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

Joanna Drummond and Andrew Perrault and Fahlam 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; Abdulkadirouglu 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 IP encoding for SMP [Roth et al., 1993]. We solve these encodings with the SAT solver Lingeling [Biere, 2013] and IBM’s IP solver CPLEX\textsuperscript{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 per-
centage 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 $\mu$ 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 $\mu$ when $\mu(d) = p$, and that $p$ is matched to $d$ when $d \in \mu^{-1}(p)$.

$S \subseteq D$ 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 over its acceptable options: $(x \succeq_a nil \land y \succeq_a nil) \rightarrow (x \succeq_a y \lor y \succeq_a x)$. Preferences between unacceptable matches need not be specified. We also standardize 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 $Ch_p(\cdot)$ 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 $ranked(x)$ to find the index of match of $x$ in a’s rol: $ranked(x) = i$ if $x$ 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 $rol_a[ranked(x)] = x$.

Solving an SMP-C instance involves finding a stable matching. Intuitively, a stable matching is a matching $\mu$ 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 $\mu$.

A pair with mutual incentive to defect from $\mu$ 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 $\mu$. 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 $\mu$ 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 $\mu$ 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 $\mu$ for $(D, C, P, \succeq)$, 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 $\mu$ respects all quotas and does contain any unacceptable matches.
1a: $\forall p \in P. |\mu^{-1}(p)| \leq cap_p$
1b: $\forall d \in S. (µ(d) \in rol_a)$
1c: $\forall (d_1, d_2) \in C. (µ(d_1), µ(d_2)) \in rol_{(d_1, d_2)}$
1d: $\forall p \in P. \forall d \in µ^{-1}(p). d \in rol_p$

To express the stability conditions we introduce the abbreviation $willAccept(p, R, µ) \equiv R \subseteq Ch_p(µ^{-1}(p) \cup R)$, 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, $µ^{-1}(p)$, it would choose a match that includes $R$. Note that if $p$ is already matched to the doctors in $R$, i.e., $R \subseteq µ^{-1}(p)$ then $willAccept(p, R, µ)$ will be true (as long as $µ^{-1}(p)$ 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 $p \succ_d µ(d)$ and a couple $c = (d_1, d_2)$ will accept $(p_1, p_2)$ over its current match if $(p_1, p_2) \succ_c (µ(d_1), µ(d_2))$.

Stability for Singles: No individual single doctor can find a
better matching than μ.
2: ∀d ∈ S. 3p ∈ P. p ∨ d μ(d) ∧ willAccept(p, {d}, μ)

Stability for Couples: No couple can find a better matching.
3a: ∀(d1, d2) ∈ C. 3(p1, p2) ∈ (P×P×P).
   p1 ≠ p2 ∧ (p1, p2) ∨ (d1, d2) μ(d1) μ(d2))
   ∧ willAccept(p1, {d1}, μ) ∧ willAccept(p2, {d2}, μ)
3b: ∀(d1, d2) ∈ C. 3p ∈ P×p × (p, p) ∨ (d1, d2) μ(d1) μ(d2))
   ∧ willAccept(p, {d1}, μ)

For case (3a) it can be that p1 = μ(d1) or p2 = μ(d2)
(i.e., only one member of the couple is switching programs)
but not both since (p1, p2) ∨ (d1, d2) μ(d1) μ(d2)) (strict preference).
If p1 = μ(d1) then, as explained above,
willAccept(p1, {d1}, μ(d1)) will automatically be true, similarly
when p2 = μ(d2). 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
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 al-
gorithm” [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 condi-
tions (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.

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

4 Encodings to NP-Complete Problems

Recent years have seen a tremendous increase in our abili-
ty to solve instances of NP-complete problems coming from
real world applications. In particular, solvers for Integer Pro-
grams (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 be-

4.1 SAT-E

We assume that the doctor (singles and couples) roles have
been preprocessed so as to remove from them any program
that does not find that doctor acceptable. For d ∈ S we re-
move p from rol,d if d /∈ rol,p. For couple (d1, d2) ∈ C we remove
(p1, p2) from rol,d1 if d1 /∈ rol,p1 or d2 /∈ rol,p2. 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] | d ∈ D ∧ p ∈ rol,d}. m_d[p]
is true if d is matched into program p. Note that
m_d[null] is true if d is unmatched.
2. Couple Matching Variables: {m_c[i] | c ∈ C ∧ (0 ≤ i < |rol,c|)}. m_c[i]
is true if couple c is matched into
a program pair \((p, p')\) that it ranks between 0 and \(i\), i.e., 
\[0 \leq \text{rank}(i, (p, p')) \leq i.\] (Lower ranks are more preferred.)

