Soft Constraints in Stable Matching Problems

P. De Caro and M. S. Pini and F. Rossi and K. B. Venable

Abstract

The stable matching problem has many practical applications in two-sided markets, like those that assign doctors to hospitals, students to schools, or buyers to vendors. Usually it is assumed that the participants of one side explicitly express a preference ordering over all those on the other side. This can be unfeasible when the number of options is large or has a combinatorial structure. We consider using soft constraints to model preferences compactly in stable matching problems. We study the impact of this choice on the computational complexity of finding stable matchings. We show that, when agents model their preferences via tree-shaped fuzzy constraint problems, this approach does not increase the complexity of stable matching procedures, while maintaining stability of the matching returned. We also evaluate the approach experimentally.

1 Introduction

The stable matching (SM) problem is a well-known problem with many practical applications. It has to do with two sets of agents, often called men and women, that should be matched in such a way that no man and woman, who are not married to each other, both prefer each other to their current partner [11, 14]. This property is called stability. Problems of this kind arise in many real-life situations, such as assigning junior doctors to hospitals [17], children to schools [18], students to campus housing, kidney transplant patients to donors, and so on.

The most well-known and used algorithm to find a stable matching is the GS algorithm [9], that runs in polynomial time. This algorithm assumes that both men and women express their preferences over all members of the other gender. However, this can be unfeasible, since the number of men and women can be very large. For example, in China, over 10 million students apply for admission to higher education annually via a centralized process.

In addition, eliciting the preferences may be a costly and time-consuming process. However, the sets of men and women may have a combinatorial structure, which allows for expressing preferences in a compact way by referring to features rather than entire men or women. For instance, consider a large set of hospitals offering residencies to doctors. Doctors might not want to rank explicitly all the hospitals, but might wish to express preferences over some of their features. For example, they might say ”I prefer a position close to my home town”, or ”If the hospital is far away from my home town, then I want a better salary”. The same can be for hospitals over doctors, which may have preferences such as ”We prefer doctors who did well on a certain exam”.

Our challenge is to understand how to adapt the GS algorithm to work with such preference statements over features, and to study the impact of this approach over the computational properties of the algorithm.

The main operations performed by the GS algorithm are the following ones: men need to exploit their preferences over women to find their most preferred woman, and possibly also their next most preferred woman (several times), while women need to compare two men according to their preferences over men. Thus we need to check what it means to perform such operations when preference statements are over features of men and women.

In this paper we consider the use of soft constraints [15] to model such preferences, both for men and for women. In this formalism, preferences are modeled in a quantitative way, with several levels of acceptance to express the degrees of preference for the variable assignments. To use such

formalism within the GS stable matching procedure, we need to model the GS operations mentioned above in a soft constraint setting. Soft constraints induce a total order over men and women, possibly with ties. The GS algorithm requires a strict total order (that is, no ties) over men and women. We consider three ways to break ties, with particular attention to fuzzy constraints.

We therefore propose a setting where SM preference lists are modelled via soft constraints, and the GS algorithm is augmented by a soft constraint solver which performs the GS operations over the soft constraints representing preferences of men and women, when needed by the GS algorithm.

It is easy to see that the use of soft constraints reduces the amount of time and space needed by each agent to specify its preference ordering. Moreover, there are benefits also for the stable matching procedure, which receives an input which is much smaller. Furthermore, when the *soft constraints of men* respect some reasonable restrictions, such as a *bounded tree-width* for the constraint graph [7], the time complexity of the stable matching procedure does not suffer in terms of time, as shown by our experimental evaluation. In fact, with this restriction, all the GS operations are polynomial in the number of features, which is much smaller than the number of men and women.

Problems with bounded tree-width occur frequently in practical settings, such as experts systems, evolution theory, and natural language processing as described in [2]. We believe that they are often adequate also when modelling the preferences in a stable matching scenario. For example, in the resident-hospital case, it may be reasonable to organize the features in a hierarchy, and this can be modelled by tree-shaped problems, which have a tree-width 1. Note also that the restriction is needed only for men, since women only perform GS operations which are computationally easy also for problems without a tree shape.

