Solving Hard Stable Matching Problems via Local Search and Cooperative Parallelization

Danny Munera
University Paris 1/CRI
danny.munera@malix.univ-paris1.fr

Daniel Diaz
University Paris 1/CRI
daniel.diaz@univ-paris1.fr

Salvador Abreu
University of Évora/CENTRIA/CRI
spa@di.uevora.pt

Francesca Rossi
University of Padova/Harvard University
frossi@math.unipd.it

Vijay Saraswat
IBM TJ Watson Research Center
vsaraswa@us.ibm.com

Philippe Codognet
JFLI-CNRS/UPMC University of Tokyo
codognet@is.s.u-tokyo.ac.jp

Abstract

Stable matching problems have several practical applications. If preference lists are truncated and contain ties, finding a stable matching with maximal size is computationally difficult. We address this problem using a local search technique, based on Adaptive Search and present experimental evidence that this approach is much more efficient than state-of-the-art exact and approximate methods. Moreover, parallel versions (particularly versions with communication) improve performance so much that very large and hard instances can be solved quickly.

1 Introduction

In 1962, Gale and Shapley introduced the Stable Marriage (SM) problem (Gale and Shapley 1962). An SM instance of size n involves a set of n men and a set of n women, each of whom have ranked all members of the other set in a strict order of preference. Solving such a problem consists in finding a *marriage*, i.e. a one-to-one matching between the men and the women. In addition the marriage 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 marriage always exists and proposed a $O(n^2)$ 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 restrictive for many real-life, large-scale applications. A natural variant of SM is the *Stable Marriage with Incomplete List and Ties* (SMTI) problem (Iwama et al. 1999; Manlove et al. 2002). In SMTI, the preference lists may include ties (to express indifference) and may be incomplete (to express that some partners are unacceptable) The goal now is to find the stable matching of maximal size (that is, with the smallest number of singles). This problem is NP-hard.

SMTI is a special case of the Hospitals/Residents problem (Manlove et al. 2002), for which there are nation-wide programmes in several countries. Matching problems can also be found in other settings, such as car sharing or bipartite market sharing, job markets and social networks. Many of these applications involve very large sets.

We show how to model SMTI problems as permutation problems and solve them using a local search approach, based on *Adaptive Search* (AS) (Codognet and Diaz 2001; 2003). We compare experimentally the sequential version of our solver with state of the art exact and approximate algorithms to solve such problems, showing significant improvement in performance or solution quality.

Moreover, our implementation uses a parallel framework (Munera et al. 2014) written in X10 (Charles et al. 2005; Saraswat et al. 2012). We show that the independent parallel version exhibits a significant speedup with increasing number of cores and the cooperative version achieves super-linear speedup on average and behaves very well on hard instances.

The rest of the paper is organized as follows. Section 2 provides the necessary background. Section 3 details the AS modeling of SMTI. In Section 4 we evaluate the performance of the AS implementation. Finally, Section 5 asseses the parallel performance of our implementation.

2 Background

Stable Marriage with Ties and Incompleteness

We recall the main definitions for SMTI problems (Scott 2005; Iwama and Miyazaki 2008).

Definition 1. *(SMTI problem) 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.*

While not required by the theory we assume that the preference list for woman w does not contain man m iff the list for m does not contain w. This is trivial to establish with a pre-pass; the generator we use (Gent and Prosser 2002) already ensures this.

Definition 2. *(Marriage) Given an SMTI instance, a* marriage *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 marriage M is the cardinality of M.*

With the introduction of ties in the preference lists, three different notions of stability may be used (Irving 1994; Manlove et al. 2002; Iwama and Miyazaki 2008). As we only consider *weak stability* (the most challenging), we simply call it stability. We define *blocking pair* in this context:

Definition 3. *(Blocking Pair) In a marriage M, (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.*

Definition 4. *(Stable marriage) Given an SMTI problem instance, a marriage M is stable iff it has no blocking pairs.*

A (weakly) stable marriage 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 those of maximum cardinality. This optimization problem (called MAX-SMTI) has many real-life applications (Iwama and Miyazaki 2008; Manlove et al. 2002) and has attracted a lot of research in recent years. 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) (Iwama et al. 1999; Manlove et al. 2002). For brevity, in the rest of this article we refer to MAX-SMTI as SMTI.