3. **Program Matching Variables:** \(\{m_{p}[i, s] \mid p \in P \land (0 \leq i \leq |\text{rol}_{p}| - 2) \land (0 \leq s \leq \text{min}(i + 1, \text{cap}_{p} + 1))\}\). \(m_{p}[i, s]\) is true iff \(s\) of the doctors in \(|\text{rol}_{p}| - 2\) which is the index the last non-nil doctors on \(\text{rol}_{p}\) (\(\text{rol}_{p}\) 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. \(\text{at-most-one}(\{m_{d}[p] \mid p \in \text{ranked}(d)\})\)

1b. \(\bigvee_{p \in \text{ranked}(d)} m_{d}[p]\)

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

**Couple Match.** The \(m_{c}[s]\) variables must have their intended meaning. For all couples \(c \subseteq C\), for all \(k\) such that \(1 \leq k \leq |\text{rol}_{c}|\), letting \(c = (d_{1}, d_{2})\) and \((p_{1}[i], p_{2}[i]) = \text{rol}_{c}[i]\),

2a. \(m_{c}[0] = m_{d_{1}}[p_{1}[0]] \land m_{d_{2}}[p_{2}[0]]\)

2b. \(m_{c}[k] = (m_{d_{1}}[p_{1}[k]] \land m_{d_{2}}[p_{2}[k]]) \lor m_{c}[k - 1]\)

2c. \(m_{c}[|\text{rol}_{c}|]\)

The final condition ensures that \(c\) is matched to some program pair on its \(\text{rol}\) (possibly nil), and the at-most-one constraint for \(d_{1}\) and \(d_{2}\) ensures that \(c\) is uniquely matched.

**Program Match.** The \(m_{p}[s, s]\) variables must have their intended meaning. For all programs \(p \in P\), for all \(i\) such that \(1 \leq i \leq |\text{rol}_{p}| - 2\), and for all \(s\) such that \(0 \leq s \leq \text{min}(i + 1, \text{cap}_{p} + 1)\), letting \(d_{i} = \text{rol}_{p}[i]\) (the \(i\)-th doctor on \(p\)’s \(\text{rol}\)).

3a. \(m_{p}[0, 0] = \neg m_{d_{1}}[p] \land (m_{p}[0, 1] = m_{d_{0}}[p])\)

3b. \(m_{p}[i, s] = (m_{p}[i - 1, s] \land \neg m_{d_{i}}[p]) \lor (m_{p}[i - 1, s - 1] \land m_{d_{i}}[p])\)

3c. \(\neg m_{p}[i, \text{cap}_{p} + 1]\)

The last condition, captured by a set 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 \(\text{rol}_{p}\) since nil does not use up any program capacity.

**Stability for Singles:** For each single doctor, \(d \in S\) and for each \(p \in \text{rol}_{d}\)

4. \((\bigvee_{p \leq \text{cap}_{p}} m_{d}[p]) \lor m_{p}[\text{rank}_{p}(d) - 1, \text{cap}_{p}]\)

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 \(m_{p}[\text{rank}_{p}(d) - 1, \text{cap}_{p}]\) means that \(p\) has been filled to capacity with doctors coming before \(d\) on its \(\text{rol}\).

**Stability for Couples (A):** For each couple \(c = (d_{1}, d_{2}) \in C\) and for each \((p_{1}, p_{2}) \in \text{rol}_{c}\) with \(p_{1} \neq p_{2}\)

5a1. \(m_{d_{1}}[p_{1}] \land \neg m_{c}[\text{rank}_{c}((p_{1}, p_{2}))\]

\(\rightarrow m_{p_{1}}[\text{rank}_{p_{1}}(d_{1}) - 1, \text{cap}_{p_{1}}]\)

5a2. \(m_{d_{2}}[p_{2}] \land \neg m_{c}[\text{rank}_{c}((p_{1}, p_{2}))\]

\(\rightarrow m_{p_{2}}[\text{rank}_{p_{2}}(d_{2}) - 1, \text{cap}_{p_{2}}]\)

5b. \(\neg m_{d_{1}}[p_{1}] \land \neg m_{d_{2}}[p_{2}] \land \neg m_{c}[\text{rank}_{c}((p_{1}, p_{2}))\]

\(\rightarrow m_{p_{1}}[\text{rank}_{p_{1}}(d_{1}) - 1, \text{cap}_{p_{1}}]\)

\(\lor m_{p_{2}}[\text{rank}_{p_{2}}(d_{2}) - 1, \text{cap}_{p_{2}}]\)

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 \((p, p_{r}) \in \text{rol}_{c}\) 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] \land \neg m_{c}[\text{rank}_{c}((p, p_{r}))\]

