SAT is an Effective and Complete Method for Solving Stable Matching Problems with Couples

Joanna Drummond and **Andrew Perrault** and **Fahiem Bacchus**
Department of Computer Science
University of Toronto
{jdrummond,perrault,fbacchus}@cs.toronto.edu

Abstract

Stable matchings can be computed by deferred acceptance (DA) algorithms. However such algorithms become incomplete when complementarities exist among the agent preferences: they can fail to find a stable matching even when one exists. In this paper we examine stable matching problems arising from labour market with couples (SMP-C). The classical problem of matching residents into hospital programs is an example. Couples introduce complementarities under which DA algorithms become incomplete. In fact, SMP-C is NP-complete.

Inspired by advances in SAT and integer programming (IP) solvers we investigate encoding SMP-C into SAT and IP and then using state-of-the-art SAT and IP solvers to solve it. We also implemented two previous DA algorithms. After comparing the performance of these different solution methods we find that encoding to SAT can be surprisingly effective, but that our encoding to IP does not scale as well. Using our SAT encoding we are able to determine that the DA algorithms fail on a non-trivial number of cases where a stable matching exists. The SAT and IP encodings also have the property that they can verify that no stable matching exists, something that the DA algorithms cannot do.

1 Introduction

Finding a stable matching is an important problem (SMP) with many real-world applications. These include college admissions, school choice, reviewer paper matching, and various labor market matching problems. Probably the best known of these applications is the problem of matching residents into hospital programs (the residency matching problem) [Niederle *et al.*, 2008; Roth, 1984; Abdulkadiroglu *et al.*, 2005]. The National Resident Matching Program (NRMP) began in 1952 to match medical students to residency positions [NRMP, 2013]. First implemented because the market was unraveling, the NRMP has adapted to the needs of participants over the years. To address the problem of couples wanting to coordinate their placements, the NRMP began allowing couples to jointly express their preferences over residency programs. This gave rise to interest in solving the Stable Matching Problem with Couples (SMP-C).

Intuitively, given participants' preferences, we want to find a match under which no pair (one from each side of the market) has an incentive to defect. Such a matching is called *stable*. In their seminal paper, Gale and Shapley provided a polynomial time deferred acceptance (DA) algorithm to solve SMP [Gale and Shapley, 1962]. They also showed that a stable matching always exists and that their DA algorithm always finds one. However, once we add couples a stable matching might not exist, and the problem of finding one becomes NP-complete [Ronn, 1990].

To address this complexity previous literature has investigated techniques to find matches that are not guaranteed to be stable, such as local search techniques [Marx and Schlotter, 2011]; or techniques that find stable matches, but are not complete [Roth and Peranson, 1999; Kojima *et al.*, 2013]. These latter works focus on extending the classic Gale-Shapley DA algorithm to incorporate couples. However, in these extensions many nice properties of DA are lost. The resulting algorithms are not able to decide whether or not a stable matching exists; even when a stable matching exists they might not be able to find one; and any matching they are able to find is no longer guaranteed to be proposer-optimal.

Since SMP-C is NP-complete it is natural to consider encoding it into SAT or Integer Programming (IP) given the tremendous advances in SAT and IP solvers over the past decade. Solving hard social choice problems (especially stable matching problems) by encoding into SAT or IP has numerous benefits. Most important is that modern SAT and IP solvers are complete, so given sufficient compute resources, they are able to find a stable match if one exists or prove that no such matching exists. Furthermore, unlike DA-style algorithms, where the algorithm must be re-designed to incorporate new constraints, adding new constraints to a SAT or IP encoding simply involves extending the encoding.

In this paper develop a SAT and IP encoding for SMP-C (SAT-E and IP-E). SAT-E is new while IP-E is based on a previous LP encoding for SMP [Roth *et al.*, 1993]. We solve these encodings with the SAT solver Lingeling [Biere, 2013] and IBM's IP solver CPLEX$^{\text{TM}}$, both of which are state-of-the-art. We also implement two previous deferred acceptance style algorithms, RP99 [Roth and Peranson, 1999] and KPR [Kojima *et al.*, 2013]. We show that using SAT is a competitive strategy for solving these kinds of problems, solving fairly large problem instances quite quickly, outperforming IP and RP99. SAT also allows us to determine how frequently a stable matching exists and see how that changes as the percentage of couples increases.

2 Background

We cast our formalization of SMP-C (stable matching with couples) in terms of matching doctors into hospital residency programs [Roth and Peranson, 1999].

In SMP-C, doctors wish to be placed into (matched with) some hospital program. Both doctors and hospitals have preferences over who they are matched with, expressed as a ranked list. Some doctors are paired into couples, and these couples wish to provide their preferences as a joint ranked list. Both doctors and hospitals can provide incomplete lists, where they list only the hospitals (resp. doctors) they would accept being matched with (i.e., unlisted alternatives are unacceptable). We wish to find a stable matching from which no doctor-hospital pair has an incentive to defect.

Notation. Let D be a set of doctors and let P be a set of hospital programs. Each doctor is looking to be matched (placed) in a program, and each program is looking to be matched with (accept) some number of doctors. Each program $p \in P$ has an integer quota $cap_p > 0$, which is the maximum number of doctors p can accept. A **matching** μ is a function from D to P, it assigns each doctor to a specific program. We say that a doctor d is matched to program p under μ when $\mu(d) = p$, and that p is matched to d when $d \in \mu^{-1}(p)$.

$S \subset D$ is the set of single doctors. The remaining doctors $D - S$ are in a couple relationship specified by a set C of *ordered* pairs of doctors, $(d_1, d_2) \subseteq (D-S) \times (D-S)$. A doctor from $D-S$ must appear in one and only position of one pair of C (monogamy).

We use nil to denote the null "doctor" or "program": matching a program p to nil ($nil \in \mu^{-1}(p)$) indicates that p has an unfilled slot, while matching a single doctor d to nil ($\mu(d) = nil$) or a couple $c = (d_1, d_2)$ to nil ($\mu(d_1) = nil$ and $\mu(d_2) = nil$) indicates that they are unmatched. We use P^+ and D^+ to denote $P \cup \{nil\}$ and $D \cup \{nil\}$.