SMTI algorithms Finding efficient algorithms to solve SMTI has been an active field of research for several years, driven by its applications. SMTI has been shown to be APX-hard (Halldórsson et al. 2003) and most of the recent research focuses on designing efficient *approximation algorithms*, i.e. algorithms running in polynomial time yet able to guarantee solutions within a constant factor of the optimum (Király 2011). Currently, the best algorithms are 3/2-approximation algorithms (McDermid 2009; Király 2013; Paluch 2014). An r-approximation algorithm always finds a stable matching M with $|M| \geq |M_{opt}|/r$ where M_{opt} is a stable marriage of maximum size. SMTI cannot be approximated within a factor 21/19 and probably not within a factor of 4/3 either (Halldórsson et al. 2007). These algorithms only find one solution for a given problem, while it is often useful to provide multiple (quasi-)optimal solutions.

Constraint Programming (CP) can be used to solve SMTIs. While there are some efforts to solve SM problems (Gent et al. 2001), including the introduction of new global constraints to achieve efficient consistency (Unsworth and Prosser 2013; Manlove et al. 2007), few papers are devoted to the MAX-SMTI variant. The reference paper (Gent and Prosser 2002) is more a study of the properties of SMTI problems of limited size (i.e. $n \leq 60$) than a search for efficient encodings to solve SMTI using CP.

Surprisingly, SAT solvers have not been extensively used for SMTI. The work by Gent (Gent et al. 2002) proposes a SAT encoding for SMTI and uses the Chaff solver for its evaluation. Linear programming (Roth, Rothblum, and Vande Vate 1993; Vande Vate 1989) and integer programming (Kwanashie and Manlove 2013) have been also studied for variants of SM like the Hospitals/Residents problem. Recently, Local Search has been successfully applied to SMTI. In (Gelain et al. 2013), the authors propose LTIU, a local search algorithm with very good performance. Comparisons with some of these papers are in Section 4.

The Adaptive Search method

Adaptive Search (AS) was proposed in (Codognet and Diaz 2001) as a generic, domain-independent, constraint-based local search method. This meta-heuristic takes advantage of the model of the problem in terms of constraints and variables in order to guide the search more precisely than a single global cost function. AS starts from a random assignment of the variables (i.e. a *configuration*) and, iteratively, tries to improve it, modifying one variable at a time until a solution is found. To this end AS:

- Needs to model the constraints with heuristic functions that compute an approximate degree of satisfaction of the goals (the current *error* on the constraint);
- Combines these errors to compute the global cost of a configuration;
- For each variable, aggregates the errors of constraints in which it occurs, and repairs the *worst* variable (highest error) with the most promising value;
- maintains a short-term memory (e.g. *tabu list*) of recently modified variables which led to local minima, together with a reset mechanism (i.e. *iterative local search*).

AS has shown good performance on combinatorial problems such as classical CSPs, and the Costas Array Problem (Caniou et al. 2014).

3 An Adaptive Search Model for SMTI

We model SMTI in AS as a permutation problem: the sequence of n $(X_1 \dots X_n)$ takes on as values permutations of the values $1 \dots n$ (implementing an `all-different` constraint). $X_i = j$ 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 this interpretation remains valid when the values of any two variables are swapped (this is how value assignment is implemented in permutation problems).

To improve stability of a marriage, we must remove 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. We thus focus on the so-called *undominated blocking pairs* (Klijn and Massó 2003; Gelain et al. 2013).

Definition 5. *(Dominated blocking pair) BP (m, w) dominates BP (m, w') iff m prefers w to w'.*

Definition 6. *(Undominated blocking pair) BP (m, w) is undominated iff there is no other BP dominating (m, w).*

These definitions are from the men's point of view: equivalent ones exist for women. From an implementation perspective, finding the undominated BP of a man m amounts to consider each woman w in his preference list in descending order of preference, stopping at the first BP encountered: (m, w) is then an undominated BP. In the following we only consider *undominated* BPs, which we simply call BPs.

The cost function of a marriage measures both its stability (number of BPs) and its quality (number of singles). Hence: $cost(M) = \#BP(M) \times n + \#Singles(M)$, where $\#BP(M)$ is the number of BPs in M, and $\#Singles(M)$ is

the number of singles in M. The number of BPs is weighted with n to prioritize stable marriages over marriages with fewer singles. A marriage 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 marriage found so far.