\(\rightarrow m_{p_{r}}[\text{rank}_{p_{r}}(d_{2}) - 1, \text{cap}_{p_{r}}]\)

6b1. \(m_{d_{2}}[p] \land \neg m_{c}[\text{rank}_{c}((p, p_{r}))\]

\(\rightarrow m_{p_{r}}[\text{rank}_{p_{r}}(d_{1}) - 1, \text{cap}_{p_{r}} - 1]\)

\(\lor m_{p_{r}}[\text{rank}_{p_{r}}(d_{2}) - 1, \text{cap}_{p_{r}} - 1]\)

Clauses 6a1 or 6b1 say that if \(d_{1}\) is already in \(p\) and \(c\) is not matched to \((p, p_{r})\) or into a more preferred program pair, then \(p\) will not accept \(d_{1}\). 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 \((p, p_{r})\) if accepting \(d_{2}\) causes \(d_{1}\) to be bumped. That is, when \(m_{p_{r}}[\text{rank}_{p_{r}}(d_{1}) - 1, \text{cap}_{p_{r}} - 1]\) is true (adding \(d_{2}\) will cause \(m_{p_{r}}[\text{rank}_{p_{r}}(d_{1}) - 1, \text{cap}_{p_{r}}\) 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 \(\mathcal{P} = (D, C, P)\) be a matching problem. We say that a matching \(\mu\) for \(\mathcal{P}\) and a truth assignment \(\pi\) for SAT-E of \(\mathcal{P}\) are corresponding when \(\pi \models m_{d}[p]\) if \(\mu(d) = p\). The following theorem shows that the satisfying assignments of SAT-E are in a 1-1 relationship with the stable models.

**Theorem 1** If \(\mu\) and \(\pi\) are corresponding, then \(\mu\) is stable if and only if \(\pi\) 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 \(\pi\) corresponding to a stable matching \(\mu\) satisfies SAT-E is straightforward: the individual rationality
conditions $\mu$ ensure that clause sets 1–3 are satisfied, and the stability conditions ensure that clause sets 4–6 are satisfied.

Suppose $\pi$ is a satisfying assignment. It is similarly straightforward to see that the fact it satisfies clause sets 1–3 implies that its corresponding matching $\mu$ 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 $\mu$ 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 $p \leq \mu(p)$. For $p > \mu(d)$ the clauses of 4 are reduced to the units $m_p[\text{rank}_p(d)] - 1, \alpha_d, \mu(d)$). These units along with the clauses of 3 ensure that all programs $p$ preferred by $d$ and $\mu$ are already filled with doctors $p$ prefers to $d$. Hence none of these more preferred programs will accept $d$ and $\mu$ 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 $d_1 > p d_2$. If $p$ was to accept $d_2$, then $m_p[\text{rank}_p(d_2)] - 1, \alpha_d$ 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 $m_p[\text{rank}_p(d_2)] - 1, \alpha_d$ must also be false. These two imply that both $m_p[\text{rank}_p(d_1)] - 1, \alpha_d$ and $m_p[\text{rank}_p(d_1)] - 1, \alpha_d$ are false, since the number accepted before $d_1$ is 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 $\mu$ 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., 1995]. 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:** $\{m_c[(p_1, p_2)] | c \in C \land (p_1, p_2) \in \text{rol}_c \}$. $m_c[(p_1, p_2)] = 1$ iff $m_c$ is matched to $(p_1, p_2)$.
2. **Doctor Matching Variables:** $\{m_d[p] | d \in D \land p \in \text{rol}_d \}$. $m_d[p] = 1$ iff $m_d$ is matched to $p$. If $d$ is in a couple $c = (d_1, d_2)$, $m_d[p]$ is an abbreviation for $\sum p'[p'] \in \text{rol}_d, m_c[(p_1, p_2)]$.
3. **Auxiliary Variables:** $\{\alpha_{d_2, p_2} | c = (d_1, d_2) \in C \land p_2 \in \text{ranked}(d_2) \}$. $\alpha_{d_2, p_2}$ 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 $d \in D$

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

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

2. $\sum_{d \in \text{ranked}(p)} m_d[p] \leq \text{cap}_p$

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

3. $\sum_{d' \geq p_2} m_{d'}[p_2] \geq \text{cap}_p, \alpha_{d_2, p_2}$

**Stability for Singles:** For each single doctor, $d \in S$ and for each $p \in \text{rol}_d$

4. $\sum_{d' \geq d} \text{cap}_p \sum_{d'' \geq d} m_{d'}[p] \geq \text{cap}_p$

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 $(p_1, p_2) \in \text{rol}_c$ with $p_1 \neq p_2$

