Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria dell'Informazione

A LOCAL SEARCH ALGORITHM FOR MATCHING HOSPITALS TO RESIDENTS

Laureando:

Matteo Sartori

Relatore:

Dott.ssa Maria Silvia Pini

22 Novembre 2013

Anno Accademico 2013/2014

Abstract

Stable matching problems are well known to have many practical applications, such as assigning doctors to hospitals or students to schools. In this thesis we consider a variant of the classical stable matching problem, called Hospital Resident matching problem (HRT). We provide a local search algorithm to solve it and we evaluate it on instances with sizes comparable to the instances of real-life applications.

Contents

List of Figures

Chapter 1

Introduction

Stable matching problems are well known problems which can be found in a wide variety of large-scale practical applications. In a stable matching problem we seek to assign a set of agents to another set of agents, typically under certain constraints involving preference lists and capacities. The preference list of an agent contains the agents of the other set, listed in order of preference. Stabilty of the returned matching is an essential property for practical involvement, and ensures that there are no two agents of the two different sets, that are not matched to each other, but they both prefer each other to their current partner. Examples of real-world instances of stable matching problems include assigning medical students to hospitals, children to schools and kidney transplant patients to donors [8].

In 1962, Gale and Shapley [1] formally define the *Stable Marriage* (SM) problem and they give a polynomial procedure to find a stable matching when the cardinality of the two set of agents is the same. In this scheme, the algorithm guarantees that everyone become engaged and that the possible returned marriage is stable.

Actually, the classic SM is quite restrictive, indeed it requires that every agent expresses a complete strict preference ordering over all the members of the other set, however, this is not what is more likely to be found in practise. For example, it's quite common that an agent find acceptable only some of the other agents, or he isn't able to decide between two of them since they are very similar. This generalization of SM is known as SMTI because it is a stable marriage problem which involves *incomplete* preference lists with *ties*. However, more general preference lists involve the possibility that there exist stable marriages where some agents are not matched and there may be matchings of different sizes which are both stable. These situations introduces new problems, such as determining if an instance I has a stable matching of

size bigger than a certain value K or determining if I has a stable matching where all agents are matched. We know from the literature that these two problems are NP-hard [7].

Another possible extension of the classical SM problem is to allow that an agent can be matched to more agents simultaneously. This is what happens to the possibly best known real world implementation of SM, the National Resident Matching Program (NRMP), in the United States, which represents a many-to-one generalization of a SMTI. In this case we talk of residents, which express a preference list of hospitals they wish to get assigned to, and a set of hospitals that consider each proposing resident in a preference list that can be incomplete and with ties. We call this *hospital-residents* matching problem, HRT.

In this thesis our goal is to solve the Hospitals-Residents problem with an approximate solution, by exploiting a Local Search approach [10]. We find in the literature another work [2] which uses this technology on SM and SMTI problems: it starts from a randomly chosen matching and, at each step, it selects a matching in the neighborhood by minimizing the distance from stability, that is, minimizing the number of unstable pairs. We show how to adapt this method to handle HRT.

We have evaluated this method on big artificial instances, that are comparable with practical ones, which involve thousands of agents. Experimental results show that our algorithm is able to return a solution in few seconds and with a high quality in terms of the size of the returned matching. Moreover, it is comparable with ad hoc powerful methods.

The dissertation is organized as follows. In Chapter 2 we provide the basic notions of stable matching problems and hospitals-residents problems. In Chapter 3 we describe new local search algorithms for the HRT problem and in Chapter 4 we evaluate experimentally this approach. Finally in Chapter 5 we summarize the results of the thesis and we provide some hints for future work.

Chapter 2

Background

In Section 2.1 we present the classical *Stable Matching* problem, also called *Stable Marriage* problem, the first matching problem to be formally studied in the literature. We come up with the main subject of this thesis in Section 2.2, a many-to-one generalization of the Stable Marriage problem known as the *Hospitals Residents* problem where some form of imprecision and incompleteness is allowed for modelling agents preferences. This introduces new concepts like *unacceptability* and *indifference*, which entail important consequences about computational complexity.

1 Stable matching problems

Following examples and definitions are from [9]. The stable matching problem is a well known problem with many practical applications. It has to do with two set of agents, often called men and women to avoid focussing on a particular application. An instance I of SM involves n men and n women, each of whom ranks all n members of the opposite gender in strict order of preference. In I we denote the set of men by $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ and the set of woman by $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$. We also express a preference list with this notation

$$r_i : \quad w_{i_1}\ w_{i_2}\ \ldots\ w_{i_n}$$

where the i-th man orders the set of all woman from w_{i_1}, the most preferred, to w_{i_n} the least preferred.

Since the main problem is to couple every man to every woman we have to find a *matching* M , that is a bijection from $\mathcal{M}$ and $\mathcal{W}$. If $(m, w) \in M$, we say that a man m is *matched to* a woman w in M, and also w is *matched to*

m for a symmetry property of this bijection. Also we say that, if $(m, w) \in M$ m and w are *partners* in M. More general, we call *assignment* A the set of pairs $(m, w) \in \mathcal{M} \times \mathcal{W}$, of course an assignment need not to be a matching. Let $A(p)$ denote a p's partner in A, where $p \in \mathcal{M} \cup \mathcal{W}$. If $A(p) \neq \emptyset$, then we say that p is *assigned* in A, otherwise p is *unassigned* in A.

Definition 1. *Given a marriage M, a pair (m, w), where m is a man and w is a woman, is a* `blocking pair` *iff m and w are not partners in M, but m prefers w to $M(m)$ and w prefers m to $M(w)$.*

Definition 2. *A marriage M is* `stable` *iff it has no blocking pairs.*

We can intuitively see stability as a property ensuring that no agent is allowed to make a request in order to change its partner, because there is no incentive for any one agent to improve. And this give us an indication that the match we are considering is a good result, otherwise it would be merely useless.

Another important practical consideration follows. It could happen that, even if a matching M is stable, every man obtains his best possible partner and, consequently, each woman obtains her worst possible partner. In the literature this situation is referred to as *man optimal marriage* (the analogous for women is *woman optimal marraige*) and, for certain applications this behaviour is unacceptable, even for stable matchings. In this case we have to consider ad hoc strategies that build stable marriages, paying attention to give balanced consideration of both genders.
In Figure 2.1 we show an example of an SM instance I and we consider this two subsequent matchings

$$M_1 = \{(m_1, w_3), (m_2, w_1), (m_3, w_2)\}$$

$$M_2 = \{(m_1, w_1), (m_2, w_3), (m_3, w_2)\}$$

we see that (m_1, w_1) is a blocking pair for matching M_1, as both m_1 prefers w_1 to $M_1(m_1) = w_3$ and w_1 prefers m_1 to $M_1(w_1) = m_2$. Instead, M_2 is one of all possible stable marriages for the instance I.