We define $R(w, m)$ 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. Suppose (m, w) and $(m', w') \in M$ form a BP (m, w'), then the error for X_m (in M) is $R(w', m') - R(w', m)$. Thus, the further the assigned man is from the BP, the larger the error. Algorithm 1 has details – it is worth pointing out that when w' is single in M, the returned score could be $R(w', m')$; this often over-estimates the importance of this case and we have empirically found that 1 is a better choice. Note that the actual implementation does some straightforward pre-computation to avoid the linear cost of recomputing $R(w, m)$.

Algorithm 1 Function to compute BP error

Input: $(m', w') \in M$ and a man m who prefers w' to his partner
Output: if (m, w') is a BP, return an error > 0 else return 0

```
1:  function BP_ERROR(m',w',m)
2:      rankM' ← R(w', m')        ▷ rank of current partner of w'
3:      rankM  ← R(w', m)         ▷ rank of m in pref. list of w'
4:      if rankM = n + 1 then     ▷ m ∉ pref. list of w'
5:          return 0              ▷ m not a valid partner for w' : not a BP
6:      end if
7:      if rankM' = n + 1 then
8:          return 1              ▷ w' is single (m' is not in her list) : BP
9:      end if
        ▷ using the pref. list of w', the error is the difference between
          the rank of her partner m' and proposed man m
10:     return max(0, rankM' − rankM)
11: end function
```

At each iteration, AS selects the "worst" variable from the current marriage M to improve it. If several variables have the same error, AS randomly picks one. 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 marriage. This heuristic avoids the cost of fixing all BPs, one by one.

Since both the evaluation of the cost function and the evaluation of variable errors require the computation of the blocking pairs, when the cost function is evaluated, the error on variables can also be computed and tabled for later use. This is shown in Algorithm 2.

In most cases, the resulting marriage 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. To this end, AS relies on a Tabu list to prohibit the use of recent culprit variables. When the list becomes too large, AS invokes a *reset procedure* to alter the current configuration. The Tabu mechanism is not used for SMTI: as soon as a local minimum is reached a customized reset procedure is invoked which basically tries to fix the 2 worst BPs and/or to assign a woman to a single man, as detailed in Algorithm 3. This procedure is stochastic; it takes a probability p to *also* fix the second worst variable: good results are obtained with a high probability, 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.

Algorithm 2 Function to evaluate a marriage

Input: M the marriage to evaluate
Output: the global cost and error on variables

```
1:  function COST_OF_MARRIAGE(M)
2:      nSingles ← 0
3:      for m ← 1 to n do
4:          w ← X_m                    ▷ (m, w) ∈ M or m is single
5:          rankW ← R(m, w)            ▷ rank of w in pref. list of m
6:          if rankW = n + 1 then      ▷ m is single
7:              nSingles ← nSingles + 1
8:          end if
9:          for all w' ∈ pref list of m with rank < rankW do
10:             let (m', w') ∈ M       ▷ m' is the partner of w' in M
                ▷ check if (m, w') forms a BP
11:             error_m ← BP_ERROR(m', w', m)
12:             if error_m > 0 then
13:                 #BP ← #BP + 1
14:                 break              ▷ only consider undominated BP
15:             end if
16:         end for
17:     end for
18:     return #BP × n + nSingles
19: end function
```

Algorithm 3 Reset procedure to escape local minima

Input: M the marriage currently trapped in a local minimum and p the probability to also fix the second worst variable
Output: M the perturbed marriage to escape the local minimum

```
1:  procedure RESET(M, p)
2:      if #BP(M) ≥ 1 then
3:          fix the worst variable                          ▷ ie. 1 swap
4:          if #BP(M) ≥ 2 and with a probability p then
5:              fix the second worst variable               ▷ ie. 1 swap
6:              return                                      ▷ exit the procedure
7:          end if
8:      end if
9:      if nSingles(M) ≥ 1 then
10:         randomly select a single man
11:         and assign him a random woman                   ▷ ie. 1 swap
12:     else
13:         randomly swap 2 variables                       ▷ ie. 1 swap
14:     end if
15: end procedure
```

4 Performance Evaluation

In this section, we compare our AS modeling of SMTI problems to other approaches, both from the point of view of performance and that of solution quality, i.e. size of a marriage.

This evaluation is important for two main reasons: first, it assesses whether AS is a useful approach to tackle SMTI problems. Since (Gelain et al. 2013) it has been known that local search is a viable way to solve SMTI problems, we will show that AS improves on that. Second, the comparison demonstrates the quality of the sequential implementation, which is important when evaluating the performance of the parallel version, relative to the sequential one.[1]

To this end, we used an X10 implementation of AS, further discussed in Section 5, running sequentially on an AMD Opteron 6376 clocked at 2.3 GHz, i.e. using only one core.