Each market participant a has preferences over their possible matches. These preferences are specified by a partial order relation $x \succeq_a y$ indicating that a prefers x to y or $x = y$. For single doctors d, $\succeq_d$ is a partial order over P^+, for couples c, $\succeq_c$ is a partial order over pairs from $P^+ \times P^+$, and for programs p, $\succeq_p$ is a partial order over D^+.

We say that x is **acceptable** for a if $x \succeq_a nil$. Although we allow a's preferences to be *incomplete*, we require that a has a total preference order order over its acceptable options: $(x \succ_a nil \land y \succ_a nil) \rightarrow (x \succ_a y \lor y \succ_a x)$. Preferences between unacceptable matches need not be specified. We also standardly define the relations $\prec_a$, $\succ_a$, and $\preceq_a$.

Since $\succeq_a$ is a total order over the acceptable matches, we can represent it using a rank order list, rol_a, which simply lists the acceptable matches from most to least preferred (nil is always least preferred).

We define a choice function $Ch_p()$ for programs $p \in P$. Given a set of doctors R, $Ch_p(R)$ returns the subset of R that p would prefer to accept. Intuitively, p will select from R only doctors it finds to be acceptable, no more doctors than its quota, and the doctors of R it most prefers. Hence, $Ch_p(R)$ is the maximal subset of R such that for all $d \in Ch_p(R)$, $d \succ_p nil$, for all $d' \in R - Ch_p(R)$, $d \succ_p d'$, and $|Ch_p(R)| \leq cap_p$. It is convenient to give the null program a choice function as well: $Ch_{nil}(O) = O$, i.e., nil will accept any and all matches.

We also introduce the notation $ranked(a)$ to denote the set of options that a could potentially be matched with. For single doctors d and programs p this is simply the set rol_d and rol_p. For a doctor d_1 that is part of a couple (d_1, d_2), $ranked(d_1) = \{p_1 | \exists p_2.(p_1, p_2) \in rol_{(d_1,d_2)}\} \cup \{nil\}$ and similarly for d_2. Note that $nil \in ranked(a)$.

Finally, we use the function $rank_a(x)$ to find the index of match x in a's rol: $rank_a(x) = i$ iff x appears at index i (zero-based) on rol_a or $|rol_a|$ if x does not appear on rol_a. Also we use rol_a as an indexable vector, e.g., if x is acceptable to a then $rol_a[rank_a(x)] = x$.

Solving an SMP-C instance involves finding a stable matching. Intuitively, a stable matching is a matching μ that (a) satisfies all quotas, and no doctor or program is matched to an unacceptable partner (Individual Rationality); and (b) has no pair consisting of doctor and a program, or consisting of a couple and a pair of hospitals, that has mutual incentive to defect from μ.

A pair with mutual incentive to defect from μ is called a **blocking pair**—stability requires that no blocking pair exists. A single doctor d and program p form a blocking pair if d and p prefer each other to their match under μ. If d and p are a blocking pair then d would prefer to switch from $\mu(d)$ to p and p would be happy to accept d in place of some other doctor in $\mu^{-1}(p)$ making μ unstable.

A couple $c = (d_1, d_2)$ and a pair of programs (p_1, p_2) form a blocking pair if c prefers (p_1, p_2) to its match under μ and p_1 and p_2 prefer to accept d_1 and d_2 respectively over their current match. For couples, however, we have to deal with the special case when $p_1 = p_2$: in this case p_1 must prefer to accept both d_1 and d_2 rather than keep its current match (i.e., it is not sufficient for p_1 to prefer each one individually).

Definition 1 *A **stable matching** μ for $\langle D, C, P, \succeq \rangle$, where $\succeq$ is the set of all preference relationships, is a matching that satisfies conditions 1-3 given below.*

Individual Rationality: This condition specifies that μ respects all quotas and does contain any unacceptable matches.
1a: $\forall p \in P. |\mu^{-1}(p)| \leq cap_p$
1b: $\forall d \in S. \mu(a) \in rol_a$
1c: $\forall (d_1, d_2) \in C. (\mu(d_1), \mu(d_2)) \in rol_{(d_1,d_2)}$
1d: $\forall p \in P. \forall d \in \mu^{-1}(p). d \in rol_p$

To express the stability conditions we introduce the abbreviation $willAccept(p, R, \mu) \equiv R \subseteq Ch_p(\mu^{-1}(p) \cup R)$, where p is a program and R is a set of doctors. This means that if p was allowed to choose from all of the doctors in R as well as the doctors it is matched to, $\mu^{-1}(p)$, it would choose a match that includes R. Note that if p is already matched to the doctors in R, i.e., $R \subseteq \mu^{-1}(p)$ then $willAccept(p, R, \mu)$ will be true (as long as $\mu^{-1}(p)$ is individually rational).

For doctors and couples the "will accept" condition is more simply expressed: a single doctor d will accept p over its current match if $p \succ_d \mu(d)$ and a couple $c = (d_1, d_2)$ will accept (p_1, p_2) over its current match if $(p_1, p_2) \succ_c (\mu(d_1), \mu(d_2))$.

Stability for Singles: No individual single doctor can find a

better matching than μ.
2: $\forall d{\in}S.\nexists p{\in}P.\, p \succ_d \mu(d) \wedge willAccept(p, \{d\}, \mu)$

Stability for Couples: No couple can find a better matching.