Men's preferences		Women's preferences	
m_1 :	$w_1\ w_3\ w_2$	w_1 :	$m_1\ m_3\ m_2$
m_2 :	$w_1\ w_2\ w_3$	w_2 :	$m_3\ m_1\ m_2$
m_3 :	$w_2\ w_1\ w_3$	w_1 :	$m_3\ m_2\ m_3$

Figure 2.1: Instance I of SM

2 Hospital-residents problem

We now introduce a many-to-one generalisation of SM called *Hospitals Residents* problem(HRT). In HRT, each hospital has one or more posts that it requires to fill, and a preference list ranking a subset of the residents. Similarly, each resident has a preference list ranking a subset of the hospitals. In fact, we permit, in the case of an hospital, that it may find a resident *unacceptable*, and also that it may express *indifference* between two of them. This reflects in having incomplete lists with ties. Obviously, we can also talk of residents that do not accept hospitals or that they are indifferent between some of them. HRTs are a generalization of the stable matching problems with ties and incomplete lists (SMTIs), that are SMs where we allow indifference in the preference ordering and truncated list of preferences [7].

The capacity of a hospital is its number of available posts. We want to match each resident to at most one hospital such that no hospital exceeds its capacity, whilst observing a stability criterion to be defined. An instance of HRT is defined formally as

- set of residents $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$
- set of hospitals $\mathcal{H} = \{h_1, h_2, \ldots, h_s\}$
- preference list for all $r_i \in \mathcal{R}$, each of whom ranks a subset of $\mathcal{H}$.
- preference list for all $h_j \in \mathcal{H}$, each of whom ranks its *applicants*, the residents who find that particular hospital acceptable.
- list of capacities c_j $(1 \leq j \leq s)$ for each hospital.

We say that a resident r_i finds a hospital h_j *acceptable* if r_i's preference list contains h_j, and vice versa. An *assignment* M for an instance I of HR is a set of pairs $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ such that $(r_i, h_j) \in M$ only if r_i and h_j find each other acceptable. If $(r_i, h_j) \in M$ we say that r_i is *assigned* to h_j, and h_j is *assigned* r_i. For any $p \in \mathcal{R} \cup \mathcal{H}$, we denote by $M(p)$ the set of assignees of p in M. If $M(p) \neq \emptyset$ we say that p is *assigned* in M, otherwise r is *unassigned* in M. We say that a hospital $h_j \in \mathcal{H}$ is *under-subscribed*, *over-subscribed* or *full* in M when $\mid M(h_j) \mid < c_j$, $\mid M(h_j) \mid > c_j$, or $\mid M(h_j) \mid = c_j$ respectively. Consider Figure 2.2 as an example of a HRT instance I_1.

A *matching* in the context of HRT, is a set of (resident, hospital) pairs such that no resident is assigned to more than one hospital and no hospital is over-subscribed. A matching is *stable* if it admits no blocking pairs, and we can define a blocking pair for an instance of HRT with respect to three different levels of stability, since the presence of ties forces us to extend the

Residents' preferences	Hospitals' preferences
$r_1: \ h_1 \ h_3$	$h_1 : (2) : \ r_3 \ (r_7 \ r_5 \ r_2) \ r_4 \ r_6 \ r_1$
$r_2: \ h_1 \ (h_5 \ h_4) \ h_3$	$h_2 : (3) : \ r_5 \ r_6 \ (r_3 \ r_4)$
$r_3: \ h_1 \ h_2 \ h_5$	$h_3 : (1) : \ (r_2 \ r_5) \ r_6 \ (r_1 \ r_7)$
$r_4: \ h_1 \ (h_2 \ h_4)$	$h_4 : (1) : \ r_8 \ r_2 \ r_4 \ r_7$
$r_5: \ h_3 \ h_1 \ h_2$	$h_5 : (1) : r_3 \ (r_7 \ r_6 \ r_8) \ r_2$
$r_6: \ (h_3 \ h_2) \ h_1 \ h_5$	
$r_7: \ h_3 \ h_4 \ h_5 \ h_1$	
$r_8: \ h_5 \ h_4$	

Figure 2.2: Instance I_1 of HRT

definition. In fact an hospital h can prefer r_a over r_b, r_b over r_a or to be indifferent between them. A pair $(r, h) \in \mathcal{R} \times \mathcal{H}$ is said to *block* a matching M for an instance of HRT, and is called a *blocking pairs* when:

- weak stability
  1. r, h find each other acceptable;
  2. r is either unassigned or strictly prefers h to his assigned hospitals in M;
  3. h either is under-subscribed or strictly prefers r to its worst assigned residents in M;
- strong stability
  1. r, h find each other acceptable;
  2. either,
     (a) r is either unassigned or strictly prefers h to his assigned hospital in M, and h is either under-subscribed or strictly prefers r to its worst assigned resident in M or is indifferent between them;
     (b) r is either unassigned or strictly prefers h to his assigned hospital in M or is indifferent between them, and h is either under-subscribed or strictly prefers r to its worst assigned resident in M.
- super stability
  1. r, h find each other acceptable;

2. r is either unassigned or strictly prefers h to his assigned hospital in M or is indifferent between them;
3. h is either under-subscribed or strictly prefers r to its worst assigned resident in M or is indifferent between them.

A matching is said to be *weakly stable*, *strongly stable* or *super-stable* if it admits no blocking pair with respect to the relevant definitions above. In this thesis, however, we consider only weak stable matchings and we refer to as simply *stable matchings*.

HRT is a generalisation of SMTI, namely if we select a HRT instance where every hospital has only one post we are in fact considering a SMTI instance, so it inherits some important properties. In particular the NP-hardness result [7] generalises to the case of finding a maximum weakly stable matching for HRT instances.

Chapter 3

A new local search algorithm for HRT

In Sections 3.1 and 3.2 we provide an introduction to local search methods and we describe how these approaches can be adapted to matching schemes. After the main algorithm is exposed in Section 3.3, Sections 3.4 and 3.5 we further refine basic techniques of the algorithm.

1 General scheme

Local search [10] is one of the fundamental paradigms for solving computationally hard combinatorial problems and in many cases represent the only feasible way for solving these large and complex instances. Local search algorithms are also naturally suited for dealing with the optimization criteria arising in many practical applications. The basic idea underlying local search is to start with a randomly or heuristically generated candidate solution of a given problem instance, which may be infeasible, sub-optimal or incomplete, and to iteratively improve this candidate solution by means of typically minor modifications. Different local search methods vary in the way in which improvements are achieved, and in particular, in the way in which situations are handled when there's no direct improvement. Most local search methods use randomization to ensure that the search process does not stagnate with unsatisfactory candidate solutions.