Problem Set For the evaluation, we used the random problem generator described in (Gent and Prosser 2002) which takes three parameters: the size (n), the probability of incompleteness ($p1$) and the probability of ties ($p2$). We generated problems of size $n = 100$, with $p1$ ranging over $[0.1, 0.9]$ and $p2$ over $[0, 1]$, with step 0.1. For each $(p1, p2)$ pair, we solved 100 instances and averaged the results.

Comparison with Local Search

We first consider the Local Search method LTIU of (Gelain et al. 2013). We ran the AS implementation on the test benchmark, in the same conditions as the LTIU paper: using the same problem generator, solving each instance once, with a limit of 50 000 iterations and using only 1 core.

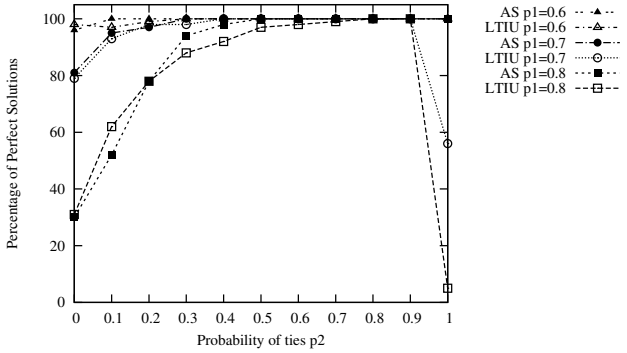


Figure 1: AS vs. LTIU - quality of solutions.

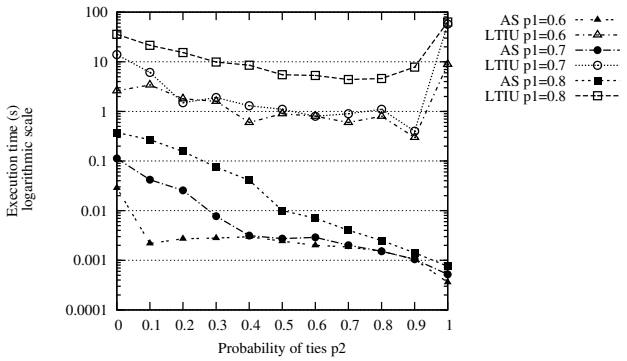


Figure 2: AS vs. LTIU - execution time.

Figure 1 compares the quality of solutions. The percentage of perfect stable marriages found is slightly better for AS. Moreover, AS does not suffer from the LTIU's dramatic performance loss when $p2 = 1$.

Figure 2 compares execution times. The AS implementation is much faster (thus the log Y scale): the LTIU method averages over $30s$ on a similar machine (Gelain et al. 2010) while AS is two orders of magnitude faster. When $p2$ increases, the lead extends even further.

Comparison with Approximation Algorithms

We compared AS against McDermid's method (MD) (McDermid 2009), a very efficient 3/2-approximation algorithm, as implemented in (Podhradsky 2010). For MD also, and for each $(p1, p2)$ pair, we ran the same 100 instances once, averaging the execution time.

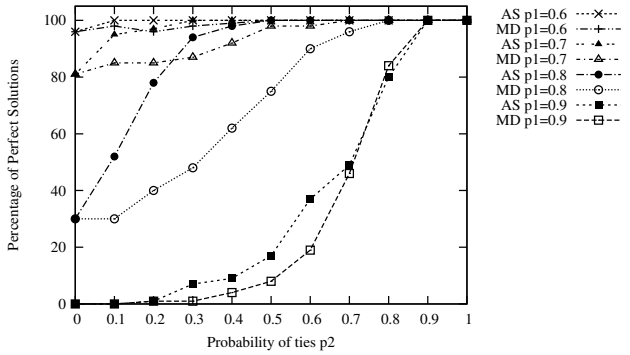


Figure 3: AS vs. MD - quality of solutions.

