Alma Mater Studiorum – Università di Bologna
Università di Padova

DOTTORATO DI RICERCA IN

Informatica

Ciclo XXIII

**Settore scientifico-disciplinare di afferenza:** INF/01

# Reasoning with incomplete and imprecise preferences

**Presentata da: Mirco Gelain**

**Coordinatore Dottorato**

**Simone Martini**

**Relatore**

**Francesca Rossi**

**Co-Relatore**

**K. Brent Venable**

**Esame finale anno 2011**

# Contents

# Chapter 1

# Introduction

While imprecision and incompleteness are pervasive in many real life scenarios which include issues regarding preferences, most of the artificial intelligence (AI) methods for handling preferences have overlooked such crucial aspects. Soft constraints are undeniably one of the most successful AI approaches to preferences.

The aim of this Ph.D. thesis is to extend the soft constraint formalism to handle both incomplete or imprecise preferences. We achieve this by providing new theoretical frameworks equipped with solving machineries that exploit both systematic and local search approaches.

Motivated by the success of our local search methods in the context of soft constraints, we study the impact of local search in a specific class of problems (stable marriage problems) where incompleteness and imprecision have a specific meaning and up to now have been tackled with different techniques.

## 1.1 Motivation and main goal

Preferences are intrinsic in almost all real life contexts. In psychology, preferences could be conceived of as an individual's attitude towards a set of objects, typically reflected in an explicit decision-making process or to mean evaluative judgment in the sense of liking or disliking an object. In artificial intelligence, preferences help to capture agents' goals. In databases, preferences help in expressing user's queries and filtering the desired items. In mathematical decision theory, preferences (often expressed as utilities) are used to model people's economic behavior.

During the past decades, in computer science many frameworks and models have been developed to make possible some form of automated preference

reasoning. In AI and in particular in the Constraint Programming (CP) area, many frameworks have the expressive power to represent preferences in forms of costs, weights, probability, etc. [6, 7, 64, 24].

However, there are situations in which the user of an automated system may have his preferences but does not want to reveal all of them, due to privacy issues or communication or elicitation costs. This is the case, in an artificial environment, like Internet where there are many distributed applications based on multi-agent systems in which agents have to share knowledge in order to achieve their tasks. In this settings, it might be the case that some agents don't want to share information for free and they require others to pay a fee to reveal the desired data. On the other hand, data may be available in subsequent times, but we want to have a desirable solution as soon as possible. In other real-life situations, the size or the complexity of the problem may prevent the user to actually be able to express completely and precisely all of his preferences. For example, in the case of on-line shops, asking the user for too many preferences may result in him leaving the web site. Also, often preferences may be specified with a certain degree of imprecision. For instance, different sensors may have different tolerance levels when they provide data. In this thesis we study how to model these situations and how to reason in these contexts. More precisely, we will consider semiring-based soft constraints as a starting preference reasoning framework [6, 7] and we will extend it to model missing and imprecise preferences by providing algorithms to solve such kind of problems.

An example of problem where preferences are crucial, and where incompleteness and imprecision may be present, is the stable marriage problem (SM) [26, 36, 54]. In the classical formulation, $n$ men and $n$ women express their preferences (via a strict total order) over the members of the other sex. Solving a SM problem means finding a stable marriage where "stable" means that there are no man and woman who are not married to each other, who would both prefer each other to their partners. The stable marriage problem has a wide variety of practical applications, ranging from matching resident doctors to hospitals, to matching students to schools, or more generally to any two-sided market. The agent's preference lists may be incomplete (to say that some agents are not accepted) and may admit ties or incomparability, allowing for some form of imprecision. We will provide algorithms to find stable matchings and we will evaluate their efficiency and sampling capabilities.

## 1.2 Main results

In the past years, several frameworks have been proposed to allow for modeling preferences in automated reasoning systems. Given a well-defined preference representation, the goal is to find the best solution. Thus, the problem is not a satisfaction problem as in classical CSP (where a solution has to "simply" satisfy all the constraints), but it becomes an optimization problem (where each solution has an associated level of preference).

The field of AI has greatly contributed in this context to the development of many formalisms to represent qualitative and quantitative preferences. For instance, CP-Nets [9] is a qualitative framework that can be used to handle conditional preference statements, which are expressed over values of given features which are used to decompose the space of outcomes.

In this thesis we focus on quantitative preference representation and, in particular, on soft constraints [6, 7]. A soft constraint is a classical constraint (where a constraint may be satisfied or not) with a way to associate a value (interpreted as a preference, a cost, a probability, etc.) to the entire constraint or to a combination of values assigned to the variables involved in that constraint. In general in a problem there are several soft constraints, therefore, a way to decide the values given by different constraints is needed in order to compute the level of preference of a global assignment (i.e., a solution). Moreover, it is also necessary to have a method to compare different solutions in order to decide which is the best one.

Many formalisms to represent soft constraints have been developed. For example, valued CSPs [24] extend the basic CSP framework by assigning a preference value to each constraint, in order to deal with over-constrained problems. As another example, Fuzzy CSPs [64] assign a value between 0 and 1 to represent a preference level associated to each constraint assignment. Another widely used modeling of soft constrains are the Weighted CSPs where a cost between 0 and $+\infty$ is assigned to each constraint assignment. In the fuzzy case the goal is to maximize the minimal global preference whereas, the goal in solving weighted CSPs is to minimize the global cost. In this thesis, we will consider the general framework based on c-semirings [6, 7]. This framework is based on the algebraic structure of a c-semiring, where the elements of a set, used to represent preferences, can be combined and compared via the operators associated to the c-semiring. According to chosen c-semiring, this framework can model the classes of soft constraints just presented, as well as others.

### Incomplete soft constraints

We extend the c-semiring framework to express the possibility of having some missing preferences. By admitting missing preferences, solving this new kind of optimization problems, that we call Incomplete Soft CSPs (ISCSPs), becomes a different task. In fact, the new goal is to find robust solutions, i.e., solutions which are the best ones independently of the missing preferences. With this in mind, we define new notions of optimality: the possibly optimal solutions, which are be the best in at least one way of revealing the missing preferences and the necessary optimal solutions, which are optimal no matter what the missing preferences are.

Since the notion of necessary optimal solution is a strong notion of optimality, it may happen that an ISCSP instance has no such solutions. However, elicitation of some preferences may make some solution necessarily optimal. We characterize the sets of necessarily and possibly optimal solutions and we define a scheme that combines search and preference elicitation. Such a scheme has a large number of different instantiations, each corresponding to a concrete algorithm to find necessarily optimal solutions. In particular, we interleave systematic and local search with an elicitation phase. The aim is to ask the user as few preferences as possible during search before finding a necessarily optimal solution.

In our systematic search approach, we exploit a modified version of the classical branch and bound scheme and we consider different elicitation strategies which differ from what is elicited, when the elicitation takes place, and who guides the search. The experiments, on randomly generated incomplete soft constraint problems (fuzzy and weighted), as well as on problems with hard constraints and on fuzzy temporal constraints, demonstrate that some of the algorithms are very good at finding necessarily optimal solutions, without eliciting too many preferences. More precisely, we compute the amount of elicited preferences, as well as the user's effort. This may be much larger than the number of elicited preferences, as it contains all the preference values the user may have to compute to be able to respond to the elicitation requests. For example, suppose we ask the user for the worst preference value among $k$ missing ones. The user will communicate only one value, but he may have to compute and consider all $k$ of them. The number of elicited preferences is important when the concern is to communicate as little information as possible. The user's effort, on the other hand, measures the hidden work the user has to do to be able to communicate the elicited preferences. Our best algorithms need to elicit as little as 10% of the missing preferences, and require the user to consider around 20-30% of the missing preferences.

Our local search approach performs elicitation at each search step. Also

in this case different elicitation strategies have been considered. The experiments on the local search approach, on randomly generated fuzzy and weighted incomplete soft constraint problems, show that this approach is promising in terms of solution quality, elicited preferences, and scaling capabilities. In particular, as local search is not a complete method, we show that the distance between the preference of the returned solution by the local search algorithms and that of the necessarily optimal solution returned by our best branch and bound algorithm, is very small. In terms of elicited preferences, our best local search algorithm asks for more preferences than our best systematic algorithm (25% in the fuzzy case) but it scales better on bigger problems.

**Interval-based soft constraints**

We also extend the c-semiring framework to model the imprecision that is often present in real-life problems. In particular, we extend soft constraints to allow for preference intervals. These intervals can contain a single element (and in this case we are in the usual soft constraint formalism), or the whole range of preference values (when there is complete ignorance about the preference value), or a strict subset of the preference values. We call Interval Valued CSPs such problems. Working with intervals makes the global solution preference an interval itself, hence, the c-semiring operators have to be redefined accordingly to combine intervals. Moreover, to compare two or more solution intervals, we have to consider different situations depending on the relative positions of those intervals. For example, one interval may be contained into another, or they may be disjoint, and so on. Therefore, we consider several notions of optimal solutions based on intervals and we provide algorithms to find such optimal solutions, and also to test whether a given solution is optimal.

By selecting a specific preference value from each interval, different scenarios may arise. For instance, if we select the lower bound of every interval, we are considering the worst possible scenario. If we eliminate the uncertainty by considering a specific scenario, this scenario is a soft constraint problem which can be solved with off the shelf algorithms. Thus, if we are operating in a context where a pessimistic approach is useful, we can obtain the optimal solutions of the worst scenario and we are sure that such solutions outperform the other assignments in that scenario.

We then pass to more general notions of optimality, which do not refer to intervals but to more general ideas that apply whenever we have several scenarios to consider. Similarly to the case of ISCSPs, we consider necessarily optimal solutions, which are optimal in all scenarios, or possibly optimal

solutions, which are optimal in at least one scenario. We also consider solutions that guarantee a certain level of preference in all (resp., some) scenarios, and we aim to find those that guarantee the highest level. By relating these general notions of optimal solutions to the specific ones based on intervals, we are then able to provide algorithms to find or test optimal solutions according to these notions. We also show that a level of precision greater than a single interval does not add useful information when looking for an optimal solution.

We test our algorithms on a version of the meeting scheduling problem [65] which contains preference intervals. In our experiments we show the execution time to find all the kinds of optimal solutions we defined. The necessary optimal solutions are certainly the most attractive, as they are the best ones in every scenario, but they need more time to be found (if they exist) than other notions. Thus, in some cases, for instance in time-critical situations, one can look for solutions that guarantee a preference level in all scenarios and that are faster to be found.

### Matching problems

Furthermore, we consider matching problems, where the concept of incompleteness and imprecision in preference representation is naturally allowed. For instance, a generalization of the Stable Marriage problem (SM) [26, 36, 47] models unacceptability of certain men by allowing women to give incomplete preference lists (and vice versa) [45, 54]. Another widely used generalization of the SM problem, that usually cohexists with incomplete lists, is the use of ties in preference lists to express some kind of imprecision or indifference between two or more options [45, 54].

We start by studying the classical stable marriage problem. It is known that a stable marriage can be found in polynomial time by applying the Gale-Shapley algorithm [26]. The drawback of this algorithm is that it has a bias towards one gender. More precisely, it finds two specific stable marriages, called male and female optimal, that favour one gender over the other one. Male-optimality (and also female-optimality) may be considered too much unfair between the two genders: although stability is assured, only one of the genders is as happy as possible. The set of all stable marriages for a given instance of the classical stable marriage problem forms a distributive lattice with the male and female optimal stable marriages representing the two extreme elements of the lattice [47].

We investigate fairness of stable marriage procedures by considering a local search approach that finds a random marriage in the stable marriage lattice. We start from a randomly generated marriage and we pass from a

marriage to another, hopefully closer to stability, by marrying a man and a woman that would prefer each other to their current partners. The process terminates when a stable marriage is reached. This simple algorithm has been proven to be effective in both speed (it takes very few steps to reach stability) and in sampling the lattice of all stable marriages of a given problem instance.

After having verified the usefulness of our local search algorithms applied on the classical problem, we move further to consider stable marriage problems with ties and incomplete lists (SMTIs). In [54] Manlove et al. show that, given an instance of an SMTI problem, it may have stable marriages with different sizes. Hence, the goal is to find a maximal cardinality stable marriage and so the problem becomes an optimization task. Moreover, it is formally proved that this task is NP-hard [54]. Using our experience on the classical problem, we provide a local search algorithm for SMTI problems which is both fast and effective at finding maximal stable marriages for large problems. The algorithm is in fact able to obtain a very good solution after a small amount of steps. We then propose a different local search method that initial breaks ties of the given SMTI obtaining a stable marriage problem with incomplete lists (SMI). Then, it passes from an SMI to another by swapping one tie of the initial SMTI problem. Also this algorithm has shown good performance.

Finally, we consider male optimality and uniqueness of stable marriages with ties. We give an algorithm to find stable marriages that are male optimal and we provide sufficient conditions on the preferences which guarantee the uniqueness of a stable marriage.

## 1.3 Publications

Most of the results described in this thesis are contained, although in a shorter form, in the following papers:

- Journal papers:
  - *Elicitation Strategies for Soft Constraint Problems with Missing Preferences: Properties, Algorithms and Experimental Studies*, Mirco Gelain, Maria Silvia Pini, Francesca Rossi, K. Brent Venable, Toby Walsh, Artificial Intelligence Journal, vol. 174, issues 3-4, March 2010, Elsevier.
  - *Interval-valued Soft Constraint Problems*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and N. Wilson, Annals of Mathematics and

Artificial Intelligence, special issue for ISAIM 2008, B. Chouery and B. Givan eds., Springer, to appear.

- Conference papers:
  - *Dealing with Incomplete Preferences in Soft Constraint Problems*, Mirco Gelain, Maria Silvia Pini, Francesca Rossi, and K. Brent Venable, in proc. of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), Springer Verlag, LNCS 4741, 2007.
  - *Elicitation strategies for fuzzy constraint problems with missing preferences: algorithms and experimental studies*, Mirco Gelain, Maria Silvia Pini, Francesca Rossi, K. Brent Venable, and Toby Walsh, in proc. of the 14th International Conference on Principles and Practice of Constraint Programming (CP 2008), Springer Verlag, LNCS 5202, 2008.
  - *Male optimality and uniqueness in stable matching problems with partial orders*, Maria Silvia Pini, Francesca Rossi, Toby Walsh, Mirco Gelain, Kristen Brent Venable, in proc. of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), IFAAMAS Press, 2010, short paper.
  - *Local search algorithms on the Stable Marriage Problem: Experimental Studies*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 19th European Conference on Artificial Intelligence (ECAI 2010), IOS Press, 2010, short paper.
  - *Local search for stable marriage problems with ties and incomplete lists*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 11th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2010), Byoung-Tak Zhang and Mehmet A. Orgun eds., Springer, LNCS 6230, 2010.
  - *Local search approach to solve incomplete fuzzy and weighted CSPs*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011), poster paper, to appear.
  - *Male optimal and unique stable marriages with partially ordered preferences*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh, in proc. of the International Workshop on Collaborative Agents - REsearch and development (CARE 2009/2010), Springer LNAI 6066, to appear.

- Workshop papers:
  - *Imprecise Soft Constraint Problems*, Mirco Gelain, Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable and Nic Wilson, in proc. of the 9th Workshop on Preferences and Soft Constraints (SofT'08), Sydney, Australia, September 2008.
  - *Male optimal and unique stable marriages with partially ordered preferences*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh, in proc. of the International Workshop on Collaborative Agents – REsearch and Development (CARE 2009), Melbourne, Australia, December 2009.
  - *Local search approach to solve incomplete fuzzy and weighted CSPs*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 10th Workshop on Preferences and Soft Constraints (SofT'10), St. Andrews, Scotland, September 2010.
  - *Local search for stable marriage problems*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 3rd International Workshop on Computational Social Choice (COMSOC 2010), Dusseldorf, Germany, September 2010.

## 1.4 Structure of the thesis

The thesis is organized as follows. In Chapter 2 we introduce the basic notions to better understand the content of the thesis. In particular, we review the fundamental aspects of soft constraint problems, stable marriage problems, and search. In Chapter 3, we present an extension of the soft constraint framework to deal with incompleteness. After characterizing new notions of optimality, we provide solving methods, based on both systematic and local search, to find solutions that are optimal according to the new notions. An experimental study comparing the different methods concludes the chapter. Imprecision in soft constraint is the focus of Chapter 4, in which we propose a new formalism which models imprecision via intervals. We define new optimality notions based on such intervals and we propose algorithms to find solutions which are optimal according to the defined criteria. We also provide an experimental evaluation on randomly generated meeting scheduling problems where we allow for imprecision. We then turn our attention to the more specific class of stable marriage problems (Chapter 5). We start by considering the standard version of the problem and we show how a local search approach can be effectively used to fairly generate a stable marriage. In Chapter 6 we move to a more general setting where the stable marriage

problem allows for both incomplete preference lists and ties. In this context, where marriages may have different sizes, we provide a local search method to find a stable marriage of maximal cardinality. Such a method is also experimentally evaluated on randomly generated problems. Another interesting question in this context is the existence of a stable marriage maximally satisfying one of the genders. On this issue we provide both theoretical and experimental results. We conclude this thesis summarizing the results and suggesting new lines for future research (Chapter 7).

# Chapter 2

# Background

In this chapter we introduce some background notions which will be useful to understand the remainder of the thesis. In particular, we will first introduce soft constraints. Moreover, we will present stable marriage problems, showing the role of preferences in such problems. Finally, we will give some notions of the main search techniques we will use in the remainder of the thesis.

## 2.1 Soft constraints satisfaction problems

Constraints [61, 17] are useful to model real-life problems when it is clear what should be accepted and what should be forbidden. For example, constraint problems are used to model cognitive tasks in vision, language comprehension, default reasoning, diagnosis, scheduling, temporal and spatial reasoning. *Constraint Programming* (CP) is the study of computational systems based on constraints. In CP a problem is represented with a set of variables, each with a domain of values, and a set of constraints. More formally, a *Constraint Satisfaction Problem* CSP is a tuple $< X, D, C >$ where $X = x_1, .., x_n$ is the set of variables, $D = \{D_1, ..., D_n\}$ is the set of variables' domains and $C = \{c_1, ..., c_m\}$ is the set of constrains. Each constraint $c_i$ involves a subset of the variables, called the *scope* of the constraint, and specifies the allowable combinations of values for that subset.
A solution of a CSP is an assignment to all variables consistent with all constraints. Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used techniques are variants of constraint propagation, systematic search algorithm (see Section 2.3), and local search (see Section 2.4).
Several modifications of the basic CSP definition have been proposed to adapt the model to a wide variety of problems, for instance *Dynamic CSPs* [56] are

useful when the original formulation of a problem is altered in some way, typically because the set of constraints to consider evolves because of the environment. Furthermore, in *Distributed CSPs* [19], a set of variables is distributed among agents. The variables are connected by constraints which define the constraints network among the agents. As a result, the search algorithm for solving these problems is a distributed algorithm. Another specialization of CSPs are the Temporal CSPs [48] which are useful for representing and answering queries about temporal occurrences and temporal relations between them.
There are real life scenarios that need a more flexible way to be modeled than standard constraints can do. Such scenarios are, for example, situations where the knowledge is not precise or where it is not complete, or situation where the preferences of the user are a central part of the problem. For these reasons, the formalism of CSPs can be naturally extended to represent not only the allowed subset of values the variables can take but, also, which is the degree of preference they are permitted.
In general, it is difficult to express the users' specification with classical constraints and so, in many cases, his requirements lead to an over-constrained problem. If the problem is over-constrained, the user has to relax some constraints, permitting some solutions. There are other situations in which it is not enough to have any solution, but we might need a solution that is "better" than others. Soft constraints [7, 8, 6] provide a mechanism to establish which constrains has to be relaxed, or which constraints are more important than others, and provide a metrics to rank the solutions.

In all cases where the expressiveness of classical constraints is not enough and/or when we want to find an optimal solution (or the best ones), soft constraints can be useful. A Soft Constraint Satisfaction Problem [7] (soft-CSP) is a CSP with preference (or weight, or cost) values associated to each constraint taken from a partially or totally ordered set. There are many approaches to soft constraints. In valued CSP [24] a value, chosen among the elements of a totally ordered set, is associated to each constraint; in Probabilistic CSP [23], each constraint has an associated probability that has to be maximized; in Fuzzy CSP [64], each tuple (so, not the entire constraint) has an associated value between 0 and 1, and the aim is to maximize the value of a complete assignment.
Assigning preference values to each constraints' tuple [7] (and also to domain values of a variable that can be seen as unary constraints on the variable itself), needs a way to *combine* preferences of an assignment to compute a global preference. It is also necessary a method to *compare* two preferences, for example to decide if a solution is better than another one. To this end, Bistarelli, Montanari, Rossi in [7] introduced a formalism based on

c-semirings, which are sets with two operators, one used to combine preferences and the other used to compare them.

**Definition 1** (semiring and c-semiring)**.** *A semiring is a tuple* $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *such that:*

- $A$ *is a set and* $\mathbf{0}, \mathbf{1} \in A$*;*
- $+$ *is commutative, associative and* $\mathbf{0} + a = a = a + \mathbf{0}$ *where* $a \in A$*, i.e.* $\mathbf{0}$ *is the neutral element;*
- $\times$ *is associative, distributes over* $+$*,* $\mathbf{1}$ *is the neutral element and* $\mathbf{0}$ *is the absorbing element (* $a \times \mathbf{0} = \mathbf{0} = \mathbf{0} \times a$*).*

*A c-semiring is a semiring* $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *such that:*

- $+$ *is defined over (possibly infinite) sets of elements of* $A$ *as follows:*
  - *for all* $a$ *which are elements of* $A$*,* $+(\{a\}) = a$*;*
  - $+(\emptyset) = \mathbf{0}$ *and* $+(A) = \mathbf{1}$*;*
  - $+(\cup A_i, i \in S) = +(\{+(A_i), i \in S\})$ *for all sets of indices of* $S$ *(flattening property).*
- $\times$ *is commutative.*

Consider the relation $\leq_S$ on $A$ such that, for $a, b \in A$, $a \leq_S b$ if and only if $a + b = b$. Is it possible to show that:

- $\leq_S$ is a partial order;
- $+$ and $\times$ are monotone over $\leq_S$;
- its minimum $\mathbf{0}$ and its maximum is $\mathbf{1}$;
- $\langle A, \leq_S\rangle$ is a complete lattice and, for every $a, b \in A$, $a + b = lub(a, b)$.

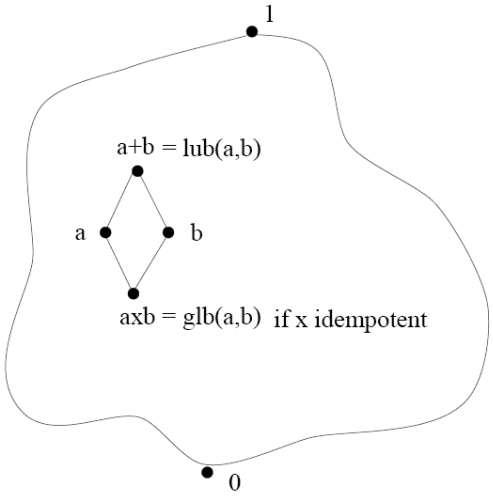



Figure 2.1: c-semiring

Furthermore, if $\times$ is idempotent, $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ is its greatest lower bound (glb). Informally speaking, the relation $\leq_S$ gives a way to compare two preferences, in fact if $a \leq_S b$ then $b$ is better than $a$. The situation is showed in Figure 2.1.

Starting from the definition of the c-semiring, in [7] they defined in a parametric way the notion of *Constraint System* (CS), constraint and constraint problem.

**Definition 2** (constraint system)**.** *A Constraint System is a tuple* $CS = \langle S, D, V \rangle$ *where,* $S$ *is a c-semiring,* $D$ *is a finite set and* $V$ *is an ordered set of variables.*

A constraint in a CS specify the involved variables and their allowed values. Each tuple of values (in $D$) of involved variables in the constraint (i.e. variables in the scope of the constraint) has an associated element $a \in A$ (the carrier of the $S$). This element can be considered as a weight of the tuple, a preference, a cost, a level of confidence or other. The meaning assigned to those elements will define, as we will see, the choice of the c-semiring to be used.

**Definition 3** (soft constraints)**.** *Given a constraint system* $CS = \langle S, D, V \rangle$*, where* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ *is a c-semiring, a soft constraint is a couple* $\langle def, con \rangle$ *where* $con \subseteq V$ *and* $def : D^{|con|} \rightarrow A$*.*

Therefore a soft constraint problem is a set of soft constraints in a constraint system, plus a selected set of variables (the variables of the problem)

of which we want to find an assignment compatible with all constraints.

**Definition 4** (semiring-based CSP)**.** *Given a constraint system* $CS = \langle S, D, V\rangle$*, a semiring-based CSP (SCSP) on* $CS$ *is a couple* $P = \langle C, con\rangle$*, where* $C$ *is a set of constraints on* $CS$*, and* $con \subseteq V$*. We assume also that, if* $\langle def_1, con'\rangle \in C$ *and* $\langle def_2, con'\rangle \in C$ *then* $def_1 = def_2$*.*

In a SCSP, the values specified for the tuple of each constraint are used to compute corresponding values for the tuples of values of the variables in *con*, according with the c-semiring operations: the multiplicative operator ($\times$) is used to combine the values of the tuples of each constraint to get the value of a tuple for all the variables, and the additive operator ($+$) is used to obtain the value of the tuples of the variables of interest. More precisely, we can define the operations of *combination* $\otimes$ and *projection* $\Downarrow$ over constraints.

**Definition 5** (combination)**.** *Given two constraints* $c_1 = \langle def_1, con_1\rangle$ *and* $c_2 = \langle def_2, con_2\rangle$*, their combination, written* $c_1 \otimes c_2$*, is the constraint* $c = \langle def, con\rangle$*, with* $con = con_1 \cup con_2$ *and* $def(t) = def_1(t \downarrow_{con_1}^{con}) \times def_2(t \downarrow_{con_2}^{con})$[1]*.*

The combination operator $\otimes$ can be easily extended to more than to arguments, say $C = c_1, ..., c_m$, by performing $c_1 \otimes c_2 \otimes ... \otimes c_m$, which will be sometimes denote by $\bigotimes C$.

**Definition 6** (projection)**.** *Given a constraint* $c = \langle def, con\rangle$ *and a subset* $I$ *of* $V$*, the projection of* $c$ *over* $I$*, written* $c \Downarrow_I$*, is the constraint* $\langle def', con'\rangle$ *with* $con' = con \cap I$ *and* $def'(t') = \sum_{t|t\downarrow_{I\cap con}^{con} = t'} def(t)$*.*

Now, using the previous definitions, we can define the notion of solution of an SCSP.

**Definition 7** (solution)**.** *Given an* $SCSPP = \langle C, con\rangle$ *over a constrain system* $CS$*, the solution of* $P$ *is a constraint defined as* $Sol(P) = (\bigotimes C) \Downarrow_c$ *on.*

So, a solution of a SCSP is a the constraint induced on the variables in *con* by the whole problem. A solution is also a complete assignment, i.e., an assignment to all the variable in the problem.

**Definition 8** (preference of an assignment)**.** *Given an assignment* $s$ *to all the variables of an SCSP* $P$*, its preference is defined as:* $pref(P, s) = \Pi_{\langle def, con\rangle \in C} def(s_{\downarrow con})$ *where* $\Pi$ *refers to the* $\times$ *operation of the c-semiring and* $s_{\downarrow con}$ *is the projection of tuple* $s$ *on the variables in con.*

[1]With $t \downarrow_Y^X$ we denote the projection of a tuple $t$, defined over a set of variables $X$, over a subset of variables $Y \subseteq X$.

In words, $pref(P, s)$, is obtained by combining the preferences associated by each constraint to the subtuples of $s$ referring to the variables of the constraint.

**Definition 9** (optimal solution)**.** *Given an SCSP $P$ and an assignment $s$ to all the variables in $P$, $s$ is an optimal solution if there is no other complete assignment $s'$ with $pref(P, s) <_S pref(P, s')$. The set of optimal solutions of an SCSP $P$ will be written as $Opt(P)$.*

Many classes of constraint satisfaction or optimization problems can be represented using the c-semiring framework, by choosing the appropriate operators:

**CSPs** To model the classical CSP framework the s-semiring $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$ can be used.
The only two preferences that can be assigned are $false$ and $true$ to denote a disallowed and allowed tuple respectively. Preferences are combined with the logic operator *and* and compared with the logic operator *or*. Each complete assignment with a preference equal to $true$ is optimal.

**fuzzy CSPs** To model the classical CSP framework the s-semiring $S_{FCSP} = \langle [0, 1], max, min, 0, 1 \rangle$ can be used.
Preferences ranges from 0 to 1, are combined using $min$ and compare using $max$ operator. The goal is to maximize the minimal preference.

**weighted CSPs** This framework can be seen as the following instance of the c-semiring framework: $S_{WCSP} = \langle \Re^+, min, +, +\infty, 0 \rangle$.
The values of the carrier are interpreted as costs and are real numbers starting from 0 to $+\infty$. Since we want to minimaze the global cost, we combine them with the $sum$ and compare them with $min$.

**probabilistic CSPs** In this case the appropriate c-semiring is $S_{PCSP} = \langle [0, 1], max, \times, 0, 1 \rangle$.
Preferences are interpreted as probability values from 0 to 1 and so are combined using the usual multiplication over reals and compared using $max$. The aim is to maximize the joint probability.

Figure 2.2 shows an example of an SCSP based on the fuzzy c-semiring. The problem is depicted as a graph where nodes are the variables and edges represents the constraints. Each constraint tuple has an associated preference in $[0, 1]$. The possible assignments are:
$< a,\ a >$ with preference $min(0.9,\ 0.8,\ 0.9) = 0.8$;

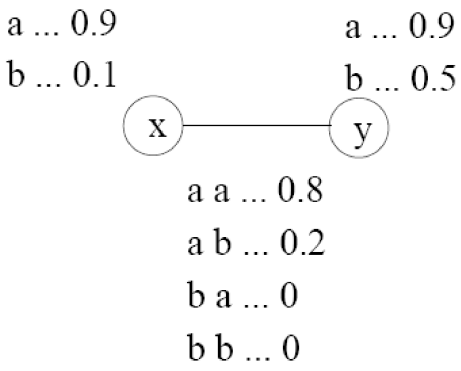



Figure 2.2: A fuzzy CSP.

$< a,\ b >$ with preference $min(0.9,\ 0.2,\ 0.5) = 0.2$;
$< b,\ a >$ with preference $min(0.1,\ 0,\ 0.9) = 0$;
$< b,\ b >$ with preference $min(0.1,\ 0,\ 0.5) = 0$.
The optimal solution is the assignment with maximal preference that is $< a, a >$.

## 2.2 Stable marriage problems

A classical mathematical problem in which preferences play a central role is the *stable marriage problem* (SM), that was first introduced by D. Gale and L. S. Shapley in [26]. Possible practical applications of SM are the assignment of medical students to hospitals (for internships), where students list hospitals in order of preference and hospitals do the same with the students. Another application is helping negotiation processes in electronic commerce. The negotiation processes can be viewed as the process where a number of autonomous agents carry out different tasks, and the concept of stable marriage can be applied when the coordination and stability of the system becomes critical.
The stable marriage problem is formally defined as follows: a certain community consists in $n$ men and $n$ women ($n$ is called the size of the problem). Each person ranks the persons of opposite sex in accordance to his/her preferences for a marriage partner. The goal is to find a satisfactory way of marrying off all the members of the community. From now, with marriage we mean a set of $n$ marriages $(m, w)$ where $m$ is a man and $w$ is a woman.

**Definition 10** (profile)**.** *Given $n$ men and $n$ women, a profile is a sequence of $2n$ strict total orders (i.e., transitive and complete binary relations), $n$*

*over the men and n over the women.*

**Definition 11** (marriage)**.** *Given an SM P of size n, a marriage M is a one-to-one matching of the men and the women. If a man m and a woman w are matched in M, we write $M(m) = w$ and $M(w) = m$.*

The following definitions can clarify the the concept of a "satisfactory" marriage.

**Definition 12** (blocking pair)**.** *Given a marriage M, a pair $(m, w)$, where m is a man and w is a woman, is a blocking pair if m and w are not partners in M, but m prefers w to $M(m)$ and w prefers m to $M(w)$.*

**Definition 13** (stable marriage)**.** *A marriage M is stable if it has no blocking pairs.*

**Definition 14** (feasible partner)**.** *Given an SM P, a feasible partner for a man m (resp., a woman w) is a woman w (resp., a man m) such that there is a stable marriage for P where m and w are married.*

Consider the following example:

| men's lists | women's lists |
|---|---|
| 1: 1 2 3 | 1: 2 3 1 |
| 2: 2 3 1 | 2: 3 1 2 |
| 3: 3 1 2 | 3: 1 2 3 |

Man $m_1$ ranks $w_1$ first, $w_2$ second and $w_3$ third, while $w_1$ ranks $m_2$ first, $m_3$ second, and $m_1$ third, etc.
In the example, there are six possible marriages, three of these are stable. One of these is: 3 1 2. With this notation we intend that the woman in first position ($w_3$) is married with $m_1$, the woman in second position is married with $m_2$ and so on, the woman in $i^{th}$ position is married with $m_i$. The other two stable marriages are: $[1, 2, 3]$ and $[2, 3, 1]$. All the other marriages are not stable.

Now, the first question that arise is: *"For any pattern of preferences, is it possible to find a stable marriage?"*. The following theorem answer this question.

**Theorem 1** ([26])**.** *There always exists a stable marriage.*

The Gale-Shapley [26] (see GS algorithm) is an iterative procedure that finds a stable marriage. At the start of the algorithm, each person is free and becomes engaged during the execution of the algorithm. Once a woman is engaged, she never becomes free again (although to whom she is engaged may change), but men can alternate between being free and being engaged. The following step is iterated until all men are engaged: choose a free man $m$, and let $m$ propose to the most preferred woman $w$ on his preference list, such that $w$ has not already rejected $m$. If $w$ is free, then $w$ and $m$ become engaged. If $w$ is engaged to man $m$', then she rejects the man ($m$ or $m$') that she least prefers, and becomes, or remains, engaged to the other man. The rejected man becomes, or remains, free. When all men are engaged, the engaged pairs are a male optimal stable marriage.

This algorithm needs a number of steps that is quadratic in $n$ (that is, the number of men), and it guarantees that, if the number of men and women coincide, and all participants express a strict order over all the members of the other group, everyone gets married, and the returned marriage is stable. Since the input includes the profiles, the algorithm is linear in the size of the input.

**Algorithm 1**: GS algorithm

```
Input: a set of men M; a set of women W;
Output: a stable matching
Initialize all m ∈ M and w ∈ W to free;
while ∃ free man m do
    w ← m's highest ranked such woman;
    if w is free then
        (m, w) become engaged
    else
        some pair (m', w) already exists;
        if w prefers m to m' then
            (m, w) become engaged;
            m' becomes free;
        else
            (m', w) remain engaged;
```

This algorithm guarantees that:

- If the number of men and women coincide, and all participants express a linear order over all the members of the other group, everyone gets married.

- The marriages are stable.

The *Extended Gale-Shapely algorithm* [36] is a version of the GS algorithm [26] where, whenever the proposal of a man $m$ to a woman $w$ is accepted, in $w$'s preference list all men less desirable than $m$ are deleted, and $w$ is deleted from the preference lists of all such men. This means that, every time that a woman receives a proposal from a man, she accepts since only most preferred men can propose to her.

Obviously, nothing prevent us to swap the roles of the two groups, with women that propose to men. If we run the GS algorithm swapping the genders to the same problem the results will not generally be the same, more precisely where men propose to the women the result will be optimal for the men, when the women propose, it is optimal for them. So the algorithm returns not only a stable marriage, but a marriage that is also optimal for applicants.

**Definition 15** (male (resp., female) optimal marriage)**.** *Given an SM $P$, a marriage is male (resp., female) optimal iff every man (resp., woman) is paired with his (resp., her) highest ranked feasible partner in $P$. We write $M_m$ (resp., $M_w$) for the male (resp., female) optimal.*

For a given SM instance, we can define a partial order relation on the set of stable marriages.

**Definition 16** (dominance)**.** *Let $M$ and $M'$ two stable marriages. $M$ dominates $M'$ if every man has a partner in $M$ which is at least as good as the one he has in $M'$.*

It is known that the set of stable marriages forms a distributive lattice under this partial order and that the male- (resp. female-) optimal marriage is the top (resp. bottom) of this lattice [36]. A clear way to represent this lattice is a Hasse diagram representing the transitive reduction of the partial order relation. Such a diagram can be easily built using *exposed rotations* [41], which are cyclic sequences of pairs $(m_1, w_1), ..., (m_k, w_k)$ used to define a spouse swapping from one stable marriage to another.

**Definition 17** (successor)**.** *Let $M$ be a stable marriage of a given SM instance. Then, for each man $m$ such that $M(m) \neq M_w(m)$, the* successor *woman of $m$ relative to $M$, denoted $s_M(m)$ is the first woman $w$ in $m$'s preference list, following $M(m)$, such that $w$ prefers $m$ to her current partner in $M$ (i.e. $M(w)$).*

| men's preference lists | women's preference lists |
|---|---|
| 1: 5 7 1 2 6 8 4 3 | 1: 5 3 7 6 1 2 8 4 |
| 2: 2 3 7 5 4 1 8 6 | 2: 8 6 3 5 7 2 1 4 |
| 3: 8 5 1 4 6 2 3 7 | 3: 1 5 6 2 4 8 7 3 |
| 4: 3 2 7 4 1 6 8 5 | 4: 8 7 3 2 4 1 5 6 |
| 5: 7 2 5 1 3 6 8 4 | 5: 6 4 7 3 8 1 2 5 |
| 6: 1 6 7 5 8 4 2 3 | 6: 2 8 5 4 6 3 7 1 |
| 7: 2 5 7 6 3 4 8 1 | 7: 7 5 2 1 8 6 4 3 |
| 8: 3 8 4 5 7 2 6 1 | 8: 7 4 1 5 2 3 6 8 |

Table 2.1: An example of an SM of size 8.

**Definition 18** (rotation)**.** *Let $M$ be a stable marriage of a given SM instance. A* rotation exposed *in $M$ is a cyclic sequence of pairs $\rho = (m_0, w_0), ..., (m_{r-1}, w_{r-1})$ such that $M(m_i) = w_i$ and $s_M(m_i) = w_{i+1}$ for all $i$, $i+1$ taken modulo $r$.*

When we pair $m_i$ with $w_{i+1}$ for all $m_i$ ($i+1$ taken modulo $r$ as usual) in the exposed rotation $\rho$ and otherwise leave the pairs unchanged, we say that we *eliminate* $\rho$ from $M$. By eliminating $\rho$ from $M$ we obtain a new marriage $M/\rho$. These rotations are called *woman-improving rotation* because, after eliminating such a rotation, each woman has the same partner or she is married with a man she prefers to the previous one. *man-improving rotations* are defined similarly by exchanging genders in all of these definitions. Moreover, it can be shown that, after performing a rotation improving one gender, the inverse is a rotation improving the other. In the following, we will use the term rotation to indicate woman-improving rotations.

Figure 2.3 shows the Hasse diagram of the example in Table 2.1.

As before, the number in position $i$ of each list of numbers indicates the woman married to man $i$ in that marriage. For example, in the marriage 5 3 8 6 7 1 2 4, man 1 is married with woman 5, man 2 with woman 3, man 3 with woman 8, and so on. Each edge indicates that the marriage above dominates the one below the edge. For example, marriage 5 3 8 6 7 1 2 4 (which is also the top of the lattice) dominates 8 3 5 6 7 1 2 4. Thus, by transitivity, considering a path from the top to the bottom of the lattice, each marriage in the path dominates the ones below it in the same path. It is known that any path from the top to the bottom of the stable marriage lattice has the same length [35].

Moreover, each edge of the marriage lattice correspond exactly to allowable rotations [36]. For example, to pass from 8 3 1 6 7 5 2 4 to 3 6 1 8 7 5

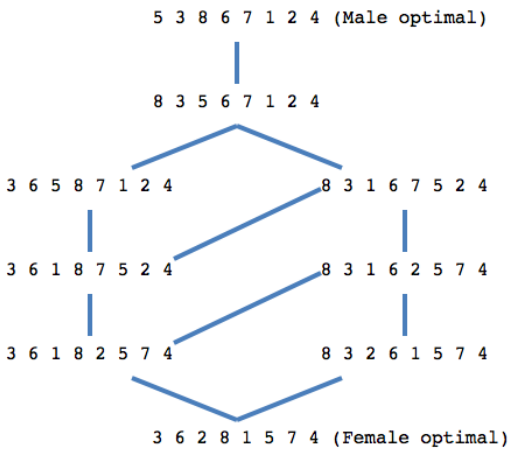



Figure 2.3: The Hasse diagram of the set of all stable marriages for the SM in Table 2.1.

2 4, rotation $(m_1, w_8)$, $(m_2, w_3)$ has to be eliminated.

**Definition 19** (rotation precedence [40])**.** *Let $\pi$ and $\rho$ two distinct rotations. Rotation $\pi$ precedes $\rho$ if, in order to obtain a stable marriage in which $\rho$ is exposed, rotation $\pi$ must be eliminated.*

The set of rotations under the partial order, induced above by Definition 19, is known as *rotation poset*. A subset $S$ of the rotation poset whit the property that $\rho \in S$, $\delta \prec \rho \Rightarrow \delta \in S$ is called *closed*.

**Theorem 2** ([40])**.** *For a given SM instance, there is a one-to-one correspondence between the stable marriages and the closed subset of the rotation poset.*

Rotations and Hasse diagram of the stable marriage lattice have some interesting properties:

1. Every stable matching $M \neq M_w$ has at least one rotation [41].
2. Every stable matching can be obtained starting from the male-optimal and successively eliminating a sequence of exposed rotations. Each stable matching is characterized by the set of rotations that must be eliminated to reach it [40].
3. In any path $P$ in the Hasse diagram $H$ from the man-optimal to the female-optimal marriage, two consecutive marriages differ by a single rotation, and the set of the rotations between marriages along $P$ contains all rotations exactly once. This means that all the paths in H

from the man-optimal to the woman-optimal marriage have the same length [35].

## Variants of the stable marriage problem

There are many variants of the stable marriage problem. Some of the most widely used present either incompleteness or imprecision, or both.

### SMs with incomplete preference lists

The stable marriage problem with incomplete lists (SMI) [36, 44, 39] is a generalization the classical stable marriage problem. In SMIs, each person may rank only a few member of the other group so, a person $p$ is acceptable to another person $q$ if $p$ is in the preference list of $q$, otherwise it is not acceptable. In this case, incompleteness is used to model the unacceptability of one on more element of one set by one element of the opposite set. Given an instance I of an SMI, a matching is a one-to-one correspondence between a subset of men and a subset of women such that each person finds the other acceptable. Given the concept of unacceptability, the definition of blocking pair has to be revised.

**Definition 20** (blocking pair in a SMI)**.** *A blocking pair in a SMI $M$ is a pair $(m, w)$, each of whom is either unmatched in $M$ and finds the other acceptable, or prefers the other to his/her partner in $M$.*

Since a stable matching in I has no blocking pair, it need not be a complete matching. Moreover, all stable matchings in I have the same size, and involve exactly the same men and women [27].

The GS algorithm can be easily modified to deal with incomplete lists leading to a polynomial-time algorithm to find a stable matching [36]. In fact, it is only need to change the condition of while loop to deal with incomplete lists as follows: *"while exists a free man who still has a woman to propose to do"*.

### SMs with ties

Another widely studied relaxation of the stable marriage problem admits ties in the preference lists (SMT) [36] to model some sort of imprecision. In fact, the user may have no idea of the precise ranking of the elements of the opposite group and so, he do not need to rank all members in strict order.

Because of ties, the definition of blocking pairs as to be changed accordingly.

| men's preference lists | women's preference lists |
|---|---|
| 1: 2 1 | 1: 3 1 (2 4) |
| 2: 2 (3 4) | 2: 1 4 2 |
| 3: (1 2 3 4) | 3: (1 2) (4 3) |
| 4: (3 2) 1 4 | 4: (3 2 4) |

Table 2.2: An example of a SMTI problem of size 4.

**Definition 21** (blocking pair in a SMT)**.** *A blocking pair in a SMT $M$ is a pair $(m, w)$, each of whom strictly prefers the other to his/her partner in $M$.*

Also in this context, a marriage $M$ is said to be stable if it does not have blocking pairs in accordance with Definition 21. This definition (which we will use) of stability is referred in [39] as *weak stability*. Other stronger notions of stability are used in the context of SMTs [39]. A matching is *strongly stable* if there is no couple $(m, w)$ such that $m$ strictly prefers $w$ to his/her partner, and $w$ either strictly prefers $m$ to his/her partner or is indifferent between them. A matching is *super-stable* if there is no couple each of whom either strictly prefers the other to his/her partner or is indifferent between them.

SMTs are not significantly difficult problems and it is known (by Gusfield and Irwing [36]) that there exists a polynomial-time algorithm to find a stable matching. In fact, by breaking ties in an arbitrary way we can transform an instance $I$ of an SMT in an instance $I'$ of an SM and then we can apply the GS algorithm to $I'$. Clearly, a stable marriage in $I'$ is also stable in $I$ and the all the stable marriages in $I$ have the same size.

### SMs with both ties and incomplete lists

An SMTI is an SM with both ties and incomplete lists. An example of a SMTI problem with four men and women is shown in Table 2.2, where preference lists may contain less than four elements and brackets are used to indicate ties. For instance, by writing $2 : 2\ (3\ 4)$ among the men's preference lists we mean that man $m_2$ strictly prefers woman $w_2$ to women $w_3$ and $w_4$, that are equally preferred.

By allowing ties and incomplete lists to the SM, we have to update the definition of blocking pair to formalize the concept of stability in SMTIs (Stable Marriage Problems with Incomplete lists and Ties) [36, 45, 30].

**Definition 22** (blocking pair in a SMTI)**.** *A blocking pair in a SMT $M$ is a pair $(m, w)$, each of whom is either unmatched in $M$ and finds the other*

*acceptable, or strictly prefers the other to his/her partner in M.*

We know from the previous sections that, in a given instance of an SMI, all the stable marriages have the same size (involves the same persons but are not necessarily complete, i.e. not all persons may be matched). We also know that, in the SMT case, all marriages are complete (and therefore of the same size). However, we can not say the same for SMTIs. In fact, in [54] Manlove et al. show that a given SMTI instance may admit stable marriegs with different sizes and thus the problem is now to find a stable marriage with the maximum (or minimum) size.

SMTIs are proved to be NP-complete by Iwata et al.[45]. Manlove et al. [54] have shown that, for a given instance of an SMTI, finding a stable matching of maximum, or minimum, size is NP-hard, even in the case where the ties occur at the tails of lists and on one side only, there is at most one tie per list, and each tie is of length 2.

## 2.3 Systematic search

Solving a CSP means finding a consistent assignment to all the constraints. A naive search method is the generate-and-test method. This method generates a complete assignments and tests if all constraints are satisfied. If not, it generates another assignment and repeats the test until a either all constraints are satisfied (a solution is found) or all possible assignment are generated (the problem has no solution). Backtracking search [15, 25] instead incrementally attempts to extend a partial assignment that specifies consistent values for some of the variables, toward a complete assignment, by repeatedly choosing a value for another variable consistent with the values in the current partial solution. During the backtracking search, variables are instantiated sequentially and, as soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. If a partial assignment violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternatives available. The late detection of inconsistency is a disadvantage of the backtracking search technique. Several consistency notions can be used in backtracking algorithms to early detect inconsistencies. The most known are node consistency, arc consistency, and path consistency.

Local consistency notions [2] are properties of constraint satisfaction problems related to the consistency of subsets of variables or constraints, and may help in discovering inconsistencies easier. In general, local consistency notions require that every consistent assignment can be consistently extended to another variable. For example, a CSP is node-consistent if the domain values

of each variable are allowed by all the unary constraint on it. Arc-consistency instead says that a binary constraint is arc-consistent if every value in each domain has a *support* in the other domain; a value $a$ has a support value $b$ if the pair $(a, b)$ satisfies the constraint. For example, consider two variables $x$ and $y$ and their domains $D(x) = \{1, 5\}$ and $D(y) = \{-1, 2, 4\}$, and a binary constraint $x < y$. Value 1 for $x$ has support in the domain of $y$ (values 2 and 4 for $y$), but there is no support for $x = 5$ (and also for $y = -1$). So, after enforcing the arc-consistency we will obtain the following result: $x = \{1\}$ and $y = \{2, 4\}$. Constraint propagation for CSP has been extended and adapted to soft constraints in [55].

Consistency techniques are usually used in conjunction with backtracking. More precisely, a backtracking procedure traverse an implicit *search tree* where each internal node is a partial assignment and there is an edge from a node $x$ to another node $y$ if $y$ was created extending the partial assignment in $x$. Solutions are the leafs of the tree. Backtracking search performs a depth-first visit of the search tree. At each node $c$, the algorithm checks whether $c$ has some chance of being completed to a valid solution. If it cannot, the whole sub-tree rooted at $c$ is skipped (pruned). Otherwise, the algorithm checks whether $c$ itself is a valid solution, and if so reports it to the user; and recursively enumerates all sub-trees rooted at $c$. At each node of the search tree, a constraint propagation procedure may be ran on the current problem. Since it reduces the cardinality of the variable domains, it reduces the search space, avoiding some branches with no solutions.

A widely used constraint propagation method applicable to search tree algorithms is *forward checking*. Forward checking revises each domain of a future variable $x_i$ by taking into account the constraints involving $x_i$ with each already instantiated variable (called past variables) and with the current variable. For example, if there are two variables and their domains, $x = \{3, 4, 5, 6, 7\}$ and $y = \{2, 4, 5\}$, and a binary constraint $x > y$, and the current instantiation is $y = 5$, then forward checking will prune values 3, 4 and 5 from $x$'s domain.

The *Branch and Bound (BB)* [51] search algorithm uses a similar tree structure as backtracking, but it is used in optimization problems where an objective function needs to be maximized. BB is based on a heuristic function used to compute, at each node of the search tree, an upper bound of the value of the objective function for the leafs of the subtree rooted at the current node. The upper bound is used for determining if it is a promising partial assignment (if the bound is better than the value of the best solution found so far) or not. If we are in a promising node, the algorithm expands the subtree beyond the node. Otherwise, the subtree rooted at the current node is pruned from the search.

It is important to notice that, in all search tree based algorithms, whenever a domain is reduced, it have to be restored upon backtracking.

## 2.4 Local search

Local search [38, 67] is one of the fundamental paradigms for solving computationally hard combinatorial problems, including the constraint satisfaction problem. Local search methods in many cases represent the only feasible way for solving these large and complex instances. Moreover, they can naturally be used to solve optimization problems.

Given a problem instance, the basic idea underlying local search is to start at an *initial search position* (typically a randomly or heuristically generated candidate solution, which may be infeasible, sub-optimal or incomplete), and to iteratively improve this candidate solution by means of typically minor modifications. In each *search step* the search process moves to a position selected from the *local neighborhood* (typically based on a heuristic evaluation function). This process is iterated until a *termination criterion* is satisfied. Different local search methods varying in the way in which improvements are achieved, and in particular, in the way in which situations are handled when no direct improvement is possible. To ensure that the search process does not stagnate in unsatisfactory candidate solutions, most local search methods use randomization; they are called *stochastic local search (SLS) methods*. SLS are often conceptually simple, easy to implement and very flexible in that it can be very easily adapted to changes in the specification of the problem. This makes them a very popular choice for solving conceptually complex application problems that are sometimes not fully formalized at the beginning of a project.

The search space of almost all local search algorithms for CSP consists of all complete variable assignments of the given CSP instance, and the so-called *1-exchange neighborhood* is used. This neighborhood relation says that two complete variable assignments are direct neighbors if and only if they differs at most in the value assigned to one variable.

In most cases the initial position is determined generating uniformly a random variable assignment and the termination criterion is usually satisfied if a solution is found or if a predetermined number of steps is reached (other variants may stop the search after a predefined amount of time).

What differ from the various local search algorithm for CSPs (but also for SAT and other classes of combinatorial problems) is their step function, which is usually an heuristic guidance in the form of an *evaluation function*. This function typically maps the current candidate solution to a real number

such that its local minima correspond to a solution of the given problem instance. Hence, the evaluation function is used to select the most preferred neighbor. For example, in CSPs, the commonly used evaluation function, given a complete assignment, counts the number of violated constraints.

The simplest local search algorithm is called *Iterative Improvement* (also known as *hill climbing*) which, at each step, selects an improving position $a'$ from the set of neighbors $N(a)$, of a given assignment $a$, such that, for an evaluation function $f$, $f(a') < f(a)$. There are various commonly used heuristics for selecting such an improving neighborhood. In *Iterative best-improvement*, a neighbor $s'$ with minimal value of $f(s')$ within $N(s)$ is chosen. In *Iterative first-improvement*, on the other hand, the neighborhood is evaluated in a given order, and the first improving neighbor is selected as the next search position. The main limitation of the iterative improvement algorithms is that they can get stuck in a local minima of the given evaluation function. A simple way to avoid this problem is to select a non-improving search step. There are various techniques to achieve this, many of these uses a randomized decision to select a new search position.

*Randomized Iterative Improvement (RII)* is an extension of Iterative Improvement where in each step, with a fixed probability $p$, a new search position is selected uniformly at random from the current neighborhood; this is called *Random Walk step*. The usual local search step is performed with probability $(1 - p)$. When run arbitrarily long, RII will find a solution to any soluble problem instance with probability approaching one. Algorithms with this property are called *probabilistically approximately complete (PAC)*.

Another general heuristic method (also called *meta-heuristic*) which is applicable to a wide range of different combination problem, is *Tabu Search (TS)* [32]. The main idea of TS is to use a short-term memory to escape from local minima and typically uses an aggressive local search that in each step tries to make the best possible move to a neighbor solution even if that move worsens the evaluation function value. To prevent local search to immediately return to a previously visited solution and to avoid cycling, in TS moves to recently visited solutions are forbidden. This can be implemented by explicitly memorizing previously visited solutions and forbidding moving to those. More commonly, reversing recent search step is prevented by forbidding the re-introduction of solution components (such as assignments of individual CSP variables) which have just been removed from the current candidate solution. A parameter called *Tabu Tenure* determines the number of search steps for which these restrictions apply. As an undesirable side-effect, the tabu conditions may be too restrictive and they may forbid moves to attractive, unvisited solutions. Therefore, many tabu search algorithms make use of a so called *aspiration criterion*, that is used to override the tabu

status of certain moves and to avoid such situations. Most commonly, the aspiration criterion drops the tabu status of moves leading to a better solution than the best candidate solution seen throughout the search process.

Another possible extension of the Iterative Improvement algorithms to escape from a local minima of a given evaluation function is to modify the evaluation function when the search process is about to stagnate in a local minimum [53]. Penalty weights are associated to solutions components or to other features of solutions. Penalty algorithms modify these weights during the search following the intuition that some components of the solution are more important than other and so local minima might be eliminated.

# Chapter 3

# Incompleteness in soft constraints

In this chapter we consider semiring-based soft constraints [6, 7] and how preferences are modeled in this framework, with the aim to extend such a modeling to allow for missing preferences. In this context, we study how to find an optimal solution without having to wait for all the preferences. In particular, we define algorithms that interleave search and preference elicitation, to find such optimal solutions with the aim to elicit as few preferences as possible.

Some of the results in this chapter are included in the following articles:

- *Elicitation Strategies for Soft Constraint Problems with Missing Preferences: Properties, Algorithms and Experimental Studies*, Mirco Gelain, Maria Silvia Pini, Francesca Rossi, K. Brent Venable, Toby Walsh, Artificial Intelligence Journal, vol. 174, issues 3-4, March 2010, Elsevier.

- *Dealing with Incomplete Preferences in Soft Constraint Problems*, Mirco Gelain, Maria Silvia Pini, Francesca Rossi, and K. Brent Venable, in proc. of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), Springer Verlag, LNCS 4741, 2007.

- *Elicitation strategies for fuzzy constraint problems with missing preferences: algorithms and experimental studies*, Mirco Gelain, Maria Silvia Pini, Francesca Rossi, K. Brent Venable, and Toby Walsh, in proc. of the 14th International Conference on Principles and Practice of Constraint Programming (CP 2008), Springer Verlag, LNCS 5202, 2008.

- *Local search approach to solve incomplete fuzzy and weighted CSPs*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of

the 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011), poster paper, to appear.

- *Local search approach to solve incomplete fuzzy and weighted CSPs*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 10th Workshop on Preferences and Soft Constraints (SofT'10), St. Andrews, Scotland, September 2010.

## 3.1 Motivations

Traditionally, tasks such as scheduling, planning, and resource allocation have been tackled using several techniques, among which constraint reasoning is one of the most promising. The task is represented by a set of variables, their domains, and a set of constraints, and a solution of the problem is an assignment to all the variables in their domains such that all constraints are satisfied. Preferences or objective functions have been used to extend this formalism and allow for the modelling of constraint optimization, rather than satisfaction, problems. In all these approaches, the data (variables, domains, constraints) are completely known before the solving process starts. However, the increasing use of web services and in general of multi-agent applications demands for the formalization and handling of data that is only partially known when the solving process works, and that can be added later, for example via elicitation [70, 71]. In many web applications, data may come from different sources, which may provide their piece of information at different times. Also, in multi-agent settings, data provided by some agents may be voluntarily hidden due to privacy reasons, and only released if needed to find a solution to the problem.

Here we consider these issues focusing on constraint optimization problems where we look for an optimal solution. In particular, we consider problems where constraints are replaced by soft constraints, in which each assignment to the variables of the constraint has an associated preference coming from a preference set [7]. We assume that variables, domains, and constraint topology are given at the beginning, while the preferences are partially specified and are elicited during the solving process.

There are several application domains where this might be useful. One regards the fact that quantitative preferences, needed in soft constraints, may be difficult and tedious to provide for a user. Another one concerns multi-agent settings, where agents agree on the structures of the problem but they may provide their preferences on different parts of the problem at different times. Finally, some preferences can be initially hidden because of privacy

reasons.

Formally, we take the soft constraint formalism and we allow for some preferences to be left unspecified. In our setting, users may know all the preferences but are willing to reveal only some of them at the beginning. Although some of the preferences can be missing, it could still be feasible to find an optimal solution. If not, we ask the user to provide some of the missing preferences and we start again from the new problem. We consider two notions of optimal solution: *possibly optimal* solutions are assignments to all the variables that are optimal in *at least one way* in which the currently unspecified preferences can be revealed, while *necessarily optimal* solutions are assignments to all the variables that are optimal in *all ways* in which the currently unspecified preferences can be revealed. This notation comes from multi-agent preference aggregation [52, 59, 58], where, in the context of voting theory, some preferences are missing but still one would like to declare a winner.

Given an incomplete soft constraint problem (ISCSP), its set of possibly optimal solutions is never empty, while the set of necessarily optimal solutions can be empty. Of course what we would like to find is a necessarily optimal solution, to be on the safe side: such solutions are optimal regardless of how the missing preferences are specified. Unfortunately, such a set may be empty. In this case there are two choices: either we may be satisfied with a possibly optimal solution, or we can elicit some of the missing preferences from the user and see if the new ISCSP has a necessarily optimal solution.

We follow this second approach and we repeat the process until the current ISCSP has at least one necessarily optimal solution. In order to do that, we exploit a modified version of the classical branch and bound scheme and we consider different elicitation strategies. In particular, we define a general algorithm scheme that is based on three parameters: *when* to elicit, *what* to elicit, and *who* chooses the value to be assigned to the next variable. For example, we may only elicit missing preferences after running branch and bound to exhaustion, or at the end of every complete branch, or even at every node in the search tree. Also, we may elicit all missing preferences related to the candidate solution, or we might just ask the user for the worst preference among some missing ones. Finally, when choosing the value to assign to a variable, we might ask the user, who knows or can compute (all or some of) the missing preferences, for help.

We test all possible instances of the scheme, obtained by selecting different elicitation strategies, on randomly generated soft constraint problems (fuzzy and weighted). By varying the number of variables, the tightness and density of constraints as well as the percentage of missing preferences, we produce a diversified and meaningful test set. The experiments demonstrate that

some of the algorithms are very good at finding necessarily optimal solutions without eliciting too many preferences. We also test some of the algorithms on problems with hard constraints and on fuzzy temporal constraints. Our experimental study on randomly generated problems permits us to filter out algorithms with a poor performance and, thus, to identify those that are more promising for future testing on real-life scenarios. We conclude our experimentation with an evaluation of our algorithms on an adaptation of the meeting scheduling problem to allow for soft constraints and missing preferences.

In our experiments, we compute the elicited preferences, that is, the missing values that the user has to provide to the system because they are requested by the algorithm. Providing these values usually has a cost, either in terms of the computational effort, or in terms of a decrease in privacy, or in terms of the communication bandwidth. Whilst knowing *how many preferences are elicited* is important, we also compute a measure of the *user's effort*. This may be much larger than the number of elicited preferences, as it contains all the preference values the user may have to compute to be able to respond to the elicitation requests. For example, suppose we ask the user for the worst preference value among $k$ missing ones. The user will communicate only one value, but he may have to compute and consider all $k$ of them. Knowing the number of elicited preferences is important when the concern is to communicate as little information as possible. The user effort, on the other hand, measures the hidden work the user has to do to be able to communicate the elicited preferences. This user's effort is therefore also an important measure.

As a motivating example, recommender systems give suggestions based on partial knowledge of the user's preferences. Our approach could improve performance by identifying some key questions to ask before giving recommendations. Privacy concerns regarding the percentage of elicited preferences are motivated by eavesdropping. User's effort is instead related to the burden on the user.

## 3.2 Basic notions

Although the SCSP framework is very expressive and powerful it can not model situations where preferences are missing. For example, in situations where privacy is critical the goal may be solving a problem knowing less preferences as possible. Moreover, in other real life context computing a preference may be costly in terms of money and/or time.
For these reasons, we extend the SCSP framework to deal with problems

with missing preferences. We call such kind of problems incomplete SCSPs, written ISCSPs.

Informally, an incomplete SCSP, is an SCSP where the preferences of some tuples in the constraints, and/or of some of the values in the domains, are not specified. In detail, given a set of variables $V$ with finite domain $D$, and c-semiring $S = \langle A, +, \times, 0, 1\rangle$, we extend the SCSP framework to incompleteness by the following definitions.

**Definition 23** (incomplete soft constraint)**.** *Given a set of variables $V$ with finite domain $D$, and a c-semiring $\langle A, +, \times, 0, 1\rangle$, an incomplete soft constraint is a pair $\langle idef, con\rangle$ where $con \subseteq V$ is the scope of the constraint and $idef : D^{|con|} \longrightarrow A \cup \{?\}$ is the preference function of the constraint. All tuples mapped into ? by $idef$ are called incomplete tuples.*

In an incomplete soft constraint, the preference function can either specify the preference value of a tuple by assigning a specific element from the carrier of the c-semiring, or leave such preference unspecified. Formally, in the latter case the associated value is ?. A soft constraint is a special case of an incomplete soft constraint where all the tuples have a specified preference.

**Definition 24** (incomplete soft constraint problem (ISCSP))**.** *An incomplete soft constraint problem is a pair $\langle C, V, D\rangle$ where $C$ is a set of incomplete soft constraints over the variables in $V$ with domain $D$. Given an ISCSP $P$, we will denote with $IT(P)$ the set of all incomplete tuples in $P$.*

**Definition 25** (completion)**.** *Given an ISCSP $P$, a completion of $P$ is an SCSP $P'$ obtained from $P$ by associating to each incomplete tuple in every constraint an element of the carrier of the c-semiring. A completion is partial if some preference remains unspecified. We will denote with $C(P)$ the set of all possible completions of $P$ and with $PC(P)$ the set of all its partial completions.*

In Example 1, we show how the ISCSP framework can be used to model a problem with missing preferences.

**Example 1.** *A travel agency is planning Alice and Bob's honeymoon. The candidate destinations are the Maldives islands or the Caribbean, and they can decide to go by ship or by plane. To go to Maldives, they have a high preference to go by plane and a low preference to go by ship. For the Caribbean, they have a high preference to go by ship, and they don't give any preference on going there by plane.*

*Assume we use the fuzzy c-semiring $\langle [0, 1], max, min, 0, 1\rangle$. We can model this problem by using two variables $T$ (standing for $Transport$) and $D$ (standing for $Destination$) with domains $D(T) = \{p, sh\}$ ($p$ stands for plane and*

*sh for ship) and* $D(D) = \{m, c\}$ *(m stands for Maldives, c for Caribbean), and an incomplete soft constraint* $\langle idef, con\rangle$ *with* $con = \{T, D\}$ *and preference function as shown in Figure 3.1. The only incomplete tuple in this soft constraint is* $(p, c)$*.*

*Also, assume that for the considered season the Maldives are slightly preferable to the Caribbean. Moreover, Alice and Bob have a high preference for plane as a means of transport, while they don't give any preference to ship. Moreover, as far as accommodations, which can be in a standard room, a suite, or a bungalow, assume that a suite in the Maldives is too expensive while a standard room in the Caribbean is not special enough for a honeymoon. To model this new information we use a variable A (standing for Accommodation) with domain* $D(A) = \{r, su, b\}$ *(r stands for room, su for suite and b for bungalow), and three constraints: two unary incomplete soft constraints,* $\langle idef1, \{T\}\rangle$, $\langle idef2, \{D\}\rangle$ *and a binary incomplete soft constraint* $\langle idef3, \{A, D\}\rangle$*. The definition of such constraints is shown in Figure 3.1. The set of incomplete tuples of the entire problem is* $IT(P) = \{(sh), (p, c), (su, c), (b, c), (r, m), (su, m)\}$*.* □

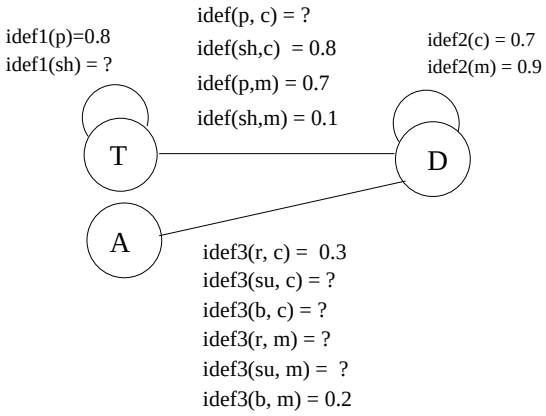



Figure 3.1: An ISCSP.

**Definition 26** (preference of an assignment, incomplete tuples)**.** *Given an ISCSP* $P = \langle C, V, D\rangle$ *and an assignment s to all its variables, we denote with* $pref(P, s)$ *the preference of s in P and with DEF(P,s) the set of soft constraints with no s-related missing preferences, that is,* $DEF(P, s) = < idef, con > \in C | idef(s_{\downarrow con}) \neq ?$*. In detail,* $pref(P, s) = \Pi_{<idef,con>\in DEF(P,s)}$ $idef(s_{\downarrow con})$*. Moreover, we denote by* $it(s)$ *the set of all the projections of s over constraints of P which have an unspecified preference.*

The preference of an assignment $s$ in an incomplete problem is thus obtained by combining the known preferences associated with the projections

of the assignment, that is, of the appropriated sub-tuples in the constraints. The projections which have unspecified preferences, that is, those in $it(s)$, are simply ignored.

**Example 2.** *Consider the two assignments* $s_1 = (p, m, b)$ *and* $s_2 = (p, m, su)$ *for the problem in Figure 3.1. We have that* $pref(P, s_1) = min\ (0.8, 0.7, 0.9, 0.2) = 0.2$*, while* $pref(P, s_2) = min\ (0.8, 0.7, 0.9) = 0.7$*. However, while the preference of* $s_1$ *is fixed, since none of its projections is incomplete, the preference of* $s_2$ *may become lower than* 0.7 *depending on the preference of the incomplete tuple* $(su, m)$*.* □

As shown by the example, the presence of incompleteness partitions the set of assignments into two sets: those which have a certain preference which is independent of how incompleteness is resolved, and those whose preference is only an upper bound, in the sense that it can be lowered in some completions. Given an ISCSP $P$, we will denote the first set of assignments as $Fixed(P)$ and the second with $Unfixed(P)$. In Example 2, $Fixed(P) = \{s_1\}$, while all other assignments belong to $Unfixed(P)$.

In SCSPs, an assignment is an optimal solution if its global preference is undominated. This notion can be generalized to the incomplete setting. In particular, when some preferences are unknown, we will speak of necessarily and possibly optimal solutions, that is, assignments which are undominated in all (resp., some) completions.