A similar study was done using CP-nets instead of soft constraints [16]. CP-nets [4] have a different expressive power compared to soft constraints. Moreover, they also behave differently in terms of computational complexity of reasoning with them. CP-nets never induce orderings with ties, which is instead typical in soft constraints, but instead present many incomparable outcomes. Also, different techniques are needed to handle CP-nets and soft constraints in SMs. Thus the linearization considered for CP-net was very different than those we define in this paper for soft constraints.

Constraints have been considered in the context of stable matching only as a means to encode the problem and to solve it efficiently [10, 13, 1], rather than as a way to compactly model the preferences of the agents and then to solve the problem by interleaving the GS algorithm and the constraint solver.

2 Background

2.1 Stable matching problems

The *stable matching problem* (SM) [11, 14] is the problem of finding a matching between the elements of two sets, usually called men and women. Given n men and n women, where each person strictly orders all members of the opposite sex, we wish to marry the men to the women such that there is not a man and woman who would both rather be married to each other than to their current partners. If there is no such couple, the matching is called *stable*.

Here we consider the classical version of the stable marriage problem, where there is the same number of men and women, and each person has a strict total order over the members of the other gender. Other versions allow for different numbers of men and women, ties and/or incomplete lists in the preference orderings [12].

The *Gale-Shapley algorithm* (GS) [9] is widely used to solve SMs. The algorithm takes $O(n^2)$ steps and constructs a stable matching. It consists of a number of rounds in which each un-engaged man proposes to his most preferred woman to whom he has not yet proposed. Each woman receiving a proposal becomes "engaged", provisionally accepting the proposal from her most preferred man. In subsequent rounds, an already engaged woman can "trade up", becoming engaged to a more

preferred man and rejecting a previous proposal, or if she prefers him, she can stick with her current partner.

Given a matching M, we will denote with $M(w)$ (resp., $M(m)$) the man (resp., woman) associated to the woman w (resp., man m) in M. Also, $pref(x)$ denotes the preference list of a man or a woman x. The GS algorithm (see Algorithm 1), includes the following operations:

- $Opt(pref(m))$: Computes the optimal woman for m (i.e., m's first proposal).
- $Next(pref(m), w)$: computes the next best woman after w for man m (i.e., a new proposal for m).
- $Compare(pref(w), m, m')$: returns true if woman w prefers man m to m'. This is needed when woman w, currently matched with m', must decide whether to accept or decline a proposal from m.

Algorithm 1: GS

```
foreach man m and woman w do
    M(m) ← null; M(w) ← null;
while ∃m such that M(m) = null do
    w ← Opt(pref(m));
    while M(m) = null do
        if M(w) = null then
            M(m) ← w; M(w) ← m;
        else
            if Compare(pref(w), m, M(w)) then
                pref(m) ← pref(m) − {w_i | w_i ⪯ M(m)};
                M(M(w)) ← null; M(w) ← m; M(m) ← w;
            else
                w ← Next(pref(m), w);
```

2.2 Soft constraints

Given a set of variables X, a soft constraint [15] over X is a way to associate a preference value to each assignment of the variables in X. Such preference values belong to a (totally or partially ordered) set which is equipped with an operator to combine preference values. A soft constraint problem (soft CSP) is a set of variables and set of soft constraints over subsets of these variables.

A *classical* CSP [8] is just a soft CSP where the preference structure has just two values (True and False) and preferences are combined via logical and. Classical CSPs are useful when we know what to accept and what to reject.

Fuzzy CSPs are instead modeled with preference values between 0 and 1, totally ordered, where a higher value denotes a more preferred item, and preferences are combined via the min operator. That is, we maximize the minimum preference. Fuzzy CSPs are useful when we have safety-critical applications, since we just focus on the worst preference value when we evaluate a complete variable assignment.

In *weighted* CSPs, preferences are interpreted as costs from 0 to $+\infty$, and we minimize the sum of costs. Weighted CSPs are the ones to use when we have costs or penalties, and it is natural to sum them up to assign a value to a complete assignment.

Given an assignment s to all the variables of a soft CSP P, its preference, written $pref(P, s)$, is obtained by combining the preferences associated by each constraint to the subtuples of s referring to the variables of the constraint. For example, in fuzzy CSPs, the preference of a complete assignment is the minimum preference given by the constraints. In weighted constraints, it is the sum of the costs given by the constraints. An optimal solution of a soft CSP P is then a complete assignment s

such that there is no other complete assignment s' with $pref(P, s) < pref(P, s')$, where $<$ is the preference ordering of the considered preference structure.

