Compact preference representation in stable marriage problems

E. Pilotto[1], F. Rossi[1], K. B. Venable[1], T. Walsh[2]

[1] Department of Pure and Applied Mathematics, University of Padova, Italy
Email: {epilotto,frossi,kvenable}@math.unipd.it
[2] NICTA and UNSW Sydney, Australia
Email: Toby.Walsh@nicta.com.au

Abstract. The stable marriage problem has many practical applications in two-sided markets like those that assign doctors to hospitals, students to schools, or buyers to vendors. Most algorithms to find stable marriages assume that the participants explicitly expresses a preference ordering. This can be problematic when the number of options is large or has a combinatorial structure. We consider therefore using CP-nets, a compact preference formalism in stable marriage problems. We study the impact of this formalism on the computational complexity of stable marriage procedures, as well as on the properties of the solutions computed by these procedures. We show that it is possible to model preferences compactly without significantly increasing the complexity of stable marriage procedures and whilst maintaining the desirable properties of the matching returned.

1 Introduction

The stable marriage problem is a well-known problem with many practical applications. It is usually defined as the problem of matching men to women so that no man and woman, who are not married to each other, both prefer each other to their current partner [6]. Problems of this kind arise in many real-life situations, such as assigning residents to hospitals, students to schools, as well as in two-sided market trading. A specific application is a web-based stable marriage system for matching sailors to ships in the US Navy.

Surprisingly, a stable matching always exists whatever preferences are held by the men and women. The Gale-Shapley algorithm finds a stable matching in polynomial time [4]. The matching computed is male-optimal since the men have the best possible partners. Since this might be considered unfair to the women, many other stable marriage algorithms have been developed. For example, Gusfield gives a polynomial algorithm to compute the stable matching where the regret of the most unsatisfied person is minimal [5]. We will focus on these two algorithms since they contain the main features of many other stable marriage algorithms.

Both algorithms assume that agents express their preferences (over the members of the other gender) explicitly as a totally ordered list of members of the other gender. In some applications, the number of men and women can be large. It may therefore be unreasonable to assume that each man and woman provides a strict ordering of the other gender. In addition, eliciting their preferences may be a costly and time-consuming process. The sets of men and women may have a combinatorial structure. It could therefore be costly to give preference over all options.

For instance, consider a large set of hospitals offering residencies. Doctors might not want to rank explicitly all the hospitals, but might wish to express preferences over 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”. Based on this information, we can rank the hospitals.

Our challenge is to adapt algorithms to find stable marriages to work with such preference statements. We will investigate whether this changes the computational complexity of stable marriage algorithms like Gale-Shapley's and Gusfield's as well as properties of the matchings computed.

To model preferences compactly, we will use (acyclic) CP-nets [1]. These let agents state their preferences simply and naturally by means of qualitative conditional statements. We will show how to use the preferences orderings induced by CP-nets within GS and Gusfield's algorithms with little additional computational cost. This claim is supported by both theoretical and experimental studies.

2 Background

2.1 CP-nets

CP-nets [1] are a graphical model for compactly representing conditional and qualitative preference relations. CP-nets are sets of *ceteris paribus (cp)* preference statements. For instance, the statement *“I prefer red wine to white wine if meat is served”* asserts that, given two meals that differ *only* in the kind of wine served *and* both containing meat, the meal with red wine is preferable to one with white wine.

A CP-net has a set of features (also called variables) $F = \{x_1, \dots, x_n\}$ with finite domains $\mathcal{D}(x_1), \dots, \mathcal{D}(x_n)$. For each feature x_i, we are given a set of *parent* features $Pa(x_i)$ that can affect the preferences over the values of x_i. This defines a *dependency graph* in which each node x_i has $Pa(x_i)$ as its immediate predecessors. Given this structural information, the agent explicitly specifies her preference over the values of x_i for *each complete assignment* on $Pa(x_i)$. This is by means of a total order over $\mathcal{D}(x_i)$. An *acyclic* CP-net is one in which the dependency graph is acyclic.

