A Local Search Algorithm for SMTI and its extension to HRT Problems

Danny Munera[1], Daniel Diaz[1], Salvador Abreu[2], Francesca Rossi[3], Vijay Saraswat[4], and Philippe Codognet[5]

[1] University of Paris 1 / CRI
`Danny.Munera@malix.univ-paris1.fr, Daniel.Diaz@univ-paris1.fr`
[2] University of Évora / LISP / CRI
`spa@di.uevora.pt`
[3] University of Padova / Harvard University
`frossi@math.unipd.it`
[4] IBM TJ Watson Research Center
`vsaraswa@us.ibm.com`
[5] JFLI-CNRS / UPMC / University of Tokyo
`codognet@is.s.u-tokyo.ac.jp`

Abstract. Hospitals/Residents with Ties (HRT) forms a class of problems with many applications, some of which are of considerable size. Solving these problems has been shown to be NP-hard. In previous work, we developed a local search algorithm which displays very high performance in solving Stable Matching with Ties and Incomplete lists (SMTI) problems. In this paper, we propose a method to tackle HRT problems with a slightly modified version of our SMTI solver. We describe our method and provide an initial performance assessment, which turns out to show that the resulting solver can deal with significant HRT problems, providing optimal solutions in most cases, in a very short time.

1 Introduction

In 1962, Gale and Shapley introduced the Stable Matching (SM) problem [4]. An SM instance of size n involves a set of n men and a set of n women, each of whom has ranked all members of the other set in strict order of preference. Solving such a problem consists of finding a *matching*, i.e. a one-to-one matching between the men and the women. In addition the matching must be *stable*, meaning that there is no man-woman pair where both would rather marry each other than their current partner – such a pair is called a *blocking pair*. Gale and Shapley proved that such a stable matching always exists and proposed an $O(n^2)$ algorithm (called GS in what follows) to find one.

However, requiring *each* member to rank *all* members of the opposite sex in a *strict* order is too strong a restriction for many real-life, large-scale applications. A natural variant of SM is the *Stable Matching with Ties and Incomplete Lists* (SMTI) problem [14, 17]. In SMTI, the preference lists may include ties (to express indifference) and may be incomplete (to express that some partners are unacceptable). More formally, an SMTI instance of size n consists of n men and n women, and a preference list for each of them, which contains some of the people of the other gender. Such preference lists are weak orders, that is, total orders possibly containing ties. Given an SMTI instance, a *matching* M is a set of pairs (m, w) representing a (possibly partial) one-to-one matching of men and women. If a man m is not matched in M (i.e. for no w is it the case that $(m, w) \in M$), we say that m is *single* in M (similarly for women). The *size* of a matching M is the cardinality of M, denoted $|M|$.

With the introduction of ties in the preference lists, three different notions of stability may be used [10, 17, 14]. As we consider only *weak stability* (the most challenging), we simply call it *stability.* In the context of M, a pair (m, w) is a *Blocking Pair* (BP) iff (a) m and w accept each other and (b) m is either single in M or strictly prefers w to his current wife, and (c) w is either single in M or strictly prefers m to her current husband. A matching M is stable iff it has no blocking pairs.

A (weakly) stable matching always exists and can be found with variants of the GS algorithm. Since any given SMTI instance may have stable matchings of different sizes, a natural requirement is to find one of maximum cardinality. This optimization problem (called MAX-SMTI) has many real-life applications [17, 14] and has attracted a lot of research in recent years because of that: car sharing or bipartite market sharing, job markets and social networks. Many of these applications involve very large sets. Unfortunately, the MAX-SMTI problem has been shown to be NP-hard, even for very restricted cases (e.g. only men declare ties, ties are of length two, the whole list is a tie) [13, 17].

We have recently proposed a Local Search (LS) algorithm for the SMTI problem [19]. For this, an SMTI problem is first modeled as a permutation problem and then solved by the *Adaptive Search* (AS) method [1, 2]. Basically, starting from a random matching, our algorithm iteratively tries to improve the current matching by performing a swap between two variables (i.e. two men exchange their partner). For this, a limited neighbourhood is explored and the most promising swap is selected based on a heuristic which selects the most significant blocking pair to fix and/or a single man to marry. The algorithm stops when a *perfect matching* is found (a stable matching with no singles) or when a given timeout is reached (in which case the best matching found so far is returned). This algorithm turned out to have very high performance and is able to optimally solve several large instances.