**Definition 27** (necessarily and possibly optimal solution)**.** *Given an ISCSP* $P = \langle C, V, D\rangle$*, an assignment* $s \in D^{|V|}$ *is a necessarily (resp, possibly) optimal solution iff* $\forall Q \in C(P)$ *(resp.,* $\exists Q \in C(P)$*)* $\forall s' \in D^{|V|}$*,* $pref(Q, s') \not> pref(Q, s)$*.*

Given an ISCSP $P$, we will denote with $NOS(P)$ (resp., $POS(P)$) the set of necessarily (resp., possibly) optimal solutions of $P$. Notice that, while $POS(P)$ is never empty, in general $NOS(P)$ may be empty. In particular, $NOS(P)$ is empty whenever the available preferences are not sufficient to establish the relationship between an assignment and all others.

**Example 3.** *In the ISCSP* $P$ *of Figure 3.1, we can easily see that* $NOS(P) = \emptyset$ *since, given any assignment, it is possible to construct a completion of* $P$ *in which it is not an optimal solution. On the other hand,* $POS(P)$ *contains all assignments not including tuple* $(sh, m)$*.* □

## 3.3 Characterizing optimal solutions

In this section we characterize the set of necessarily and possibly optimal solutions of an ISCSP given the preferences of the optimal solutions of two

of the completions of $P$. All the results are given for ISCSPs defined on totally ordered c-semirings. In particular, given an ISCSP $P$ defined on the c-semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, we consider:

- the SCSP $P_0 \in C(P)$, called the **0**-completion of $P$, obtained from $P$ by associating preference $\mathbf{0}$ to each tuple of $IT(P)$.
- the SCSP $P_1 \in C(P)$, called the **1**-completion of $P$, obtained from $P$ by associating preference $\mathbf{1}$ to each tuple of $IT(P)$.

Let us indicate respectively with $pref_0$ and $pref_1$ the preference of an optimal solution of $P_0$ and $P_1$. Due to the monotonicity of $\times$, and since $\mathbf{0} \leq \mathbf{1}$, we have that $pref_0 \leq pref_1$.

**Example 4.** *Consider the problem shown in Figure 3.1. We have that* $pref_0 = 0.2$ *and* $pref_1 = 0.7$. □

We will now give some lemmas that will be useful to show the following theorems.

**Lemma 1.** *Given an ISCSP $P$ and the completion $P_1 \in C(P)$ as defined above, we have that $pref(P, s) = pref(P_1, s)$.*

*Proof.* Follows immediately from the definition of $pref(P, s)$ and from the fact that in a c-semiring $\mathbf{1}$ is the unit element. □

**Lemma 2.** *Given an ISCSP $P$ and the completion $P_1 \in C(P)$ as defined above, there always exists an assignment $s$ such that $pref(P, s) = pref_1$.*

*Proof.* Follows from Lemma 1 and choosing any $s \in Opt(P_1)$. □

**Lemma 3.** *Given an ISCSP $P$, the completions $P_0, P_1 \in C(P)$ as defined above, and another completion $P' \in C(P)$, then, $\forall s \in Opt(P')$, $pref_0 \leq pref(P', s) \leq pref_1$.*

*Proof.* Due to monotonicity, for any solution $s$ we have that $pref(P', s) \leq pref(P_1, s) \leq pref_1$, since $pref_1$ is the optimal preference of $P_1$. Assume there is a solution $s \in Opt(P')$ such that $pref(P', s) < pref_0$. Then, for any solution $s_0 \in Opt(P_0)$, we have, by monotonicity, $pref(P', s) < pref_0 = pref(P_0, s_0) \leq pref(P', s_0)$. Thus, we have a contradiction, since $s$ is an optimal solution of $P'$. □

**Lemma 4.** *Given an ISCSP $P$ and the completion $P_1 \in C(P)$ as defined above, if $pref_1 > pref_0$, then $Opt(P_1) \subseteq Unfixed(P)$.*

*Proof.* Assume there is a fixed solution $s$ such that $s \in Opt(P_1)$. Then we would have that $pref(P_0, s) = Pref(P_1, s) = pref_1$ and thus $pref(P_0, s) > pref_0$ which is a contradiction, since $pref_0$ is the optimal preference in $P_0$. □

**Lemma 5.** *Given an ISCSP $P$, we have that $NOS(P) = \cap_{P' \in C(P)} Opt(P')$.*

*Proof.* Any solution $s \in \cap_{P' \in C(P)} Opt(P')$ satisfies the definition of necessarily optimal. Consider now $s \in NOS(P)$ and a completion $P'$ of $P$. Then, by Definition 27, $s$ cannot be dominated by another solution $s'$, and thus $s \in Opt(P')$. □

In the following theorem we will show that, if $pref_0 > \mathbf{0}$, there is a necessarily optimal solution of $P$ iff $pref_0 = pref_1$, and in this case $NOS(P)$ coincides with the set of optimal solutions of $P_0$.

**Theorem 3.** *Given an ISCSP $P$ and the two completions $P_0, P_1 \in C(P)$ as defined above, if $pref_0 > \mathbf{0}$ we have that $NOS(P) \neq \emptyset$ iff $pref_1 = pref_0$. Moreover, if $NOS(P) \neq \emptyset$, then $NOS(P) = Opt(P_0)$.*

*Proof.* Since we know that $pref_0 \leq pref_1$, if $pref_0 \neq pref_1$ then $pref_1 > pref_0$. We prove that, if $pref_1 > pref_0$, then $NOS(P) = \emptyset$. Let us consider any assignment $s$ of $P$. Due to the monotonicity of $\times$, for all $P' \in C(P)$, we have $pref(P', s) \leq pref(P_1, s) \leq pref_1$.

- If $pref(P_1, s) < pref_1$, then $s$ is not in $NOS(P)$ since $P_1$ is a completion of $P$ where $s$ is not optimal.
- If instead $pref(P_1, s) = pref_1$, then, $s \in Opt(P_1)$ and, by Lemma 4 we have that $s \in Unfixed(P)$. Thus we can consider completion $P_1'$ obtained from $P_1$ by associating preference $\mathbf{0}$ to the incomplete tuples of $s$. In $P_1'$ the preference of $s$ is $\mathbf{0}$ and the preference of an optimal solution of $P_1'$ is, due to the monotonicity of $\times$, at least that of an optimal solution of $P_0$, that is $pref_0 > \mathbf{0}$ Thus $s \notin NOS(P)$.

Next we consider when $pref_0 = pref_1$. From Lemma 5 follows that $NOS(P) \subseteq Opt(P_0)$. We will show that $NOS(P) \neq \emptyset$ by showing that any $s \in Opt(P_0)$ is in $NOS(P)$. Let us assume, on the contrary, that there is $s \in Opt(P_0)$ such that $s \notin NOS(P)$. Thus there is a completion $P'$ of $P$ with an assignment $s'$ with $pref(P', s') > pref(P', s)$. By construction of $P_0$, any assignment $s \in Opt(P_0)$ must be in $Fixed(P)$. In fact, if it had some incomplete tuple, its preference in $P_0$ would be $\mathbf{0}$, since $\mathbf{0}$ is the absorbing element of $\times$. Since $s \in Fixed(P)$, $pref(P', s) = pref(P_0, s) = pref_0$. By construction of $P_1$ and monotonicity of $\times$, we have $pref(P_1, s') \geq pref(P', s')$. Thus

the contradiction $pref_1 \geq pref(P_1, s') \geq pref(P', s') > pref(P', s) = pref_0$. This allows us to conclude that $s \in NOS(P) = Opt(P_0)$. □

In the theorem above we have assumed that $pref_0 > \mathbf{0}$. The case in which $pref_0 = \mathbf{0}$ needs to be treated separately. We consider it in the following theorem.

**Theorem 4.** *Given ISCSP $P = \langle C, V, D\rangle$ and the two completions $P_0, P_1 \in C(P)$ as defined above, assume $pref_0 = \mathbf{0}$. Then, if $pref_1 = \mathbf{0}$, $NOS(P) = D^{|V|}$. If $pref_1 > \mathbf{0}$, $NOS(P) = \{s \in Opt(P_1) | \forall s' \in D^{|V|}$ with $pref(P_1, s') > \mathbf{0}$ we have $it(s) \subseteq it(s')\}$.*

*Proof.* We prove the two items separately.

- If $pref_0 = pref_1 = \mathbf{0}$, then, from Lemma 3 follows that the preference level of the optimal solution of SCSP $P'$ is $\mathbf{0}$. Thus all assignments have always the same preference equal to 0. Thus they are all necessarily optimal solutions.

- Let us now assume that $\mathbf{0} = pref_0 < pref_1$. From Lemma 5, only assignments in $Opt(P_1)$ can be in $NOS(P)$ since all other assignments are not optimal in $P_1$. Let us now consider $s \in Opt(P_1)$. By Lemma 4 we have that $it(s) \neq \emptyset$. If there exists $s' \in D^{|V|}$, with $pref(P_1, s') > \mathbf{0}$, such that $it(s) \not\subseteq it(s')$ then we can construct a completion of $P$, say $P'$ where $s$ is not optimal. It is sufficient to set the preference of the tuples in $it(s')$ to $\mathbf{1}$ and the tuples in $it(s) \setminus it(s')$ to $\mathbf{0}$. We have that $pref(P', s) = \mathbf{0}$, since $\mathbf{0}$ is the absorbing element of $\times$, and $pref(P', s') = pref(P_1, s')$. Thus, in $P'$ we have $pref(P', s') = pref(P_1, s') > pref(P', s) = \mathbf{0}$.

  We will now show that, if given $s \in Opt(P_1)$ there is no $s' \in D^{|V|}$ with $pref(P_1, s') > \mathbf{0}$ such that $it(s) \not\subseteq it(s')$, then $s \in NOS(P)$.

  First notice that, since $\mathbf{1}$ is the unit element of $\times$, $\forall P' \in C(P)\, pref(P', s) = pref(P_1, s) \times it\text{-}pref(P', s)$ and $pref(P', s') = pref(P_1, s') \times it\text{-}pref(P', s')$ where $it\text{-}pref(P', s)$ (resp. $it\text{-}pref(P', s')$) is the combination of the preferences associated in $P'$ to the incomplete tuples in $it(s)$ (resp. $it(s')$).

  Since for every $s' \in D^{|V|}$ with $pref(P_1, s') > \mathbf{0}$ we are assuming that $it(s) \subseteq it(s')$, then $\forall P' \in C(P)$, $it\text{-}pref(P', s) \geq it\text{-}pref(P', s')$, due to the intensive property of $\times$. Moreover, since $s \in Opt(P_1)$, $pref(P_1, s) = pref_1 > pref(P_1, s')$. Thus, for every $P' \in C(P)$, $\forall s' \in D^{|V|}$ (trivially for those with $pref(P_1, s') = \mathbf{0}$) we have that $pref(P', s) \geq pref(P', s')$. This allows us to conclude that $s \in NOS(P)$. □

Intuitively, if the tuples of $s$ are not a subset of the incomplete tuples of some assignment $s'$, then we can make $s'$ dominate $s$ in a completion by setting all the incomplete tuples of $s'$ to $\mathbf{1}$ and all the remaining incomplete tuples of $s$ to $\mathbf{0}$. In such a completion $s$ is not optimal. Thus $s$ is not a necessarily optimal solution. However, if the tuples of $s$ are a subset of the incomplete tuples of all other assignments, then it is not possible to lower $s$ without lowering all other tuples even further. This means that $s$ is a necessarily optimal solution.

We now turn our attention to possible optimal solutions. Given a c-semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, it has been shown in [8] that idempotency and strict monotonicity of the $\times$ operator are incompatible, that is, at most one of these two properties can hold. In the following two theorems we show that the presence of one or the other of such two properties plays a key role in the characterization of $POS(P)$ where $P$ is an ISCSP. In particular, if $\times$ is idempotent, then the possibly optimal solutions are the assignments with preference in $P$ between $pref_0$ and $pref_1$. If, instead, $\times$ is strictly monotonic, then the possibly optimal solutions have preference in $P$ between $pref_0$ and $pref_1$ and dominate all the assignments which have as set of incomplete tuples a subset of their incomplete tuples.

**Theorem 5.** *Given an ISCSP $P$ defined on a c-semiring with idempotent $\times$ and the two completions $P_0, P_1 \in C(P)$ as defined above, if $pref_0 > \mathbf{0}$ we have that: $POS(P) = \{s \in D^{|V|} | pref_0 \leq pref(P, s) \leq pref_1\}$.*

*Proof.* First we show that any $s$ such that $pref_0 \leq pref(P, s) \leq pref_1$ is in $POS(P)$. Let us consider the completion of $P$, $P'$, obtained by associating preference $pref(P, s)$ to all the incomplete tuples of $s$ and $\mathbf{0}$ to all other incomplete tuples of $P$. For any other assignment $s'$ we can show that it never dominates $s$:

- $s' \in Fixed(P)$ and thus $pref(P', s') = pref(P_0, s') \leq pref_0 \leq pref(P, s)$;
- $s' \in Unfixed(P)$ and
  - $it(s') \not\subseteq it(s)$, then $pref(P', s') = \mathbf{0}$ since in $P'$ the incomplete tuples in $it(s')$ which are not in $it(s)$ have been associated with preference $\mathbf{0}$;
  - $it(s') \subseteq it(s)$. By construction of $P'$ and since $\times$ is idempotent and associative we have that: $pref(P', s) = (pref(P, s) \times (\Pi_{|it(s)|} pref(P, s))) = pref(P, s)$ and $pref(P', s') = (pref(P, s') \times (\Pi_{|it(s')|} pref(P, s))) = pref(P, s') \times pref(P, s)$. Since $\times$ is intensive, $pref(P', s') = (pref(P, s') \times pref(P, s)) \leq pref(P, s) = pref(P', s)$.

Thus in $P'$ no assignment dominates $s$. This means that $s \in POS(P)$.

We will now show that if $s \in POS(P)$, $pref_0 \leq pref(P,s) \leq pref_1$. If $s \in POS(P)$, then $s \in Opt(Q)$ form some $Q \in C(P)$. Thus we can conclude by Lemma 3. □

Informally, given a solution $s$ such that $pref_0 \leq pref(P,s) \leq pref_1$, it can be shown that it is an optimal solution of the completion of $P$ obtained by associating preference $pref(P,s)$ to all the incomplete tuples of $s$, and $\mathbf{0}$ to all other incomplete tuples of $P$. On the other hand, by construction of $P_0$ and due to the monotonicity of $\times$, any assignment which is not optimal in $P_0$ cannot be optimal in any other completion. Also, by construction of $P_1$, there is no assignment $s$ with $pref(P,s) > pref_1$.

**Theorem 6.** *Given an ISCSP $P$ defined on a c-semiring with a strictly monotonic $\times$ and the two completions $P_0, P_1 \in C(P)$ as defined above, if $pref_0 > \mathbf{0}$ we have that: $s \in POS(P)$ iff $pref_0 \leq pref(P,s) \leq pref_1$ and $pref(P,s) = max\{\ pref(P,s')|\ it(s') \subseteq it(s)\}$.*

*Proof.* Let us first show that if assignment $s$ is such that $pref_0 \leq pref(P,s) \leq pref_1$ and $pref(P,s) = max\{pref(P,s')|it(s') \subseteq it(s)\}$ it is in $POS(P)$. We must show there is a completion of $P$ where $s$ is undominated. Let us consider completion $P'$ obtained by associating preference $\mathbf{1}$ to all the tuples in $it(s)$ and $\mathbf{0}$ to all the tuples in $IT(P) \setminus it(s)$. First we notice that $pref(P',s) = pref(P,s)$, since $\mathbf{1}$ is the unit element of $\times$. Let us consider any other assignment $s'$. Then we have one of the following:

- $it(s') = \emptyset$, which means that $s' \in Fixed(P)$ and thus $pref(P',s') = pref(P_0,\ s') \leq pref_0 \leq pref(P,s) = pref(P',s)$;
- $it(s') \not\subseteq it(s)$, which means that there is at least one incomplete tuple of $it(s')$ which is associated with $\mathbf{0}$. Since $\mathbf{0}$ is the absorbing element of $\times$, $pref(P',s') = \mathbf{0}$ and thus $pref(P',s') < pref_0 \leq pref(P',s)$;
- $it(s') \subseteq it(s)$, in this case $pref(P',s') = pref(P,s')$ since all tuples in $it(s')$ are associated with $\mathbf{1}$ in $P'$. However since $pref(P,s) = max\{pref(P,s')|it(s') \subseteq it(s)\}$, $pref(P',s') \leq pref(P',s)$.

We can thus conclude that $s$ is not dominated by any assignment in $P'$. Hence $s \in POS(P)$.

Let us now prove the other direction by contradiction. If $pref(P,s) < pref_0$ then we can conclude by Lemma 2. We must prove that if $pref_0 \leq pref(P,s) \leq pref_1$ and $pref(P,s) < max\{pref(P,s')|it(s') \subseteq it(s)\}$ then $s$ is not in $POS(P)$. In any completion $P'$ of $P$ we have that $pref(P',s) =$

$pref(P,s) \times it\text{-}pref(P',s)$ and $pref(P',s') = pref(P,s') \times it\text{-}pref(P',s')$ where $it\text{-}pref(P',s)$ (resp. $it\text{-}pref(P',s')$) is the combination of the preferences associated to the incomplete tuples in $it(s)$ (resp. $it(s')$). Since $it(s') \subseteq it(s)$, for any completion $P'$ we have that $it\text{-}pref(P',s) \leq it\text{-}pref(P',s')$. Moreover, let $s''$ be such that $pref(P,s'') = max\{pref(P,s')|it(s') \subseteq it(s)\}$. Then we have that for any completion $P'$, $pref(P',\ s'') > pref(P',s)$ since $pref(P,s'') > pref(P,s)$ and $it\text{-}pref(P',\ s'') \geq it\text{-}pref(P',s)$ and $\times$ is strictly monotonic. Thus, if $pref_0 \leq pref(P,s) \leq pref_1$ and $pref(P,s) < max\{pref(P,s')|\ \ it(s') \subseteq it(s)\}$, then $s$ is not in $POS(P)$. □

The intuition behind the statement of this theorem is that, if assignment $s$ is such that $pref_0 \leq pref(P,s) \leq pref_1$ and $pref(P,s) = max\{pref(P,s')|it(s') \subseteq it(s)\}$, then it is optimal in the completion obtained associating preference $\mathbf{1}$ to all the tuples in $it(s)$ and $\mathbf{0}$ to all the tuples in $IT(P) \setminus it(s)$. On the contrary, if $pref(P,s) < max\{pref(P,s')|it(s') \subseteq it(s)\}$, there must be another assignment $s''$ such that $pref(P,\ s'') = max\{pref(P,s')|it(s') \subseteq it(s)\}$. It can then be shown that, in all completions of $P$, $s$ is dominated by $s''$.

In contrast to $NOS(P)$, when $pref_0 = \mathbf{0}$ we can immediately conclude that $POS(P) = D^{|V|}$, independently of the nature of $\times$, since all assignments are optimal in $P_0$.

**Corollary 6.1.** *Given an ISCSP $P = \langle C, V, D\rangle$, if $pref_0 = \mathbf{0}$, then $POS(P) = D^{|V|}$.*

For ease of clarity, the results shown in this section can be summarized as follows:

- when $pref_0 = pref_1 = \mathbf{0}$
  - $NOS(P) = D^{|V|}$ (by Theorem 4);
  - $POS(P) = D^{|V|}$ (by Corollary 6.1) ;
- when $\mathbf{0} = pref_0 < pref_1$
  - $NOS(P) = \{s \in Opt(P_1)|\forall s' \in D^{|V|}$ with $pref(P_1,s') > \mathbf{0}$ we have $it(s) \subseteq it(s')\}$ (by Theorem 4);
  - $POS(P) = D^{|V|}$ (by Corollary 6.1);
- when $\mathbf{0} < pref_0 = pref_1$
  - $NOS(P) = Opt(P_0)$ (by Theorem 3);
  - if $\times$ is idempotent: $POS(P) = \{s \in D^{|V|}|pref_0 \leq pref(P,s) \leq pref_1\}$ (by Theorem 5);

  - if $\times$ is strictly monotonic: $POS(P) = \{s \in D^{|V|} | pref_0 \leq pref(P,s) \leq pref_1, pref(P,s) = max\{\ pref(P,s') | it(s') \subseteq it(s)\}\}$ (by Theorem 6);

- when $\mathbf{0} < pref_0 < pref_1$
  - $NOS(P) = \emptyset$ (by Theorem 3);
  - $POS(P)$ as for the case when $\mathbf{0} < pref_0 = pref_1$.

## 3.4 Solving incomplete soft constraints via a systematic search

In this section we first describe a general schema for solving ISCSPs based on interleaving branch and bound search with preference elicitation. Such a general schema is instantiated to different elicitation strategies generating several concrete algorithms. A computational analysis of the algorithms is provided both in terms of the number of elicited preferences and of the user effort for revealing some of the missing preferences.

### The solver schema and its instances

The solving strategy which we propose for ISCSPs is based on the idea of combining a branch and bound search (B&B) with elicitation steps in which the user is asked to provide some type of missing information. In general, B&B proceeds by considering the variables in some order, by choosing a value for each variable in the order, and by computing, using some heuristics, an upper bound on the global preference of any completion of the current partial assignment. B&B also stores the highest preference (assuming the goal is to maximize) of a complete assignment found so far. If at any step the upper bound is lower than the preference of the current best solution, the search backtracks.

When some of the preferences are missing, as in ISCSPs, the agent may be asked for some preferences or other information regarding the preferences in order to know the true preference of a partial or complete assignment or in order to choose the next value for some variable. Preferences can be elicited after each run of B&B or during a B&B run while preserving the correctness of the approach. For example, we can elicit preferences at the end of every complete branch (that is, regarding preferences of every complete assignment considered in the branch and bound algorithm), or at every node in the search tree (thus considering every partial assignment). Moreover, when choosing

the value for the next variable to be assigned, we can ask the user (who knows the missing preferences) for help. Finally, rather than eliciting all the missing preferences in the possibly optimal solution, or the complete or partial assignment under consideration, we can elicit just some of the missing preferences.

For example, with incomplete fuzzy constraint problems (IFCSPs), eliciting just the worst preference among the missing ones is sufficient, since only the worst value is important to the computation of the overall preference value. Instead, with incomplete weighted constraint problems (IWCSPs), we need to elicit as many preference values as needed to decide whether the current assignment is better than the best one found so far.

More precisely, the algorithm schema we propose is based on the following parameters:

1. WHO chooses the value of a variable:
   (a) the algorithm, using one of the following heuristics:
      i. values are picked in decreasing order w.r.t. their preference values in the **1**-completion. The order is maintained dynamically. We denote this heuristic with *dp*;
      ii. values are picked in decreasing order w.r.t. the preferences in the **0**-completion of the initial ISCSP. The order is thus static. We denote this heuristic with *dpi*.
   (b) The user, revealing the value that he prefers according to one of the following criteria:
      i. the value is the most preferred among those in the domain which haven't been considered yet (*lazy user*, *lu* for short);
      ii. the value is the most preferred among those which haven't been considered yet given the constraints involving the current variable and the past variables in the search order (*smart user*, *su* for short);
2. WHAT must be elicited:
   (a) the preferences of all the incomplete tuples of the current assignment (denoted with *all*);
   (b) for IFCSPs, only the preference of the worst tuple of the current assignment, if it is worse than the known ones (denoted with *worst*);
   (c) for IWCSPs:

- the worst missing cost (that is, the highest) until either all the costs are elicited or the current global cost of the (possibly partial) assignment is higher than the optimum found so far. This strategy is denoted by WW.
- the best (i.e. the minimum) cost until either all the costs are elicited or the current global cost of the (possibly partial) assignment is higher than the optimum found so far. This strategy is denoted by BB.
- the best and the worst cost in turn. This strategy is denoted by BW.

3. WHEN elicitation should take place:

   (a) at the end of the branch and bound search (at *tree* level).

   (b) during the search, when we have a complete assignment to all the variables (i.e., when we have reached a leaf of the search tree, and thus when we are at the end of a *branch*). We will refer to such a heuristics, by saying "at *branch* level".

   (c) during search, whenever a new value is assigned to a variable. We will refer to such a heuristics, by saying "at the *node* level".

Summarizing, we have three features which we call *who, what* and *when*. There are four possible choices for *who*: *dp*, *dpi*, *lu*, and *su*. If we work with IFCSPs, there are two possibilities for *what*: *all* and *worst*. Instead, with IWCSPs, there are four options: *all*, WW, BB, and BW. Finally, there are three options for *when*: *tree*, *branch*, and *node*. If $when = tree$, elicitation takes place only when the search is completed. This means that the B&B search can be performed more than once. In contrast, if $when = branch$ or $when = node$, the B&B search is performed only once and the elicitation is done either at every node of the search tree or at every leaf.

By choosing a value for each of the three parameters above in a consistent way, we obtain, for IFCSPs, a total of 16 different algorithms, as summarized in Figure 3.2.

If instead we work with IWCSPs, we have a total of 32 algorithms, as can be seen in Figure 3.3.

The pseudocode of our general solver, which we call ISCSP-SCHEME, is shown in Algorithms 2 and 3. Every point in Figure 3.2 and Figure 3.3 represents an instantiation of ISCSP-SCHEME to specific values for parameters *who*, *what* and *when*.

ISCSP-SCHEME takes in input an ISCSP $P$ and the values for the three parameters: *who*, *what*, and *when*. It returns an ISCSP $Q$, a complete

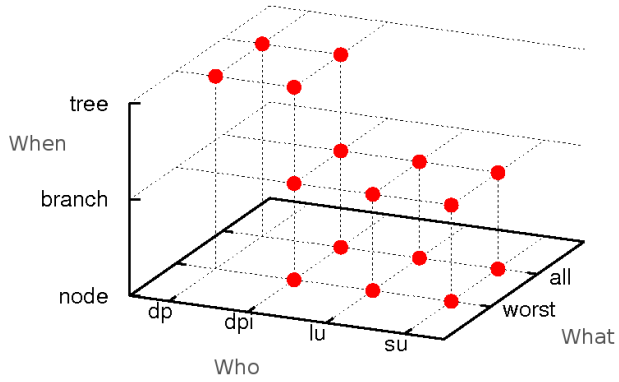



Figure 3.2: The algorithms for IFCSPs.

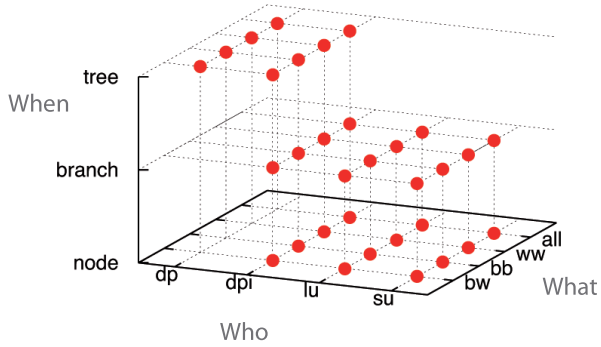



Figure 3.3: The algorithms for IWCSPs.

assignment $s$ and a preference $p$. In Theorems 7 and 9 we will show that $Q$ is a partial completion of $P$ and $s$ is a necessarily optimal solution of $Q$ with preference $p$. As a first step, ISCSP-SCHEME computes the **0**-completion of $P$, called $P_0$, and finds one of its optimal solutions, say $s_{max}$, and its preference, say $pref_{max}$, by applying a standard branch and bound procedure (denoted by $B\&B$). Next, procedure $BBE$ is called. If $BBE$ succeeds, it returns a partial completion of $P$, one of its necessarily optimal solutions, and its associated preference. Otherwise, it returns a solution equal to $nil$. In the first case the output of ISCSP-SCHEME coincides with that of BBE, otherwise ISCSP-SCHEME returns $P_0$ and one of its optimal solutions with the corresponding preference.

Procedure BBE takes as input the same values as ISCSP-SCHEME and, in addition, a solution $sol$ and a preference $lb$ representing the current lower

**Algorithm 2**: ISCSP-SCHEME

**Input**: an ISCSP $P$, a parameter *who* indicating the method of values instantiation, a parameter *what* indicating the elicitation policy, a parameter *when* indicating the level at which the elicitation must be done
**Output**: an ISCSP $Q$, an assignment s, a preference p
compute $P_0$
$Q \leftarrow P_0$
$s_{max}$, $pref_{max} \leftarrow B\&B(P_0)$
$Q'$,$s_1$,$pref_1 \leftarrow BBE(P, 0, who, what, when, s_{max}, pref_{max})$
**if** $s_1 \neq nil$ **then**
  $s_{max} \leftarrow s_1$
  $pref_{max} \leftarrow pref_1$
  $Q \leftarrow Q'$
**return** $Q$, $s_{max}$, $pref_{max}$

bound of the optimal preference level. Solution $sol'$ and preference $pref'$ are initialized to such values at the beginning of BBE. Procedure $nexVariable$ applied to the **1**-completion of the ISCSP in input (denoted by $P[?/1]$) allows to assign to $currentVariable$ the next variable to be assigned. The algorithm then assigns a value to this variable. If the Boolean function $nextValue$ returns true (if there is a value in the domain), we select a value for $currentVariable$ according to the value of parameter *who*.

The computation of the upper bound for the preference that can be obtained by any completion of the current partial assignment is performed by procedure *UpperBound*. In general, any kind of upper bound can be used. However, we have chosen to estimate it by combining the preferences of the constraints involving only variables that have already been instantiated. Formally, let $t$ be the current partial assignment to variables in $\{v_1, \ldots, v_k\} \subseteq V$, and let $c_i = \langle def_i, con_i \rangle$ be a constraint, such that $con_i \subseteq \{v_1, \ldots, v_k\}$. Then, the value $ub$ returned by *UpperBound* is:

$$ub = \prod_{i=1}^{k} def_i(t \downarrow_{con_i}),$$

where $\prod$ is the combination operator of the semiring.

We will now describe procedure BBE by considering the various values for parameter *when*. This corresponds to consider the algorithms in Figures 3.2 and 3.3 divided into the three horizontal planes obtained fixing the value on the *when*-axis.

**Algorithm 3**: BBE

**Input**: an ISCSP $P$, the number of currently instantiated variables $nInstVar$, a parameter $who$ indicating the method of values instantiation, a parameter $what$ indicating the elicitation policy, a parameter $when$ indicating the level at which the elicitation must be done, a reference to a solution $sol$, $lb$ lower bound
**Output**: an ISCSP $P$, a solution $sol$ and its preference $pref$

$sol' \leftarrow sol$
$pref' \leftarrow lb$
$currentVariable \leftarrow nextVariable(P[?/1])$
**while** $nextValue(currentVariable, who)$ **do**
  **if** $when = node$ **then**
    $P, pref \leftarrow Elicit@Node(what, P, currentVariable, lb)$
  $ub \leftarrow UpperBound(P[?/1], currentVariable)$
  **if** $ub >_S lb$ **then**
    **if** $nInstvar = number\ of\ variables\ in\ P$ **then**
      **if** $when = branch$ **then**
        $P,\ pref \leftarrow Elicit@branch(what, P, lb)$
      **if** $pref >_S lb$ **then**
        $sol \leftarrow getSolution(P[?/1])$
        $lb \leftarrow pref(P[?/1], sol)$
    **else**
      $BBE(P, nInstVar + 1, who, what, when, sol, lb)$
**if** $when$=$tree\ and\ nInstVar = 0$ **then**
  **if** $sol = nil$ **then**
    $sol \leftarrow sol'$
    $pref \leftarrow pref'$
  **else**
    $P,\ pref \leftarrow Elicit@tree(what, P,\ sol, lb)$
    **if** $pref >_S pref'$ **then**
      $BBE(P, 0, who, what, when, sol, pref)$
    **else**
      $BBE(P, 0, who, what, when, sol', pref')$

- If *when* = *tree*, elicitation is handled by procedure *Elicit@tree* and takes place only at the end of the search over the **1**-completion. The user is not involved in the value assignment steps within the search and thus there are only two possible values for variable *who*, i.e. *dp* and *dpi*. At the end of the search, if a solution is found, the user may be asked to reveal all the preferences of the incomplete tuples in the solution (if *what* = *all*). If we work with IFCSPs, we could also ask for just the worst one among the missing preferences if it is worst than the known ones (if *what* = *worst*). If instead we work with IWCSPs, preferences can be asked in decreasing (*what* = BB), increasing (*what* = WW), or alternating order (*what* = BW) until we have enough information. If the preference of this solution is better than the best found so far, BBE is called recursively with the new best solution and preference, otherwise the recursive call is done with the old solution and preference.

- If *when* = *branch*, B&B is performed only once and not several times as in the previous case. The user may be asked to choose the next value for the current variable being instantiated. Preference elicitation, which is handled by function *Elicit@branch*, takes place during search, whenever all variables have been instantiated. As above, the user can be asked either to reveal the preferences of all or some of the incomplete tuples depending on the value of *what*. In all cases the information gathered is sufficient to compare the preference of the current assignment against the current lower bound.

- If *when* = *node*, preferences are elicited every time a new value is assigned to a variable, and it is handled by procedure *Elicit@node*. The tuples to be considered for elicitation are those involving the value which has just been assigned and belonging to constraints between the current variable and already instantiated variables. The value of *what* determines whether one or all or some preference values involving the new assignment are asked to the user. With the information given by the user, the preference of the current partial assignment is updated in order to determine if the subtree rooted at the current node can be pruned.

## Termination and correctness

We will now prove that algorithm ISCSP-SCHEME, when given an ISCSP in input, always terminates generating a completion of the ISCSP and one of its necessarily optimal solutions.

**Theorem 7.** *Given an ISCSP $P$ and when = tree, if*

- *what = all, or*
- *what = worst and $P$ is an IFCSP, or*
- *(what = WW or what = BB or what = BW) and $P$ is an IWCSP,*

*algorithm* ISCSP-SCHEME *always terminates and returns an ISCSP $Q$ such that $Q \in PC(P)$, an assignment $s \in NOS(Q)$, and its preference in $Q$.*

*Proof.* Clearly ISCSP-SCHEME terminates if and only if BBE terminates. If we consider the pseudo-code of procedure BBE shown in Algorithm 3, we see that if $when = tree$, BBE terminates when $sol = nil$. This happens only when the search fails to find a solution of the current problem with a preference strictly greater than the current lower bound, i.e., when the condition $pref >_S lb$ is never satisfied. Let us denote with $Q^i$ and $Q^{i+1}$ respectively the ISCSPs given in input to the $i$-th and $(i+1)$-th recursive call of BBE. First we notice that only procedure *Elicit@tree* modifies the ISCSP in input by possibly adding new elicited preferences. Moreover, whatever the value of parameter *what* is, the returned ISCSP is either the same as the one in input or it is a (possibly partial) completion of the one in input. Thus we have $Q^{i+1} \in PC(Q^i)$ and $Q^i \in PC(P)$. Since the search is always performed on the **1**-completion of the current ISCSP, we can conclude that for every solution $s$, $pref(Q^{i+1}, s) \leq_S pref(Q^i, s)$. Let us now denote with $lb^i$ and $lb^{i+1}$ the lower bounds given in input respectively to the $i$-th and $(i+1)$-th recursive call of BBE. It is easy to see that $lb^{i+1} \geq_S lb^i$. Thus, since at every iteration the preferences of solutions cannot increase and the bound cannot decrease, and since we have a finite number of solutions, we can conclude that BBE always terminates.

The reasoning that follows relies on the fact that value $pref$ returned by function *Elicit@tree* is the final preference after elicitation of assignment *sol* given in input. This is true since either $what = all$ and thus all preferences have been elicited and the overall preference of *sol* can be computed, or only the *worst* preference has been elicited but in a fuzzy context where the overall preference coincide with the worst one, or we are in a IWCSP and we have elicited enough preferences to discover that the current solution is worst then the optimum found so far or we have elicited all its costs.

If called with $when = tree$ ISCSP-SCHEME exits when the last branch and bound search has ended returning $sol = nil$. In such a case *sol* and $pref$ are updated to contain the best solution and associated preference found so far, i.e. $sol'$ and $pref'$. Then, the algorithm returns the current ISCSP, say

$Q$, and *sol* and $pref$. Following the same reasoning as above done for $Q^i$, we can conclude that $Q \in PC(P)$.

At the end of a while loop execution of the first call of BBE (the bottom of the call stack), assignment *sol* either contains an optimal solution *sol* of the **1**-completion of the current ISCSP or $sol = nil$. $sol = nil$ iff there is no assignment with preference higher than *lb* in the **1**-completion of the current ISCSP. In this situation, $sol'$ and $pref'$ are an optimal solution and preference of the 1-completion of the current ISCSP. However, since the preference of $sol'$, $pref'$ is fixed and since, due to monotonicity, the optimal preference value of the **1**-completion is always better than or equal to that of the **0**-completion, we have that $sol'$ and $pref'$ are an optimal solution and preference of the 0-completion of the current ISCSP as well.

By Theorems 3 and 4, we can conclude that $NOS(Q)$ is not empty. If $pref = \mathbf{0}$, then $NOS(Q)$ contains all the assignments and thus also *sol*. The algorithm correctly returns the same ISCSP given in input, assignment *sol* and its preference $pref$. If instead $\mathbf{0} < pref$, again the algorithm is correct, since by Theorem 3 we know that $NOS(Q) = Opt(Q[?/0])$, and we have shown that $sol \in Opt(Q[?/0])$. □

Moreover, if parameter $when = tree$, then no useless work is done since we only elicit preferences of assignments that occur in an optimal solution of some completion of the current incomplete problem.

**Theorem 8.** *If* ISCSP-SCHEME *is given in input* $when = tree$*, then only preferences of tuples of solutions in* $POS(P)$ *are elicited.*

*Proof.* If $when = tree$ then, during the execution of ISCSP-SCHEME, preferences are elicited only by procedure *Elicit@tree*. A call to such a procedure, such as *Elicit@tree* (*what*, *P*, *sol*, *lb*), depending on the value of parameter *what*, elicits all or a subset of the preferences of the incomplete tuples of assignment *sol*, returning the (eventually) new global preference of *sol*, $pref$ and the completion of $P$ obtained adding the new elicited preferences. During the execution of ISCSP-SCHEME, *Elicit@tree* is called on the current partial completion of the ISCSP given in input, $P$ and on an optimal solution of its **1**-completion, *sol*. By Theorems 5 and 6, any optimal solution of the **1**-completion of the current partial completion of $P$ is a possibly optimal solution of such a partial completion. □

We will now consider other values for parameter *when*.

**Theorem 9.** *Given a fuzzy or weighted ISCSP* $P$ *and (*$when = branch$ *or* $when = node$*), Algorithm* ISCSP-SCHEME *always terminates, and it*

*returns an ISCSP $Q$ such that $Q \in PC(P)$, an assignment $s \in NOS(Q)$, and its preference in $Q$.*

*Proof.* In order to prove that the algorithm terminates, it is sufficient to show that $BBE$ terminates. Since the domains are finite, the labelling phase produces a number of finite choices at every level of the search tree. Moreover, since the number of variables is limited, then, we have also a finite number of levels in the tree. Hence, $BBE$ considers at most all the possible assignments, that are a finite number. At the end of the execution of ISCSP-SCHEME, $sol$, with preference $pref$ is one of the optimal solutions of the current $P[?/1]$. Thus, for every assignment $s'$, $pref(P[?/1], s') \leq_S pref(P[?/1], sol)$. Moreover, for every completion $Q' \in C(P)$ and for every assignment $s'$, $pref(Q', s') \leq_S pref(P[?/1], s')$. Hence, for every assignment $s'$ and for every $Q' \in C(P)$, we have that $pref(Q', s') \leq_S pref(P[?/1], sol)$. In order to prove that $sol \in NOS(P)$, now it is sufficient to prove that for every $Q' \in C(P)$, $pref(P[?/1], sol) = pref(Q', sol)$. This is true, since $sol \in Fixed(P)$ both when eliciting all the missing preferences, and when eliciting only the worst one for fuzzy ISCSPs, and when eliciting via BB, BW, or WW in weighted ISCSPs. In fact, in both cases, the preference of $sol$ is the same in every completion. To show that the final problem $Q$ returned by BBE is in $PC(P)$, it is sufficient to note that only the procedures *Elicit@node* and *Elicit@branch* modify the ISCSP in input by possibly adding some missing preferences. Thus, the returned ISCSP is in $PC(P)$. □

## 3.5 Solving incomplete soft constraints via a local search

A local search approach has been defined in [1] to find an optimal solution in a soft constraint problem. This approach starts from a randomly chosen assignment to all the variables, say $s$, and at each step it moves to a new assignment which is obtained by changing the value of one variable. Such a variable is one of those whose *local preference* is minimal in $s$. The local preference of a variable in an assignment $s$ is the combination of the preferences identified by $s$ in all constraints involving the variable. The new value for the chosen variable is the one (in its domain) with the best local preference immediate value, that is, the value for which the global assignment preference is the best.

Consider the example in Figure 3.4 which shows an example of an IFCSP with three variables $A$, $B$, and $C$, with domains $D(A) = \{a, b, c\}$, $D(B) = \{d, e\}$, and $D(C) = \{r, s\}$. The presence of the question marks identifies

the missing preference values. In this example, the preference of the variable assignment $\langle A = a, B = e, C = r\rangle$ is $min(0.3, 0.4, 1, 0.9) = 0.3$. Since this variable assignment involves some missing preferences, such as the one for the tuple $\langle A = a, B = e\rangle$, its preference should be interpreted as an upper bound of the actual preference for this assignment.

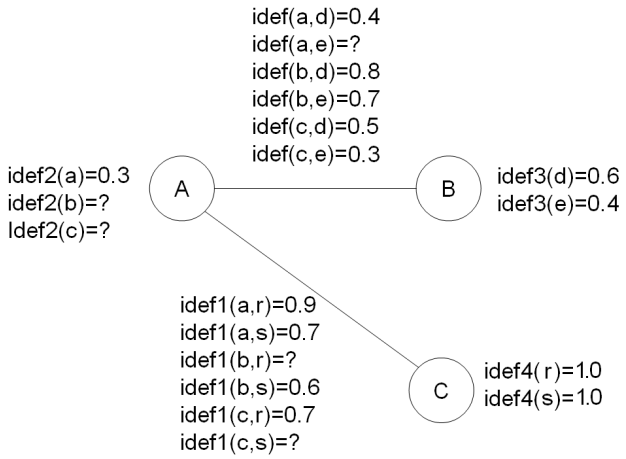



Figure 3.4: An example of ISCSP.

Now consider the problem in Figure 3.4 where all missing preferences are set to 1. This is a classical fuzzy constraint problem. Consider now the assignment $\langle A = a, B = d, C = r\rangle$. In this assignment, the local preference of variable A is $min(0.3, 0.4, 0.9) = 0.3$, while it is 0.4 for B and 0.9 for C. Thus, variable $A$ would be chosen by the local search algorithm.

Once the variable, say $x$, is chosen, its new value is identified by computing, for each new value $v$, the preference of the new assignment (that is, $s$ with the new value $v$ for $x$). The value which gives the best preference for the new assignment is chosen.

Consider again the example in Figure 3.4 where all missing preferences are set to 1, and assignment $\langle A = a, B = d, C = r\rangle$. This assignment has preference 0.3. Variable $A$ can be changed to values $b$ or $c$. With $A = b$, the new assignment has preference 0.6, while with $A = c$, the new assignment has preference 0.5, Thus the algorithm would move to the assignment $\langle A = b, B = d, C = r\rangle$ what has preference 0.6.

In the following we adapt this algorithm to deal with incompleteness.

To start, we randomly generate an assignment of all the variables. To assess the quality of such an assignment, we compute its preference. However, since some missing preferences may be involved in the chosen assignment, we ask the user to reveal them.

In each step, when a variable is chosen, its local preference is computed by setting all the missing preferences to the best preference value (which is 1 for fuzzy constraints and 0 for weighted constraints). In other words, if there are missing preferences, they are not considered in computing the local preference of a variable in a given assignment (since the best value is the neutral element for the combination).

Consider again the example in Figure 3.4 and the assignment $\langle A = a, B = e, C = r\rangle$. In this assignment, the local preference of variable $A$ is $min(0.3, 0.4, 0.9) = 0.3$, while for $B$ is 0.4, and for $C$ is $min(0.9, 1) = 0.9$. Thus our algorithm would choose variable $A$.

To choose the new value for the selected variable, we compute the preferences of the assignments obtained by choosing the other values for this variable. Since some preference values may be missing, in computing the preference of a new assignment we just consider the preferences which are known at the current point. We then choose the value which is associated to the best new assignment. If two values are associated to assignments with the same preference, we choose the one associated to the assignment with the smaller number of incomplete tuples. In this way, we aim at moving to a new assignment which is better than the current one and has the fewest missing preferences.

In the running example above, from assignment $\langle A = a, B = e, C = r\rangle$, once we know that variable $A$ will be changed, we compute $pref(\langle A = b, B = e, C = r\rangle) = 0.4$ and $pref(\langle A = c, B = e, C = r\rangle) = 0.3$. Thus we would select the value $b$ for $A$.

Since the new assignment, say $s'$, could have incomplete tuples, we ask the user to reveal enough of this data to compute the actual preference of $s'$. Of course, asking for all the missing preferences is always a correct strategy; we call ALL the elicitation strategy that elicits all the missing preferences associated to the tuples obtained projecting $s'$ on the constraints. However, depending on the class of soft constraints considered, asking for less than all the preferences could be sufficient. For fuzzy constraints, we also consider an elicitation strategy, called WORST, that asks the user to reveal only the worst preference among the missing ones, if it is less than the worst known preference. This is enough to compute the actual preference of $s'$ since the preference of an assignment coincides with the worst preference in its constraints.

For weighted constraints, we consider the following three strategies (besides ALL):

- WW: we elicit the worst missing cost (that is, the highest) until either all the costs are elicited or the current global cost of the assignment is

higher than the preference of the best assignment found so far;

- BB: we elicit the best (i.e., the minimum) cost with the same stopping condition as for WW;

- BW: we elicit the best and the worst cost in turn, with the same stopping condition as for WW.

As in many classical local search algorithms, to avoid stagnation in local minima, we employ tabu search and random moves. Our algorithm has two parameters: $p$, which is the probability of a random move, and $t$, which is the tabu tenure. When we have to choose a variable to re-assign, the variable is either randomly chosen with probability $p$ or, with probability (1-$p$), we perform the procedure described above. Also, if no improving move is possible, i.e., all new assignments in the neighborhood are worse than or equal to the current one, then the chosen variable is marked as tabu and not used for $t$ steps.

While in classical local search scenarios the underlying problem is always the same, and we just move from one of its solutions to another one, in our scenario we also change the problem via the elicitation strategies. Since the change involves only the preference values, the solution set remains the same, although the preferences of the solutions may decrease over time.

During search, the algorithm maintains the best solution found so far, which is returned when the maximum number of allowed steps is exceeded. In the ideal case, the returned solution is a necessarily optimal solution of the initial problem with the preferences added by the elicitation. However, there is no guarantee that this is so: via elicitation we can reach a problem with necessarily optimal solutions, but the algorithm may fail to find one of those. However, we will show later that, even in this case, the quality of the returned solutions is not very far from that of the necessarily optimal solutions.

## 3.6 Problem generator and experimental design

In this section we present the test environments we defined to test our algorithms. Then, we show and discuss the experimental results.

## Randomly generated incomplete soft constraint problems

To test the performance of these different algorithms, we created Fuzzy ISCSPs using a generator which is a simple extension of the standard random model for hard constraints to soft and incomplete constraints. The generator has the following parameters:

- $n$: number of variables;
- $m$: cardinality of the variable domains;
- $d$: density, that is, the percentage of binary constraints present in the problem w.r.t. the total number of possible binary constraints that can be defined on $n$ variables;
- $t$: tightness, that is, the percentage of tuples with preference 0 in each constraint and in each domain w.r.t. the total number of tuples ($m^2$ for the constraints, since we have only binary constraints, and $m$ in the domains);
- $i$: incompleteness, that is, the percentage of incomplete tuples (that is, tuples with preference ?) in each constraint and in each domain.

Given values for these parameters, we generate IFCSPs as follows. We first generate $n$ variables and then $d\%$ of the $n(n-1)/2$ possible constraints. Then, for every domain and for every constraint, we generate a random preference value in $(0, 1]$ for each of the tuples (that are $m$ for the domains, and $m^2$ for the constraints); we randomly set $t\%$ of these preferences to 0; and we randomly set $i\%$ of the preferences as incomplete.

For example, if the generator is given in input $n = 10$, $m = 5$, $d = 50$, $t = 10$, and $i = 30$, it generates a binary IFCSP with 10 variables, each with 5 elements in the domain, 22 constraints (that is 50% of $45 = 10(10-1)/2$), 2 tuples with preference 0 (that is, 10% of $25 = 5 \times 5$) and 7 incomplete tuples (that is, 30% of $25 = 5 \times 5$) in each constraint, and 1 missing preference (that is, 30% of 5) in each domain. Notice that we use a model B generator: density and tightness are interpreted as percentages, and not as probabilities [29].

We also generate random IWSCSPs using the same parameters as for IFCSPs, with costs in $[0, 10] \cup \{+\infty\}$.

Our experiments measure the *percentage of elicited preferences* (over all the missing preferences) as the generation parameters vary. Since some of the algorithm instances require the user to suggest the value for the next

variable, or ask for the worst value among several, we also show the *user's effort* in the various solvers, formally defined as the percentage of missing preferences the user has to consider to give the required help.

Besides the 16 instances of the scheme for IFCSPs described above, we also considered a "baseline" algorithm that elicits preferences of randomly chosen tuples every time branch and bound ends. All algorithms are named by means of the three parameters. For example, algorithm DPI.WORST. BRANCH has parameters $who = dpi$, $what = worst$, and $when = branch$. For the baseline algorithm, we use the name DPI.RANDOM.TREE.

For every choice of parameter values, 100 problem instances are generated. The results shown are the average over the 100 instances. Also, when it is not specified otherwise, we set $n = 10$ and $m = 5$. However, for IFCSPs, we have similar results for $n = 5$, 8, 11, 14, 17, and 20. All our experiments have been performed on an AMD Athlon 64x2 2800+, with 1 Gb RAM, Linux operating system, and using JVM 6.0.1.

Regarding the experimental tests for local search algorithms, we executed them using a step limit of 100000, a random walk probability of 0.2 and tabu tenure of 1000. All results are an average over 100 problem instances also in this case.

## A structured problem: the incomplete meeting scheduling problem

The meeting scheduling problem is a benchmark for CSPs [65], and we have adapted it to allow also for missing preferences.

A meeting scheduling problem (MSP) is informally the problem of scheduling some meetings by allowing the participants to attend all the meetings they are involved in. More formally, a MSP can be described by

- a set of agents;
- a set of meetings, each with a location and a duration;
- a set of time slots where meetings can take place;
- for each meeting, a subset of agents that are supposed to attend such a meeting;
- for each pair of locations, the time to go from one location to the other one.

Typical simplifying assumptions concern having the same duration for all meetings (one time slot), and the same number of meeting for each agent. To solve a MSP, we need to allocate each meeting in a time slot in a way that each agent can participate in his meetings. The only way that an agent cannot participate has to do with the time needed to go from the location of a meeting to the location of his next meeting.

The MSP can be easily seen as a CSP: variables represent meetings and variable domains represent all time slots. Each constraint between two meetings models the fact that one or more agents must participate in both meetings, and it is satisfied by all pairs of time slots that allow the participation to both meetings according to the time needed to pass between the corresponding locations. For this reason, it is often used as a typical benchmark for CSPs.

For our purposes, we consider a generalization of the MSP, called IMSP, where constraints are replaced by preferences and each agent may give only some of the preferences over the meetings he would like to attend. The preferences given by the agents over the time slots of each meeting are collected and then, for each time slot, the system assigns the average preference value if the majority of agents express a preference, otherwise it marks that preference as unknown. For each combination of time slots in binary constraints, the system sets as unknown the preference of a tuple if one of the values has an associated incomplete preference, otherwise it takes the average preference. The problem of solving an IMSP concerns finding time slots for the meetings such that all agents can participate and, among all possible solutions, to choose an optimal one according to some optimality criteria. In this context, given an IMSP $P$, necessarily optimal solutions (i.e., solutions in $NO(P)$) are meeting schedulings that are optimal no matter how the missing preferences are revealed. Thus, if there is at least one of such solutions, this is certainly preferred to any other.

We randomly generate meeting scheduling problems according to the following parameters:

- $m$: number of meetings (default 12);
- $n$: number of agents (default 5);
- $k$: number of meetings per agent (default 3);
- $l$: number of time slots (fixed to 10);
- $min$ and $max$: minimal (default 1) and maximal (default 2) distance in time slots between two locations;

- $i$: percentage of incomplete preferences (default 30%).

Given such parameters, we generate an IMSP with $m$ variables, representing the meetings, each with domain in $\{1 \dots l\}$ to represent the time slots from 1 to $l$. Such time slots are assumed to be all corresponding to one time unit and to be adjacent to each other. Given two time slots $a$ and $b$, they can be used for two meetings only if the distance between their locations is $b - a - 1$ or more.

For each of the $n$ agents, we generate randomly $k$ integers between 1 and $m$, representing the meetings he needs to participate in. Also, for each pair of time slots, we randomly generate a integer between $min$ and $max$ that represents the time needed to go from one location to the other one. This will be called the distance table.

Given two meetings, if there is at least one agent who needs to participate in both, we generate a binary constraint between the corresponding variables. Such a constraint is satisfied by all pairs of time slots that are compatible according to the distance table.

We then generate the preferences over the domain values according to parameter $i$. We then set the preference of each compatible pair in the binary constraints by assigning the average preference that the values in the pair have in their domain when both preferences are known. Otherwise, we assign an unknown preference.

As an example, assume to have $m = 5$, $n = 3$, $k = 2$, $l = 5$, $min = 1$, $max = 2$, and $i = 30$. According to these parameters, we generate an IMSP with the following features:

- 5 meetings: $m_1$, $m_2$, $m_3$, $m_4$, and $m_5$;
- 3 agents: $a_1$, $a_2$, and $a_3$;
- 5 time slots: $t_1, \dots, t_5$;
- agents' participation to meetings: we randomly generate 2 meetings for each agent, for example
  - $a_1$ must participate in meetings $m_1$ and $m_2$;
  - $a_2$ must participate in meetings $m_4$ and $m_5$;
  - $a_3$ must participate in meetings $m_2$ and $m_3$;
- distance table: we randomly generate its values, for example as in Table 3.1;

Table 3.1: Distance between meeting locations.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 1 | 2 |
| 2 | 1 | 0 | 2 | 1 | 2 |
| 3 | 2 | 2 | 0 | 1 | 1 |
| 4 | 1 | 1 | 1 | 0 | 2 |
| 5 | 2 | 2 | 1 | 2 | 0 |

- we randomly generate the preferences associated to the variable values in a way that 30% of the preferences are missing. Then, we compute the preferences of the compatible pairs in binary constraints as stated before.

### 3.6.1 Results for the systematic search approach

In this section we summarize and discuss our experimental comparison of the different algorithms. We first focus on Fuzzy ISCSPs. We then consider two special cases: incomplete CSPs where all constraints are hard, and incomplete fuzzy temporal problems. Furthermore, we consider incomplete weighted CSPs. Finally we show how algorithms perform on the incomplete meeting scheduling problem. In all the experimental results, the association between an algorithm name and a line symbol is shown in Figure 3.5.

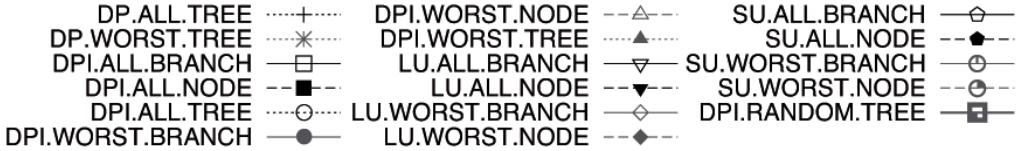



Figure 3.5: Algorithm names and corresponding line symbols.

#### Incomplete fuzzy CSPs

We first described the experiments on incomplete fuzzy CSPs. Figures 3.6 3.7 and 3.8 show the percentage of elicited preferences when we vary the incompleteness, the density, and the tightness, respectively. We show only the results for specific values of the parameters. However, the trends observed here hold in general. It is easy to see that the best algorithms are those that

elicit at the branch level. In particular, algorithm SU.WORST.BRANCH elicits a very small percentage of missing preferences (less than 5%), no matter the amount of incompleteness in the problem, and also independently of the density and the tightness. This algorithm outperforms all others, but relies on help from the user. The best algorithm that does not need such help is DPI.WORST.BRANCH. This never elicits more than about 10% of the missing preferences. Notice that the baseline algorithm is always the worst one, and needs nearly all the missing preferences before it finds a necessarily optimal solution. Notice also that the algorithms with $what = worst$ are almost always better than those with $what = all$, and that $when = branch$ is almost always better than $when = node$ or $when = tree$.

Figure 3.9 shows the user's effort as incompleteness varies. As could be predicted, the effort grows slightly with the incompleteness level, and it is equal to the percentage of elicited preferences only when $what = all$ and $who = dp$ or $dpi$. For example, when $what = worst$, even if $who = dp$ or $dpi$, the user has to consider more preferences than those elicited, since to identify the worst preference value the user needs to check all of them (that is, those involved in a partial or complete assignment). DPI.WORST.BRANCH requires the user to look at 60% of the missing preferences at most, even when incompleteness is 100%.

Figure 3.10 shows the user's effort as density varies. Also in this case, as expected, the effort grows slightly with the density level. In this case DPI.WORST.BRANCH requires the user to look at most 40% of the missing preferences, even when the density is 80%.

All these algorithms have a useful anytime property, since they can be stopped even before their end obtaining a possibly optimal solution with preference value higher than the solutions considered up to that moment. Figure 3.11 shows how fast the various algorithms reach optimality. The $y$ axis represents the solution quality during execution, normalized to allow for comparison among different problems. The algorithms that perform best in terms of elicited preferences, such as DPI.WORST.BRANCH, are also those that approach optimality fastest. We can therefore stop such algorithms early and still obtain a solution of good quality in all completions.

Figure 3.12 (a) shows the percentage of elicited preferences (white part of the bar) over all the preferences (white + grey part), as well as the user's effort (black part) for DPI.WORST.BRANCH. Even with high levels of incompleteness, this algorithm elicits only a very small fraction of the preferences, while asking the user to consider at most half of the missing preferences. For example, with incompleteness at 60%, the user effort is at less than 30% and the elicited preferences are at less than 10%

Figure 3.12 (b) shows results for LU.WORST.BRANCH, where the user

is involved in the choice of the value for the next variable. Compared to DPI.WORST.BRANCH, this algorithm is better both in terms of elicited preferences and user's effort (while SU.WORST.BRANCH is better only for the elicited preferences). We conjecture that the help the user gives in choosing the next value guides the search towards better solutions, thus resulting in an overall decrease of the number of elicited preferences.

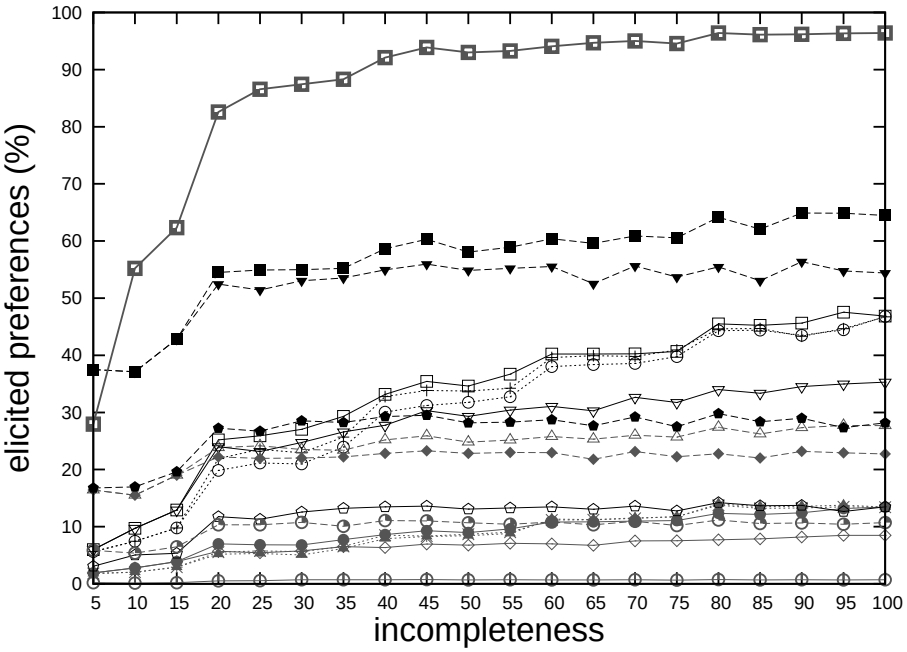



Figure 3.6: Percentage of elicited preferences in incomplete fuzzy CSPs, varying incompleteness ($d$=50%, $t$=10%).

### Incomplete CSPs

We also tested these algorithms on incomplete hard CSPs. In this case, preferences are only 0 and 1, and necessarily optimal solutions are complete assignments which are feasible in all completions. The problem generator is adapted accordingly. The parameter *what* now has a specific meaning: $what = worst$ means asking if there is a 0 among the considered missing preferences. If there is no 0, we can infer that all the considered missing preferences are 1s.

Figures 3.13, 3.14 and 3.15 show the percentage of elicited preferences in terms of amount of incompleteness, density, and tightness respectively. Notice that the scale on the $y$ axis varies to include the highest values. The

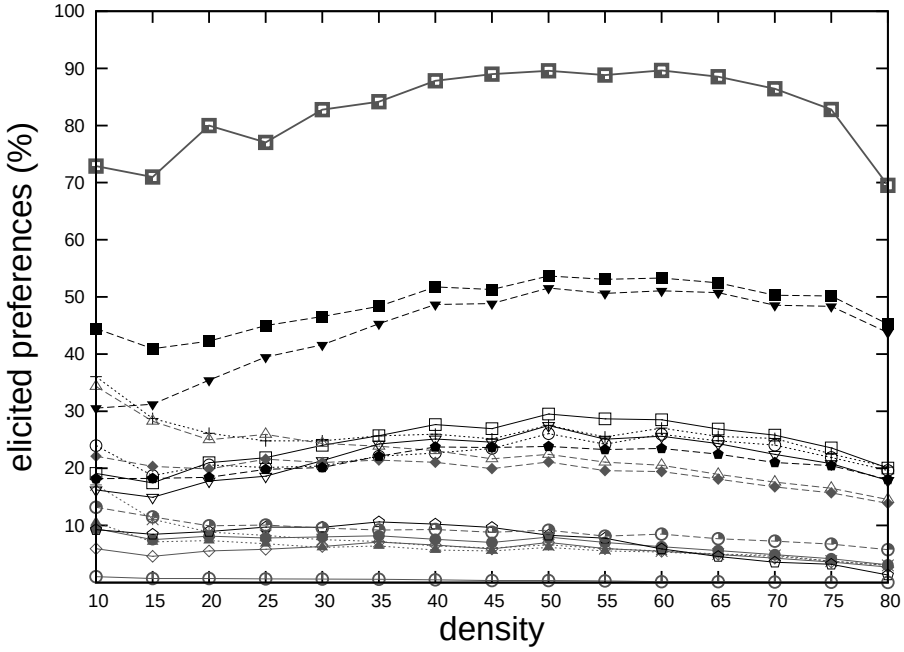



Figure 3.7: Percentage of elicited preferences in incomplete fuzzy CSPs, varying density ($t$=35%, $i$=30%).

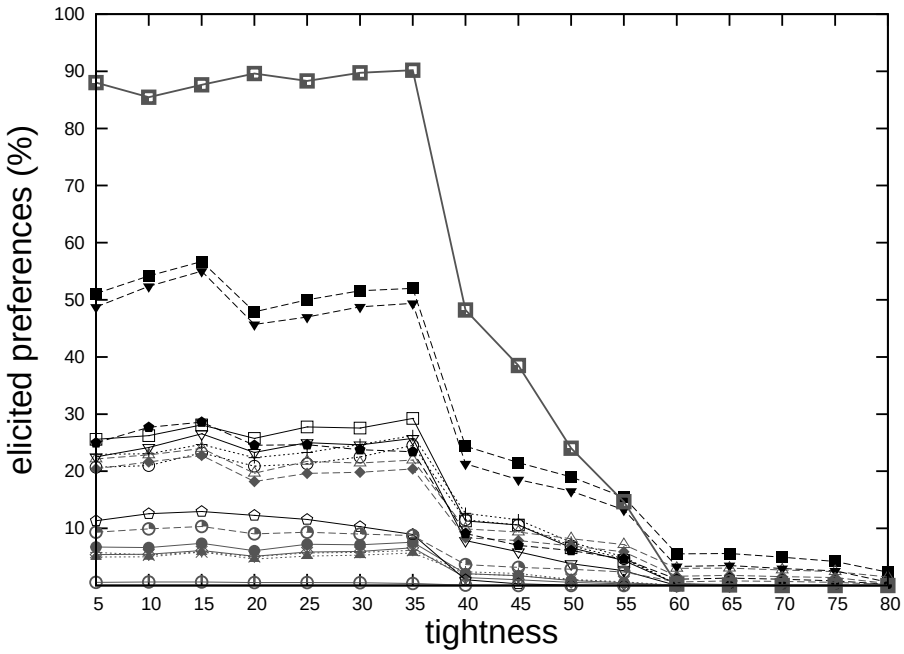



Figure 3.8: Percentage of elicited preferences in incomplete fuzzy CSPs, varying tightness ($d$=50%, $i$=30%).

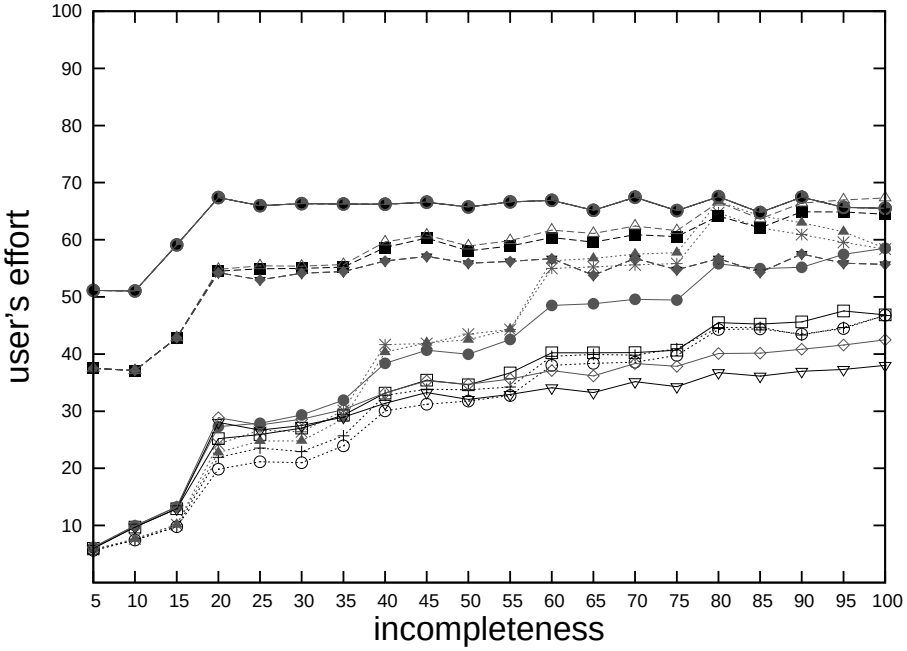



Figure 3.9: Incomplete fuzzy CSPs: user's effort varying incompleteness ($d$=50%, $t$=10%).

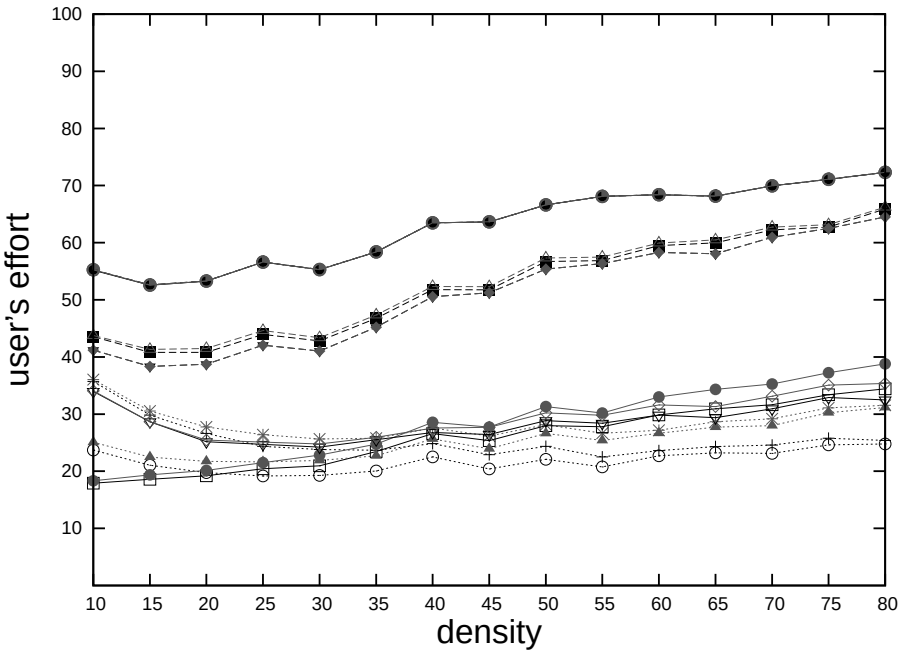



Figure 3.10: Incomplete fuzzy CSPs: user's effort varying density ($t$=10%, $i$=30%).