3a: $\forall(d_1, d_2){\in}C.\nexists(p_1, p_2){\in}(P^+{\times}P^+).$
$p_1 {\neq} p_2 \wedge (p_1, p_2) \succ_{(d_1,d_2)} (\mu(d_1), \mu(d_2))$
$\wedge\, willAccept(p_1, \{d_1\}, \mu) \wedge willAccept(p_2, \{d_2\}, \mu)$

3b: $\forall(d_1, d_2){\in}C.\nexists p{\in}P^+.(p, p) \succ_{(d_1,d_2)} (\mu(d_1), \mu(d_2))$
$\wedge willAccept(p, \{d_1, d_2\}, \mu)$

For case (3a) it can be that $p_1 = \mu(d_1)$ or $p_2 = \mu(d_2)$ (i.e., only one member of the couple is switching programs) but not both since $(p_1, p_2) \succ_{(d_1,d_2)} (\mu(d_1), \mu(d_2))$ (strict preference). If $p_1 = \mu(d_1)$ then, as explained above, $willAccept(p_1, \{d_1\}, \mu(d_1))$ will automatically be true, similarly when $p_2 = \mu(d_2)$. Hence case (3a) covers the case when only one member of the couple wants to switch. Case (3b) covers the case when both couples want to enter the same program p (even if one member of the couple is already in p).

3 Prior DA algorithms

The basic principle of DA algorithms is that agents from one group propose down their rol until they are accepted. Agents from the other group can reject a previously made match if they obtain a better proposal, in which case the rejected agent must continue proposing down its rol.

Roth and Peranson developed a DA algorithm, RP99, capable of dealing with couples [1999]. This well known algorithm has been used with considerable success in practice, including most famously for finding matches for the NRMP which typically involves about 30,000 doctors [NRMP, 2013]. Using the description in [Roth and Peranson, 1999] we have implemented RP99. RP99 employs an iterative scheme. After computing a stable matching for all single doctors, couples are added one at a time and a new stable matching computed after each addition. The algorithm uses DA to find a matching. Matching a couple can make previously made matches unstable and in redoing these matches the algorithm might start to cycle. Hence, cycle checking (or a timeout) is needed to terminate the algorithm. Randomization can be used to restart the algorithm to obtain a different outcome.[1]

Kojima et al. develop a simple "sequential couples algorithm" [2013]. This algorithm is analyzed to prove that the probability of a stable matchings existing goes to one under certain assumptions. However, this simple algorithm is not useful in practice as it declares failure under very simple conditions (this algorithm and analysis was extended in [Ashlagi *et al.*, 2014]). Kojima et al. also provide a more practical DA algorithm, KPR, that they use in their experiments. We have implemented KPR. The main difference between KPR and RP99 is that KPR deals with all couples at the same time—it does not attempt to compute intermediate stable matchings. As will be seen this makes KPR much more successful (and efficient) in practice.

[1]We implemented randomization but did not find a significant difference in our experiments, so we omit further discussion of randomization here.

4 Encodings to NP-Complete Problems

Recent years have seen a tremendous increase in our ability to solve instances of NP-complete problems coming from real world applications. In particular, solvers for Integer Programs (IP) and Satisfiability (SAT) have both seen orders of magnitude performance improvements.

One of the contributions of this paper is to show that SAT solvers in particular, can be quite effective for solving SMP-C. Besides solving matching problems, exact solvers can also be used to evaluate and analyze the empirical behaviour of popular DA algorithms. This can be a useful addition to recent theoretical results [Ashlagi *et al.*, 2014; Kojima *et al.*, 2013], as most of these are results that hold as the market size tends to infinity. Such results in the limit are useful, but not completely informative when dealing with markets seen in practice.

In order to use SAT or IP solvers we need to encode the matching problem into SAT and IP. A major contribution of this paper is SAT-E, an innovative SAT encoding utilizing a collection of variables that track the remaining capacity of each program at each level of that program's preference. These variables allow us to express the stability conditions more compactly. We present the details of SAT-E below. For the IP encoding, IP-E, we extended a prior linear programming (LP) encoding for solving SMP, [Roth *et al.*, 1993], to handle couples. The extension requires integrality constraints making the LP into an IP.

Little work has investigated using constraint programming, SAT solvers, or IP solvers for NP-complete stable matching problems. Some work investigated encoding Stable Matching with Incomplete Lists and Ties as either a CSP encoding or a SAT encoding (e.g., [Gent *et al.*, 2001; Unsworth and Prosser, 2005; Prosser, 2014; Manlove *et al.*, 2007; Gent and Prosser, 2002]); integer programming has been used to find minimum regret matchings with partial preference information ([Drummond and Boutilier, 2013]). To our knowledge, only one other encoding for SMP-C exists; Biró et al. independently developed an IP encoding for SMP-C [2014]. While similar to the IP encoding we developed and present in Section 4.2, the simulation results described in their paper are not comparable to ours, as we draw from very different preference distributions.

4.1 SAT-E

We assume that the doctor (singles and couples) $rols$ have been preprocessed so as to remove from them any program that does not find that doctor acceptable. For $d \in S$ we remove p from rol_d if $d \notin rol_p$. For couple $(d_1, d_2) \in C$ we remove (p_1, p_2) from $rol_{(d_1,d_2)}$ if $d_1 \notin rol_{p_1}$ or $d_2 \notin rol_{p_2}$. This ensures that individual rationality conditions **1b-c** are trivially satisfied in our encodings.

Variables. We utilize three sets of Boolean variables.
1. Doctor Matching Variables:$\{m_d[p] \mid d \in D \wedge p \in rol_d\}$. $m_d[p]$ is true iff d is matched into program p. Note that $m_d[nil]$ is true if d is unmatched.
2. Couple Matching Variables: $\{m_c[i] \mid c \in C \wedge (0 \leq i < |rol_c|)\}$. $m_c[i]$ is true iff couple c is matched into

a program pair (p, p') that it ranks between 0 and i, i.e., $0 \le rank_c((p, p')) \le i$. (Lower ranks are more preferred).

3. Program Matching Variables: $\{m_p[i, s] \mid p \in P \wedge (0 \le i \le |rol_p| - 2) \wedge (0 \le s \le \min(i+1, cap_p + 1))\}$. $m_p[i, s]$ is true iff s of the doctors in $rol_p[0]$ to $rol_p[i]$ have been matched into p. Note that i ranges up to $|rol_p| - 2$ which is the index the last non-nil doctors on rol_p (rol_p is terminated by nil).