Another very useful variant of SM is the *Hospitals / Residents problem with Ties* (HRT) [12, 11, 17]. An HRT instance consists of two sets: the residents $R = \{r_1, \dots r_{n_1}\}$ who apply to the hospitals $H = \{h_1, \dots h_{n_2}\}$. The preference list of a resident $r_i \in R$ consists of the ordered list of *acceptable* hospitals (a subset of H). The preference list of a hospital $h_j \in H$ contains the ordered list of residents (a subset of R) who consider h_j acceptable. All preference lists are allowed to contain ties. In addition, each hospital $h_j \in H$ has a capacity c_j indicating the maximum number of positions it offers.

The problem consists of finding a stable matching between residents and hospitals satisfying both the preference lists (the matching must be stable) and the capacities (each resident being assigned to at most one hospital and the number of residents assigned to any hospital h_j must not exceed c_j). At any stage during the matching process, a hospital h_j with a_j assignees is said to be *over-subscribed* if $a_j > c_j$, *full* if $a_j = c_j$, and *under-subscribed* if $a_j < c_j$.

The previously discussed notion of weak stability can be adapted to HRT: in the context of M, a pair (r, h) give rise to a blocking pair iff (a) r and h accept each other and (b) r is either unassigned in M or strictly prefers h to his assigned hospital, and (c) h is either under-subscribed or strictly prefers r to the worst resident assigned to it. As for SMTI, a matching M is stable iff it has no blocking pairs.

HRT problem has many practical applications, e.g. assignment of applicants to positions in job markets. In the medical employment domain, there are national programs in various countries such as the Scottish Foundation Allocation Scheme (SFAS), the Canadian Resident Matching Service (CARMS) or the National Resident Matching Program (NRMP) in the USA. Obviously, such programs involve very large sets. Unfortunately, as for SMTI, the problem of finding a stable matching of maximum cardinality (called MAX-HRT) for a given instance of HRT is NP-hard (even for restricted cases, e.g. if the ties are only allowed on one side). Finding an efficient algorithm to solve HRT problems is thus a true challenge with many real applications.

We deem it interesting to see if we can attack the HRT problem with our LS algorithm. While SMTI is a special case of HRT (where each hospital has capacity one) we chose to adopt a reverse approach, considering an HRT as a special case of SMTI so as to keep the main lines of our algorithm (permutation-based, which ensures a compact memory representation and an implicit modeling of the *all-different* constraint). To this end, we can use the so-called *cloning technique* [9] which basically consists of creating c_j copies of the hospital h_j, each of capacity 1, and to use these copies (inside a tie) each time this hospital is referenced in a resident's preference list. Strictly speaking, the resulting problem instance is not exactly an STMI instance since the resulting sets (residents and cloned hospitals) can have different sizes but it is trivial to add dummy elements (residents or hospitals). The extension of our algorithm to deal with this feature is very simple. All of this makes HRT and SMTI equivalent problems.

This *RISC*-like approach is analogous to what occurs with SAT modeling: the object formulation is often voluminous and cumbersome but its resolution by the best SAT solvers is very efficient – often faster than what is obtained with dedicated solvers which take higher-level formulations, such as CSP or constraint programming. Upon dealing with hard problem instances, we try to improve the solver at low level, meaning that the techniques which we may come up with to better solve HRT will also benefit SMTI, in the general case. We stress that our LS algorithm gets much better performance than complete methods (i.e. enumerative, branch and bound, linear and integer programming, etc.), even though we do not always reach the optimum solution.

As the rest of this paper will show, we manage to get very competitive performance on real-world data sets of considerable size and difficulty. We are also convinced that this will further benefit from the efficiency improvements we explored in [19], namely parallelism.

2 A Local Search Method for SMTI

Local search is a meta-heuristic method for solving optimization problems. It requires a cost function to evaluate the quality of a given assignment of variables (i.e. a *configuration*). The method also needs a transition function which defines, for each configuration, a set of neighbours. The simplest Local Search algorithm starts from a random configuration, explores the neighbourhood, selects a promising neighbour and moves to it. This iterative process continues until a solution is found. In this paper, we use a Local Search method developed by our team: the Adaptive Search method [1].

Adaptive Search (AS) is a generic, domain-independent, constraint-based local search method. This meta-heuristic takes advantage of the modeling of the problem in terms of constraints and variables, in order to guide the search more precisely than a single global cost function.