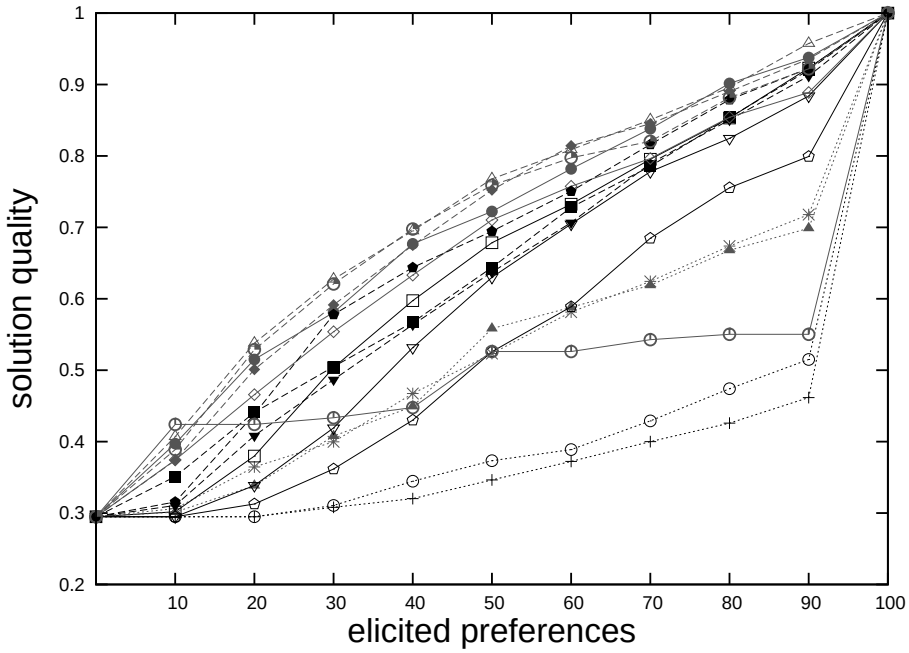


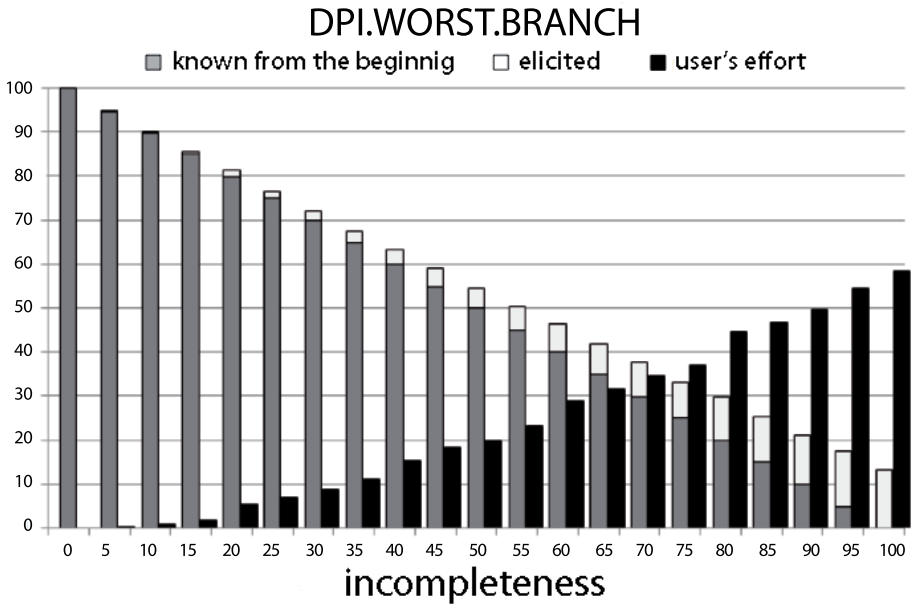

Figure 3.11: Incomplete fuzzy CSPs: solution quality.

best algorithms are those with $what = worst$, where the inference explained above about missing preferences can be performed. It is easy to see a phase transition at tightness about 35% , which is when problems pass from being solvable to having no solutions. However, the percentage of elicited preferences is below 20% for all algorithms even at the peak.

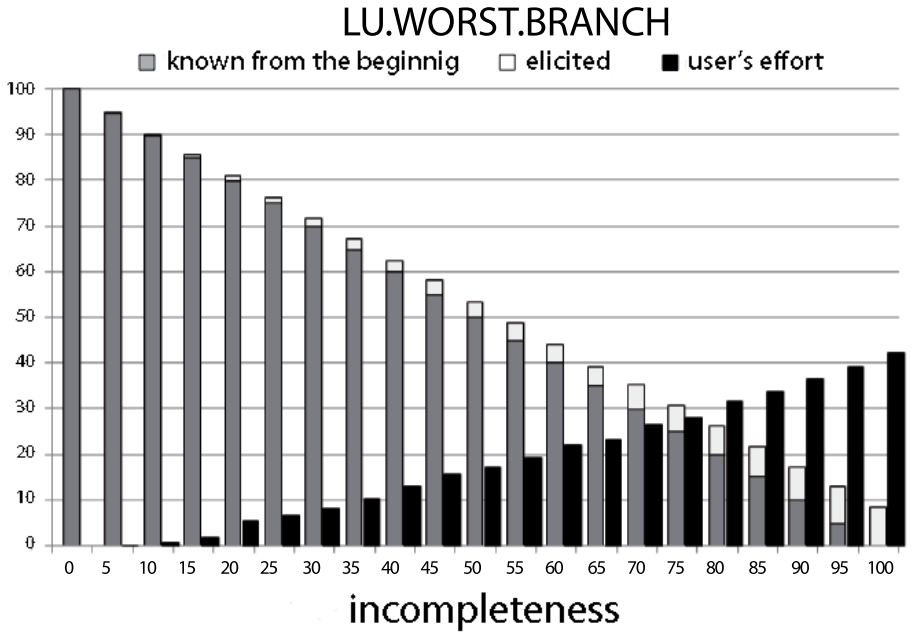
Figure 3.16 shows the user's effort in terms of amount of incompleteness and in terms of density. Overall, the best algorithm is again DPI.WORST. BRANCH, whose percentage of elicited preferences and users effort are shown in Figure 3.17 in detail. In this figure we also show the percentage of 1s that are inferred by the system (light grey area). It is possible to note that also with the 100% of missing preferences the user's effort is below 22%.

### Incomplete temporal fuzzy CSPs

We also performed some experiments on fuzzy simple temporal problems [46]. In such problems variables represent instantaneous events and constraints model time intervals for durations and distances of such events. Moreover, it is possible to associate a fuzzy preference to each possible duration or distance. Thus, a fuzzy temporal constraint on variables $X$ and $Y$ has the form $\langle [a, b], f \rangle$ where $[a, b]$ is an interval such that $a \leq Y - X \leq b$ and $f$ is a preference function associating a preference in [0,1] to each value in $[a, b]$.




(a) $d$=50%, $t$=10%




(b) $d$=50%, $t$=10%

Figure 3.12: Incomplete fuzzy CSPs: best algorithms.

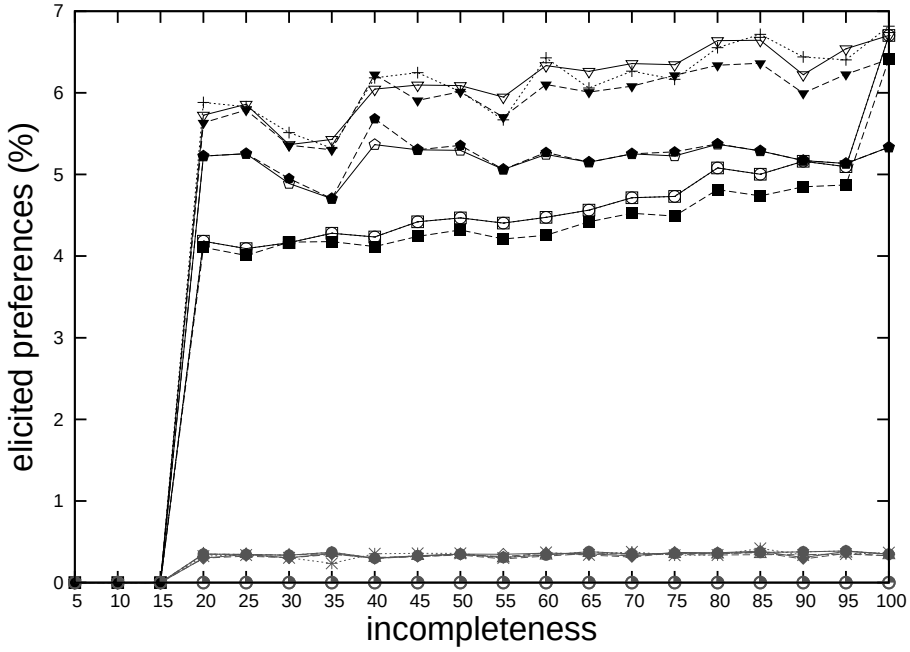



Figure 3.13: Percentage of elicited preferences in incomplete CSPs, varying incompleteness ($d$=50%, $t$=10%).

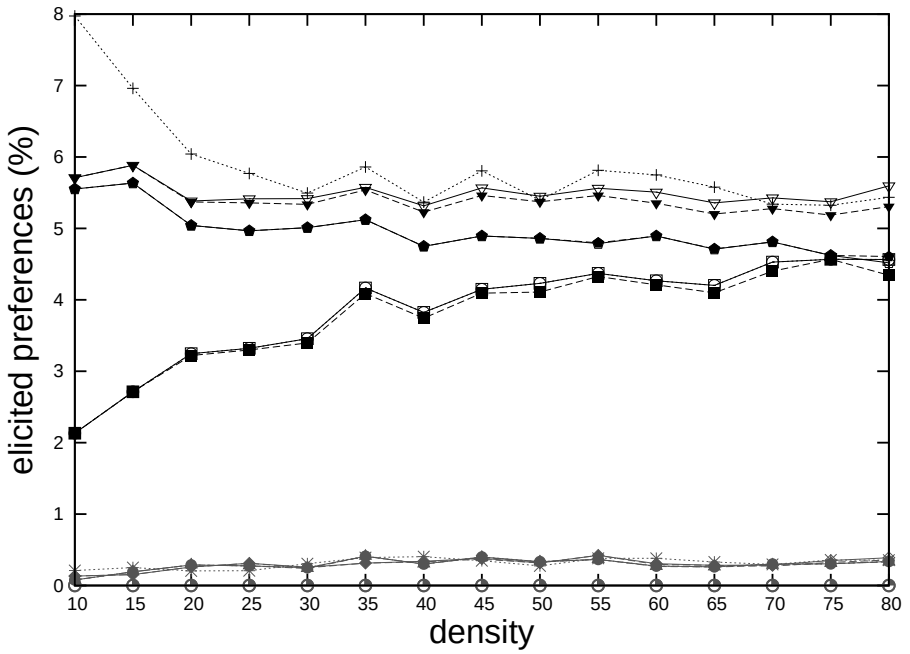



Figure 3.14: Percentage of elicited preferences in incomplete CSPs, varying density ($t$=10%, $i$=30%).

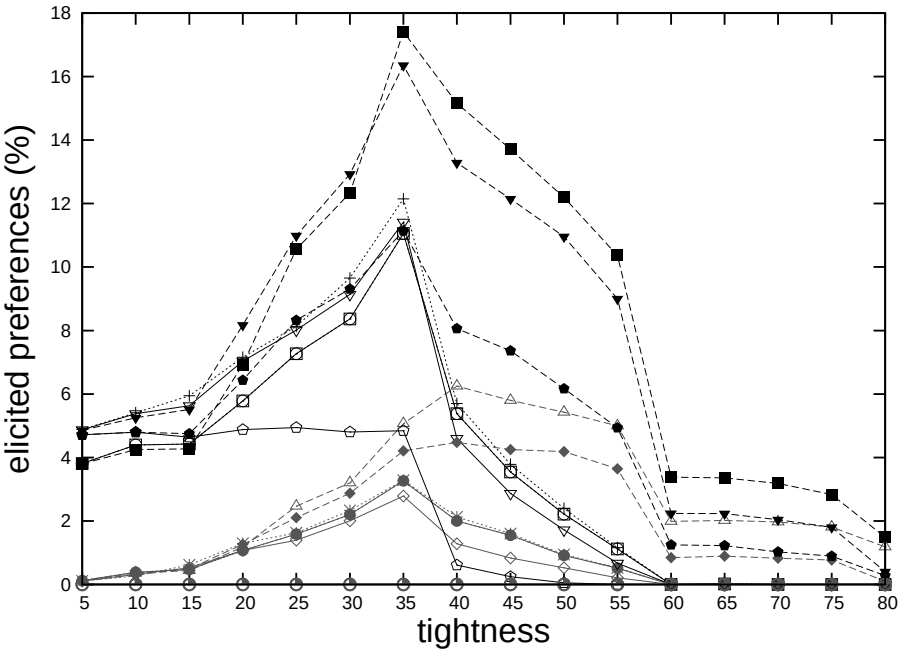


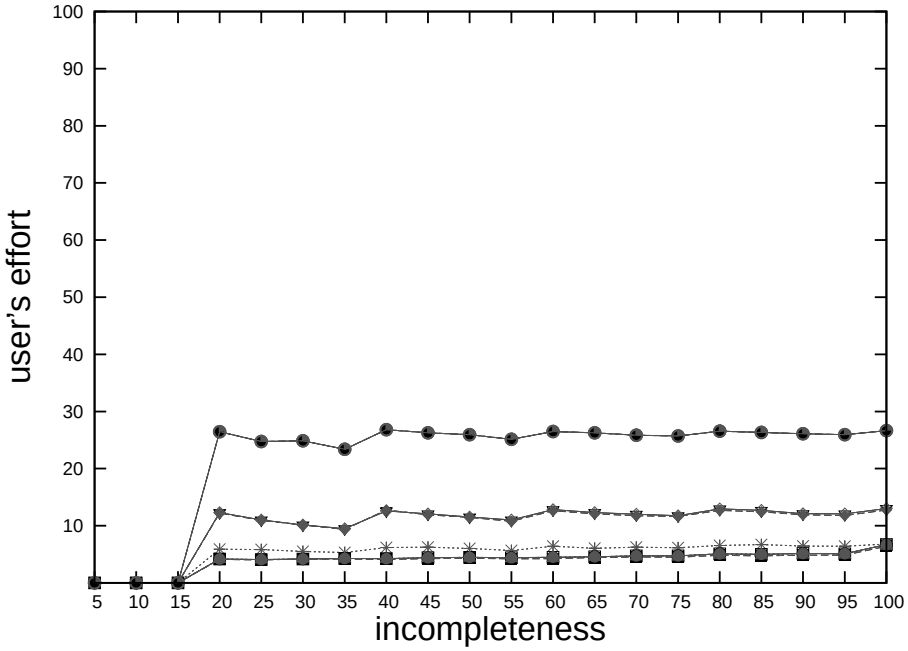

Figure 3.15: Percentage of elicited preferences in incomplete CSPs, varying tightness ($d$=50%, $i$=30%).

Fast consistency-based solvers have been developed for a tractable subclass of such problems, where all the preference functions are semi-convex[46]. Such solvers are however unable to deal with missing preferences since they make the problems intractable in general. We have thus decided to experiment on this class of problems our branch-and-bound-based techniques. In fact, in addition to the value of testing on problems with such a specific structure, the large amount of information required by the specification of such problems makes missing preferences very likely to appear in practice.

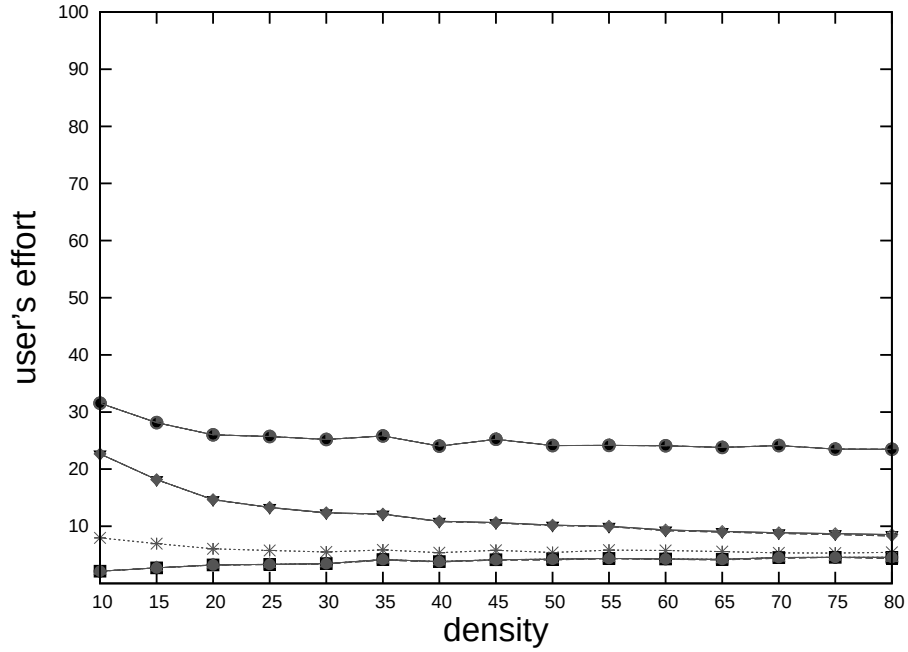
We have generated classes of such problems following the approach in [46], adapted to consider incompleteness. Figure 3.18 shows that even in this domain it is possible to find a necessarily optimal solution by asking about 10% of the missing preferences, for example via algorithm DPI.WORST.BRANCH.

### Incomplete weighted CSPs

We considered incompleteness also on weighted CSPs (WCSPs). WCSPs model optimization problems where the goal is to minimize the total cost (time, space, number of resources, etc...) of the proposed solution. To handle these problems, we instantiated our general framework using the $\langle A, min, +, +\infty, 0\rangle$ c-semiring. In a similar way as in IFCSPs, in Incom-

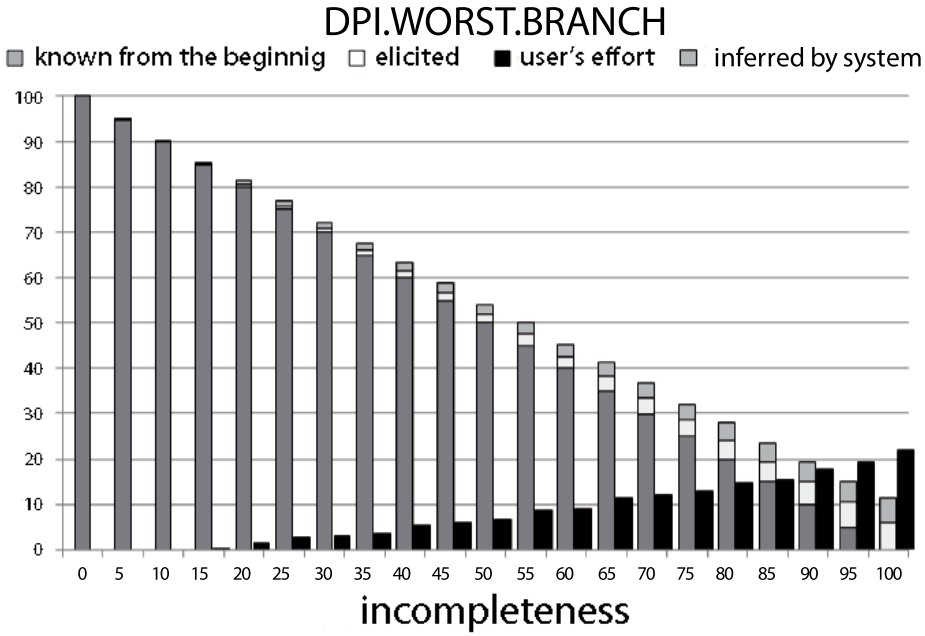



(a) $d$=50%, $t$=10%




(b) $t$=10%, $i$=30%

Figure 3.16: Incomplete CSPs: user's effort.




(a) $d$=50%, $t$=10%

Figure 3.17: Incomplete CSPs: best algorithm.

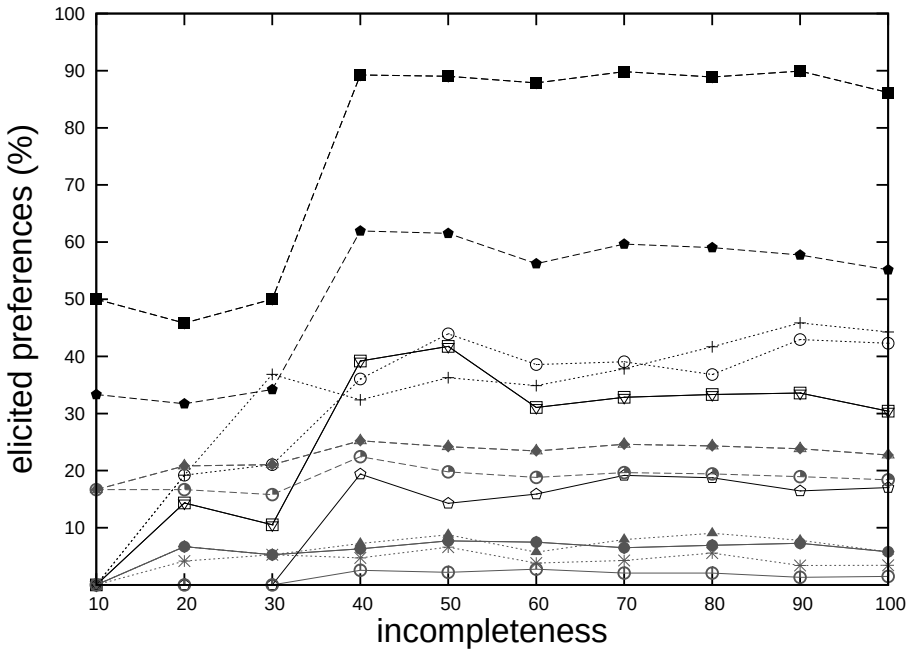



Figure 3.18: Percentage of elicited preferences in incomplete fuzzy temporal CSPs.

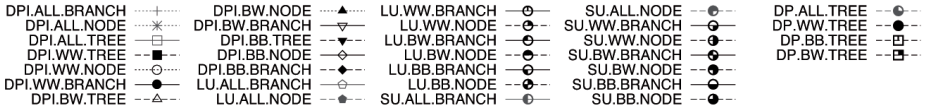



Figure 3.19: Algorithms for IWCSPs.

plete Weighted CSPs (IWCSPs) some costs, associated with each tuple, are missing. In this case the multiplicative operator is not idempotent (as in the fuzzy setting) and thus, we have to know all the missing costs associated with a given assignment to obtain its global cost.

To adapt our algorithms to deal with IWCSPs, it is necessary to develop new elicitation strategies optimized for this new environment. We defined three different ways to ask the user for the missing costs. The resulting strategies, as already defined in Section 3.4, are identified with the following values for the *what* parameter: WW, BB, and BW. Notice that, with WW, we elicit the worst missing cost (that is, the highest) until either all the costs are elicited or the current global cost of the (possibly partial) assignment is higher than the optimum found so far. By doing this, we know if the global cost exceeds the optimum as early as possible. With BB, we elicit the best (i.e. the minimum) cost until either all the costs are elicited or the current global cost of the (possibly partial) assignment is higher than the optimum found so far. Knowing the best cost of a given assignment allows the system to infer that all the other missing costs are at least as high as the last one elicited. This inference allows us to update the **1**-completion during the search by lowering the upper bound of the missing costs. In this way, we overestimate the real value of the unknown costs. Let ${P_1}^*$ the **1**-completion updated with the inferred costs. Given an assignment $s$, $\text{pref}(P_1, s) \geq_S \text{pref}({P_1}^*, s) \geq_S \text{pref}(P', s)$ where $P'$ is $P$ in which all the incomplete tuples of $s$ are elicited. With BW, we elicit in turn the best and the worst cost. In this case we want to test empirically if the combination of the previous two strategies is better in practice.

In all the experimental results, the association between an algorithm name and a line symbol is shown in Figure 3.19.

We tested our algorithms on randomly generated IWCSPs, and we used the $what = all$ method as a baseline because it is the most general elicitation strategy that can be applied in every possible setting: Hard CSPs, Fuzzy ISCSPs, etc.

The randomly generated IWCSPs have the same default parameter values as in the IFCSPs experiments, except for the tightness that has a default value of 25%. We choose this value because it is near to the middle of the

interval from 0 to 40%, where we will see a phase transition in the percentage of elicited tuples (Figure 3.22). We recall that each cost has a value in $[0, 10] \cup \{+\infty\}$.

Figures 3.20, 3.21 and 3.22 show the percentage of elicited preferences as we vary density, incompleteness and tightness respectively. As expected, the number of elicited tuples increases with the incompleteness of the problem (see Figure 3.20). As we vary the tightness (Figure 3.22), we can observe a phase transition at t=40%. At that point, most of the problems have an optimal solution with infinite cost, thus the algorithms do not need more information to find that extreme solution.

When the density increases (Figure 3.21), the percentage of elicited costs tends to decrease slightly. This may be surprising, since, increasing the number of constraints, the number of incomplete tuples increases, thus the percentage of elicited costs should increase. However, the number of infinite costs increases together with the incompleteness. In this particular case, using t=25% and i=30%, the number of infinite cost values increases enough to make the problem easier to solve, thus it requires less elicitation. We performed other experiments decreasing the default tightness value to t=10%. In this case the percentage of elicited values increases slightly, because there are not enough infinite costs to make the problem easier to solve. On the other hand, the number of incomplete tuples increases with density, making the problem harder to solve (thus requiring more elicitation). Summarizing, increasing density, both incompleteness and tightness increase. When t=25% the contribution of the tightness is more important than incompleteness and the problems become easier to solve. On the contrary, when t=10%, the contribution of the incompleteness is greater and thus the problems becomes harder.

It easy to see that the best algorithms are those with $when = branch$ and $who = su$. These algorithms ask for only around 30% of the costs needed to solve a totally incomplete problem. The parameter $who = su$ forces the user to select the value to instantiate, which implies an additional effort by the user. This behavior is depicted in Figure 3.24(b) where the algorithm elicits a very small number of tuples but the user has to check almost all the incomplete tuples every time.

Among the algorithms where the user does not help the system in the value instantiation, the algorithm that elicits less values is DPI.BW.TREE (see Figure 3.23(a)).

DPI.BW.TREE elicits less that half of the unknown costs on 100% of incompleteness, requiring the user to look at 70% of incomplete tuples to answer the system's query. All our experiments shown that iteratively asking the user for the best and the worst preference of a given assignment is the

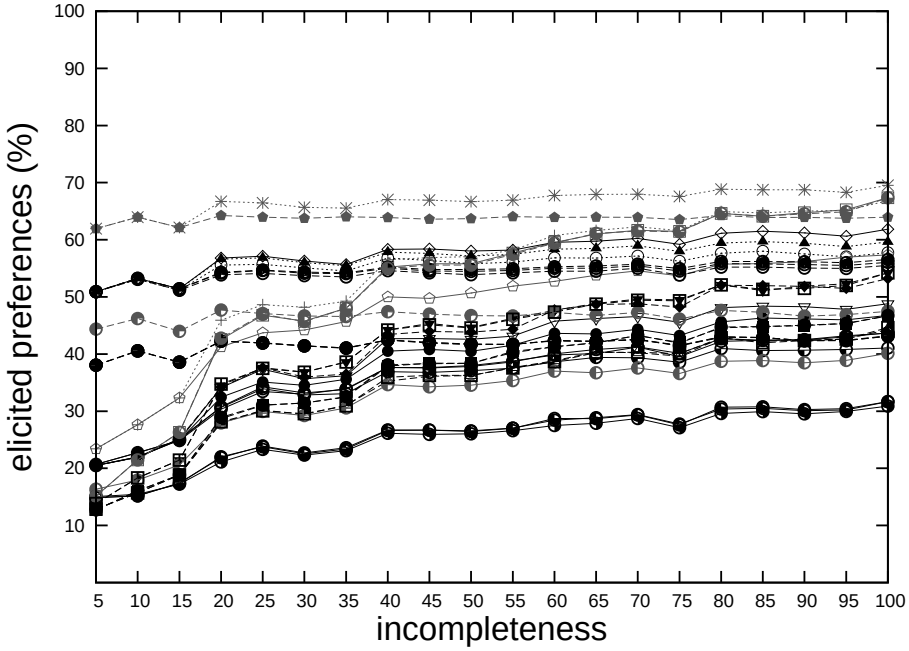



Figure 3.20: Percentage of elicited preferences in incomplete weighted CSPs, varying incompleteness ($d$=50%, $t$=25%).

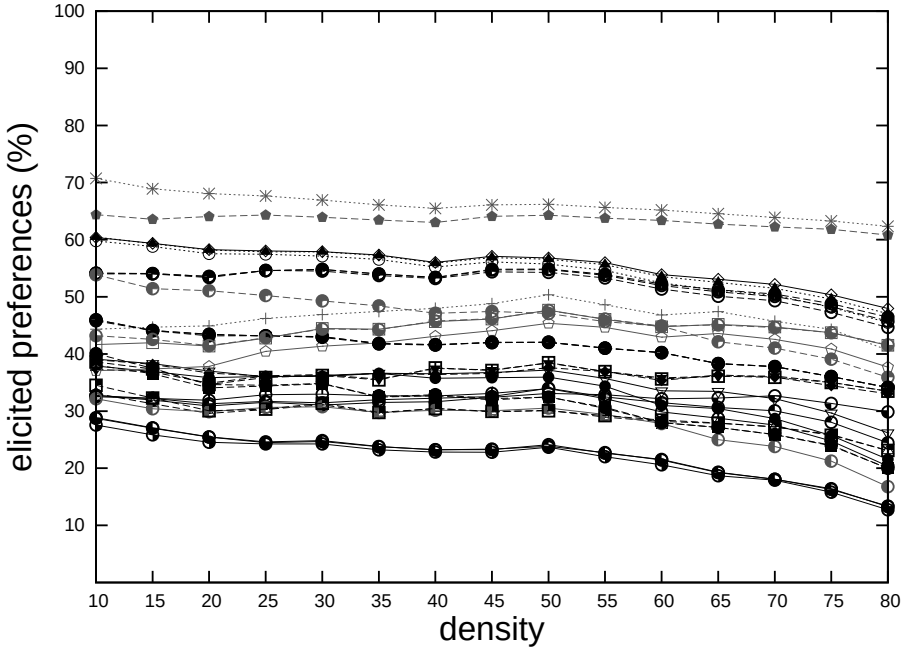



Figure 3.21: Percentage of elicited preferences in incomplete weighted CSPs, varying density ($t$=25%, $i$=30%).

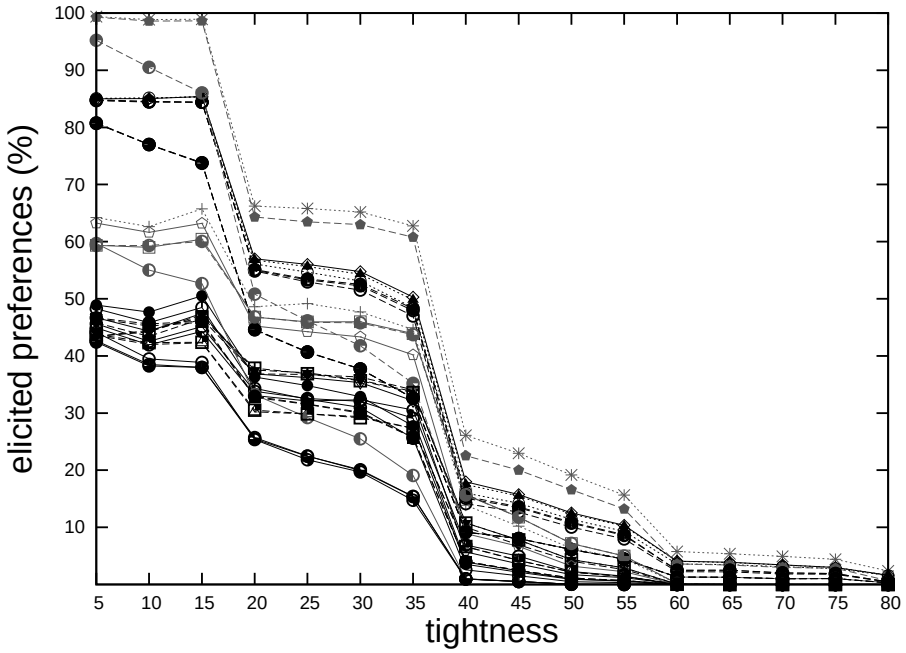
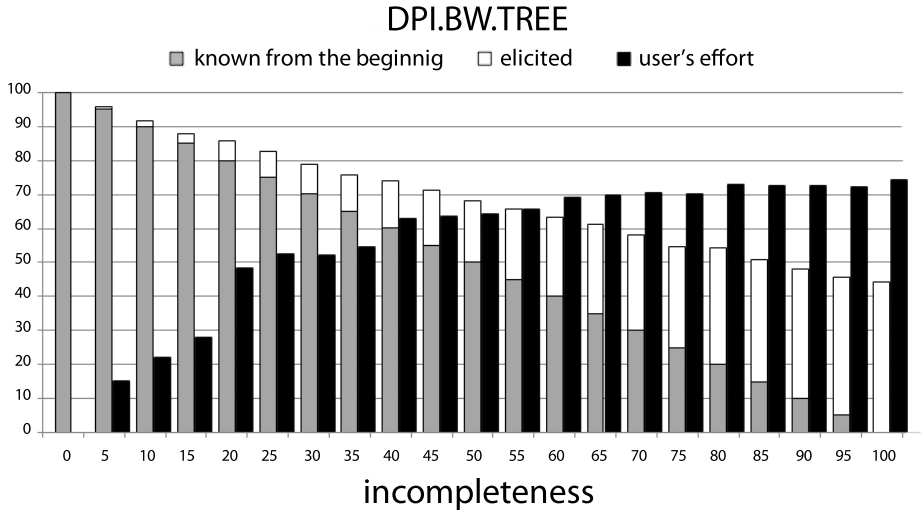



Figure 3.22: Percentage of elicited preferences in incomplete weighted CSPs, varying tightness ($d$=50%, $i$=30%).

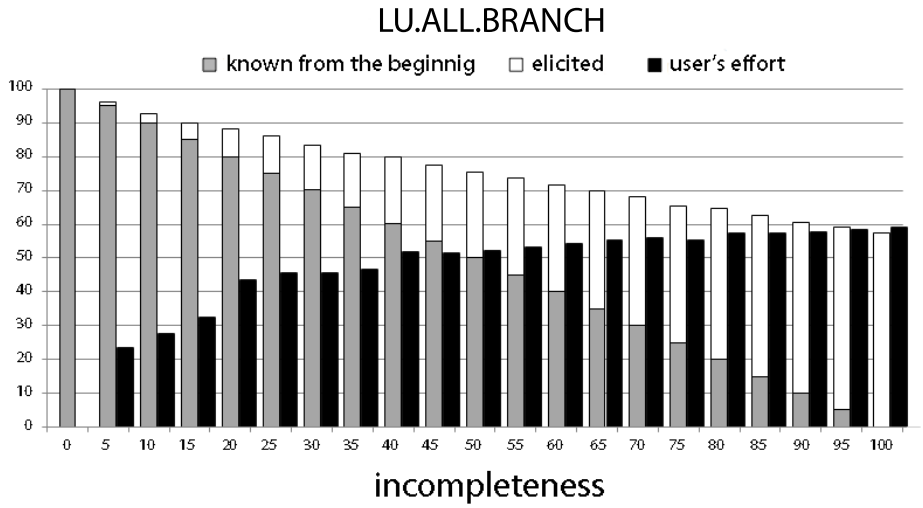
best compromise when the value instantiation is totally done by the system. On the other hand, it forces the user to consider more than 70% of the incomplete tuples.

In situations where we want to minimize the user effort, the best choice is the LU.ALL.BRANCH algorithm (see Figure 3.23(b)). As shown in Figure 3.25, the user does less work with LU.ALL.BRANCH than with the other algorithms when the incompleteness is varying. We obtain the same result when the density or the tightness is varying (see Figures 3.26, 3.27). If we want a balance between the percentage of elicited costs and the user effort, the best compromise is LU.BB.BRANCH. Figure 3.24(a) shows that, on IWCSPs with no initial costs, it elicits 40% of incomplete tuples with a user effort of about 60%.

Summarizing, SU.WW.BRANCH (Figure 3.24(b)) is the algorithm which elicits less tuples but, if we want the user to just answer the elicitation queries, the best algorithm is DPI.BW.TREE. The algorithm that minimizes the user effort is LU.ALL.BRANCH, whereas the best compromise between user effort and elicitation is LU.BB.BRANCH.

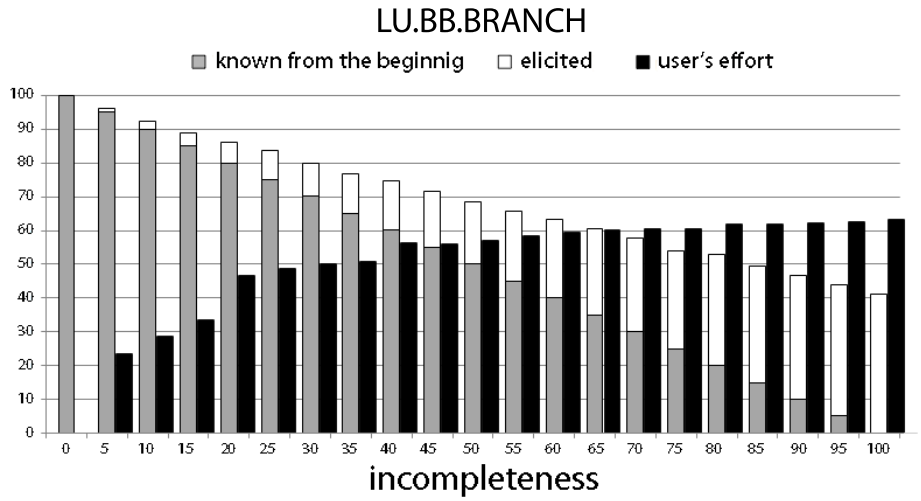



(a) $t$=25% and $d$=50%

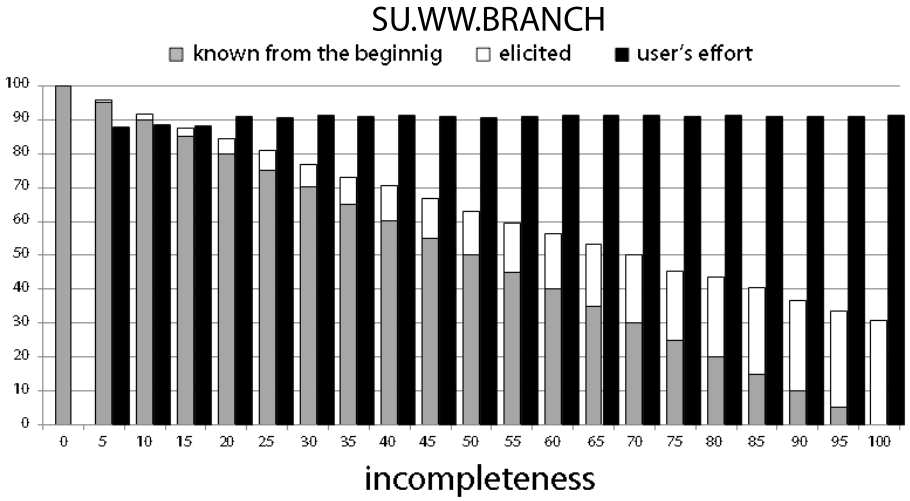



(b) $t$=25% and $d$=50%

Figure 3.23: Best algorithms for incomplete weighted CSPs.




(a) $t$=25% and $d$=50%




(b) $t$=25% and $d$=50%

Figure 3.24: Best algorithms for incomplete weighted CSPs.

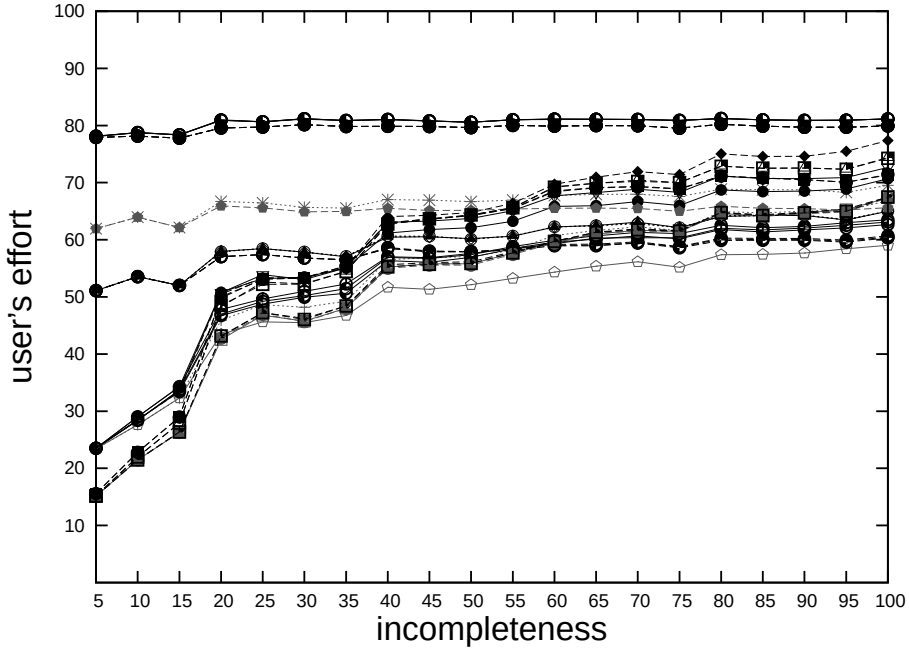



Figure 3.25: User's effort in incomplete weighted CSPs, varying incompleteness ($d$=50%, $t$=25%).

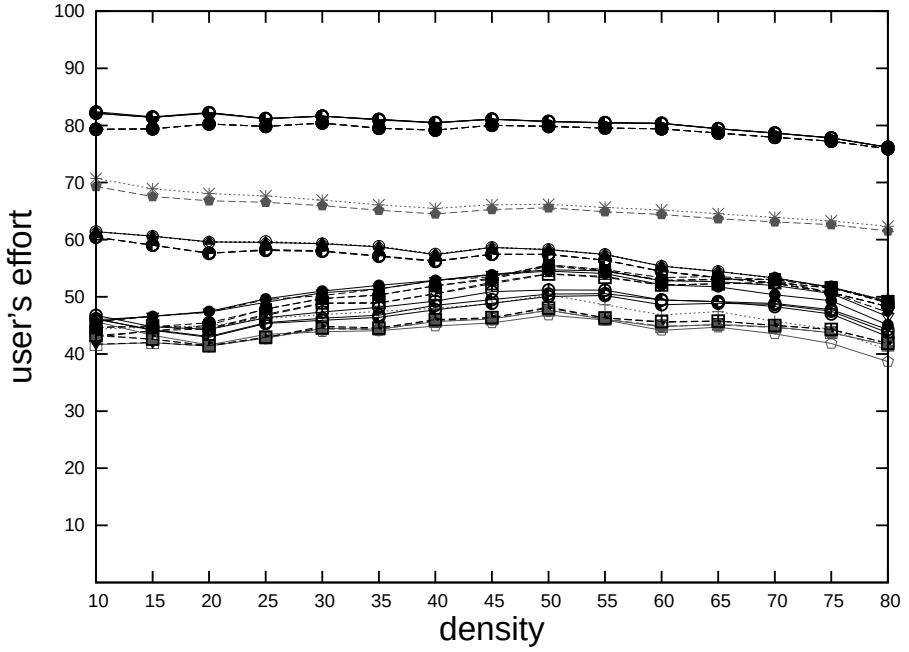



Figure 3.26: User's effort in incomplete weighted CSPs, varying density ($t$=25%, $i$=30%).

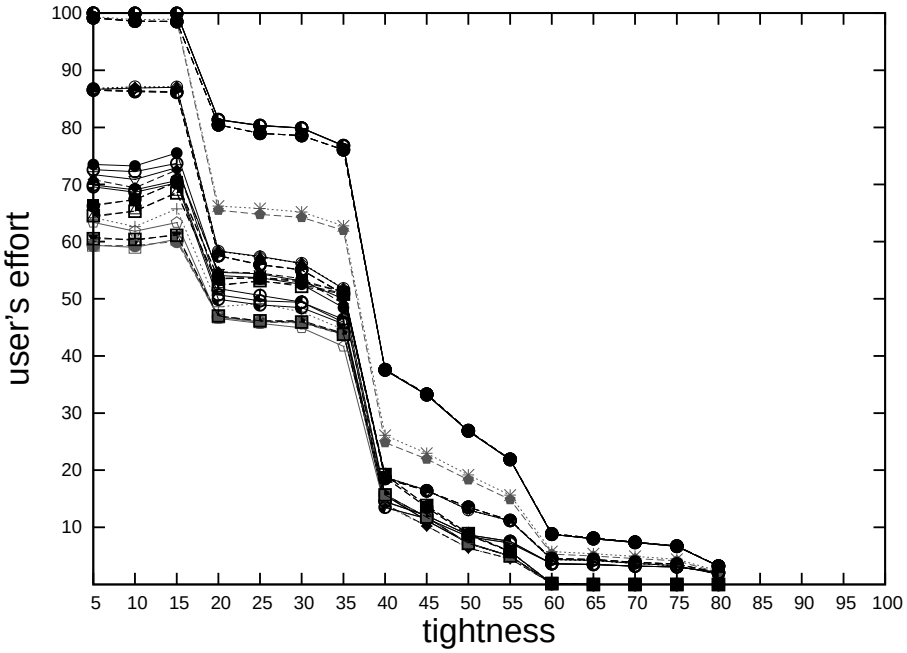



Figure 3.27: User's effort in incomplete weighted CSPs, varying tightness ($d$=50%, $i$=30%).

### Incomplete meeting scheduling problems

In this section we consider incomplete meeting scheduling problems with fuzzy preferences in order to evaluate our algorithms on problems with more structure w.r.t. randomly generated problems. Hence, to deal with such problems, we have instantiated our framework using the fuzzy c-semiring $(\langle A, +, \times, 0, 1\rangle)$. We randomly generate IMSPs by varying the number of meetings $m$ from 5 to 14, the number of agents $n$ from 4 to 10, the number of meetings per agent $k$ from 2 to 5, and the percentage of incompleteness $i$ from 10% to 100%.

We measure the percentage of the elicited tuples and the user's effort. Figures 3.28, 3.29, 3.30 and 3.31 show the percentage of elicited preferences as we vary the number of agents, the number of meetings per agent, the number of meetings, and the amount of incompleteness respectively. As expected, as the incompleteness in the problem rises, the percentage of elicited tuples tends to increase slightly (see Figure 3.28). The algorithms that elicit less tuples in this context are SU.WORST.BRANCH and LU.WORST.BRANCH (less than 1% of incomplete tuples), among the ones that need the user's help to choose the values to instantiate, and DPI.WORST.BRANCH (less than 5% of incomplete tuples) among the others. The common features of these

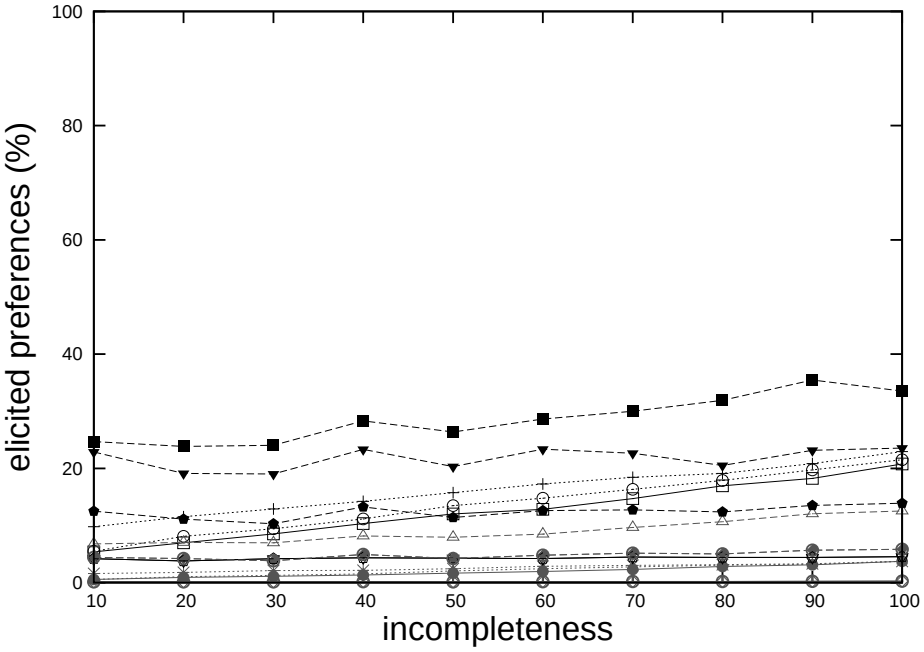



Figure 3.28: Percentage of elicited preferences in IMSPs, varying incompleteness ($m$=12, $n$=5, $k$=3, $l$=10, $min$=1, $max$=2).

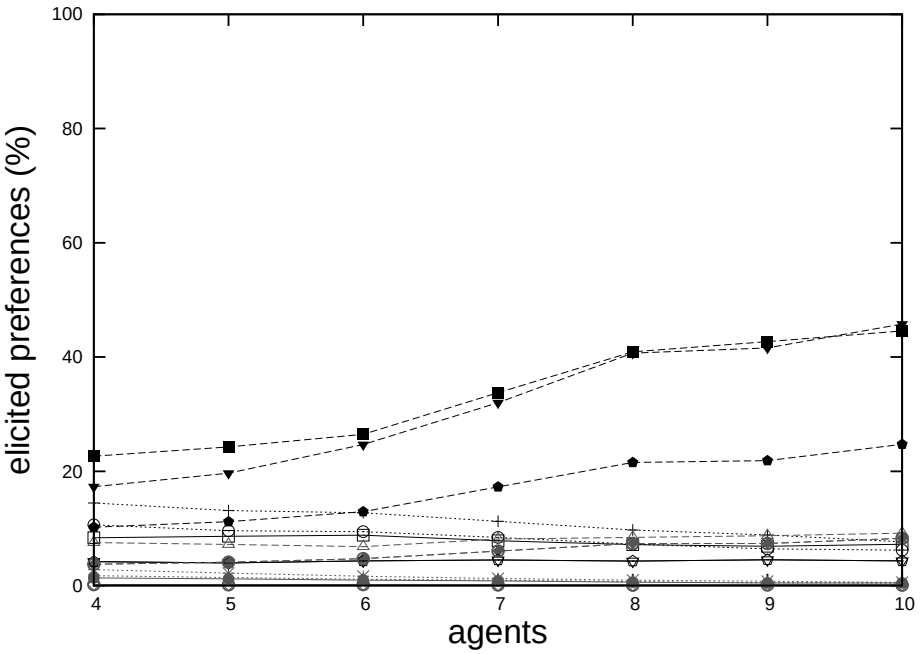



Figure 3.29: Percentage of elicited preferences in IMSPs, varying the number of agents ($m$=12, $k$=3, $i$=30%, $l$=10, $min$=1, $max$=2).

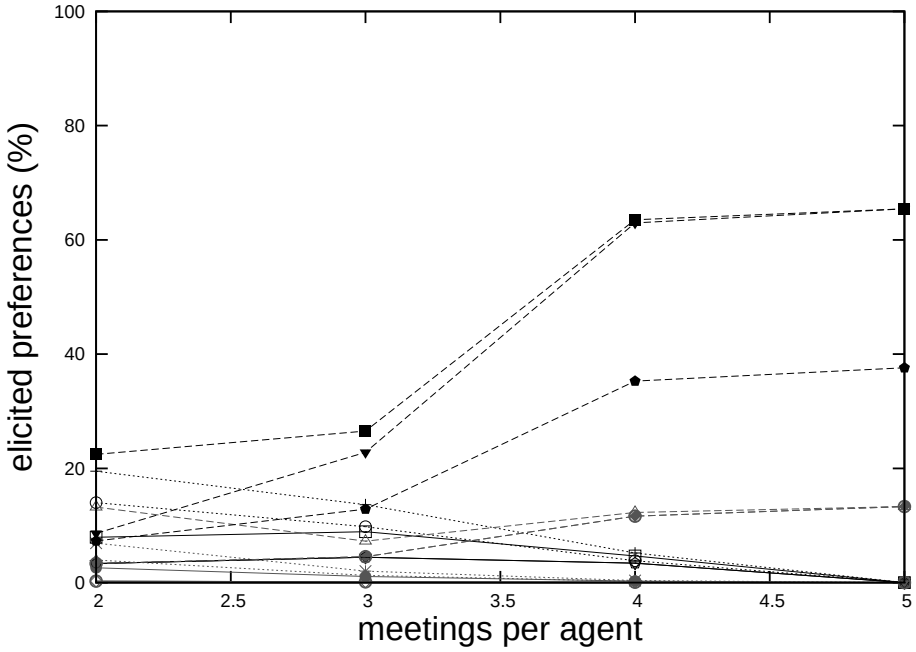



Figure 3.30: Percentage of elicited preferences in IMSPs, varying the meetings per agent ($m$=12, $n$=5, $i$=30%, $l$=10, $min$=1, $max$=2).

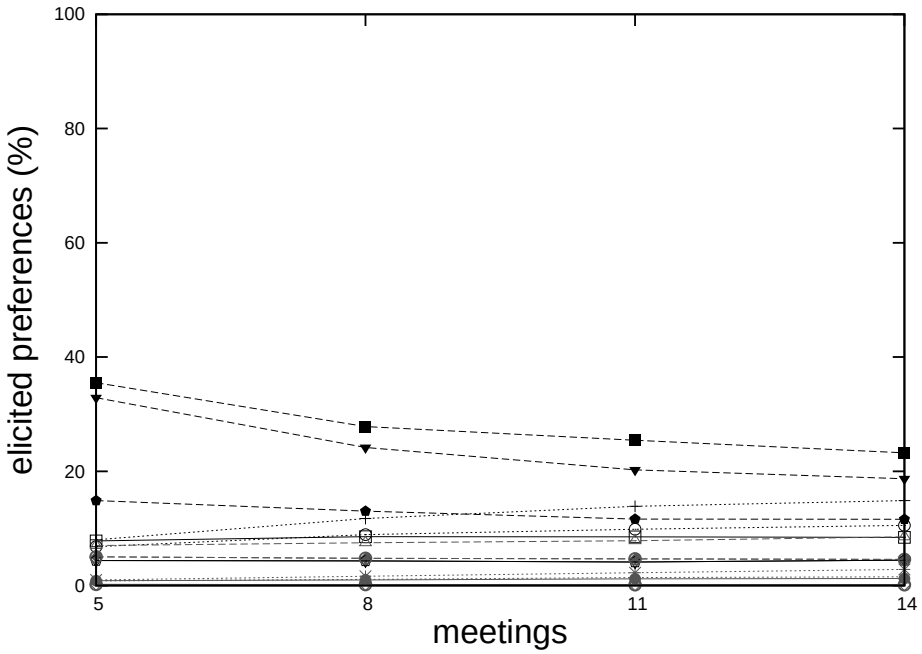



Figure 3.31: Percentage of elicited preferences in IMSPs, varying the number of meetings ($n$=5, $k$=3, $i$=30%, $l$=10, $min$=1, $max$=2).

algorithms are that they elicits the worst preference every time they reach a complete assignment.

If we vary the number of agents (Figure 3.29) we see that, for the best algorithms, the percentage of elicited preference does not vary as the number of agents increases. Furthermore, the two best algorithms mentioned before are among the bests also when we vary the number of agents.

Figure 3.30 shows the percentage of elicited preferences as we vary the meetings per agent. If we consider the best algorithms found so far (i.e. SU.WORST.BRANCH, LU.WORST.BRANCH and DPI.WORST.BRANCH), they tend to elicit less tuples as the meetings per agent rises. They reach practically the 0% at 5 meetings per agent because, in that setting, the optimal preference is 0 and so, they need to ask only for the preferences to fix the lower bound in the first branch and then they do not reach a complete assignment again. Finally, we can see from Figure 3.31 that our best algorithms need more or less the same amount of elicitation (in percentage) as the number of meetings increases.

Summarizing, the best algorithms, in terms of elicited tuples, are the ones with *when* = *branch* and *who* = *su* or *lu* if we allow the user to instantiate values, and *who* = *dpi* if the instantiation is done by the system.

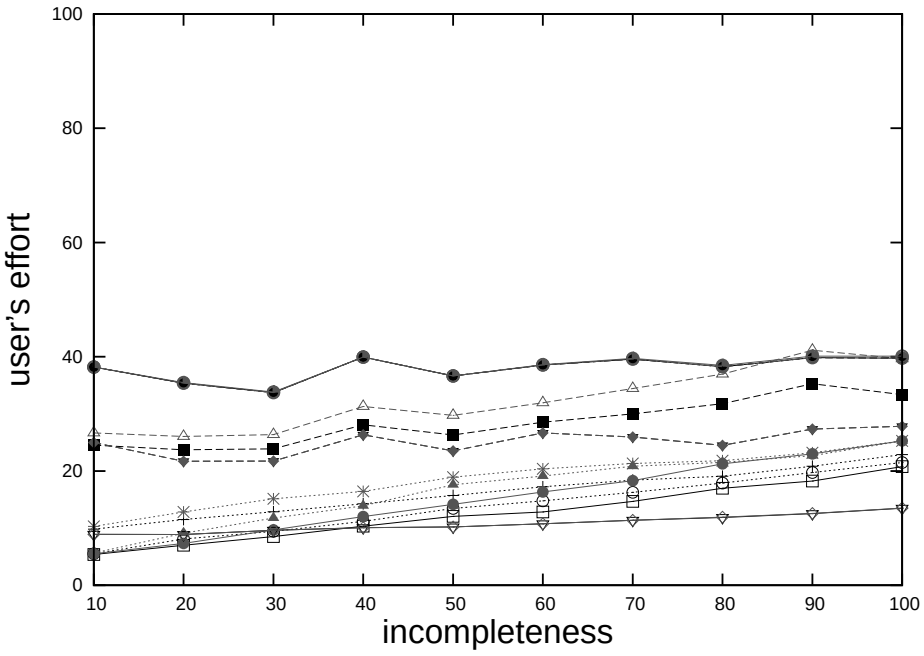



Figure 3.32: User's effort in IMSPs, varying incompleteness ($m$=12, $n$=5, $k$=3, $l$=10, $min$=1, $max$=2).

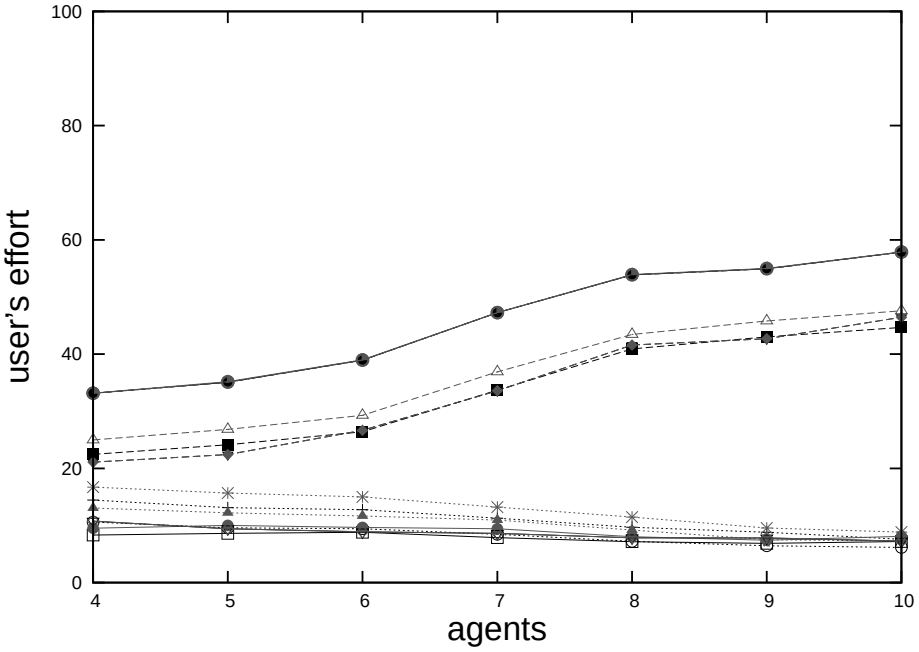



Figure 3.33: User's effort in IMSPs, varying the number of agents ($m$=12, $k$=3, $i$=30%, $l$=10, $min$=1, $max$=2).

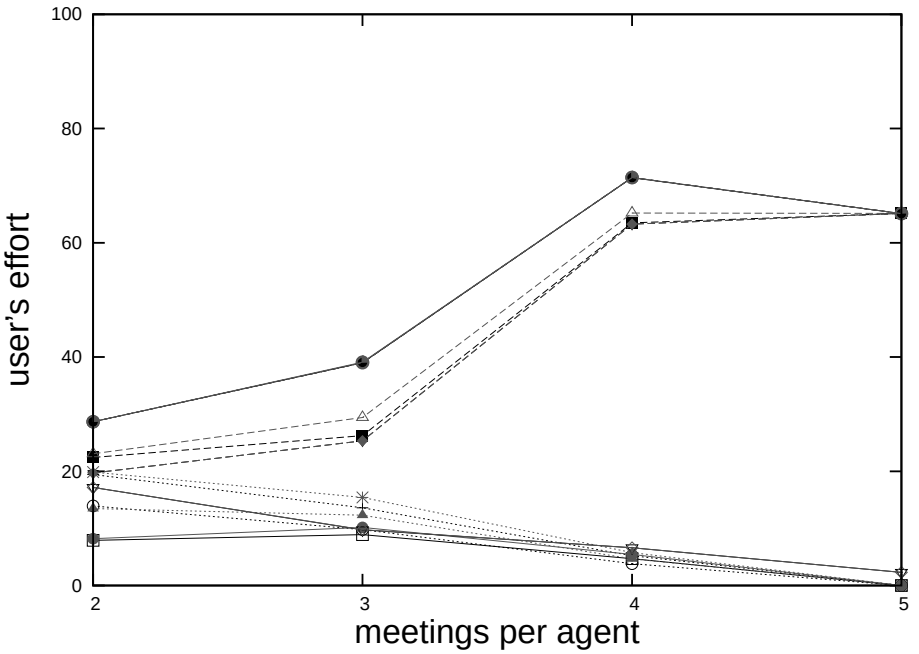



Figure 3.34: User's effort in IMSPs, varying meetings per agent ($m$=12, $n$=5, $i$=30%, $l$=10, $min$=1, $max$=2).

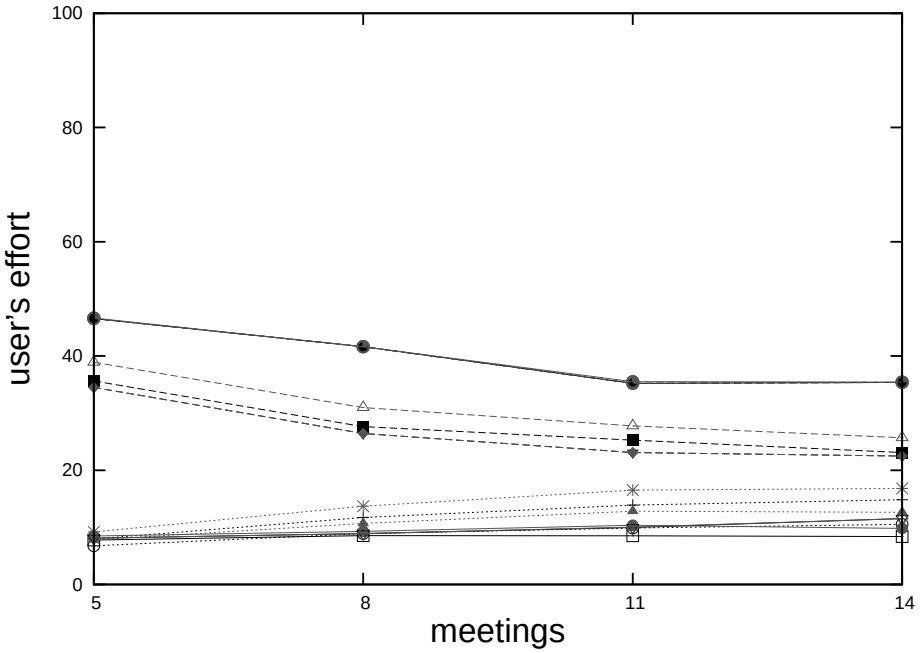



Figure 3.35: User's effort in IMSPs, varying the number of meetings ($n$=5, $k$=3, $i$=30%, $l$=10, $min$=1, $max$=2).

To better understand which are the best algorithms when we have to deal with meeting scheduling problems, we also measure the amount of tuples the user has to consider to answer the queries of the system.

Figure 3.32 shows the user's effort when the incompleteness is varying. The algorithms with parameter $who = lu$ or $su$ need an effort that increases less rapidly than the effort done by the users with algorithms with $who = dpi$ or $dp$. Among our candidates for the best algorithm, LU.WORST.BRANCH is the best and DPI.WORST.BRANCH has to take into consideration as well.

When we vary the number of agents (see Figure 3.33), we can notice a clear distinction among the algorithms with $when = node$ or $who = su$ which need a user's effort greater than 20% and the others. Moreover, the algorithms with *when* different from *node* or *who* different from *su*, need an effort that decreases as the agents increase. Also in this settings, LU.WORST.BRANCH and DPI.WORST.BRANCH are among the best algorithms. The algorithms with *when* different from *node* or *who* different from *su* need less effort than the others also when we vary the meetings per agent (Figure 3.34) and such an effort decreases as the meeting per agents increases.

Finally, we measure the user's effort varying the number of meetings (see Figure 3.35). Considering the best algorithms, the effort does not increase as the meetings rises and LU.WORST.BRANCH and DPI.WORST.BRANCH are still among the best ones. We have identified two algorithms that show

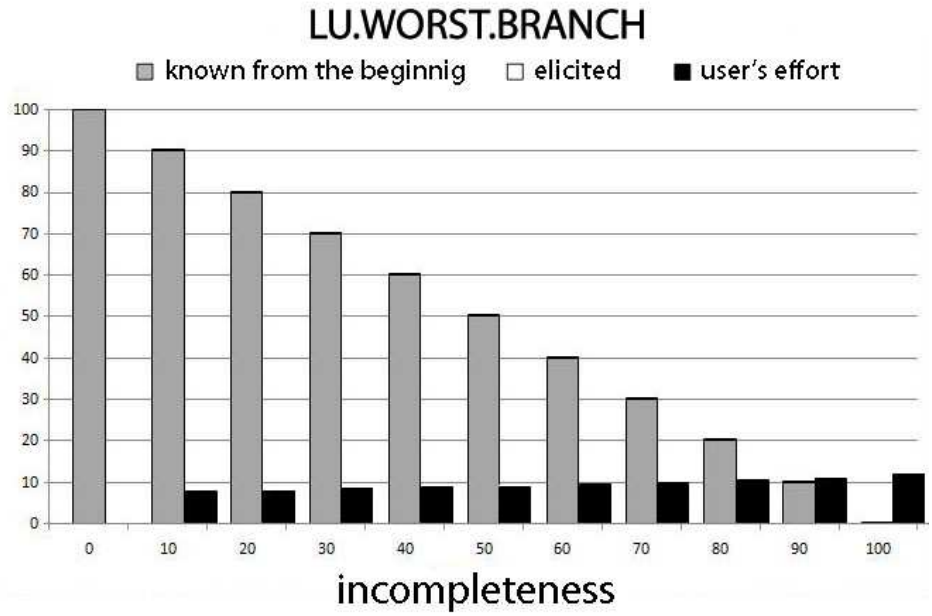



Figure 3.36: LU.WORST.BRANCH varying incompleteness ($m$=12, $n$=5, $k$=3, $l$=10, $min$=1, $max$=2) in IMSPs.

good results in both elicitation and effort asked to the user. Figure 3.36 shows the results of LU.WORST.BRANCH. It elicits a very small amount of preferences also in totally incomplete settings. The elicitation is always less than 1% and the user has to consider only around 10% of the incomplete tuples to give the preferences asked by the system. This behaviour is due to the fact that the search is guided by the user that chooses the value to instantiate. If we are working in settings where only the system can instantiate the domain values, the best algorithm is DPI.WORST.BRANCH (see Figure 3.37). DPI.WORST.BRANCH elicits more preferences than LU.WORST.BRANCH but the percentage remains always under 5% so it gives very good results as well. Also the effort required to the user is very small: it is at most 23% even if the incompleteness is 100%.