The program matching variables are the main innovation of our encoding. We found that we could maintain the proper truth value for these variables with a small set of clauses, and with them express the stability constraints more compactly.

Now we give the **clauses** of the encoding. Rather than give the more lengthy clauses directly, we often give higher level constraints whose CNF encoding is straightforward.

Unique Match. A doctor must be matched into exactly one program (possibly the nil program). For all $d \in D$

1a. **at-most-one**$(\{m_d[p] | p \in ranked(d)\})$
1b. $\bigvee_{p \in ranked(d)} m_d[p]$

In our experiments we converted the **at-most-one** constraint to CNF using the binomial encoding which requires $O(|ranked(d)|^2)$ binary clauses but no additional variables. More compact linear encodings can be used for large $ranked(d)$ sets [Frisch and Giannaros, 2010]. **1b** ensures that some match (possibly to the nil program) is made.

Couple Match. The $m_c[*]$ variables must have their intended meaning. For all couples $c \in C$, for all k such that $1 \le k \le |rol_c|$, letting $c = (d_1, d_2)$ and $(p_1[i], p_2[i]) = rol_c[i]$,

2a. $m_c[0] \equiv m_{d_1}[p_1[0]] \wedge m_{d_2}[p_2[0]]$
2b. $m_c[k] \equiv (m_{d_1}[p_1[k]] \wedge m_{d_2}[p_2[k]]) \vee m_c[k-1]$
2c. $m_c[|rol_c|]$

The final condition ensures that c is matched to some program pair on its rol (possibly nil), and the **at-most-one** constraint for d_1 and d_2 ensures that c is uniquely matched.

Program Match. The $m_p[*, *]$ variables must have their intended meaning. For all programs $p \in P$, for all i such that $1 \le i \le |rol_p| - 2$, and for all s such that $0 \le s \le \min(i+1, cap_p + 1)$, letting $d_i = rol_p[i]$ (the i-th doctor on p's rol),

3a. $\left(m_p[0, 0] \equiv \neg m_{d_0}[p]\right) \wedge \left(m_p[0, 1] \equiv m_{d_0}[p]\right)$
3b. $m_p[i, s] \equiv (m_p[i-1, s] \wedge \neg m_{d_i}[p]) \vee (m_p[i-1, s-1] \wedge m_{d_i}[p])$
3c. $\neg m_p[i, cap_p + 1]$

The last condition, captured by a set of unit clauses, ensures that p's quota is not exceeded at any stage. Falsifying these variables along with the other clauses ensures that no more matches can be made into p once p hits its quota. Note that i only indexes up to the last non-nil doctor on rol_p since nil does not use up any program capacity.

Stability for Singles: For each single doctor, $d \in S$ and for each $p \in rol_d$

4. $\left(\bigvee_{p' \succeq_d p} m_d[p']\right) \vee m_p[rank_p(d) - 1, cap_p]$

This clause says that if d has not been matched into a program preferred to or equal to p, then it must be the case that p will not accept d. Note that $m_p[rank_p(d) - 1, cap_p]$ means that p has been filled to capacity with doctors coming before d on its rol.

Stability for Couples (A): For each couple $c = (d_1, d_2) \in C$ and for each $(p_1, p_2) \in rol_c$ **with $\mathbf{p_1 \ne p_2}$**

5a1. $m_{d_1}[p_1] \wedge \neg m_c[rank_c((p_1, p_2))] \rightarrow m_{p_2}[rank_{p_2}(d_2) - 1, cap_{p_2}]$
5a2. $m_{d_2}[p_2] \wedge \neg m_c[rank_c((p_1, p_2))] \rightarrow m_{p_1}[rank_{p_1}(d_1) - 1, cap_{p_1}]$
5b. $\neg m_{d_1}[p_1] \wedge \neg m_{d_2}[p_2] \wedge \neg m_c[rank_c((p_1, p_2))] \rightarrow m_{p_1}[rank_{p_1}(d_1) - 1, cap_{p_1}] \vee m_{p_2}[rank_{p_2}(d_2) - 1, cap_{p_2}]$

Clause **5a1** says that when d_1 is already matched to p_1 but c has not been matched into (p_1, p_2) or into a more preferred program pair, then it must be the case that p_2 will not accept d_2. **5a2** is analogous.

Clause **5b** says that if neither d_1 nor d_2 is matched into p_1 or p_2 and c has not been matched into (p_1, p_2) or into a more preferred program pair, then either p_1 will not accept d_1 or p_2 will not accept d_2.

Stability for Couples (B): For each couple $c = (d_1, d_2) \in C$ and for each $(p, p) \in rol_c$ we have one of constraint **6a1** or **6b1**. **6a1** is needed when $d_1 \succ_p d_2$, while **6b1** is needed when $d_2 \succ_p d_1$.

6a1. $m_{d_1}[p] \wedge \neg m_c[rank_c((p, p))] \rightarrow m_p[rank_p(d_2) - 1, cap_p]$
6b1. $m_{d_1}[p] \wedge \neg m_c[rank_c((p, p))] \rightarrow m_p[rank_p(d_1) - 1, cap_p - 1]$
6c. $\neg m_{d_1}[p] \wedge \neg m_{d_2}[p] \wedge \neg m_c[rank_c((p, p))] \rightarrow m_p[rank_p(d_1)-1, cap_p] \vee m_p[rank_p(d_1)-1, cap_p-1] \vee m_p[rank_p(d_2)-1, cap_p] \vee m_p[rank_p(d_2)-1, cap_p-1]$

Clauses **6a1** or **6b1** say that if d_1 is already in p and c is not matched to (p, p) or into a more preferred program pair, then p will not accept d_2. **6b1** differs because when $d_2 \succ_p d_1$ and d_1 is already in p, p will definitely accept d_2. In this case, however, the couple is not accepted into (p, p) if accepting d_2 causes d_1 to be bumped. That is, when $m_p[rank_p(d_1) - 1, cap_p - 1]$ is true (adding d_2 will cause $m_p[rank_p(d_1) - 1, cap_p]$ to become true).