Consider a CP-net whose features are A, B, C, and D, with binary domains containing f and $\overline{f}$ if F is the name of the feature, and with the following preference statements: $a \succ \overline{a}$, $b \succ \overline{b}$, $(a \wedge b) \vee (\overline{a} \wedge \overline{b}) : c \succ \overline{c}$, $(a \wedge \overline{b}) \vee (\overline{a} \wedge b) : \overline{c} \succ c$, $c : d \succ \overline{d}$, $\overline{c} : \overline{d} \succ d$. Here, $a \succ \overline{a}$ represents the unconditional preference for $A = a$ over $A = \overline{a}$, while $c : d \succ \overline{d}$ states that $D = d$ is preferred to $D = \overline{d}$ given that $C = c$.

The semantics of CP-nets depends on the notion of a *worsening flip*. This is a change in the value of a feature to a less preferred value according to the preference statement for that feature. For example, in the CP-net above, passing from $abcd$ to $ab\overline{c}d$ is a worsening flip since c is better than $\overline{c}$ given a and b.

A solution (also called outcome) of a CP-net is an assignment to all its variables of values from their domains. One solution α is *better* than another solution β (written

$\alpha \succ \beta$) iff there is a chain of worsening flips from α to β. This definition induces in general a preorder over the solutions. If the CP-net is acyclic, the solution ordering is a partial order with only one top element.

In general, finding the optimal solution of a CP-net is NP-hard. However, in acyclic CP-nets, the unique optimal solution can be found in linear time. We simply sweep through the dependency graph assigning each variable to the its most preferred value. For instance, in the CP-net above, we would choose $A = a$ and $B = b$, then $C = c$, and then $D = d$.

Determining if one solution is better than another (called a dominance query) is NP-hard even for acyclic CP-nets. Whilst tractable special cases exist, there are also acyclic CP-nets in which there are exponentially long chains of worsening flips between two solutions.

2.2 Stable marriage problems

The *stable marriage problem* (SMP) is the problem of finding a matching between the elements of two sets. Usually, the members of the two sets are called men and women. More precisely, 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*. We will write $pref(x)$ for the preference ordering of man or woman x.

The *Gale-Shapley algorithm* (GS) [4] is a well-known algorithm to solve the SMP problem:

Algorithm 1: GS

```
Set all men and women as free
while there is a free man $m$ do
    $w \leftarrow$ the first woman in $pref(m)$ to which he has not yet proposed
    if $w$ is free then
        match $m$ with $w$
    if $m >_{pref(w)} z$, where $z$ is $w$'s current partner then
        match $m$ with $w$ and set $z$ free
    else
        $w$ rejects $m$ and $m$ remains free
```

This algorithm consists of a number of rounds in which each un-engaged man proposes to the 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. The algorithm takes $O(n^2)$ steps and construct a matching that is *male-optimal*, since every man is paired with his highest ranked feasible partner, and *female-pessimal*, since each woman is paired with her lowest ranked feasible partner.

Consider $n = 3$. Let $W = \{w_1, w_2, w_3\}$ and $M = \{m_1, m_2, m_3\}$ be respectively the set of women and men. The following sequence of strict total orders defines an SMP:

– $m_1 : w_1 > w_2 > w_3$ (i.e., man m_1 prefers woman w_1 to w_2 to w_3); $m_2 : w_2 > w_1 > w_3$; $m_3 : w_3 > w_2 > w_1$
– $w_1 : m_1 > m_2 > m_3$; $w_2 : m_3 > m_1 > m_2$; $w_3 : m_2 > m_1 > m_3$

For this SMP, the Gale-Shapley algorithm returns the male-optimal marriage $\{(m_1, w_1)$, (m_2, w_2), $(m_3, w_3)\}$. On the other hand, the female-optimal marriage is $\{(w_1, m_1)$, (w_2, m_3), $(w_3, m_2)\}$.