2 Definitions

Given an HRT instance P , we have to define an heuristic procedure that finds the initial match to begin the computation. We consider each resident in random order. Let

$$r_i : \ h_{i_1} \ h_{i_2} \ \ldots \ h_{i_s}$$

be the preference list of the i-th resident. We start by letting r_i make a proposal to h_{i_1}, if this hospital still has free places then we match r_i to h_{i_1}, otherwise we go ahead considering the next hospital h_{i_2}, and so on. If we aren't able to find an undersubscribed hospital, we set r_i as single. Since we let each hospital to become filled, we obtain a match with just few singles and, of course this match is not stable, but in this way we can start with a good number of residents already matched.

The next step is to define the catalog of minor modifications that will be used at every iteration step, in order to choose the next candidate solution. If BP is the set of blocking pairs in M we can compute the neighborhood this way:

$$NGHB_M = \{M \setminus b : b \in BP_M\}$$

which is the set of all matchings obtained by removing one of the blocking pairs in BP from M. More precisely, removing $b = (r, h)$ from M (written $M \setminus b$) means obtaining a new match M' in which r is matched with h and if h is now oversubscribed then we have to replace r with the worst preferred resident already matched to it (written $M(h) \setminus r^*$, where r^* was the worst preferred resident matched to h in M). In this setting a minor modification of a given matching M is the one obtained by changing one pair that was blocking in M. Note that by removing a blocking pair we can produce new singles.

To sistematically select the best adjacent solution M', we have to define a function

$$f(M') = \ \mid BP_{M'} \mid + \mid S_{M'} \mid$$

on the space of all possible matchings and with $S_{M'}$ as the number of singles in M. In words we have to select the neighbor with the minimum number of blocking pairs as well as the minimum number of singles. There are alternative evaluation functions we will consider in following sections.

From what we've seen so far there's nothing that prevents f to exhibit a local minimum for some matching, and if this is the case, our algorithm will not leave it, since the criterion for which it decides the next solution is based exclusively on f. To avoid such a situation, at each search step we perform a *random walk* with probability p (where p is a parameter of the algorithm), which removes a randomly chosen blocking pair in BP from the

current matching M. We'll se in Chapter 4 that even for big value of p our local search method reaches satisfisable solutions.

Since the notion of neighborhood is based on the set of blocking pairs in a given matching M, we encounter a problem if this set is empty, and the canditate solution we're considering in that moment is not good. In this case we decide to restart the computation from the beginning, that's to generate a random matching and to improve it step by step.

Finally, there may be conditions under which the implementation of our code takes a lot of steps to reach a stable match and this turns into an unfeasible situation in real systems. A possible way to prevent these long runs, is to count the number of steps and abort if the execution get over a default bound.

3 Local search algorithm for HRT

Algorithm 1 implements the logic described in Section 3.2. Here we give some information to understand its behaviour. In the input P is a legal HRT instance and in line `1` and `19` we calculate a random match from P, following the procedure given at the beginning of Section 3.2. Parameters *maxsteps* and *maxrestarts* serve to limit the execution and to prevent infinite runs, which can happen in theory. Although we can keep *maxrestarts* under the effect of *maxteps* and nothing would change with only one parameter, we use both of them. The possibility of making a restart it's very important because, if in line `14` we find UBP empty it follows that M is stable, and since nothing prohibit that this stable match is not of the maximum size as possible, we have to retry starting with another random match. *maxrestarts* permits a finer tuning of computation, expecially it allows us to control how many new random matches our code can exploit in order to search for a good maximum size match. Since every restart represent a new chance to improve the current match, we experiments with this parameter and we discover that even with a low number of restarts our code can reach a good maximum size match.

Line `7` is about calculating the neighborhood. Lines `10`, `11` and `12` take care of the random walk, which is performed by comparing a random number, returned between 0 and 1, with p, given as input. From line `23` to line `29`, the algorithm compare the evaluation function on each adjacent matching, which is defined by UBP and chooses the best solution to move to.

Algorithm: HRTLocal

Input: an HRT instance P, an integer $maxsteps$, an integer $maxrestarts$, a probability p

Output: a matching

```
1  M ← random match for P
2  steps ← 0
3  restarts ← 0
4  M_best ← M
5  f_best ← f(M)
6  repeat
7  |  UBP ← blocking pairs in M
8  |  if f(M) = 0 then // prefect match
9  |  |  return M
10 |  if rand() ≤ p then
11 |  |  randomly select b in UBP
12 |  |  M ← M \ b
13 |  else
14 |  |  if UBP is empty then // M is stable, random restart
15 |  |  |  if f(M) < f_best then
16 |  |  |  |  f_best ← f(M)
17 |  |  |  |  M_best ← M
18 |  |  |  end
19 |  |  |  M ← random match for P
20 |  |  |  restarts ← restarts + 1
21 |  |  else
22 |  |  |  minbp ← (number of residents) × (number of hospitals)
23 |  |  |  foreach blocking pair b in UBP do
24 |  |  |  |  M' ← M \ b
25 |  |  |  |  nbp ← f(M')
26 |  |  |  |  if nbp < minbp then
27 |  |  |  |  |  best_bp ← b, minbp ← nbp
28 |  |  |  end
29 |  |  |  M ← M \ best_bp
30 |  steps ← steps + 1
31 until (steps ≤ maxsteps) ∧ (restarts ≤ maxrestarts)
32 return M_best
```

4 Reducing neighborhood

If we consider pseudocode of Algorithm 1 we easly see that lines `23-28` are very heavy in terms of computational time. We evaluate, for each element in UBP, the set of blocking pairs for a given matching, and we know this is at least quadratic in the size of the problem [1]. Moreover the set UBP calculated in line `7` could be very large and some of them may be useless, since their removal would surely lead to new matchings that will not be chosen by the evaluation function. Let's look an example to calrify this idea. If we consider a generic preference list

$$r_i : h_{i_1}\ h_{i_2}\ h_{i_3} \dots h_{i_{t-1}}\ h_{i_t}\ h_{i_{t+1}} \dots h_{i_{r-2}}\ h_{i_{r-1}}\ h_{i_r}$$

with the highlighted term as blocking pairs for the current match M, it's obvious that there is a strict order relation between blocking pairs involving the same resident. This introduces a new definition

Definition 3. *A blocking pairs $bp = (r, h)$ is said to* `dominate` *another blocking pair (r, h') from the resident point of view if r prefers h to h'. A similar reasoning can be done from the hospital's point if view.*

In the previous example (r_i, h_{i_2}) dominates (r_i, h_{i_t}) and $(r_i, h_{i_{r-1}})$ from the resident point of view. Directly from this case, we can deduce something important for the pair (r_i, h_{i_2}). It is not dominated by any other blocking pair and we see that, if it is removed from the current matching, also pairs like (r_i, h_{i_t}) and $(r_i, h_{i_{r-1}})$ turns out to be no more blocking. In this perspective we provide the following definition.