There are also analogous clauses **6a2** and **6b2** (one of which is used) to deal with the case when d_2 is already in p and we need to ensure that p won't accept d_1. Clause **6c** handles the case when neither member of the couple is currently matched into p.

Let $\mathcal{P} = \langle D, C, P, \succeq \rangle$ be a matching problem. We say that a matching μ for $\mathcal{P}$ and a truth assignment π for SAT-E of $\mathcal{P}$ are **corresponding** when $\pi \models m_d[p]$ iff $\mu(d) = p$. The following theorem shows that the satisfying assignments of SAT-E are in a 1-1 relationship with the stable models.

Theorem 1 *If μ and π are corresponding, then μ is stable if and only if π is satisfying.*

Proof Sketch: First we observe that there is only one satisfying assignment for any fixed setting of the doctor matching variables $m_d[p]$—the other variables' truth assignments are determined by clause sets **1–3**. So any stable matching has a single corresponding truth assignment if it has any.

Showing that any π corresponding to a stable matching μ satisfies SAT-E is straightforward: the individual rationality conditions μ ensure that clause sets **1–3** are satisfied, and the stability conditions ensure that clause sets **4-6** are satisfied.

Suppose π is a satisfying assignment. It is similarly straightforward to see that the fact it satisfies clause sets 1–3 implies that its corresponding matching μ is individually rational. For example, clause set **3** ensures that no program is matched beyond its quota.

To see that all singles are stable in the μ let $m_d[\mu(d)]$ be true. This forces all other $m_d[p]$ variables to be false by the clauses **1a**. $m_d[\mu(d)]$ appears in and hence satisfies all of the clauses of **4** for $p \preceq_d \mu(p)$. For $p \succ_d \mu(d)$ the clauses of **4** are reduced to the units $m_p[rank_p(d)-1, cap_p]$. These units along with the clauses of **3** ensure that all programs p preferred by d to $\mu(d)$ are already filled with doctors p prefers to d. Hence none of these more preferred programs will accept d and μ must satisfy the stability condition for singles **2**.

The argument is similar for couple stability. The only other potentially complex case is clause set **6c** which handles the case where both members of a couple (d_1, d_2) would prefer to go into the same program p, and neither are matched into p. Without loss of generality, suppose $d_1 \succ_p d_2$. If p was to accept d_2, then $m_p[rank_p(d_2)-1, cap_p]$ must be false. Furthermore, if p also accepts d_1 the number of doctors accepted who are preferred by p to d_2 will go up by one. Hence, for d_2 to be accepted $m_p[rank_p(d_2)-1, cap_p-1]$ must also be false. These two imply that both $m_p[rank_p(d_1)-1, cap_p]$ and $m_p[rank_p(d_1)-1, cap_p-1]$ are false, since the number accepted before d_1 must be less than the number accepted before d_2. Thus, p will only accept (d_1, d_2) if all four conditions are false. The other clauses of **5–6** show that in all cases no couple c will be accepted into a program pair it prefers to $\mu(c)$ and hence that μ satisfies the stability condition for couples **3**.

4.2 IP-E

We briefly describe the IP encoding (IP-E) we developed to compare against SAT-E. IP-E is an extension of a previous LP encoding for stable matching without couples (SMP) [Roth *et al.*, 1993]. Constraints **1**, **2**, and **4** are the natural many-to-one extension of the constraints in the Roth et al. LP; constraints **1**, **4**, **5**, and **6** are analogous to those constraints in SAT-E.

Variables. We utilize three sets of binary variables.

1. Couple Matching Variables: $\{m_c[(p_1,p_2)] \mid c \in C \wedge (p_1,p_2) \in rol_c\}$. $m_c[(p_1,p_2)] = 1$ iff m_c is matched to (p_1,p_2).

2. Doctor Matching Variables: $\{m_d[p] \mid d \in D \wedge p \in rol_d\}$. $m_d[p] = 1$ iff m_d is matched to p. If d is in a couple $c = (d, d')$, $m_d[p]$ is an abbreviation for $\sum_{p' \mid (p,p') \in rol_c} m_c[(p,p')]$.

3. Auxiliary Variables: $\{\alpha_{d_2,p_2} \mid c = (d_1,d_2) \in C \wedge p_2 \in ranked(d_2)\}$. α_{d_2,p_2} being true implies that p_2 is full to capacity with doctors it prefers to d_2.

Unique Match. A doctor must be matched into exactly one program (possibly the *nil* program). For all $d \in D$

$$\mathbf{1}.\quad \sum_{p \in ranked(d)} m_d[p] = 1$$

Program Capacity. For all $p \in P$,

$$\mathbf{2}.\ \sum_{d \in ranked(p)} m_d[p] \le cap_p$$

Auxiliary Variables. The α_{d_2,p_2} variables capture capacity/preference constraints. For all $c = (d_1, d_2) \in C$ and for each $p_2 \in ranked(d_2)$

$$\mathbf{3}.\ \sum_{d' \succeq_{p_2} d_2} m_{d'}[p_2] \ge cap_{p_2} \alpha_{d_2,p_2}$$

Stability for Singles: For each single doctor, $d \in S$ and for each $p \in rol_d$

$$\mathbf{4}.\ \sum_{d' \succeq_p d} m_{d'}[p] + cap_p \sum_{p' \succeq_d p} m_d[p'] \ge cap_p$$

Intuitively, for all couple stability constraints, the first line checks if the constraint is applicable, the second line checks if the couple is already matched to a program pair at least as desirable as the potential blocking program pair, and the third line checks if the program or programs are full. If the constraint is applicable, only one of the terms must be "true" to prevent a blocking pair.