Summarizing, the best compromise between elicitation and user's effort is the LU.WORST.BRANCH algorithm. In situation where the system has to carry our the value instantiation, DPI.WORST.BRANCH has to be pre-

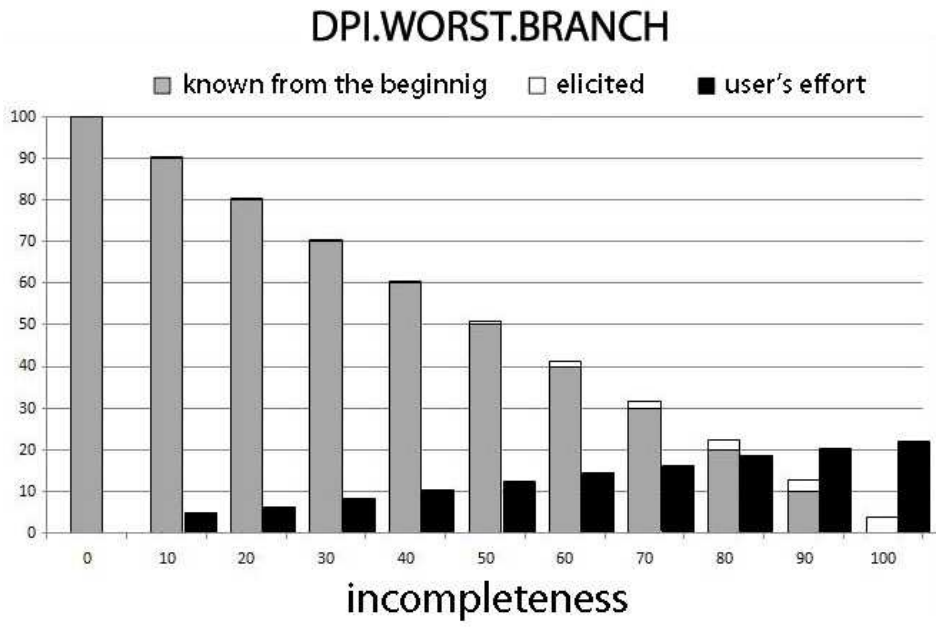



Figure 3.37: DPI.WORST.BRANCH varying incompleteness ($m$=12, $n$=5, $k$=3, $l$=10, $min$=1, $max$=2) in IMSPs.

ferred.

### 3.6.2 Results for the local search approach

#### Incomplete Fuzzy CSPs

We tested the performance of our algorithm using both the ALL and the WORST elicitation strategy. We also compared the result with one of the best algorithms in Section 3.6.1, called here FBB (which stands for fuzzy branch and bound). In Section 3.6.1, this algorithm corresponds to the one called DPI.WORST. BRANCH.

We first considered the quality of the returned solution. To do this, we computed the distance between the preference of the returned solution and that of the necessarily optimal solution returned by algorithm FBB. Such a distance is measured as the percentage over the whole range of preference values. For example, if the preference of the solution returned is 0.4 and the one of the solution given by FBB is 0.5, the preference error reported is 10%. A higher error denotes a lower solution quality.

Figures 3.38, 3.39, 3.40 and 3.41 show the preference error when density,

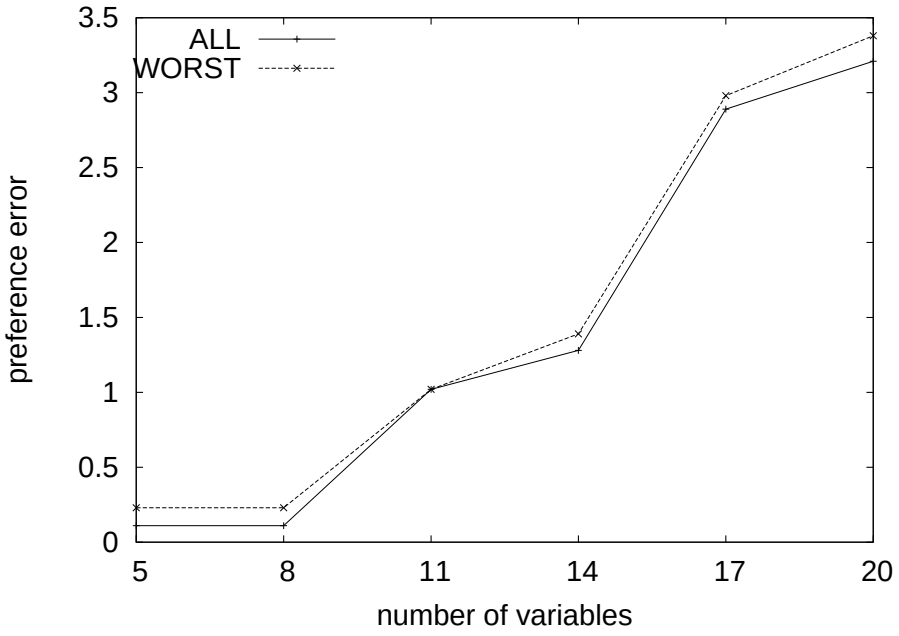



Figure 3.38: Solution quality varying the number of variables ($m$=5, $d$=50%, $i$=30%, $t$=10%) in incomplete fuzzy CSPs.

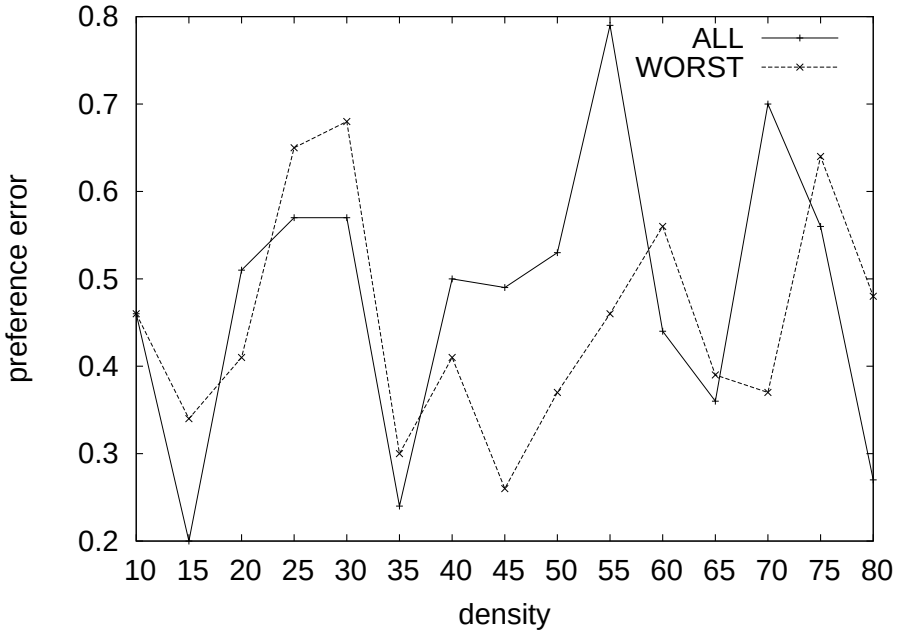



Figure 3.39: Solution quality varying density ($n$=10, $m$=5, $i$=30%, $t$=10%) in incomplete fuzzy CSPs.

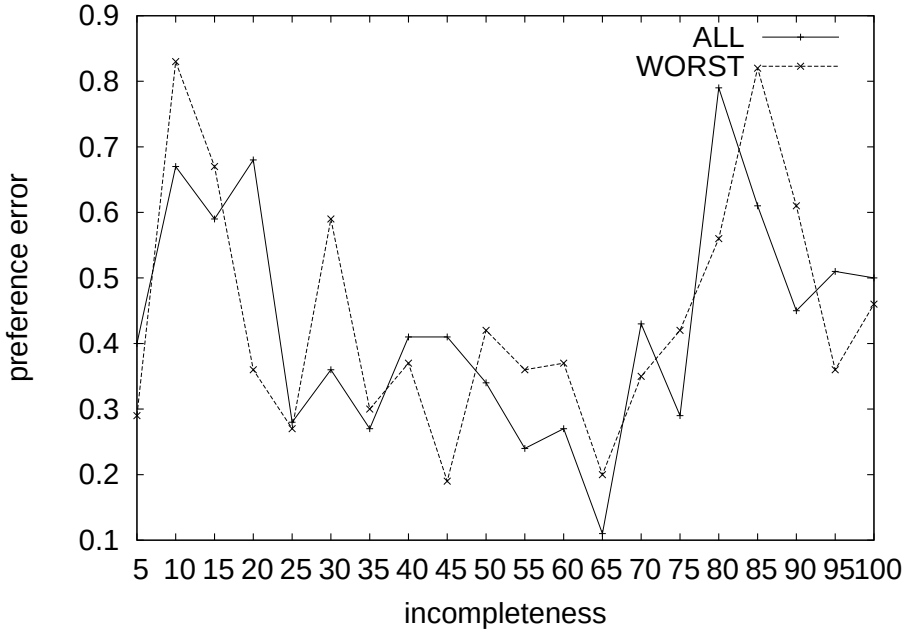



Figure 3.40: Solution quality varying incompleteness ($n$=10, $m$=5, $d$=50%, $t$=10%) in incomplete fuzzy CSPs.

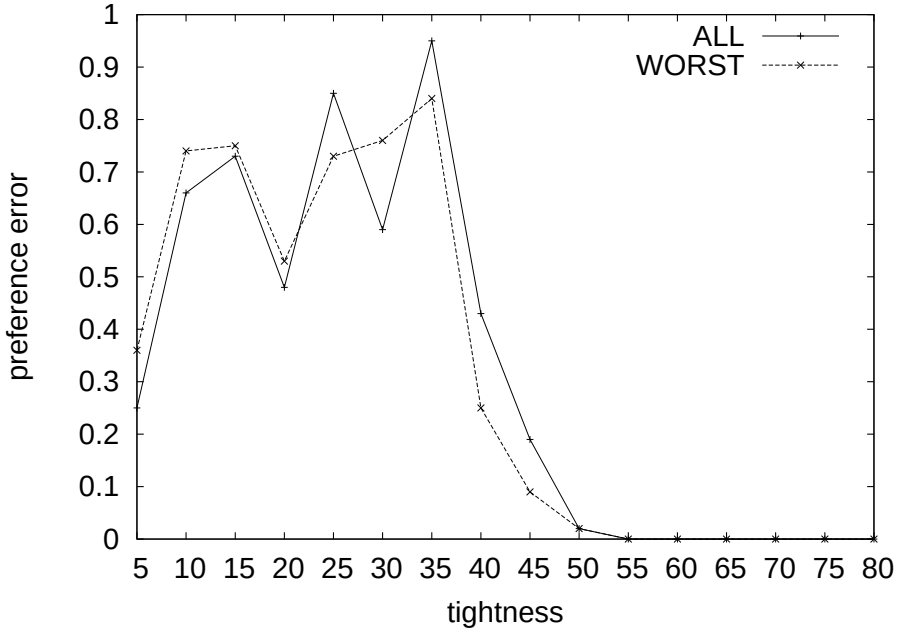



Figure 3.41: Solution quality varying tightness ($n$=10, $m$=5, $d$=50%, $i$=30%) in incomplete fuzzy CSPs.

incompleteness, tightness, and the number of variables vary (please notice that the y-axis ranges from 0% to 10%). We can see that the error is always very small and its maximum value is 3.5% when we consider problems with 20 variables. In most of the other cases, it is below 1.5%. We also can notice that the solution quality is practically the same for both elicitation strategies.

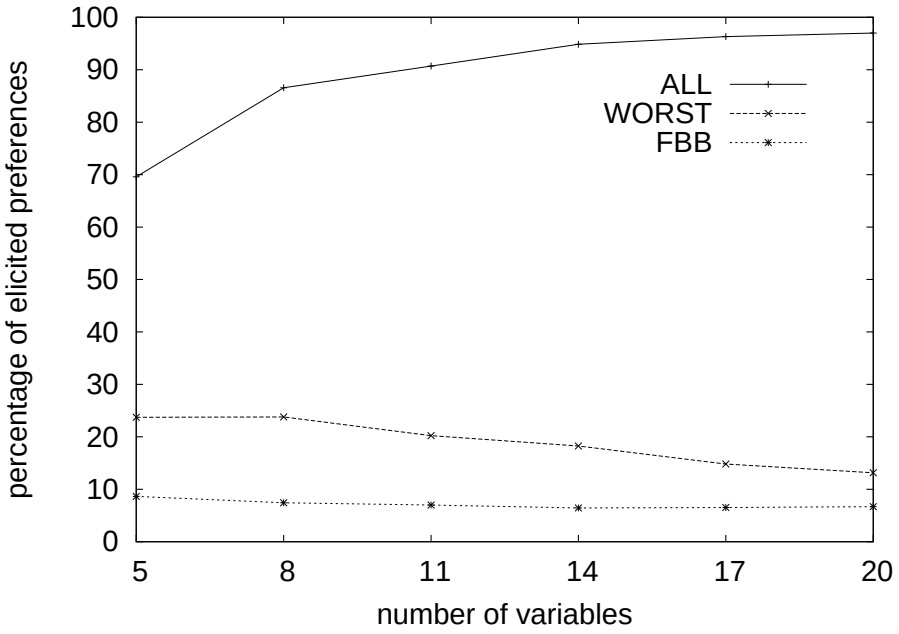



Figure 3.42: Percentage of elicited tuples varying the number of variables ($m$=5, $d$=50%, $i$=30%, $t$=10%) in incomplete fuzzy CSPs.

If we look at the percentage of elicited tuples (Figures 3.42, 3.43, 3.44, 3.45), we can see that the WORST strategy elicits always less tuples than ALL, eliciting only 20% of incomplete tuples in most of the cases. When tightness is above 40%, WORST elicits very few tuples since the algorithm discovers soon that there are no solutions. The FBB algorithm elicits about half as many preferences as WORST. Thus, with 10 variables, FBB is better than our local search approach, since it guarantees to find a necessarily optimal solution while eliciting a smaller number of preferences.

We also tested the WORST strategies varying the number of variables from 10 to 100. In Figure 3.46(a) we show how the elicitation varies up to 100 variables. It is easy to notice that with more than 70 variables the percentage of elicited tuples decreases. This is because the probability of a complete assignment with a 0 preference rises (since density remains the same). Moreover, we can see how the local search algorithms scale better

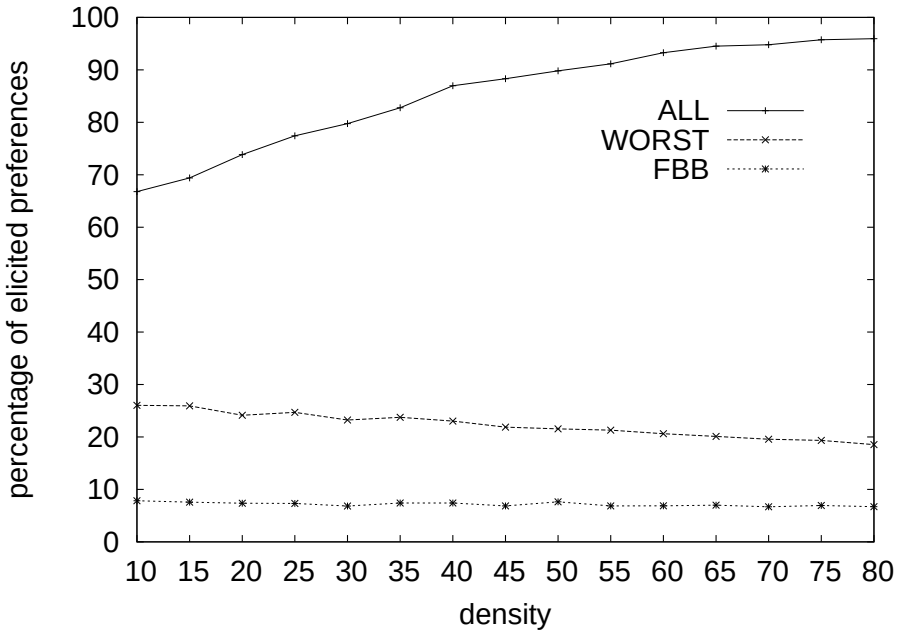



Figure 3.43: Percentage of elicited tuples varying density ($n$=10, $m$=5, $i$=30%, $t$=10%) in incomplete fuzzy CSPs.

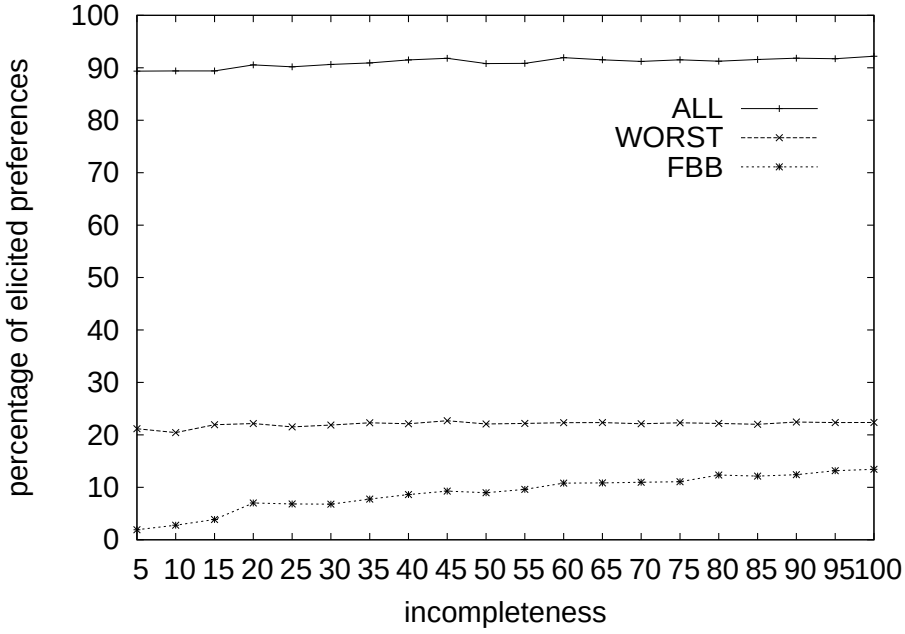



Figure 3.44: Percentage of elicited tuples varying incompleteness ($n$=10, $m$=5, $d$=50%, $t$=10%) in incomplete fuzzy CSPs.

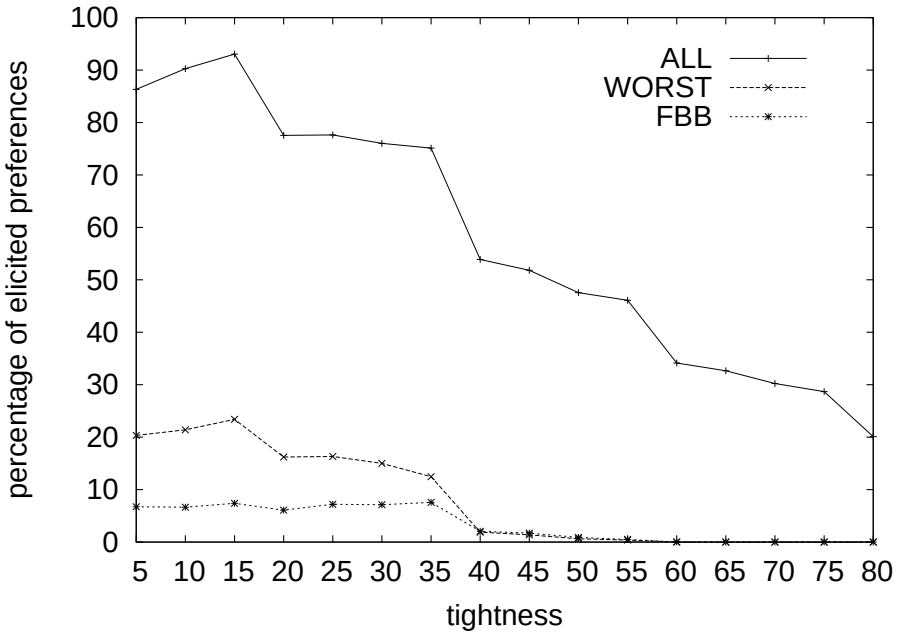



Figure 3.45: Percentage of elicited tuples varying tightness ($n$=10, $m$=5, $d$=50%, $i$=30%) in incomplete fuzzy CSPs.

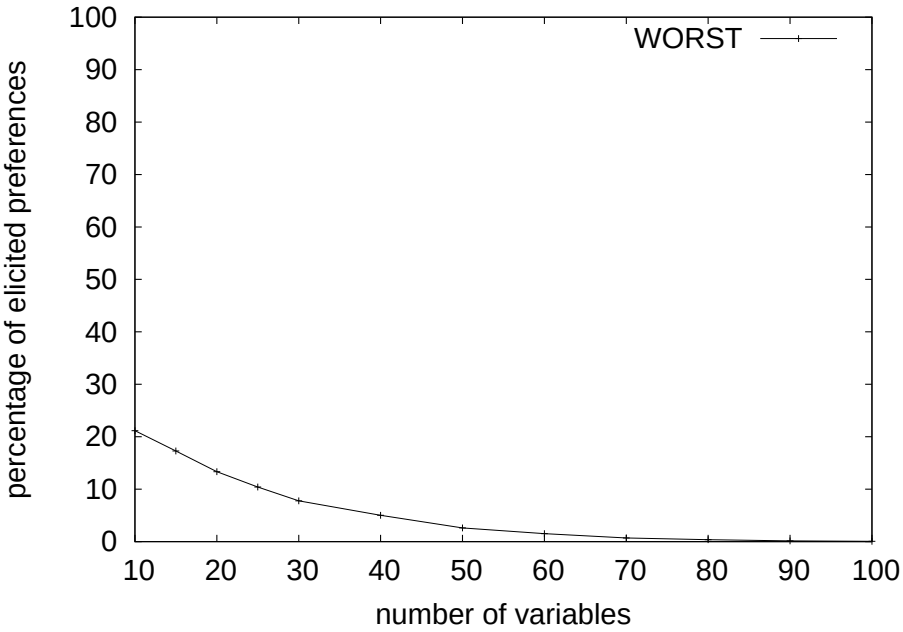
than the branch and bound approach. In Figure 3.46(b) the FBB reaches a time limit of 10 minutes with just 25 variables, while the WORST algorithm needs the same time to solve instances of size 100.

We also investigated the runtime behavior of our local search algorithm. Surprisingly, the algorithm elicits almost all the tuples it needs within the first 10000 steps. Moreover the distance from the necessarily optimal preference decreases significantly during the first 20000 steps and then it decreases slightly until the end of the search. This behavior is the same no matter which parameter is varying. Hence we can stop our algorithm after 20-30000 steps whilst still ensuring a good solution quality.

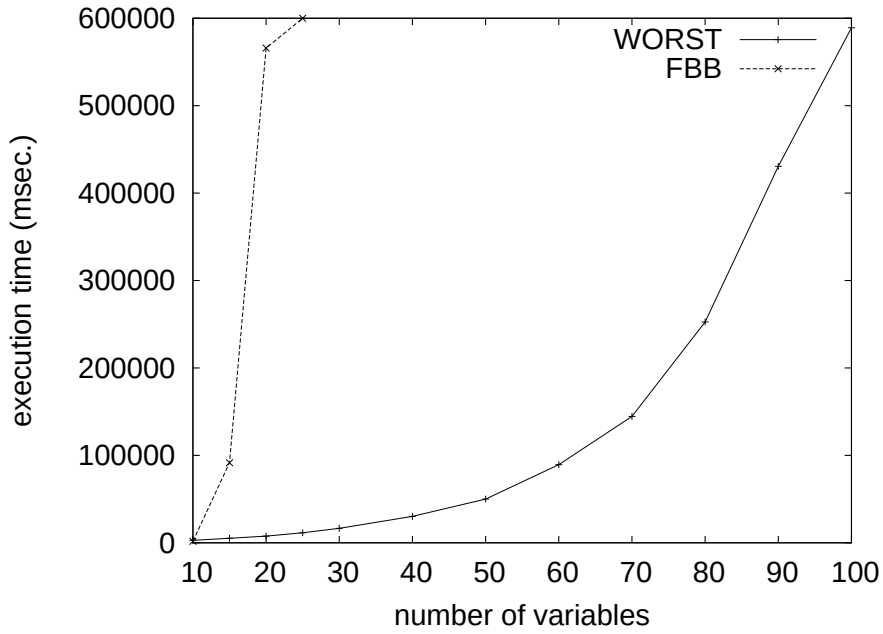
### Incomplete Weighted CSPs

We tested the performance of our algorithm using the ALL, BB, WW and BW elicitation strategies. We also compared the result with one of the best algorithms in Section 3.6.1 for solving incomplete weighted CSPs, which we call WBB here (in Section 3.6.1 it was called DPI.BW.TREE).

Figures 3.51, 3.52, 3.53, 3.54 shows the solution quality, in terms of the preference error, which is measured similarly to the fuzzy context but with a specific treatment to deal with $+\infty$. More precisely, the error is the per-

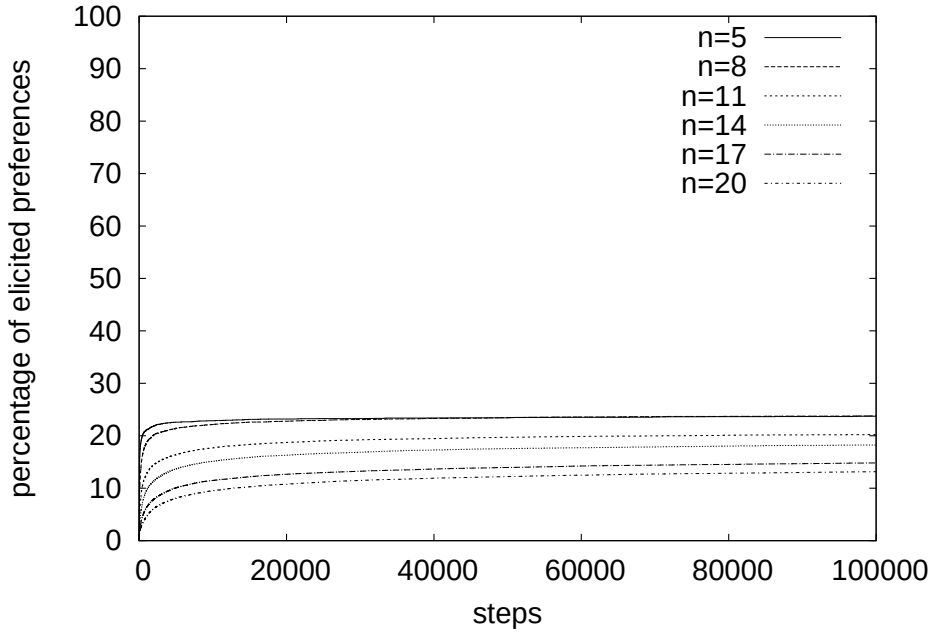



(a) percentage of elicited tuples

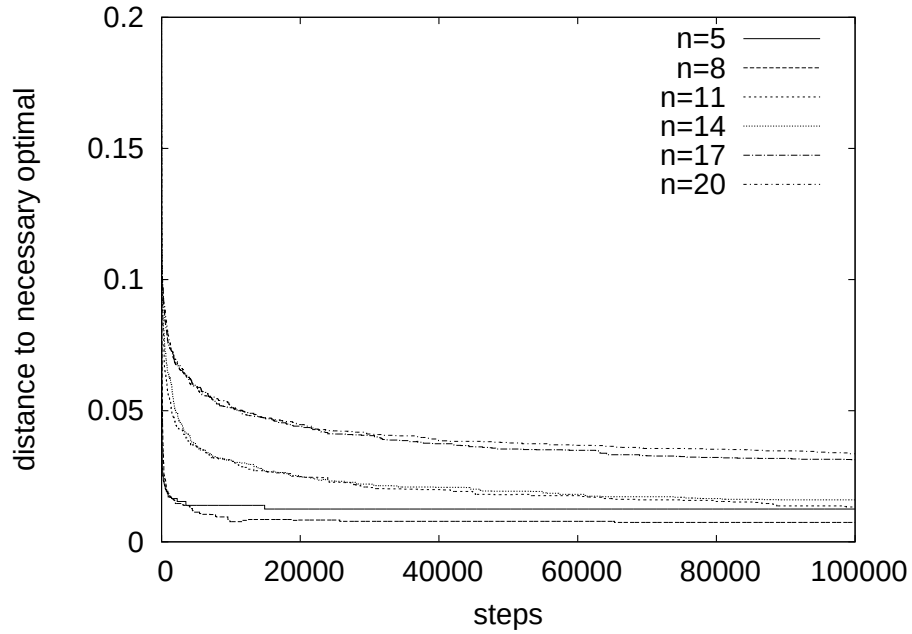



(b) execution time

Figure 3.46: WORST strategy on incomplete fuzzy CSPs. Values of the fixed parameters: $m$=10, $d$=35%, $i$=30%, $t$=5%.

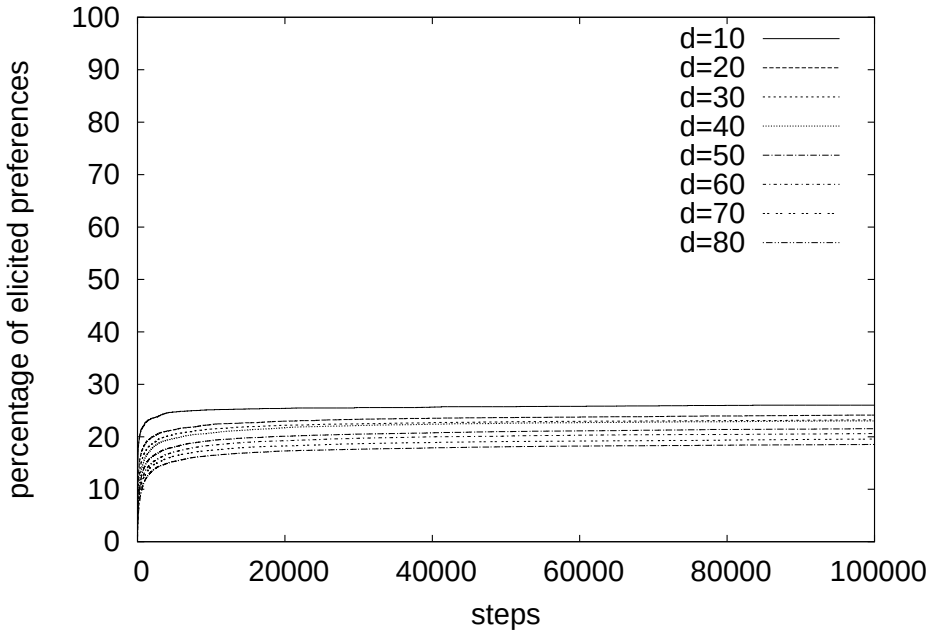



(a)

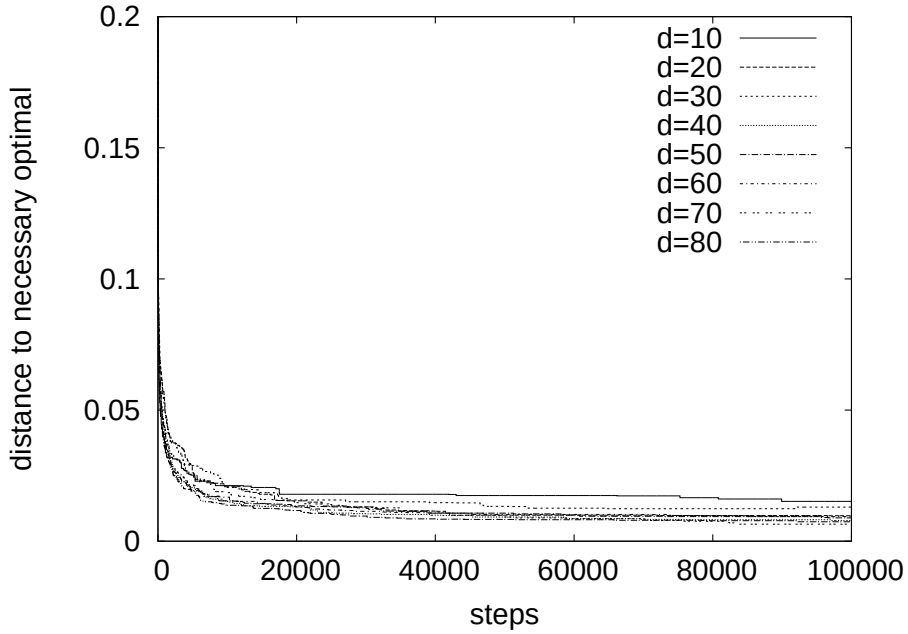



(b)

Figure 3.47: Runtime behavior of WORST varying the number of variables ($m$=5, $d$=50%, $i$=30%, $t$=10%) in incomplete fuzzy CSPs.

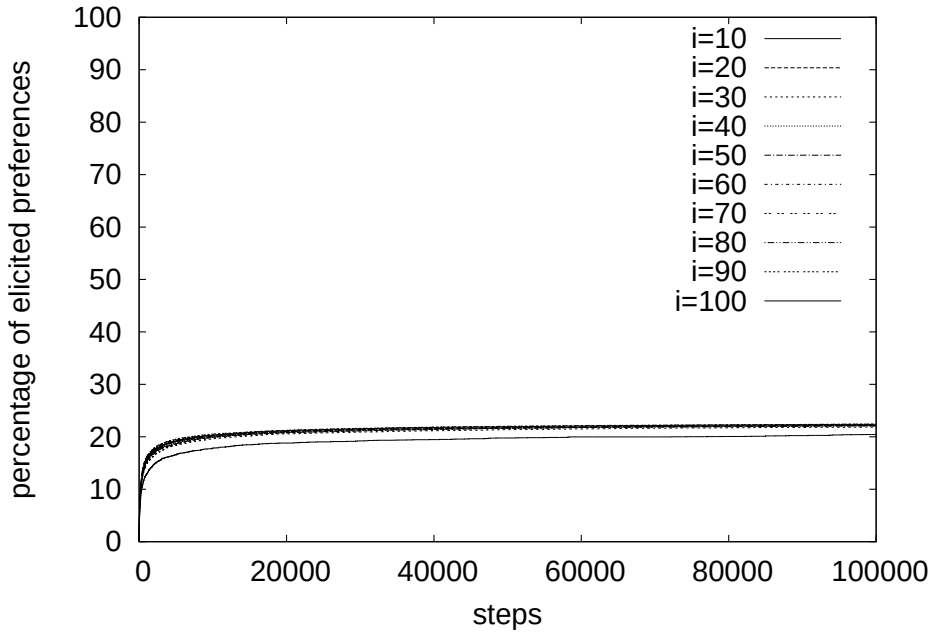



(a)

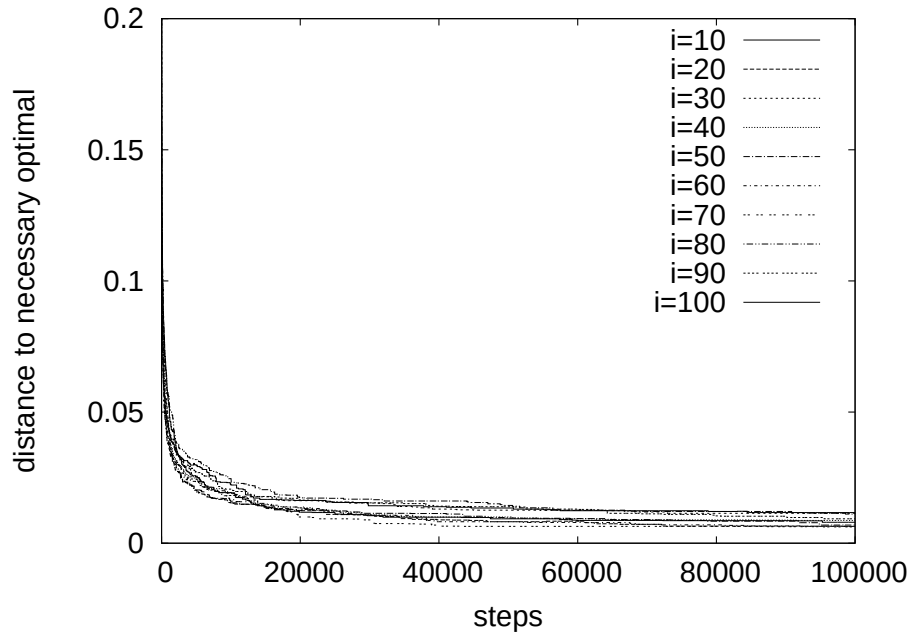



(b)

Figure 3.48: Runtime behavior of WORST varying density ($n$=10, $m$=5, $i$=30%, $t$=10%) in incomplete fuzzy CSPs.

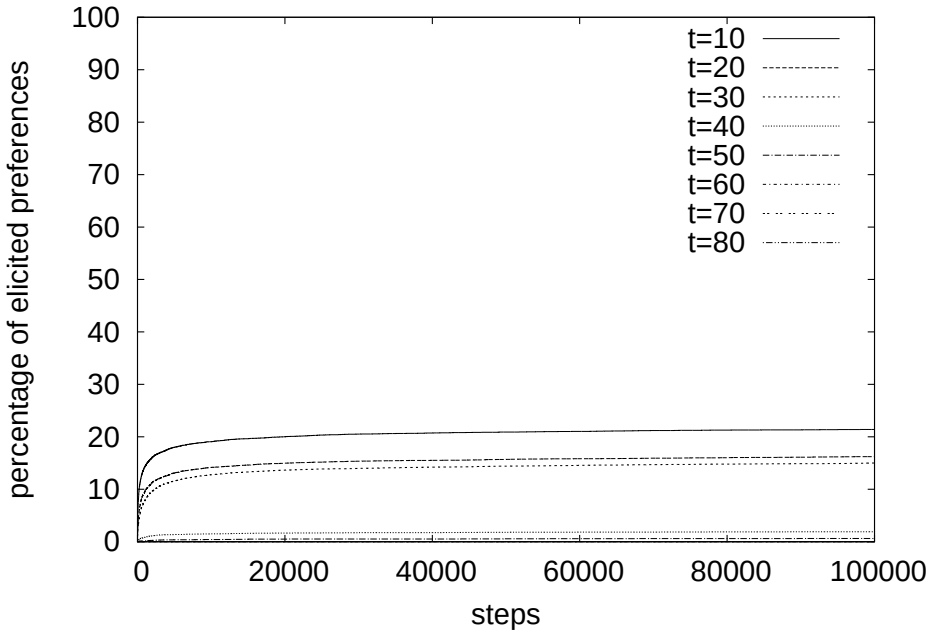



(a)

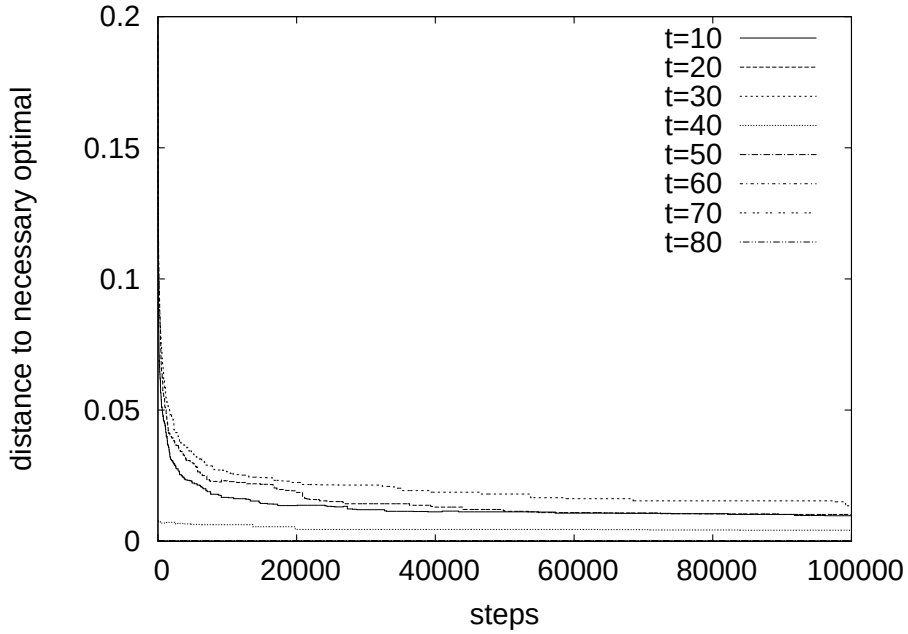



(b)

Figure 3.49: Runtime behavior of WORST varying incompleteness ($n$=10, $m$=5, $d$=50%, $t$=10%) in incomplete fuzzy CSPs.




(a)




(b)

Figure 3.50: Runtime behavior of WORST varying tightness ($n$=10, $m$=5, $d$=50%, $i$=30%) in incomplete fuzzy CSPs.

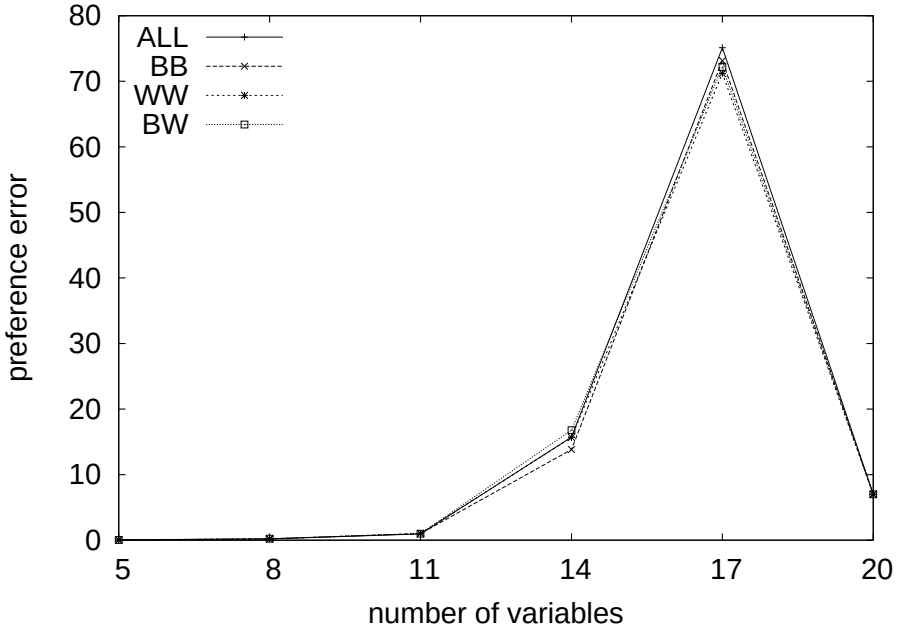



Figure 3.51: Solution quality varying the number of variables ($m$=5, $d$=50%, $i$=30%, $t$=10%) in incomplete weighted CSPs.

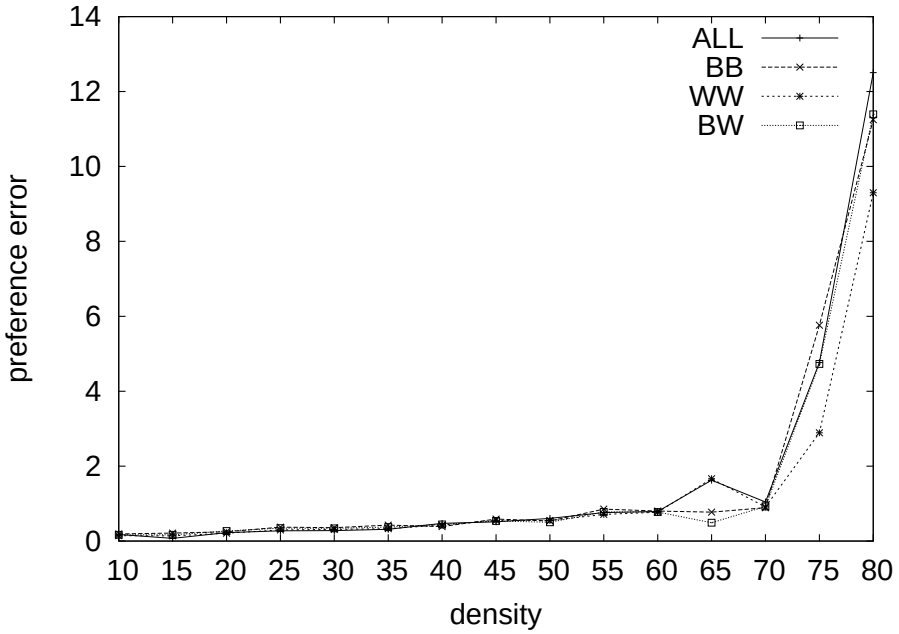



Figure 3.52: Solution quality varying density ($n$=10, $m$=5, $i$=30%, $t$=10%) in incomplete weighted CSPs.

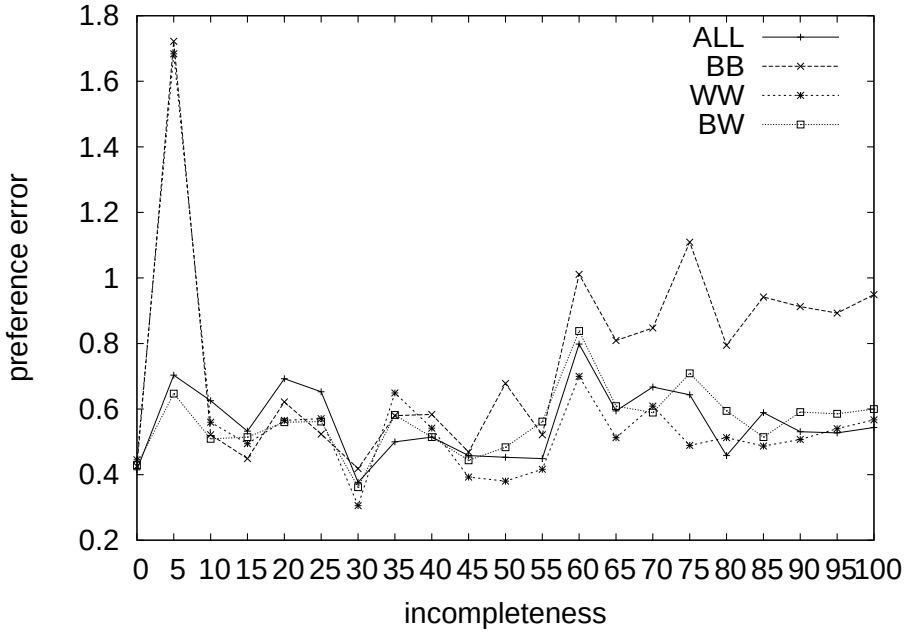



Figure 3.53: Solution quality varying incompleteness ($n$=10, $m$=5, $d$=50%, $t$=10%) in incomplete weighted CSPs.

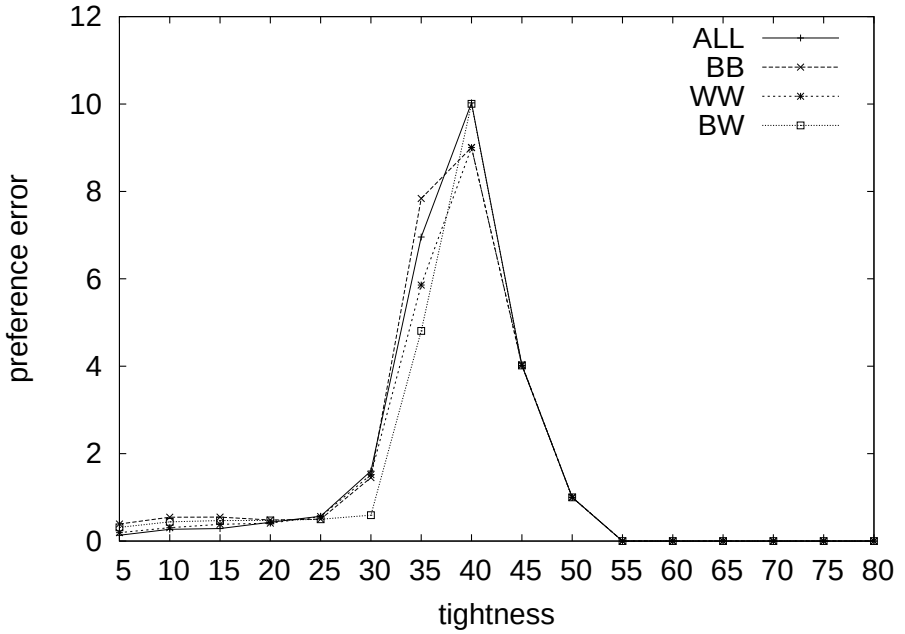



Figure 3.54: Solution quality varying tightness ($n$=10, $m$=5, $d$=50%, $i$=30%) in incomplete weighted CSPs.

centage difference between the solution preference found by the local search algorithm and the one found by WBB over the preference range. Notice that, with costs in $[0, 10] \cup +\infty$, solutions have preferences that may be between 0 and $(((n * (n-1))/2) + n) * 10$, or $+\infty$. When the returned preference is $+\infty$ and the correct preference is different, we report an error of 100%. For example, with 10 variables and preferences in $[1, 10] \cup +\infty$, if the local search algorithm returns a solution with preference 120 and the necessarily optimal preference is 100, then the error is (120-100)*100/550 = 3.64%.

In most cases, all the local search algorithms find a solution with a preference very close to the necessarily optimal one. The peaks at 17 variables in Figure 3.51, at $d = 80\%$ in Figure 3.52, and at $t = 40\%$ in Figure 3.54 show a phase transition where the number of solutions with infinite cost increase significantly. This affects the quality of the solutions found by local search, while the branch and bound approach is able to find an optimal solution with a finite cost.

As in the fuzzy case, we measured the percentage of elicited tuples. As before, the local search approach elicits more preferences than our branch and bound based algorithms. However, the difference is fairly small and independent of the amount of incompleteness (see Figure 3.57). Moreover, it is small also with density is below 60%, or when tightness is less than 30% or greater than 65% (Figure 3.56). Among the local search algorithms, as expected, the algorithms BB, WW, and BW elicit less preferences than ALL.

To study the runtime behavior of our local search approach, we focused on algorithm BB, which is one of the best algorithms. We empirically study how the percentage of elicited tuples and the distance from the necessarily optimal preference vary as the execution proceeds. From Figures 3.59(a), 3.60(a), 3.61(a) and 3.62(a) we can see that the elicitation process takes place mainly in the first 10000 steps. This behavior does not depend on the parameter that is varying.

Furthermore, if we consider the distance from the necessarily optimal preference, we can see that the algorithm finds the best solution in the first 20000 steps (see Figures 3.59(b), 3.60(b), 3.61(b) and 3.62(b)). Exceptions occur around the peaks of Figures 3.51, 3.52, 3.53, 3.54. For example, in Figure 3.59(b) with $n = 17$ or $n = 20$ the solution preference is improved during the whole execution and not only in the first steps. Other examples are for $d = 80\%$ in Figure 3.60(b) and for $t = 40\%$ in Figure 3.62(b).

As in the fuzzy case, we tested our local search algorithms (using the BW strategy in this case) on instances up to 100 variables. From Figure 3.63(a) we can see that the algorithm elicits around 30% of incomplete preferences from 10 to 90 variables. From 90-100 variables, the instances start to have solutions equal to $+\infty$ and the algorithm elicits more preferences. In Figure

3.63(b) we measured the execution time of BW compared with WBB. The branch and bound algorithm reaches the time limit of 10 minutes per instance with just 15 variables so we stopped the execution at 30 variables. On the other hand, the local search algorithm, can solve instances up to 90 variables taking less time.

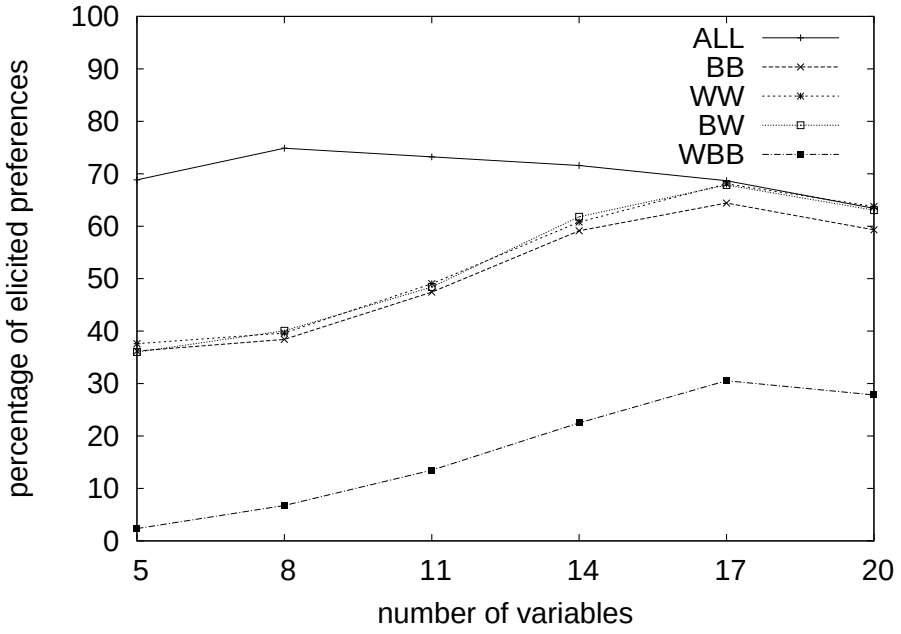



Figure 3.55: Percentage of elicited tuples varying the number of variables ($m$=5, $d$=50%, $i$=30%, $t$=10%) in incomplete weighted CSPs.

Summarizing, we can see that our local search approach finds a solution with a quality which is very close (with an error of at most 2%) to the quality of the necessarily optimal ones. Moreover, such a quality can be obtained also if execution is stopped after only 10000 steps.

### Incomplete meeting scheduling problems

We tested the performance of our local search algorithm, instantiated with both ALL and WORST elicitation strategies, on incomplete meeting scheduling problems as defined in Section 3.6. We compared our local search algorithms with DPI.WORST.BRANCH which has shown very good performance in the same test set. In this section we call FBB the DPI.WORST.BRANCH algorithm.

First we consider the quality of the returned solution compared with the necessarily optimal solutions' preference. We recall that the quality of the

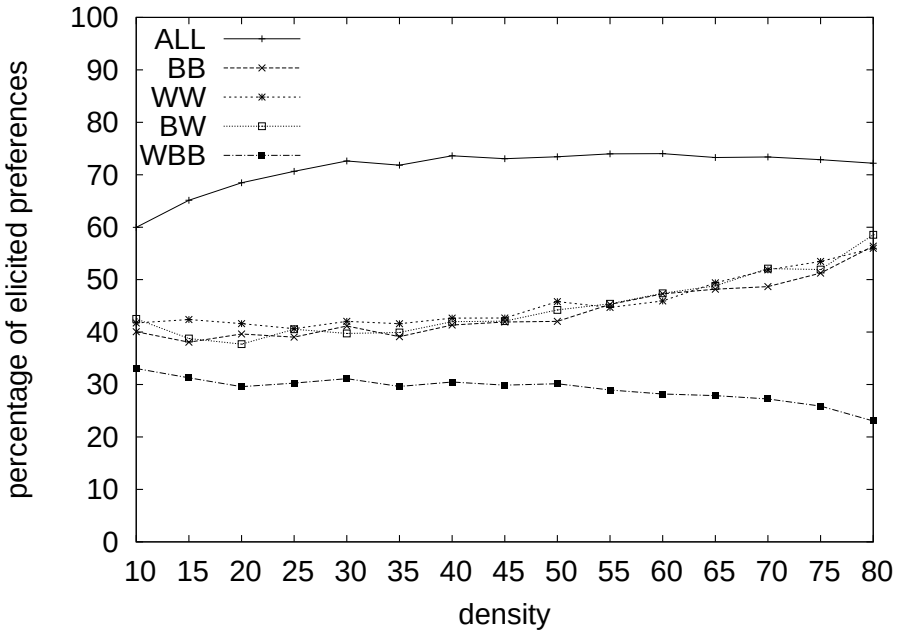



Figure 3.56: Percentage of elicited tuples varying density ($n$=10, $m$=5, $i$=30%, $t$=10%) in incomplete weighted CSPs.

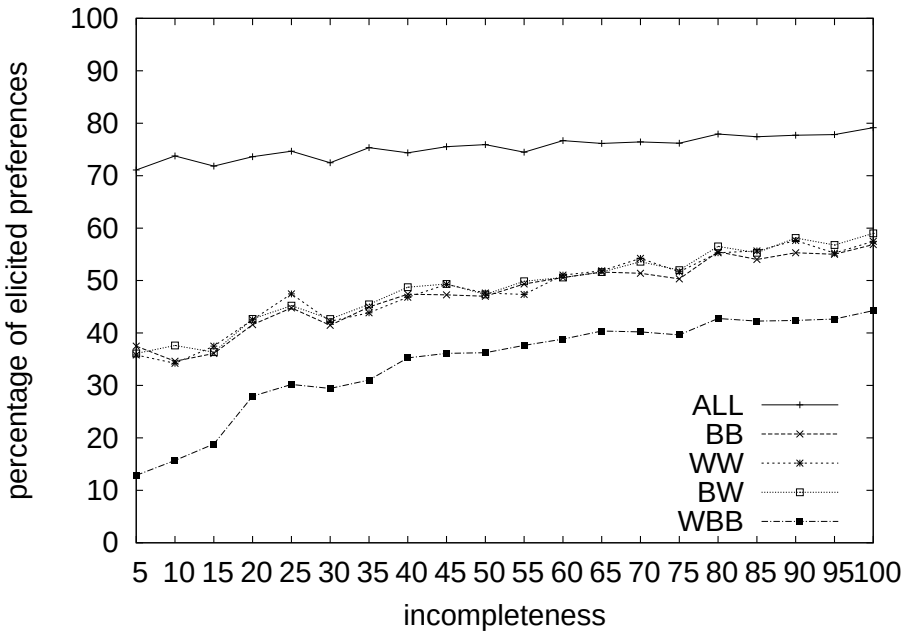



Figure 3.57: Percentage of elicited tuples varying incompleteness ($n$=10, $m$=5, $d$=50%, $t$=10%) in incomplete weighted CSPs.

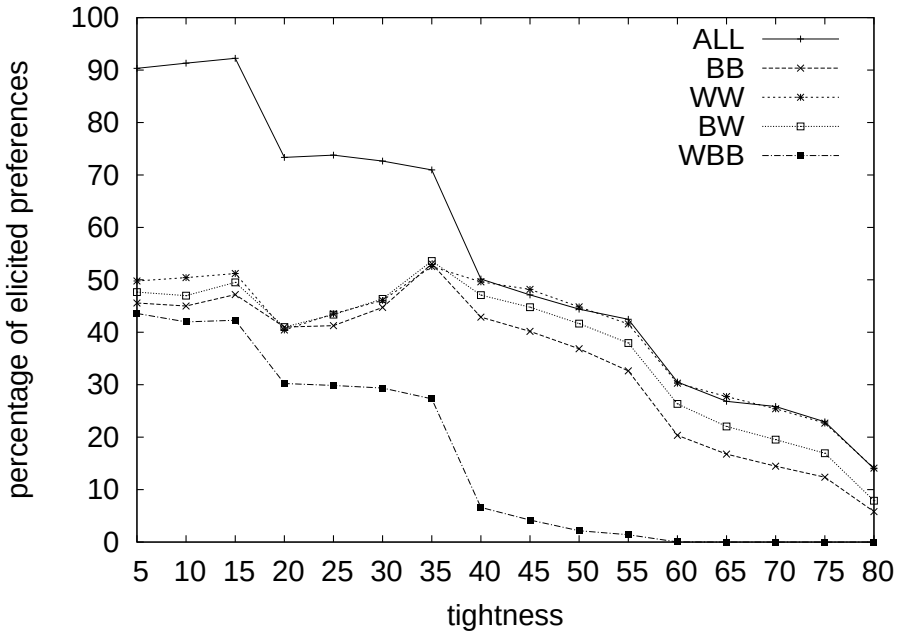


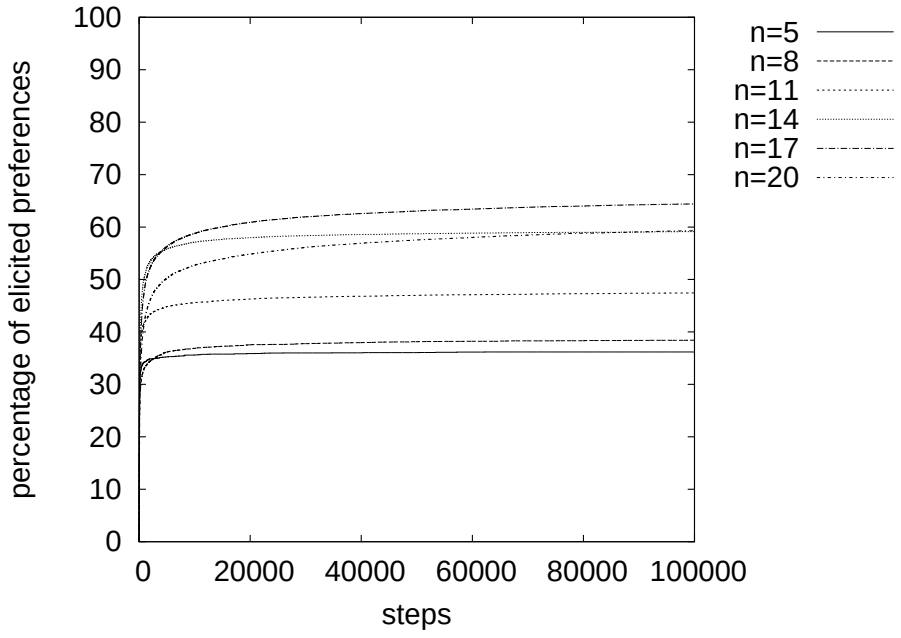

Figure 3.58: Percentage of elicited tuples varying tightness ($n$=10, $m$=5, $d$=50%, $i$=30%) in incomplete weighted CSPs.

solution is shown in terms of the preference error, i.e., the distance, in percentage over the whole range of preference values, between the preference of the solution returned by local search and the necessarily optimal preference.

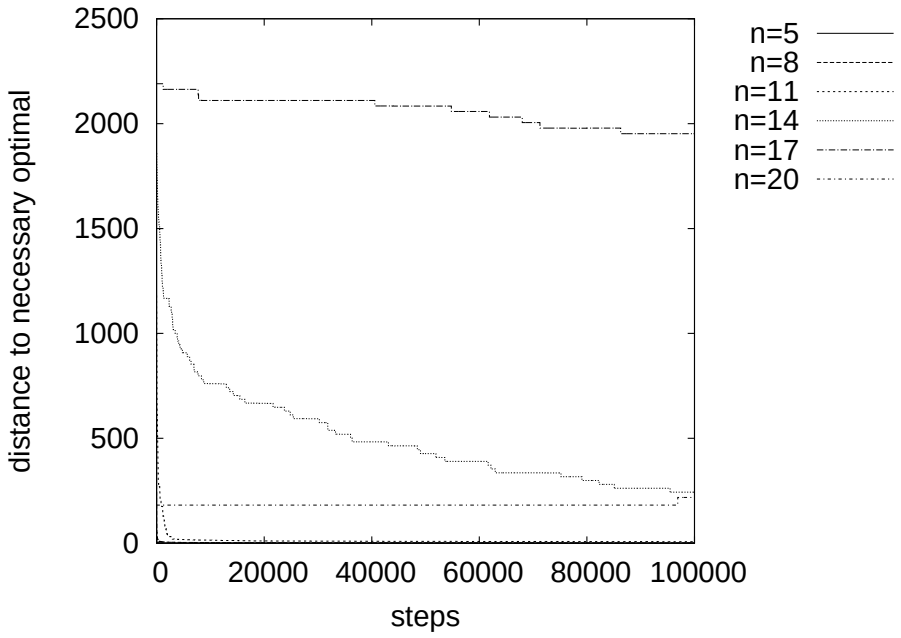
We start showing solution quality when we vary the amount of incompleteness in the problem (see Figure 3.64). Surprisingly, the preference error is very small, and always less than 2.5% both for ALL and WORST elicitation strategies.

Figures 3.65, 3.66 and 3.67 show the preference error varying agents, meetings and meetings per agent, respectively. In all cases the preference error rises as the varying parameter increases. For 5 meetings per agent in Figure 3.67 the problems has almost all solution preferences equal to 0 and so the error is 0 as well. Note the phase transition at 4 meetings per agent. These figures also show that the WORST elicitation strategy returns solutions with comparable or better quality in all settings. Finally, we have to notice that the preference error in the worst case is always less than 1%.

We now consider the percentage of elicited preferences. Figures 3.68, 3.69, 3.70, and 3.71 show the comparison between the WORST strategy and the FBB algorithm varying incompleteness, number of agents, number of meetings, and meetings per agent respectively. We omit the results of ALL

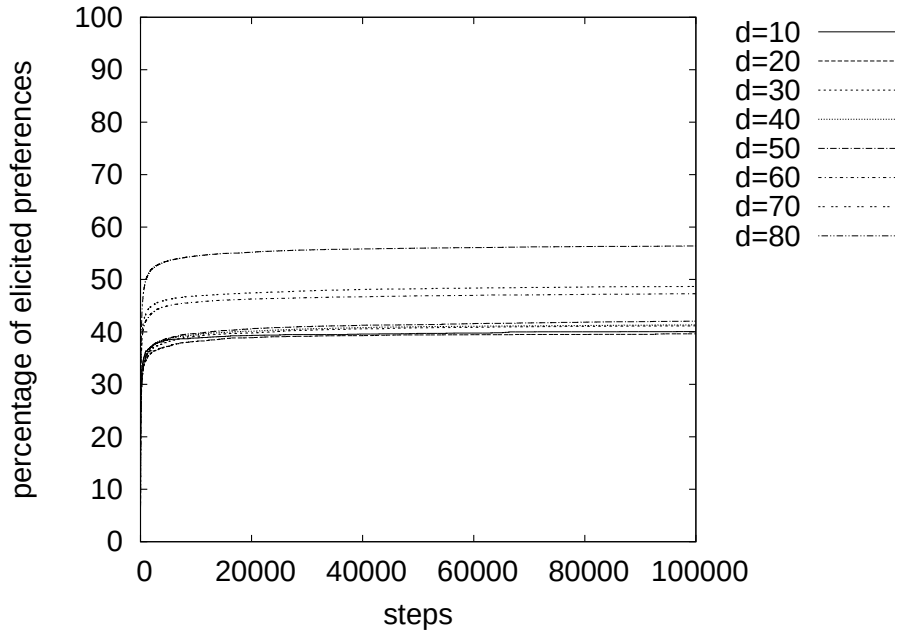



(a)

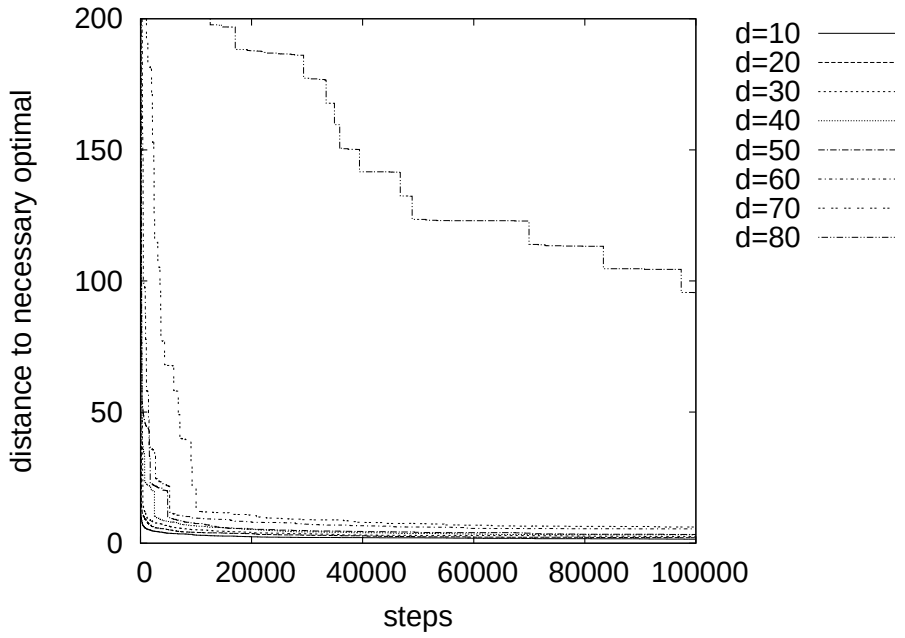



(b)

Figure 3.59: Runtime behavior of BB varying the number of variables ($m$=5, $d$=50%, $i$=30%, $t$=10%) in incomplete weighted CSPs.