Male-optimality might be considered unfair to the women. Other proposal-based algorithms to compute stable matchings have been proposed that might be considered fairer. For example, Gusfield gives an algorithm to compute the minimum-regret stable matching [5]. This is the best stable matching as measured by the person who has the largest regret in it. The regret of a man in a matching is the number of women that are more preferred than its current partner. The regret of a woman is defined analogously. Gusfield's algorithm passes from one matching to another. In each step, the person with the maximum regret is identified, and their current marriage is broken to pass to another matching with a smaller maximum regret.

3 Operations Opt, Next, and Compare in the SMP algorithms

In algorithms such as GS and Gusfield's, men make proposals, starting from their most preferred woman and going down in their ordering, whilst women receive proposals and compare these against the men to whom they are currently engaged. Moreover, in both algorithms, proposals are made in increasing order of regret. This is especially exploited by Gusfield's algorithm, where the notion of regret is also used to decide how to modify the current matching in order to obtain one with a smaller regret. Three operations are thus needed by both algorithms:

– $Opt(pref(m))$: Given a man m, we compute his optimal woman. This is needed the first time a man makes a proposal.
– $Next(pref(m), w)$: Given a man m and a woman w, we compute the next best woman for m. This is needed when a man makes a new proposal.
– $Compare(pref(w), m_1, m_2)$: Given a woman w and two men m_1 and m_2, we decide if m_2 is preferred to m_1 for w. This is needed when a woman compares two proposals to decide whether to remain with the current man (m_1) or to leave him for a new man who is proposing (m_2).

Operations Opt and $Next$ return a woman, while $Compare$ returns a Boolean value.

If preferences are given explicitly as strict total orders, as in the traditional SMP setting, these operations all take constant time. However, if preferences are represented with a compact representation language such as CP-nets, then this is not the case. Thus, to understand the impact of using a compact preference formalisms within algorithms like GS, we consider the computational complexity of these operations on CP-nets.

4 Opt, Next, and Compare on CP-nets

We consider a stable marriage problem with n men and women, where each man and each woman specifies their preferences over the other sex via a CP-net. We call this a Compact SMP (CSMP). $pref(m)$ is now the solution ordering induced by a CP-net, which can be a partial ordering.

For simplicity, we will consider CP-nets with two values in each domain. However, the results can be easily generalized to non-binary domains. Each man and woman is described by a set of Boolean features, and n is the size of the Cartesian product of such domains. Thus the number of features f of each CP-net is $log(n)$. Conversely, if we are given a set of f features, we assume that each assignment to such features corresponds to a man (resp. a woman). We could, however, relax this assumption by using a constrained CP-net to rule out infeasible combinations.

Let us consider the three operations used within the SMP algorithms. In general, finding the optimal solution of a CP-net is a computationally difficult problem, as is dominance testing (the problem of comparing two solutions in the CP-net ordering) [1]. We are not aware of any study of the complexity of finding the next best solution. However, as two of the three operations are computationally intractable in general, and as we wish to find settings where such operations take just polynomial time, we turn our attention to acyclic CP-nets. These are more restrictive but may be sufficiently expressive in many contexts [1].

As mentioned before, in acyclic CP-nets there is always one optimal solution, and it can be found in linear time in the number of features f by a simple forward sweep algorithm. Operation Opt thus takes $O(f)$ time.

While the solution ordering of an acyclic CP-net may be partial, operations $Next$ and $Compare$ need a total order, since $Next$ returns one new proposal to be made, and $Compare$ chooses between two proposals. Therefore, we will consider linearizations of the CP-net solution ordering.

This does not contradict a user's preference statements, since a linearization only orders pairs of elements that were incomparable. Notice also that, even if we could work with partial orders, dominance testing (and thus $Compare(pref(w), m_1, m_2)$) is intractable in general for an acyclic CP-net. Here, on the other hand, we aim to find linearizations where all three operations are tractable.