Definition 4. *A* `resident undominated blocking pair` *(respectively, a* `hospital undominated blocking pair`*) is a blocking pair such that there is no other blocking pair that dominate it from the resident point of view (resp. hospital). When the point of view is clear we will omit it*

It is easy to see that, if we remove (r, h) which is an undominated blocking pair from both resident and hospital point of view, there are no more blocking pairs in which r and h are involved. This property would not be true if we remove a dominated blocking pair. We would like to focus on the removal of undominated blocking pairs to pass from one marriage to another, and this because the number of blocking pairs to consider is drastically reduced.

In algorithm UB1, we find the undominated blocking pairs from the resident's point of view and, among these, we keep only the undominated blocking pairs from the hospital's point of view. This is done using array upb which,

Algorithm: UB1

Input: an HRT instance P with n number of residents and s number of hospitals, a matching M

Output: a set BP of undominated blocking pairs

```
1  BP ← ∅
2  ubp ← array of length s with indices h_1, ..., h_s
3  foreach h_i, i = 1...s do
4  |  ubp[h_i] ← null
5  end
6  foreach r_j, j = 1...n do
7  |  found ← false
8  |  foreach (h_i in r_j's preference list better than M(r_j)) ∧ !(found) do
9  |  |  if (r_j, h_i) is a blocking pair then
10 |  |  |  if ubp[h_i] = null or h_i prefer r_j to ubp[h_i] then
11 |  |  |  |  ubp[h_i] ← r_j
12 |  |  |  found ← true
13 |  |  end
14 |  end
15 end
16 foreach h_i, i = 1,...s do
17 |  if ubp[h_i] ≠ null then
18 |  |  BP ← BP ∪ {(ubp[h_i], h_i)}
19 end
20 return BP
```

after lines 6-15, contains, for each hospital h, a resident r such that (r, h) is an undominated blocking pair from both resident's point of view and hospital's point of view, if it exists. At the end of algorithm UB1, the set BP of blocking pairs returned contains at most one undominated blocking pair for each resident and hospital.

We modify our algorithm by using Algorithm UB1 to compute the blocking pair set UBP in line 7 of HRTLocal. Notice that, considering the set of undominated blocking pairs instead of all blocking pairs, we limit the size of the neighborhood, and, since each resident or each hospital is involved in at most one of the undominated blocking pairs found by UB1, we have at most $n + s$ neighboring matchings to evaluate, where n is the number of residents and s is the number of hospitals. Algorithm UB1 is developed following the structure of the homonymous algorithm UB1 presented in [3] for SMTIs.

Let us now analyze more carefully the set of blocking pairs obtained by procedure UB1. Consider the case in which a resident r_i is in two blocking pairs, say (r_i, h_j) and (r_i, h_k), and assume that (r_i, h_j) dominates (r_i, h_k) from the resident's point of view. Then, let h_j be in another blocking pair, say (r_z, h_j), such that (r_z, h_j) dominates (r_i, h_j) from the hospital's point of view. In this situation, UB1 returns (r_z, h_j). The elimination of this blocking pair automatically eliminates (r_i, h_j) from the matching, since it is dominated by (r_z, h_j); however, it does not eliminate the blocking pair (r_i, h_k). We would like to obtain a new algorithm that will also return the blocking pair (r_i, h_k), so to avoid having to consider it again in the subsequent step of the local search algorithm. This is the rationale underlying Algorithm UB2 which adapts to HRTs the algorithm UB2 for SMTIs, shown in [3].

Summarizing, we have defined two algorithms, HRTLocal with UB1, HRTLocal with UB2, to find a stable matching for a given HRT instance. These algorithms differ only in the set of blocking pairs considered when defining the neighborhood. Due to their ability to restart, our algorithms have the PAC (probabilistically approximate complete) property [5]. That is, as their runtime goes to infinity, the probability that the algorithm does not return an optimal solution goes to zero. Starting from the initial matching, the algorithm performs one or more steps in which we remove a blocking pair. This sequence of blocking pair removals has been shown to converge to a stable matching with non-zero probability in the context of SMs with incomplete preference lists [11].

Algorithm: UB2

Input: an HRT problem P of n residents and s hospitals, a matching M
Output: a set BP of undominated blocking pairs

```
1  pos ← array of length n with residents as indices
2  fnd ← boolean array of length n with residents as indices
3  ubp ← array of length s with hospitals as indices
4  R(m_j, M(m_j)) ← position of hospital M(m_j) in m_j's preference list
5  for j = 1...n do
6  |  pos[r_j] ← 1, fnd[r_j] ← false
7  end
8  for i = 1...s do
9  |  ubp[h_j] ← null
10 end
11 finished ← false, BP ← ∅
12 while !finished do
13 |  foreach r_j, j = 1...n do
14 |  |  for i = pos[r_j] to R(r_j, M(r_j)) and !fnd[r_j] do
15 |  |  |  if (r_j, h_i) is a blocking pair then
16 |  |  |  |  if ubp[h_i] = null or h_i prefers r_j to ubp[h_i] then
17 |  |  |  |  |  if h_i prefers r_j to ubp[h_i] then
18 |  |  |  |  |  |  fnd[ubp[r_j]] = false
19 |  |  |  |  |  ubp[h_i] ← r_j
20 |  |  |  |  fnd[r_j] ← true
21 |  |  end
22 |  |  pos[r_j] ← i + 1
23 |  end
24 |  finished ← true
25 |  foreach r_j, j = 1...n do
26 |  |  if !fnd[r_j] and pos[j] ≤ R(r_j, M(r_j)) then
27 |  |  |  finished ← false
28 |  end
29 end
30 foreach i = 1 to s do
31 |  if ubp[i] ≠ null then
32 |  |  BP ← BP ∪ {(ubp[h_i], h_i)}
33 end
34 return BP
```

5 Different evaluation functions

The operation to compute the number of blocking pairs in a given matching is quite expensive. If P is an HRT instance with n residents and s hospitals, we have to consider $n \times s$ combinations of pairs to be blocking pair and, given the definition of blocking pair, for a resident preference list $r^* : \ h_1 \ \ldots \ h^* \ h_s$ and $M(r^*) = h_s$, in the worst case one have to scan all the s elements of the preference list to discover that r^* prefers h^* to h_s. Of course this last situation depends on how one has implemented the preference list data structure, so this comment it's not genereal and serves only to understand how complicated the computation can be.

Let's focus on the operation that removes one blocking pair from a matching to understand how many new blocking pairs may arise. So from matching M we can build $M' = M \backslash (r, h)$. If both r was single and h was undersubscribed in M then in the new match there will be one less blocking pair for sure, because there are no other pairs involved. Let's consider other cases separately:

- if h was full then it has to replace let's say r^*, the worst element that was matched to it. In this case r^* becomes single and like we see in r^* preference list, from $h_{i_{t+1}}$ to h_{i_s} there may be new blocking pairs.

$$r^* : h_{i_1} h_{i_2} \ldots h = h_{i_t} \, h_{i_{t+1}} \ldots h_{i_{s-1}} h_{i_s}$$

  Thus we can have only elements of type (r^*, h_{i_j}) with $j = t + 1 \ldots s$ as new blocking pairs involving r^*.