Stability for Couples (A): For each couple $c = (d_1, d_2) \in C$ and for each $(p_1, p_2) \in rol_c$ **with** $p_1 \neq p_2$

$$\begin{aligned}\mathbf{5a1}.\ & cap_{p_2}(1 - m_{d_1}[p_1]) \\ & + cap_{p_2} \textstyle\sum_{(p_1',p_2') \succeq_c (p_1,p_2)} m_c[(p_1',p_2')] \\ & + \textstyle\sum_{d' \succeq_{p_2} d_2} m_{d'}[p_2] \ge cap_{p_2} \\ \mathbf{5b}.\ & cap_{p_1} m_{d_1}[p_1] + cap_{p_1} m_{d_2}[p_2] \\ & + cap_{p_1} \textstyle\sum_{(p_1',p_2') \succeq_c (p_1,p_2)} m_c[(p_1',p_2')] \\ & + \textstyle\sum_{d' \succeq_{p_1} d_1} m_{d'}[p_1] + cap_{p_1} \alpha_{d_2,p_2} \ge cap_{p_1}\end{aligned}$$

There is an analogous clause **5a2** to deal with the case when d_2 is already in p_2 and we need to ensure that p_1 won't accept d_1.

Stability for Couples (B): We assume, without loss of generality, that $d_1 \succ_p d_2$. For each couple $c = (d_1, d_2) \in C$ and for each $(p, p) \in rol_c$

$$\begin{aligned}\mathbf{6a1}.\ & cap_p(1 - m_{d_1}[p]) \\ & + cap_p \textstyle\sum_{(p_1',p_2') \succeq_c (p,p)} m_c[(p_1',p_2')] \\ & + \textstyle\sum_{d' \succeq_p d_2} m_{d'}[p] \ge cap_p \\ \mathbf{6b}.\ & cap_p m_{d_1}[p] + cap_p m_{d_2}[p] \\ & + cap_p \textstyle\sum_{(p_1',p_2') \succeq_c (p,p)} m_c[(p_1',p_2')] \\ & + \textstyle\sum_{d' \succeq_p d_1} m_{d'}[p] + cap_p \alpha_{d_2,p} \ge cap_p - 1\end{aligned}$$

There is an analogous clause **6a2** to deal with the case when d_2 is already in p and we need to ensure that p won't accept d_1.

5 Empirical Results

We evaluate our SAT encoding, SAT-E (Sec. 4.1), by comparing it to previously published DA algorithms KPR and RP99 (Sec. 3) and to our IP encoding, IP-E (Sec. 4.2). We use IBM's CPLEX$^{\text{TM}}$ system to solve IP-E, and Lingeling [Biere, 2013] to solve SAT-E. All instances were run on Intel Xeon E5540 2.53GHz CPUs. We used a timeout of 5400 seconds and a memory limit of 16GB. We draw instances from the same uniform preferences model presented by Kojima et al. [2013] (Section B of the online appendix). In our experiments we drew 50 instances per instantiation of the parameters.

We first evaluate one-to-one matching problems. In these instances we set the number of programs to be equal to the number of single doctors, and included some extra number of couples. We focus on the highest density of couples in the market presented in Kojima et al.'s empirical evaluation [2013], as these are the more difficult cases for their KPR algorithm. All results for the one-to-one instances are presented in Tables 1 and 2. Table 1 shows satisfiability performance results, and Table 2 shows runtimes.

For Table 1, the fraction of satisfiable instances was calculated as follows. SAT-E solved all instances of size less than size 50,000. In these cases it either found a stable matching or proved that none exists. Hence we know the exact fraction of instances that are satisfiable up to size 20,000. For 50,000, KPR successfully solved 86% of all instances, giving us a lower bound on the fraction of satisfiable instances. SAT-E on these problems returned UNSAT for 12% and SAT for 4% of the instances (timing out on the remaining 84%). The 12% UNSAT result gives us an upper bound on the fraction of satisfiable instances of 88%. SAT-E was unable to solve any instance of size 100,000 before the timeout, so we only have the lower bound provided by KPR of 96% for the fraction of satisfiable instances.

First, note that KPR outperforms RP99 with respect to all measures presented in Tables 1 and 2. KPR solves many more instances, and always does so faster than RP99. KPR is by far the fastest method, returning a matching in under a second for small instances, and only a few seconds for large ones. SAT-E scales quite well for moderately-sized instances, solving all until market size of 50,000. Interestingly, it seems as though declaring unsatisfiability may be easier than finding a satisfying assignment, as most of the solvable instances with 50,000 residents were UNSAT. SAT-E outperforms IP-E on all measures; while IP-E solved the same number of small instances as SAT-E, it required more time (and significantly more memory) to do so, and also scaled poorly.

Most importantly, note that there are instances for which KPR does not find a solution even though one exists; with couples consisting of roughly 15% of the market, KPR has a failure rate of 1.55%. Thus, even though KPR is very fast, its incompleteness leads to it being unable to solve certain instances that are easy for SAT-E.

One concern with SAT-E is that since KPR and RP99 are resident proposing algorithms the resulting match might be better suited for the resident. Good outcomes for residents is stated an important desiderata by Roth and Peranson [1999] and no resident bias is designed into SAT-E. We analyse the matchings where both KPR and SAT-E found a stable matching. In 86.5% of these cases, the matchings that KPR and SAT-E found were identical. For the remaining instances where KPR and SAT-E found a different stable matching, very few of the residents were assigned to a different program; an average of 0.29% of all single residents and 1.07% of all couples were affected. On average, KPR tended to find better matches for the residents; when residents could improve, single residents tended to improve by an average of 2.58 positions on their ROL (of length 11), and couples tended to improve 4.09 positions (out of a joint ROL of expected length 41). However, there are instances where KPR significantly outperforms SAT-E, and where SAT-E significantly outperforms KPR. In one case, KPR finds a matching where some single resident is 9 positions better than SAT-E's matching; in another, SAT-E finds a matching where some single resident is 9 positions better than KPR's matching. Likewise with couples, for some instance KPR finds a matching where a couple improves 19 positions, and there is an instance where SAT-E improves a couple's matching by 14 positions. Thus, while KPR tends to provide more resident-optimal stable matchings than the unguided SAT-E, it certainly does not always find a matching that's better for residents than SAT-E.