In general, finding an optimal solution for a soft CSP is computationally hard. However, it is polynomial for some classes of soft constraints. This is the case for tree-shaped fuzzy CSPs, where a technique called directional arc-consistency, applied bottom-up on the tree shape of the problem, is enough to make the search for an optimal solution backtrack-free and thus polynomial. A tree-shaped soft CSP is a soft CSP whose constraint graph (where nodes represent variables and arcs connect variables involved in the same constraint) is a tree. Given a variable ordering o, a fuzzy CSP is directional arc-consistent (DAC) if, for any two variables x and y linked by a fuzzy binary constraint, such that x precedes y in the ordering o, we have that, for each a in the domain of x, $f_x(a) = max_{b \in D(y)}(min(f_x(a), f_{xy}(a, b), f_y(b)))$, where f_x, f_y, and f_{xy} are the preference functions of c_x, c_y and c_{xy}. If a soft CSP is not DAC, it is possible to make it DAC in polynomial time. Once the fuzzy CSP is DAC, we may find an optimal solution by instantiating each variable in the ordering o in linear time.

The tree-like restriction is not the only one to assure tractability. In fact, it is polynomial to find a solution even if, instead of a tree, we have a graph with cycles but with bounded tree-width. Many classes of graphs do have bounded tree-width, such as cactus graphs (where every two cycles have at most one vertex in common), pseudo-forests (in which every connected component has at most one cycle), series-parallel graphs, outerplanar graphs, Halin graphs, Apollonian networks [3], as well as the control flow graphs arising in the compilation of structured programs.

Fuzzy CSPs can also be solved via a cut-based approach. Given a fuzzy CSP P, an α-cut of P, where α is between 0 and 1, is a classical CSP with the same variables, domains, and constraint topology as the given fuzzy CSP, and where each constraint allows only the tuples that have preference above α in the fuzzy CSP. We will denote such a problem by $cut(P, \alpha)$. The set of solutions of P with preference greater than or equal to α coincides with the set of solutions of $cut(P, \alpha)$. Thus, to find an optimal solution for a given fuzzy CSP, it is enough to find a solution of the CSP $cut(P, \alpha)$ with the highest α such that the problem has some solution.

Fuzzy and weighted CSPs generate a solution ordering which is a *total order with ties*, where the ties are given by all the solutions with the same preference value, and a solution dominates another one if its preference value is higher. Thus, linearizing the solution ordering just means giving an order over the elements in each tie. It has been shown in [5] that it is possible to define a linearization of the solution ordering of a tree-shaped fuzzy CSP where finding the next solution is computationally easy.

For weighted CSPs, instead, unfortunately there in no linearization with this property. However, in [7], it has been shown that, for weighted CSPs with bounded tree-width, it is polynomial to find the top k solutions when k is bounded. So, to find the next solution after we already have the top $k - 1$ ones, it is easy if k is bounded.

3 Modelling SMs via soft constraints

We consider a stable marriage problem with n men and n women, where each man and each woman specify their preferences over the members of the other gender via a set of soft constraints. We call this a Soft CSP based SM (SSM).

Each man and woman is described by a set of features, that are represented by the variables of the soft constraint problems. If each variable has d possible values, the number of variables, say f, of each soft constraint problem is $O(log_d n)$. Summarizing, we have $2f$ features, of which f describe men and f describe women.

In the GS algorithm, men make proposals, starting from their most preferred woman and going down in their ordering, while women receive proposals and compare these against the man to whom they are currently engaged. We will now model the GS operations.

$Opt(pref(m))$ must return the optimal solution of a soft constraint problem defining the preferences of man m over the women. In general, finding the optimal solution of a soft constraint problem is a computationally difficult problem. However, if the soft constraint problem has a tree-like shape, or bounded tree-width, it can be done in polynomial time [6].

$Compare(pref(w), m_1, m_2)$ compares two complete assignments m_1 and m_2 and checks if m_1 is strictly more preferred to m_2. In fuzzy constraint problems, this is computationally easy to do, if the combination operator is polynomially computable and there is a polynomial number of constraints. In fact, m_1 is strictly preferred to m_2 when the preference of m_1 for w is strictly greater than that of m_2 for w. Notice that women need only to perform Compare operations. Thus we do not need any restriction on the shape of the constraint graph for women's preferences to make Compare polynomial.