- at the same way h^* is the old hospital matched to r and it now has free places, so it may introduce new blocking pairs involving itself and possible residents found in its preference list.

At the opposite side, we have r and h that were blocking in M and if we consider for example r's preference list

$$r : h_{j_1} h_{j_2} \ldots h = M'(r) \ldots h^* = M(r) \ldots h_{j_{n-1}} h_{j_n}$$

where h have to come first in order respect to h^*, because of the blocking pairs (r, h), we deduce that the number of blocking pairs involving r can only decrease from M to M'. The same results by considering h's preference list.

In conclusion, the only sources of new blocking pairs are those which can involve r^* and h^* against elements found in theirs preference lists, because these are the only elements which undergo a change of partner in the new

match and because, with respect to r and h, the number of blocking pairs involving them can only decrease.

In the context given by considering $M' = M \setminus (r, h)$ for (r, h) blocking pair, $h^* = M(r)$ and r^* as the worst element $\in M(h)$, we can define new functions

$$f_\alpha(M') = \text{number of blocking pairs considering } (r, h_r) \text{ and } (r_h, h)$$

$$f_\beta(M') = \text{number of blocking pairs considering } (r^*, h_{r^*}) \text{ and } (r_{h^*}, h^*)$$

with $r_{(\cdot)}$ and $h_{(\cdot)}$ representing generic elements which belong to r's and h's prederence lists. After all, with these new defitions we can change the way we select a new candidate solution and most importantly make a decision in $O(\lambda)$ where λ is the maximum lenght in preference lists.

Chapter 4

Experimental results

This chapter is devoted to present experimental results about how HRTLocal performs on artificial situations. Since matching algorithms are primally used in practice as large-scale centralized matching infrastructures and given that stability is a required property for this schemes to be useful, the issue to obtain the largest possible match is greatly important. This is the main foucus of Section 4.1. In Section 4.2 we consider the temporal behaviour, and we learn about how efficiently HRT local approaches can perform.

1 Maximum match size

In generating HRT instances, a whole range of parameters and data characteristics can be varied, including

- number of residents, hospitals and posts
- length of residents' preference lists
- way in which posts are distributed among hospitals
- variation in hospitals and residents popularity
- number, length and distribution of ties

Since the number of possible combinations of these parameters is huge, in what follows we state a default instance which, as the discussion proceeds, we modify in order to show how the variation of this key factors affect the performance of our algorithms.

We now give an insight of the procedure used to build artificial instances. With a setting of 100 residents and 10 hospitals, we choose to assign 10 posts each hospitals, in order to potentially match all residents and to guarantee that all posts are uniformally distributed among hospitals. For each resident we randomly select, with uniform distribution, a sequence of hospitals for a lenght of $npref$ (this, of course, have to be less then the number of hospitals). For instance, if we are building the preference list of r and we select a random hospital h then we add r to the preference list of h, this way the acceptance between residents and hospitals is symmetric. Once we've considered all the residents we shuffle all the hospitals' preference list so, in this manner, no one resident is preferred over another. Then for setting ties we iterate over each hospital's preference list as follows: for each resident in a preference list, in position $j \geq 2$, with probability $ptie$ we set the preference for that resident as the preference for the resident in position $j - 1$ (thus putting the two in a tie). Note that we consider to allow indecision only for hospitals side. With a tie probability of 0.5, we decided to run our HRTLocal with $maximumstep = 5000$, $maxrestarts = 10$ and a probability of random walk of 0.2. For each instance, since we let the execution employing different restarts (that is $maxrestarts$), we sample the max value, min and average on that instance, and we execute the computation on 100 instances for each value of the parameter under analysis. At the end we calculate an arithmetic mean for each value.

Now, we want to study the best value for $npref$. In Figure 4.1, where it's plotted the match size when varying the number of preferences for each resident, we see that for $npref \geq 6$, HRTLocal almost always finds a perfect matching. Since we are considering the maximum match size problem we have to set this parameter in such a way we can produce a sort of worst-case analysis. Moreover, with this setting, we can also show another key factor for this problem, tie probabilty. For what follows we decide $npref = 2$.

Figure 4.2 serves to show a further refinement in the worst case analysis. Here we vary the number of hospital, paying attention to calculate the number of free posts in each hospital in order to potentially match almost all residents. Thus

$$hcap = \frac{100}{nhos}$$

where $hcap$ is the hospital capacity and $nhos$ the total number of hospitals. $hcap$ has to be an integer so, as an example, for $nhos = 30$ the operation yields $hcap = 3$, and the total number of available posts is $ncap \times nhos = 90$. This is why we plot a normalized number of singles: that is the number obtained by comparing the maximum number of resident that can be match for a given $nhos$

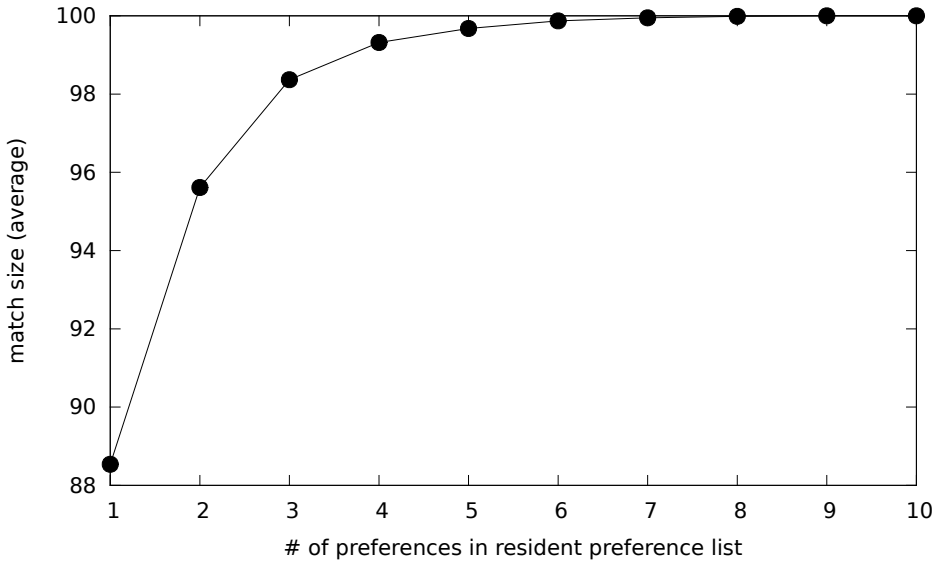

Figure 4.1: The matching size varying the length of the residents' preference list

