    mprefer(M1, W2, W1), wprefer(W2, M1, M2).
% M and W2 form a blocking pair.
:- man(M), woman(W1; W2), marry(M, W1), wsingle(W2),
                          acceptable(M, W2), mprefer(M, W2, W1).
% M1 and W2 form a blocking pair.
:- man(M1; M2), woman(W), marry(M1, W), msingle(M2),
                          acceptable(M2, W), wprefer(W, M2, M1).
% M and W form a blocking pair.
:- man(M), woman(W), msingle(M), wsingle(W), acceptable(M, W).

#show marry/2.
#show msingle/1.
#show wsingle/1.
```

Figure A.1 ASP formulation for solving SMTI

```
:~ msingle(M). [1@1,m,M]
:~ wsingle(W). [1@1,w,W]
```

Figure A.2 Weak constraint for solving Max Cardinality SMTI

```
:~ marry(M, W), mrank(M, W, R1), wrank(W, M, R2). [R1+R2@1,M,W]
```

Figure A.3 Weak constraint (basic) for solving Egalitarian SMTI

```
mancost(M, R1) :- marry(M, W), mrank(M, W, R1).
womancost(M, R2) :- marry(M, W), wrank(W, M, R2).
:~ mancost(M, R1). [R1@1, m, M]
:~ womancost(W, R2). [R2@1, w, W]
```

Figure A.4 Weak constraint (version 1) for solving Egalitarian SMTI

```
:~ marry(M, W), mrank(M, W, R1). [R1@1,m,M]
:~ marry(M, W), wrank(W, M, R2). [R2@1,w,W]
```

Figure A.5 Weak constraint (version 2) for solving Egalitarian SMTI

```
mcost(M, R1) :- marry(M, W), mrank(M, W, R1).
mcost(M, R1 - 1) :- mcost(M, R1), R1 > 1.

wcost(W, R2) :- marry(M, W), wrank(W, M, R2).
wcost(W, R2 - 1) :- wcost(W, R2), R2 > 1.

:~ mcost(M,C). [1@1, m, M, C]
:~ wcost(W,C). [1@1, w, W, C]
```

Figure A.6 Weak constraint with chaining (version 3) for solving Egalitarian SMTI

```
:~ T = #sum{R1-R2, M, W: marry(M, W), mrank(M, W, R1),
                                      wrank(W, M, R2)}. [|T|@1]
```

Figure A.7 Weak constraint (basic) for solving Sex-Equal SMTI

```
mcost(M, R1) :- marry(M, W), mrank(M, W, R1).
mcost(M, R1 - 1) :- mcost(M, R1), R1 > 1.

wcost(W, R2) :- marry(M, W), wrank(W, M, R2).
wcost(W, R2 - 1) :- wcost(W, R2), R2 > 1.

mcost(T) :- T = #count{1,C,M: mcost(M,C)}.
wcost(T) :- T = #count{1,C,W: wcost(W,C)}.

cost(|T1-T2|) :- mcost(T1), wcost(T2).
cost(T-1)  :- cost(T), T>1.
:~ cost(T), T!=0. [1@1,T]
```

Figure A.8 Weak constraint (chaining) for solving Sex-Equal SMTI

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()
x = {}
y = {}
# creating variables
for mIndex in range(1, numberOfMan+1):
    # for each man, create a variable with
    #     the domain of his acceptable partners
    x[mIndex] = model.NewIntVarFromDomain(cp_model.Domain.FromValues(
        getAcceptableWomenSet(mIndex)), name='m{}'.format(mIndex))
for wIndex in range(1, numberOfWoman+1):
    # for each woman, create a variable with
    #     the domain of his acceptable partners
    y[wIndex] = model.NewIntVarFromDomain(cp_model.Domain.FromValues(
        getAcceptableMenSet(wIndex)), name='w{}'.format(wIndex))

# for each man-woman pair, check the conditions
#           where an illegal marriage or a blocking pair occur
for mIndex in range(1, numberOfMan+1):
    for wIndex in range(1, numberOfWoman+1):
        iswm, i = isWomanInManList(mIndex, wIndex)
        ismw, j = isManInWomanList(mIndex, wIndex)
        # ensure that they are mutually acceptable
        if iswm and ismw:
            # eliminate illegal marriages
            mpref = breakTies(manDict[mIndex])
            wpref = breakTies(womanDict[wIndex])
            widx = mpref.index(wIndex)
            midx = wpref.index(mIndex)
            for k in range(len(mpref)):
                if k != widx:
                    # ensure that there cannot be the case where man
                    # with mIndex is matched to a woman other than woman
                    # with wIndex, but woman with wIndex is matched to him
                    model.AddForbiddenAssignments([x[mIndex], y[wIndex]],
                                                  [(mpref[k], mIndex)])

            for l in range(len(wpref)):
                if l != midx:
                    # ensure that there cannot be the case where woman
                    # with wIndex is matched to a man other than man
                    # with mIndex, but man with mIndex is matched to her
                    model.AddForbiddenAssignments([y[wIndex], x[mIndex]],
                                                  [(wpref[l], wIndex)])
            # eliminate blocking pairs
            # find next woman/man in the pref lists of mIndex/wIndex
            nextwidx, nextmidx = findNext(mIndex, wIndex)
            # ensure that next elements exist for both
            if nextwidx != -1 and nextmidx != -1:
                for k in range(nextwidx, len(mpref)):
                    for l in range(nextmidx, len(wpref)):
                   # ensure that there cannot be the case where man
                    # with mIndex and woman with wIndex are both matched to
                    # less preferred partners
                        model.AddForbiddenAssignments(
                        [y[wIndex], x[mIndex]],[(wpref[l], mpref[k])])
```