We will focus on those linearizations where the regret is larger as we descend the order. As noted before, our stable marriage algorithms make proposals in this order.

5 A linearization of the CP-net solution ordering

Linearizations of the solution ordering of acyclic CP-nets have been considered in [2, 3]. A consequence of these results is that, given a feature order which is compatible with their topological order in the dependency graph, any lexicographical ordering over the solutions is a linearization of the original partial ordering. Computing Next in such a linearization is polynomial since it simply requires the next tuple of feature values in the lexicographic ordering. Also the Compare operation is polynomial since it reduces to a comparison of tuples over the lexicographical relation. Unfortunately, such linearizations do not in general satisfy the regret condition. We therefore consider a different linearization, where the Next and Compare operations are polynomial, and where solutions closer to the top of the partial order (that is, with a smaller regret) come first.

We recall that the regret of a man is the distance between his partner in the current marriage and his most preferred woman. This notion has been originally defined over total orders [6]. However, it can be generalized to be used on partial orders. More precisely, the regret of a man m when married to a woman w is the longest path, in the preference ordering of m, between w and the top element of his ordering.

For example, let us consider the CP-net, as well as its induced solution ordering, shown in Figure 1 (where $\bar{x}$ is written as $-x$ for all values x). This CP-net has three features A, B, and C, where B depends on A. The regret of $\bar{a}bc$ is 3 since there are at most 2 solutions between the top and this one.

While $pref(m)$ is the preference ordering induced by the CP-net of m, we will call ***lex-pref(m)*** our linearization of $pref(m)$.

This linearization is based on a lexicographical order over feature levels. Given an acyclic CP-net, we divide its features into levels, each containing all the features that have the same longest path length to a feature without outgoing edges in the dependency graph. For example, in the CP-net of Figure 1, we have two levels: level 2, containing only A and corresponding to a longest path of length 1, and level 1, containing B and C, corresponding to a longest path of length 0.

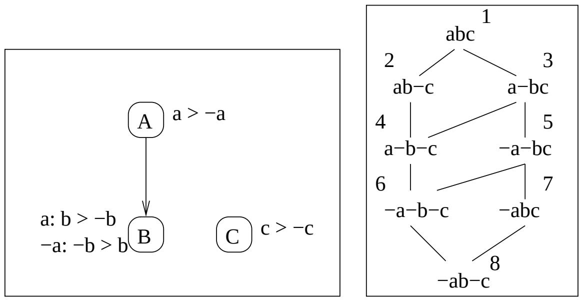

Fig. 1. A CP-net and its induced solution ordering.

Given a solution, we then associate to it a vector v of length equal to the number of levels, say k, whose elements $v_1, \ldots, v_k$, corresponding to levels k to 1, are Boolean vectors of length equal to the number of features in each level. Features are ordered within each level in some fixed order.

For the previous example, we have a vector with two elements (since we have two feature levels), where the first one has one Boolean value (corresponding to the value for A), and the second one is a two-element Boolean vector (corresponding to the values for B and C). The Boolean values in the vectors are set according to the values of the features: given the values of the parents, if the value of the considered variable is the most preferred, we put 0, otherwise 1.

Consider again the CP-net in Figure 1. The solution $a\bar{b}\bar{c}$ gives the vector $[0, 11]$, since a is the most preferred value in the CP-table of feature A, while, given $A = a$, both $\bar{b}$ and $\bar{c}$ are the least preferred values in the CP-tables of features B and C.