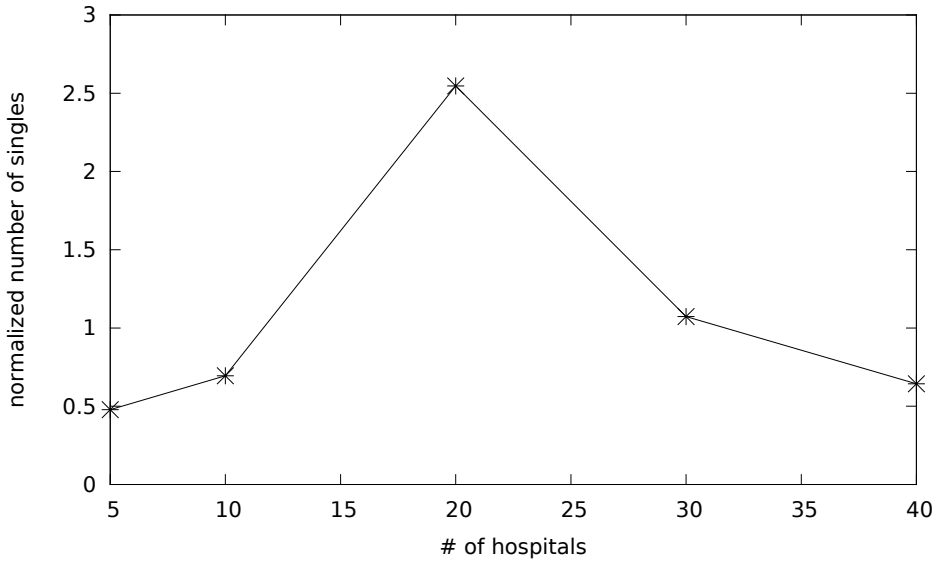

Figure 4.2: Normalized number of singles varying the number of hospitals

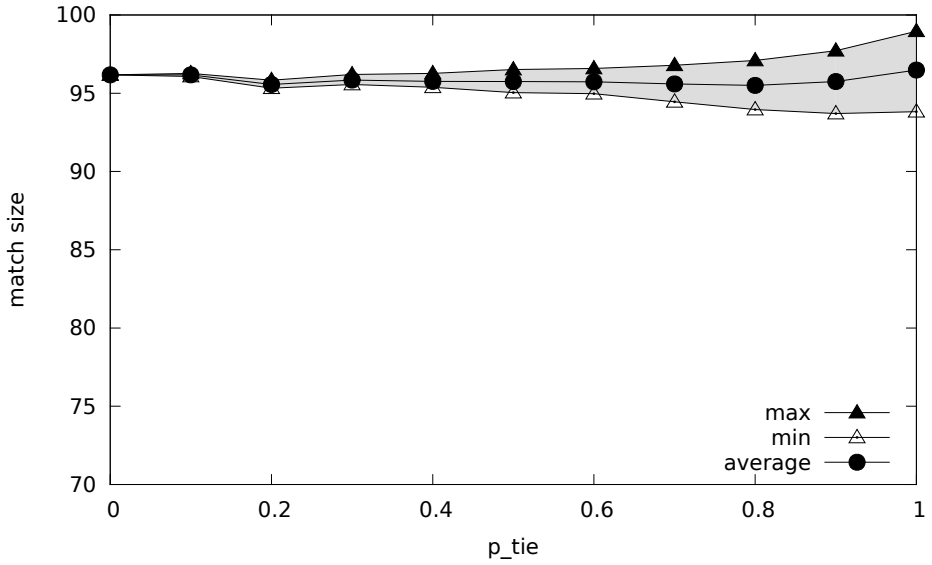

Figure 4.3: Uniform hospital popularity

value and the actual number of singles in the match, returned by HRTLocal execution. Form the figure we deduce that for $nhos = 20$, and $hcap = 5$, we have the largest number of singles and so we keep this value as default for the rest of the analysis.

At this point we have arranged the parameters for the maximum match size problem and we can now test the presence of ties in our local search approach. This is an important issue because, from the literature, we already know that the key in obtaining a big match size is to find the proper way of breaking ties [7] and since our approach doesn't care of how ties are broken, it's quite interesting to see how it can resolve this hard problem. In fact our algorithm only uses the definition of blocking pair.

In Figure 4.3 we can see that bigger ties probability provides bigger spread around the average value, which is about 96, starting from a tie probability of 0, where all the match sizes (max, min and average) overlap around the same value, and ending with those values spaced around the average. As the tie probability increases the max value is barely two resident more than the average, so it doesn't vary so much. The same reasoning for skewed hospital probability in Figure 4.4. We modified the way in which we generate the instance, precisely, when each resident has to choose some hospitals, the random choice for the most popular hospital is five times more popular then the less one, and the distribution is approximately linear between this two extremes. In this new setting we see the mean value (this time around 89) decreses but

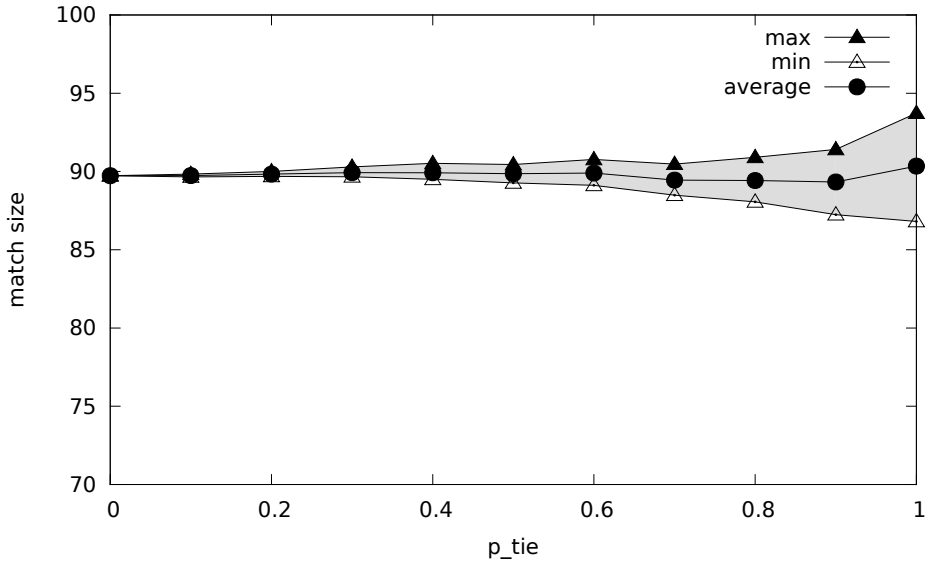

Figure 4.4: Skewed hospital popularity

the progress is the same as above.