The error function in AS is a heuristic value which stands for the degree of satisfaction of the constraints. The method combines the error for each constraint to obtain a global cost and then, for each variable, AS projects constraint errors on the involved variables. AS repairs the *worst* variable (highest error) with the *best* (most promising) available value.

AS also includes an adaptive memory inspired by Tabu Search [8] in which each variable leading to a local minimum is marked and cannot be selected for the next few iterations. A local minimum is a configuration for which none of the neighbours improve the current cost. Finally, the algorithm also includes partial resets in order to escape stagnation around local minima.

For this work we use a particular implementation of AS, specialized for permutation problems. In this case all n variables have the same initial domain of size n and are subject to an implicit *all-different* constraint.

2.1 AS Model for SMTI Problems

We recently developed an efficient Adaptive Search model to solve SMTI problems (AS-SMTI). In this section we sketch the main features of the modeling; the interested reader may refer to [19] for details.

To use AS, we model the SMTI problem as a permutation problem: we define a sequence of n variables $(X_1 \dots X_n)$ which take for values permutations of the vector $(1 \dots n)$. $X_i = j$ is interpreted as either $(m_i, w_j) \in M$, or m_i is single if w_j is not on its preference list. Note that this interpretation remains valid when the values of any two variables are swapped (this is how value assignment is implemented in permutation problems).

The AS method seeks to improve the stability of a matching by removing blocking pairs (BPs). Some BPs may be useless in that fixing them does not improve things since the man involved remains part of another BP. To avoid this, the method focuses only on the so-called *undominated blocking pairs* [15, 5]. Let (m, w) and (m, w') be BPs. BP (m, w) dominates (from the men's point of view) BP (m, w') iff m prefers w to w'. A BP (m, w) is *undominated* iff there is no other BP dominating (m, w). In the following we only consider *undominated* BPs, which we simply call BPs.

Adaptive Search relies on a global objective function (called cost function) to measure the degree of error of a configuration. The cost function of a matching M is defined as follows:

$$cost(M) = \#BP(M) \times n + \#Singles(M)$$

where $\#BP(M)$ is the number of BPs in M, and $\#Singles(M)$ is the number of single men in M. The number of BPs is weighted with n to prioritize stable matchings over matchings with fewer singles. A matching M is stable iff $cost(M) < n$, and *perfect* iff $cost(M) = 0$. AS stops as soon as the cost function reaches 0 or when a given time limit is hit, in which case it returns the best matching found so far.

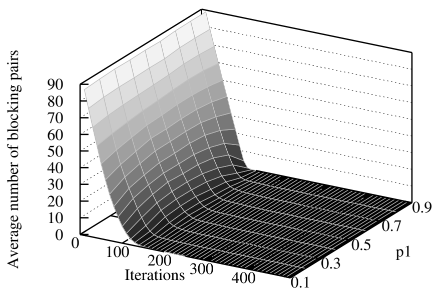
The AS-SMTI modeling defines the function $R(w, m)$ as the *rank* of m in the preference list of w, ranging over $1..(n+1)$, with $i < j$ implying w prefers (man with rank) i to (man with rank) j, and $R(w, m) = n + 1$ iff m is not in the preference list of w. The implementation does some straightforward pre-computation to avoid the linear cost of recomputing $R(w, m)$. The algorithm computes the BPs in the match M, going through all men in the problem. For each men m, let w the current partner of m such that $(m, w) \in M$, AS-SMTI verifies in the preference list of m if there is a woman w' with a higher level of preference than w. If w' exists, let m' the current partner of w' such that $(m', w') \in M$, the algorithm checks in the preference list of w' if the man m has a higher level of preference than m'. If this happens, the algorithm has found the BP (m, w'). The associated error for this BP is determined by the following expression: $R(w', m') - R(w', m)$. Thus, the further the assigned man is from the BP (in the preference list of w'), the larger the error.

It is worth noticing that while (undominated) BPs are considered from the men point of view, the associated errors are computed from the women point of view. Since preference lists can include ties, a man can be implied in several undominated BPs. For efficiency and simplicity reasons, AS-SMTI only considers the first encountered BP for a given man m and computes its error as explained above. Other strategies exist to aggregate the error associated to all BPs (select the maximum error, the average error, randomly one error, . . .).