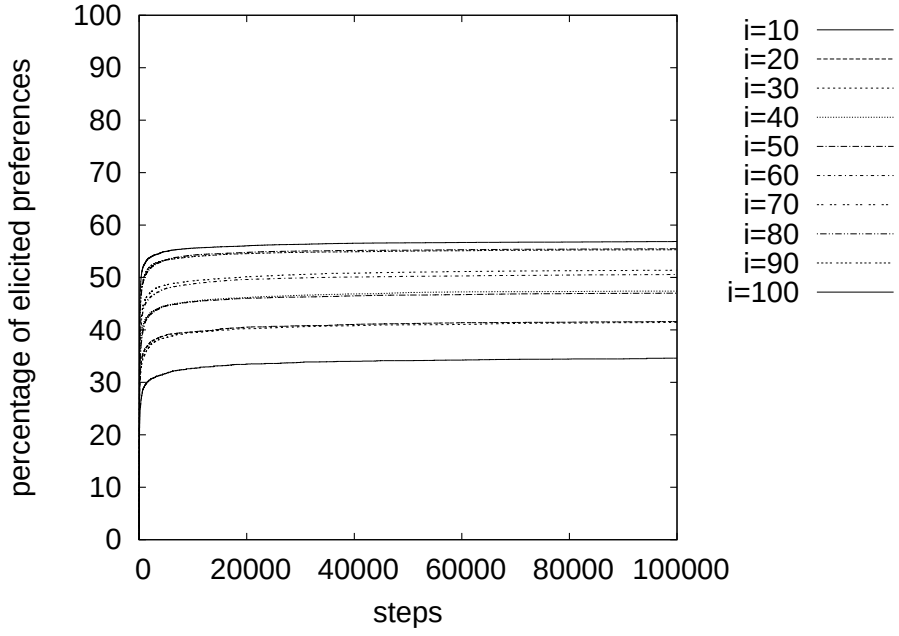



(a)

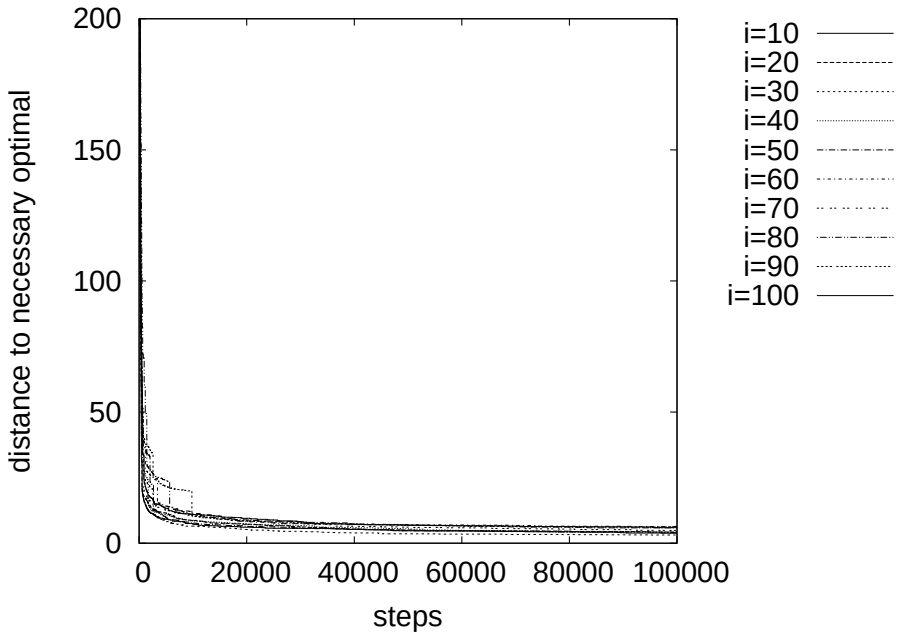



(b)

Figure 3.60: Runtime behavior of BB varying density ($n$=10, $m$=5, $i$=30%, $t$=10%) in incomplete weighted CSPs.

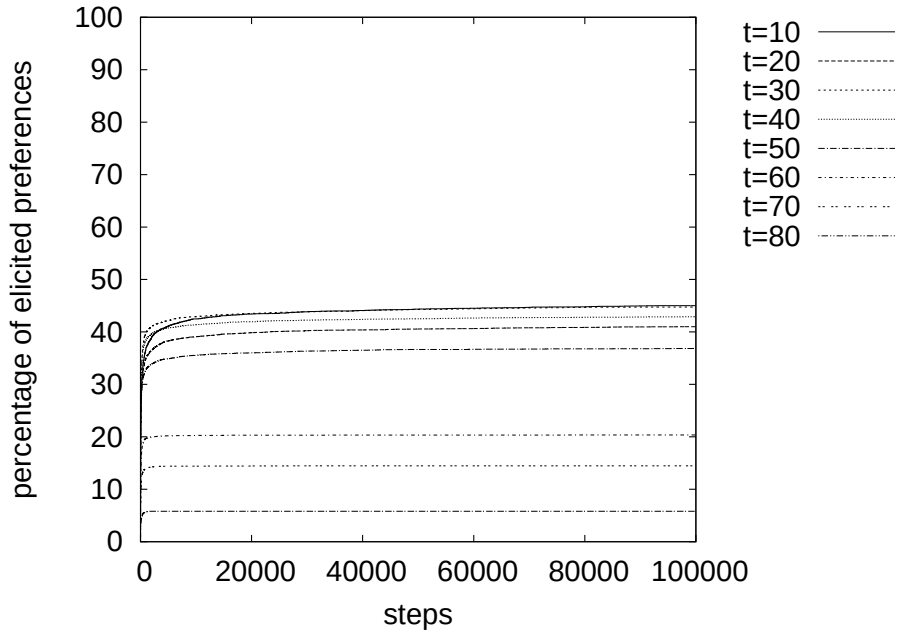



(a)

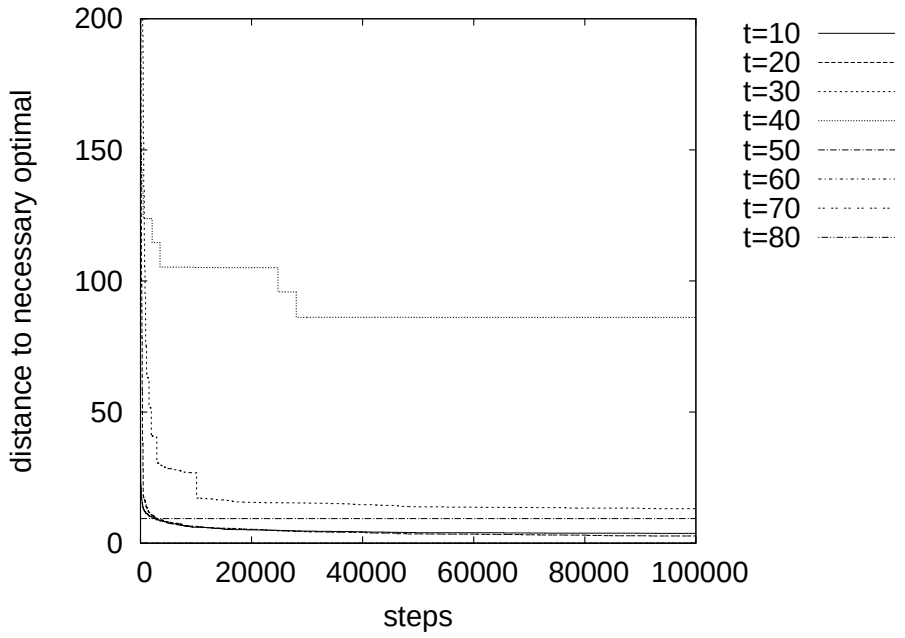



(b)

Figure 3.61: Runtime behavior of BB varying incompleteness ($n$=10, $m$=5, $d$=50%, $i$=30%, $t$=10%) in incomplete weighted CSPs.

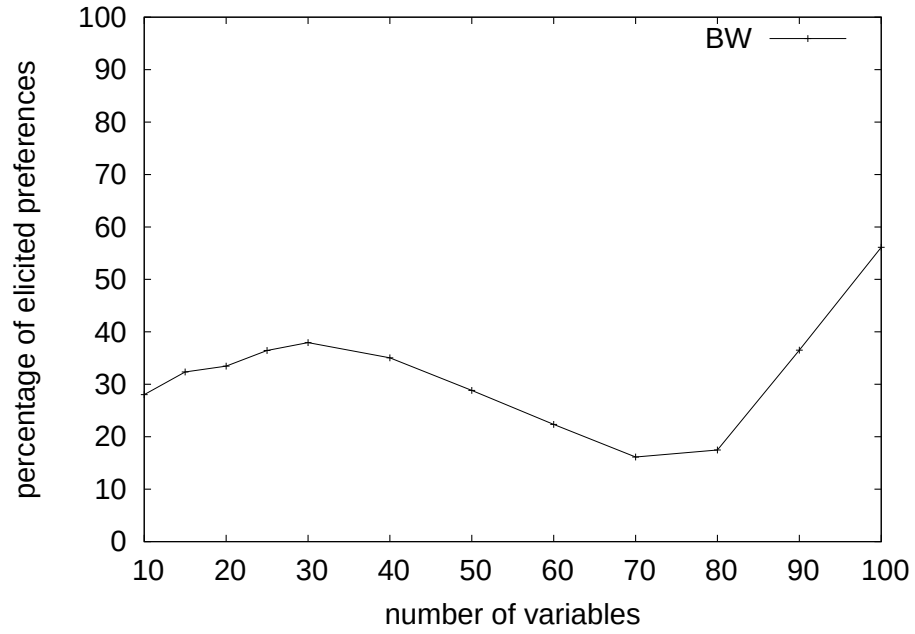



(a)

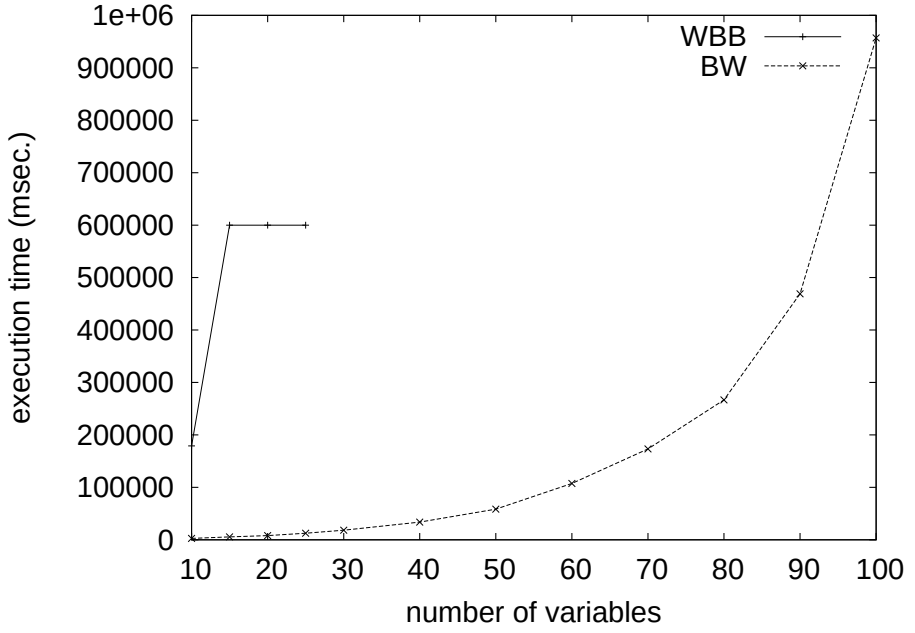



(b)

Figure 3.62: Runtime behavior of BB varying tightness ($n$=10, $m$=5, $d$=50%, $i$=30%, $t$=10%) in incomplete weighted CSPs.




(a) percentage of elicited tuples




(b) execution time

Figure 3.63: BW strategy on incomplete weighted CSPs. Values of the fixed parameters: $m$=10, $d$=35%, $i$=30%, $t$=5%.

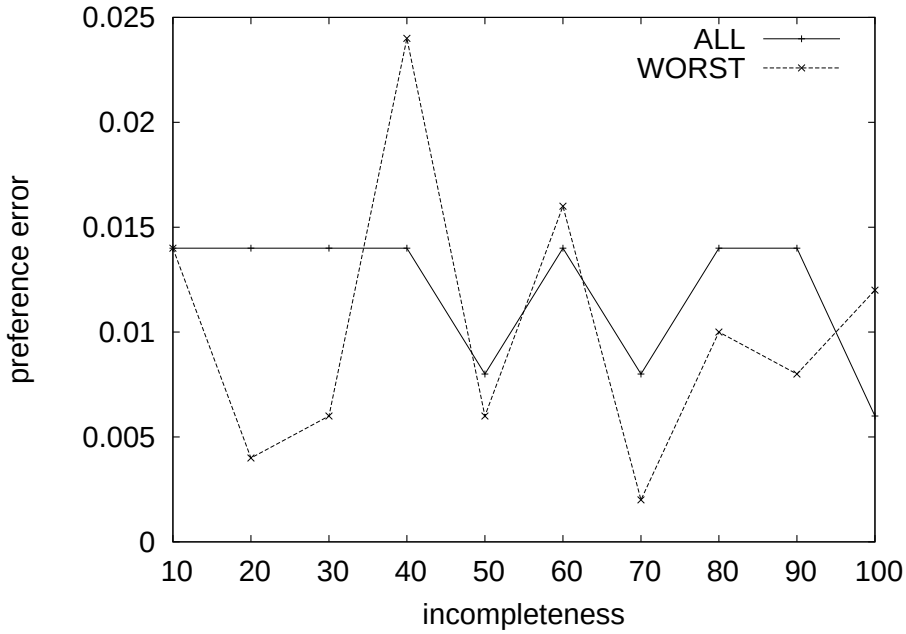



Figure 3.64: Solution quality varying incompleteness ($m$=10, $n$=5, $k$=3, $l$=10, $min$=1, $max$=2) in IMSPs.

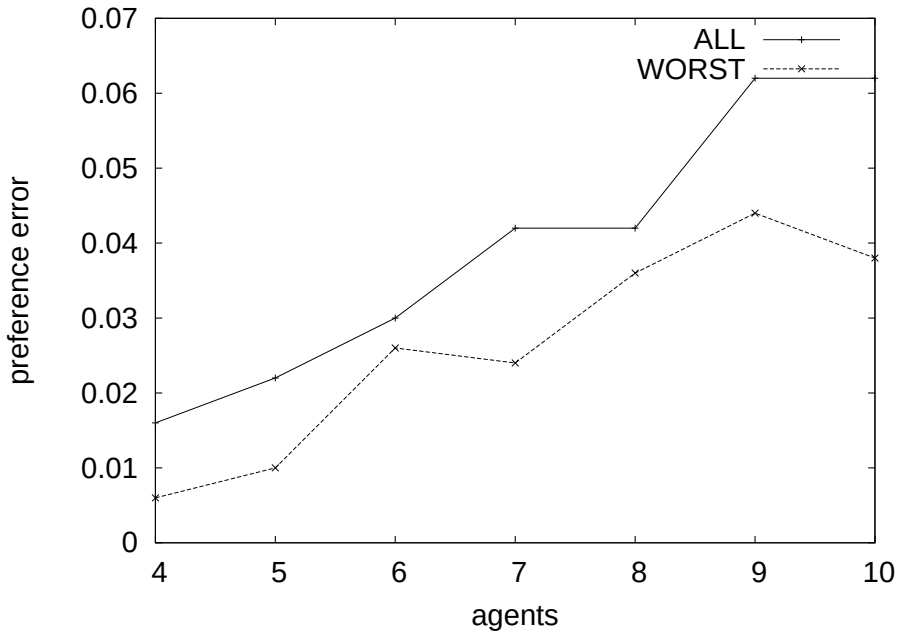



Figure 3.65: Solution quality varying the number of agents ($m$=10, $k$=3, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

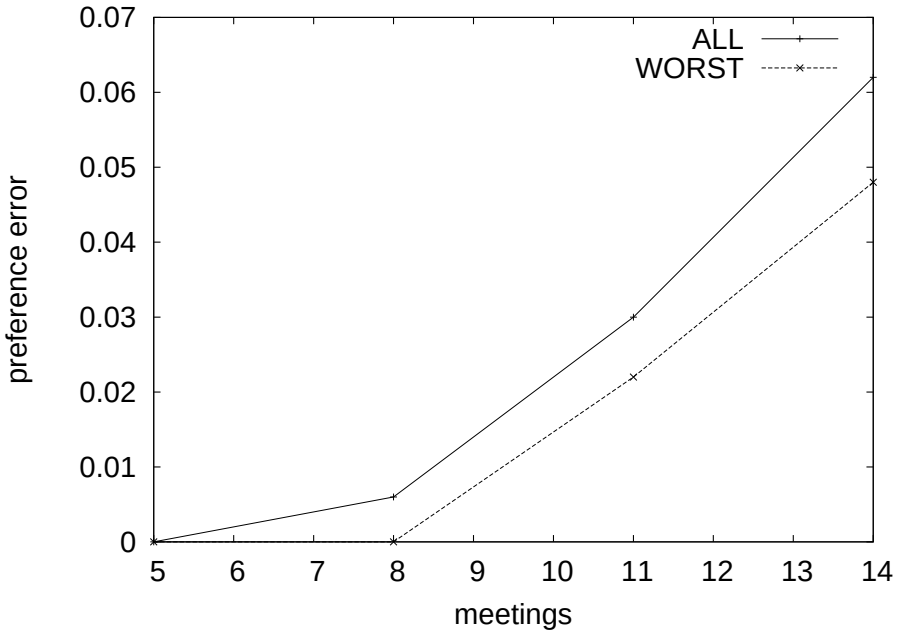



Figure 3.66: Solution quality varying the number of meetings ($n$=5, $k$=3, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

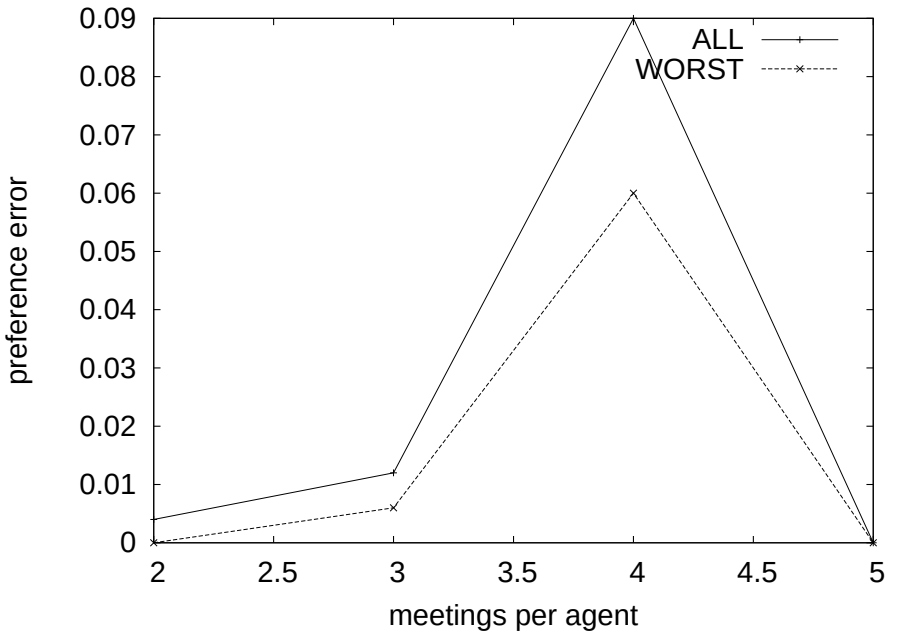



Figure 3.67: Solution quality varying meetings per agent ($m$=10, $n$=5, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

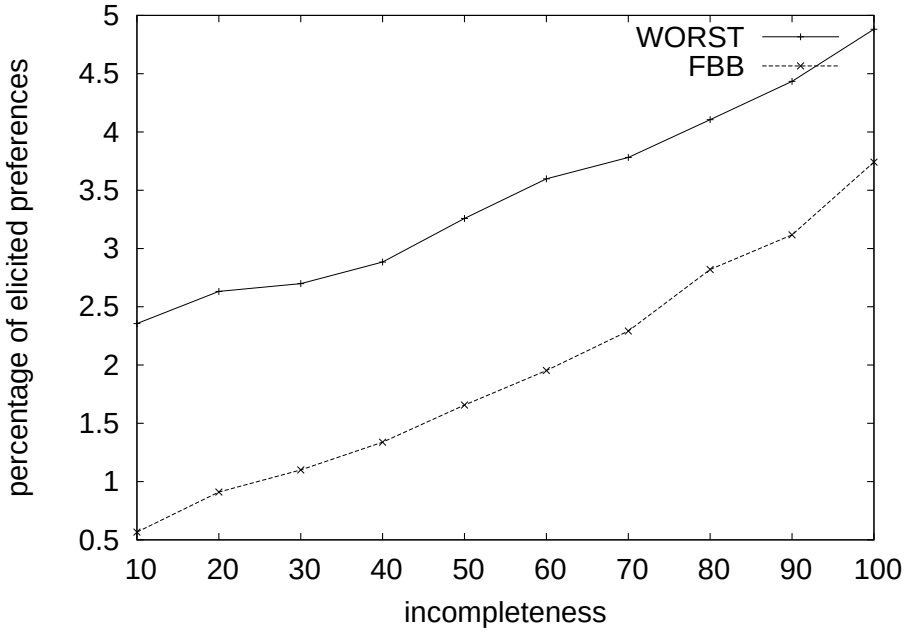



Figure 3.68: Percentage of elicited tuples varying incompleteness ($m$=10, $n$=5, $k$=3, $l$=10, $min$=1, $max$=2) in IMSPs.

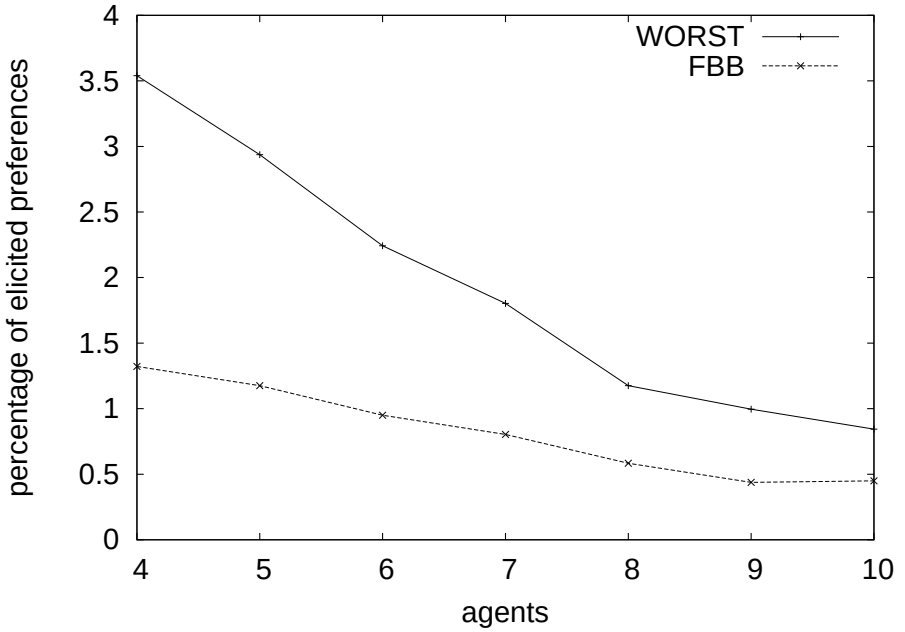



Figure 3.69: Percentage of elicited tuples varying the number of agents ($m$=10, $k$=3, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

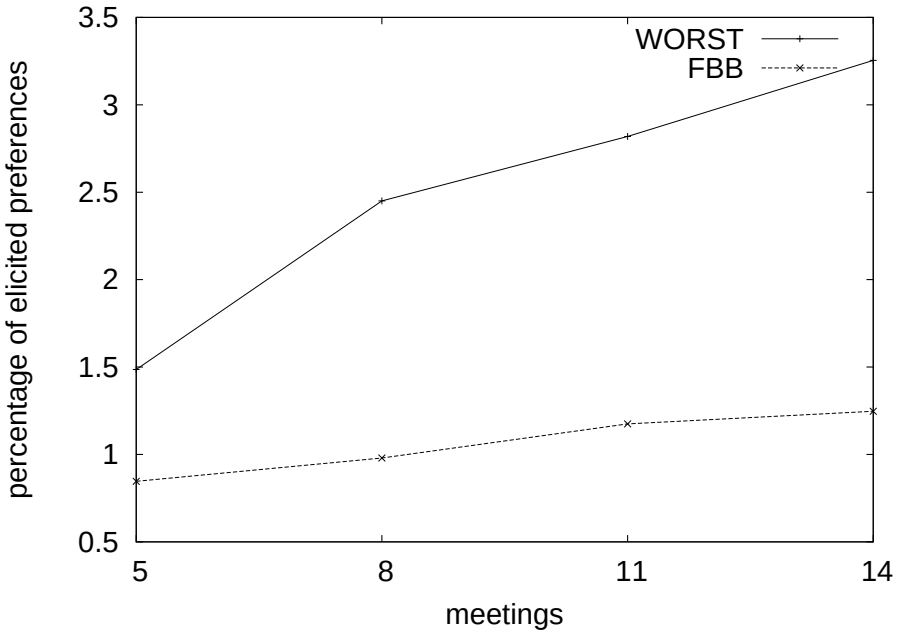



Figure 3.70: Percentage of elicited tuples varying the number of meetings ($n$=5, $k$=3, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

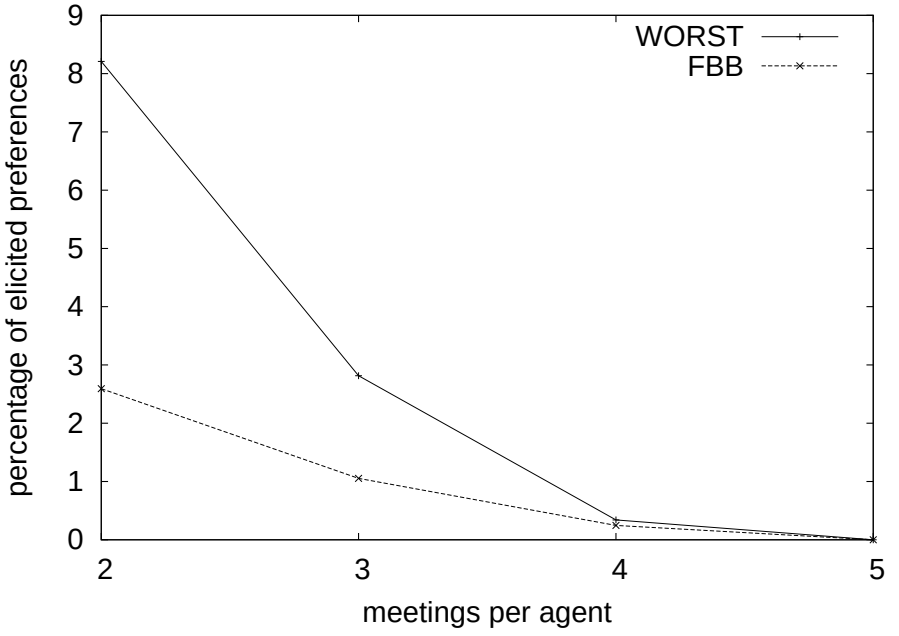



Figure 3.71: Percentage of elicited tuples varying meetings per agent ($m$=10, $n$=5, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

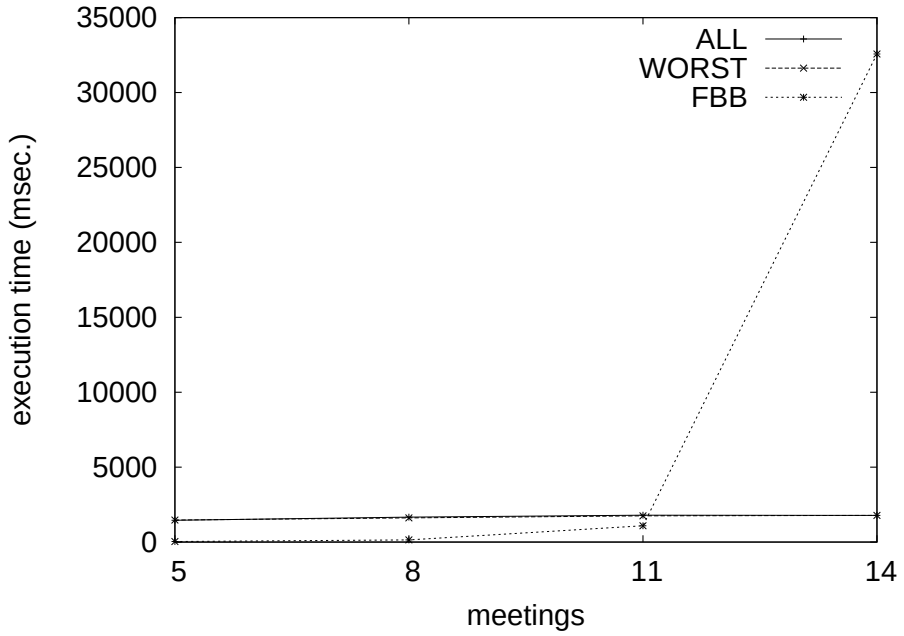
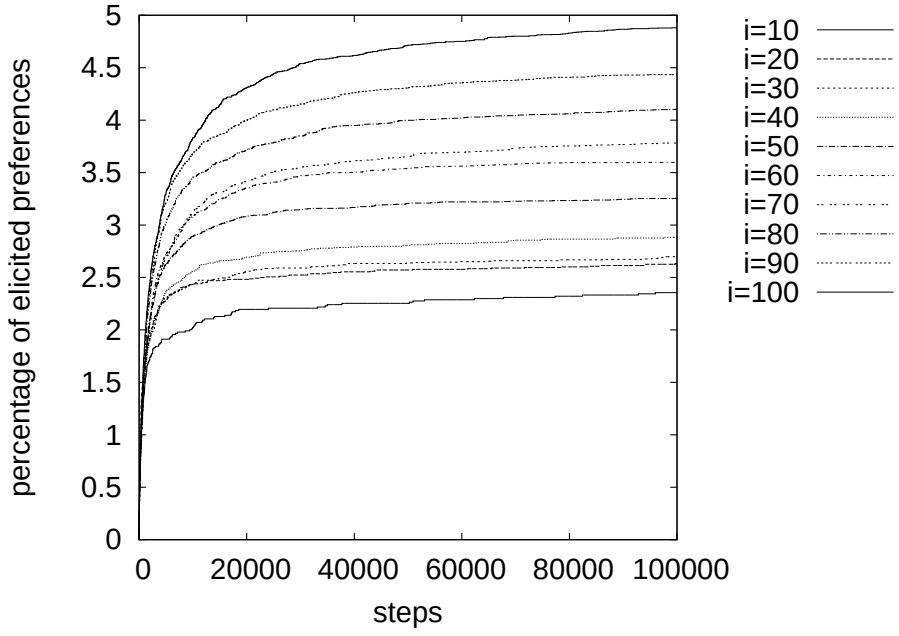


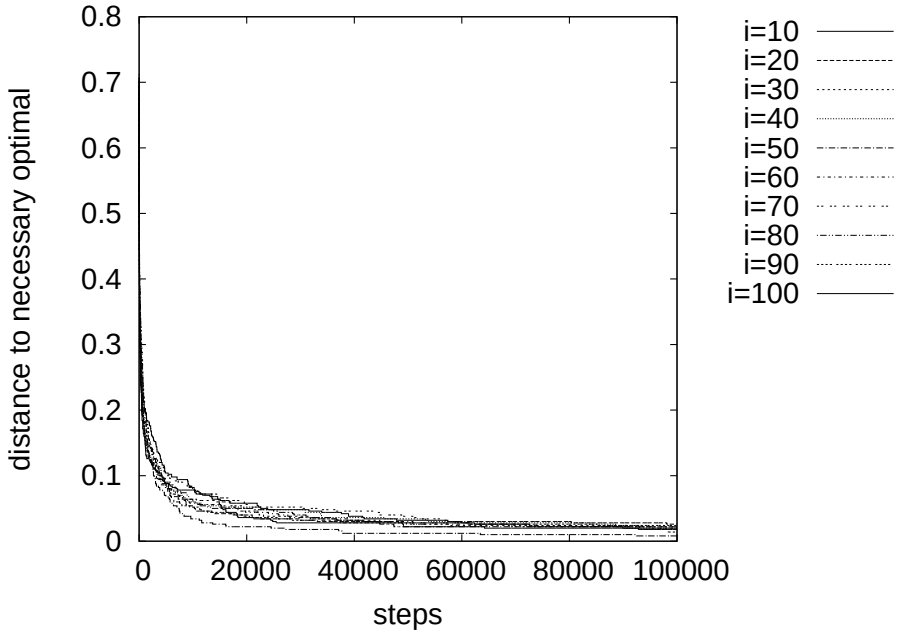

Figure 3.72: Execution time varying the number of meetings ($n$=5, $k$=3, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

strategy as it elicits from 60% to 80% of the incomplete preferences. In all settings, the WORST strategy elicits more preferences than FBB. This is because, in general, the branch and bound search approach used in FBB is more efficient in finding the way to reach a solution than local search. The local search approach is more efficient in terms of execution time when the number of meetings (i.e., the variables in our modeling) rises. In Figure 3.72 we can easily see that, for more than 11 meetings, FBB takes much more time than the WORST and ALL local search strategies.

To complete our treatment of incomplete meeting scheduling problems via a local search approach, we consider the runtime behavior of the best local search strategy that we found, that is, WORST. As for other classes of incomplete soft constraint problems, WORST elicits all the preferences needed to reach a solution the during the first steps. Moreover, the distance to the necessarily optimal preference decreases very rapidly during the first 20000 steps and then decreases slightly as the search continues. This behavior is the same no matter which parameter is varying. Thus we could stop the algorithm after only 20-30000 steps whilst still ensuring good solutions and faster execution time. Finally, we have to notice that the particular structure of the meeting scheduling problems does not influence the runtime of our

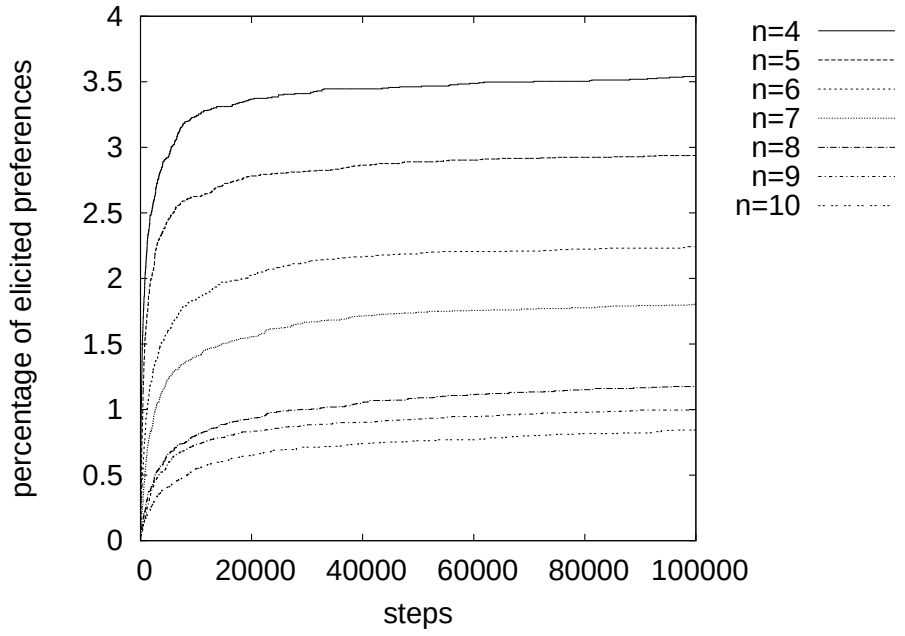



(a)

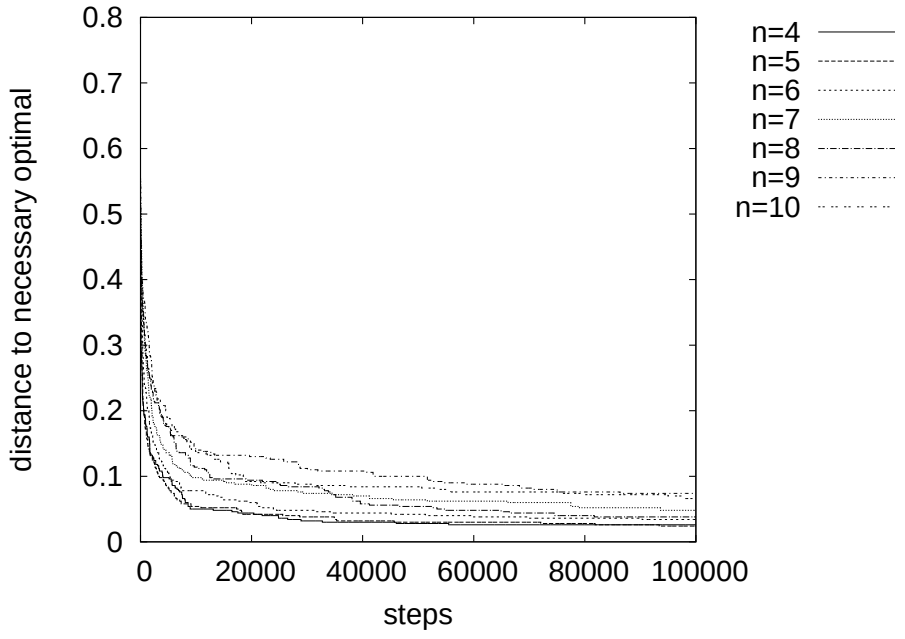



(b)

Figure 3.73: Runtime behavior of WORST varying incompleteness ($m$=10, $n$=5, $k$=3, $l$=10, $min$=1, $max$=2) in IMSPs.

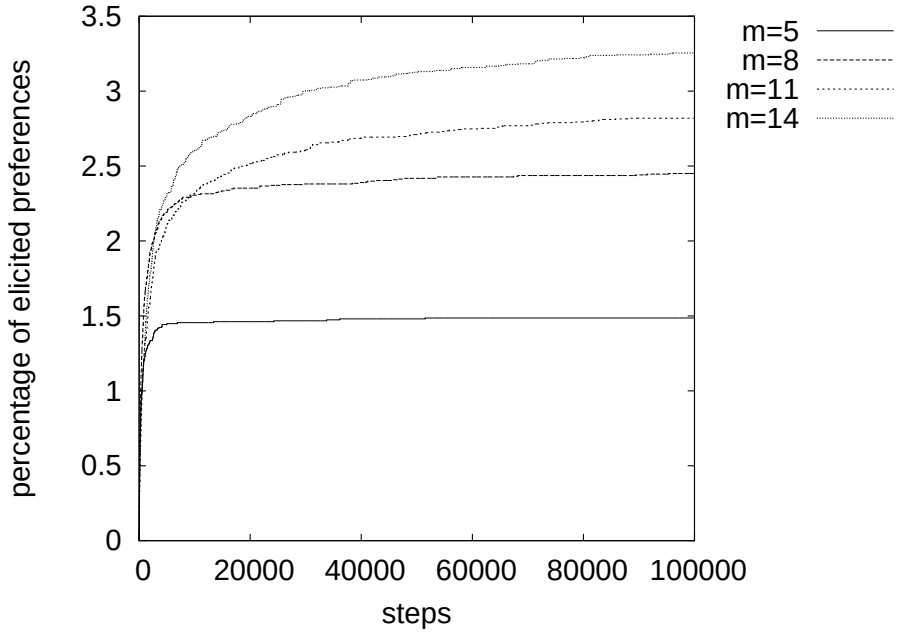



(a)

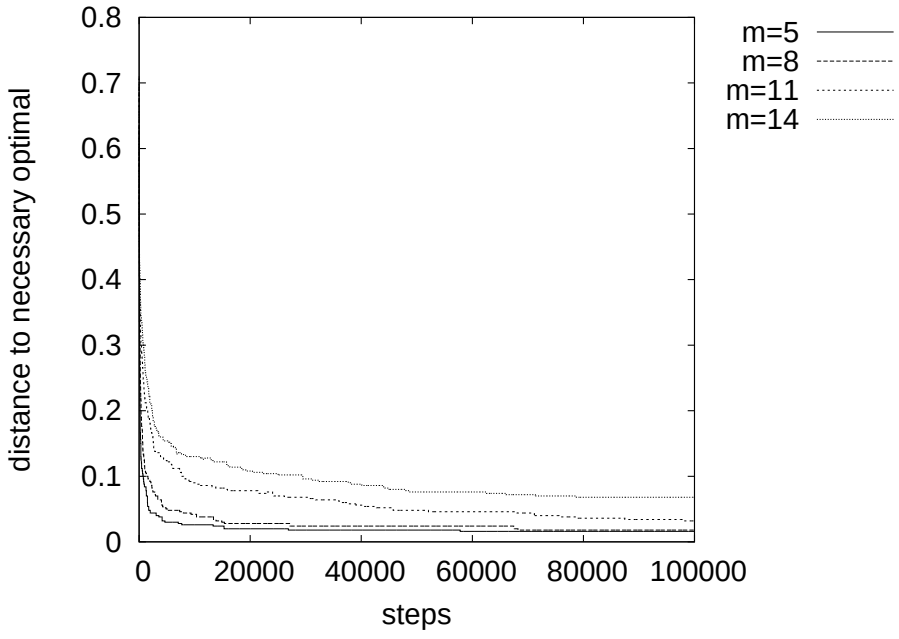



(b)

Figure 3.74: Runtime behavior of WORST varying the number of agents ($m$=10, $k$=3, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

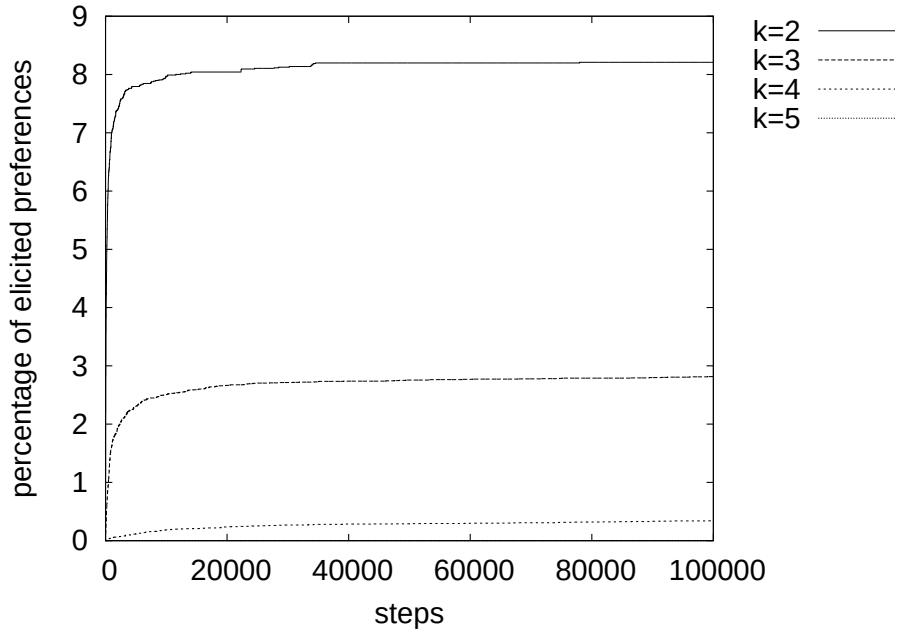



(a)

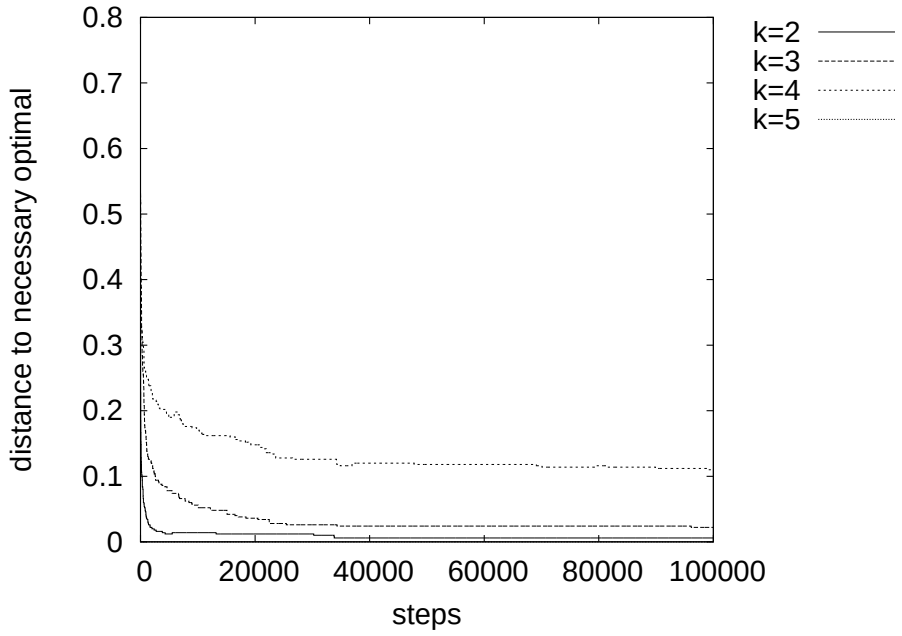



(b)

Figure 3.75: Runtime behavior of WORST varying the number of meetings ($n$=5, $k$=3, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.




(a)




(b)

Figure 3.76: Runtime behavior of WORST varying meetings per agent ($m$=10, $n$=5, $l$=10, $i$=30%, $min$=1, $max$=2) in IMSPs.

algorithm.

## 3.7 Related work

Open settings in CSPs was addressed in past years by both Open Constraint Satisfaction Problems (OCSPs) [20, 22] and Interactive CSPs [16] where domains can be partially specified.

### Open constraint programming

Instead of restricting to a *closed world* scenarios as classical CSPs, the OCSP framework can be used to model *open world* settings where domains and constraints are incrementally discovered through the network. In fact, tradictionaly, variables, domains and constraints are completely known before the search process starts. In OCSPs values are missing and so, most CSPs techniques (such as arc consistency for example) no longer work because they are based on the complete knowledge of the problem. With this in mind, the aim of OCSP algorithms is to solve problems by asking (through a mediator) only a fraction of the values. The mediator is simply a directory that indexes the informations available in different servers.

More precisely, an OCSP is a, possibly unbounded, partially ordered set $\{CSP(0), CSP(1), ...\}$ of constraint satisfaction problems. The set is ordered by the relation $\prec$ where $CSP(i) \prec CSP(j)$ if and only if $(\forall k \in [1, ..., n])$ $d_k(i) \subseteq d_k(j)$ and $(\exists k \in [1, ..., n])$ $d_k(i) \subset d_k(j)$.

A solution of an OCSP is an assignment to all variables such that all constraints as satisfied for some instance $CSP(i)$ and any instance $CSP(j) \succ CSP(i)$.

Since in the OCSP framework new information represents additional options and thus enables additional solutions, the algorithms gather new values until a solution is found.

To simplify the algorithm, authors make the assumption that all domain are discrete, and all constraints are encoded using the *hidden variable encoding* [62, 66]. This means that constraints are represented with additions variables with tuple-valued domains representing the corresponding constraints relations.

Instead of asking new values at random, the algorithms focus on those parts of the problem that cause failure. In fact, it will not be possible to create a solution by information gathering unless values are added to variables and relations of that subproblem. Thus, the idea is to find a variable that must be part of an unsolvable subproblem as a promising candidate for adding

extra values. To do this, a backtrack search algorithm is used to find an inconsistent variable. Then the variable is passed to the mediator, which will search for relevant information on the network. When there are no additional values for this variable other variables are then considered.

Therefore, the algorithm does the following:

1. Use a backtracking procedure to find, if any, a variable $x_k$ that has no consistent value in its domain. If it finds a solution, returns it and exit.

2. If there are no other possible values to ask for, exits with failure.

3. Otherwise, asks more consistent values for $x_k$.

4. Restart from step 1 with reordered variable such that $x_k$ becomes $x_1$ ($x_k$ is instantiated first).

An important thing to notice is that the set of unsolvable subproblems in successive instances of an OCSP is monotonically non-increasing and so, as the information is gathered, the number of unsolvable subproblem does not increase and, if an OCSP is solvable, eventually the algorithm asks for all the undefined values and finds a solution.

OCSP are then extended by Faltings et al. in [21, 22] to *Open Constraint Optimization* (OCOP). They made the constraint $soft$ by assigning a cost to each tuple. To solve such a problems in an open setting they show that is required an additional assumption that the variable domains and relations are revealed in non-decreasing order of preference/cost, presenting a variety of algorithms for solving OCOP in the probabilistic and weighted model.

To solve OCOP the authors use the same approach used for OCSP that is, finds conditions that guarantee that a solution of an instance $COP(i)$ is also optimal for all $COP(j)$, $COP(i) \prec COP(j)$. Moreover, similar to the constraint satisfaction case, the hidden-variable encoding is used to tranform the problem into one where only variable domains are open.

## Interactive constraint satisfaction

In [16] Cucciara et. al, propose the *Interactive Constraint Satisfaction* (ICSP) model, in which some (or all) domains can be partially defined. They propose this framework mainly to deal with problems in which retrieving new information is computationally expensive and so, their aim is the performance and not, for example, minimizing the number of values requested to the user.

The ICSP model deals with the incompleteness of the domains by adding new constraints on undefined parts of domains. These new constraints can

be added incrementally to the problem, acquiring new information without restarting a constraint propagation process from scratch each time new information is available. Hence, in ICSPs domains can be partially defined when the constraint satisfaction process starts.

Each domain $D_i$ of variable $x_i$ has a defined part named $Def_i$ and an undefined part called $UnDef_i$. To model the undefined part the authors use a domain variable representing the information which is not yet available.

In [50] Cucciara et. al propose a variant of forward checking algorithm applied to ICSP (called *Interactive Forward Checking, IFC*). At every step they interleave a search phase with a value acquisition phase guiding the search and the value acquisition with new acquired knowledge expressed in form of new constraints on the yet unknown part of the domains.

Experimental results on randomly generated CSPs and for 3D object recognition showed the effectiveness of the ICSP approach with a speedup ranging between 2 and 8 respect to CSPs approach.

## Expected cost-based interactive constraint satisfaction problems

Another different way to deal with incomplete problems is to take the elicitation cost into account. In [72] Wilson et al. use the cost to model the difficulty that could be rise to do constraint check in certain situation like, for example, when we need to pay for an option to be available. So they consider algorithms which take these costs into account and the target is to minimize the total cost.

Constraints may be initially incomplete: it may be unknown whether certain tuples satisfy the constraint or not. It is assumed that such an unknown tuple can be determined, but doing so incurs a known cost, which may vary between tuples. They also assume that they know the probability of an unknown tuple satisfying a constraint. These problems are called *Expected Cost-based Interactive CSP (ECI CSP).*

The authors use a set of variables $U$, which they call the set of *unknowns*. Variables in $U$ are boolean and uncertain and there is no control over them. To model the incompleteness of the problem each constraint tuple has an associated unknown.

To evaluate an assignment, the idea is to delay determining an unknown, in order to find out if it is worth doing so, starting from the ones that are less expensive and with less probability to be equal to 1. It is because is better to fail as soon as possible avoiding a fail with an high cost.

The same idea is also behind an algorithm developed to solve ECI CSP.

It performs a series of depth-first backtree-like search using a variable $Q$, interpreted as the cost that the algorithm is currently prepared to incur to find a solution, that is used in the backtracking condition and is incremented with each new search. At each leaf of the search tree, we can no longer delay determining unknowns, so we determine each current unknown until we fail, or until all have been determined successfully. If an unknown is determined unsuccessfully, then there is no solution beneath this node.

## 3.8 Conclusions

We have considered soft constraint problems with missing preferences. We have extended the soft constraint formalism to handle such a form of incompleteness. More precisely, we have defined two new notions of optimality: the possibly optimal solutions are solution which are optimal in at least one way of revealing missing preferences, while the necessarily optimal solutions are optimal in all ways of revealing the missing preferences. We have also characterized the sets of these optimal solutions. Moreover, we have presented an algorithmic schema, based on an branch & bound approach, that can be instantiated in several ways depending on who elicits, what to elicit, and when to elicit. We compared our algorithms, both on incomplete fuzzy CSPs and on weighted CSPs, by measuring the percentage of elicited preferences and the user's effort to answer the elicitation queries. For fuzzy problems, the percentage of elicited preferences for the best algorithms is below 10%. As expected, weighted problems are more difficult to handle, due to their additive nature. However, the best solvers need to elicit at most 30% of the missing preferences. In addition, we have considered incomplete CSPs, where the soft constraints are indeed hard constraints. Experimental results have shown a trend which is similar to the one registered for incomplete fuzzy CSPs. We have also considered structured problems, such as fuzzy simple temporal problems and meeting scheduling problems. In the temporal problem context, the experimental results show that, to find a necessarily optimal solution, it is sufficient to ask about 10% of the missing preferences. Concerning the meeting scheduling problems, the best algorithms elicit about 5% (and even less) of the missing preferences.

We then have focused our attention on solving larger instances of incomplete soft constraint problems. To do this, we have developed a local search algorithm to find necessarily optimal solutions, and we have tested it on both incomplete fuzzy and weighted CSPs. We have considered several elicitation strategies. In both test sets, our best strategies have shown good results compared with our systematic search algorithms in terms of the quality of

the solutions. Moreover, our local search approach has also shown a much better scalability behaviour than the branch & bound method.

# Chapter 4

# Imprecision in soft constraints

In this chapter we extend the semiring-based soft constraints [6, 7] in order to allow for imprecise preferences. More precisely, we allow for a preference interval instead of a single preference value. In this context, we study how to find an optimal solution according to several optimality notions. We also define algorithms to find such optimal solutions and to test whether a solution is optimal.

Some of the results in this chapter are included in the following articles:

- *Interval-valued Soft Constraint Problems*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and N. Wilson, Annals of Mathematics and Artificial Intelligence, special issue for ISAIM 2008, B. Chouery and B. Givan eds., Springer, to appear.

- *Imprecise Soft Constraint Problems*, Mirco Gelain, Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable and Nic Wilson, in proc. of the 9th Workshop on Preferences and Soft Constraints (SofT'08), Sydney, Australia, September 2008.

## 4.1 Motivations

Constraints [61] are useful to model real-life problems when it is clear what should be accepted and what should be forbidden. Soft constraints [55] extend the constraint notion by allowing several levels of acceptance. This allows one to express preferences and/or costs rather than just strict requirements.

In soft constraints, each instantiation of the variables of a constraint must be associated to a precise preference or cost value. Sometimes it is not possible for a user to know exactly all these values. For example, a user may

have a vague idea of the preference value, or may not be willing to reveal his preference, for example for privacy reasons.

In this chapter we consider these forms of imprecision, and we handle them by extending soft constraints to allow users to state an interval of preference values for each instantiation of the variables of a constraint.

This interval can contain a single element (in this case we have usual soft constraints), or the whole range of preference values (when there is complete ignorance about the preference value), or it may contain more than one element but a strict subset of the set of preference values. We call such problems *interval-valued* soft CSPs (or also IVSCSPs).

In an elicitation procedure there will typically be some degree of imprecision, so attributing an interval rather than a precise preference degree can be a more reliable model of the information elicited. Also, linguistic descriptions of degrees of preference (such as "quite high" or "low" or "undesirable") may be more naturally mapped to preference intervals, especially if the preferences are being elicited from different experts, as they may mean somewhat different things by these terms.

Two examples of real world application domains where preference intervals can be useful or necessary are energy trading and network traffic analysis [73], where the data information is usually incomplete or erroneous. In energy trading, costs may be imprecise because they may evolve due to market changes; in network traffic analysis, the overwhelming amount of information and measurement difficulties force the use of partial or imprecise information. Many other application domains that are usually modelled via hard or soft constraints could benefit by increased expressed power of preference intervals. To give a concrete example we consider the meeting scheduling problem, that is a typical benchmark for CSPs, and we allow the specification of preference intervals. This benchmark will be used both to clarify notions related to IVCSPs and to run experimental tests.

Given an IVSCSP, we consider several notions of optimal solutions. We first start with general notions of optimality, which apply whenever we have several scenarios to consider. For example, as done in Chapter 3, we consider *necessarily optimal* solutions, which are optimal in all scenarios, or *possibly optimal* solutions, which are optimal in at least one scenario. We then pass to *interval-based optimality notions*, that define optimality in terms of the upper and lower bounds of the intervals associated to the solution by the constraints.

Since IVSCSPs generalize soft constraint problems, the problem of finding an optimal solution in an IVSCP (according to any of the considered optimality notions) is at least as difficult as finding an optimal solution in a soft constraint problem and thus it is NP-hard.

We provide algorithms to find solutions according to all the notions defined, and also to test whether a given solution is optimal. In most of the cases, finding or testing an optimal solution amounts to solving a soft constraint problem. Thus, even if our formalism significantly extends soft constraints, and gives users much more power in modelling their knowledge of the real world, in the end the work needed to find an optimal solution (or to test if it is optimal) is not more than that needed to find an optimal solution in a soft constraint problem. This claim is supported by the experimental results we present, obtained by extensive tests over instances of the meeting scheduling problem.

We also show that for some classes of IVSCSPs the optimality notions would not produce different results if users were allowed to use *multiple disjoint intervals* rather than a single one. This means that a level of precision greater than a single interval does not add any useful information when looking for an optimal solution.

## 4.2 Interval-valued soft constraints

Soft constraint problems require users to specify a preference value for each tuple in each constraint. Sometimes this is not reasonable, because a user may have a vague idea of what preferences to associate to some tuples. In Chapter 3 we give generalization allowed users to specify either a fixed preference (as in usual soft constraints) or the complete $[\mathbf{0}, \mathbf{1}]$ interval. Thus an assumption of complete ignorance was made when the user was not able to specify a fixed preference. Here we generalize further by allowing users to state any interval over the preference set.

**Definition 28** (interval-valued soft constraint)**.** *Given a set of variables $V$ with finite domain $D$ and a* totally-ordered *c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, an interval-valued soft constraint is a pair $\langle int, con\rangle$ where $con \subseteq V$ is the scope of the constraint and int: $D^{|con|} \longrightarrow A \times A$ specifies an interval over $A$ by giving its lower and upper bound. If $int(x) = (a, b)$, it must be $a \leq_S b$.*

In the following we will denote with $l(int(x))$ (resp., $u(int(x))$) the first (resp., second) component of $int(x)$, representing the lower and the upper bound of the preference interval.

**Definition 29** (IVSCSP)**.** *An interval-valued soft constraint problem (IVSCSP) is a 4-tuple $\langle V, D, C, S\rangle$, where $C$ is a set of interval-valued soft constraints over $S$ defined on the variables in $V$ with domain $D$.*

Figure 4.1 shows an IVSCSP $P$ defined over the fuzzy c-semiring $\langle [0,1], max, min, 0, 1\rangle$, that contains three variables $X_1$, $X_2$, and $X_3$, with domain $\{a, b\}$, and five constraints: a unary constraint on each variable, and two binary constraints on $(x_1, x_2)$ and $(x_2, x_3)$.

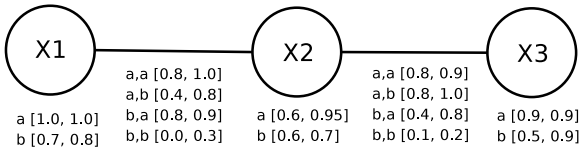



Figure 4.1: An IVSCSP over fuzzy semiring.

In an IVSCSP, a complete assignment of values to all the variables can be associated to an interval as well. The lower bound (resp., the upper bound) of such an interval is obtained by combining all the lower bounds (resp., the upper bounds) of the preference intervals of the appropriate subtuples of this assignment in the various constraints.

**Definition 30** (preference interval)**.** *Given an IVSCSP* $P = \langle V, D, C, S\rangle$ *and an assignment* $s$ *to all its variables over* $D$*, the preference interval of* $s$ *in* $P$ *is* $[L(s), U(s)]$*, where* $L(s) = \Pi_{<int,con>\in C} l(int(\ s_{\downarrow con}))$ *and* $U(s) = \Pi_{<int,con>\in C} u(int(s_{\downarrow con}))$*, and* $\Pi$ *is the combination operator of the c-semiring* $S$*.*

Figure 4.2 shows all the complete assignments of the IVSCSP in Figure 4.1, together with their preference interval and the computation details for $s_1$.

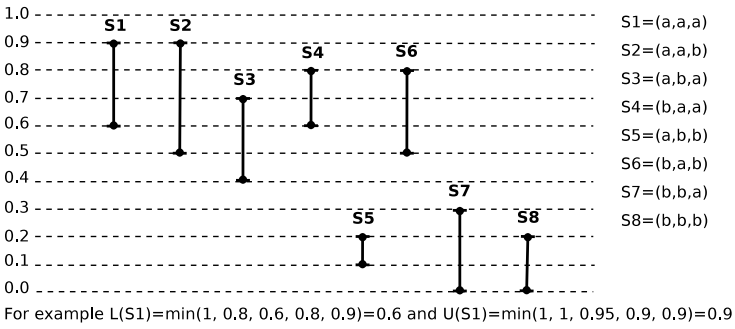



Figure 4.2: Solutions of the IVSCSP shown in Figure 4.1.

Once we have an IVSCSP, it is useful to consider specific scenarios arising from choosing a preference value from each interval.

**Definition 31** (scenario)**.** *Given an IVSCSP $P$, a scenario of $P$ is an SCSP $P'$ obtained from $P$ as follows: given any constraint $c = \langle int, con\rangle$ of $P$, we insert in $P'$ the constraint $c' = \langle def, con\rangle$, where $def(t) \in [l(int(t)), u(int(t))]$ for every tuple $t \in D^{|con|}$.*

We will denote with $Sc(P)$ the set of all possible scenarios of $P$.

**Definition 32** (best and worst scenario)**.** *Given an IVSCSP $P$, the best scenario ($bs(P)$) (resp., the worst scenario ($ws(P)$)) of $P$ is the scenario obtained by replacing every interval with its upper (resp., lower) bound.*

We will denote with $l_{opt}$ and $u_{opt}$ the optimal preferences of the worst and best scenario.

The preference interval of a complete assignment is a good way of representing the quality of the solution in all scenarios, as stated by the following theorem.

**Theorem 10.** *Consider an IVSCSP $P$ over a c-semiring $S$ and a complete assignment $s$ of its variables. Then, for all $Q \in Sc(P)$, $\text{pref}(Q, s) \in [L(s), U(s)]$. Also, for $p \in \{L(s), U(s)\}$, there exists a $Q \in Sc(P)$ such that $p = \text{pref}(Q, s)$. If the c-semiring $S$ is idempotent, then for all $p \in [L(s), U(s)]$, there exists a $Q \in Sc(P)$ such that $p = \text{pref}(Q, s)$.*

**Proof:** $\text{pref}(Q, s) \in [L(s), U(s)]$ follows by monotonicity. If $p = L(s)$ (resp., $p = U(s)$), it is possible to build a scenario where $p = pref(Q, s)$, by fixing all the tuples of $s$ to their lower bound (resp., to their upper bound). If the c-semiring is idempotent, since we are considering totally ordered c-semirings, the operator $\times$ is minimum (with respect to the total order), so there exists some interval-valued constraint $\langle int, con\rangle$ in $P$ such that $l(int(s_{\downarrow con})) = L(s)$. We must also have $u(int(s_{\downarrow con})) \geq U(s)$. Let $p$ be an element of $[L(s), U(s)]$. Define a scenario $Q$ by replacing this interval-valued constraint with any soft constraint which assigns the tuple $s_{\downarrow con}$ the preference value $p$, and replacing any of the other elements of $P$ with soft constraints which assign preference value $U(s)$ to the appropriate projection of $s$. We then have $p = \text{pref}(Q, s)$. □

This means that, in general, the upper and lower bounds of the solution preference interval always model preferences of solutions in some scenarios. In the idempotent case we have more: the whole interval, and not just the bounds, represents all and only the preferences coming from the scenarios. Intuitively, if $\times$ is idempotent (let us consider min for simplicity): given an assignment $s$, for every element $x$ in $[L(s), U(s)]$, we can construct a scenario where $s$ has preference $x$ by fixing preference $x$ on at least one tuple (that

has $x$ in its interval) and by fixing all other preferences of tuples in $s$ to their upper bound.

## 4.3 Necessary and possible optimality

We will now consider general notions of optimality, that are applicable to any setting where the lack of precision gives rise to several possible scenarios. First we define optimal solutions that guarantee optimality in some or all scenarios (i.e., the possibly and the necessarily optimal solutions we see in Chapter 3), and then we will define solutions that guarantee a certain level of preference in some or all scenarios.

**Definition 33** (necessarily optimal)**.** *Given an IVSCSP $P = \langle V, D, C, S\rangle$ and an assignment $s$ to the variables in $V$, $s$ is necessarily optimal iff it is optimal in all scenarios.*

Given an IVSCSP $P$, the set of its necessarily optimal solutions will be denoted by $NO(P)$. Necessarily optimal solutions are very attractive because they are very robust: they are optimal independently of the uncertainty of the problem. Unfortunately, $NO(P)$ may be empty, as in the IVSCSP $P$ of Figure 4.1.

**Definition 34** (possibly optimal)**.** *Given an IVSCSP $P = \langle V, D, C, S\rangle$ and an assignment $s$ to the variables in $V$, $s$ is possibly optimal iff it is optimal in some scenario.*

Given an IVSCSP $P$, the set of possibly optimal solutions of $P$ will be denoted by $PO(P)$. In the IVSCSP $P$ of Figure 4.1 we have $PO(P) = \{s_1, s_2, s_3, s_4, s_6\}$. $PO(P)$ is never empty. However, the possibly optimal solutions are less attractive than the necessarily optimal ones, in fact they guarantee optimality only for a specific completion of the uncertainty.

We assume now to want to guarantee a certain level of preference in some or all the scenarios.

**Definition 35** (necessarily of at least preference $\alpha$)**.** *Given an IVSCSP $P = \langle V, D, C, S\rangle$ and an assignment $s$ to the variables in $V$, $s$ is necessarily of at least preference $\alpha$ iff, for all scenarios, its preference is at least $\alpha$.*

The set of all solutions of an IVSCSP $P$ with this feature will be denoted by $Nec(P, \alpha)$. In our running example, if we consider $\alpha = 0.5$, we have $Nec(P, 0.5) = \{s_1, s_2, s_4, s_6\}$. If $\alpha$ is a satisfactory preference level, elements in $Nec(P, \alpha)$ are ideal, because they guarantee such a preference level independently of the uncertainty of the problem.

**Definition 36** (possibly of at least preference $\alpha$)**.** *Given an IVSCSP $P = \langle V, D, C, S\rangle$ and an assignment $s$ to the variables in $V$, $s$ is possibly of at least preference $\alpha$ iff, for some scenario, its preference is at least $\alpha$.*

The set of all solutions of an IVSCSP $P$ with this feature will be denoted by $Pos(P, \alpha)$. In the IVSCSP $P$ of Figure 4.1, if we take $\alpha = 0.3$, we have $Pos(P, 0.3) = \{s_1, s_2, s_3, s_4, s_6, s_7\}$.

## 4.4 Interval-based optimality notions

In an IVSCSP, uncertainty is specified via the preference intervals. Depending on how one decides to deal with this form of uncertainty, different notions of optimality can be given. Here we will consider interval-based optimality notions, and we will relate them to the necessarily and possibly optimal solutions.

### Interval-dominant assignments

In the attempt to characterize the necessarily optimal solutions, we can consider the following notion.

**Definition 37** (interval-dominant)**.** *Given an IVSCSP $P = \langle V, D, C, S\rangle$ and an assignment $s$ to the variables in $V$, $s$ is interval-dominant iff, for every other complete assignment $s'$, $L(s) \geq U(s')$.*

Interval-dominant assignments are better than or equal to all others in all scenarios, and thus are very robust w.r.t. uncertainty. We denote with $ID(P)$ the set of the interval dominant assignments of $P$. The IVSCSP $P$ of Figure 4.1 has $ID(P) = \emptyset$.