# programs/ # singles	# couples	% satisfiable instances	SAT-E	IP-E	KPR	RP99
250	20	98	1.00	1.00	* 0.98	0.88
500	50	92	1.00	1.00	0.90	0.70
1,000	100	90	1.00	1.00	0.88	0.74
2,000	250	98	1.00	1.00	0.96	0.60
5,000	500	90	1.00	1.00	0.88	0.62
10,000	1,000	88	1.00	1.00	* 0.88	0.68
20,000	2,000	92	1.00	1.00	0.90	0.62
50,000	5,000	86 – 88	0.16	*TO*	0.86	0.68
100,000	10,000	$\geq$ 96	*TO*	*TO*	0.96	0.72

Table 1: Fraction of one-to-one instances solved by each solving method. As KPR and RP99 are sound but not complete, * denotes when all possible solutions were found. *TO* denotes all instances timed out.

# programs /singles	# couples	SAT-E	IP-E	KPR	RP99
250	20	1.414	1.871	0.001	0.004
500	50	3.046	4.788	0.003	0.0241
1,000	100	6.700	10.795	0.009	0.204
2,000	250	17.779	36.394	0.027	1.616
5,000	500	80.927	113.683	0.104	10.965
10,000	1,000	262.539	315.408	0.341	165.475
20,000	2,000	945.721	1202.060	1.005	780.384
50,000	5,000	1,844.627	*TO*	2.485	592.780
100,000	10,000	*TO*	*TO*	6.919	2,500.593

Table 2: Average runtime of one-to-one instances, in seconds. Timeouts excluded. *TO* denotes all instances timed out.

We also investigate how the various solution techniques compare as the density of couples in the match grows. In Figure 1 we examine the performance of KPR, compared to the true fraction of satisfiable instances. All instances were one-to-one matching problems, drawn with 24,000 residents and 20,000 programs and varying percentage of residents that were in couples, from 16.67% to 50.00%. SAT-E returned either SAT or UNSAT for all instances allowing us to know the satisfiability of each instance; KPR never solved all satisfiable instances. The fraction of satisfiable instances was above 80% until the percentage of residents in couples exceeded 37%. Even with half of the residents in couples, 56% were satisfiable. However, KPR did not find a solution for any of these.

We next investigate the solvers' performance on many-to-one stable matching problems with couples; many real-world instances of SMP-C are many-to-one (e.g., NRMP). For our experiments, each program uniformly draws a quota of 5–9 available slots (resulting in an expected 7 slots per program). We use the same number of singles and couples as is Table 1, and $n/7$ programs, where n is the number of singles for that instance (giving us one slot per single in expectation as in our one-to-one experiments). Residents rank their top 10 programs. Results are presented in Tables 3 and 4. Due to the poor performance of RP99 on one-to-one problems, it

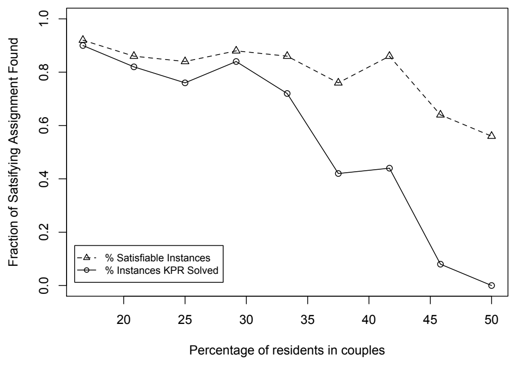

Figure 1: KPR performance as density of couples increases, one-to-one instances. SAT-E solved all instances, giving known percentage of satisfiable instances.

# singles	# couples	# programs	% satisfiable	SAT-E	IP-E	KPR
250	20	35	96	1.00	0.96	0.94
500	50	71	94	1.00	0.08	0.84
1,000	100	142	96	1.00	*TO*	0.84
2,000	250	285	98	1.00	*TO*	0.82
5,000	500	714	96	1.00	*TO*	0.74
10,000	1,000	1,428	≥ 76	*TO*	†	0.76
20,000	2,000	2,857	≥ 70	*TO*	†	0.70

Table 3: Percentage of many-to-one instances solved by each method. As KPR and RP99 are sound but not complete, * denotes when all possible solutions were found. *TO* denotes all instances timed out. †denotes instances were not run due to poor performance on smaller instances.

was not included in this analysis. First, note that these instances seem much more difficult than their corresponding one-to-one instances listed in Table 1. SAT-E times out on all of size 10,000 and higher (we only ran up to 20,000). IP-E performed very poorly, being unable to find all solutions within the 90 minute timeout even for the smallest problem size (250). Furthermore, KPR never successfully finds a solution for all satisfiable instances. KPR had an average failure rate of 12.89%, with a maximum failure rate of 22.91%; furthermore, this failure rate increased monotonically with the size of the problem. However, KPR was able to solve some of the problems that were too large for SAT-E, finding stable matches for 76% and 70% of the larger instances. While SAT-E solves most (before it times out) in a reasonable timeframe, taking at most roughly half an hour, KPR solves all problems in less than a second, except for the largest, which it solves in roughly 2 seconds.

The residents' outcome with respect to their preferred match has similar trends in the many-to-one case as in the one-to-one case. More matchings were different; 23.4% of the instances had different matchings between KPR and SAT-E. Again, in those matchings that differed, very few residents were affected; 0.68% of singles were affected, and 1.64% of couples were affected. KPR on average outperformed SAT-E: single residents had an average rank improvement of 6.57, and couples had an average rank improvement of 3.84. As before, there exists instances where KPR significantly outperforms SAT-E, and SAT-E significantly outperforms KPR.