For the $Next(pref(m), w)$ operation, we need to understand how to linearize the solution ordering of a soft constraint problem. In fact, this operation is used to find the next most preferred woman in a man's preference ordering, so when two or more women are tied, we need to put an order over them to understand who to propose first.

3.1 Linearizations

In fuzzy constraints, the solution ordering is in general a total order with ties: some solutions are equally preferred and a solution dominates another one if its preference value is higher. In this context, linearizing the solution ordering means giving an order over the elements in each tie.

We aim to define linearizations where finding the next best solution (that is, applying operation $Next$) is tractable and where solutions which are less distant from optimal ones appear earlier in the linearized order.

We will define three linearizations L_1, L_2, and L_3 which break ties by taking into account the distance of a solution preference from the optimal preference (L_1), or also the minimum number of preference values for parts of the solutions to be changed to make the solution optimal (L_2), or also the amount of change required (L_3). Among the solutions which are still in ties, we put first those that are lexicographically earlier, according to an ordering over variables and domain values.

We are given an ordering over variables and an ordering over the elements in each variable domain. This allows us to define a lexicographic ordering, say $\prec_{lex}$, over complete or incomplete variable assignments, and also another ordering $\prec_{o_T}$, which orders first partial assignments with higher preference and then orders them lexicographically.

Given a solution s, we define $tuple(s)$ as the first tuple of s according to lex that has preference equal to $pref(s)$. We can now define an ordering $\prec_t$ over solutions: $s_1 \prec_t s_2 \iff tuple(s_1) \prec_{o_T} tuple(s_2)$.

We now define our linearizations:

- $\mathbf{L_1}$: $s_1 \prec_{L_1} s_2$ iff $(opt - pref(s_1)) < (opt - pref(s_2))$, or $(opt - pref(s_1)) = (opt - pref(s_2))$ and $s_1 \prec_t s_2$, or $(opt - pref(s_1)) = (opt - pref(s_2))$, $s_1 =_t s_2$ and $s_1 \prec_{lex} s_2$.

- $\mathbf{L_2}$: $s_1 \prec_{L_2} s_2$ iff $(opt - pref(s_1)) < (opt - pref(s_2))$, or $(opt - pref(s_1)) = (opt - pref(s_2))$ and $t(s_1) < t(s_2)$, or $(opt - pref(s_1)) = (opt - pref(s_2))$ and $t(s_1) = t(s_2)$ and $s_1 \prec_t s_2$, or $(opt - pref(s_1)) = (opt - pref(s_2))$, $t(s_1) = t(s_2)$, $s_1 =_t s_2$, and $s_1 \prec_{lex} s_2$, where $t(s)$ is the minimum number of tuples of s that must be changed to make s optimal. For fuzzy constraints, this is the number of tuples of s with preference value less than opt.

- $\mathbf{L_3}$: $s_1 \prec_{L_3} s_2$ iff $(opt - pref(s_1)) < (opt - pref(s_2))$, or $(opt - pref(s_1)) = (opt - pref(s_2))$ and $ct(s_1) < ct(s_2)$, or $(opt - pref(s_1)) = (opt - pref(s_2))$ and $ct(s_1) = ct(s_2)$ and $s_1 \prec_t s_2$, or $(opt - pref(s_1)) = (opt - pref(s_2))$, $ct(s_1) = ct(s_2)$, $s_1 =_t s_2$, and $s_1 \prec_{lex} s_2$, where $ct(s)$ is $sum_{t_i}(opt - pref(t_i))$, where t_i is any tuple of s with preference less than opt.

We will now see how to perform operations $Next$ on such three linearizations. We will call these operations $Next_i$, for $L_i = 1, 2, 3$.