**Proposition 1.** *If $ID(P) \neq \emptyset$, either $ID(P)$ contains a single solution, or all the solutions in $ID(P)$ have their lower bound equal to their upper bound, and all these bounds are equal to the same value. Given an IVSCSP $P$, $ID(P)$ may be empty.*

**Proof:** $ID(P)$ may be empty as in the IVSCSP $P$ of Figure 4.1.

We now show, by contradiction, that if $ID(P) \neq \emptyset$, either $ID(P)$ contains a single solution, or several solutions all with the lower bound equal to the upper bound, and all equal to the same value. If $ID(P)$ contains two solutions, say $s_1$ and $s_2$, with different values of lower and upper bounds, then $L(s_1) < U(s_1)$ and $L(s_2) < U(s_2)$. Since $s_1 \in ID(P)$, then for

any other solution $s'$, $L(s_1) \geq U(s')$ and thus also $L(s_1) \geq U(s_2)$. Similarly, since $s_2 \in ID(P)$, then for any other solution $s'$, $L(s_2) \geq U(s')$ and thus $L(s_2) \geq U(s_1)$. Therefore, $L(s_1) \geq U(s_2) > L(s_2) \geq U(s_1)$ and so $L(s_1) > U(s_1)$, that is a contradiction. □

It is possible to show that the interval-dominant optimality notion is stronger than the necessary optimality notion. More precisely:

**Proposition 2.** *Given an IVSCSP $P$, we have that $ID(P) \subseteq NO(P)$. Also, if $ID(P) \neq \emptyset$, then $ID(P) = NO(P)$.*

**Proof:** We first show that $ID(P) \subseteq NO(P)$. If a solution is in $ID(P)$, its preference is always greater than or equal to the upper bounds of all the other solutions, hence it is optimal in all the scenarios.

We now prove that, if $ID(P) \neq \emptyset$, then $ID(P) = NO(P)$. We have already shown that $ID(P) \subseteq NO(P)$. It remains to prove that $NO(P) \subseteq ID(P)$. Let us denote with $s^*$ a solution of $ID(P)$. If a solution $s$ of $P$ is not in $ID(P)$ and $ID(P) \neq \emptyset$, then $s$ is not in $NO(P)$. In fact, if $L(s^*) \neq U(s^*)$, then $U(s^*) > L(s^*) \geq U(s)$, and so $s$ is not optimal in the best scenario. If $L(s^*) = U(s^*)$, since $s \notin ID(P)$, $L(s) < L(s^*)$ and so $s$ is not optimal in the worst scenario. □

## Weakly-interval-dominant assignments

A more relaxed interval-based optimality notion is the following one.

**Definition 38** (weakly-interval-dominant)**.** *Given an IVSCSP $P = \langle V, D, C, S \rangle$ and an assignment $s$ to the variables in $V$, $s$ is weakly-interval-dominant iff, for every other complete assignment $s'$, $L(s) \geq L(s')$ and $U(s) \geq U(s')$.*

Weakly-interval-dominant assignments are better than or equal to all others in both the worst and the best scenario. We denote with $WID(P)$ the set of the weakly interval dominant assignments of $P$. The IVSCSP $P$ of Figure 4.1 has $WID(P) = \{s_1\}$.

**Proposition 3.** *Given an IVSCSP $P$, $WID(P)$ may be empty. Moreover, $ID(P) \subseteq WID(P)$.*

**Proof:** $WID(P)$ may be empty. For example, one can construct an IVSCSP over fuzzy c-semiring with only three solutions, say $s_1$, $s_2$, and $s_3$, with the following lower and upper bounds: $L(s_1) = 0.2$, $U(s_1) = 0.6$, $L(s_2) = 0.3$, $U(s_2) = 0.8$, $L(s_3) = 0.4$, and $U(s_3) = 0.7$.

We now show that $ID(P) \subseteq WID(P)$. If $s \in ID(P)$, then $L(s) \geq U(s')$ for every other $s'$. Hence, since $U(s) \geq L(s)$ and $U(s') \geq L(s')$ for every other $s'$, we have $U(s) \geq L(s) \geq U(s') \geq L(s')$ for every other $s'$, that is, $U(s) \geq U(s')$ and $L(s) \geq L(s')$ for every other $s'$, hence $s \in WID(P)$. □

The weakly-interval-dominant optimality notion is weaker than the necessary optimality notion. In fact, $NO(P) \subseteq WID(P)$ and for some IVSCSP $P$ (for example, the IVSCSP of Figure 4.1) this inclusion is strict. More precisely:

**Proposition 4.** *Given an IVSCSP $P$, we have that $ID(P) \subseteq NO(P) \subseteq WID(P)$.*

**Proof:** By Proposition 2, we know that $ID(P) \subseteq NO(P)$.

We now show that $NO(P) \subseteq WID(P)$. If $s \in NO(P)$, then $s$ must be optimal in every scenario and so also in the best and in the worst scenario. Given that $s$ is optimal in the worst scenario, then $L(s) \geq L(s')$ for every other solution $s'$. Moreover, as $s$ is optimal in the best scenario, then $U(s) \geq U(s')$ for every other solution $s'$. Therefore, $L(s) \geq L(s')$ and $U(s) \geq U(s')$ for every other solution $s'$. This allows us to conclude that $s \in WID(P)$. □

Since $ID(P) \subseteq NO(P) \subseteq WID(P)$, $ID(P)$ and $WID(P)$ can be seen as lower and upper approximations of $NO(P)$.

## Lower and upper optimal assignments

Until now we have considered how to characterize, via interval-based optimality notions, the necessarily optimal solutions. In particular, we have found lower and upper approximations of these optimal solutions. We now move to consider possibly optimal solutions via new interval-based optimality notions.

**Definition 39** (lower and upper optimal)**.** *Given an IVSCSP $P = \langle V, D, C, S\rangle$ and an assignment $s$ to the variables in $V$, $s$ is lower-optimal (resp., upper-optimal) iff, for every other complete assignment $s'$, $L(s) \geq L(s')$ (resp., $U(s) \geq U(s')$).*

A lower-optimal (resp., an upper-optimal) assignment is better than or equal to all other complete assignments in the worst scenario (resp., in the best scenario). Lower-optimal (resp., upper-optimal) assignments are useful in pessimistic (resp., optimistic) approaches to uncertainty, because they outperform the other assignments in the worst (resp., in the best) case. We

denote with $LO(P)$ (resp., $UO(P)$) the set of the lower (resp., upper) optimal assignments of $P$. The IVSCSP $P$ of Figure 4.1 has $LO(P) = \{s_1, s_4\}$ and $UO(P) = \{s_1, s_2\}$.

Lower and upper optimal solutions are never empty. Moreover, they are related to weakly-interval-dominant and interval-dominant solutions as follows.

**Proposition 5.** *Given an IVSCSP $P$, and the optimal preference $l_{opt}$ (resp., $u_{opt}$) of $ws(P)$ (resp., $bs(P)$),*

- *$LO(P)$ and $UO(P)$ are never empty;*
- *$UO(P) \cap LO(P) = WID(P)$;*
- *if $l_{opt} = u_{opt}$, then $ID(P) = LO(P)$;*
- *if $l_{opt} < u_{opt}$, and $|UO(P)| \geq 2$, then $ID(P) = \emptyset$;*
- *if $|UO(P)| = 1$, let us call $s$ this single solution. If $L(s) \neq l_{opt}$ then $ID(P) = \emptyset$.*

**Proof:** $LO(P)$ is never empty because it is always possible to find the solutions with the lower bound greater than or equal to all the other solutions. A similar argument shows that $UO(P)$ is never empty.

We now show that $UO(P) \cap LO(P) = WID(P)$. We first show that $UO(P) \cap LO(P) \subseteq WID(P)$. If $s \in UO(P) \cap LO(P)$, then, by definition of $UO(P)$, $U(s) \geq U(s')$ for every other $s'$ and, by definition of $LO(P)$, $L(s) \geq L(s')$ for every other $s'$, therefore $s \in WID(P)$. We now show that $WID(P) \subseteq UO(P) \cap LO(P)$. If $s \in WID(P)$, by definition of $WID(P)$, $U(s) \geq U(s')$ and $L(s) \geq L(s')$ for every other $s'$, hence both $s \in LO(P)$ and $s \in UO(P)$, therefore $s \in LO(P) \cap UO(P)$.

To show that, if $l_{opt} = u_{opt}$, then $ID(P) = LO(P)$, it is sufficient to show that $l_{opt} = u_{opt}$ implies $LO(P) \subseteq ID(P)$, as $ID(P) \subseteq LO(P)$ follows from Theorem 2. In fact, if $s \in ID(P)$, then $s \in Opt(ws(P))$ and thus, by Theorem 2, $s \in LO(P)$. If $s \in LO(P)$ then $L(s) = l_{opt}$. Moreover, since $l_{opt} = u_{opt}$, $L(s) = u_{opt}$, and so $L(s) \geq U(s')$, for every other solution $s'$, that is $s \in ID(P)$.

We now prove, by contradiction, that, if $l_{opt} < u_{opt}$ and $|UO(P)| \geq 2$, then $ID(P) = \emptyset$. Suppose $ID(P) \neq \emptyset$. Let us denote with $s$ one of the solutions of $ID(P)$. Then, by definition of $ID(P)$, $L(s) \geq U(s')$, for every other solution $s'$. Since $|UO(P)| \geq 2$, we are sure that there is a solution $s'' \neq s$ such that $U(s'') = u_{opt}$. Hence, $L(s) \geq U(s'') = u_{opt} > l_{opt}$, and so $L(s) > l_{opt}$, that is a contradiction, because, by the definition of $l_{opt}$, $l_{opt}$ is greater than or equal to the lower bound of every solution.

Assume that $|UO(P)| = 1$ and let us call $s$ this single solution. We now show, by contradiction, that, if $L(s) \neq l_{opt}$, then $ID(P) = \emptyset$. Let us denote with $s_1$ one of the solutions with $L(s_1) = l_{opt}$. Suppose that $ID(P) \neq \emptyset$, and let $s'$ be an element of $ID(P)$. If $s' \neq s$ then $U(s') \geq L(s') \geq U(s)$, which implies that $s' \in UO(P)$, a contradiction. Hence $s' = s$. But then $s' \neq s_1$, so $L(s') \geq U(s_1) \geq L(s_1) = l_{opt}$, which contradicts $L(s) \neq l_{opt}$. □

As every lower (resp., upper) optimal solution is optimal in the worst (resp. best) scenario, then $LO(P) \subseteq PO(P)$, $UO(P) \subseteq PO(P)$, and these inclusions may be strict, because there may be solutions that are optimal only in scenarios that are different from the best and the worst scenario.

**Proposition 6.** *Given an IVSCSP $P$, we have that $LO(P) \cup UO(P) \subseteq PO(P)$.*

**Proof:** Let $s$ be a complete assignment to the variables of $P$.

$LO(P) \subseteq PO(P)$. In fact, if $s \in LO(P)$, then $s$ is optimal in the worst scenario and so $s \in PO(P)$.

$UO(P) \subseteq PO(P)$. In fact, if $s \in UO(P)$, then $s$ is optimal in the best scenario and so $s \in PO(P)$.

Therefore, $LO(P) \cup UO(P) \subseteq PO(P)$. □

Therefore, the lower and upper optimality notions are stronger than the possible optimality notion.

The lower and upper optimal assignments are also related to the necessarily and possibly of at least preference $\alpha$ assignments as follows.

**Proposition 7.** *Given an IVSCSP $P$ and the optimal preference $l_{opt}$ of $ws(P)$,*

- *$Nec(P, \alpha) \neq \emptyset$ iff $\alpha \leq l_{opt}$;*
- *if $\alpha \leq l_{opt}$, $LO(P) \subseteq Nec(P, \alpha)$;*
- *let $\alpha_*$ be the maximum $\alpha$ such that there exists a solution in $Nec(P, \alpha)$, then $\alpha_* = l_{opt}$ and $Nec(P, \alpha_*) = LO(P)$, and so $Nec(P, \alpha_*) \subseteq PO(P)$.*

**Proof:** Let us show the first item of the theorem. To show that $Nec(P, \alpha) \neq \emptyset$ iff $\alpha \leq l_{opt}$, we first prove that, if $Nec(P, \alpha) \neq \emptyset$, then $\alpha \leq l_{opt}$. If $Nec(P, \alpha) \neq \emptyset$, then there is a solution, say $s$, such that $\text{pref}(Q_i, s) \geq \alpha$ for every scenario $Q_i$ of $P$ and so also for the worst scenario. Hence, $l_{opt} \geq \text{pref}(ws(P), s) \geq \alpha$. Therefore, $l_{opt} \geq \alpha$. We now show that, if $\alpha \leq l_{opt}$, then $Nec(P, \alpha) \neq \emptyset$. If $Nec(P, \alpha) = \emptyset$, then for every solution $s$ we have that

$\text{pref}(Q_i, s) < \alpha$ for some scenario $Q_i$. This holds also for any solution, say $s^*$, such that $\text{pref}(ws(P), s^*) = l_{opt}$, and so $l_{opt} = \text{pref}(ws(P), s^*) < \alpha$.

We now show the second item of the theorem: given $\alpha \leq l_{opt}$, $LO(P) \subseteq Nec(P, \alpha)$. If $LO(P) \not\subseteq Nec(P, \alpha)$, then there is a solution, say $s$, such that $s \in LO(P) \setminus Nec(P, \alpha)$. Since $s \in LO(P)$, $\text{pref}(ws(P), s) = l_{opt}$. Since $s \notin Nec(P, \alpha)$, then $\text{pref}(Q_i, s) < \alpha$ for some scenario $Q_i$, and so, as $ws(P)$ is the worst scenario, $l_{opt} = \text{pref}(ws(P), s) \leq \text{pref}(Q_i, s) < \alpha$. Therefore, $l_{opt} < \alpha$.

We now show, by contradiction, that $\alpha_* = l_{opt}$. If $\alpha_* > l_{opt}$, then, by the previous part of the proof, $Nec(P, \alpha_*) = \emptyset$, that is a contradiction because $\alpha_*$ is the maximum $\alpha$ such that $Nec(P, \alpha) \neq \emptyset$. If $\alpha_* < l_{opt}$, then $\alpha_*$ is not the maximum $\alpha$ such that $Nec(P, \alpha) \neq \emptyset$, since such a value is $l_{opt}$, and so we have a contradiction.

We now prove that, if $\alpha_* = l_{opt}$, then $Nec(P, \alpha_*) = LO(P)$. Let $s$ be a complete assignment to the variables of $P$. If $s \in Nec(P, l_{opt})$, then for every scenario $Q$, $\text{pref}(Q, s) \geq l_{opt}$ and so also for the worst scenario. Therefore, as $l_{opt}$ is the optimal preference of the worst scenario, $s \in LO(P)$. If $s \in LO(P)$, then $\text{pref}(ws(P), s) = l_{opt}$. Since for every scenario $Q$, $\text{pref}(Q, s) \geq \text{pref}(ws(P), s) = l_{opt}$, then $s \in Nec(P, l_{opt})$.

Since $Nec(P, \alpha_*) = LO(P)$ and since, by Proposition 6, $LO(P) \subseteq PO(P)$, then $Nec(P, \alpha^*) \subseteq PO(P)$. □

Thus, in general, $Nec(P, \alpha)$ is not empty only if $\alpha$ is at most the optimal preference of the worst scenario, and in such a case every lower-optimal solution is in $Nec(P, \alpha)$. Moreover, if we consider a particular value of $\alpha$, also the converse holds. Therefore, in this case the necessarily of at least preference $\alpha$ solutions are lower-optimal solutions and thus they are possibly optimal solutions.

Moreover, a solution is in $Pos(P, \alpha)$ only if $\alpha$ is at most the optimal preference of the best scenario, and in such a case, for a particular value of $\alpha$, the possibly of at least preference $\alpha$ solutions coincide with the upper optimal solutions, and thus they are possibly optimal solutions.

**Proposition 8.** *Given an IVSCSP $P$ and an assignment $s$ to the variables of $P$,*

- *$s$ is in $Pos(P, \alpha)$ if and only if $\alpha \leq U(s)$;*
- *let $\alpha^*$ be the maximum $\alpha$ such that $Pos(P, \alpha)$ is not empty, then $Pos(P, \alpha^*) = UO(P)$, and so $Pos(P, \alpha^*) \subseteq PO(P)$.*

**Proof:** We first show that $s$ is in $Pos(P, \alpha)$ if and only if $\alpha \leq U(s)$. If $s \in Pos(P, \alpha)$, then there is a scenario where $\text{pref}(Q, s) \geq \alpha$. By Theorem 10,

we know that $U(s)$ is the highest preference associated to $s$ in any scenario, then $U(s) \geq \text{pref}(Q, s)$ and so $U(s) \geq \alpha$. If $\alpha \leq U(s)$, then, by Theorem 10, there is a scenario $Q$, where $\text{pref}(Q, s) = U(s)$. Since $U(s) \geq \alpha$, then $s \in Pos(P, \alpha)$.

We now show that $Pos(P, \alpha^*) = UO(P)$. If $s \in Pos(P, \alpha^*)$, then there is a scenario $Q$ where $\text{pref}(Q, s) \geq \alpha^*$. Since $\alpha^*$ is the maximum $\alpha$ such that $Pos(P, \alpha) \neq \emptyset$, then, $\alpha^* = u_{opt}$, where $u_{opt}$ is the optimal preference in the best scenario. Hence, $s \in UO(P)$. If $s \in UO(P)$, then $\text{pref}(Q, s) = u_{opt}$, hence in the best scenario $\text{pref}(bs(P), s) = u_{opt}$ and thus $s \in Pos(P, \alpha^*)$, where $\alpha^* = u_{opt}$.

Since by Proposition 6, $UO(P) \subseteq PO(P)$, then $Pos(P, \alpha^*) \subseteq PO(P)$. □

## Lower and upper lexicographically-optimal assignments

We now introduce two optimality notions that refine the lower and upper optimal notions.

**Definition 40** (Lower and upper lexicographically-optimal)**.** *Given an IVSCSP $P = \langle V, D, C, S\rangle$ and an assignment $s$ to the variables in $V$, $s$ is lower (resp., upper) lexicographically-optimal iff, for every other complete assignment $s'$, either $L(s) > L(s')$ (resp., $U(s) > U(s')$), or $L(s) = L(s')$ and $U(s) \geq U(s')$ (resp., $U(s) = U(s')$ and $L(s) \geq L(s')$).*

Lower (resp., upper) lexicographically-optimal assignments are those optimal assignments of the worst scenario (resp., best scenario) that are the best ones in the best scenario (resp., in the worst scenario). We denote with $LLO(P)$ (resp., $ULO(P)$) the set of the lower (resp., upper) lexicographically-optimal assignments of $P$. The IVSCSP $P$ of Figure 4.1 has $LLO(P) = ULO(P) = \{s_1\}$.

**Proposition 9.** *Given an IVSCSP $P$,*

- *$LLO(P) \subseteq LO(P)$ and so $LLO(P)$ is never empty;*
- *$ULO(P) \subseteq UO(P)$ and so $ULO(P)$ is never empty;*
- *$ID(P) \subseteq (LLO(P) \cap ULO(P)) = WID(P)$.*

**Proof:** We show that $LLO(P) \subseteq LO(P)$. The relation $ULO(P) \subseteq UO(P)$ can be shown similarly. If $s \in LLO(P)$, then, by definition of $LLO(P)$, $L(s) > L(s')$ or $(L(s) = L(s')$ and $U(s) \geq U(s'))$ for every other $s'$, hence $L(s) \geq L(s')$ for every other $s'$ and so $s \in LO(P)$.

Since $LLO(P)$ is contained in $LO(P)$ and, by Proposition 5, $LO(P)$ is never empty, then $LLO(P)$ is never empty. Similarly, it is possible to show that $ULO(P)$ is never empty.

We now prove that $(LLO(P) \cap ULO(P)) = WID(P)$. We first show that $(LLO(P) \cap ULO(P)) \subseteq WID(P)$. If $s \in (LLO(P) \cap ULO(P))$, then, by definition of $LLO(P)$, $L(s) \geq L(s')$ for every other $s'$ and, by definition of $ULO(P)$, $U(s) \geq U(s')$ for every other $s'$, hence $s \in WID(P)$. We now show that $WID(P) \subseteq (LLO(P) \cap ULO(P))$. If $s \in WID(P)$, then, by definition of $WID(P)$, $L(s) \geq L(s')$ and $U(s) \geq U(s')$ for every other $s'$. It could happen that ($L(s) > L(s')$ and $U(s) > U(s')$) or ($L(s) > L(s')$ and $U(s) = U(s')$) or ($L(s) = L(s')$ and $U(s) > U(s')$) or ($L(s) = L(s')$ and $U(s) = U(s')$) for every other $s'$. If $L(s) > L(s')$ and $U(s) > U(s')$ for every other $s'$, then $s \in LLO(P) \cap ULO(P)$ by the first part of the definitions of $LLO(P)$ and $ULO(P)$. If $L(s) > L(s')$ and $U(s) = U(s')$ for every other $s'$ , then $s \in LLO(P) \cap ULO(P)$ by the first part of the definition of $LLO(P)$ and by the second part of the definition of $ULO(P)$. If $L(s) = L(s')$ and $U(s) > U(s')$ for every other $s'$, then $s \in LLO(P) \cap ULO(P)$ by the second part of the definition of $LLO(P)$ and by the first part of the definition of $ULO(P)$. If $L(s) = L(s')$ and $U(s) = U(s')$ for every other $s'$, then $s \in LLO(P) \cap ULO(P)$ by the second part of the definitions of $LLO(P)$ and $ULO(P)$. □

Since lower and upper lexicographically-optimal solutions are refinements of lower and upper optimal solutions, they are possibly optimal solutions as well. However, the converse does not hold in general.

**Proposition 10.** *Given an IVSCSP $P$, $(LLO(P) \cup ULO(P)) \subseteq PO(P)$.*

**Proof:** We know, by Proposition 9, that $LLO(P) \subseteq LO(P)$ and $ULO(P) \subseteq UO(P)$. Since, by Proposition 6, $LO(P)$ and $UO(P)$ are contained $PO(P)$, then also $LLO(P)$ and $ULO(P)$ are contained in $PO(P)$. □

## Interval-optimal assignments

Until now we have considered optimality notions that are stronger than the possibly optimal notion. In the attempt to fully characterize possibly optimal solutions, we now consider an interval-based optimality notion that is weaker than the lower and upper optimality notions.

**Definition 41** (interval-optimal)**.** *Given an IVSCSP $P = \langle V, D, C, S\rangle$ and an assignment $s$ to the variables in $V$, $s$ is defined to be interval-optimal iff,*

*for every other complete assignment $s'$, $L(s) \geq L(s')$ or $U(s) \geq U(s')$.*

An interval-optimal assignment is a complete assignment with either a higher or equal lower bound, or a higher or equal upper bound, w.r.t. all other assignments. This means that, for every other complete assignment, it must be better than, or equal to it in either the worst or the best scenario. We denote with $IO(P)$ the set of the interval optimal assignments of $P$. The IVSCSP $P$ of Figure 4.1 has $IO(P) = \{s_1, s_2, s_4\}$.

**Proposition 11.** *Given an IVSCSP $P$, $(UO(P) \cup LO(P)) \subseteq IO(P)$ and so $IO(P)$ is never empty.*

**Proof:** Let $s$ be a complete assignment to the variables of $P$. Suppose that $s \in UO(P) \cup LO(P)$. There are two cases, (i) $s \in UO(P)$, and (ii) $s \in LO(P)$. Suppose (i) that $s \in UO(P)$. Then $U(s) \geq U(s')$ for every other complete assignment $s'$ and so $s \in IO(P)$. Similarly, (ii) if $s \in LO(P)$ then $L(s) \geq L(s')$ for every other $s'$, hence $s \in IO(P)$.

Since $(UO(P) \cup LO(P)) \subseteq IO(P)$ and, by Proposition 5, $LO(P)$ and $UO(P)$ are never empty, then $IO(P)$ is never empty. □

The interval-optimal solutions are possibly optimal solutions, but the converse does not hold in general, as shown in the following proposition. Therefore, also the interval-optimality notion is stronger than the possible optimality notion.

**Proposition 12.** *Given an IVSCSP $P$, if the c-semiring is strictly monotonic or idempotent, then $IO(P) \subseteq PO(P)$. Moreover, $PO(P) \not\subseteq IO(P)$.*

**Proof:** Let $s$ be a complete assignment to the variables of $P$.

Let us consider a strictly monotonic c-semiring. We know, by Theorem 19, that $s \in PO(P)$ iff $s \in Opt(Q^s)$, where $Q^s$ is the scenario where all the preferences of tuples in $s$ are set to their upper bound and all other tuples are associated to the lower bound of their preferences. We now show that, if $s \in IO$, then $s \in Opt(Q^s)$ and so, by Theorem 19, $s \in PO(P)$. Assume that $s \notin Opt(Q^s)$, we will show that $s \notin IO(P)$. If $s \notin Opt(Q^s)$, then there is a solution $s'$ such that $\text{pref}(Q^s, s') > \text{pref}(Q^s, s)$.

- If $s$ has no tuples in common with $s'$, then, by construction of $Q^s$, $\text{pref}(Q^s, s') = L(s')$ and $\text{pref}(Q^s, s) = U(s)$. Since $\text{pref}(Q^s, s') > \text{pref}(Q^s, s)$, and for every solution its lower bound is lower than or equal to its upper bound, then $U(s') \geq L(s') > U(s) \geq L(s)$ and so $U(s') > U(s)$ and $L(s') > L(s)$, that implies that $s \notin IO(P)$.

- If $s$ has some tuple in common with $s'$, then, $\text{pref}(Q^s, s') = \lambda \times u$, and $\text{pref}(Q^s, s) = \mu \times u$, where $\lambda$ (resp., $\mu$) is the combination of the preferences of the tuples that are in $s'$ but not in $s$ (resp., in $s$ but not in $s'$), and $u$ is the combination of the preferences of the tuples that are both in $s$ and in $s'$. By hypothesis, $\text{pref}(Q^s, s') > \text{pref}(Q^s, s)$, i.e., $\lambda \times u > \mu \times u$. By construction of $Q^s$, $U(s') \geq \lambda \times u > \mu \times u = U(s)$, and so $U(s') > U(s)$. Moreover, since the combination operator is monotonic, if $\lambda \times u > \mu \times u$, then $\lambda > \mu$. In fact, if $\lambda \leq \mu$, by monotonicity, $\lambda \times u \leq \mu \times u$. Let us denote with $u'$ (resp., $\mu'$) the combination of the lower bounds of the preferences of the tuples that are both in $s$ and in $s'$ (resp., in $s$ but not in $s'$). Then, by strict monotonicity and by construction of $Q^s$, $L(s') = \lambda \times u' > \mu \times u' \geq \mu' \times u' = L(s)$, and so $L(s') > L(s)$. Therefore, if $s$ has some tuple in common with $s'$, then $U(s') > U(s)$ and $L(s') > L(s)$, i.e., $s \notin IO(P)$.

Let us now consider an idempotent c-semiring. We want to show that if $s \in IO(P)$, then $s \in PO(P)$. We will show that, if $s \in IO(P)$, then $s \in Opt(Q^*)$, where $Q^*$ is the scenario such that all the preferences of the tuples of $s$ are set to $U(s)$, if $U(s)$ is contained in their preference interval, and to their upper bound, if $U(s)$ is not contained in their preference interval, and all other tuples are associated to the lower bound of their preferences. First, we show that $\text{pref}(Q^*, s) = U(s)$. Then, we show that $\text{pref}(Q^*, s) \geq \text{pref}(Q^*, s')$, for every other solution $s'$ that has no tuples in common with $s$ and for every solution $s'$ that has some tuple in common with $s$.

- $\text{pref}(Q^*, s) = U(s)$, by construction of $Q^*$, by Theorem 10 and by idempotency. In fact, by Theorem 10, $\text{pref}(Q^*, s) \leq U(s)$. Moreover, $\text{pref}(Q^*, s) \not< U(s)$. In fact, we now show that $\text{pref}(Q^*, s)$ is given by the combination of the preferences that are all greater than or equal to $U(s)$. By construction of $Q^*$ we have two results. (1) Every tuple of $s$ in $Q^*$ with preference interval that contains $U(s)$ is assigned to $U(s)$ and, by definition of $U(s)$ and by idempotency, there must be at least one of these preferences. (2) Every tuple with preference interval that does not contain $U(s)$ is assigned to its upper bound that must be a value greater than $U(s)$, since, by definition of $U(s)$, the upper bound of every tuple of $s$ must be greater than or equal to $U(s)$, otherwise the upper bound of $s$ is not $U(s)$ but a value lower than $U(s)$, that is a contradiction. Therefore, $\text{pref}(Q^*, s) \not< U(s)$ and so $\text{pref}(Q^*, s) = U(s)$.

- If $s$ has no tuples in common with $s'$, then, by construction of $Q^*$, $\text{pref}(Q^*, s') = L(s')$ and $\text{pref}(Q^*, s) = U(s)$. Since $s \in IO(P)$, then $L(s) \geq L(s')$ or $U(s) \geq U(s')$. If $L(s) \geq L(s')$, then $\text{pref}(Q^*, s) =$

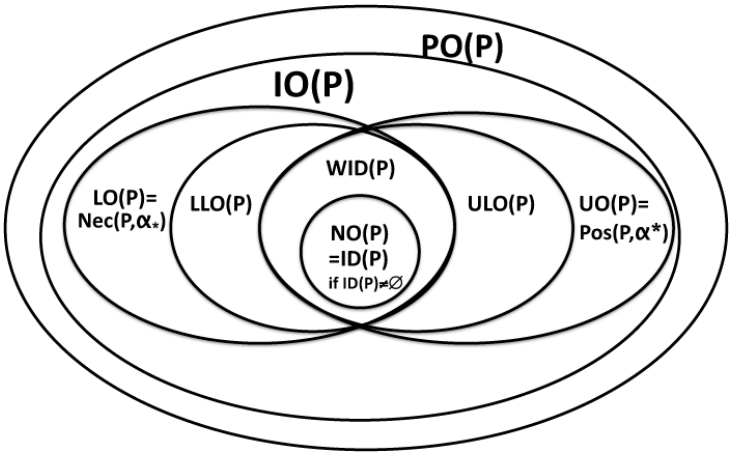



Figure 4.3: Relation among optimality sets.

$U(s) \geq L(s) \geq L(s') = \text{pref}(Q^*, s')$. If $U(s) \geq U(s')$, then $\text{pref}(Q^*, s) = U(s) \geq U(s') \geq L(s') = \text{pref}(Q^*, s')$.

- If $s$ has some tuple in common with $s'$, then, by construction of $Q^*$ $\text{pref}(Q^*, s') \leq U(s) = \text{pref}(Q^*, s)$.

Therefore, for every solution $s'$, $\text{pref}(Q^*, s') \leq U(s) = \text{pref}(Q^*, s)$. Hence, $s$ is optimal in $Q^*$ and so $s \in PO(P)$.

$PO(P) \not\subseteq IO(P)$. In fact, assume to have an IVSCSP over a fuzzy c-semiring, where there is only one variable $x$ with three values in its domain, say $x_1$, $x_2$, and $x_3$, with preference intervals respectively $[0.4, 0.6]$, $[0.5, 0.7]$, and $[0.5, 0.8]$. Then, $x_1 \notin IO(P)$, because $L(x_1) < L(x_2)$ and $U(x_1) < U(x_2)$. However, $x_1 \in PO(P)$, because $x_1$ is optimal in the scenario where we associate to $x_1$ the value 0.6 and to $x_2$ and $x_3$ the value 0.5.

□

## Summary of the various notions of optimality and of their relations

The various notions of optimality defined above are summarized in Table 4.1. For each notion, we refer to a solution $s$ and we describe compactly when $s$ belongs to each of the optimality sets.

The set-based relations between the various optimality notions are described in Figure 4.3.

Table 4.1: Optimality notions.

| *Optimality notions* | *Definition* |
|---|---|
| $NO(P)$ | $s \in Opt(Q)$, $\forall Q \in Sc(P)$ |
| $PO(P)$ | $s \in Opt(Q)$, $\exists Q \in Sc(P)$ |
| $Nec(P, \alpha)$ | $pref(Q, s) \geq \alpha$, $\forall Q \in Sc(P)$ |
| $Pos(P, \alpha)$ | $pref(Q, s) \geq \alpha$, $\exists Q \in Sc(P)$ |
| $ID(P)$ | $L(s) \geq U(s')$, $\forall s' \in Sol(P)$ |
| $WID(P)$ | $L(s) \geq L(s')$ and $U(s) \geq U(s')$, $\forall s' \in Sol(P)$ |
| $LO(P)$ | $L(s) \geq L(s')$, $\forall s' \in Sol(P)$ |
| $UO(P)$ | $U(s) \geq U(s')$, $\forall s' \in Sol(P)$ |
| $LLO(P)$ | $L(s) > L(s')$ or $(L(s) = L(s')$ and $U(s) \geq U(s'))$, $\forall s' \in Sol(P)$ |
| $ULO(P)$ | $U(s) > U(s')$ or $(U(s) = U(s')$ and $L(s) \geq L(s'))$, $\forall s' \in Sol(P)$ |
| $IO(P)$ | $L(s) \geq L(s')$ or $U(s) \geq U(s')$, $\forall s' \in Sol(P)$ |

## An example: imprecise meeting scheduling problems

To better explain how to use the various optimality notions introduced in the previous sections, we consider an example of a class of problems, related to meeting scheduling. The meeting scheduling problem (MSP) is a benchmark for CSPs [65] we previously adapted to allow for missing preferences in Chapter 3. Here we adapt it to allow for preference intervals. In the following we recall what are MSPs and how such problems can be modeled in constraint programming. Then, we present our generalization to allow for imprecise preferences.

A meeting scheduling problem can be described by

- a set of agents;
- a set of meetings, each with a location and a duration;
- a set of time slots where meetings can take place;
- for each meeting, a subset of agents that are supposed to attend such a meeting;
- for each pair of locations, the time to go from one location to the other one.

Typical simplifying assumptions concern having the same duration for all meetings (one time slot), and the same number of meeting for each agent. A solution of a meeting scheduling problem is an allocation of each meeting in a time slot in a way that each agent can participate in his meetings. The only way that an agent cannot participate has to do with the time needed to go from the location of a meeting to the location of his next meeting.

A CSP model of a MSP involves variables representing meetings and variable domains representing all time slots. Each constraint between two meetings models the fact that one or more agents must participate in both meetings, and it is satisfied by all pairs of time slots that allow the participation to both meetings according to the time needed to travel between the corresponding locations. For this reason, it is often used as a typical benchmark for CSPs.

For our purposes, we consider a generalization of the MSP, called IVMSP, where there is a chair, who is in charge of the meeting scheduling, and who declares his preferences over the variable domains and over the compatible pairs of time slots in the binary constraints. The preferences over the variable domains can model the fact that the chair prefers some time slots to others for a certain meeting. On the other hand, the preferences in the binary constraints can model a preference for certain feasible pairs of time slots, over others, for the two meetings involved in the constraint.

Such preferences can be exact values when the chair works with complete information. However, at the time the meeting scheduling has to be done, it may be that some information, useful for deciding the preferences, is still missing. For example, the chair could have invited agents to meetings, but he does not yet know who will accept his invitations. As other examples, weather considerations or the presence of other events in the same time slots may affect the preferences. Because of this uncertainty, some preferences may be expressed by using an interval of values, which includes all preference values that are associated to all possible outcomes of the uncertain events.

Since MSPs can be expressed as CSPs, it is thus clear that IVMSPs can be expressed as IVSCSPs. The problem of solving an IVMSP concerns finding time slots for the meetings such that all agents can participate and, among all possible solutions, to choose an optimal one according to some optimality criteria. We will now consider several of the optimality notions defined above and describe their use in this class of problems.

In this context, given an IVMSP $P$, necessarily optimal solutions (i.e., solutions in $NO(P)$) are meeting schedulings that are optimal no matter how the uncertainty is resolved. Thus, if there is at least one of such solutions, this is certainly preferred to any other. By working with the optimality notions defined over intervals, to find a solution in $NO(P)$, we may try to

find a solution in $ID(P)$, given that solutions in $ID(P)$, if any, coincide with solutions in $NO(P)$. Otherwise, if $ID(P)$ is empty, and given that $NO(P)$ is included in $WID(P)$, we may look for a solution in $WID(P)$. We recall that solutions in $ID(P)$ are meeting schedulings where the preference interval of the optimal solution is above the preference intervals of all other solutions, while solutions in $WID(P)$ have the upper bound of their preference interval above the upper bounds of the preference intervals of all other solutions, and the same for the lower bound.

Solutions in $Nec(P, \alpha_*)$ are also attractive, because they guarantee a preference level of $\alpha_*$ in all scenarios. Since $LO = Nec(P, \alpha_*)$, we may find a solution in $LO(P)$, that is, a solution which is optimal in the worst scenario. This solution will guarantee the chair against the uncertainty of the problem by assuring a certain level of overall preference. This notion can be useful if the chair is pessimistic, because such solutions provide a preference guarantee over all scenarios. However, such a guaranteed preference level may be very low.

If instead the chair is optimistic, he may ask for a solution in $Pos(P, \alpha^*)$, that is, a solution with the highest preference level in some scenario. Since $UO(P) = Pos(P, \alpha_*)$, we may find a solution in $UO(P)$, that is, a solution which is optimal in the best scenario.

When looking for solutions in $LO(P)$ and $UO(P)$, we may want to be as close as possible to solutions in $NO(P)$, as $NO(P)$ is included in $LO(P)$ and $UO(P)$. To do this, we can try to find solutions in $LLO(P)$ or $ULO(P)$, respectively. For example, solutions in $LLO(P)$ are solutions in $LO(P)$ that have the highest upper bound of their preference interval. This means that, depending on how the uncertainty is resolved, they give more hope of achieving a higher level of preference.

## 4.5 Finding and testing optimal assignments

In this section we analyze how to determine if a complete assignment is one of the different kinds of optimal assignments previously defined in Section 4.4, and how to find such optimal assignments. These results will be useful to find and test possibly and necessarily optimal solutions.

### Lower and upper optimal assignments

It is easy to show that, by following directly the definitions of lower and upper optimal assignments, the lower (resp., upper) optimal solutions coincide with the optimal elements of the worst (resp., best) scenario.

**Theorem 11.** *Given an IVSCSP $P$, $LO(P) = Opt(ws(P))$ and $UO(P) = Opt(bs(P))$.*

**Proof:** We show that $LO(P) = Opt(ws(P))$. Let $s$ be a solution of $P$. If $s \in LO(P)$, then $L(s) \geq L(s')$ for every other solution $s'$, hence if we consider $ws(P)$, i.e., the worst scenario of $P$, that is the scenario where we fix all the preference intervals to their lower bound, then $\text{pref}(ws(P), s) = L(s)$ and so $\text{pref}(ws(P), s) \geq \text{pref}(ws(P), s')$ for every other solution $s'$, hence $s \in Opt(ws(P))$. If $s \in Opt(ws(P))$, then $\text{pref}(ws(P), s) \geq \text{pref}(ws(P), s')$ for every other solution $s'$ of $P$, that is, by definition of worst scenario, $L(s) \geq L(s')$ for every $s'$ and so $s \in LO(P)$. Similarly, it is possible to show that $UO(P) = Opt(bs(P))$. □

A lower-optimal solution is a complete assignment whose lower bound is greater than or equal to the lower bound of every other complete assignment. Thus, it is a complete assignment that is better than or equal to all other assignments in the scenario obtained by replacing every interval with its lower bound, i.e., the worst scenario.

Thus, finding a lower-optimal (resp. upper-optimal) solution is as complex as solving an SCSP. This holds also for testing if an assignment $s$ is in $LO(P)$ (resp. in $UO(P)$), since it is enough to solve the SCSP representing the worst or the best scenario and to check if the preference of the optimal solution coincides with $L(s)$ (resp. $U(s)$).

## Interval optimal assignments

To find an interval optimal assignment, it is sufficient to find a lower-optimal solution or an upper-optimal solution, because $(UO(P) \cup LO(P)) \subseteq IO(P)$, and neither $UO(P)$ nor $LO(P)$ can be empty. Thus, finding assignments of $IO(P)$ can be achieved by solving an SCSP.

To test if a solution is interval optimal, if the c-semiring is idempotent, we can exploit the preference levels of the best and worst scenarios, as stated by the following theorem.

**Theorem 12.** *Given an IVSCSP $P$ defined over an idempotent c-semiring, and an assignment $s$, we have $s \in IO(P)$ iff the CSP obtained by joining*[1] *$scut_{L(s)}(ws(P))$ and $scut_{U(s)}(bs(P))$ has no solution.*

**Proof:** Let us denote with $Q$ the CSP defined in the theorem. We first show that, if $Q$ has no solution, then $s \in IO(P)$. Suppose that $s \notin IO(P)$. Then

[1]The join of two CSPs $P_1$ and $P_2$ is the CSP whose set of variables (resp., constraints) is given by the union of the sets of variables (resp., constraints) of $P_1$ and $P_2$.

there exists some complete assignment $s'$ with $L(s') > L(s)$ and $U(s') > U(s)$. Then $pref(ws(P), s') = L(s') > L(s)$ and $pref(bs(P), s') = U(s') > U(s)$, so $s'$ is a solution of $Q$. We now show that, if $s \in IO(P)$, then $Q$ has no solution. If $Q$ has a solution, say $s^*$, then, by definition of $Q$, $L(s^*) > L(s)$ and $U(s^*) > U(s)$, and so $s \notin IO(P)$. □

In fact, all and only the solutions of such a CSP strictly dominate $s$ with respect to both the lower and the upper bound. Thus, testing membership in $IO(P)$ when the semiring is idempotent amounts to solving a CSP.

More generally (that is, even if the combination operator is not idempotent), we can test interval optimality by checking if a suitably defined SCSP has solutions with preference above certain threshold.

**Theorem 13.** *Given an IVSCSP $P$ and an assignment $s$, let $l_{opt}$ and $u_{opt}$ be the optimal preferences of the worst and best scenario. Then, $s \in IO(P)$ iff at least one of the following conditions holds: (1) $L(s) = l_{opt}$; (2) $U(s) = u_{opt}$; (3) the SCSP $Q$ with the same variables, domains, and constraint topology as $P$, defined on the c-semiring $\langle (A \times A), (+,+), (\times,\times), (\mathbf{0},\mathbf{0}), (\mathbf{1},\mathbf{1})\rangle$, where the preference of each tuple in each constraint is set to the pair containing the lower and upper bound of its interval in $P$, has no solution $s'$ with preference pair $(L(s'), U(s'))$ pointwise greater than $(L(s), U(s))$, i.e., such that $L(s') > L(s)$ and $U(s') > U(s)$.*

**Proof:** We first show that if $L(s) = l_{opt}$, $U(s) = u_{opt}$, or $Q$ has no solution with preference greater than $(L(s), U(s))$, then $s \in IO(P)$. If $L(s) = l_{opt}$ (resp., $U(s) = u_{opt}$), then $L(s) \geq L(s')$ (resp., $U(s) > U(s')$) for every other solution $s'$, hence $s \in LO(P)$ (resp., $s \in UO(P)$) and so, since $LO(P) \cup UO(P) \subseteq IO(P)$, $s \in IO(P)$. If $Q$ has no solution with preference greater than $(L(s), U(s))$, then $s \in IO(P)$. In fact, if $s \notin IO(P)$, then there is a solution, say $s^*$, such that $L(s^*) > L(s)$ and $U(s^*) > U(s)$, and so $Q$ has a solution with preference greater than $(L(s), U(s))$.

We now show, that if $s \in IO(P)$, then $L(s) = l_{opt}$, $U(s) = u_{opt}$, or $Q$ has no solution with preference greater than $(L(s), U(s))$. If $L(s) \neq l_{opt}$, $U(s) \neq u_{opt}$ and $Q$ has a solution $s^*$ with preference greater than $(L(s), U(s))$, then, by definition of $Q$, the preference of $(L(s^*), U(s^*))$ is greater than the preference of $(L(s), U(s))$, hence $L(s^*) > L(s)$ and $U(s^*) > U(s)$ and so $s \notin IO(P)$. □

The first two conditions simply check if $s$ is either lower or upper optimal. The second condition is satisfied when there is no solution better than $s$ on both bounds. Notice that this can be checked for example by running branch and bound on $Q$ with a strict bound equal to $(L(s), U(s))$. Therefore, testing

membership in $IO(P)$ with any c-semiring can be achieved by solving at most three SCSPs.

## Lower and upper lexicographically optimal assignments

To find the lower-lexicographically optimal solutions of an IVSCSP $P$ we consider the optimal solutions of a suitable SCSP, as described by the following theorem.

**Theorem 14.** *Given an IVSCSP $P$ over a strictly monotonic c-semiring $S$, let us consider the SCSP $Q$ with the same variables, domains, and constraint topology as $P$, and defined over the c-semiring $\langle A \times A, \max_{lex}, (\times, \times), (\mathbf{0}, \mathbf{0}), (\mathbf{1}, \mathbf{1})\rangle$. The binary operation $\max_{lex}$ is defined to be the maximum with respect to the ordering $\succeq_{lex}$ defined as follows: for each $(a, a'), (b, b') \in (A \times A)$, $(a, a') \succeq_{lex} (b, b')$ iff $a >_S b$ or $a = b$ and $a' \geq_S b'$. For each tuple in each constraint of $Q$, its preference is set to the pair containing the lower and upper bound of its interval in $P$. Then, $LLO(P) = Opt(Q)$.*

**Proof:** We first show that $LLO(P) \subseteq Opt(Q)$. If $s \in LLO(P)$, then $s \in Opt(Q)$. In fact, if $s \notin Opt(Q)$, then, there is a solution, say $s'$, of $Q$ such that $\text{pref}(Q, s') > \text{pref}(Q, s)$, that is, by definition of preference given in the theorem, $(L(s'), U(s')) \succ_{lex} (L(s), U(s))$, that is, by definition of $\succ_{lex}$, either $L(s') > L(s)$ or $(L(s') = L(s)$ and $U(s') > U(s))$, and so $s \notin LLO(P)$.

We now show that $Opt(Q) \subseteq LLO(P)$. If $s \in Opt(Q)$, then $\text{pref}(Q, s') \geq \text{pref}(Q, s)$, for every $s'$, that is, $(L(s'), U(s')) \succeq_{lex} (L(s), U(s))$, for every other $s'$, that is, for every other $s'$, either $L(s') > L(s)$ or $(L(s') = L(s)$ and $U(s') \geq U(s))$, and so $s \in LLO(P)$.

Note that the assumption of strict monotonicity of $S$ guarantees that the structure defined in the theorem $\langle A \times A, \max_{lex}, (\times, \times), (\mathbf{0}, \mathbf{0}), (\mathbf{1}, \mathbf{1})\rangle$ is a c-semiring. If we don't make this assumption, then distributivity property does not hold and so the structure above is not a c-semiring. □

In words, the first component of the pairs in the semiring of Theorem 14 is the most important, and the second one is used to break ties. To find the upper-lexicographically optimal solutions, it is sufficient to consider the same SCSP as defined above except for the ordering which considers the second component as the most important. Thus, finding assignments in $LLO(P)$ and $ULO(P)$ can be achieved by solving one SCSP.

To test if a solution $s$ is in $LLO(P)$, it is enough to find the preference pair, say $(p1, p2)$, of an optimal solution of the SCSP defined above and to check if $(L(s), U(s)) = (p1, p2)$. Similarly to test if a solution is in $ULO(P)$.

## Weakly interval dominant assignments

We know that $WID(P) = LO(P) \cap UO(P)$. Thus a straightforward, but costly, way to find a solution in $WID(P)$ is to compute all the optimal solutions of the best and the worst scenario and to check if there is a solution in the intersection of the two sets. However, if the c-semiring is idempotent, this is not necessary, as shown by the following theorem.

**Theorem 15.** *Given an IVSCSP $P$ defined over an idempotent c-semiring, and $l_{opt}$ and $u_{opt}$ as defined above, an assignment $s$ is in $WID(P)$ iff it is a solution of the CSP obtained by joining $cut_{l_{opt}}(ws(P))$ and $cut_{u_{opt}}(bs(P))$.*

**Proof:** Let us denote with $Q$ the CSP described in the theorem. We first show that, if $s$ is a solution of $Q$, then $s \in WID(P)$. If $s$ is a solution of $Q$, then, by definition of $Q$, $s$ is a solution of the CSP $cut_{l_{opt}}(ws(P))$ obtained from the worst scenario by allowing only the tuples with preference greater than or equal to $l_{opt}$, hence, by definition of $l_{opt}$, $L(s) \geq L(s')$ for every other solution $s'$. Moreover, by definition of $Q$, $s$ is also a solution of the CSP $cut_{u_{opt}}(bs(P))$ obtained from the best scenario by allowing only the tuples with preferences greater than or equal to $u_{opt}$. Hence, by the definition of $u_{opt}$, $U(s) \geq U(s')$, for every other $s'$. Therefore, if $s$ is a solution of $Q$, then $L(s) \geq L(s')$ and $U(s) \geq U(s')$ for every other $s'$, and so $s \in WID(P)$.

We now show that, if $s \in WID(P)$, then $s$ is a solution of $Q$. If $s$ is not a solution of $Q$, then $L(s) < l_{opt}$ or $U(s) < u_{opt}$. If $L(s) < l_{opt}$ (resp., $U(s) < u_{opt}$), then $L(s) < L(s')$ (resp., $U(s) < U(s')$) for any solution $s'$ such that $\text{pref}(ws(P), s') = l_{opt}$ (resp., $\text{pref}(bs(P), s') = u_{opt}$). Therefore, $s \notin WID(P)$. □

In words, any solution of the join CSP is optimal both in the worst and in the best scenario and this implies that it is undominated on both bounds. Thus, if the c-semiring is idempotent, finding a weakly interval dominant solution amounts to solving two SCSPs and one CSP. Moreover, to test whether a solution $s$ is in $WID(P)$, it is sufficient to check if $L(s) = l_{opt}$ and $U(s) = u_{opt}$, which amounts to solving two SCSPs.

## Interval dominant assignments

To find an assignment in $ID(P)$, we can use Proposition 5. Thus, if $l_{opt} = u_{opt}$, then it is sufficient to find a lower-optimal solution. If instead $l_{opt} < u_{opt}$ then, if $|UO(P)| \geq 2$, then we know that $ID(P) = \emptyset$. Moreover, if $|UO(P)| = 1$ (let us call $s$ this single solution), if $L(s) \neq l_{opt}$ then we know that $ID(P) = \emptyset$.

If the c-semiring is idempotent, cuts can be exploited in the same style as above, to build a suitably defined CSP, leading to a sound and complete procedure to find an assignment, if any, in $ID(P)$.

**Theorem 16.** *Given an IVSCSP $P$ over an idempotent c-semiring, and $l_{opt}$ as defined above, if $scut_{l_{opt}}(bs(P))$ has no solution, then $ID(P) = LO(P)$. If $scut_{l_{opt}}(bs(P))$ has one solution, say $s$, and $L(s) = l_{opt}$, then this solution is the only one in $ID(P)$. Otherwise, $ID(P) = \emptyset$.*

**Proof:** Let us denote with $Q$ the CSP $scut_{l_{opt}}(bs(P))$. We first show that if $Q$ has no solution, then $ID(P) = LO(P)$. If $Q$ has no solution, then, since $Q$ is the CSP obtained by the best scenario by allowing only tuples with preference greater than $l_{opt}$, there is no solution with upper bound greater than $l_{opt}$, that is, for all the solutions $s'$ of $P$, $l_{opt} \geq U(s')$. To show that $ID(P) = LO(P)$ it is sufficient to show that $LO(P) \subseteq ID(P)$, since Theorem 2 implies that $ID(P) \subseteq LO(P)$. Let $s$ be a solution of $P$. If $s \in LO(P)$, then $L(s) = l_{opt}$ and thus, by the reasoning above, $L(s) \geq U(s')$ for every other $s'$, hence $s \in ID(P)$.

If $Q$ has a solution, say $s$, then $U(s) > l_{opt} \geq L(s')$ for all solutions $s'$, and so $ID(P)$ is either empty or equal to $\{s\}$. Therefore if $Q$ has more than one solution then $ID(P)$ is empty. Suppose that $Q$ has exactly one solution, $s$. If $L(s) < l_{opt}$ then $L(s) < L(s')$ for any solution $s'$ with $L(s') = l_{opt}$, and so $L(s) < U(s')$, which implies that $s \notin ID(P)$ and so $ID(P) = \emptyset$. If $L(s) = l_{opt}$ then for any other solution $s'$ we have $U(s') \leq l_{opt}$ (since $Q$ has only one solution), and so $L(s) \geq U(s')$ which implies that $s \in ID(P)$ and so $ID(P) = \{s\}$. □

Performing a strict cut of the best scenario at the optimal level of the worst scenario means isolating solutions that have an upper bound higher than $l_{opt}$. If there is no such solution, then the upper bound of the lower-optimal solutions must coincide with their lower bound ($l_{opt}$). Thus, lower-optimal solutions coincide with interval dominant solutions. If, instead, such a CSP has only one solution, all other solutions must have an upper bound which is at most $l_{opt}$. This means that, if this solution is also lower-optimal, then it is the only interval dominant solution. Finally, if there is more than one solution with an upper bound above $l_{opt}$, then there cannot be any solution whose lower bound dominates the upper bound of all others and, thus, $ID(P)$ is empty.

Summarizing, when the c-semiring is idempotent, to find a solution in $ID(P)$ we need to solve an SCSP and then one CSP. Proposition 5 and Theorem 16 can also be used to test if a solution is interval dominant.

## 4.6 Finding and testing necessarily and possibly optimals

We will now show how to test if an assignment is possibly or necessarily optimal (or of at least preference $\alpha$) and how to find these kinds of assignments. To do that, we will exploit the relation between possibly and necessarily optimal assignments and the various kinds of interval-based optimal assignments, shown in Section 4.4.

### Necessarily optimal solutions

To find a necessarily optimal solution, we exploit the results shown in Propositions 2 and 4 (i.e., if $ID(P) \neq \emptyset$ then $NO(P) = ID(P)$, and $ID(P) \subseteq NO(P) \subseteq WID(P)$), and thus we perform the following steps:

1. If $ID(P) \neq \emptyset$, then return $s \in ID(P)$;
2. If $WID(P) = \emptyset$, then $NO(P) = \emptyset$;
3. Otherwise, return the first solution in $WID(P)$ that is necessarily optimal. If none, $NO(P) = \emptyset$

Testing if a solution is necessarily optimal when $ID(P) \neq \emptyset$ coincides with testing if it is in $ID(P)$. Otherwise, we need to test if it is an optimal solution of some suitably defined SCSPs, as shown by the following theorem.

**Theorem 17.** *Consider an IVSCSP $P$ and an assignments $s$. Let $Q_s$ (resp., $Q^s$) be the scenario where every preference associated to a tuple of $s$ is set to its lower bound (resp., upper bound) and the preferences of all other tuples are set to their upper bound (resp., lower bound). The following results hold:*

- *If $s \in NO(P)$, then $s \in Opt(Q_s)$. Moreover, if the c-semiring is strictly monotonic, the converse holds as well: $s \in NO(P) \iff s \in Opt(Q_s)$.*
- *If $s \in NO(P)$ then, for every $s'$, $s \in Opt(Q^{s'})$. If the c-semiring is idempotent, the converse holds as well: $s \in NO(P) \iff$ for every $s'$, $s \in Opt(Q^{s'})$.*

**Proof:** We first show that, if $s \in NO(P)$, then $s \in Opt(Q_s)$. If $s \in NO(P)$, then it is optimal in all scenarios and so also in $Q_s$.

We now show that, if the c-semiring is strictly monotonic and if $s \in Opt(Q_s)$, then $s \in NO(P)$. If $s \in Opt(Q_s)$, then $\text{pref}(Q_s, s) \geq \text{pref}(Q_s, s')$

for every other solution $s'$. For every other $s'$, let $\lambda$ (resp., $\mu$) be the combination of the preference values of tuples associated to $s$ but not to $s'$ (resp., associated to $s'$ but not to $s$) in $Q_s$, and let $u$ be the combination of the preference values of tuples associated to both $s$ and $s'$ in $Q_s$. Since, for every $s'$, $\text{pref}(Q_s, s) \geq \text{pref}(Q_s, s')$, then for every $s'$, $\lambda \times u \geq \mu \times u$ that implies that $\lambda \geq \mu$. In fact, if $\lambda < \mu$, then, by strict monotonicity of $\times$, then $\lambda \times u < \mu \times u$. For every scenario $Q_i$, for every $s'$, let $\lambda_i$ (resp., $\mu_i$) be the combination of the preference values of tuples associated to $s'$ but not to $s$ (resp., associated to $s'$ but not to $s$) in $Q_i$ and let $u_i$ be the combination of the preference values of tuples associated to both $s$ and $s'$ in $Q_i$. Since $Q_s$ is the least favorable scenario for $s$, then for every scenario $Q_i$, $\lambda_i \times u \geq \lambda \times u$ that implies $\lambda_i \geq \lambda$. In fact, if $\lambda_i < \lambda$, then, by strict monotonicity, $\lambda_i \times u < \lambda \times u$. Since $Q_s$ is the most favorable scenario for the tuples in $s'$ but not in $s$, then $\mu \geq \mu_i$ for every scenario $Q_i$. Therefore, for every scenario $Q_i$, for every $s'$, we have that $\lambda \geq \mu$, $\lambda_i \geq \lambda$ and $\mu \geq \mu_i$, hence, by monotonicity, $\text{pref}(Q_i, s) = \lambda_i \times u_i \geq \lambda \times u_i \geq \mu \times u_i \geq \mu_i \times u_i = \text{pref}(Q_i, s')$, hence $s$ is optimal in every scenario and so $s \in NO(P)$.

If $s \in NO(P)$, then $s$ is optimal in all the scenarios and so, for every $s'$, $s$ is optimal in $Q^{s'}$. If the c-semiring is idempotent and, for every $s'$, $s \in Opt(Q^{s'})$, then $s \in NO(P)$. In fact, assume that $s \notin NO(P)$, then there is a scenario $Q$, where $s$ is not optimal, i.e., there is $s'$ such that $\text{pref}(Q, s) < \text{pref}(Q, s')$. We want to show that this holds also in the scenario $Q^{s'}$. If we consider the scenario $Q_1$ obtained from $Q$ by putting the preference value of any tuple that is in $s$ but not in $s'$ to its lower bound, then, the preference of $s$ decreases or remains the same, by monotonicity, and the preference of $s'$ does not change. Hence, $\text{pref}(Q_1, s) \leq \text{pref}(Q, s) < \text{pref}(Q, s') = \text{pref}(Q_1, s')$, and so $\text{pref}(Q_1, s) < \text{pref}(Q_1, s')$. If we consider the scenario $Q_2$ obtained from $Q_1$ by setting the preference value of any tuple that is in $s'$ but not in $s$ to its upper bound, then the preference of $s'$ increases or remains the same, by monotonicity, and the preference of $s$ does not change. Hence, $\text{pref}(Q_2, s) = \text{pref}(Q_1, s) < \text{pref}(Q_1, s') \leq \text{pref}(Q_2, s')$ and so $\text{pref}(Q_2, s) < \text{pref}(Q_2, s')$. If we consider the scenario obtained from $Q_2$ by setting the preference value of the tuples that are in $s$ and $s'$ to their upper bound, then we have the scenario $Q^{s'}$. The preferences of the tuples that are in $s$ and $s'$ does not modify $\text{pref}(Q_2, s)$ and $\text{pref}(Q_2, s')$. In fact, since the c-semiring is idempotent, then $\text{pref}(Q_2, s)$ (resp., $\text{pref}(Q_2, s')$) is given by the tuple with the worst preference of $s$ (resp., $s'$), and, since $\text{pref}(Q_2, s) < \text{pref}(Q_2, s')$, $\text{pref}(Q_2, s)$ and $\text{pref}(Q_2, s')$ must be given by different tuples, otherwise $\text{pref}(Q_2, s) = \text{pref}(Q_2, s')$. Hence, $\text{pref}(Q^{s'}, s) = \text{pref}(Q_2, s) < \text{pref}(Q_2, s') = \text{pref}(Q^{s'}, s)$. Therefore, there is a solution $s'$ such $s' \notin Opt(Q^{s'})$. □

The intuition behind this theorem is that, in order for a solution to be necessarily optimal, it must be optimal also in its least favorable scenario, when the c-semiring is strictly monotonic, and it must be optimal in the most favorable scenario of every other solution, when the c-semiring is idempotent.

## Necessarily above threshold $\alpha$

By Proposition 7, we know that $s \in Nec(P, \alpha)$ if and only if $\alpha \leq L(s)$. Thus, testing whether a solution $s$ is in $Nec(P, \alpha)$ amounts at checking this condition that takes linear time.

To find a solution in $Nec(P, \alpha)$, we know, by Proposition 7, that $Nec(P, \alpha)$ is not empty only if $\alpha$ is at most the optimal preference of the worst scenario, and in such a case any lower-optimal solution is in $Nec(P, \alpha)$. This amounts to solving one SCSP. However, if the c-semiring is idempotent, it is sufficient to solve one CSP, as shown by the following theorem.

**Theorem 18.** *Given an IVSCSP $P$, if the c-semiring is idempotent, then $Nec(P, \alpha)$ coincides with the set of solutions of $cut_\alpha(ws(P))$.*

**Proof:** Let us denote with $SL$ the set of the solutions of $cut_\alpha(ws(P))$. We first show that $Nec(P, \alpha) \supseteq SL$ and then we show that $Nec(P, \alpha) \subseteq SL$. Let be $s$ a solution of $P$. If $s \in SL$, then, since $cut_\alpha(ws(P))$ is the CSP obtained from the worst scenario of $P$ by allowing only tuples with preference greater than or equal to $\alpha$, $\text{pref}(ws(P), s) \geq \alpha$, by idempotence. Since $ws(P)$ is the worst scenario of $P$, then $\text{pref}(Q_i, s) \geq \text{pref}(ws(P), s) \geq \alpha$ for every scenario $Q_i$ and so $s \in Nec(P, \alpha)$. Therefore, $Nec(P, \alpha) \supseteq SL$. If $s \in Nec(P, \alpha)$, then $\text{pref}(Q_i, s) \geq \alpha$ for every scenario $Q_i$ and so also for the worst scenario. Hence, $\text{pref}(ws(P), s) \geq \alpha$ and so, by definition of $cut_\alpha(ws(P))$, $s \in SL$. Therefore, $Nec(P, \alpha) \subseteq SL$. □

By Proposition 7, we know that $Nec(P, \alpha_*) = LO(P)$. Therefore, to find a solution in $Nec(P, \alpha_*)$, it is sufficient to find a solution of the worst scenario, and thus to solve one SCSP.

## Possibly optimal solutions

To find a solution in $PO(P)$, we can observe that $LO(P)$, $UO(P)$, $LLO(P)$, and $ULO(P)$ are all contained in $PO(P)$ (Propositions 6 and 10) and they are never empty (Propositions 5 and 9).

To test if a solution is in $PO(P)$, it is sufficient to test if $s$ is optimal in one of the two scenarios defined in the following theorem. This amounts to solving an SCSP.

**Theorem 19.** *Given an IVSCSP $P$ and an assignment $s$ to the variables of $P$, let $Q^s$ be the scenario where all the preferences of tuples in $s$ are set to their upper bound and all other tuples are associated to the lower bound of their preferences, and let $Q^*$ be the scenario where all the preferences of the tuples of $s$ are set to $U(s)$, if $U(s)$ is contained in their preference interval, and to their upper bound otherwise, and all other tuples are associated to the lower bound of their preferences. Then,*

- *if the c-semiring is strictly monotonic, $s \in PO(P) \iff s \in Opt(Q^s)$;*
- *if the c-semiring is idempotent, $s \in PO(P) \iff s \in Opt(Q^*)$.*

**Proof:** We first show that, if $s \in Opt(Q^s)$, then $s \in PO(P)$. If $s \in Opt(Q^s)$, then $s$ is optimal in the scenario $Q^s$, and so $s \in PO(P)$. We now show that, if $s \in PO(P)$ then $s \in Opt(Q^s)$. If $s \in PO(P)$, then there is a scenario, say $Q_i$, where $s$ is optimal, that is, $\text{pref}(Q_i, s) \geq \text{pref}(Q_i, s')$, for every other solution $s'$. Assume to use the same notations used in the proof of Theorem 17. Using these notations, since $\text{pref}(Q_i, s) \geq \text{pref}(Q_i, s')$, for every other solution $s'$, then, for every other s', $\lambda_i \times u_i \geq \mu_i \times u_i$ in the scenario $Q_i$. This implies that, for every other s', $\lambda_i \geq \mu_i$. In fact, if $\lambda_i < \mu_i$, then, by strict monotonicity, $\lambda_i \times u_i < \mu_i \times u_i$. Since $Q^s$ is the most favorable scenario for $s$, then for every scenario and so also for the scenario $Q_i$, by monotonicity, $\lambda \times u \geq \lambda \times u_i \geq \lambda_i \times u_i$, that implies $\lambda \geq \lambda_i$. In fact, if $\lambda < \lambda_i$, then, by strict monotonicity, $\lambda \times u_i < \lambda_i \times u$. Since $Q_s$ is the least favorable scenario for the tuples in $s'$ but not in $s$, then $\mu_i \geq \mu$ for every scenario and so also for $Q_i$. Hence, since for every $s'$, $\lambda \geq \lambda_i$, $\lambda_i \geq \mu_i$, and $\mu_i \geq \mu$, then, by monotonicity, for every $s'$, $\text{pref}(Q^s, s) = \lambda \times u \geq \lambda_i \times u \geq \mu_i \times u \geq \mu \times u = \text{pref}(Q^s, s')$, hence $s$ is optimal in the scenario $Q^s$.

If $s \in Opt(Q^*)$, then $s \in PO(P)$. We now show that, if $s \in PO(P)$, then $s \in Opt(Q^*)$. If $s \notin Opt(Q^*)$, then there is a solution $s'$ such that $\text{pref}(Q^*, s') > \text{pref}(Q^*, s)$. By construction of $Q^*$, by Theorem 10 and by idempotency, we have that $\text{pref}(Q^*, s) = U(s)$. In fact, by Theorem 10, $\text{pref}(Q^*, s) \leq U(s)$. Moreover, $\text{pref}(Q^*, s) \not< U(s)$. In fact, we now show that $\text{pref}(Q^*, s)$ is given by the combination of the preferences that are all greater than or equal to $U(s)$. By construction of $Q^*$ we have two results. (1) Every tuple of $s$ in $Q^*$ with preference interval that contains $U(s)$ is assigned to $U(s)$ and, by definition of $U(s)$ and by idempotency, there must be at least one of these preferences. (2) Every tuple with preference interval that does not contain $U(s)$ is assigned to its upper bound that must be a value greater than $U(s)$, since, by definition of $U(s)$, the upper bound of every tuple of $s$ must be greater than or equal to $U(s)$, otherwise the upper

bound of $s$ is not $U(s)$ but a value lower than $U(s)$, that is a contradiction. Therefore, $\text{pref}(Q^*, s) \not< U(s)$ and so $\text{pref}(Q^*, s) = U(s)$. If $s$ and $s'$ have tuples in common, by construction of $Q^*$, $\text{pref}(Q^*, s') \leq U(s)$. In such a case, since we have shown above that $\text{pref}(Q^*, s) = U(s)$, and since we are assuming that there is a solution $s'$ such that $\text{pref}(Q^*, s') > \text{pref}(Q^*, s)$, then $U(s) \geq \text{pref}(Q^*, s') > \text{pref}(Q^*, s) = U(s)$, and so we have a contradiction. If $s$ and $s'$ have no tuples in common, then, for every scenario $Q$, $\text{pref}(Q, s') \geq L(s') = \text{pref}(Q^*, s') > \text{pref}(Q^*, s) = U(s) \geq \text{pref}(Q, s)$, and so $s \notin PO(P)$. □

In Theorem 19 we have characterized possibly optimal solutions for IVSCSPs with idempotent c-semiring and for IVCSPs with strictly monotonic c-semiring. The characterization of possibly optimal solutions for IVSCSPs with a c-semiring that is neither idempotent nor strictly monotonic is an open question.

## Possibly above threshold $\alpha$

We know, by Proposition 8, that, given an IVSCSP $P$ and an assignment $s$, $s$ is in $Pos(P, \alpha)$ if and only if $\alpha \leq U(s)$. Thus, to test whether a solution is in $Pos(P, \alpha)$, it is enough to check this condition, that takes linear time.

If the c-semiring is idempotent, to find a solution in $Pos(P, \alpha)$ it is sufficient to solve one CSP, as shown in the following theorem.

**Theorem 20.** *Given an IVSCSP $P$ over an idempotent c-semiring and an assignment $s$, $s \in Pos(P, \alpha)$ iff it is a solution of $cut_\alpha(bs(P))$.*

**Proof:** We first show that, if $s$ is a solution of $cut_\alpha(bs(P))$, then $s \in Pos(P, \alpha)$. If $s$ is a solution of $cut_\alpha(bs(P))$, then, since $cut_\alpha(bs(P))$ is the CSP obtained from the best scenario by allowing only tuples with preference greater than or equal to $\alpha$, $\text{pref}(bs(P), s) \geq \alpha$. Hence, in the best scenario $s$ has preference greater than or equal to $\alpha$, hence $s \in Pos(P, \alpha)$.