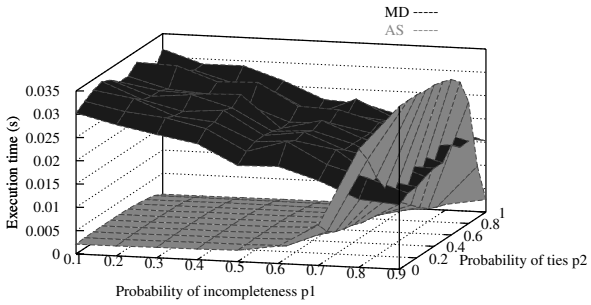


Figure 4: AS vs. MD - execution time.

Figure 3 compares the quality of solutions. The percentage of perfect stable marriages found by the AS algorithm is considerably higher than those found by MD, in particular using a probability of ties $p2 \in [0.1..0.7]$.

Figure 4 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

[1]Both the source code and problem instances are available at `http://cri-hpc1.univ-paris1.fr/smti/`

the time-complexity of MD which is proportional to the total length of the preference lists, i.e. it linearly decreases as $p1$ increases.

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.

Comparison with SAT

We also compare AS to the SAT encoding for SMTI of (Gent et al. 2002), restricting thus to the decision problem: *is there a stable matching of size n?* which we answer by actually finding a perfect stable matching.

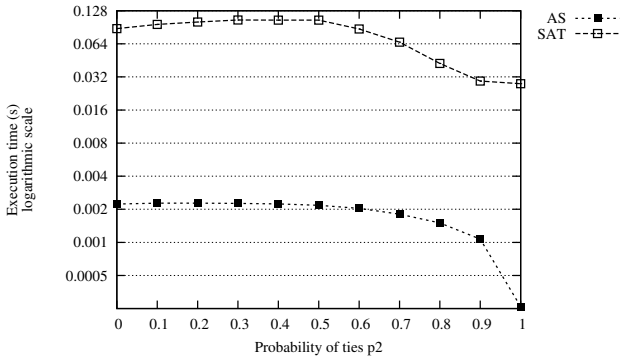


Figure 5: AS vs. SAT - execution time ($p1 = 0.5$).

Figure 5 presents the execution times using weak stability, for $n = 100$, i.e. the case of figure 2, top left from (Gent et al. 2002). The execution times for SAT were divided by 25.2, according to the ratio of the SPECint CPU performance ratings of both machines but, even so, AS outperforms the SAT version by a factor of about 50.

5 Parallelization

Parallel versions of local search procedures have been proposed already (Alba 2005; Alba, Luque, and Nesmachnow 2013). In this article we are interested in *multi-walks* methods (also called *multi-starts*) which consist in a concurrent exploration of the search space, either *independently* or *cooperatively* with some communication between concurrent processes. The *Independent Multi-Walks* method (IW) (Verhoeven and Aarts 1995) is the easiest to implement since the solver instances do not communicate with each other. However, the resulting gain tends to flatten when scaling over a hundred of processors (Caniou et al. 2014), and can be improved upon. In the *Cooperative Multi-Walks* (CW) method (Toulouse, Crainic, and Sans 2004), the solver instances exchange information (through communication), hoping to hasten the search process. However, implementing an efficient cooperative method is a very complex task: several choices have to be made about the communication (Toulouse, Crainic, and Sans 2004) which influence each other and which are problem-dependent.

The Cooperative Parallel Local Search Framework

We build on the framework for Cooperative Parallel Local Search (CPLS) proposed in (Munera et al. 2014; Munera, Diaz, and Abreu 2013). This framework, made available as an open source library in X10, allows the programmer to tune the search process through an extensive set of parameters. CPLS augments the IW strategy with a tunable communication mechanism, which allows for the cooperation between the multiple instances to seek either an intensification or diversification strategy for the search.

Explorer nodes are the basic components in the framework: each consists in a local search solver instance. The point is to use all the available processing units by mapping each *explorer node* to a physical core. Explorer nodes are grouped into *teams*, of fixed parametric size. Each team seeks to *intensify* the search in the most promising neighborhood found by any of its members. The parameters which guide the intensification are the *Report Interval* (R) and *Update Interval* (U): every R iterations, each explorer node sends its current configuration and the associated cost to its *head node*. The head node is the team member which collects and processes this information, retaining the best configurations in an *Elite Pool* (EP) whose size is $|EP|$. Every U iterations, explorer nodes randomly retrieve a configuration from the EP, in the head node. An explorer node may *adopt* the configuration from the EP, if it is "better" than its own current configuration with a probability *pAdopt*. Simultaneously, the teams implement a mechanism to cooperatively *diversify* the search, i.e. they try to extend the search to different regions of the search space. A detailed description of this framework may be found in (Munera et al. 2014).