Next$_1$. From results in [5], we know that performing $Next_1$ can be accomplished by just using the Next operation (as defined in the background section) when we have a tree-like fuzzy CSP, and that this operation takes polynomial time for such problems. In Algorithm 2 we describe procedure $Next_1$ for fuzzy CSPs which is a slightly modified version of the one presented in [5]:

- $next(p)$ is the preference value, among those appearing in P, following p in decreasing order;
- given a fuzzy CSP P and one of its tuples $t = (x_i = v, x_j = w)$, $fix(P, t)$ returns the fuzzy CSP obtained from P by removing from the domains of x_i and x_j all values except v and w;
- given a fuzzy CSP P and a preference p, $cut(P, p)$ returns the CSP obtained from P by zeroing all preferences less than p in all the constraints;
- given a tree shaped CSP P, $cspSolve(P)$ returns the first solution in lexicographic order given the variable and domain orderings;
- given a tree-shaped CSP and one of its solutions s, $cspNext(P, s)$ returns the solution following s in lexicographic order if one exists.

In a fuzzy CSP, a solution has preference p only if it includes a tuple that has preference p. When a solution s is given in input, we look for t_s, the smallest tuple of s w.r.t. ordering o_T that has preference p in the corresponding constraint. This is the tuple that generates solution s. Thus, we fix tuple t_s via $fix(P, t_s)$ and cut the obtained fuzzy CSP at level p. By calling $cspNext$ we look for the solution lexicographically following s. If it doesn't exist, s must be the last solution generated by tuple t_s with preference p.
The next solution may have preference p or lower. If it has preference p, such a preference must come from a tuple with preference p which follows t_s w.r.t. ordering o_T. To avoid finding solutions with preference p that come from tuples preceding t_s we zero out tuple t_s and all tuples with preference p preceding t_S w.r.t ordering o_T. If none of the tuples with preference p following t_s generates a valid solution with preference p, we move down one preference level, restoring all zeroed tuples back to their original values. This search continues until a solution is found or all tuples with preference greater than 0 have been considered.

Next$_2$ and Next$_3$. To perform $Next_2$ and $Next_3$ for tree-shaped fuzzy CSPs, when we already have the top $k-1$ solutions, we find the top k solution according L_2 and L_3 by computing the top k solutions of a set of weighted CSPs.

Our algorithm, which we call $KCheapest$, will work, with suitable variants, for both L_2 and L_3. The input is a tree-shaped fuzzy CSP P, an integer k, and a linearization L (either L_2 or L_3). The output is a set of top k solutions of P according to the linearization. The first step of $KCheapest$ is to look for k optimal solutions of P. As mentioned in background, given a fuzzy CSP P with optimal preference opt, the set of optimal solutions of P coincides with the set of solutions of the CSP $cut(P, opt)$, obtained by allowing only tuples mapped to preferences equal or above opt. Thus, to look for k optimal solutions of P, it is sufficient to compute opt and generate solutions of $cut(P, opt)$. If there are at least k optimal solutions, $KCheapest$ will return them and stop.

Otherwise, we need to consider other, non-optimal solutions, in decreasing order of preference. To do so, we exploit the fact that, in fuzzy CSPs, all solutions with a given preference level, say pl, must have at least a tuple with preference pl. Let us now consider the fuzzy CSP which we obtain from P by fixing a tuple, say t, with preference pl, in a constraint (that is, by forbidding all other tuples in that constraint), and by forbidding all tuples with preference smaller than pl. Such a problem will have either no solution (with a non-zero preference), or all its solutions will have

Algorithm 2: Next1

Input : tree-shaped and DAC Fuzzy CSP P, orderings $o, o_1, \ldots, o_n, o_T$, assignment s with preference p
Output : an assignment s', or "no more solutions"
compute tuple t_s
$t^* = t_s$; $p^* = p$; $P' = cut(fix(P, t^*), p^*)$
if $cspNext(P', s) \neq$ *"no more solutions"* **then**
  return $cspNext(P', s)$
$pref(t) = 0, \forall t \in T$ s.t. $pref(t) = p^*$ and $t \leq_{o_T} t^*$
$cpref = p*$
foreach *tuple* $t >_{o_T} t^*$*with* $pref(t) > 0$ **do**
  if $pref(t) < cpref$ **then**
    reset all preferences previuosly set to 0 to their original values
  if $pref(cspSolve(cut(fix(P, t), pref(t)))) = pref(t)$ **then**
    return $cspSolve(cut(fix(P, t), pref(t)))$
  $cpref = pref(t)$
  $pref(t) = 0$
return *"no more solutions"*