Given any two such vectors, say $v = [v_1, \ldots, v_k]$ and $v' = [v'_1, \ldots, v'_k]$, we now define how to order them. Let us denote by $sum(x)$ the sum of all elements in vector x. Then $v <_{lex-pref} v'$ (that is, v precedes v' in the ordering and is more preferred) iff $[sum(v_1), \ldots, sum(v_k)]$ is lexicographically smaller than $[sum(v'_1), \ldots, sum(v'_k)]$, or $[sum(v_1), \ldots, sum(v_k)] = [sum(v'_1), \ldots, sum(v'_k)]$ and $[v_1, \ldots, v_k]$ is lexicographically smaller than $[v'_1, \ldots, v'_k]$.

For example, vector $[00, 01]$ is preferred to vector $[00, 10]$, since the vectors of the sums are equal ($[0, 1]$) and $[00, 01]$ is lexicographically smaller than $[00, 10]$. Also, $[00, 01]$ is preferred to $[00, 11]$, and $[10, 00]$ is preferred to $[01, 01]$.

In Figure 1, the linearization of the ordering is shown via the numbers above each solution. As the following theorem shows, this is a linearization of the solution ordering induced by the CP-net.

Theorem 1. *Given a CP-net and two solutions s and s', with associated vectors v and v', if $s \succ s'$, then $v <_{lex-pref} v'$.*

Proof. The vectors of the sums represents the number of violations in each level of the CP-net. If s' has a larger number of violations in higher levels of the CP-net with respect to s (thus s' is less preferred than s), the vector of the sums of v' will be larger than the one of v. Notice also that solutions which are incomparable in the CP-net ordering are strictly ordered in our linearization.$\square$

6 Complexity of Opt, Next, and Compare, and regret condition

Since we have linearized a partial order with one top element, $Opt(pref(m)) = Opt($*lex-pref(m)*$)$. Therefore, finding $Opt($*lex-pref(m)*$)$ is polynomial since we can use the sweep forward algorithm to find $Opt(pref(m))$ in acyclic CP-nets.

We will now consider the complexity of the operations $Next$ and $Compare$ on this linearization.

For operation *Compare(lex-pref(w),*m_1,m_2), we can directly use the definition of $<_{lex-pref}$ given above: *Compare(lex-pref(w),*m_1,m_2) $= true$ iff $v(m_2) <_{lex-pref} v(m_2)$, where $v(m_i)$ is the vector associated to m_i in the CP-net $pref(w)$.

Theorem 2. *Given a CSMP of size n, a woman w and two men m_1 and m_2, $Compare($ lex-pref $(w), m_1, m_2)$ can be computed in $O(f)$, where f is the number of features of the CP-net of w.*

Proof. The vectors associated to each of the two men can be computed in linear time in the number of features of the CP-net: for each feature, we check the position of the feature value in a row of the appropriate CP-table. Given the two vectors, we need to compute the vectors of their sums, which is linear in the number of features, and to compare them (and possibly also the original vectors) lexicographically. This takes time linear in the number of features. □

For operation $Next(\textit{lex-pref}(m), w)$, given a vector, we need to find the next vector in the $<_{lex-pref}$ ordering. This can be done by following a procedure similar to incrementing a Boolean counter.

Given a vector $v = [v_1, \dots, v_k]$, we build a new vector v' as follows. For j from k to 1, we compute the first j such that $sum(v_j) = sum(\textit{next-lex}(v_j))$, where *next-lex* is just the next element in a standard lexicographical order. If such a j exists, then $v' = [v_1, \dots, \textit{next-lex}(v_j), v'_{j+1}, \dots, v'_k]$, where v'_h with $h > j$ is the minimum vector with sum equal to $sum(v_h)$. Otherwise, we compute $\textit{next-lex}([sum(v_1), \dots, sum(v_k)])$, and set v' as the smallest vector with these sums.