Thus, as a summary of our results, IP-E is inefficient w.r.t. both space and time constraints, and never outperforms SAT-E. While KPR scales quite well, it misses solutions to a large

# singles	# couples	# programs	SAT-E	IP-E	KPR
250	20	35	13.27	1258.33	0.0021
500	50	71	38.24	2216.33	0.0085
1,000	100	142	111.18	*TO*	0.0182
2,000	250	285	398.99	*TO*	0.0585
5,000	500	714	1,917.30	*TO*	0.2038
10,000	1,000	1,428	*TO*	†	0.7388
20,000	2,000	2,857	*TO*	†	2.2915

Table 4: Average runtime of many-to-one instances, in seconds. Timeouts excluded. *TO* denotes all instances timed out. †denotes instances were not run due to poor performance on smaller instances

proportion of instances, particularly in the many-to-one case, but even in the one-to-one case. As the density of couples in the problem increases, the performance of KPR declines rapidly, quickly being unable to solve any instances. SAT-E is remarkably effective for small to medium sized markets, solving every instance with up to 50,000 residents in the one-to-one case (which is a similar size to the NRMP), and up to and including 5,000 residents in the many-to-one case. Furthermore, performance did not degrade as the density of couples in the match increased. There is mixed evidence for how desirable a match is for residents; SAT-E and KPR do not always find the same matching. Though KPR tends to find a better match than the unguided SAT-E, it doesn't always.

6 Conclusions and Future Work

In this paper we presented a new SAT encoding, SAT-E, for SMP-C and showed that it can solve SMP-C quite effectively. Incomplete DA algorithms can still be much faster and solve larger problems, but they also miss many problems that can be solved with SAT-E. We also found evidence that SAT is more effective for solving SMP-C than IP (although perhaps other IP models might perform better).

We believe that there are many future research opportunities for the further use of SAT technology in solving stable matching problems. For example, when no stable match exists it is possible to extract from the SAT solver a proof of unsatisfiability. Potentially, techniques could be developed for analyzing these proofs to find adjustments that would permit a stable match. Similarly, MAXSAT could be used to find matchings that optimize some form of social welfare. Finally, applications like NRMP might require further constraints and refinements to the problem formulation as the market develops (e.g., minimum diversity quotas for programs, even/odd conditions for program acceptance lists, other complementarities, etc.). When using a SAT encoding like SAT-E to solve the problem, incorporating these new constraints is greatly simplified.

Acknowledgements. We acknowledge the support of NSERC. Drummond and Perrault were additionally supported by OGS. We thank the reviewers for helpful suggestions.

References

[Abdulkadiroglu *et al.*, 2005] Atila Abdulkadiroglu, P.A. Pathak, Alvin E. Roth, and Tayfun Sönmez. The Boston public school match. *American Economic Review*, 95(2):368–371, 2005.

[Ashlagi *et al.*, 2014] I. Ashlagi, M. Braverman, and A. Hassidim. Stability in large matching markets with complementarities. *Operations Research*, 62(4):713–732, 2014.

[Biere, 2013] Armin Biere. Lingeling, plingeling and treengeling entering the SAT competition 2013. In *Proceedings of the SAT Competition 2013*, pages 51–52, 2013. `http://www.satcompetition.org/2013/proceedings.shtml`.

[Biró *et al.*, 2014] Péter Biró, David F Manlove, and Iain McBride. The hospitals/residents problem with couples: Complexity and integer programming models. In *Experimental Algorithms*, pages 10–21. Springer, 2014.

[Drummond and Boutilier, 2013] Joanna Drummond and Craig Boutilier. Elicitation and approximately stable matching with partial preferences. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 97–105, 2013.

[Frisch and Giannaros, 2010] A. Frisch and P. A. Giannaros. SAT encodings of the at-most-k constraint some old, some new, some fast, some slow. In *The 9th International Workshop on Constraint Modelling and Reformulation (ModRef)*. `http://www.it.uu.se/research/group/astra/ModRef10/programme.html`, 2010.

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

[Gent and Prosser, 2002] Ian P Gent and Patrick Prosser. SAT encodings of the stable marriage problem with ties and incomplete lists. In *Proceedings of Theory and Applications of Satisfiability Testing (SAT)*, pages 133–140, 2002.

[Gent *et al.*, 2001] Ian P Gent, Robert W Irving, David F Manlove, Patrick Prosser, and Barbara M Smith. A constraint programming approach to the stable marriage problem. In *Principles and Practice of Constraint Programming (CP)*, pages 225–239, 2001.

[Kojima *et al.*, 2013] F. Kojima, P. Pathak, and A. E. Roth. Matching with couples: Stability and incentives in large markets. *Quarterly Journal of Economics*, 128(4):1585–1632, 2013.

[Manlove *et al.*, 2007] David Manlove, Gregg O'Malley, Patrick Prosser, and Chris Unsworth. A constraint programming approach to the hospitals/residents problem. In *Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, pages 155–170, 2007.

[Marx and Schlotter, 2011] Dániel Marx and Ildikó Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discrete Optimization*, 8(1):25–40, 2011.

[Niederle *et al.*, 2008] Muriel Niederle, Alvin E. Roth, and Tayfun Sonmez. Matching and market design. In Steven N. Durlauf and Lawrence E. Blume, editors, *The New Palgrave Dictionary of Economics (2nd Ed.)*, volume 5, pages 436–445. Palgrave Macmillan, Cambridge, 2008.

[NRMP, 2013] NRMP. National resident matching program, results and data: 2013 main residency match, 2013.

[Prosser, 2014] Patrick Prosser. Stable roommates and constraint programming. In *Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, pages 15–28, 2014.

[Ronn, 1990] Eytan Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11(2):285–304, 1990.

[Roth and Peranson, 1999] A. E. Roth and E. Peranson. The redesign of the matching market for American physicians: Some engineering aspects of economic design. *The American Economic Review*, 89(1):748–780, September 1999.

[Roth *et al.*, 1993] Alvin E Roth, Uriel G Rothblum, and John H Vande Vate. Stable matchings, optimal assignments, and linear programming. *Mathematics of Operations Research*, 18(4):803–828, 1993.

[Roth, 1984] Alvin E. Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.

[Unsworth and Prosser, 2005] Chris Unsworth and Patrick Prosser. A specialised binary constraint for the stable marriage problem. In *Abstraction, Reformulation and Approximation (SARA)*, pages 218–233, 2005.