Algorithm 3: KCheapest

1. Find k optimal solutions of P, or all optimal solutions if they are less than k. If the number of solutions found is k, we stop, otherwise let k' be the number of remaining solutions to be found.
2. Look for the remaining top solutions within non-optimal solutions. More in detail, until k' best solutions have been found or all solutions of P have been exhausted, consider each preference pl associated to some tuple in P in decreasing order and, for each tuple t of P with preference pl, perform the following:

1. Compute the new fuzzy CSP, $P_t = fix(P, t)$.
2. If $cspSolve(cut(P_t, pl))$ has no solution, restart the loop with next iteration from **2**.
3. Compute a new soft CSP, say P_t^w, associated to P_t as follows:
   (a) the constraint topology of P_t^w and P_t coincide;
   (b) each tuple with a preference greater than or equal to opt in P_t has weight 0 in P_t^w;
   (c) each tuple with a preference pt s. t. $pl \leq pt < opt$ in P_t has weight c in P_t^w defined as follows: $c = 1$ if $L = L_2$, $c = pt - opt$ if $L = L_3$;
   (d) each tuple with preference less than pl in P_t has weight $+\infty$ in P_t^w.

   Thus, P_t^w is a weighted CSP if $L = L_2$ or $L = L_3$;
4. Compute the k' best solutions or all the solutions if they are less than k' of P_t^w.
5. Take the k' top solutions (or all solutions if less than k') among the sets of best solutions computed for P_t^w, $\forall t$ with $pref(t) = pl$.

preference pl. Let us consider the set of such problems which have at least one solution. It is easy to see that the set of all solutions with preference pl of P coincides with the union of the sets of optimal solutions of such new fuzzy problems. L_2 linearizes solutions with the same preference by counting the number of tuples that have preference lower than opt, and L_3 by weighting such a count with the distance from opt. To account for this we transform each of the fuzzy problems for preference pl, which have solutions, into a weighted problem. The main idea is to make solutions costs in the weighted problem coincide with the component of the L_2 and L_3 which depends on the changes to

be performed to make a solution optimal. Thus, the weighted problems will have the same variables and the same constraint topology as the fuzzy CSPs. All tuples with preference below pl will be forbidden by setting their cost to be $+\infty$, and all tuples with preference equal or above opt will have cost 0. All the remaining tuples, that is, tuples with a non-zero preference smaller than opt, are those that determine the distance to optimality of solutions, according to both L_2 and L_3. If we are linearizing according to L_2, they will be assigned cost 1, while, in the case of L_3, the cost will be the difference between opt and the preference of the tuple. Assuming we are still looking for $k' \leq k$ solutions we must compute the top k' solution for each weighted CSP and then compute the union of such sets. Once we have the union, we pick the top k' solutions of the union. Of course we may be able to obtain less than k' solutions, in which case we proceed as described above for each other preference level in decreasing order.

Theorem 1. *Given a fuzzy CSP P with a tree-shape, computing a set of top k outcomes according to L_2 and L_3 is in $\mathcal{P}$ when k is polynomial in f and d.*

As stated before, from [16] we know that on average only 2% of all proposals are made, so the main idea is to call $KCheapest$ with $k = 2\%$ of the maximum number of proposals and cache the returned set of solutions. Only when all cached solutions have already been returned, we need to call $KCheapest$ again.

4 Complexity issues and stability

Let us now compare the worst case space and time complexity of using the GS algorithm with and without soft constraints. In the classical setting, with n men and n women, and preference lists given explicitly, an SM needs $O(n^2)$ space and $O(n^2)$ time [11]. When we use soft constraints, each man and woman needs $O(d^k m)$ space and time to state their preferences, where d is the size of domains, k is the number of variables of the largest soft constraint, and m is the number of soft constraints. When we use soft constraints with bounded tree-width for the men, each proposal in the GS algorithm takes $O(poly(f))$ time, where f is the number of variables and each Compare operation takes $O(poly(m))$. So, overall, the GS algorithm may need up to $O(n^2 \times poly(f) \times poly(m))$ time, although the number of proposals have been shown to be much lower in practice [16].