At each iteration, AS selects the "worst" variable from the current matching M to improve it (the man involved in the BP with the largest error as explained above). In case of several man have the same highest cost, one is selected randomly. AS then fixes the culprit by swapping X_m and $X_{m'}$. In short, AS considers all BPs, chooses the variable corresponding to the worst one, fixes it by moving to a new configuration and re-evaluates the cost of the resulting matching. This heuristic avoids the cost of fixing all BPs, one by one.

In most cases, the resulting matching improves on the current one, and AS continues iteratively. When this is not the case, AS has reached a minimum (global or local). As AS has no way of knowing when the optimum has been reached (except when the cost is 0) it handles both cases similarly trying to escape the minimum invoking a *"reset"* procedure. This procedure slightly alters the current assignment of variables, trying to fix the 2 worst BPs and/or to assign a woman to a single man. The *reset* procedure is stochastic; it will *also* fix the second worst variable with a probability p: good results are obtained with a high value, e.g. $p \simeq 0.98$. This procedure turns out to be very effective: while preserving most of the configuration (no more than 2 swaps are performed), it enables AS to escape all local minima and reach very good solutions.

AS stops when one of the following conditions is reached: (a) a perfect solution has been found (i.e. cost = 0), (b) a given *target cost* T is reached (it is possible to ask the solver to find a solution with at most T singles) or (c) a timeout is reached (in which case the best solution found so far is returned).



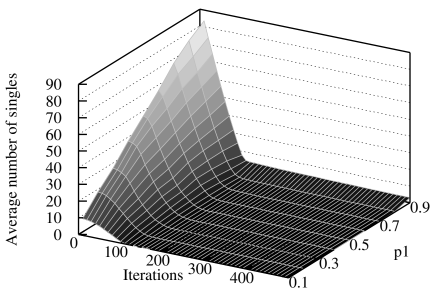


Fig. 1: a) average number of BP b) average number of singles when solving SMTI problems ($n = 100$, $p2 = 0.5$, varying $p1$)

2.2 Performance evaluation on SMTI problems

We present here an evaluation of our AS-SMTI algorithm. For this, we used a dataset composed of random problems created using the generator described in [6] which takes three parameters: the size (n), the probability of incompleteness ($p1$) and the probability of ties ($p2$). Given a triple, $(n, p1, p2)$, a SMTI problem instance with n men and n women is generated as follows: for each man and woman, the algorithm generates a random permutation of size n, as a preference list. Then, the algorithm iterates over each object in the preference lists, and with a probability $p1$, this object is *deleted* from the preference list. Finally, the algorithm iterates again over each remaining object (in the men and women preference lists) and with a probability $p2$, a *tie* is created between the current object and the previous one.

For a given combination of $p1$ and $p2$ 100 different problems were generated. Since AS is a stochastic procedure, each problem is solved 50 times and results are averaged. We used an X10 implementation of AS running sequentially on an AMD Opteron 6380 clocked at 2.5 GHz, i.e. using only one core.

The first experiment analyzes the number of iterations needed to solve an SMTI problem. Every 10 iterations, the solver reports the number of BPs and the number of singles of the current configuration. Due to space limitation, we here consider problems of size 100, for different values of $p1$ in $[0.1, 0.9]$ and for $p2 = 0.5$ (ties may appear in both sides). Figure 1 presents the averaged results of this experimentation. It appears that the average number of BPs quickly decreases. For instance, on average, after 200 iterations, a stable matching is already reached in 99, 88% of the cases. It is worth noticing that some difficult instances can require more iterations (the maximum observed has been 460 iterations for a problem generated with $p1 = 0.5$). Figure 1 b) shows the evolution of the number of singles with respect to the number of iterations. Again, this number quickly decreases. It is worth noticing that when the incompleteness of the problem is high (e.g. when $p1 = 0.9$), some problems do not have a perfect solution and the number of singles does not fall below some boundary value.