Independence vs Cooperation

In this section we assess the parallelization of the AS model for SMTI testing both IW and CW strategies. These experiments are made relatively simple when using the X10 implementation of the CPLS framework which clearly separates the local search algorithm proper from the management of parallelism (e.g. process management, communication, synchronization.) This separation allowed us to focus on the (sequential) local search algorithm and on its X10 encoding. Tuning is done by setting the parameters controlling the parallel execution, for instance *nodes*, *cores per node*, *communication scheme*, *delays*. To test with IW, all communications are simply deactivated. To experiment with CW, the parameters controlling the cooperation have to be fine tuned.

Cooperation parameters We experimented with the most important parameters in order to analyze the impact of each one over the global performance. Due to space limitations, we do not detail these experiments – we only state the retained parameters. It turned out that *intensification* is much more important than *diversification* for SMTI. The best results are obtained with 2 *teams*. The *number of explorers* of a team is half the number of used cores. Inside a team, each explorer periodically exchanges information with the *Elite Pool* according to the *report interval* (R) and *update interval* (U). The best settings were found to be $R = 50$, $U = 100$ ($\frac{U}{R} = 2$ being the best ratio), $|EP| = 4$ and $pAdopt = 1$.

Problem Set As shown in the previous section, SMTI problems of size 100 are very easy to solve with sequential AS. We therefore consider problems of size 1000 generated with $p1 = 0.95$ and $p2 = 0.8$. We selected these parameters because the resulting problems involve a large number of variables, a huge search space ($1000! \simeq 10^{2567}$) and because they are difficult to solve due to the high level of incompleteness in the preference lists. For this experiment, we generated 10 random problems and executed each one 50 times (the results are averaged), varying the number of cores from 1 (sequential) to 128. Using an unlimited timeout forced the solver to discover perfect stable marriages.

Parallel hardware All parallel experiments have been carried out on a cluster of 16 machines, each with 4×16-core AMD Opteron 6376 CPUs running at 2.3 GHz and 128 GB of RAM. The nodes are interconnected with InfiniBand FDR $4\times$ (i.e. 56 GBPS.) We had access to 4 nodes and used up to 32 cores per node, i.e. 128 cores. We made no attempt to control thread placement. In the rest of this section, the execution times are given in seconds and correspond to *wall time* which is the real elapsed time, and includes the time to install all solver instances, the time to solve the problem, the time for communications and the time to detect and propagate the termination.

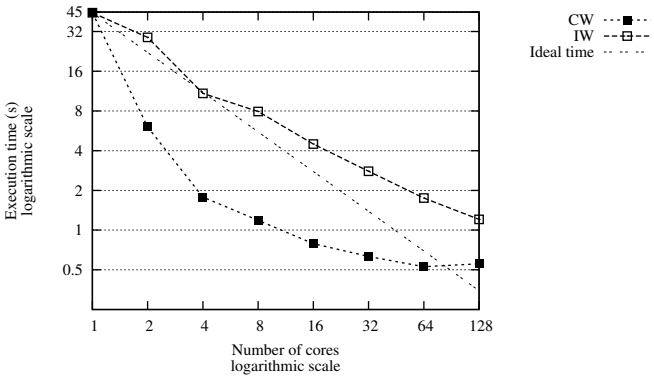


Figure 6: Execution time using IW and CW.

Figure 6 presents a log-log graph of execution times using IW and CW. The *Ideal time* corresponds to linear speedup: time is halved when the number of cores is doubled. It is worth noticing that IW is rather efficient: using 128 cores the average time decreases from $44.453s$ to $1.207s$ which corresponds to a speedup factor of 37. However, as is often the case with IW, it is difficult to obtain a linear speedup. Moreover, this sub-linear speedup tends to taper off. The results with CW are much better. The best recorded speedup is 86 using 80 cores ($44.5s$ to $0.519s$). Beyond this number of cores, the time tends to stabilize. This could mean that we have reached an incompressible limit to solve problems of size 1000, or that we are being limited by low-level factors.

Evaluation on Hard Problems

We evaluate the benefit of parallelism on hard problems. To do so, we generated 100 random problems of size $n = 1000$, ran them sequentially and selected the 10 hardest instances. We then repeated the previous experiment, using only these instances.