For example, consider solution $a\bar{b}\bar{c}$ for the CP-net in Figure 1. The corresponding vector is $[0, 11]$. There is no way to increase lexicographically either 11 or 0 while maintaining the same sums, so we must consider the sum vector $[0, 2]$, increment it to $[1, 0]$, and then build vector $[1, 00]$, which corresponds to solution $\bar{a}bc$. If instead we consider solution $ab\bar{c}$, whose vector is $[0, 01]$, we can modify 01 into 10 while maintaining the same sums, so we get vector $[0, 10]$, corresponding to solution $a\bar{b}c$.

Algorithm 2: $Next$

```
Input: acyclic CP-net $N$ of man $m$, vector $v = [v_1, \dots, v_k]$,
Output: vector $v'$, successor of $v$ in $<_{lex-pref(m)}$
$v' \leftarrow v$
$i \leftarrow k$
while $i > 1$ and $v'_i = LEX\_NEXT(v'_i)$ do
    $i \leftarrow i - 1$
if $i \geq 1$ then
    $v'[i] \leftarrow LEX\_NEXT(v'_i)$
    $RESET\_ALL\_SUCC(v', i)$
    return $v'$
else
    $P \leftarrow SUM\_NEXT([sum(v'_1), ..., sum(v'_k)])$
    if $P = nil$ then
        return "No more solutions"
    return $SUM\_MIN(P)$
```

In Algorithm 2, procedure `LEX_NEXT` takes as input a vector v'_i and returns the lexicographical successor of v'_i that has the same sum, if it exists, and v'_i itself otherwise. Procedure `RESET_ALL_SUCC`, given a vector v' and an index i, resets the sub-vector with components v'_j, with index $j \geq i$, to the minimal lexicographic Boolean vector with sum equal to $sum(v'_j), j \geq i$. Procedure `SUM_NEXT` computes the lexicographic successor of a vector of sums taking into account the maximum cost sum of each level. If no such successor exists, it returns `nil`. Finally, procedure `SUM_MIN` computes the lexicographical minimal vector having the sum vector given in input.

Theorem 3. *Given a CSMP of size n, a woman w and a man m with a CP-net with f features, Algorithm 2 computes $Next($ lex-pref $(m), w)$ in $O(f)$ time.*

Proof. Computing the next vector in a lexicographical order, computing the sum of the vector, and computing the minimum vector with the same sum are all tasks that can be done in time linear in the size of the vector. Thus the above algorithms can run in time linear in the size of the vector associated to woman w, which is the number of features of the CP-net of man m. □

We have shown that all the three operations (Opt, Next, and Compare) can be computed in polynomial time on our linearization of the CP-net ordering. We now show that this linearization also has the property that later elements have a larger regret.

Theorem 4. *Given a CP-net of a man m, and two women w_1 and w_2, if $s' <_{lex-pref} s$, then the regret of m when married with w_1 is smaller than or equal to his regret when married to w_2.*

7 Properties of the generated stable marriage: stability, male-optimality, and minimum regret

7.1 Stability

If we run the GS algorithm on the linearization just defined, by definition we obtain a matching which is stable w.r.t. this linearization. However, one may wonder if the generated matching is stable w.r.t. the partial order of the CP-net.

As is standard in SMPs with ties [6], also in our case, where the orders induced by the CP-nets may be partial, we define a matching to be stable when there is no man and woman who *strictly prefer* each other to their partner in the matching.

Since our linearization orders more pairs than the partial order of the CP-net, it is easy to see that any matching which is stable for the linearization is also stable for the partial order. In fact, if there is a blocking pair (that is, a man and a woman who would prefer to be together rather than with their current partners) in the partial order, such a pair will be blocking also according to the linearization. Therefore stability is assured, both w.r.t. the linearization and w.r.t. the original CP-net solution ordering.

7.2 Male-optimality

The matching found by the GS algorithm on the linearization will of course be male-optimal w.r.t. the linearization. However, it may be not male-optimal w.r.t. the original partial order.

In the presence of partial orders, the definition of male-optimality is the same as for total orders: a stable marriage is male-optimal if, for each man m and partner w, there is no stable marriage in which m is married to another woman w' which he strictly prefers to w. Notice that, while a male-optimal matching always exists when we work with totally ordered preferences, this is not true when we have partial orders.