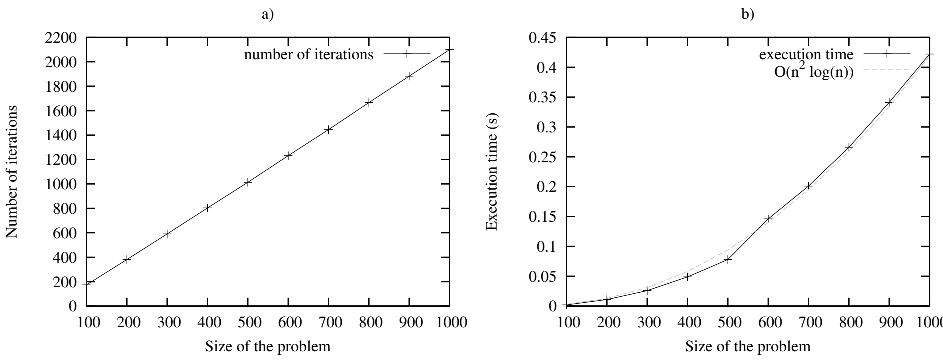
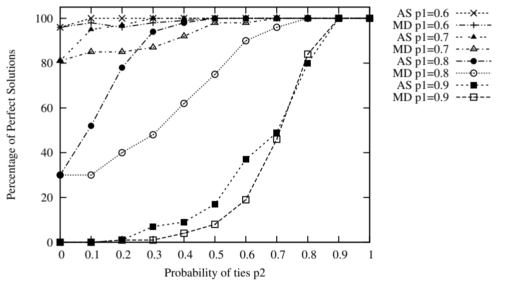


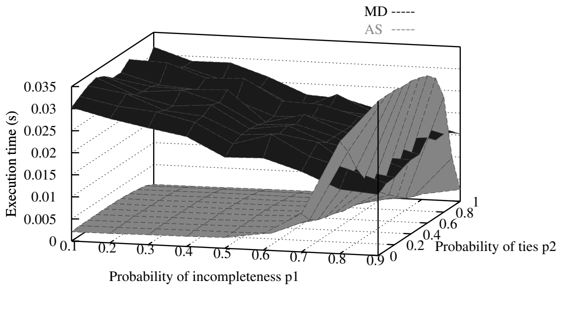
Fig. 2: Runtime behaviour of AS-SMTI: a) number of iterations, b) execution time (varying the size of the problem, p1=0.5 p2=0.5)

The second experiment analyzes the scalability of the AS-SMTI algorithm. For this we fixed the parameters $p1$ and $p2$ to 0.5, and we varied the size n of the problem in the range of $[100, 1000]$ using steps of 100. Figure 2 shows the curves corresponding to the number of iterations and to the runtime when varying n. We can observe that the number of iterations to obtain a perfect solution for SMTI problems varies linearly from 200 iterations for $n = 100$ to 2000 iterations for $n = 1000$, with a corresponding runtime in $O(n^2 \log n)$. Obviously, the results of this test cannot be generalized and we plan to extend the experimentation with other values for $p1$ and $p2$.

Finally, we compared our AS solver with McDermid's method (MD) [18], a very efficient 3/2-approximation algorithm, as implemented in [20]. For this test we used a data set composed of SMTI problems of size $n = 100$, with $p1$ ranging over $[0.1, 0.9]$ and $p2$ over $[0, 1]$, with step 0.1. With MD, for each $(p1, p2)$ pair, we solved the 100 instances once and averaged the the execution time.



(a)



(b)

Fig. 3: AS vs. MD: a) quality of solutions b) execution time.

Figure 3a compares the quality of solutions. The percentage of perfect stable matchings found by the AS algorithm is considerably higher than those found by MD, in particular using a probability of ties $p2 \in [0.1..0.8]$.

Figure 3b compares the execution times, as a 3D chart. In many cases, AS is up to an order of magnitude faster than MD. With higher probability of incompleteness (e.g. $p1 = 0.9$), MD outperforms AS. This can be explained by the time-complexity of MD which is proportional to the total length of the preference lists, i.e. it linearly decreases as $p1$ increases. The MD algorithm seems to perform faster than our AS approach only when $p1 = 0.9$. We note that MD always returns the same, single and (sub)optimal solution, while AS will yield more than one solution, with observably better quality. Moreover, a *solution quality* vs. *performance* trade-off is always possible in AS, by tweaking the timeout parameter. A complete comparison of the AS-SMTI algorithm with a state-of-the-art Local Search method [5] and a SAT encoding of the SMTI problem [7] may be found in [19].