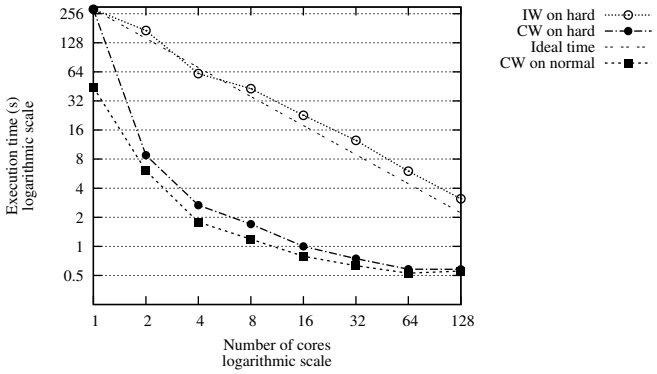


Figure 7: Execution time on hard problems

Figure 7 presents a log-log graph of the execution time using IW and CW for these hard problems.We also recall the CW curve obtained in the previous experiment (i.e. on *normal* problems), for reference.

The difficulty of the problems is clear from the sequential time: in the previous experiment, a problem was solved in about $44s$ while about $285s$ are needed for the hardest instances. Using IW, we reach a quasi-linear speedup: 91.5 for 128 cores which corresponds to a reduction of the execution time from $284.5s$ to $3.1s$. However the execution time remains much slower than for *normal* problems. It turns out that, when using cooperation in CW, execution times are drastically improved and the best speedup is 492 with 128 cores corresponding to an execution time of $0.579s$. It is worth noticing that this time is very similar to the best time ($0.519s$) obtained in the previous experiment for normal problems.

From a practical point of view, it appears that parallelism with cooperation neutralizes the relative difficulty of problems, as instances which are originally about 6 times harder get solved in approximately the same time. Of course, the problem retains its worst-case NP-hard complexity, and parallel search cannot change this.

6 Conclusion

We proposed to model SMTIs as permutation problems and solve them using a local search approach based on Adaptive Search. The sequential version of this solver outperforms state-of-the-art solvers for SMTIs. Moreover, the parallel version, especially the one with cooperation among concurrent processes, shows super-linear speedup and performs exceptionally well, particularly on very hard instances.

We plan to experiment with very large instances on a massively parallel machine to test the scaling limitations. We also plan to study which features of the stable matching problem make it so suitable for cooperation. We also intend to adapt our solver to tackle related problems, such as the Hospitals/Residents problem allowing ties (at first restricted to the hospitals preferences and, later, also for residents).

References

Alba, E.; Luque, G.; and Nesmachnow, S. 2013. Parallel metaheuristics: recent advances and new trends. *Int. Trans. Oper. Res.* 20(1):1–48.

Alba, E. 2005. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience.

Caniou, Y.; Codognet, P.; Richoux, F.; Diaz, D.; and Abreu, S. 2014. Large-scale parallelism for constraint-based local search: the Costas array case study. *Constraints* (published online 2014/7/25).

Charles, P.; Grothoff, C.; Saraswat, V.; Donawa, C.; Kielstra, A.; Ebcioglu, K.; von Praun, C.; and Sarkar, V. 2005. X10: An Object-oriented Approach to Non-uniform Cluster Computing. *SIGPLAN Not.* 40(10):519–538.

Codognet, P., and Diaz, D. 2001. Yet Another Local Search Method for Constraint Solving. In *proceedings of SAGA'01*, 73–90. Springer Verlag.

Codognet, P., and Diaz, D. 2003. An Efficient Library for Solving CSP with Local Search. In *5th international Conference on Metaheuristics*, 1–6.

Gale, D., and Shapley, L. S. 1962. College Admissions and the Stability of Marriage. *The American Mathematical Monthly* 69(1):9–15.

Gelain, M.; Pini, M.; Rossi, F.; Venable, K. B.; and Walsh, T. 2010. Local search for stable marriage problems with ties and incomplete lists. In *PRICAI 2010: Trends in Artificial Intelligence*, 64–75.

Gelain, M.; Pini, M.; Rossi, F.; Venable, K.; and Walsh, T. 2013. Local Search Approaches in Stable Matching Problems. *Algorithms* 6(4):591–617.

Gent, I. P., and Prosser, P. 2002. An Empirical Study of the Stable Marriage Problem with Ties and Incomplete Lists. In *in ECAI 2002*, 141–145. IOS Press.