In these experiments we sampled, and then reported, all values (like max, min and average), among different restarts. We see that as the max doesn't increase a lot, also the min value doesn't loose too much residents. This is significant because it suggests that if one is not interested in obtained the most largest match size, however, he is capable of running the algorithm for few restarts and yet obtaining a good solutions in terms of the size of the returned matching and time needed to find it. We will consider in Section 4.2, about time analysis, that this is good because a solution, even for big instances, can be obtained in few seconds.

In following table we consider a comparison between HRTLocal and another heuristic-based approach found in literature [6]. This method is very intelligent in the way it breaks ties and it is considered ad hoc for this kind of problems.

	Uniform hos. pop.			Skewed hos. pop.		
Algo	Max	Min	Ave	Max	Min	Ave
IM	999.6	993.0	994.9	975.2	971.8	973.5
HRTLocal	994.6	991.5	993.8	965.8	961.2	963.6

Instances of this type are generated with 1000 residents, 100 hospitals, 1000 posts and residents' preference list of lenght 5. Under a tie probability of 0.5 we

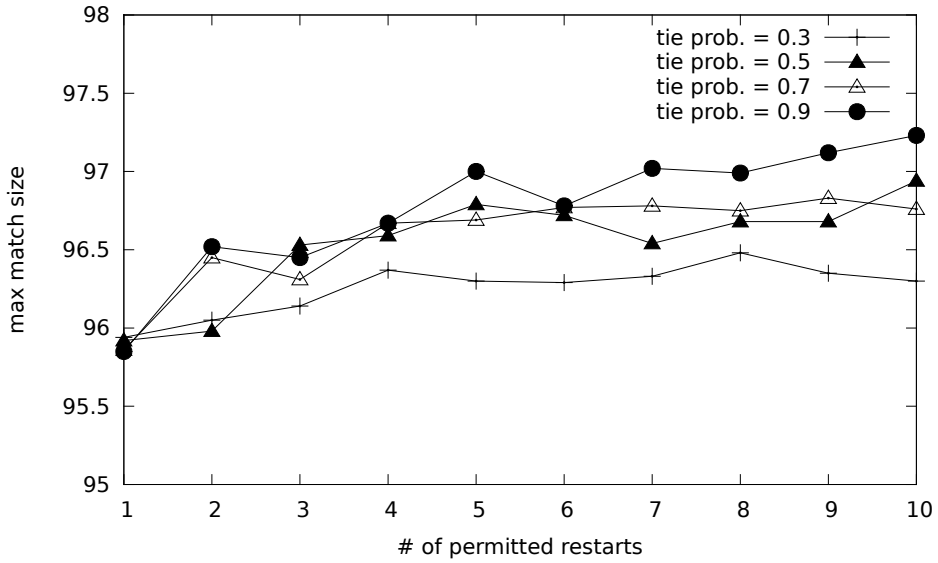

Figure 4.5: Max match size varying the number of permitted restarts

let hospitals either to be uniformly popular or to exhibit a skewed popularity, with the most popular hospital attracting 5 times the number of applicants as the least popular. We see that what is a little better behaviour, under uniform popularity, becomes quite considerable under skewed conditions. This shows how much an implementation, which considers as primary logic advanced method of tie-breaking, can beat our local search approach, which does not inspect into how ties are involved.

Let's consider Figure 4.5 where we have plotted the maximum match obtained when varying the number of restarts permitted, that's *maxrestarts*. As usual, we average samples on 100 instances for a given parameter value and we let *maxrestarts* varying from 1 to 10 (more than this doesn't show interesting results). In addition, we consider this experiment under different tie probabilities, low as 0.3 and very high. We see that, even if the idea for which the more restarts permitted, the more the probability to get a great max match is right, the gradient of such variation is really smooth and we deduce that the random match upon which the computation is started is quite irrelevant. HRTLocal needs an handful of restarts to reach a max match size as big as it can. That's why we can set the number of allowed restarts to values even around 10 and have a good probability to obtain a big max match size.

To summarize, we used the first part of this section to find the parameters of artificial instances generation by which the algorithm give the lower max size match. Thus, we considered a sort of worst case analysis and we saw that

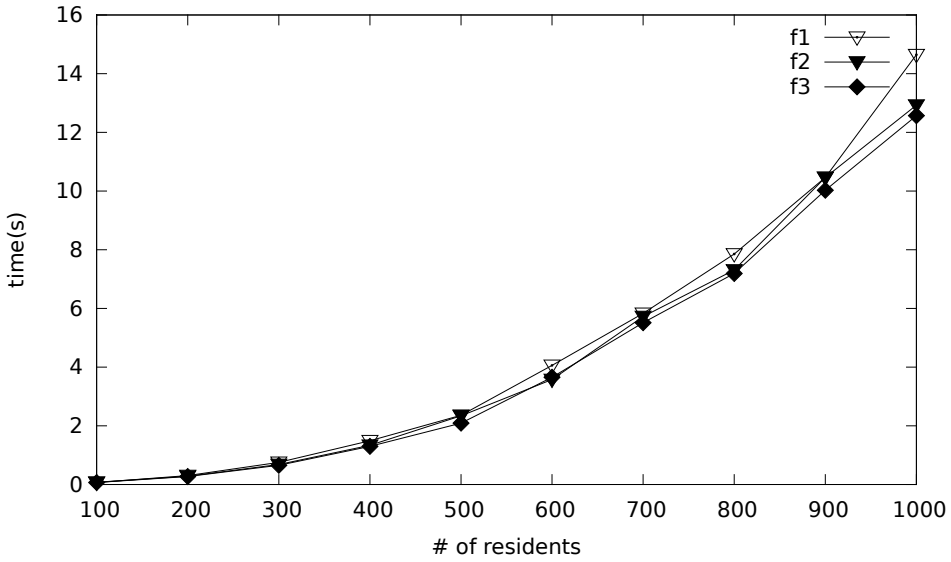

Figure 4.6: Temporal execution on different problem sizes

HRTLocal can match a good number of residents, even under these conditions. When we compared our algorithm to ad-hoc and powerful tie breaking strategies, we discovered that it has difficulties on very specific instances. We also noticed that the number of restarts does not affect too much the maximum size match returned by HRTLocal.