3 Solving HRT Problems

In this section, we propose an algorithm to solve the HRT problem based on the algorithm presented above (we call this extension AS-HRT). Our goal is to obtain an HRT solver with *minimum changes* in the AS-SMTI algorithm. For this purpose, AS-HRT resorts to the "cloning" technique described in [3, 9, 21]. The main idea in cloning an HRT problem is to define a match between residents and *positions* instead of hospitals. A position being a single post offered by a hospital (a hospital h_j can offer c_j positions), each position can only be assigned to only one resident (capacity equal to one). To convert an HRT problem into an SMTI formulation, we create a new set of positions composed of the single positions offered by the hospitals. Each position has the same preference list as its "root" hospital. In the residents' preference lists, each hospital h_j is replaced by a sequence composed of the associated c_j positions (all forming a tie). The resulting equivalent SMTI problem consists of matching residents and positions. The main drawback of the cloning process is a significant increase in the size of the problem (but this remains manageable with modern computers).

Using cloning we may convert an HRT problem into an (asymmetric) SMTI problem, in a polynomial time. We use the term "asymmetric" because the resulting SMTI problem can have sets of different sizes. More formally, an *asymmetric SMTI* problem is specified by (a) two sets M and W of cardinality m and n respectively, (b) a ranking function $R : M \times W \rightarrow \{1 \ldots n+1\}$ and $R : W \times M \rightarrow \{1 \ldots m+1\}$ (we use the same name R for the ranking function for men and women). Note that the only point of generality over the standard SMTI formulation is that m is not required to be identical to n. From the AS-SMTI algorithm this comes down handling a vector of $max(m, n)$ values with some "dummy" values for missing elements.

It is interesting to characterize the c_j clones in the resulting SMTI problem since they play the same role (they are interchangeable). Given an SMTI instance, $m_1, m_2 \in M$ are said to be *equivalent* (written $m_1 \sim m_2$) if $\forall w, R(m_1, w) = R(m_2, w)$. Similarly, $w_1, w_2 \in W$ are said to be equivalent if $\forall m, R(w_1, m) = R(w_2, m)$. Note that $\sim$ is an equivalence relation.

A given SMTI problem may have equivalent elements or not. When we translate an HRT problem into SMTI we get equivalences: the c_j elements corresponding to a hospital h_j are all equivalent to each other.

When walking through several matchings, it would be nice to avoid exploring equivalent matchings (i.e. those whose difference only concerns equivalent elements). Indeed, only one matching from an equivalence class needs to be examined. It is worth observing that given a matching M if (m, w) and $(m', w') \in M$ give rise to a blocking pair (m, w'), then $m \not\sim m'$ and $w \not\sim w'$. Therefore, the swap executed by the main loop of the AS-SMTI algorithm to fix a BP already ensures that a matching is not changed to an equivalent matching. However, it is not the case of the reset procedure invoked to escape a local minimum which performs some random swaps. We have not yet improved this point (if a reset procedure swaps two equivalent elements the local minimum is not escaped and another reset will occur resulting in a waste of time). To be fully aware of equivalences in the current AS-SMTI algorithm, we should first compute equivalences upfront (this is an easy linear operation) and avoid swaps between equivalent elements in the reset procedure. We plan to do this in a second version of the algorithm.

3.1 Implementation

To implement the extension to HRT problems, we developed a pre-processor and a post-processor (see Figure 4). The pre-processor converts an HRT problem instance I into an equivalent SMTI problem instance I'. The post-processor systems takes the match found by the AS-SMTI solver for I' and converts back into HRT match form. The resulting HRT match is the solution to the initial problem.

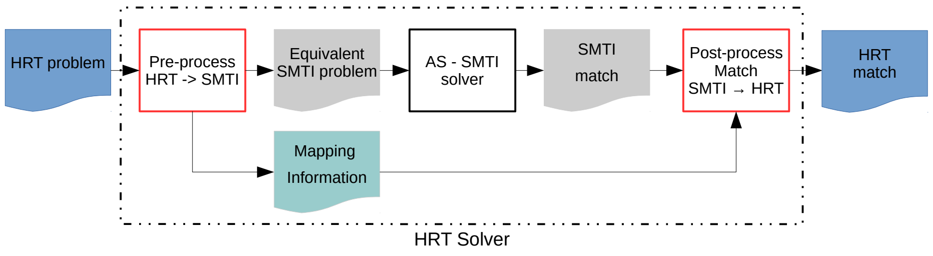


Fig. 4: Description of the solver extension to solve HRT problems

3.2 Preliminary Performance Evaluation

We did preliminary experiments to assess the performance of the HRT extension to AS-SMTI. We used a data set composed by randomly generated problems,[6] with the the same parameters as in [16]:

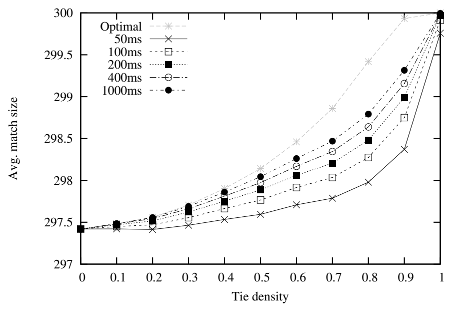
– Number of residents $n1 = 300$ (size of the problem).