To conclude the proof, we show that if $s \in Pos(P, \alpha)$, then $s$ is a solution of $cut_\alpha(bs(P))$. If $s \in Pos(P, \alpha)$, then there is a scenario, say $Q_i$, where $\text{pref}(Q_i, s) \geq \alpha$. Hence, since the preference of a solution in a scenario is always lower than or equal to its preference in the best scenario, then $\text{pref}(bs(P), s) \geq \text{pref}(Q_i, s) \geq \alpha$, and so $s$ is a solution of $cut_\alpha(bs(P))$. □

By Proposition 8, we know that $Pos(P, \alpha^*) = UO(P)$. Therefore, to find a solution in $Pos(P, \alpha^*)$, it is sufficient to find an optimal solution of the best scenario of $P$, i.e., a solution in $UO(P)$, and thus to solve one SCSP.

Table 4.2: Finding and testing optimal solutions.

| *Optimality notion* | *c-semiring* | *Finding* | *Testing* |
|---|---|---|---|
| $LO(P)$ | generic | 1 SCSP | 1SCSP |
| $UO(P)$ | generic | 1 SCSP | 1SCSP |
| $IO(P)$ | generic | 1 SCSP | 3 SCSPs |
| | idempotent | 1SCSP | 1CSP |
| $LLO(P)$ | strictly monotonic | 1 SCSP | 1 SCSP |
| $WID(P)$ | idempotent | 2 SCSPs + 1 CSP | 2SCSPs |
| $ID(P)$ | generic | 2 SCSPs | 2 SCSPs |
| | idempotent | 1 SCSP + 1 CSP | 1 SCSP + 1 CSP |
| $NO(P)$ | idempotent | 2 SCSPs + 2 CSPs | 2 SCSPs + 1 CSP |
| | strictly monotonic | 1 SCSP | 1 SCSP |
| $Nec(P, \alpha)$ | generic | 1 SCSP | linear time |
| | idempotent | 1 CSP | linear time |
| $Nec(P, \alpha_*)$ | generic | 1 SCSP | linear time |
| $PO(P)$ | idempotent | 1 SCSP | 1 SCSP |
| | strictly monotonic | 1 SCSP | 1 SCSP |
| $Pos(P, \alpha)$ | idempotent | 1 CSP | linear time |
| $Pos(P, \alpha^*)$ | generic | 1 SCSP | linear time |

### Finding and testing optimality notions: summary of the results

We have provided algorithms to find solutions according to the various optimality notions and also to test whether a given solution is optimal. In most of the cases, these algorithms amounts to solving a soft constraint problem as shown in Table 4.2.

## 4.7 Necessary and possible dominance

Besides finding or testing for optimality, it may sometimes be useful to know if a solution dominates another one. We will consider four notions of dominance, which are related to the general optimality notions defined above.

**Definition 42** ((strictly) dominance)**.** *Given a scenario $Q$, a solution $s$ strictly dominates (resp., dominates) a solution $s'$ if and only if* $\text{pref}(Q, s) >$

$\text{pref}(Q, s')$ *(resp.,*$\text{pref}(Q,\ s) \geq \text{pref}(Q, s')$*) in the ordering of the considered c-semiring.*

**Definition 43** (necessarily (strictly) dominance)**.** *Given an IVSCSP $P$ and two solutions $s$ and $s'$ of $P$, $s$ necessarily strictly dominates (resp., necessarily dominates) $s'$ if and only if, in all scenarios, $s$ strictly dominates (resp., dominates) $s'$. We will denote with $NDTOP(P)$ (resp., $NSDTOP(P)$) the undominated elements in the binary relation given by the necessarily dominance (resp., strictly necessarily dominance).*

**Definition 44** (possibly (strictly) dominance)**.** *Given an IVSCSP $P$ and two solutions $s$ and $s'$ of $P$, $s$ possibly strictly dominates (resp., possibly dominates) $s'$ if and only if there is at least one scenario where $s$ strictly dominates (resp., dominates) $s'$. We will denote with with $PDTOP(P)$ (resp., $PSDTOP(P)$) the undominated elements of the binary relation given by the possibly dominance (resp., strictly possible dominance).*

In the IVSCSP $P$ of Figure 4.1, $s_1$ necessarily strictly dominates $s_8$. In the best scenario, $s_2$ strictly dominates $s_4$, while in the worst scenario $s_4$ strictly dominates $s_2$. Thus $s_2$ possibly strictly dominates $s_4$, and vice versa.

**Theorem 21.** *Consider an IVSCSP $P$. The following results hold:*

- $NO(P) \subseteq NDTOP(P) \subseteq NSDTOP(P)$.
- $NSDTOP(P) \supseteq PO(P)$.
- *If the c-semiring is strictly monotonic or idempotent, then* $NDTOP(P) \subseteq PO(P)$.
- *If the c-semiring is strictly monotonic,* $NSDTOP(P) = PO(P)$.
- *The sets $PSDTOP(P)$ and $PDTOP(P)$ may be empty.*
- *If* $PDTOP(P) \neq \emptyset$*, then* $|PDTOP(P)| = 1$.
- $PDTOP(P) \subseteq PSDTOP(P) = NO(P)$.

**Proof:** Let $s$ be a solution of $P$.

We first show that $NO(P) \subseteq NDTOP(P)$. If $s \notin NDTOP(P)$, then there a solution $s'$ that necessarily dominates $s$, and so there is a scenario $Q$ where $s'$ strictly dominates $s$, that is, $\text{pref}(Q, s') > \text{pref}(Q, s)$. Hence, $s$ is not optimal in that scenario and so $s \notin NO(P)$.

We now show that $NDTOP(P) \subseteq NSDTOP(P)$. If $s \notin NSDTOP(P)$, then there is a solution $s'$ that necessarily strictly dominates $s$ and so $s'$ necessarily dominates $s$ and thus $s \notin NDTOP(P)$.

We now show that $PO \subseteq NSDTOP(P)$. If $s \notin NSDTOP(P)$, then there is a solution $s'$ that necessarily strictly dominates $s$, hence, for every scenario $Q$, $s'$ strictly dominates $s$, that is, for every scenario $Q$, $pref(Q, s') > pref(Q, s)$, hence for every scenario $Q$, $s$ is not optimal, hence $s \notin PO(P)$.

To prove that $NDTOP(P) \subseteq PO(P)$ when $P$ is idempotent, we will show that if $s \in NDTOP(P)$ then $s$ is optimal in the scenario $Q^s$, where every tuple in $s$ is set to its maximum preference value and all other tuples are set to their minimum preference value. This then implies that $s$ is possibly optimal, and hence in $PO(P)$, as required.

Suppose, that $s \in NDTOP(P)$ is not optimal in the scenario $Q^s$, so there exists some solution $s'$ with $pref(Q^s, s') > pref(Q^s, s)$. Since $s \in NDTOP(P)$ there exists a scenario $Q$ with $pref(Q, s) > pref(Q, s')$ or else $s'$ would necessarily dominate $s$. We have $pref(Q^s, s') > pref(Q, s')$. Since the combination is minimum, this means that the preference value of the worst tuple of $s'$ (i.e., of the worst constraint) is worse in $Q$ than it is in $Q^s$. The definition of $Q^s$ means that this tuple is also in $s'$ (i.e., $s$ and $s'$ agree on the scope of the worst constraint). This implies that $pref(Q, s) \leq pref(Q, s')$, which contradicts $pref(Q, s) > pref(Q, s')$, completing the proof that $NDTOP(P) \subseteq PO(P)$ when $P$ is idempotent.

If the c-semiring is strictly monotonic, $NSDTOP(P) = PO(P)$. We have already shown that $NSDTOP(P) \supseteq PO(P)$. We now show that $NSDTOP(P) \subseteq PO(P)$. If $s \in NSDTOP(P)$, then there is no solution $s'$ such that for every scenario $Q_i$, $\text{pref}(Q_i, s') > \text{pref}(Q_i, s)$. Hence, for every $s'$, there is a scenario $Q_i$ where $\text{pref}(Q_i, s') \leq \text{pref}(Q_i, s)$. By following the same reasoning done above, it is possible to show that, $\forall s'$, $\text{pref}(Q^s, s') \leq \text{pref}(Q^s, s)$. Therefore, $s$ is optimal in $Q^s$ and so $s \in PO(P)$.

Furthermore, if the c-semiring is strictly monotonic, then we have $NDTOP(P) \subseteq PO(P)$ since $NDTOP(P) \subseteq NSDTOP(P) = PO(P)$.

$PSDTOP(P)$ and $PDTOP(P)$ may be empty, because there can be cycles in the *possibly dominates* and *possibly strictly dominates* relations. Let us consider the solutions $s_2$ and $s_4$ in the running example. $s_2$ has preference interval $[0.5, 0.9]$ and $s_4$ has preference interval $[0.6, 0.8]$. Then, $s_2$ possibly strictly dominates (and so possibly dominates) $s_4$, since $s_2$ strictly dominates $s_4$ in the best scenario, and $s_4$ possibly strictly dominates (and so possibly dominates) $s_2$, since $s_4$ strictly dominates $s_2$ in the worst scenario.

If $PDTOP(P) \neq \emptyset$, then $|PDTOP(P)| = 1$. In fact, assume that $PDTOP(P)$ contains two complete assignments $s_1$ and $s_2$. If $s_1$ and $s_2$ are in $PDTOP(P)$, then $s_1$ does not possibly dominate $s_2$ and $s_2$ does not possibly dominate $s_1$. Since $s_1$ does not possibly dominate $s_2$, then for every scenario $Q$ of $P$, $\text{pref}(Q, s_1) < \text{pref}(Q, s_2)$, and, since $s_2$ does not possibly dominate $s_1$, then for every scenario $Q$ of $P$, $\text{pref}(Q, s_2) < \text{pref}(Q, s_1)$, that

is a contradiction.

$PSDTOP(P) = NO(P)$. In fact, $s \in PSDTOP(P)$ iff there is no solution $s'$ such that $s'$ possibly strictly dominates $s$, iff there is no solution $s'$ that strictly dominates $s$, iff there is no solution $s'$ such that $\text{pref}(Q, s') > \text{pref}(Q, s)$ for some scenario $Q$, iff for every solution $s'$, $\text{pref}(Q, s) \geq \text{pref}(Q, s')$ for every scenario $Q$, iff $s \in NO(P)$.

$PDTOP(P) \subseteq PSDTOP(P)$. In fact, if $s \notin PSDTOP(P)$, then there is a solution $s'$ that possibly strictly dominates $s$ and thus $s'$ possibly dominates $s$ and so $s \notin PDTOP(P)$. □

Summarizing, given an IVSCSP $P$ with an idempotent or a strictly monotonic c-semiring, we have the following inclusions, that are shown in Figure 4.4: $PDTOP(P) \subseteq PSDTOP(P) = NO(P) \subseteq NDTOP(P) \subseteq PO(P) \subseteq NSDTOP(P)$. Moreover, when the c-semiring is strictly monotonic, we have also $NSDTOP(P) = PO(P)$. Therefore, the set of the necessarily optimal solutions of $P$ coincides with the set of the undominated elements of the binary relation given by the possibly strictly dominance over $P$, both if the c-semiring is strictly monotonic and if it idempotent. Moreover, the set of the possibly optimal solutions of $P$ coincides with the set of the undominated elements of the binary relation given by the necessarily strictly dominance over $P$, if the c-semiring is strictly monotonic.

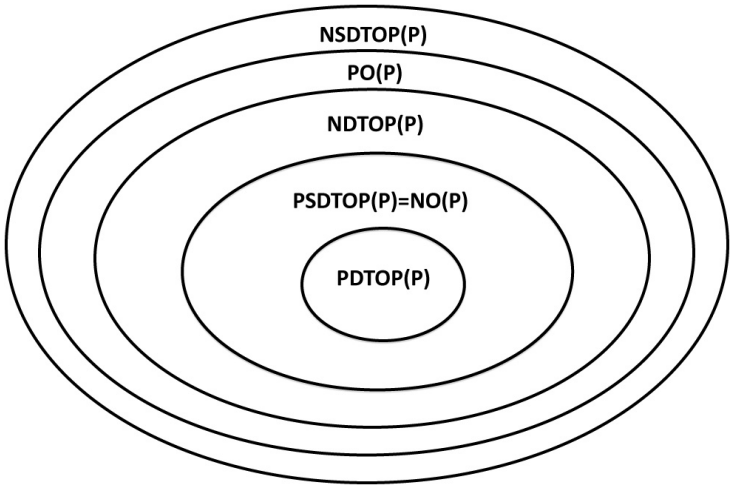



Figure 4.4: Relation between undominated elements of the binary relation given by the (strictly) necessarily dominance and the undominated elements of the binary relation given by the (strictly) possibly dominance for an IVSCSP $P$ defined over an idempotent or a strictly monotonic c-semiring.

To test if $s$ possibly strictly dominates (resp., possibly dominates) $s'$ we

can set each interval associated with $s$ but not with $s'$ to its upper bound; let $\lambda$ be the combination of these values. Then we set each interval associated with $s'$ but not with $s$ to its lower bound; let $\mu$ be the combination of these values. Finally, we compare the preference values of $s$ and $s'$, by testing if the condition $\lambda \times u_1 \times \cdots \times u_k > \mu \times u_1 \times \cdots \times u_k$ (resp., $\lambda \times u_1 \times \cdots \times u_k \geq \mu \times u_1 \times \cdots \times u_k$ ) holds for any selections of values $u_1, \ldots, u_k$ in the intervals of both $s$ and $s'$. If we have strict monotonicity, testing this condition amounts to testing if $\lambda > \mu$ (resp., $\lambda \geq \mu$). If we have idempotence, we can replace each $u_i$ with its upper bound, and then test the condition.

To test if $s$ necessarily dominates $s'$, we first check if $s$ possibly strictly dominates $s'$. Then:

- If $s$ possibly strictly dominates $s'$, then there is a scenario where $s$ strictly dominates $s'$ and so $s'$ does not necessarily dominate $s$. Then, we check if $s'$ possibly strictly dominates $s$. If so, then there is a scenario where $s'$ strictly dominates $s$, hence $s$ does not necessarily dominate $s'$. Therefore, $s$ and $s'$ are incomparable w.r.t. the necessarily dominance relation and so we conclude negatively. Otherwise, if $s'$ does not possibly strictly dominates $s$, then, for every scenario, $s$ dominates $s'$ and, since, by hypothesis, there is a scenario where $s$ strictly dominates $s'$, then $s$ necessarily dominates $s'$ and so we conclude positively.

- If $s$ does not possibly strictly dominate $s'$, then, for every scenario, $s'$ dominates $s$, i.e., for every scenario $Q$, $\text{pref}(Q, s') \geq \text{pref}(Q, s)$. Then, we check if $s'$ possibly strictly dominates $s$. If so, then $s'$ necessarily dominates $s$ and so we conclude negatively. Otherwise, if $s'$ does not possibly strictly dominates $s$, then, for every scenario, $s$ dominates $s'$, i.e., for every scenario $Q$, $\text{pref}(Q, s) \geq \text{pref}(Q, s')$, and so, since by the hypothesis above $\text{pref}(Q, s') \geq \text{pref}(Q, s)$, we have that, for every scenario $Q$, $\text{pref}(Q, s) = \text{pref}(Q, s')$, hence $s$ does not necessarily dominates $s'$ and so we conclude negatively.

To test if $s$ necessarily strictly dominates $s'$, we follow a reasoning similar to the one presented above, but we consider the possibly dominance relation instead of the possibly strictly dominance relation. Moreover, when $s$ does not possibly dominate $s'$ (i.e., the second item above), we can conclude immediately negatively, since in this case $s'$ necessarily strictly dominates $s$.

## 4.8 Multiple intervals

One may wonder if IVSCSPs would be more expressive if we allowed not just a single preference interval for each assignment, but a set of such intervals.

For example, instead of giving us the interval [0.1, 0.8], a user could be more precise and give us [0.1,0.5] and [0.7,0.8]. This would reduce the uncertainty of the problem. We will now show that all the interval-based optimality notions and all the scenario-based optimality notions that guarantee a certain level of preference would give the same set of optimals in this more general setting. Moreover, when the c-semiring is strictly monotonic, also the possibly and necessary optimality notions give the same set of optimals. Also, when the c-semiring is idempotent, the necessary optimality notions give the same set of optimals. In the other cases, we are however able to find approximations of the possibly and necessarily optimal solutions. More precisely, we have the following results, that are also summarized in Table 4.3.

**Theorem 22.** *Consider an IVSCSP $P$. Take now a new problem $P'$ with the same variables, domains, and constraint topology as $P$, where, for each interval $[l,u]$ in $P$, there is a set of intervals $[l,u_1],[l_2,u_2],\ldots,[l_n,u]$ such that $u_i < l_{i+1}$ for $i = 1,\ldots,n-1$. Then:*

- $X(P) = X(P')$ *for* $X \in \{LO, UO, IO, LLO, ULO, WID, ID\}$.
- $Nec(P,\alpha) = Nec(P',\alpha)$ *for all* $\alpha$.
- $Pos(P,\alpha) = Pos(P',\alpha)$ *for all* $\alpha$.
- $NO(P') \supseteq NO(P)$.
- $PO(P') \subseteq PO(P)$.
- *If the c-semiring is strictly monotonic,* $NO(P) = NO(P')$ *and* $PO(P) = PO(P')$.
- *If the c-semiring is idempotent,* $NO(P) = NO(P')$.

**Proof:** To show that $X(P) = X(P')$ for $X \in \{LO, UO, IO, LLO, ULO, WID, ID\}$, it is sufficient to recall that all solutions in $\{LO, UO, IO, LLO, ULO, WID, ID\}$ are computed by considering for every tuple associated with interval $[l,u]$ only the lower bound $l$ and the upper bound $u$ that, by construction of $P'$, are the same in $P$ and $P'$.

Let $s$ be a complete assignment of $P$. Let us consider a generic $\alpha$. To show that $Nec(P,\alpha) = Nec(P',\alpha)$, we first show that $Nec(P,\alpha) \subseteq Nec(P',\alpha)$. If $s \in Nec(P,\alpha)$, then, for every scenario $Q$ of $P$, $\text{pref}(Q,s) \geq \alpha$. Since the set of the scenarios of $P$ is a superset of the scenarios of $P'$, this holds also for every scenarios of $P'$. Therefore, $s \in Nec(P',\alpha)$. We now show that $Nec(P',\alpha) \subseteq Nec(P,\alpha)$. If $s \notin Nec(P,\alpha)$, then $\text{pref}(Q,s) < \alpha$ for some scenario $Q$ of $P$ and this holds also for the worst scenario, since

$\text{pref}(ws(P), s) \leq \text{pref}(Q, s) < \alpha$. Since the worst scenario is one of the scenario of $P'$, then $s \notin Nec(P', \alpha)$.

To show that $Pos(P, \alpha) = Pos(P', \alpha)$, we first show that $Pos(P', \alpha) \subseteq Pos(P, \alpha)$. If $s \in Pos(P', \alpha)$, then for some scenario $Q$ of $P'$, $\text{pref}(Q, s) \geq \alpha$. Since every scenario of $P'$ is also a scenario of $P$, then $s \in Pos(P, \alpha)$. We now show that $Pos(P, \alpha) \subseteq Pos(P', \alpha)$. If $s \in Pos(P, \alpha)$, then $\text{pref}(Q, s) \geq \alpha$ for some scenario $Q$ of $P$, and this holds also for the best scenario, since $\text{pref}(bs(P), s) \geq \text{pref}(Q, s) \geq \alpha$. Since the best scenario is one of the scenarios of $P'$, then $s \in Pos(P', \alpha)$.

Since the set of the scenarios of $P$ is a superset of the scenarios of $P'$, then $NO(P) \subseteq NO(P')$. In fact, if $s \in NO(P)$, then it is optimal for every scenario of $P$ and also for every scenario of $P'$.

Moreover, $PO(P') \subseteq PO(P)$. In fact, if $s \in PO(P')$, then there is a scenario of $P'$ where $s$ is optimal and, as every scenario of $P'$ is also a scenario of $P$, then $s \in PO(P)$.

If the c-semiring is strictly monotonic, then $NO(P) = NO(P')$. By Theorem 17, we know that, if the c-semiring is strictly monotonic, then $s \in NO(P)$ iff $s \in Opt(Q_s)$, where $Q_s$ is the scenario where every preference associated to a tuple of $s$ is set to its lower bound and the preferences of all other tuples are set to their upper bound. Since $Q_s$ is one of the scenarios of $P'$, it is possible to show that $s \in NO(P')$ iff $s \in Opt(Q_s)$, by following the same proof of Theorem 17. Hence, $NO(P') = NO(P)$.

Similarly, if the c-semiring is strictly monotonic, then $PO(P) = PO(P')$. By Theorem 19, we know that, if the c-semiring is strictly monotonic, then $s \in PO(P)$ iff $s \in Opt(Q^s)$, where $Q^s$ is the scenario where all the preferences of tuples in $s$ are set to their upper bound and all other tuples are associated to their lower bound. Since $Q^s$ is one of the scenarios of $P'$, it is possible to show, by following the same proof of Theorem 19, that $s \in PO(P)$ iff $s \in Opt(Q^s)$.

If the c-semiring is idempotent, $NO(P) = NO(P')$. In fact, by Theorem 17, we know that $s \in NO(P)$ iff for every $s'$, $s \in Opt(Q^{s'})$, where $Q^{s'}$ is the scenario where we put every tuple of $s'$ to its upper bound and every other tuple to its lower bound. Since, for every $s'$, $Q^{s'}$ is a scenario of $P'$, then by following the same proof of Theorem 17, we can show that $s \in NO(P')$ iff for every $s'$, $s \in Opt(Q^{s'})$, Hence, $NO(P') = NO(P)$. □

Table 4.3: Comparison of the optimality sets of problems $P$ (with single intervals) and $P'$ (with multiple intervals), as defined in Theorem 22.

| *Optimality notion* | *c-semiring* | *Comparison* |
|---|---|---|
| $LO$ | generic | $LO(P) = LO(P')$ |
| $UO$ | generic | $UO(P) = UO(P')$ |
| $IO$ | generic | $IO(P) = IO(P')$ |
| $LLO$ | generic | $LLO(P) = LLO(P')$ |
| $ULO$ | generic | $ULO(P) = ULO(P')$ |
| $Nec(\alpha)$ | generic | $Nec(P, \alpha) = Nec(P', \alpha)$ |
| $Pos(\alpha)$ | generic | $Pos(P, \alpha) = Pos(P', \alpha)$ |
| $NO$ | generic | $NO(P) \subseteq NO(P')$ |
| | idempotent | $NO(P) = NO(P')$ |
| | strictly monotonic | $NO(P) = NO(P')$ |
| $PO$ | generic | $PO(P) \supseteq PO(P')$ |
| | strictly monotonic | $PO(P) = PO(P')$ |

## 4.9 Experimental results

### Instance generator

We randomly generated fuzzy IVMSPs (as defined in Section 4.4) according to the following parameters:

- $m$: number of meetings (default 12);
- $n$: number of agents (default 5);
- $k$: number of meetings per agent (default 3);
- $l$: number of time slots (default 10);
- $min$ and $max$: minimal (default 1) and maximal (default 2) distance (in time slots) between two locations;
- $i$: percentage of preference intervals (default 30%).

Given such parameters, we generate an IVSCSP with $m$ variables, representing the meetings, each with domain of size $l$. The domain values $1, \ldots, l$ represent the time slots, that are assumed to all have the same length equal to one time unit, and to be adjacent to each other. Thus, for example, time slot $i$ ends when time slot $i+1$ starts. Given two time slots $i$ and $j > i$, they

can be used for two meetings only if the distance between their locations (see later) is at most $j - i - 1$.

For each of the $n$ agents, we generate randomly $k$ integers between 1 and $m$, representing the meetings he needs to participate in. Also, for each pair of time slots, we randomly generate a integer between $min$ and $max$ that represents the time needed to go from one location to the other one. This will be called the distance table.

Given two meetings, if there is at least one agent who needs to participate in both, we generate a binary constraint between the corresponding variables. Such a constraint is satisfied by all pairs of time slots that are compatible according to the distance table.

We then generate the preferences over the domain values and the compatible pairs in the binary constraints, by randomly generating a number in $(0, 1]$ or an interval over $(0, 1]$, according to the parameter $i$.

As an example, assume to have $m = 5$, $n = 3$, $k = 2$, $l = 5$, $min = 1$, $max = 2$, and $i = 30$. According to these parameters, we generate a IVMSP with the following features:

- 5 meetings: $m_1$, $m_2$, $m_3$, $m_4$, and $m_5$;
- 3 agents: $a_1$, $a_2$, and $a_3$;
- 5 time slots: $t_1, \ldots, t_5$;
- agents' participation to meetings: we randomly generate 2 meetings for each agent, for example
  - $a_1$ must participate in meetings $m_1$ and $m_2$;
  - $a_2$ must participate in meetings $m_4$ and $m_5$;
  - $a_3$ must participate in meetings $m_2$ and $m_3$;
- distance table: we randomly generate its values, for example as in Table 4.4;
- we randomly generate the preferences associated to domain values and compatible pairs in the constraints, in a way that 30% of the preferences are preference intervals contained in $(0, 1]$ and 70% of the preferences are single values in $(0, 1]$.

In this example, a feasible meeting scheduling is obtained by assigning the following time slots to meetings: $(m_1, t_3), (m_2, t_1), (m_3, t_5), (m_4, t_2), (m_5, t_5)$. The preference interval for such a scheduling will depend on the preference values in the domains and constraints. More precisely, as we use preference

Table 4.4: Distance between meeting locations.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 1 | 2 |
| 2 | 1 | 0 | 2 | 1 | 2 |
| 3 | 2 | 2 | 0 | 1 | 1 |
| 4 | 1 | 1 | 1 | 0 | 2 |
| 5 | 2 | 2 | 1 | 2 | 0 |

values between 0 and 1 and we adopt the fuzzy criteria, the preference interval will be $[l, u]$, where $l$ (resp., $u$) is the minimum among all the lower (resp., upper) bounds of the preference intervals selected by this assignment in the constraints.

## Experimental tests

We implemented our algorithms using a Java (version 1.6.0_07) c-semiring based framework and the Choco constraint programming toolkit (version 1.2.06). Experiments were run on AMD Opteron 2.3GHz machines with 2GB of RAM.

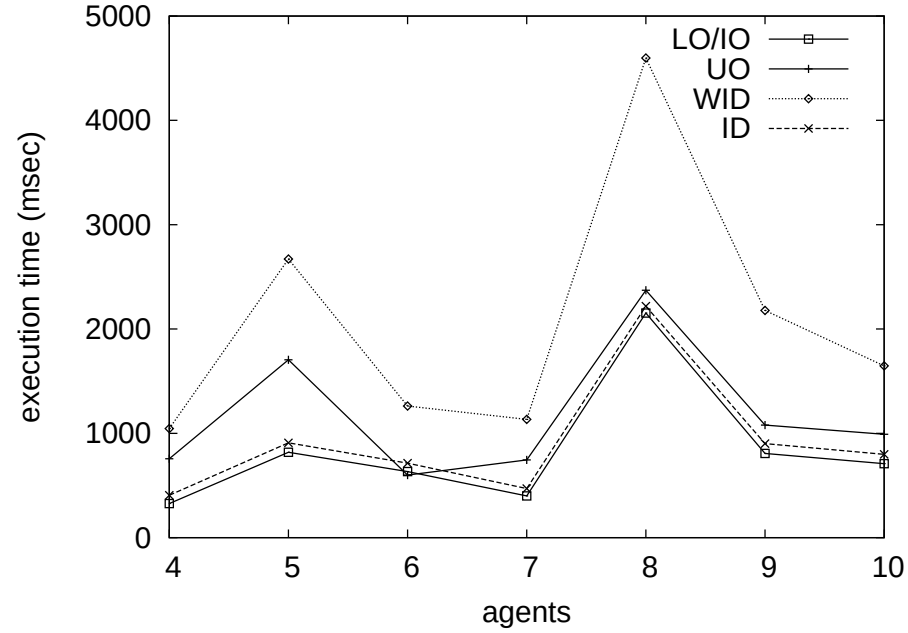
We used 4 different test sets, each one generated varying in turn $n$, $m$, $k$, and $i$, while fixing the others to their default values. Moreover, $\alpha$, i.e., the minimum level of preference used in $Pos(P, \alpha)$ and $Nec(P, \alpha)$, is always 0.5.[2] Each data point is the average of the execution on 50 problem instances.

Figure 4.5(a) shows the execution time (measured in milliseconds) of the algorithms to find a solution, belonging to each type of the interval-based optimality notions, as a function of the number of agents. We can notice that there is a peak when the number of agents is 8, which represents problems with a small number of solutions. With more agents, the problems have no solution, while with a smaller number of agents there are many solutions. In both such cases, it is easy to find a feasible meeting scheduling.

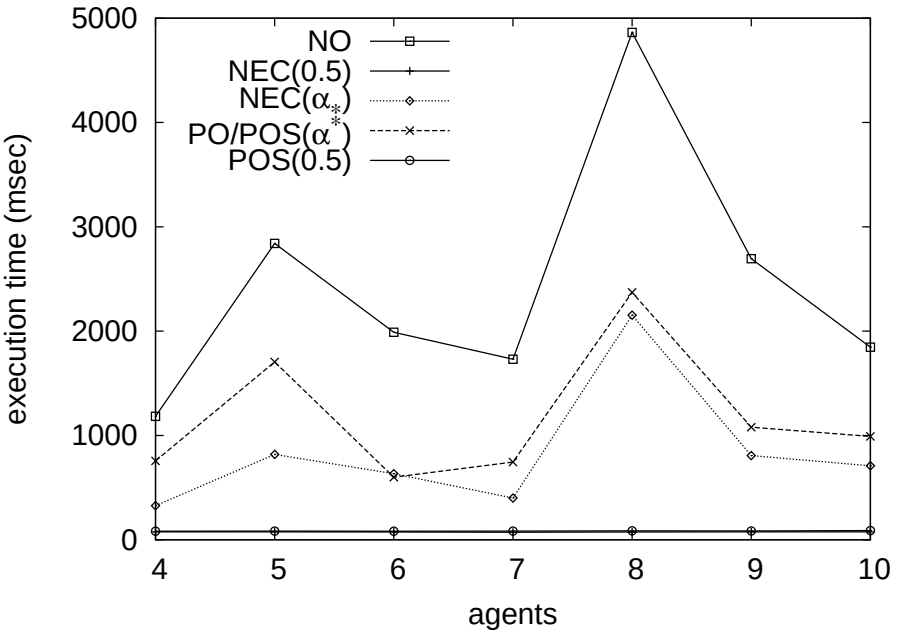
For the more general optimality notions, Figure 4.5(b) shows that the behavior is the same except for POS(0.5) and NEC(0.5) because, in these algorithms, we need to solve a CSP, while in the other algorithms we solve at least one SCSP. In fact, POS(0.5) and NEC(0.5) takes approximately the same time no matter the number of agents in the problem.

Figures 4.6(a) and 4.6(b) show the performance of the algorithms for all optimality notions, as a function of the number of meetings per agent. Since

[2]In the following figures, we will omit writing $P$ in the names of the algorithms.

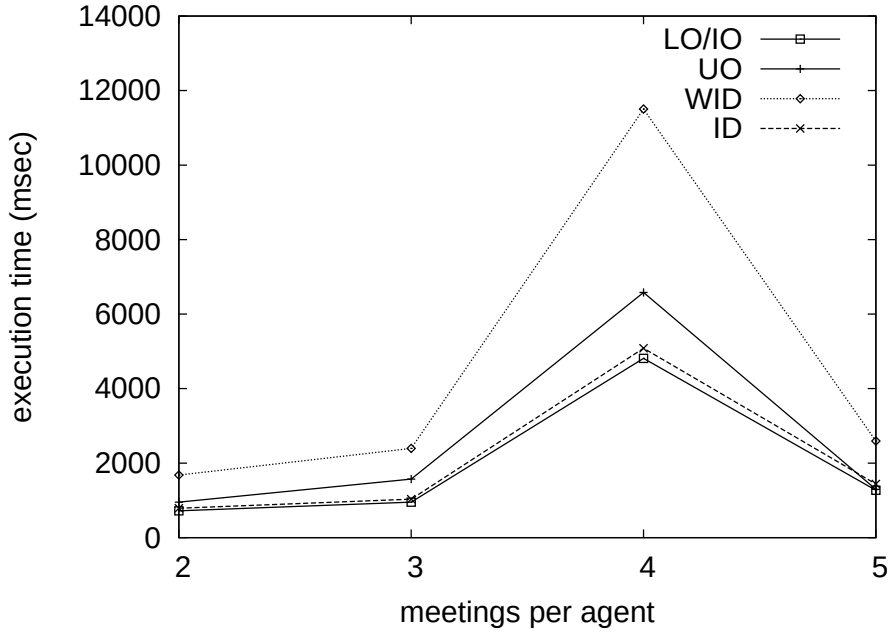



(a)

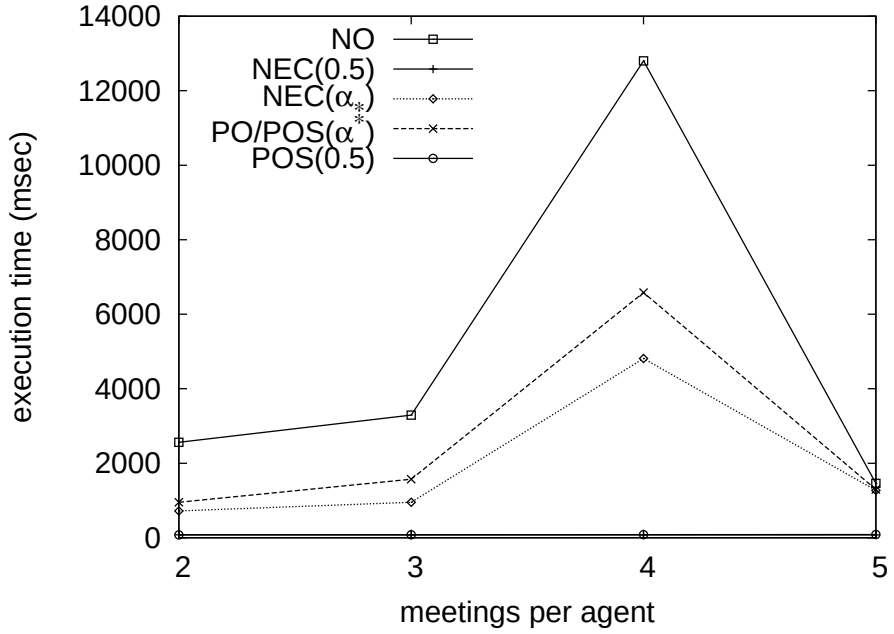



(b)

Figure 4.5: Execution time (msec.) as a function of number of agents.

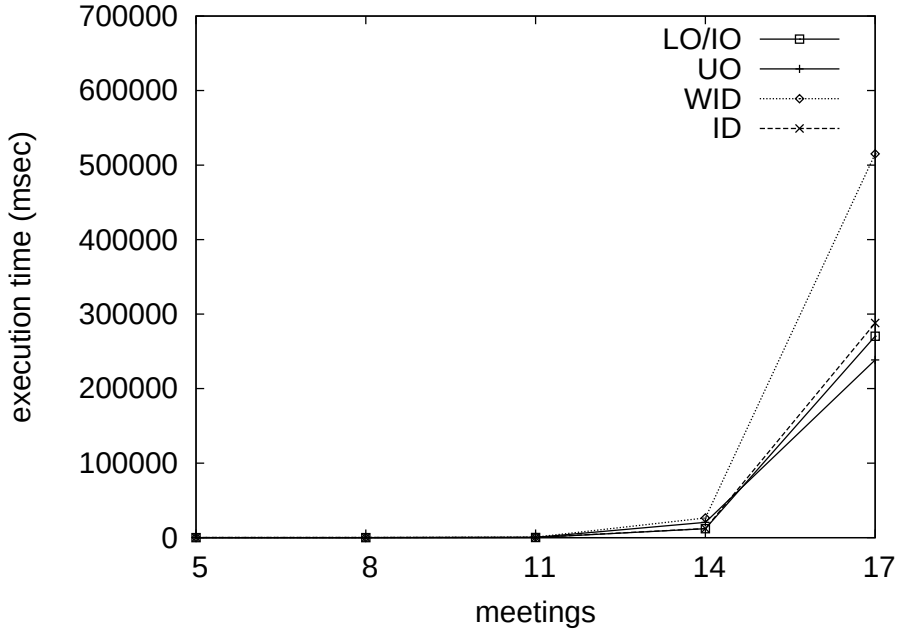



(a)

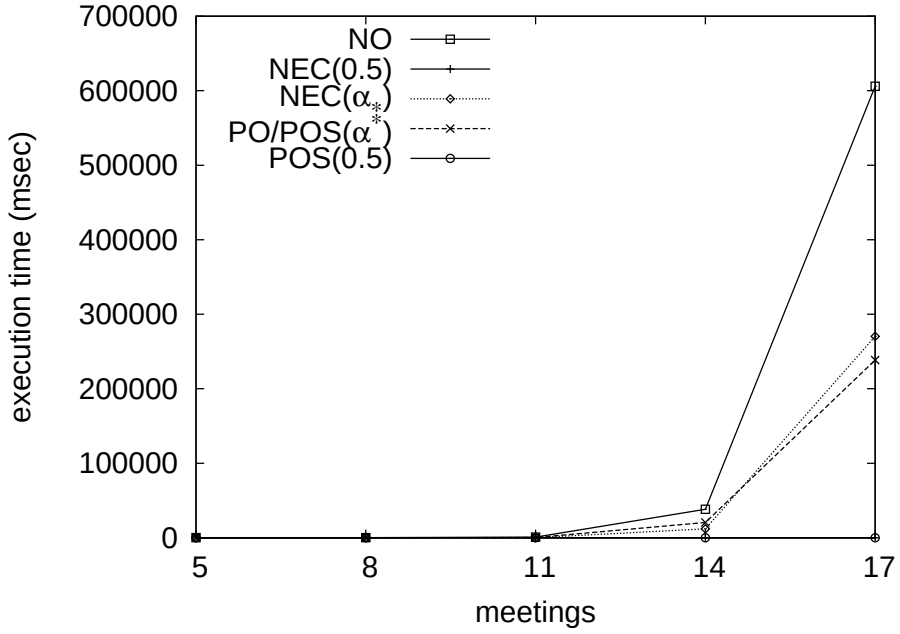



(b)

Figure 4.6: Execution time (msec.) as a function of meetings per agent.

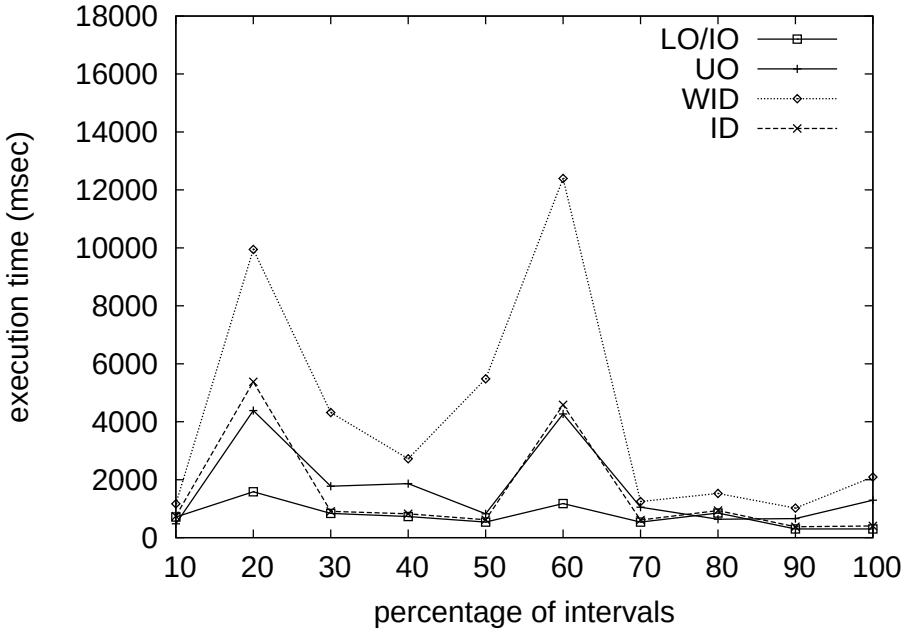



(a)

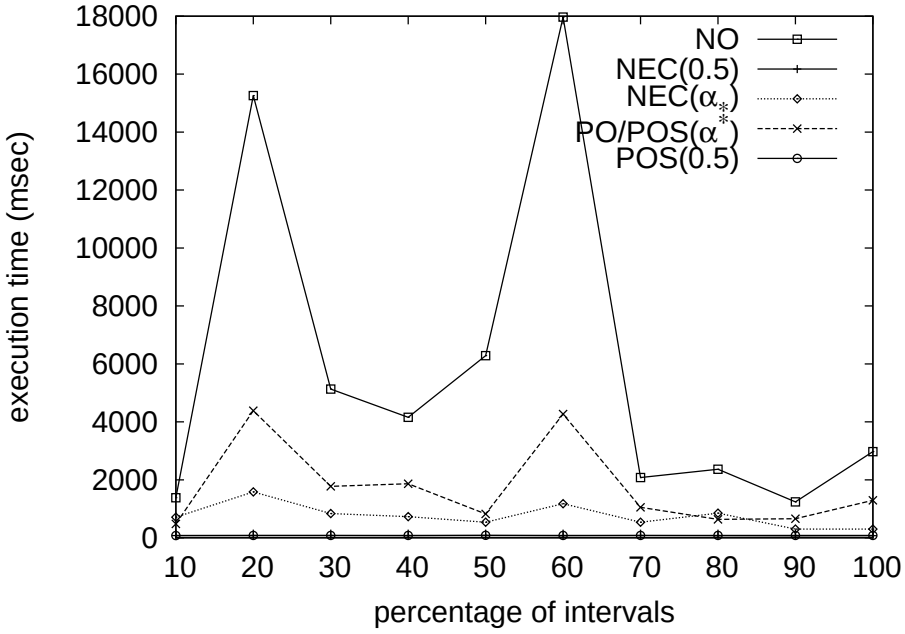



(b)

Figure 4.7: Execution time (msec.) as a function of the number of meetings.

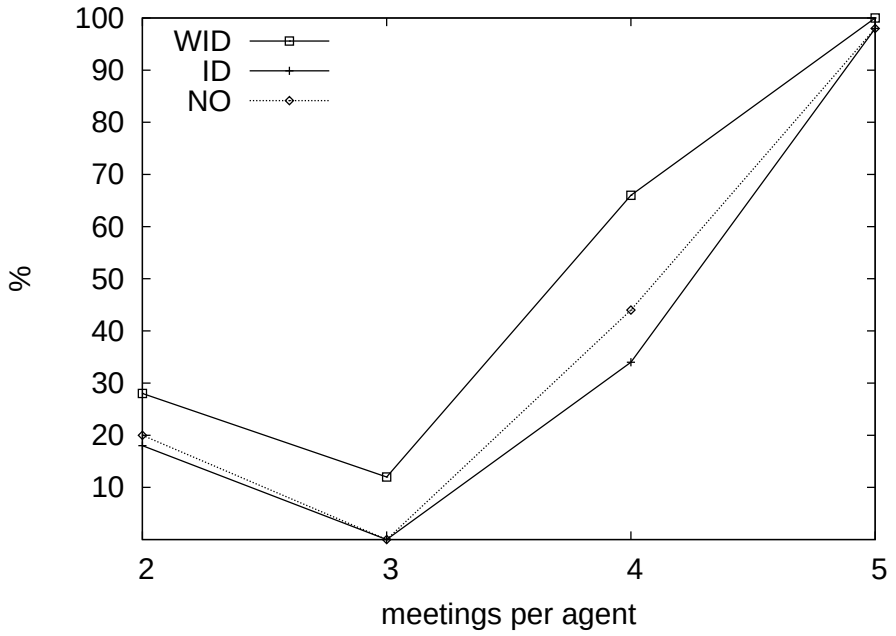



(a)

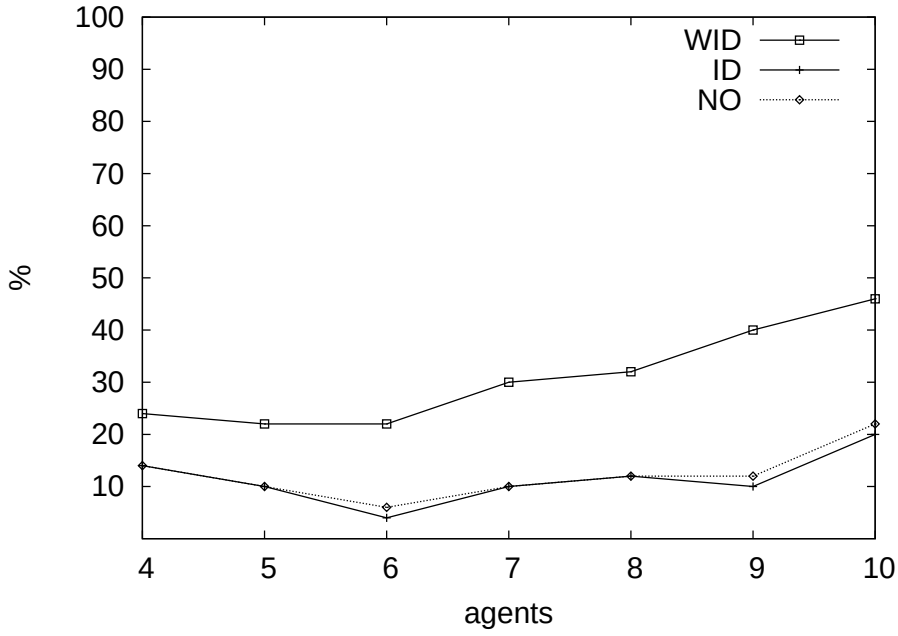



(b)

Figure 4.8: Execution time (msec.) as a function of the percentage of intervals.

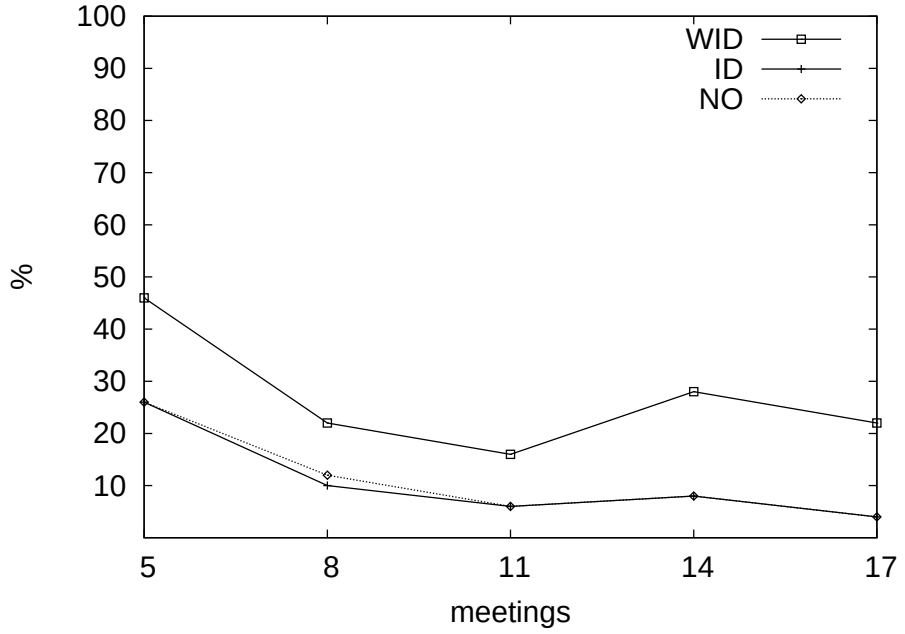



(a)

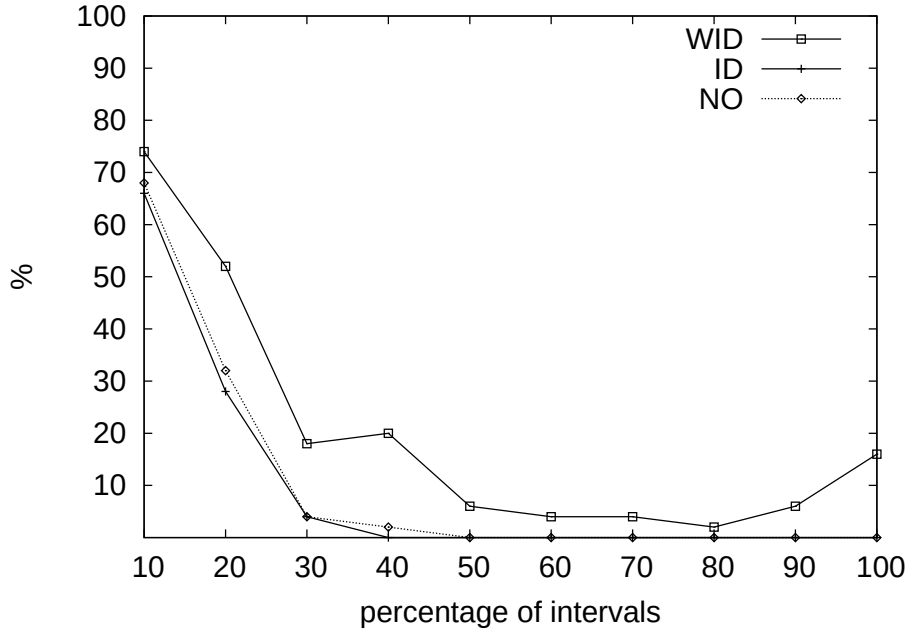



(b)

Figure 4.9: Existence of WID, ID, and NO solution, varying agents and meetings per agent.




(a)




(b)

Figure 4.10: Existence of WID, ID, and NO solution, varying meetings and intervals.

$LO(P) = Nec(P, \alpha_*)$ and $UO(P) = Pos(P, \alpha^*)$, these curves in the two graphs coincide. The lines corresponding to the WID algorithm in Figure 4.6(a) and to the NO algorithm in Figure 4.6(b) are similar, and are above the others in both figures, because the WID algorithm needs to find the lower and upper optimal preference, to perform two cuts, and to solve the CSP obtained combining the cuts, while the other algorithms (expect NO) only need to solve an SCSP. Moreover, the WID algorithm is a sub-routine of the NO algorithm.

Notice that finding solutions in NO, $Nec(P, \alpha_*)$, or $POS(P, \alpha^*)$ is more expensive than finding solutions in $Nec(P, 0.5)$, or $POS(P, 0.5)$, as expected since $\alpha_*$ and $\alpha^*$ are the best preference levels that one can reach.

The peak at 4 meetings per agents, shown in Figures 4.6(a) and 4.6(b), corresponds to problems which are more difficult to solve because they have very few solutions. This is analogous to what we have noticed in Figures 4.5(a) and 4.5(b) with the peak at 8 agents.

Figures 4.7(a) and 4.7(b) show that the execution time increases exponentially when the number of meetings (i.e., the number of variables in the problem) arises. In this case, the execution time is mainly influenced by the size of the problems, no matter which algorithm is used.

Figures 4.8(a) and 4.8(b) show that the execution time is not influenced by the amount of intervals in the problem. As in all the other graphs, finding a WID or an NO solution is more expensive than finding other kinds of solutions. The two peaks at 20% and 60% of intervals are due to two very hard problems inside the test set.

Figure 4.9(a) and Figure 4.9(b) consider those optimality sets that can be empty (that is, WID, ID, and NO) and show the percentage of times a solution of a certain kind exists. Clearly, when there is no solution, WID, ID and NO contain all assignments and coincide. This is the case when the number of meetings per agents is larger (more than 3 meetings per agent in our settings). When we consider less constrained problems with 2-3 meetings per agent, as expected, we have more WID solutions than ID and NO solutions. Notice that the size of WID, ID and NO varies very little when the number of agents is between 4 and 8 (Figure 4.9(b)). However, when such a number is between 8 to 10, the size of the solution sets is larger because there are more instances with no solution. If we vary the number of meetings, we can see in Figure 4.10(a) that the number of such a kind of solutions tends to decrease slightly as the number of variables (i.e. meetings) arises. In fact, a larger number of variables may imply a larger number of constraints, which may imply a smaller number of WID, ID, and NO solutions.

In figure 4.10(b) we consider instances where we vary the percentage of intervals from 10 to 100%. When incompleteness is higher than 40%, most

of the instances don't have WID, ID, and NO solutions. This is predictable, because a larger number of intervals makes it less probable the existence of solutions that are optimal in all scenarios, since the number of scenarios is larger.

## 4.10 Related work

Previous approaches to uncertainty in soft constraint problems assumed either a complete knowledge of the preference value, or a complete ignorance. In other words, a preference value in a domain or a constraint was either present or not [22, 33, 72]. Then, the solver was trying to find optimal solutions with the information given by the user or via some form of elicitation of additional preference values. Here instead we consider a more general setting where the user may specify preference intervals. Also, we assume that the user has given us all the information he has about the problem, so we do not resort to preference elicitation (or the elicitation phase is over with the user being unable or unwilling to give us more precise information). Another work that analyzes the impact of the uncertainty in soft constraint problems is shown in [57]. However, while we assume to have only preference intervals, in [57] it is assumed that all the preferences are given and some of them are tagged as possibly unstable and are provided with a range, of possible variations, around their value. The work by Petit, Regin and Bessiere cannot model the problems we consider since costs are single values and not intervals.

Other papers consider preference intervals, such as the work in [12]. However, these lines of work focus on specific preference aggregation mechanisms (such as the Choquet integral) and of modelling issues without addressing the algorithmic questions related to finding optimal solutions according to different risk attitudes. We are instead interested in providing efficient algorithms to find optimal solutions according to different risk attitudes (that we call pessimistic and optimistic), besides the modelling concerns. For this reason, we model imprecise problems within an extension of soft constraints that allows us to exploit the existing machinery to solve soft constraint problems to obtain optimal solutions in the presence of preference intervals.

## 4.11 Conclusions

Given an IVSCSP $P$, the solutions in $NO(P)$ are certainly the most attractive, as they are the best ones in every scenario. However, if there is

none, we can look for solutions in $Nec(P, \alpha_*)$ (which coincide with solutions in $LO(P)$), which guarantee a preference level $\alpha_*$ in all scenarios. If $\alpha_*$ is too low, we can consider other notions of optimality; for example, if we feel optimistic, we can consider the solutions in $Pos(P, \alpha^*)$(which coincide with solutions in $UO(P)$): they guarantee that it is possible to reach a higher level of preference, although not in all scenarios.

If we allow users to associate to each partial assignment in the constraints not just a single interval, but a set of multiple intervals, this would reduce the uncertainty of the problem. However, when the c-semiring is strictly monotonic (resp., idempotent), this added generality does not change the set of the optimal solutions in any of the considered notions (resp., in any of the considered notions with the exception of the possibly optimal notions). This means that a level of precision greater than a single interval does not add useful information when looking for an optimal solution.

# Chapter 5

# Local search for stable marriage problems

In this chapter we consider the stable marriage problem and we study the effectiveness of a local search approach in finding a stable marriage and sampling the lattice of all stable marriages. The intent is to develop fair procedure to find stable marriages.

Some of the results in this chapter are included in the following articles:

- *Local search algorithms on the Stable Marriage Problem: Experimental Studies*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 19th European Conference on Artificial Intelligence (ECAI 2010), IOS Press, 2010, short paper.

- *Local search for stable marriage problems*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 3rd International Workshop on Computational Social Choice (COMSOC 2010), Dusseldorf, Germany, September 2010.

## 5.1 Motivations

Gale and Shapley give a quadratic time algorithm to solve the stable marriage problem, based on a series of proposals of men to women (or vice versa) [26]. The problem has received a great deal of attention also from the constraint programming community. Various CP models have been proposed [28] and used in both centralized [69, 68] and distributed solving schemes [10].

However, such approaches aim to find a stable but "extreme" marriage (either the male-optimal or the female-optimal) by favouring one gender at the expense of the other. In this chapter we are interested in finding a

"fairer" approach to find stable marriages which samples well the set of all stable marriages. This means not to find a marriage in the middle of the lattice with the same distance from the top and from the bottom, but a method to find stable marriages that does not favour one particular gender during the search. To do this, we will employ a local search approach because its intrinsic randomness can be useful to guarantee the fairness of the search process and because it can be easily extended to deal with stable marriage problems with ties and incomplete lists which is NP-hard (see Chapter 6 for more details).

Our local search approach follows a simple schema: we start from a randomly chosen marriage and, at each step, we move to a neighbor marriage by minimizing the distance to stability, which is measured by the number of blocking pairs. To avoid redundant computation due to the possibly large number of blocking pairs, we consider only those that are undominated, since their elimination minimizes the distance to stability. Random moves are also used, to avoid stagnation in local minima. The algorithm stops when it finds a stable marriage, or when a given limit on the number of steps is reached.

## 5.2 A local search approach

We adapt the classical local search schema to SMs as follows. Given an SM instance $P$, we start from a randomly generated marriage $M$. Then, at each search step, we compute the set $BP$ of blocking pairs in $M$ and we compute the neighborhood, which is the set of all marriages obtained obtained by removing one of the blocking pairs in $BP$ from $M$. More precisely, let $M$ a marriage, $bp = (m, w)$ a blocking pair in $M$, $m' = M(w)$, and $w' = M(m)$. Then, removing $bp$ from $M$ (written $M \backslash bp$) means obtaining a marriage $M'$ in which $m$ is married with $w$ and $m'$ is married with $w'$, leaving the other pairs unchanged. To select the neighbor $M'$ of $M$ to move to, we use an evaluation function that counts the number of blocking pairs in all neighboring marriages, and we move to the one with the smallest number of blocking pairs.

To avoid stagnation in a local minimum of the evaluation function, 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 marriage $M$.

The algorithm terminates if a stable marriage is found or when a maximal number of search steps or a timeout is reached. The pseudocode of our algorithm, called SML, is shown in Algorithm 4.

Notice that in line 4 of algorithm SML we compute all blocking pairs in

**Algorithm 4**: SML

**Input** : a marriage problem $P$ of size $n$, a timeout $max\_time$, a maximal number of steps $max\_steps$, a probability $p$
**Output**: a stable marriage $M$

```
1  M ← random marriage
2  steps ← 0
3  repeat
4      BP ← blocking pairs in M
5      if |BP| = 0 then
6          return M
7      if rand() ≤ p then
8          select randomly a blocking pair bp in BP
9          M ← M\bp
10     else
11         minbp ← n * (n − 1)
12         foreach blocking pair b in BP do
13             M' ← M\b
14             nbp ← number of blocking pairs in M'
15             if nbp < minbp then
16                 best_bp ← b, minbp ← nbp
17         M ← M\best_bp
18     steps ← steps + 1
19 until steps ≥ max_steps or current_time > max_time ;
```

the current marriage. The number of such blocking pairs may be very large. Also, some of them may be useless, since their removal would surely lead to new marriages that will not be chosen by the evaluation function. This is the case for the so-called *dominated* blocking pairs. We will modify SML to consider only undominated blocking pairs.

**Definition 45** (dominance in blocking pairs)**.** *Let $m$ be a man and $(m, w)$ and $(m, w')$ two blocking pairs. Then $(m, w)$ dominates (from the men's point of view) $(m, w')$ if $m$ prefers $w$ to $w'$. A similar reasoning can be done from the women's point of view.*

**Definition 46** (undominated blocking pair)**.** *A men- (resp., women-) undominated blocking pair is a blocking pair such that there is no other blocking pair that dominates it from the men's (resp., women's) point of view. When the point of view (men or women) is clear or not important, we will omit it.*

| men's preference lists | women's preference lists |
|---|---|
| 1: 5 7 1 2 6 8 4 3 | 1: 5 3 7 6 1 2 8 4 |
| 2: 2 3 7 5 4 1 8 6 | 2: 8 6 3 5 7 2 1 4 |
| 3: 8 5 1 4 6 2 3 7 | 3: 1 5 6 2 4 8 7 3 |
| 4: 3 2 7 4 1 6 8 5 | 4: 8 7 3 2 4 1 5 6 |
| 5: 7 2 5 1 3 6 8 4 | 5: 6 4 7 3 8 1 2 5 |
| 6: 1 6 7 5 8 4 2 3 | 6: 2 8 5 4 6 3 7 1 |
| 7: 2 5 7 6 3 4 8 1 | 7: 7 5 2 1 8 6 4 3 |
| 8: 3 8 4 5 7 2 6 1 | 8: 7 4 1 5 2 3 6 8 |

Table 5.1: An example of an SM of size 8.

It is easy to see that, if $M$ is an unstable marriage, $(m, w)$ an undominated blocking pair in $M$, $m' = M(w)$, $w' = M(m)$, and $M' = M \backslash (m, w)$, there are no blocking pairs in $M'$ in which $m$ and $w$ are involved. This property would not be true if we remove a dominated blocking pair. This is why we would like to focus on the removal of undominated blocking pairs to pass from one marriage to another one in our local search algorithms.

Consider the problem in Table 5.1 and the marriage 2 7 4 8 6 3 5 1. The blocking pair $(m_8, w_4)$ dominates (from the men's point of view) $(m_8, w_2)$. If we remove $(m_8, w_2)$ from the marriage, $(m_8, w_4)$ will remain. On the other hand, removing $(m_8, w_4)$ also eliminates $(m_8, w_2)$. Thus, removing $(m_8, w_4)$ is more useful than removing $(m_8, w_2)$.

In algorithm UB1, we find the undominated blocking pairs from the men's point of view and, among those, we keep only the undominated blocking pairs from the women's point of view. This is done using array *upb* which, after lines 2-10, contains, for each woman $w$, a man $m$ such that $(m, w)$ is an undominated blocking pair from both women's point of view and men'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 man and woman.

We call SML1 the algorithm obtained from SML by using algorithm UB1 to compute the blocking pair set BP in line 4 of SML. Notice that, by using the undominated blocking pairs instead of all the blocking pairs, we also limit the size of the neighborhood, since each man or woman is involved in at most one of the undominated blocking pairs found by UB1, thus we have at most $2n$ neighboring marriages to evaluate.

Let us now analyze more carefully the set of blocking pairs obtained by procedure UB1. Consider the case in which a man $m_i$ is in two blocking pairs, say $(m_i, w_j)$ and $(m_i, w_k)$, and assume that $(m_i, w_j)$ dominates $(m_i, w_k)$

**Algorithm 5**: UB1

**Input** : a marriage problem $P$ of size $n$, a marriage $M$
**Output**: a set BP of blocking pairs.

```
1  BP ← ∅
2  ubp ← array of length n with indices w_1, ..., w_n
3  foreach w_i, i = 1..n do ubp[w_i] ← null
4  foreach m_j, j = 1..n do
5      found ← false
6      foreach (w_i in m_j's list better than M(m_j)) and (!found)
       do
7          if (m_j, w_i) is a blocking pair then
8              if ubp[w_i] = null or w_i prefers m_j to ubp[w_i] then
9                  ubp[w_i] ← m_j
10             found ← true
11 foreach w_i, i = 0..n do
12     if ubp[w_i] ≠ null then
13         BP ← BP ∪ {(ubp[w_i], w_i)}
14 return BP
```

from the men's point of view. Then, let $w_j$ be in another blocking pair, say $(m_z, w_j)$, such that $(m_z, w_j)$ dominates $(m_i, w_j)$ from the women's point of view. In this situation, UB1 returns $(m_z, w_j)$. The elimination of this blocking pair automatically eliminates $(m_i, w_j)$ from the marriage, since it is dominated by $(m_z, w_j)$; however, it does not eliminate the blocking pair $(m_i, w_k)$. We would like to obtain a new algorithm that will also return the blocking pair $(m_i, w_k)$, so to avoid having to consider it again in the subsequent step of the local search algorithm. This is the rationale underlying algorithm UB2.

In UB2, blocking pairs are found by scanning the men's preference lists. At each step of such a scan, array *pos* is used to contain, for each man, the position of the next woman in men's preference lists. The Boolean array $fnd$ tells us, for each man, if he is in an undominated blocking pair from the men's point of view. Finally, array *ubp* contains, for each woman $w$, the man $m$ such that $(m, w)$ is an undominated blocking pair from the point of view of $w$.

The set of blocking pairs returned by UB2 is larger than that returned by UB1; however, it still contains at most one blocking pair for each man. Some

**Algorithm 6**: UB2

**Input** : a marriage problem $P$ of size $n$, a marriage $M$
**Output**: a list BP of undominated blocking pairs.

```
1  pos ← array of length n with the men as indices.
2  fnd ← boolean array of length n with the men as indices.
3  ubp ← array of length n with the women as indices.
4  R(m_j, M(m_j)) ← position of woman M(m_j) in m_j's preference
   list.
5  for i=0 to n do ubp[w_i] ← null, pos[m_i] ← 1,
   fnd[m_i] ← false
6  finished ← false, BP ← ∅
7  while !finished do
8      foreach m_j, j = 1..n do
9          for i = pos[m_j] to R(m_j, M(m_j)) and !fnd[m_j] do
10             if (m_j, w_i) is a blocking pair then
11                 if ubp[w_i] = null or w_i prefers m_j to ubp[w_i]
                   then
12                     if w_i prefers m_j to ubp[w_i] then
13                         fnd[ubp[w_i]] = false
14                     ubp[w_i] ← m_j
15                 fnd[m_j] ← true
16         pos[m_j] ← i + 1
17     finished ← true
18     foreach m_i, i = 0..n do
19         if !fnd[m_i] and pos[i] < R(m_i, M(m_i)) then
20             finished ← false
21 for i=0 to n do
22     if ubp[i] ≠ null then
23         BP ← BP ∪ {(ubp[w_i], w_i)}
24 return BP
```

of the added blocking pairs involve men that are not involved in the blocking pairs of UB1, while other added blocking pairs improve the blocking pairs in UB1 from men's point of view. We call SML2 the algorithm obtained from SML by using UB2 to compute BP in line 4.

Since dominance between blocking pairs is defined from one gender's point

of view, at each search step we change the role of the two genders. For example, in UB1, in one step we find the undominated blocking pairs from the men's point of view and, among those, we keep only the undominated blocking pairs from the women's point of view, and in the following step we do the opposite: we find the undominated blocking pairs from the women's point of view and, among those, we keep only the undominated blocking pairs from the men's point of view.

In line 14 of Algorithm SML, the evaluation function counts the number of blocking pairs in each neighboring marriage, to select the most promising one (i.e., the one with smallest number of blocking pairs). Notice that computing the blocking pairs of a SM of size $n$ takes $O(n^2)$ time for every neighbor obtained removing a blocking pair from the current (unstable) marriage.

Summarizing, we have defined three algorithms, called SML, SML1, and SML2, to find a stable marriage for a given SM 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 [37]. 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 marriage, the algorithm performs one or more steps in which we remove a blocking pair. This sequence of blocking pair removal have been shown to converge to a stable marriage with non-zero probability in the context of SMs with incomplete preference lists [63].

## 5.3 Experimental evaluation

We tested our algorithms on randomly generated sets of SM instances. We generated stable marriage problems of size $n$ using the impartial culture model (IC) [34] which assigns to each man and to each woman a preference list uniformly chosen from the $n!$ possible total orders of $n$ persons. This means that the probability of any particular ordering is $1/n!$.

### Experimental results

We measured the performance of our algorithms in terms of number of search steps. For these tests, we generated 100 SMs for each of the following sizes: 100, 200, 300, 400 and 500. In Figure 5.1, we can see that SML2 always takes fewer search steps than SML1 and the difference becomes larger as problem size increases. We do not show results for SML since it is much worse than the others even for small sizes. Thus, from now on, we will focus on SML2.

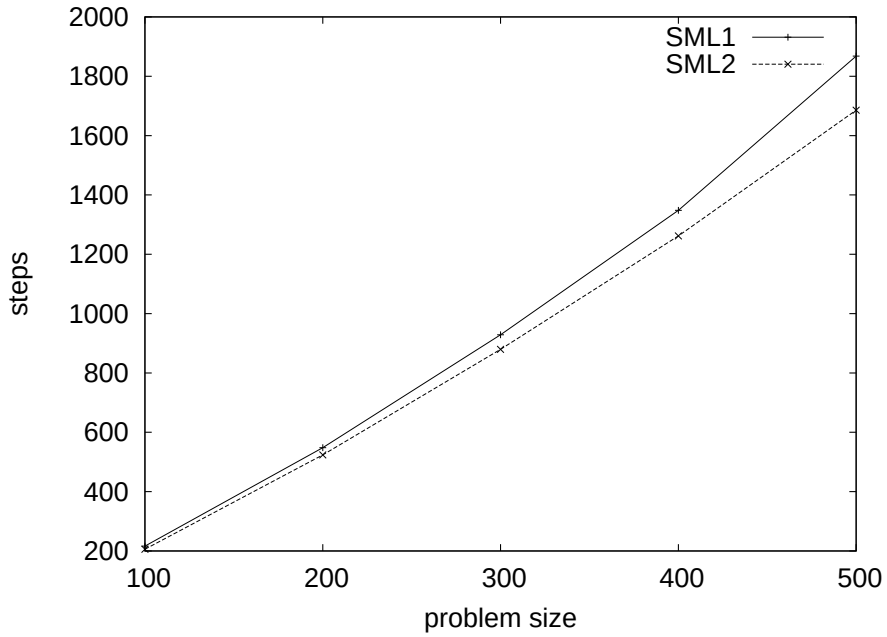



Figure 5.1: Average number of steps for SML1 and SML2.