If we run the GS algorithm on any of the linearizations we defined, by definition we obtain a matching which is stable w.r.t. this linearization. In SMs with ties [11], a matching is said to be *weakly stable* when there is no man and woman who *strictly prefer* each other to their partner in the matching. Since our linearizations order more pairs than the ordering of the soft CSPs, it is easy to see that any matching which is stable for the linearizations is also weakly stable for the initial orderings.

5 Experimental setting and results

The described linearizations can be used in two different ways within the GS algorithm. One way is to compute the whole preference lists for each man and then run GS over this strict total order as usual. The other is to use the linearization only when GS requires a new proposal. We ran experiments to compare the running time of these two scenarios on a 2,4 Ghz Intel Core i5 machine with 8 GB of RAM, and averaged values over 100 executions, setting a time limit of 10 minutes. For each man a tree-shaped Fuzzy CSP tree over f features is generated by randomizing the preference values from domains of size d. Each woman is represented by a randomly generated Fuzzy CSP with a generic topology over the same number of f features and a constraint density of 50%. Thus, the whole generated SSM problem consists of $n = d^f$ individuals on each side.

In the first test we fixed d, and measured the execution time needed to find a stable matching while increasing the number of features f. Results for $Next_1$ are shown in Fig.1(a), where GS-next1-lists is GS run on precomputed lists obtained running $Next_1$ exactly n^2 times, whereas SoftGS1 is GS which calls $Next_1$ on demand. We note that for $f = 15$ GS-next1-lists didn't complete within the time limit an that, in general, SoftGS substantially outperforms GS-next1-lists both in space and time. This result is not surprising as we know from [16] that GS makes on average only 2% of all possible proposals. This justifies the advantage of an on demand $Next_1$, $Next_2$, and $Next_3$ implementation.

We ran the same tests on $Next_{2-3}$, that is, the algorithm that calls $KCheapest$, only when needed, for either L_2 or L_3. The results are plotted in Fig.1(c) where GS-next23-lists is again the implementation that runs over full precomputed lists while SoftGS23 only precomputes the first 2% of every preference list and then eventually asks for more. SoftGS23 is considerably better than its equivalent that runs on precomputed lists, with an average running time of $4.875s$ versus $107.202s$. In Fig.1(d) we compared the performance of the different linearizations L_1 and L_{2-3}. As expected, solving the Weighted CSP cost minimization problem brings additional overhead to the Next operation, resulting in a performance worsening w.r.t. $Next_1$.

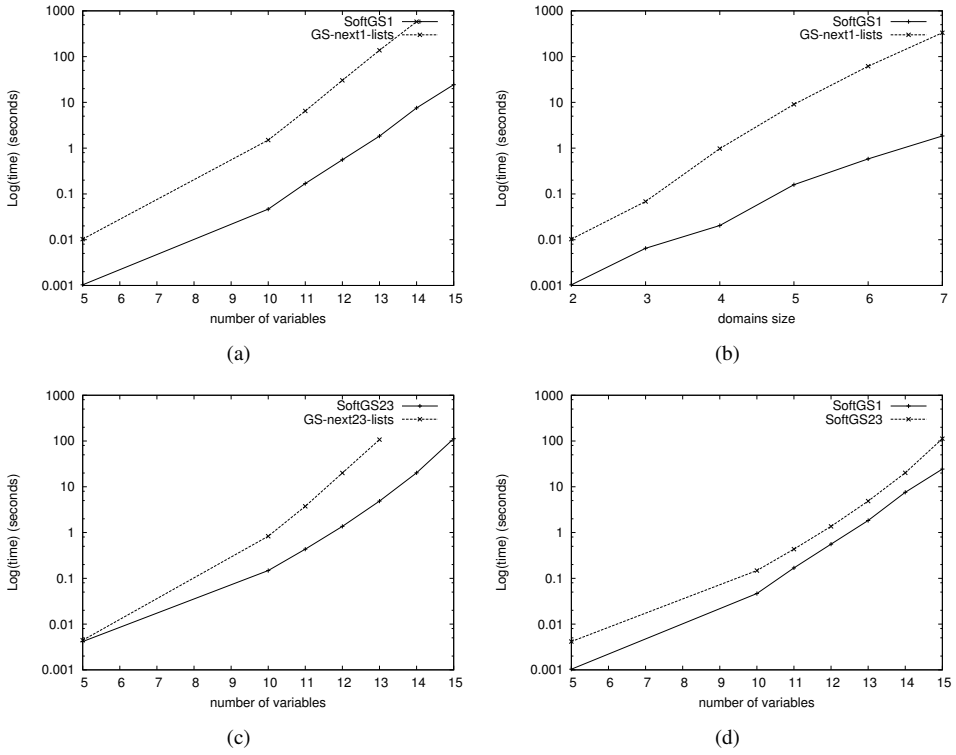

Figure 1: Computation time varying the number of variables with $d = 2$ and varying d with 5 variables ($f = 5$).