2 Temporal analysis

In this section we study the temporal behaviour of HRTLocal. As we did in Section 4.1 for the maximum match size problem, here, we consider different parameters and how our algorithm performs on them. In particular from Section 3.5 we test all the evaluation functions we have defined.

$$f1(M') = f_\alpha(M') + \mid S_{M'} \mid$$

$$f2(M') = f_\beta(M') + \mid S_{M'} \mid$$

$$f3(M') = \mid S_{M'} \mid$$

In our code these functions are used to choose the matching that represents the best canditate solution. Figure 4.6 shows an experiment where we let the number of residents varying from 100 to 1000. For each resident size we

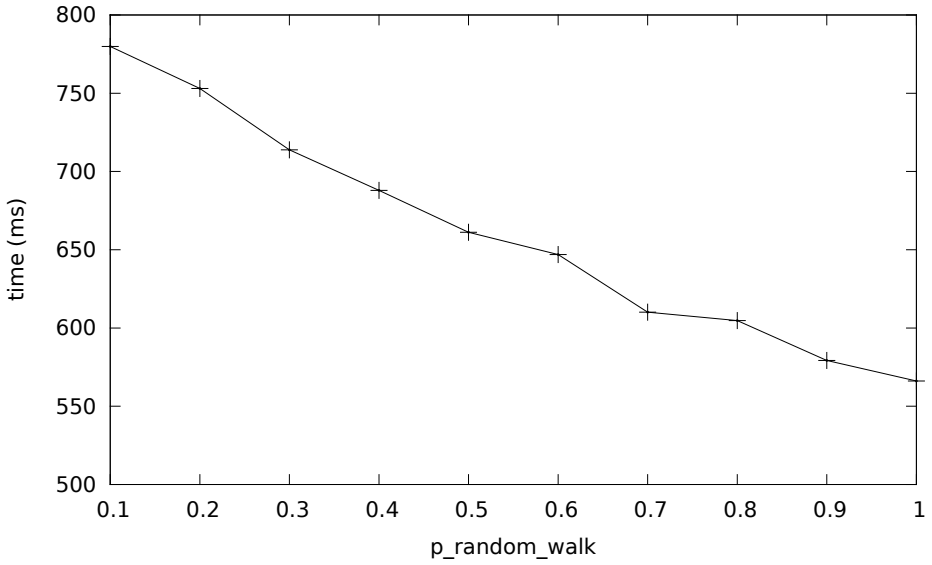

Figure 4.7: Time versus random walk probability

calculate the number of hospitals to be present in that instance using

$$hcap = \frac{\text{number of residents}}{nhos} = 10$$

thus keeping capacities constant. We also set $npref = 5$, the number of preference a resident can express. Since we are considering the temporal execution, we fix $maxrestarts = 1$ and we average on 10 instances for each set of parameters. We note that the behaviour is quadratic and this happen because of the computation of the neighborhood in line 7 of HRTLocal. Also, $f3$ is the best one, in fact we have used it in all previous experiments.

In Figure 4.7 we show that, increasing the random walk probability parameter p, we speed the execution and this is possible because we spend no time choosing the proper neighboring. We do this in order to prevent stagnation in local minimums for the evaluation function.

Section 3.4 gives foundation for two different methods which compute the set of undominated blocking pairs, that's UB1 and UB2. In Figure 4.8 there is a comparison, which shows that UB1, the simpler, is nearly faster (we have used UB1 in all previous experiments). In fact, UB2 returns a set which, in theory, would save computation time because it prevents to reconsider the same blocking pair twice. Maybe, real benefits of such logic can be seen in larger instances than those we have considered in our experiments.

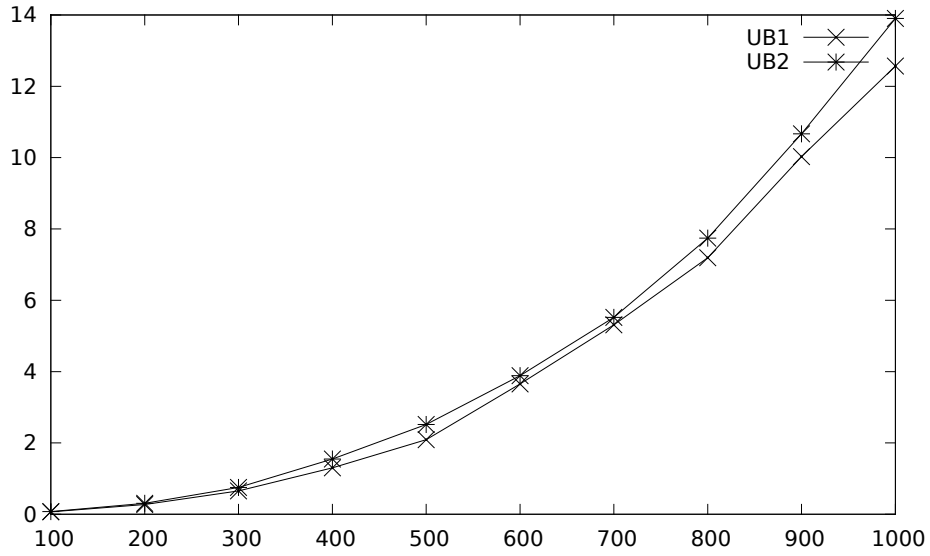

Figure 4.8: Comparison between UB1 and UB2

We conclude this chapter noting that our algorithm is effective. Indeed, it computes a stable matching for instances of size 1000 in ten seconds, and it returns a match size which is at least comparable with powerful adhoc strategies.

Chapter 5

Conclusions and future work

We have studied the HRT problem due to its important practical applications. We have discovered that it comes as a generalization of a mathematical model to matching two generic sets and, in respect to the classic one, which is polynomial, it introduces computational complexity because of the presence of a certain degree of imprecision and incompleteness. We have defined a local search algorithm for finding the largest stable matchings and we have evaluated it on large instances. It turns out that it behaves similarly to ad hoc methods when each hospital has the same popularity. In fact, for certain configurations our algorithm always succeeds in matching all residents involved, but for instances where hospitals exhibit a skewed popularty, it loses performance.

A possible future work direction is to develop a more specific algorithm. For example it would be worth to increase the awareness in tie-breaking, since it has been proven to be a key factor[7], and our method is completely indipendent of how ties are considered. Another direction is to specialize our local search approach to specific input, for example to handle instances where some hospitals are more popular than others, and that occurs often in real scenarios. This could be done by defining in more sophisticated way the neighborhood and the evaluation function.

Bibliography

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

[2] M. Gelain, M.S. Pini, F. Rossi, K.B. Venable, T. Walsh. *Local Search Approaches in Stable Matching Problems.* Algorithm 6(4). 591-617. 2013

[3] M. Gelain. *Reasoning with incomplete and imprecise preferences.* PhD thesis. Universita' di Padova e Universita' di Bologna. 2010.

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

[5] H. Hoos. *On the runtime behaviour of stochastic local search algorithms for SAT.* In Proc. AAAI'99, 661-666. 1999.

[6] R. Irving, D. Manlove. *Finding large stable matchings.* ACM Journal of Experimental Algorithmics 14, 2009.

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

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

[9] G. O'Malley. *Algorithmic aspects of stable matching problems.* PhD thesis. University of Glasgow.

[10] F. Rossi, P. Van Beek, T. Walsh, editors. *Handbook of Constraint Programming.* Elsevier. 2006.

[11] Alvin E. Roth and John H. Vande Vate. *Random paths to stability in two-sided matching.* Econometrica, 58(6):1475-1480. 1990.