Even if a male-optimal matching exists, running the GS algorithm on our linearization can return a matching which is not male-optimal. Consider the following SMP with

2 men and 2 women, where $\bowtie$ means incomparability: $m_1 : w_1 \bowtie w_2$; $m_2 : w_1 \prec w_2$; $w_1 : m_1 \bowtie m_2$; $w_2 : m_1 \bowtie m_2$. Given the linearization where we order incomparable element in incresing index order, the stable matching obtained by GS on this linearization is $((m_1, w_1), (m_2, w_2))$. However, the only male-optimal matching in the original problem is $((m_1, w_2), (m_2, w_1))$.

Notice that, when preferences are totally ordered, the output of the GS algorithm is not affected by the order in which men propose to women. With partial orders, this order may affect the result, since women leave the current partner only if the new proposal is strictly better. For example, in ther SMP above, if man m_2 proposes first (to w_1), then the resulting marriage is the male-optimal one ($((m_1, w_2), (m_2, w_1))$).

There is a way to ”do our best” w.r.t. male-optimality, by following the policy that the next men making a proposal is one of those, if any, with a single next best woman. If no such man exists, then we can choose any man. If we do this, then it is possible to show that the generated marriage is never worse w.r.t. male-optimality than the one obtained by any other policy.

To (partially) implement this policy, we can exploit the fact that only incomparable women can have the same vectors of sums (see Theorem 1). Therefore, we can identify a man with a single next best woman by executing twice the Next operation and by comparing the vectors of the sums of the two women obtained. If they are the same, then the two women are incomparable (thus the man does not have a single next best woman). On the other hand, if they are different, the two women may be ordered or incomparable in the partial order. Thus we should choose a man with two different vectors of the sums. Notice that this is an approximation of the desired policy, since some incomparable women may have different vectors of the sums. Notice also that the implementation of the proposal policy does not add to the worst-case cost of the GS algorithm, since the result of the second Next operation can be saved for future use.

7.3 Minimum regret

Computing a stable matching which minimizes the regret of the person who is worst-off may be perceived to be fairer than computing the male-optimal matching. As mentioned in Section 2, such a stable matching is found by Gusfield's algorithm. As with GS, our linearization allows us to use Gusfield's algorithm with compact preference formalisms.

Theorem 5. *Running Gusfield's algorithm on a CSMP where each CP-net has f features, with Opt to find the first proposal, $Next$ to find the next proposal, $Compare$ to compare two proposals, and r to compute the regret of a matching, we obtain a stable matching with minimum regret in $O(n^2 f)$ time.*

Proof. Our linearization respects the ordering induced by the regret and allows to compute the regret efficiently. Notice that this is important in Gusfield's algorithm, since regret is not only used to establish a proposal order but also to identify the person who is worst-off in the current matching.

Given that Gusfield's algorithm runs in $O(n^2)$ time, and considering the complexity of Opt, $Next$, and $Compare$ and that of computing r, it is easy to see that our version of the algorithm runs in $O(n^2 f)$ time. $\square$

8 Experimental analysis

Given the linearization described above, we can either pre-compute it and then run GS as usual over a strict linear order, or we can just compute the part of the linearization that GS needs during the execution of the algorithm. In the first scenario, we need to compute the $Next$ operation n^2 times (n times for each man and woman), and then GS can run in the usual $O(n^2)$ time. In terms of space, however, we need to store all the n linearizations, which takes $O(n^2)$ space. In the second scenario, there is no pre-computation burden, but each step of GS requires additional time to perform a $Next$ (and possibly a $Compare$) operation. Given the theoretical complexity results above, GS will run in $O(n^2 log(n))$ time. The space needed is just to store the CP-nets, and not the linearization, which is now $O(nlog(n))$.