In the second test setting, we fixed f, and measured execution time as a function of the domains cardinality d. As shown in Fig.1(b), SoftGS1 clearly behaves better than GS with precomputed lists, despite the fact that $n = d^f$. The performance of SoftGS1 is very interesting as for a setting of $d = 2$ and $f = 12$ (which means 8192 individuals to be matched), the average computation time is $1.83s$ versus $137.43s$ of the version with precomputed lists. In a real world scenario of this size, ranking 8192 individuals of the opposite sex may be impractical, while the compact preference representation makes it feasible, as each individual has just to express his/her preferences over 12

features.

6 Conclusions and future work

We have considered the use of fuzzy constraints in the context of stable marriage problems, to model compactly the preference orderings of men and women. Our experiments suggests that this reduces the amount of time and space needed by each agent to specify its preference ordering, as well as the overall time complexity of the GS algorithm to find a stable matching, if men can model their preferences via soft CSPs with bounded tree-width. The operations we have studied in this context can be useful also in other multi- or single-agent settings, such as in web search, or when an additional solution is looked for. We plan to study stable matching procedures where agents can express their preferences both in a quantitative way via soft constraints and in a qualitative way via CP-nets.

References

[1] S. Bistarelli, S. N. Foley, B. O'Sullivan, and F. Santini. From marriages to coalitions: A soft csp approach. In *Proc. of CSCLP 2008*, pages 1–15, 2009.

[2] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybern*, 11, 1993.

[3] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[4] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JAIR*, 21:135–191, 2004.

[5] R. I. Brafman, F. Rossi, D. Salvagnin, K. B. Venable, and T. Walsh. Finding the next solution in constraint- and preference-based knowledge representation formalisms. In *Proc. KR 2010*, 2010.

[6] R. Dechter. Tractable structures for CSPs. In F. Rossi, P. Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2005.

[7] R. Dechter, N. Flerova, and R. Marinescu. Search algorithms for m best solutions for graphical models. In *Proceedings of AAAI 2012*. AAAI Press, 2012.

[8] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[9] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *Amer. Math. Monthly*, 69:9–14, 1962.

[10] I. P. Gent, R. W. Irving, D. Manlove, P. Prosser, and B. M. Smith. A constraint programming approach to the stable marriage problem. In *Proc. of CP*, pages 225–239, 2001.

[11] D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Boston, Mass., 1989.

[12] D. Manlove. The structure of stable marriage with indifference. *Discrete Applied Mathematics*, 122(1-3):167–181, 2002.

[13] D. Manlove, G. O'Malley, P. Prosser, and C. Unsworth. A constraint programming approach to the hospitals/residents problem. In *Proc. CPAIOR'07*, pages 155–170, 2007.

[14] D. F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific Publishing, 2013.

[15] P. Meseguer, F. Rossi, and T. Schiex. Soft constraints. In F. Rossi, P. Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2005.

[16] E. Pilotto, F. Rossi, K. B. Venable, and T. Walsh. Compact preference representation in stable marriage problems. In *Proc. ADT 2009*, pages 390–401, 2009.

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

[18] C.-P. Teo, J. Sethuraman, and W.-P. Tan. Gale-shapley stable marriage problem revisited: Strategic issues and applications. *Manage. Sci.*, 47(9):1252–1267, 2001.

Pietro De Caro and Maria Silvia Pini
Dep. of Information Engineering
University of Padova
Padova, Italy
Email: pietro.decaro@gmail.com, pini@dei.unipd.it

Francesca Rossi
Dep. of Mathematics
University of Padova
Padova, Italy
Email: frossi@math.unipd.it

K. Brent Venable
Tulane University
Ocala, FL, USA
IHMC
New Orleans, LA, USA
Email: kvenabl@tulane.edu