[6] The data set was kindly provided by Augustine Kwanashie and David F. Manlove from the University of Glasgow, UK, and is the same as that in [16].

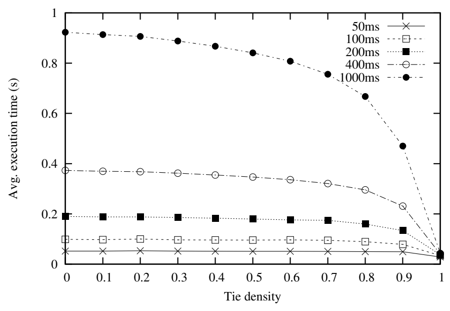
– Number of hospitals $n2 = 21$.
– Length of the residents' preference list $l = 5$.
– Total number of positions $C = 300$.
– Tie density td ranging over $[0, 1]$ with step 0.1.

For each td value, we used 10000 instances. The experiment was carried out on a machine with 4×16-core AMD Opteron 6380 CPUs, running at 2.5 GHz and 128 GB of RAM, using only 1 core. We solve each problem instance once, collecting the maximum size of the match and the execution time. We tested the timeout for AS-SMTI solver at $50, 100, 200, 400$ and $1000ms$.

Figure 5a shows the maximum size of the match found by AS-HRT, varying the tie density and using different timeout limits. We also include the optimal match size found with the Integer Programming method developed in [16]. AS-HRT almost reaches the optimal match size when using low values of the tie density ($td < 0.6$) and when the tie density is 1. These results are obtained even with low timeout values, i.e. $50ms$. When the tie density is between 0.6 and 0.9, AS-HRT does not always reach the optimal solution: yet in these cases, the minimum ratio to the optimal size we obtained was 0.998 times (for the $td = 0.9$).



(a)



(b)

Fig. 5: AS-HRT: a) match size and b) execution time (size=300, varying td)

Figure 5b presents the average execution time of AS-HRT. The results show that when using $td = 1$, the execution time to get the optimal solution is *almost* constant, at about $40ms$. When the tie density decreases, the average execution time tends to be the same as the chosen timeout value. This behaviour can be explained because, when the optimal solution of a problem instance is not perfect ($|M| < 300$), the AS-SMTI method, which doesn't know this, will keep trying to improve on it until the timeout is reached.

4 Conclusions and Future Work

We recently developed a Local Search solver for SMTI problems based on the Adaptive Search method. To use this, we need to model problems as permutations and provide some heuristics based on the study of relevant blocking pairs to guide the search process, in order to iteratively improve the current matching. A reset procedure is invoked when the solver is trapped in local minima. This solver is very efficient and can solve optimally large instances quickly. For the most difficult problems, it is possible to tune the trade-off between solution quality and solving performance by tweaking the timeout parameter.

On the top of this solver, we have built a solver for HRT which basically maps an HRT problem to an equivalent SMTI problem using the *cloning technique*. This required a slight modification to the core solver. We also characterized the resulting clones with the notion of *equivalence* which captures the fact that clones are interchangeable and thus it is not desirable to replace a matching by another which is equivalent (i.e. one that only differs in equivalent elements). We showed that the core LS algorithm mainly satisfies this property, save in the reset procedure. We have plans to improve on this situation.

We presented a preliminary experimental evaluation based on thousands of problems of size 300. The solver already performs very well: using a timeout of $1s$, it reaches the optimal solution for most of the instances. For the most difficult instances, when the optimum is not reached within this timeout, the returned solutions are very good with size within a factor 0.998 w.r.t. the optimal solution. We plan to experiment with other dataset reflecting a more realistic case of the HRT problem, e.g. modeling the popularity or unpopularity of the hospitals. Moreover, we also plan to use larger problem instances, as a future development.