5a. $\text{cap}_p(1 - m_{d_1}[p_1]) + \text{cap}_p \sum_{(p_1', p_2') \geq (p_1, p_2)} m_{c'}[(p_1', p_2')] \geq \text{cap}_p$

5b. $\text{cap}_p m_{d_1}[p_1] + \text{cap}_p m_{d_2}[p_2] + \text{cap}_p \sum_{(p_1', p_2') \geq (p_1, p_2)} m_{c'}[(p_1', p_2')] \geq \text{cap}_p$

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 $d_1 > p d_2$. For each couple $c = (d_1, d_2) \in C$ and for each $(p, p) \in \text{rol}_c$

6a. $\text{cap}_p (1 - m_{d_1}[p]) + \text{cap}_p \sum_{(p_1', p_2') \geq (p, p)} m_{c'}[(p_1', p_2')] \geq \text{cap}_p$

6b. $\text{cap}_p m_{d_1}[p] + \text{cap}_p m_{d_2}[p] + \text{cap}_p \sum_{(p_1', p_2') \geq (p, p)} m_{c'}[(p_1', p_2')] \geq \text{cap}_p - 1$

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™ 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.

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

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.

<table>
<thead>
<tr>
<th># programs/singles</th>
<th># couples</th>
<th>% satisfiable instances</th>
<th>SAT-E</th>
<th>IP-E</th>
<th>KPR</th>
<th>RP99</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>20</td>
<td>98</td>
<td>1.00</td>
<td>1.00</td>
<td>* 1.00</td>
<td>0.98</td>
</tr>
<tr>
<td>500</td>
<td>50</td>
<td>92</td>
<td>1.00</td>
<td>1.00</td>
<td>0.90</td>
<td>0.70</td>
</tr>
<tr>
<td>1,000</td>
<td>100</td>
<td>90</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>0.74</td>
</tr>
<tr>
<td>2,000</td>
<td>250</td>
<td>98</td>
<td>1.00</td>
<td>1.00</td>
<td>0.96</td>
<td>0.60</td>
</tr>
<tr>
<td>5,000</td>
<td>500</td>
<td>90</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>0.62</td>
</tr>
<tr>
<td>10,000</td>
<td>1,000</td>
<td>88</td>
<td>1.00</td>
<td>0.88</td>
<td>0.68</td>
<td></td>
</tr>
<tr>
<td>20,000</td>
<td>2,000</td>
<td>92</td>
<td>1.00</td>
<td>1.00</td>
<td>0.90</td>
<td>0.62</td>
</tr>
<tr>
<td>50,000</td>
<td>5,000</td>
<td>86–88</td>
<td>0.16</td>
<td>TO</td>
<td>0.86</td>
<td>0.68</td>
</tr>
<tr>
<td>100,000</td>
<td>10,000</td>
<td>&gt; 96</td>
<td>TO</td>
<td>TO</td>
<td>0.96</td>
<td>0.72</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th># programs/singles</th>
<th># couples</th>
<th>SAT-E</th>
<th>IP-E</th>
<th>KPR</th>
<th>RP99</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>20</td>
<td>1.141</td>
<td>1.871</td>
<td>0.001</td>
<td>0.004</td>
</tr>
<tr>
<td>500</td>
<td>50</td>
<td>3.046</td>
<td>4.788</td>
<td>0.003</td>
<td>0.024</td>
</tr>
<tr>
<td>1,000</td>
<td>100</td>
<td>6.700</td>
<td>10.795</td>
<td>0.009</td>
<td>0.204</td>
</tr>
<tr>
<td>2,000</td>
<td>250</td>
<td>17.579</td>
<td>36.394</td>
<td>0.027</td>
<td>1.616</td>
</tr>
<tr>
<td>5,000</td>
<td>500</td>
<td>80.927</td>
<td>113.683</td>
<td>0.101</td>
<td>10.965</td>
</tr>
<tr>
<td>10,000</td>
<td>1,000</td>
<td>262.539</td>
<td>315.408</td>
<td>0.341</td>
<td>165.475</td>
</tr>
<tr>
<td>20,000</td>
<td>2,000</td>
<td>945.721</td>
<td>1,202.960</td>
<td>1.005</td>
<td>760.384</td>
</tr>
<tr>
<td>50,000</td>
<td>5,000</td>
<td>1,844.021</td>
<td>2,485</td>
<td>592.780</td>
<td></td>
</tr>
<tr>
<td>100,000</td>
<td>10,000</td>
<td>TO</td>
<td>TO</td>
<td>6.919</td>
<td>2,500.593</td>
</tr>
</tbody>
</table>
programs. Results are presented in Tables 3 and 4. Due to the poor performance of RP99 on one-to-one problems, it 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 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 increased.

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.

### 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.
References