We ran some experiments to see which of these two scenarios is more effective. Given a number of features f, we randomly generate acyclic CP-nets with f Boolean features where each feature has at most two parents. For each feature, we then generate a CP-table by making sure that the dependency graph is respected. To generate a whole CSMP with $n = 2^f$ men and women, we generate $2n$ CP-nets with f features each. The experiments have been performed on an Intel Core Duo 3GHz processor with 4GB RAM, and show the average over 100 instances.

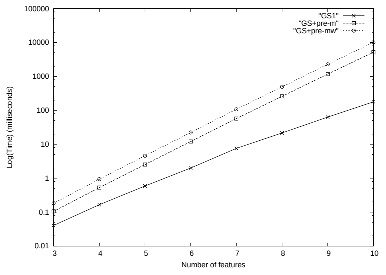

Fig. 2. Execution time for three versions of GS.

Figure 2 gives the log-scale time needed to run three different versions of the GS algorithm: the one with the Next and Compare operation executed on demand (GS1), the one with the pre-computation of the linearization for the men, and the Compare operation executed on demand (GS+pre-m), and the one with the pre-computation of all the linearizations (GS+pre-mw). It is easy to see that it is inefficient to pre-compute the linearizations, even for just the men.

This is perhaps not too surprising, since computing the linearizations needs to run the Next operation exactly n^2 (or $2n^2$) times, while the GS algorithm needs $O(n^2)$ time in the worst case but may in practice require only a much smaller number of proposals. This is confirmed by Figure 3, where we plot the number of proposals made by the GS1 algorithm as a function of the number of features in the CP-nets. The GS algorithm usually makes only a small number of proposals. For example, for 10 features, we have $n = 2^{10} = 1024$, thus $n^2 = 1,048,576$, but the GS algorithm makes less than 16,000 proposals on average.

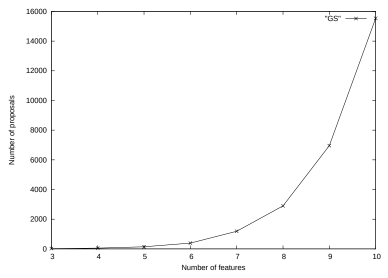

Fig. 3. Number of proposals made by the GS1 algorithm.

Notice that algorithm GS1 takes very little time even for CP-nets with 10 features (less than 1 second). This scenario is realistic, since it models problems with about a thousand members of each sex. In this setting, it might be impractical to ask each agent to rank all members of the other sex, whilst it is more practical to specify a CP-net over just 10 features.

9 Conclusions and future work

We have considered using a qualitative compact preference representation, namely CP-nets, in the context of stable marriage problems. We have shown that the benefits brought by compactness do not impact greatly on the complexity of computing stable matchings nor on the properties of the returned matching.

The significance of our study on the complexity of the Next operation on CP-nets goes beyond its use in SMPs. In fact, such an operation is also needed when computing the top k solutions, such as in web search, or when an additional solution is looked for.

In the future, we plan to investigate the use of other compact approaches to preferences such as soft constraints, as well as to consider other versions of the stable marriage problem (such as with ties). We also plan to see whether tractable cases exists for operations like Next on classes of soft constraints or constrained CP-nets.

References

1. 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. *J. Artif. Intell. Res. (JAIR)*, 21:135–191, 2004.
2. Ronen I. Brafman and Yuri Chernyavsky. Planning with goal preferences and constraints. In *ICAPS*, pages 182–191, 2005.
3. Ronen I. Brafman, Carmel Domshlak, and Tanya Kogan. Compact value-function representations for qualitative preferences. In *UAI*, pages 51–59. AUAI Press, 2004.
4. D. Gale and L. S. Shapley. College admissions and the stability of marriage. *Amer. Math. Monthly*, 69, 1962.
5. D. Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM Journal of Computing*, 16(1), 1987.
6. D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.