There are several avenues for improvement, of which we name a few: we already mentioned how to take equivalence into account in the reset procedure. It is also possible to see if other problem reduction techniques are fruitful (e.g. those mentioned in [16] for the IP formulation). It would also be interesting to start from a pertinent solution, instead of a pure random assignment: this could be done with the help of a very fast approximation algorithm. The size of this solution could be also used as a lower bound.

Finally, as our base method is amenable to massive parallelisation, we will explore parallelism to tackle both hard and large problem instances, as we already did in [19].

Acknowledgments

We would like to thank Augustine Kwanashie and David F. Manlove from the University of Glasgow, for kindly providing us with the extensive HRT data sets.

References

1. P. Codognet and D. Diaz. Yet Another Local Search Method for Constraint Solving. In K. Steinhöfel, editor, *Stochastic Algorithms: Foundations and Applications*, pages 342–344. Springer Berlin Heidelberg, London, 2001.
2. P. Codognet and D. Diaz. An Efficient Library for Solving CSP with Local Search. In *5th international Conference on Metaheuristics*, pages 1–6, Kyoto, Japan, 2003.
3. L. Dubins and D. Freedman. Machiavelli and the Gale-Shapley Algorithm. *American mathematical monthly*, 88(7):485–494, 1981.
4. D. Gale and L. Shapley. College Admissions and the Stability of Marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.
5. M. Gelain, M. Pini, F. Rossi, K. Venable, and T. Walsh. Local Search Approaches in Stable Matching Problems. *Algorithms*, 6(4):591–617, Oct. 2013.

6. I. Gent and P. Prosser. An Empirical Study of the Stable Marriage Problem with Ties and Incomplete Lists. In *in ECAI 2002*, pages 141–145. IOS Press, 2002.
7. I. Gent, P. Prosser, B. Smith, and T. Walsh. SAT Encodings of the Stable Marriage Problem with Ties and Incomplete Lists. In *SAT*, volume 8, pages 133–140, Cincinnati, USA, 2002.
8. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, July 1997.
9. D. Gusfield and R. Irving. *The Stable Marriage Problem: Structure and Algorithms.* MIT press, 1989.
10. R. Irving. Stable Marriage and Indifference. *Discrete Applied Mathematics*, 48(3):261–272, Feb. 1994.
11. R. Irving and D. Manlove. Finding Large Stable Matchings. *ACM Journal of Experimental Algorithmics*, 14(September), 2009.
12. R. Irving, D. Manlove, and S. Scott. The Hospitals/Residents Problem with Ties. In *7th Scandinavian Workshop on Algorithm Theory, SWAT'00 Lecture Notes in Computer Science*, number i, pages 259–271, Berlin, 2000.
13. K. Iwama, D. Manlove, S. Miyazaki, and Y. Morita. Stable Marriage with Incomplete Lists and Ties. In *In Proceedings of ICALP 99: the 26th International Colloquium on Automata, Languages and Programming*, number ii, pages 443–452. Springer-Verlag, 1999.
14. K. Iwama and S. Miyazaki. A Survey of the Stable Marriage Problem and its Variants. In *International Conference on Informatics Education and Research for Knowledge-Circulating Society , ICKS'08*, number i, pages 131–136. IEEE Press, 2008.
15. F. Klijn and J. Masso. Weak Stability and a Bargaining Set for the Marriage Model. *Games and Economic Behavior*, 42(1):91–100, 2003.
16. A. Kwanashie and D. Manlove. An Integer Programming Approach to the Hospital/Residents Problem with Ties. *CoRR*, abs/1308.4:1–10, Aug. 2013.
17. D. Manlove, R. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard Variants of Stable Marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002.
18. E. McDermid. A 3/2-approximation Algorithm for General Stable Marriage. In *International Colloquium on Automata, Languages and Programming, ICALP'2009*, pages 689–700, Rhodes, Greece, 2009.
19. D. Munera, D. Diaz, S. Abreu, F. Rossi, V. Saraswat, and P. Codognet. Solving Hard Stable Matching Problems via Local Search and Cooperative Parallelization. In *AAAI*, 2015.
20. A. Podhradsky. Stable Marriage Problem Algorithms. Master's thesis, Masaryk University, 2010.
21. A. Roth and O. Sotomayor. Two Sided Matching: A Study in Game-Theoretic Modeling and Analysis. *Econometric Society Monographs*, 18, 1990.