Gent, I. P.; Irving, R. W.; Manlove, D. F.; Prosser, P.; and Smith, B. M. 2001. A Constraint Programming Approach to the Stable Marriage Problem. In *CP01*, 225–239. Springer.

Gent, I.; Prosser, P.; Smith, B.; and Walsh, T. 2002. SAT Encodings of the Stable Marriage Problem with Ties and Incomplete Lists. In *SAT*, volume 8, 133–140.

Halldórsson, M. M.; Irving, R. W.; Iwama, K.; Manlove, D.; Miyazaki, S.; Morita, Y.; and Scott, S. 2003. Approximability results for stable marriage problems with ties. *Theor. Comput. Sci.* 306(1-3):431–447.

Halldórsson, M. M.; Iwama, K.; Miyazaki, S.; and Yanagisawa, H. 2007. Improved approximation results for the stable marriage problem. *ACM Transactions on Algorithms* 3(3).

Irving, R. W. 1994. Stable Marriage and Indifference. *Discrete Applied Mathematics* 48(3):261–272.

Iwama, K., and Miyazaki, S. 2008. A Survey of the Stable Marriage Problem and Its Variants. In *International Conference on Informatics Education and Research for Knowledge-Circulating Society*, ICKS '08.

Iwama, K.; Manlove, D.; Miyazaki, S.; and Morita, Y. 1999. Stable Marriage with Incomplete Lists and Ties. In *ICALP, Prague, Czech Republic*, 443–452.

Király, Z. 2011. Approximation of Maximum Stable Marriage. Technical Report TR-2011-03, Egerváry Research Group, Budapest.

Király, Z. 2013. Linear time local approximation algorithm for maximum stable marriage. *Algorithms* 6(3):471–484.

Klijn, F., and Massó, J. 2003. Weak stability and a bargaining set for the marriage model. *Games and Economic Behavior* 42(1):91–100.

Kwanashie, A., and Manlove, D. 2013. An integer programming approach to the hospital/residents problem with ties. *CoRR* abs/1308.4064.

Manlove, D.; Irving, R. W.; Iwama, K.; Miyazaki, S.; and Morita, Y. 2002. Hard variants of stable marriage. *Theor. Comput. Sci.* 276(1-2):261–279.

Manlove, D.; O'Malley, G.; Prosser, P.; and Unsworth, C. 2007. A Constraint Programming Approach to the Hospitals / Residents Problem. In *CPAIOR, Brussels, Belgium, May 23-26, 2007*.

McDermid, E. J. 2009. A 3/2-Approximation Algorithm for General Stable Marriage. In *International Colloquium on Automata, Languages and Programming (ICALP) 2009*, 689–700.

Munera, D.; Diaz, D.; Abreu, S.; and Codognet, P. 2014. A parametric framework for cooperative parallel local search. In *Evolutionary Computation in Combinatorial Optimisation - 14th European Conference, EvoCOP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers*, 13–24.

Munera, D.; Diaz, D.; and Abreu, S. 2013. Towards parallel constraint-based local search with the X10 language. In *Declarative Programming and Knowledge Management - Declarative Programming Days, KDPD*, 169–184.

Paluch, K. E. 2014. Faster and Simpler Approximation of Stable Matchings. *Algorithms* 7(2):189–202.

Podhradsky, A. 2010. Stable Marriage Problem Algorithms. Master's thesis, Masaryk University.

Roth, A. E.; Rothblum, U. G.; and Vande Vate, J. H. 1993. Stable Matchings, Optimal Assignments, and Linear Programming. *Mathematics of Operations Research* 18(4):803–828.

Saraswat, V.; Bloom, B.; Peshansky, I.; Tardieu, O.; and Grove, D. 2012. X10 language specification - Version 2.3. Technical report, IBM Research.

Scott, S. 2005. *A Study of Stable Marriage Problems with Ties*. Ph.D. Thesis, University of Glasgow.

Toulouse, M.; Crainic, T. G.; and Sans, B. 2004. Systemic behavior of cooperative search algorithms. *Parallel Computing* 57–79.

Unsworth, C., and Prosser, P. 2013. An n-ary Constraint for the Stable Marriage Problem. *CoRR* abs/1308.0183.

Vande Vate, J. H. 1989. Linear programming brings marital bliss. *Oper. Res. Lett.* 8(3):147–153.

Verhoeven, M., and Aarts, E. 1995. Parallel Local Search. *Journal of Heuristics* 1(1):43–65.