Figure A.9 Python implementation of Gent and Prosser's CP model for SMTI

```
# Max Cardinality optimization
mm_vars = [model.NewBoolVar('mm{}'.format(mIndex))
                            for mIndex in range(1,numberOfMan+1)]
for i, mvar in enumerate(mm_vars):
    model.Add(x[i+1] <= numberOfWoman).OnlyEnforceIf(mvar)
    model.Add(x[i+1] > numberOfWoman).OnlyEnforceIf(mvar.Not())
model.Maximize(sum(mm_vars))
```

Figure A.10 Python implementation for solving Max Cardinality SMTI

```
# Egalitarian optimization
costs = []
for mIndex in range(1, numberOfMan + 1):
    for wIndex in range(1, numberOfWoman + 1):
        if mrank[mIndex][wIndex] is not False and
                            inst[wIndex][mIndex] is not False:
            b = model.NewBoolVar()
            cost = model.NewIntVar(0, 2*numberOfMan + 1)
            # ensure b_ij is true if and only if x_i and y_j are married
            model.Add(x[mIndex] == wIndex).OnlyEnforceIf(b)
            model.Add(x[mIndex] != wIndex).OnlyEnforceIf(b.Not())
            # cost for x_i and y_j is 0 if they are not married,
            # else it is equal to the sum of ranks
            # that they give to each other
            model.Add(cost == 0).OnlyEnforceIf(b.Not())
            model.Add(cost == (mrank[mIndex][wIndex] +
                               wrank[wIndex][mIndex])).OnlyEnforceIf(b)
            costs.append(cost)
# minimize total cost
model.Minimize(sum(costs))
```

Figure A.11 Python implementation for solving Egalitarian SMTI

```
mcosts = []
wcosts = []
z = model.NewIntVar(0, numberOfMan**2, 'z')
for mIndex in range(1, numberOfMan + 1):
    for wIndex in range(1, numberOfWoman + 1):
        if inst.mrank[mIndex][wIndex] is not False and
                    inst.wrank[wIndex][mIndex] is not False:
            b = model.NewBoolVar(str(mIndex) + '-' + str(wIndex))
            mcost = model.NewIntVar(0, numberOfWoman, 'mcost'+str(mIndex))
            wcost = model.NewIntVar(0, numberOfMan, 'wcost'+str(wIndex))
            # ensure b_ij is true if and only if x_i and y_j are married
            model.Add(x[mIndex] == wIndex).OnlyEnforceIf(b)
            model.Add(x[mIndex] != wIndex).OnlyEnforceIf(b.Not())
            # ensure mcost for the pair (x_i,y_j)
                                    equals to the mrank(x_i, y_j)
            model.Add(mcost == 0).OnlyEnforceIf(b.Not())
            model.Add(mcost == (inst.mrank[mIndex][wIndex]))
                                                .OnlyEnforceIf(b)
            # ensure wcost for the pair (x_i,y_j)
                                    equals to the wrank(y_j, x_i)
            model.Add(wcost == 0).OnlyEnforceIf(b.Not())
            model.Add(wcost == (inst.wrank[wIndex][mIndex]))
                                                 .OnlyEnforceIf(b)
            mcosts.append(mcost)
            wcosts.append(wcost)
# ensure z equals to |sum of mcosts - sum of wcosts|
model.Add(z >= (sum(mcosts) - sum(wcosts)))
model.Add(z >= -(sum(mcosts) - sum(wcosts)))
# minimize the abs value
model.Minimize(z)
```

Figure A.12 Python implementation for solving Sex-Equal SMTI