Figure 5.2 shows the runtime distribution of SML2. We can see that it is able to find a stable marriage for all the problems in the test set within about 3000 steps (notice that, for convenience, we used a logarithmic scale in the x axis). Figure 5.2 also shows that, for each set of problems of the same size, the probability to find a stable marriage grows very fast within a small interval of search steps. This means that it is possible to predict the amount of steps needed by SML2 to find a stable marriage with a reasonable precision. For example, we can say that, given an SM instance of size 100, and running SML2 for 300 steps, we are almost sure that it will find a stable marriage. We also study how fast SML2 converges to a stable marriage, by measuring the ratio between the number of blocking pairs and the size of the problem during the execution. Figure 5.3 shows that SML2 has a very simple scaling behavior. Let us denote by $\langle b \rangle$ the average number of blocking pairs of the marriage found by SML2 for SMs of size $n$ after $t$ steps. Then the experimental results shown in Figure 5.3 show a very good fit with the function

$$\langle b \rangle = an^2 2^{\frac{-bt}{n}} \tag{5.1}$$

where $a$ and $b$ are constants computed empirically ($a \approx 0.25$ and $b \approx 5.7$). In fact, Figure 5.3 shows that the analytical function $\langle b \rangle$ has practically the

same curve as the experimental data. The figure shows also that the average number of blocking pairs, normalized by dividing it by $n$, decreases during the search process in a way that is independent from the size of the problem.

We can use function $\langle b \rangle$ Equation (5.1) to conjecture the runtime behavior of our local search method. Consider the median number of steps, $t_{med}$, taken by SML2. Assume this occurs when half the problems have one blocking pair left and the other half have zero blocking pairs. Thus, $\langle b \rangle = \frac{1}{2}$. Substituting this value in the equation in (5.1), we get:

$$\frac{1}{2} = an^2 2^{\frac{-bt_{med}}{n}}$$

solving for $t_{med}$, and grouping constant terms, we get

$$t_{med} = cn(d + 2log_2(n)) \tag{5.2}$$

Hence, we can conclude that $t_{med}$ grows as $O(nlog(n))$.

We then fit the equation for $t_{med}$ to the experimental data (using $c \approx 0.26$ and $d \approx -5.7$). The result is shown in Figure 5.4, where we can see that the experimental data have the same curve as function $t_{med}$. This means that we can use such an equation to predict the number of steps our algorithms needs to solve a given SM instance.

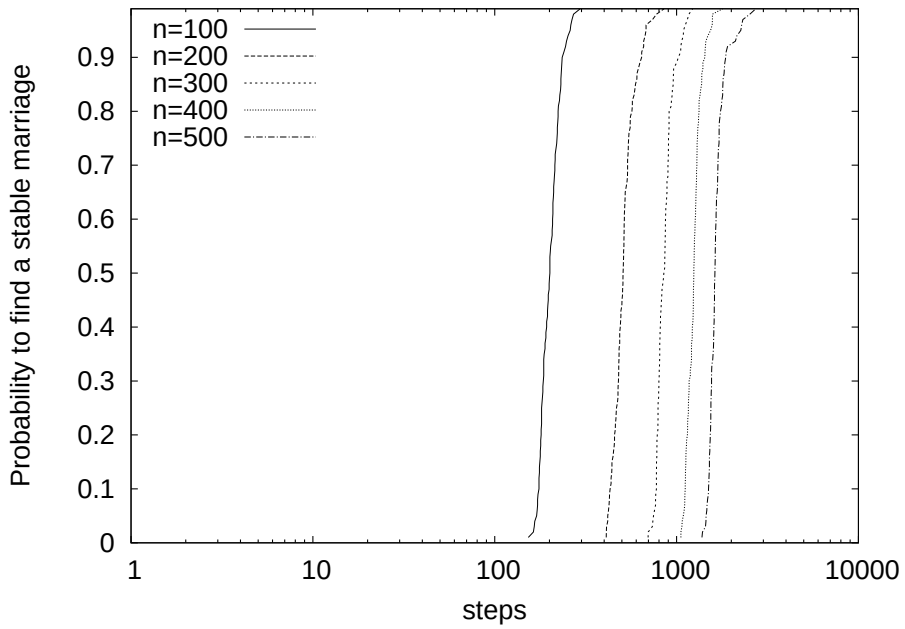



Figure 5.2: Runtime distribution of SML2.

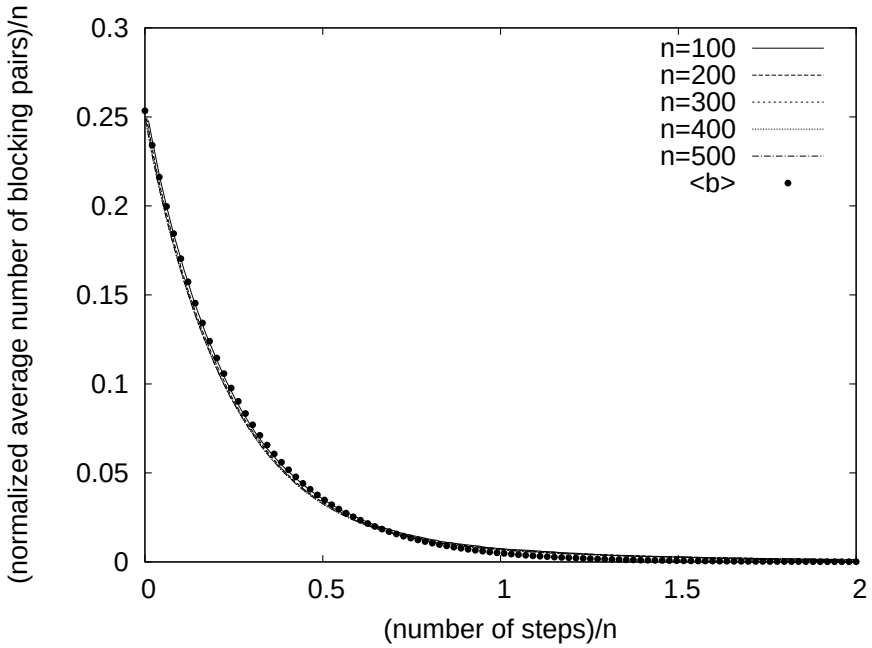



Figure 5.3: Blocking pair ratio during the execution of SML2.

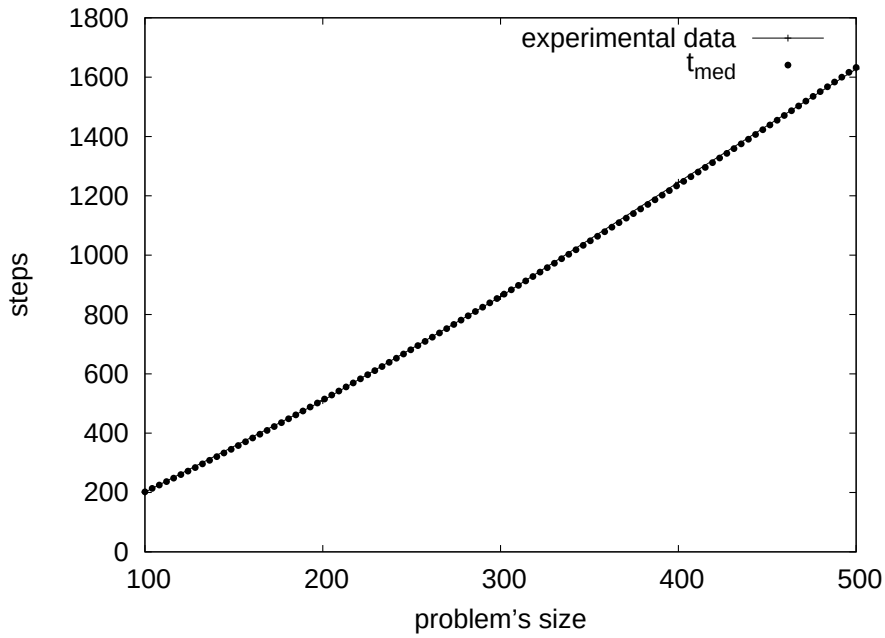



Figure 5.4: Number of steps for SML2.

## Sampling the stable marriage lattice

We now evaluate the sampling capability of SML2 over the lattice of stable marriages of a given SM. To do this, we randomly generated 100 SM instances for each size between 10 and 100, with step 10. Then, we ran the SML2 algorithm 500 times on each instance.

To evaluate the sampling capabilities of SML2, we first measure the distance of the found stable marriages (on average) from the male-optimal marriage (the one that would be returned by the GS algorithm).

Given an SM $P$, consider a stable marriage $M$ for $P$. The distance of $M$ from $M_m$ is the number of arcs from $M$ to $M_m$ in the Hasse diagram of the stable marriage lattice for $P$, and will be denoted by $d_m(M, P)$. We will instead denote with $d_{mw}(P)$ the distance between the male and female optimal for $P$. For each SM instance, we compute the average normalized distance from the male-optimal marriage considering 500 runs. Notice that normalizations is needed since different SM instances with the same size may have a different number of stable marriages in their lattice. Then, we compute the average $D_m$ of these distances over all the 100 problems with the same size, which is therefore formally defined as follows:

$$D_m = \frac{1}{100} \sum_{j=1}^{100} \frac{1}{500} \sum_{i=1}^{500} \frac{d_m(M_i, P_j)}{d_{mw}(P_j)} \tag{5.3}$$

If $D_m$ is equal to 0, it means that all the stable marriages returned coincide with the man-optimal marriage. On the other extreme, if $D_m = 1$, it means that all stable marriages returned coincide with the female-optimal one. Figure 5.5 shows that, for the stable marriages returned by algorithm SML2, the average distance from the male-optimal is around 0.5. This is encouraging but not very informative, since also an algorithm which always returns the same stable marriage, with distance 0.5 from the male-optimal, would have $D_m = 0.5$. The same is for an algorithm that returns the male-optimal half the times and the female-optimal the other half.

To have more informative results on the sampling capabilities of SML2, we consider the entropy of SML2, that is, the uncertainty associated with the outcomes of the algorithm. Let $f(M_i)$ the frequency that SML2 finds a marriage $M_i$ for an SM instance $P$, that is:

$$f(M_i) = \frac{1}{500} \sum_{j=1}^{500} \mathbf{1}_{M_i}(c_j) \tag{5.4}$$

where $\mathbf{1}_{M_i}$ is the indicator function that returns 1 if in the $c$-th execution the algorithm finds $M_i$, and 0 otherwise. The entropy $E(P)$ for each SM

instance $P$ (i.e., for each lattice) of size $n$ is then:

$$E(P) = -\sum_{i=1 \in \{1..|S|\}} f(M_i) log_2(f(M_i)) \tag{5.5}$$

where $S$ is the set of all possible stable marriages of $P$. In an ideal case, when each node in the stable marriage lattice has a uniform probability of $1/|S|$ to be reached, the entropy is $log_2(|S|)$. On the other hand, the worst case is when it is returned always the same stable marriage, and the entropy is thus 0. Since we want a measure that is independent of the problem's size, we consider a normalized entropy, that is $E(P)/log_2(|S|)$, which is in [0,1].

As we have 100 different problems for each size, we compute the average of the normalized entropies for each class of problems with the same size:

$$E_n = \frac{1}{100} \sum_{i=1}^{100} E(P_i)/log_2(|S_i|) \tag{5.6}$$

where $S_i$ is the set of stable marriages of $P_i$.

Figure 5.6 shows that SML2 is not far from the ideal behavior. In fact, the normalized entropy starts from a value of 0.85 at size 10, decreasing to above 0.6 as the problem's size grows.

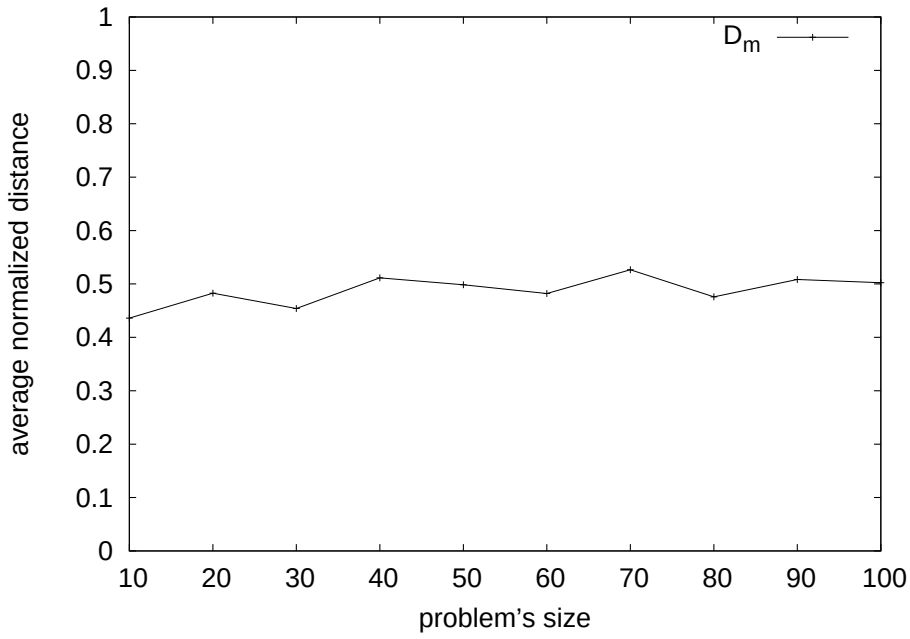



Figure 5.5: Average normalized distance $D_m$ for SML2.

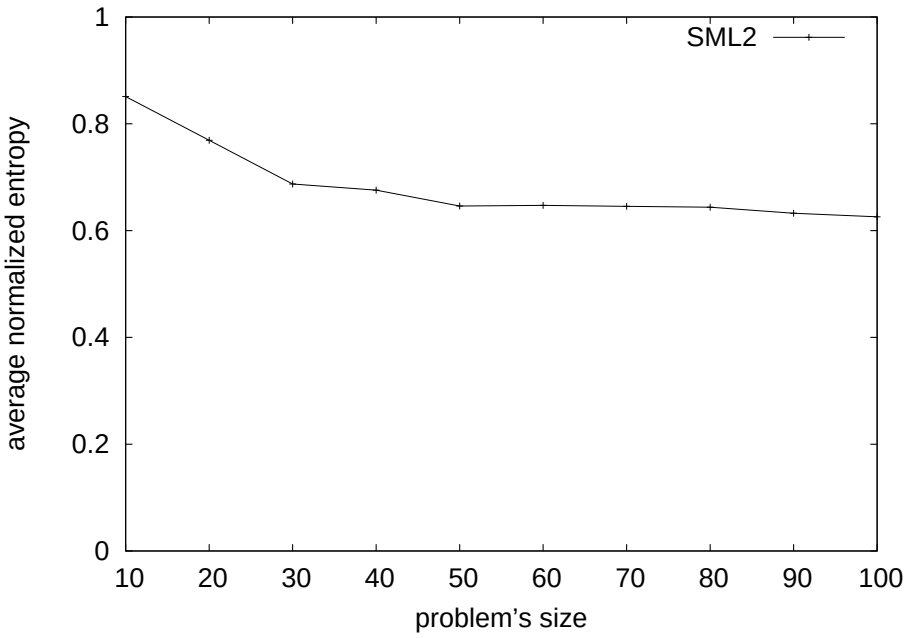



Figure 5.6: Entropy of SML2.

Considering both Figures 5.5 and 5.6, we can claim that SML2 samples the stable marriage lattice very well. In fact, the entropy tells us that the uncertainty to find a specific stable marriage is very high. Considering also the distance $D_m$ (Figure 5.5), the possible outcomes appear to be equally distributed along the paths from the top to the bottom of the lattice.

### Comparing to a Markov chain approach

Markov chain Monte Carlo methods [3] are a class of algorithms for sampling from probability distributions based on constructing a Markov chain with the desired distribution as its equilibrium distribution. A Markov chain is a discrete random process with the property that, given the present state, the future does not depend on the past.

Roughly speaking, the principle is to build a succession of states, and once convergence is reached, the consecutive states are assumed to be drawn from the target probability distribution. With these methods, it is possible to sample from general probability distributions.

The difficult task in sampling using such methods is not constructing the Markov chain but instead is to determine how many steps are needed to converge to the stationary distribution within an acceptable error. If the chain reaches the desired distribution quickly starting from an arbitrary

state, the chain is said to have a *rapid mixing* time. The mixing time $\tau(\epsilon)$ of a Markov chain is the time to come within $\epsilon$ total variation distance (that is the largest possible difference between the probabilities that two probability distributions can assign to the same event) of its stationery distributions (i.e. the desired distribution to converge to). A Markov chain is rapidly mixing if the mixing time is upper bounded by a polynomial. If the mixing time is exponential, the chain is *slowly mixing*. In [4], Bhatnagar et al. use a Markov chain approach to sample the stable marriage lattice for different families of allowable preference sets.

Since the aim is to sample the stable marriage lattice with uniform probability, they define the Markov chain by using rotations exposed in each stable marriage. More precisely, suppose that $M_i$ is current marriage. Then the next marriage $M_{i+1}$ is computed follows:

- with probability 1/3: it randomly chooses a man and, if he is part of a woman-improving rotation $\rho$, it moves to $M_{i+1} = M_i/\rho$;
- with probability 1/3: it randomly chooses a man and, if he is part of a man-improving rotation $\rho$, it moves to $M_{i+1} = M_i/\rho$;
- with probability 1/3, it moves to $M_{i+1} = M_i$.

Since a rotation and its inverse contain the same people, and the probability of picking a particular rotation is proportional to the number of couples it contains, this Markov chain is reversible. Since this chain is aperiodic (i.e., returns to state $i$ can occur at irregular times) and irreducible (i.e., it connects the state space of the stable marriages), it converges to the uniform distribution over the stable marriages.

In [4] the authors show that this Markov chain approach has an exponential convergence time in their test sets.

We compared algorithm SML2 with the Markov chain random walk algorithm (MC) just defined. We first plot the entropy and distance from the male-optimal (in Figures 5.7 and 5.8 respectively) of MC computed on executions varying the number of steps from 10 to 200. We can see the the entropy of MC increases quite rapidly but the distance from the top of the lattice increases more slowly.

For each problem instance in the test set, we started the MC algorithm from the male optimal marriage and took the stable marriage returned by MC after exactly the same number of steps needed by SML2 to find a stable marriage for that instance. Then we measured the entropy and the distance from the male optimal for MC, and compared it to those of SML2. The results are shown in Figure 5.9, where we can see that the entropy of MC is

roughly the same as that of SML2. On the other hand, if we consider the distance from the male-optimal, the marriages reached by MC are closer to the top of the lattice than the ones found by SML2. In fact, the average distance from the male-optimal marriage of MC is 0.2, while it is 0.5 for SML2.

Summarizing, we can say that SML2 has sampling capabilities comparable with the Markov chain approach considering the same number of steps, and may even perform slightly better considering the distance measured from the top or bottom of the lattice.

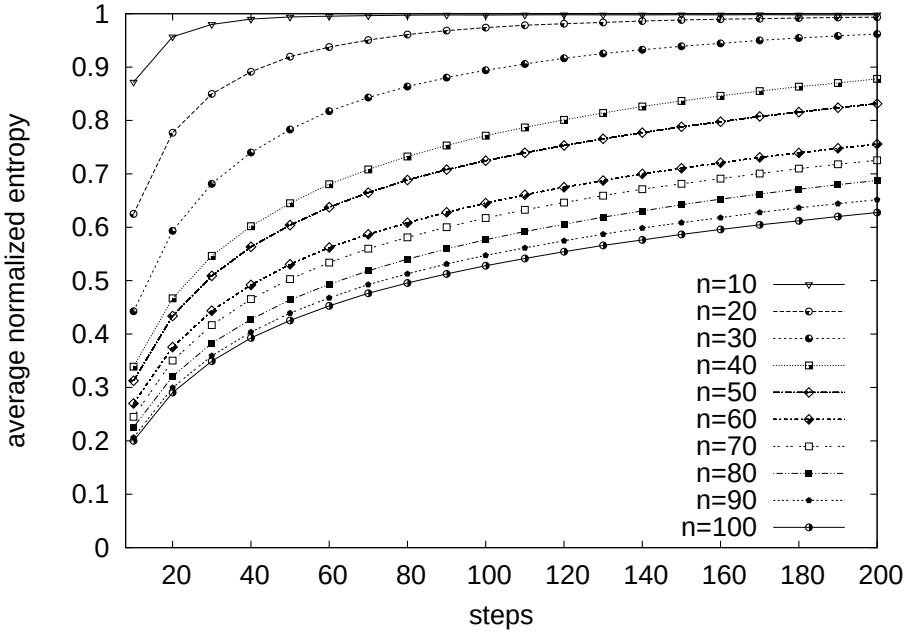



Figure 5.7: Average runtime entropy of MC.

## 5.4 Related work

In this chapter we consider the fairness of the methodology to generate stable marriages. Other works have considered the fairness with the meaning of finding a stable marriages where the overall happiness of the persons is maximized. One kind of fairer stable marriage that have been considered, is the *minimum regret* stable marriage [47, 35]. The regret for each person is the position in his/her preference list of the persons to whom he/she is married. The regret of a marriage $M$ is the maximum regret of any person. Another way characterize the overall happiness of a marriage is to consider sum the

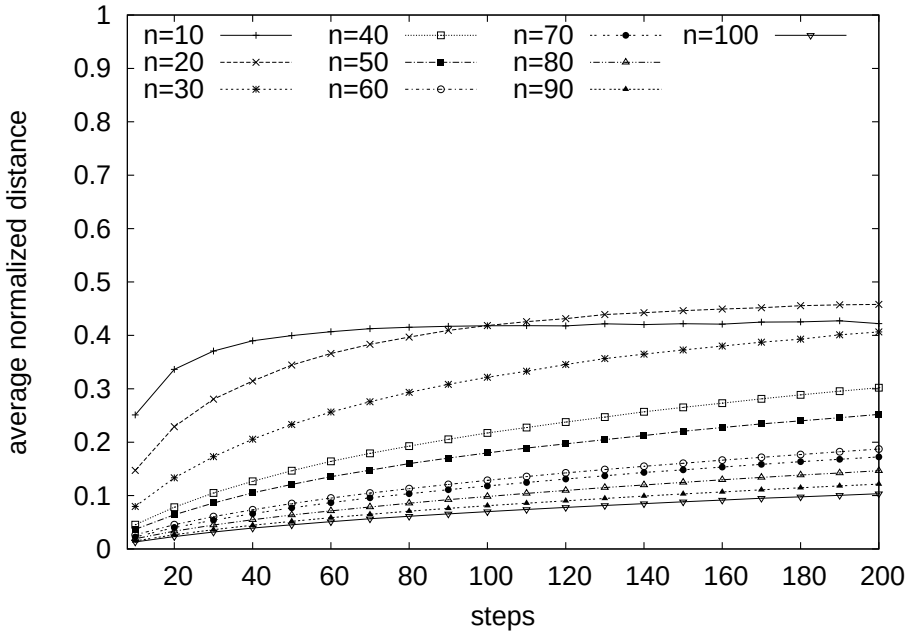



Figure 5.8: Average runtime distance from the male-optimal of MC.

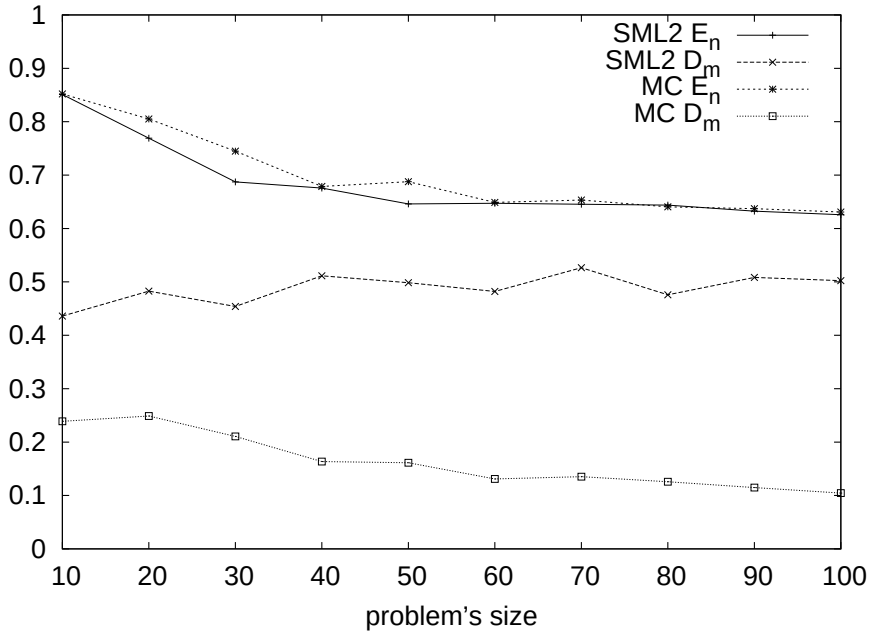



Figure 5.9: SML2 compared with MC.

regret of every person. The *egalitarian* stable marriage [42] minimize the total sum of the regrets. Both minimum regret and egalitarian stable marriage can be found in polynomial time [35, 42].

In [63] Roth and Vande Vate show that, beginning from an arbitrary marriage, and removing a blocking pair at random, we will eventually reach a stable marriage with probability one. Our local search approach exploit this result by building sequence of blocking pairs removal that rapidly lead to stability thanks to the use of undominated blocking pairs.

## 5.5 Conclusions

In this chapter we presented a local search approach to fairly generate stable marriages. Removing undominated blocking pairs has increased the convergence speed to stability. Exploiting the use of undominated blocking pairs, our algorithm has a size independent behaviour which appears to scale well. Moreover, it is able to find a stable marriage in a number of steps which grows as $O(nlog(n))$. We have also shown that our approach is able to sample the stable marriage lattice reasonably well, also when compared with a Markov chain approach.

Finally we have to notice that the algorithms proposed in this chapter can be adapted to stable marriage problems with ties and stable marriage problems with incomplete lists very easy. In fact, the only thing to do is using the appropriate the definition of blocking pair and the whole algorithmic skeleton remain the same.

# Chapter 6

# Incompleteness and imprecision in stable marriages

In this chapter we consider variants of the stable marriage problem where some form of imprecision and incompleteness is allowed for modeling agents preferences. At the light of the results of the previous chapter, we study the effectiveness of a local search approach in finding a maximum cardinality stable marriage. Furthermore, we study male optimality and uniqueness of stable marriages in stable marriage problems with ties. In particular, we give an algorithm to find stable marriages that are male optimal and we give sufficient conditions on the preferences which guarantee the uniqueness of a stable marriage.

Some of the results in this chapter are included in the following articles:

- *Local search for stable marriage problems with ties and incomplete lists*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 11th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2010), Byoung-Tak Zhang and Mehmet A. Orgun eds., Springer, LNCS 6230, 2010.

- *Local search for stable marriage problems*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, T. Walsh, in proc. of the 3rd International Workshop on Computational Social Choice (COMSOC 2010), Dusseldorf, Germany, September 2010.

- *Male optimality and uniqueness in stable matching problems with partial orders*, Maria Silvia Pini, Francesca Rossi, Toby Walsh, Mirco Gelain, Kristen Brent Venable, in proc. of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), IFAAMAS Press, 2010, short paper.

- *Male optimal and unique stable marriages with partially ordered preferences*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh, in proc. of the International Workshop on Collaborative Agents - REsearch and development (CARE 2009/2010), Springer LNAI 6066, to appear.

- *Male optimal and unique stable marriages with partially ordered preferences*, M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh, in proc. of the International Workshop on Collaborative Agents – REsearch and Development (CARE 2009), Melbourne, Australia, December 2009.

## 6.1 Motivations

There are many variants of the traditional formulation of the stable marriage problem. Some of the most useful in practice include incomplete preference lists (SMI), that allow us to model unacceptability of one person for certain members of the other sex, and preference lists with ties (SMT), that model some king of vagueness or imprecision in preference ordering.

Unfortunately, when we allow for both ties and incomplete preference lists, the problem, called stable marriage problem with ties and incomplete lists (SMTI) becomes NP-hard [54]. Since there may be several stable marriages of different sizes, solving such a problem means finding a stable marriage of maximum size.

In this context we aim to investigate if a local search approach, similar to the one used to solve the stable marriage problem in Chapter 5, which exploits properties of the problem to reduce the size of the neighborhood and to make local moves efficiently, is effective in solving such NP-hard problem.

In the context of centralized matching scheme, some participating hospitals with many applicants have found the task of producing a strictly ordered preference list difficult, and have expressed a desire to use ties [44]. Ties also naturally occur when assigning students to schools, since many students are indistinguishable from the point of view of a given school.

We also consider cases where men and women can express their preferences via partial orders allowing ties or saying that two or more people are incomparable. More precisely, we study male optimality and uniqueness of solution in this more general context. Male optimality can be a useful property since it allows us to give priority to one gender over the other. For example, in matching residents to hospitals in the US, priority is given to the residents. We present an algorithm, based on an extended version of the

Gale-Shapely algorithm, to find a male optimal solution in stable marriage problems with partially ordered preferences (SMPs).

The uniqueness of a male optimal solution is another interesting concept. For instance, it guarantees that the solution is optimal since it is as good as possible for all the participating agents. Uniqueness is also a barrier against manipulation. This is important as all stable marriage procedures can be manipulated. In [18] sufficient conditions on the preferences are given, that guarantee uniqueness of stable marriages when only strictly ordered preferences are allowed. Such conditions identify classes of preferences that are broad and of particular interest in many real-life scenarios [14]. We show that it is possible to generalize these sufficient conditions for uniqueness to SMs with partially ordered preferences, by considering in some cases uniqueness up to indifference and incomparability.

## 6.2 Local search by removing blocking pairs

We adapt the classical local search schema to SMTI problems as follows. Given a SMTI problem $P$, we start from a randomly generated marriage $M$ for $P$. At each search step, we move to a new marriage in the neighborhood of the current one. For each marriage $M$, the neighborhood $N(M)$ is the set of all marriages obtained by removing one blocking pair from $M$. Consider a blocking pair $bp = (m, w)$ in $M$ and assume $m' = M(w)$ and $w' = M(m)$. Then, removing $bp$ from $M$ (written $M \backslash bp$) means obtaining a marriage $M'$ in which $m$ is married with $w$ and both $m'$ and $w'$ become single, leaving the other pairs in the marriage $M$ unchanged. Notice that, if $M$ is stable, its neighborhood is empty. Notice also that this notion of neighborhood is not symmetric.

To select the neighbor to move to, we use an evaluation function $f : \mathcal{M}_n \rightarrow Z$, where $\mathcal{M}_n$ is the set of all possible marriages of size $n$, and $f(M) = nbp(M) + ns(M)$. For each marriage $M$, $nbp(M)$ is the number of blocking pairs in $M$, while $ns(M)$ is the number of singles in $M$ which are not in any blocking pair. The algorithm moves to a marriage $M' \in N(M)$ such that $f(M') \leq f(M'') \; \forall M'' \in N(M)$.

During the search, the algorithm maintains the best marriage found so far, defined as follows: if no stable marriage has been found, then the best marriage is the one with the smallest value of the evaluation function; otherwise, it is the stable marriage with less singles.

To avoid stagnation in a local minimum of the evaluation function, at each search step we perform a random walk with probability $p$ (where $p$ is a parameter of the algorithm). In the random walk, we move to a randomly

selected marriage in the neighborhood (we tried also to move to a generic random marriage, but this gave worse behavior). If a stable marriage is reached, its neighborhood is empty and a random restart is performed.

The algorithm terminates if a perfect marriage (that is, a stable marriage with no singles) is found, or when a maximal number of search steps is reached. Upon termination, the algorithm returns the best marriage found during the search.

The pseudo-code of our algorithm, called LTI, is shown in Algorithm 7. In the pseudo-code, $M_{best}$ is the best marriage found so far, and $f_{best}$ its evaluation (number of blocking pairs plus number of singles). Function *best_neighbor* returns one of the best marriages in the neighborhood of the current marriage, according to the evaluation function.

In addition to this simple local search algorithm which directly applies standard local search approaches to SMTI problems, we have also designed a more sophisticated algorithm which has been tailored to exploit the specific features of SMTI problems. The main difference is in the definition of the neighborhood, which refers to the notion of *undominated* blocking pairs that we already seen in Chapter 5 and we recall here.

**Definition 47** (dominance in blocking pairs)**.** *Let $(m, w)$ and $(m, w')$ two blocking pairs. Then $(m, w)$ dominates (from the men's point of view) $(m, w')$ if $m$ prefers $w$ to $w'$. There is an equivalent concept from the women's point of view.*

**Definition 48** (undominated blocking pair)**.** *An men- (resp., women-) undominated blocking pair is a blocking pair such that there is no other blocking pair that dominates it from the men's (resp., women's) point of view. When the point of view (men or women) is clear or not important, we will omit it.*

For example, consider the SMTI problem in Table 6.1, the marriage 1 2 3 4, and two blocking pairs $(m_1, w_2)$ and $(m_4, w_2)$. Using the definitions above, $(m_1, w_2)$ dominates $(m_4, w_2)$ from the women's point of view. If we remove $(m_4, w_2)$ from the marriage, $(m_1, w_2)$ will remain. On the other hand, removing $(m_1, w_2)$ also eliminates $(m_4, w_2)$. Thus removing undominated blocking pairs may reduce the number of blocking pairs more than eliminating dominated pairs.

We call LTIU the algorithm LTI where the neighborhood is defined as the set of marriages obtained from the current one by removing any dominated blocking pair. More precisely, at each step we consider the undominated blocking pairs from the men's point of view which are also undominated from women's point of view. Then, in the next step, we swap genders and

**Algorithm 7**: LTI

**input** : a SMTI problem $P$, an integer $max\_steps$
**output**: a marriage

```
1  M ← random marriage
2  steps ← 0
3  M_best ← M
4  f_best ← f(M)
5  repeat
6      if f(M) = 0 then
7          return M
8      if rand() ≤ p then
9          M ← RandomWalk(M)
10     else
11         PAIRS ← blocking pairs in M
12         if PAIRS is empty then
13             perform a random restart
14         else
15             M ← best_neighbor(M, PAIRS)
16     if M is the first stable marriage found so far then
17         f_best ← f(M)
18         M_best ← M
19     if M_best is not stable and f_best > f(M) then
20         f_best ← f(M)
21         M_best ← M
22     if both M_best and M are stable and f_best > f(M) then
23         f_best ← f(M)
24         M_best ← M
25     steps ← steps + 1
26 until steps ≥ max_steps ;
27 return M_best
```

do the same, to ensure gender neutrality in our algorithm[1].

Due to their ability to restart, our algorithms have the PAC (probabilistically approximate complete property) [37]. That is, as their runtime goes

[1]Gender neutrality is usually considered a desirable feature in a stable marriage procedure.

| men's preference lists | women's preference lists |
|---|---|
| 1: 2 1 | 1: 3 1 (2 4) |
| 2: 2 (3 4) | 2: 1 4 2 |
| 3: (1 2 3 4) | 3: (1 2) (4 3) |
| 4: (3 2) 1 4 | 4: (3 2 4) |

Table 6.1: An example of a SMTI problem of size 4.

to infinity, the probability that the algorithm returns an optimal solution goes to one. If the algorithm starts at a stable marriage, the algorithms will perform a random restart, which will end up in an optimal solution with probability greater than zero. On the other hand, if the algorithm starts from a non-stable marriage, we perform one or more steps in which we remove a blocking pair. This sequences of blocking pair removal have been shown to converge to a stable marriage with non-zero probability in the context of SMs with incomplete preference lists [63]. The proof of this result can be adapted to our context, as we have ties in the preference lists. Since a stable marriage can be reached with non-zero probability, and as we have argued above that from any stable marriage random restarting will reach an optimal solution with non-zero probability, the PAC property holds.

## 6.3 Experimental evaluation

We tested our algorithms on randomly generated sets of SMTI instances. We generated problems using the same method as in [30]. More precisely, the generator takes three parameters: the problem's size $n$, the probability of incompleteness $p_1$, and the probability of ties $p_2$. Given a triple $(n, p_1, p_2)$, a SMTI problem with $n$ men and $n$ women is generated, as follows:

1. For each man and woman, we generate a random preference list of size $n$, i.e., a permutation of $n$ people;

2. We iterate over each man's preference list: for a man $m_i$ and for each women $w_j$ in his preference list, with probability $p_1$ we delete $w_j$ from $m_i$'s preference list and $m_i$ from $w_j$'s preference list. In this way we get a possibly incomplete preference list.

3. If any man or woman has an empty preference list, we discard the problem and go to step 1.

4. We iterate over each person's (men and women's) preference list as follows: for a man $m_i$ and for each woman in his preference list, in position $j \geq 2$, with probability $p_2$ we set the preference for that woman as the preference for the woman in position $j-1$ (thus putting the two women in a tie).

Note that this method generates SMTI problems in which the acceptance is symmetric. If a man $m$ does not accept a woman $w$, $m$ is removed from $w$'s preference list as well. This does not introduce any loss of generality because $m$ and $w$ cannot be matched together in any stable marriage.

Notice also that this generator will not construct a SMTI problem in which a man (resp., woman) accepts only women (resp., men) who do not find him (resp, her) acceptable. Such a man (resp., woman) will remain single in every stable matching. A simple preprocessing step can remove such men and women from any problem, giving a smaller instance of the form constructed by our generator.

Moreover, we plan a test to see if Tabu search could be useful if applied to our algorithms. We use a buffer to store the last 1000 marriages we touch and we count how many times we hit a marriage in the buffer also distinguishing if it is a local minima or not.

## Experimental results

We run our experiments on 2 x Quad-Core AMD Opteron 2.3GHz CPU with 2GB of RAM. In practice we used ony one core because our algorithm is not designed for multi threading.

We first analyzed the behavior of the base algorithm, LTI. Unfortunately this algorithm fails to find a stable marriage in most of our test problems (see Figure 6.1). In fact, LTI can find a stable marriage only for easy problems, where there are many ties (that is, $p_2$ high) and/or a lot of incompleteness (that is, $p_1$ high).

On the other hand, algorithm LTIU finds a stable marriage in 100% of the runs. Since stability is essential in our context, from now on we will only show the experimental results for algorithm LTIU.

We start by showing the average size of the marriages returned by LTIU. In Figure 6.2 we can see that LTIU finds a perfect marriage (that is, a stable marriage with no singles) almost always. Even in settings with a large amount of incompleteness (that is, $p_1 = 0.7$ - $0.8$) the algorithm finds very large marriages, with only 2 singles on average.

We also consider the number of steps needed by our algorithm. From Figure 6.3(a), we can see that the number of steps is less than 2000 most

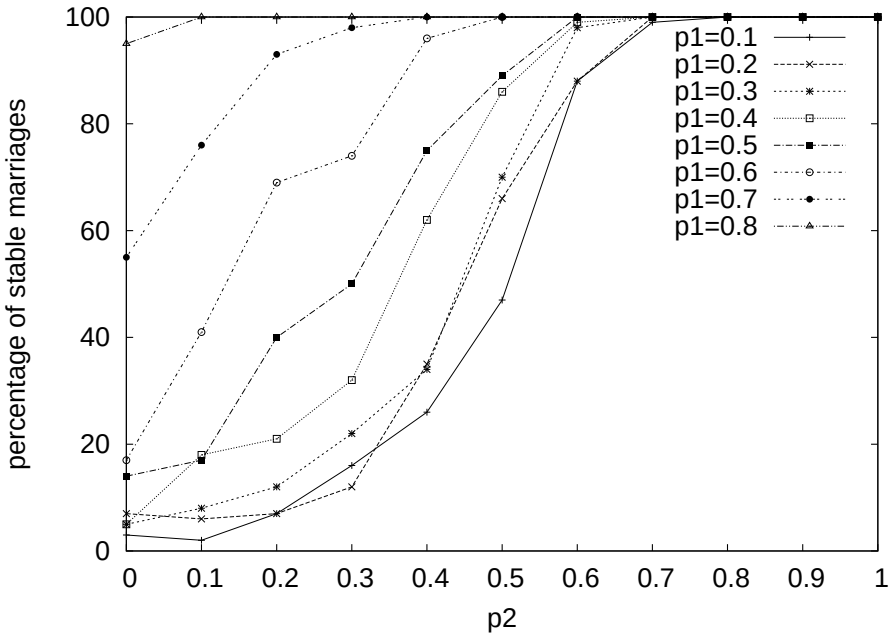



Figure 6.1: Average number of stable marriages found by LTI.

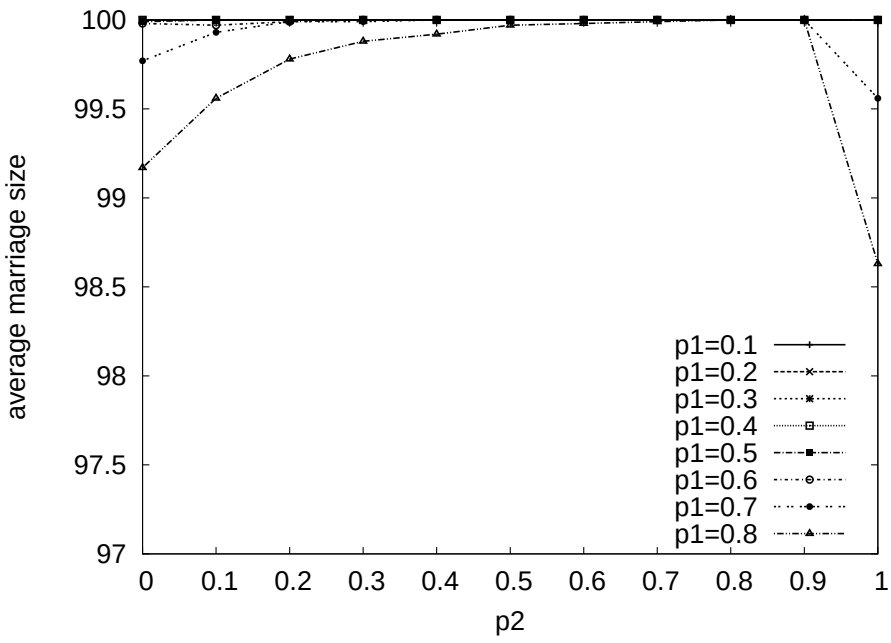



Figure 6.2: Average size of marriages with LTIU.

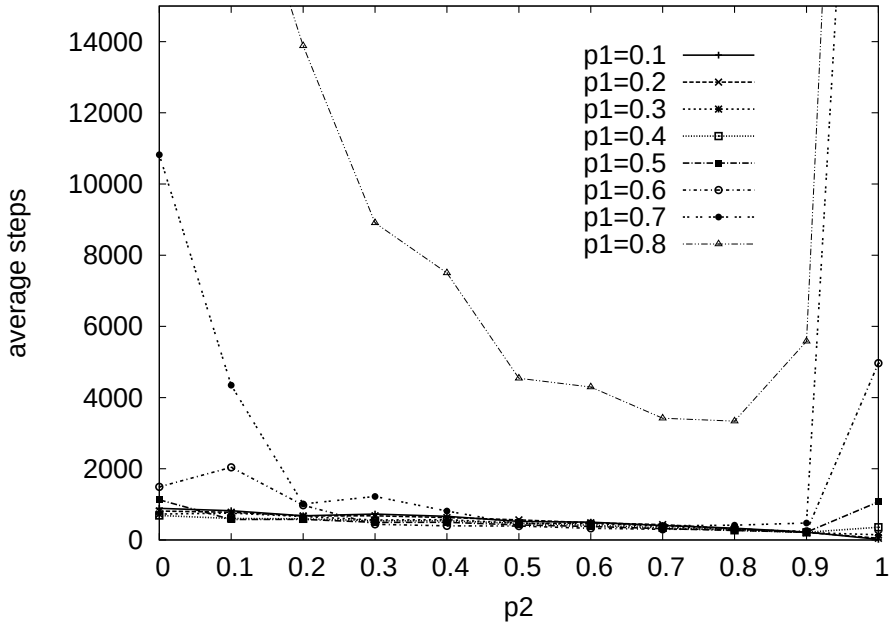
of the time, except for problems with a large amount of incompleteness (i.e. $p_1 = 0.8$). As expected, with $p_1$ greater than 0.6, the algorithm requires more steps. In some cases, it reaches the step limit of 50000. Moreover, as the percentage of ties rises, stability becomes easier to achieve and thus the number of steps tends to decrease slightly. We note that complete indifference (i.e. $p_2$=1) is a special case. In fact, in this situation, the number of steps increases for almost every value of $p_1$. This is because the algorithm makes most of its progress via random restarts. In these problems every person (if accepted) is equally preferred to all the others accepted. This means that the only blocking pairs are those involving singles who both accept each other. In this situation, after a few steps all singles that can be married are matched, stability is reached, and the neighborhood becomes empty. The algorithm therefore performs another random restart. It is therefore very difficult to reach a perfect matching and the algorithm often runs until the step limit.

The algorithm is fast. It takes, on average, less than 40 seconds to give a result even for problems with a lot of incompleteness (see Figure 6.3(b)). As expected, with $p_2 = 1$ the time increases for the same reason discussed above concerning the number of steps.

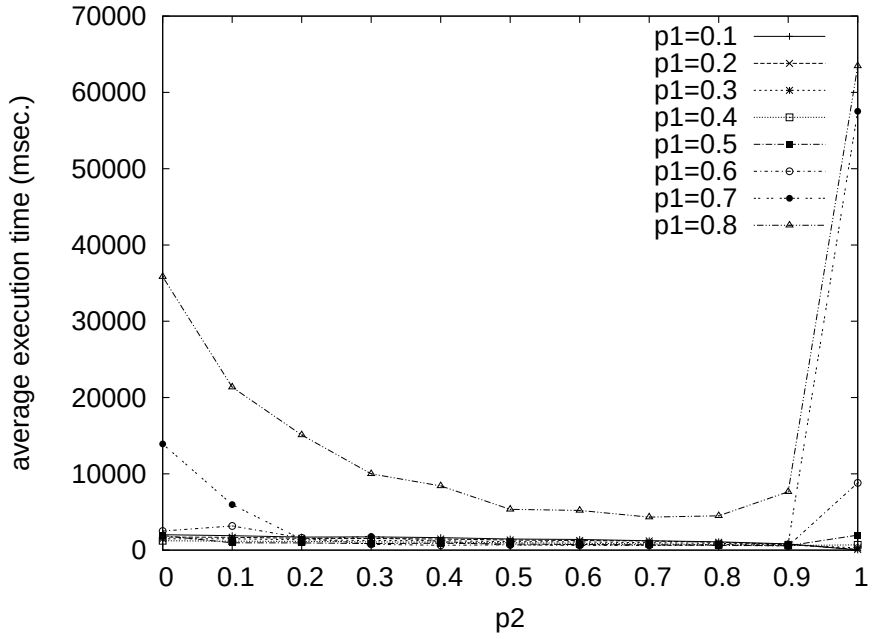
Re-considering Figure 6.2 and the fact that all the marriages the algorithm finds are stable, we notice that most of the marriages are perfect. From Figure 6.4 we see that the average percentage of matchings that are perfect is almost always 100% and this percentage only decreases when the incompleteness is large.

We compared our local search approach to the complete method from [30]. In their experiments, they measured the maximum size of the stable marriages in problems of size 10, fixing $p_1$ to 0.5 and varying $p_2$ in [0,1]. We did the same experiments (generating new instances), and obtained stable marriages of a very similar size to those reported in [30]. This means that although our algorithm is incomplete in principle, it always finds an optimal solution in practice, and for small sizes it behaves as a complete algorithm in terms of size of the returned marriage. However, we can also tackle problems of much larger sizes (at least 100), still obtaining optimal solutions most of the times.

We also considered the runtime behavior of our algorithm. In Figure 6.5 we show the average normalized number of blocking pairs and, in Figure 6.6, the average normalized number singles of the best marriage as the execution proceeds. Although the step limit is 50000, we only plot results for the first steps because the rest is a long plateau that is not very interesting. We shows the results only for $p_2 = 0.5$. However, for greater (resp., lower) number of ties the curves are shifted slightly down (resp., up). From Figure 6.5 we can see that the average number of blocking pairs decreases very fast, reaching




(a) average number of steps




(b) average execution time

Figure 6.3: Average number of steps and execution time for LTIU.

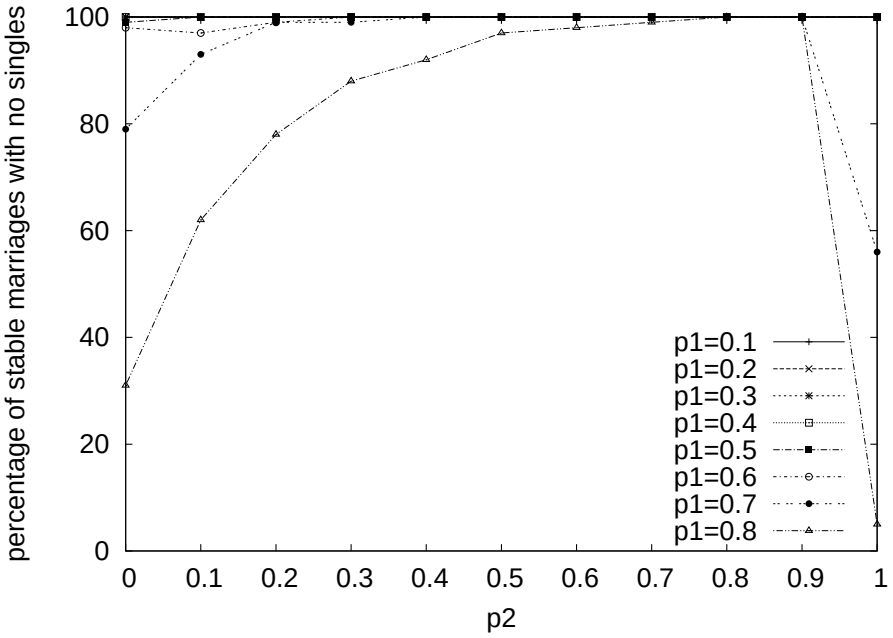



Figure 6.4: Percentage of perfect matchings.

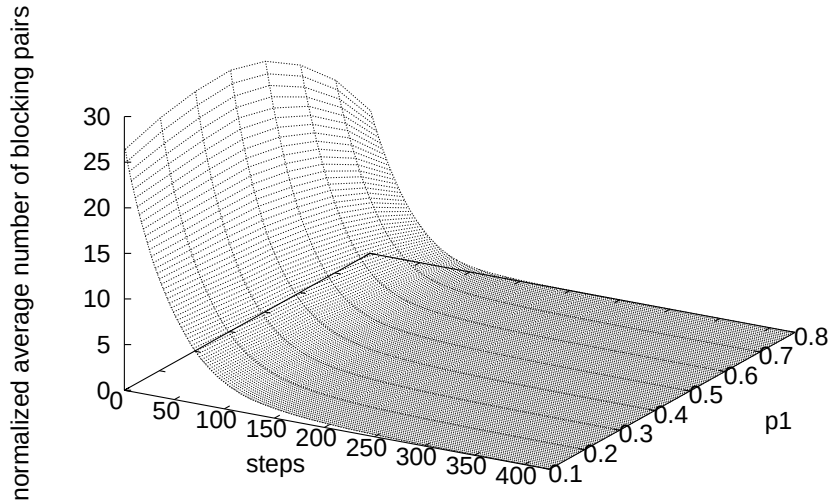



Figure 6.5: Normalized average number of blocking pairs ($p_2$=0.5).

5 blocking pairs after only 100 steps. Then, after 300-400 steps, we reach a stable marriage almost all the times for all values of $p_1$. Considering Figure 6.6, we can see that the algorithm starts with more singles for greater values of $p_1$. This happens because, with more incompleteness, it is more unprobable for a person to be accepted. However, after 200 steps, the average number of singles becomes very small no matter the incompleteness in the problem.

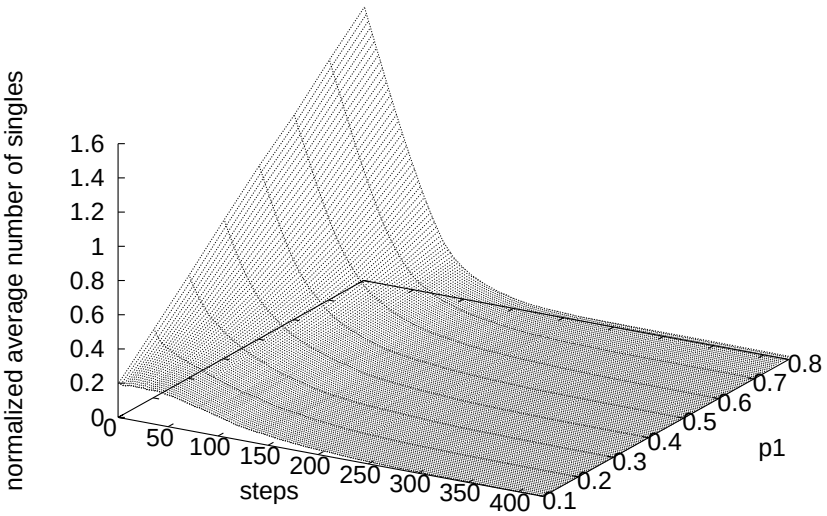



Figure 6.6: Normalized average number of singles ($p_2$=0.5).

In order to test if the introduction of some diversity to the search would be useful, we consider the choice of selecting undominated blocking pairs with a probability $u$ when we build the neighborhood. More precisely, at each search step, with probability $u$ we consider the neighborhood obtained by removing undominated blocking pairs and, with probability $1 - u$ we consider the neighborhood obtained by removing dominated blocking pairs. Comparing Figures 6.5 and 6.7 we can see that, removing a dominated blocking pair with probability 0.5 make the search a bit slower to converge to stability. In fact, after 350-400 steps, the stability is not reached yet.

Looking at both Figures 6.5 and 6.6, we observe that, although we set a step limit $s = 50000$, the algorithm reaches a very good solution after just 300-400 steps. In fact, after this number of steps, the best marriage found by the algorithm usually has no blocking pairs nor singles. This appears largely independent of the amount of incompleteness and the number of ties in the

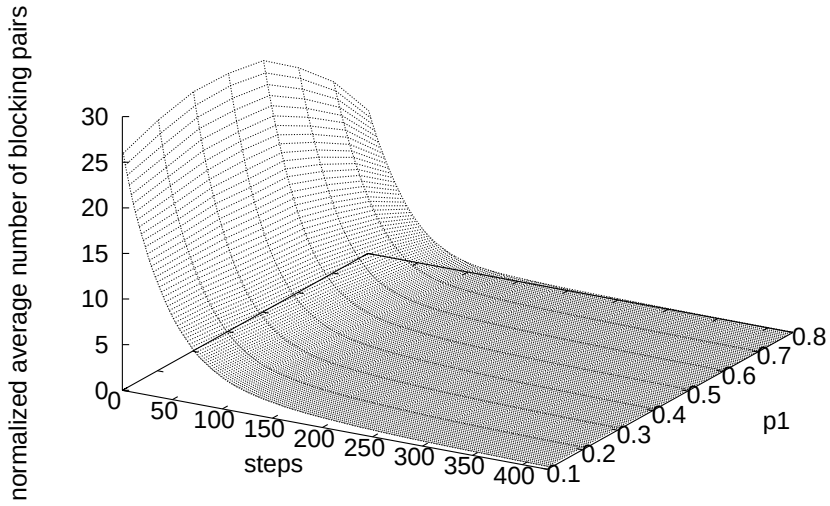



Figure 6.7: Normalized average number of blocking pairs ($p_2$=0.5, $u$=0.5).

problems. Hence, for SMTI problems of size 100 we could set the step limit to just 400 steps and still be reasonably sure that the algorithm will return a stable marriage with a large size, no matter the amount of incompleteness and ties.

Finally we test the effectiveness of the use of a short term memory (like Tabu search techniques) to avoid stagnation in local minima. Figure 6.8 shows how many times one of the last 1000 marriages is encountered during the search considering problems of size 50 and running LTIU on 100 problems per size. We can see that we have hits only when the incompleteness is high. In fact, with short lists, the number of possible matchings is not so high thus, the probability of coming back in a n already visited position arises.

We count also the number of times LTIU the already visited marriage is also a local minima (see Figure 6.9). The algorithm hits a local minima less than 10 times (in average) in the last 1000 steps only with $p_1$=0.7-0.8 and considering problems of size 50. We run the same test also on problems of size up to 150 observing that the number of hits is zero in practice. Hence, we can conclude that the use of a short term memory is not needed in this context.

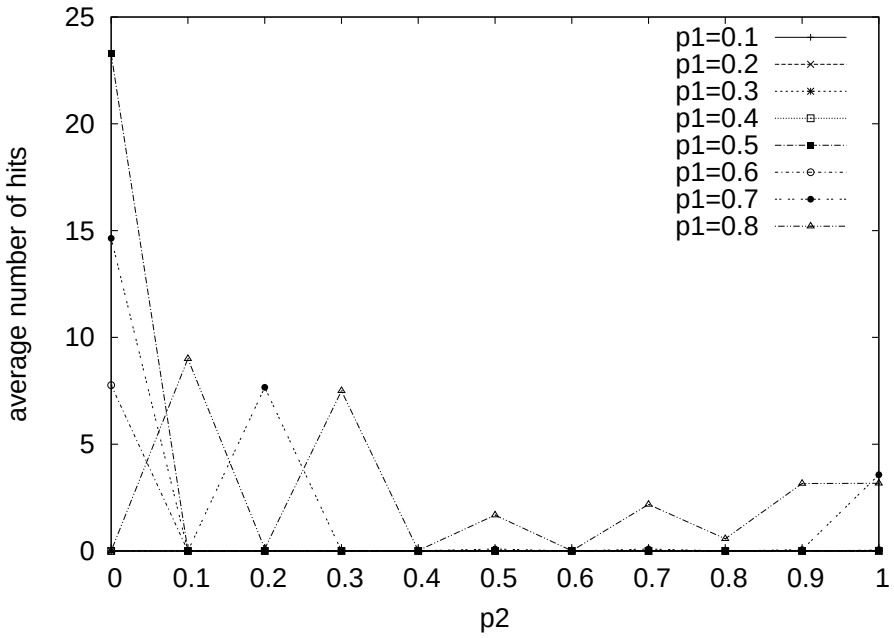



Figure 6.8: Average number of times one of the last 1000 marriages is hit by LTIU (n=50).

## 6.4 Local search by swapping ties

In Section 6.2 we presented two local search algorithms which start from a random marriage and try to converge to a stable marriage with maximum size via a sequence of blocking pairs removal. In this section we present another local search approach [60] to find the largest stable marriage of a given SMTI instance $I$. This approach is based on the observation that, by breaking all ties, $I$ becomes an SMI, say $I'$, and a stable marriage in $I'$ is also stable in $I$, since we are considering weak stability. Furthermore, we recall that all stable marriages of a given SMI have the same size, and one of them can be found in polynomial time using the Gale Shapley algorithm.

More precisely, we consider SMTIs with ties of length two, and we associate a weight in [0,1] to each way of breaking a tie. Initially, such weights are all set to 0.5.

Our approach works as follows: starting from an SMTI instance $P$, it breaks ties in $P$ obtaining an SMI $P'$ and then it finds the male optimal stable marriage of $P'$ using the Gale Shapley algorithm. Then, in the first search step, it moves to another SMI, with larger stable marriages (if possible). To select the problem to move to, our approach considers the neighborhood

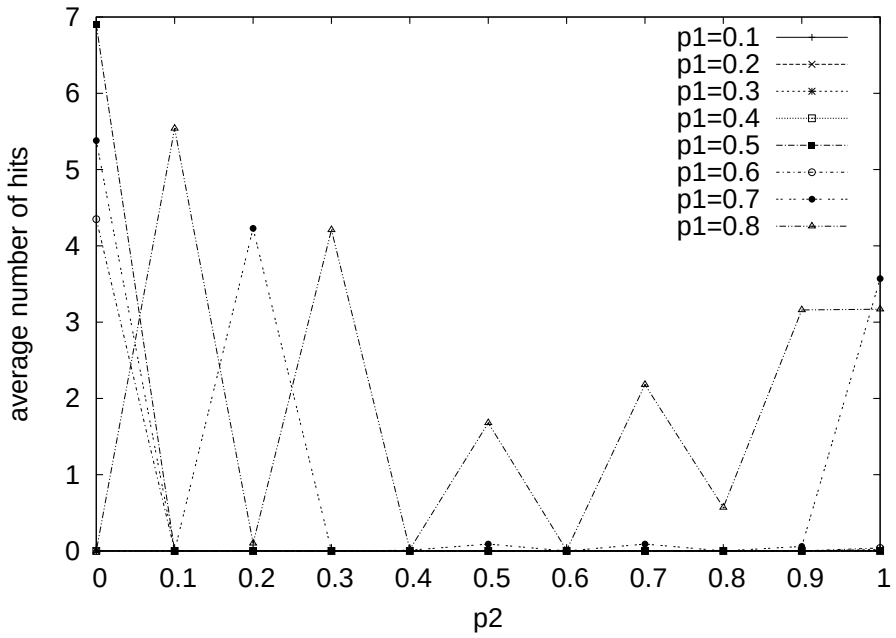



Figure 6.9: Average number of times one of the last 1000 marriages, which is also a local minima, is hit by LTIU (n=50).

made by the SMIs obtained from the current one by swapping the order of two men (or women) which were in tie in the initial SMTI. Then it selects the SMI with the largest stable marriages, say $P^*$. If the stable marriages in $P^*$ are larger than the maximal size found so far, the weight of the tie swapped to obtain $P^*$ is increased by 0.05, otherwise it is decreased by the same amount. The pseudo-code of our local search method is shown in algorithm LST.

In LST, the procedure *GS(Q)* applies the Gale-Shapley algorithm to the problem $Q$ and returns the male optimal marriage. The procedure *swap_tie(Q, ti)* returns an SMI that is the SMI $Q$ given in input where the order of the persons in tie *ti* in $P$ (the SMTI given in input to LST) is swapped.

We can have different versions of the algorithm LST depending on the meaning of the sentence *allowed tie t in P* in line 12. We consider "allowed" the ties which have a weight greater than a fixed threshold or certain percentage of ties with highest weight. In this way we speed up the search by reducing the size of the neighborhood. We call LSTt (where $t$ is the threshold) the algorithm that limits the neighborhood via a threshold and LSTk (where $k$ is the percentage of best ties considered) the other one.

**Algorithm 8**: LST

**input** : a SMTI problem $P$, an integer $max\_steps$, a probability $p$ of random walk
**output**: a marriage

```
1  Q ← breakties(P)
2  steps ← 0
3  max_size ← −1
4  max_neigh_size ← −1
5  repeat
6      M ← GS(Q)
7      if M is a perfect matching then
8          return M
9      if rand() ≤ p then
10         tie_neighbest ← a random tie in P
11     else
12         foreach allowed tie ti in P do
13             R ← swap_tie(Q, ti)
14             M ← GS(R)
15             if size of M > max_neigh_size then
16                 max_neigh_size ← size of M
17                 M_neighbest ← M
18                 tie_neighbest ← ti
19     Q ← swap_tie(Q, tie_neighbest)
20     if max_neigh_size > max_size then
21         max_size ← max_neigh_size
22         M_best ← M_neighbest
23         increase weight of tie_neighbest
24     else
25         decrease weight of tie_neighbest
26     steps ← steps + 1
27 until steps ≥ max_steps ;
28 return M_best
```

## 6.5 Experimental evaluation

We generate SMTI problems as in Section 6.3 except for the probability of ties ($p_2$). For example, if we generate a problem of size $n$=100, with probability

of incompleteness $p_1$=0.1 and probability of ties $p_2$=0.2, the average length of preference lists will be 90, each one with about 9 ties of length 2. In fact, $p_1$=0.1 implies an approximate length of 90 for each list, and $p_2$=0.2 (with ties of length 2) implies about 9 ties.

We generate 100 problems for each combination of $n$, $p_1$ and $p_2$ varying $n$ in {10, 30, 50, 70, 90}, $p_1$ in [0.1, 0.8] and $p_2$ in [0.1, 1.0] and fixing a limit of 20000 steps.

We run our algorithms LSTt and LSTk on this test set and we also compare the results against our LTIU algorithm presented in Section 6.2.

We first measure the average size (normalized w.r.t. the size of the problem) of the stable marriages returned by our algorithms. All three algorithms, find larger marriages when the number of ties increases and when the incompleteness in preference lists decreases. In fact, with more ties and longer preference lists, there is less probability of having a blocking pair and so more chances for singles to get married.

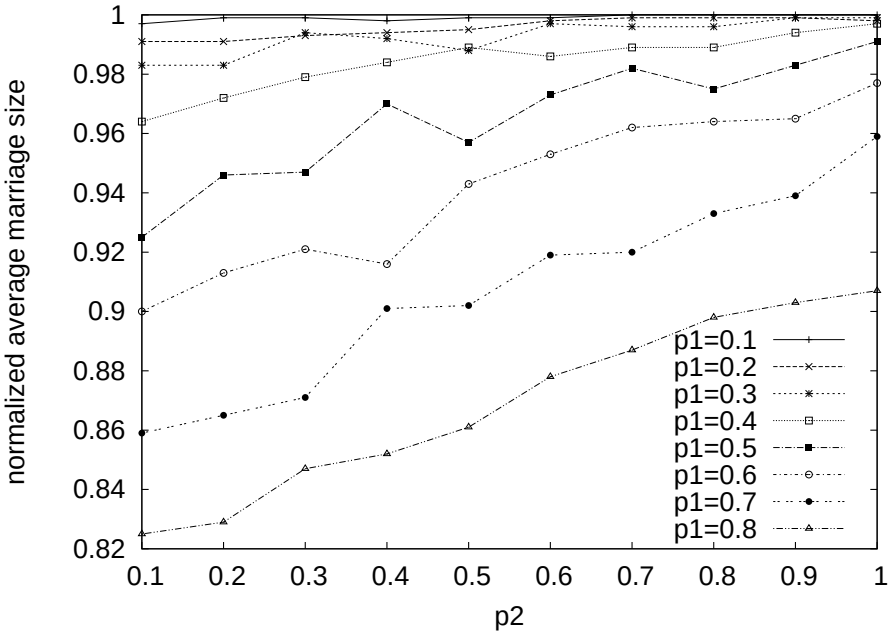



Figure 6.10: Normalized average size of marriages found by LSTk using $k = 50\%$ on SMTIs of size 10.

For instance, Figure 6.10 shows the results for LSTk when $n$=10 and $k$=50%. The results for LSTt and LTIU are very similar. Only for $p_1$=0.7-0.8 and high values of $p_2$ LTIU finds slightly smaller marriages. Figure 6.11 shows a comparison of the three algorithms on problems of size 10 and 30.

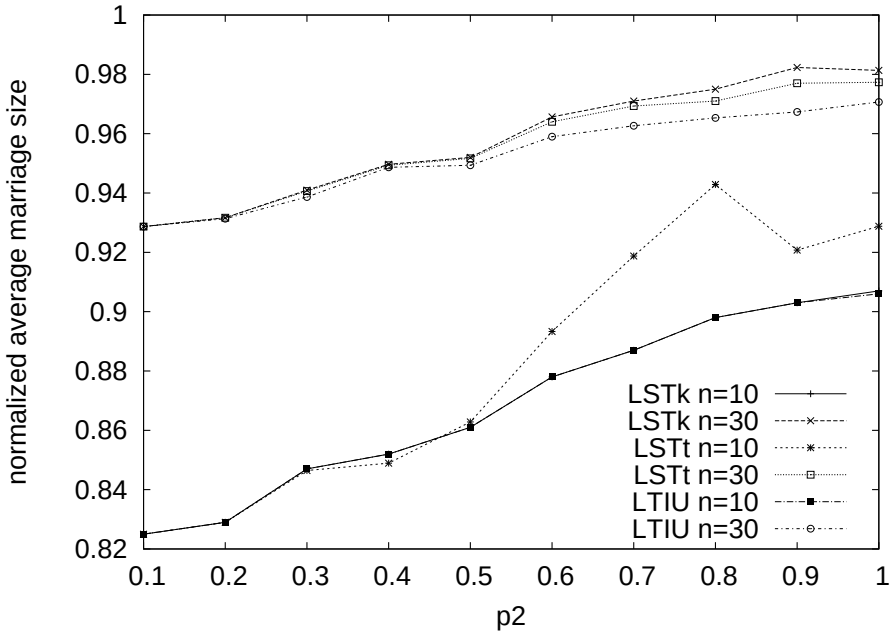



Figure 6.11: Normalized average size for LSTk, LSTt and LTIU on problems of size 10 and 30. Fixing $p_1$=0.8, $k = 50\%$ and $t = 0.5$.

For $n$=10, the size of the marriages vary at most of only 0.02 comparing LSTk versus LSTt (LTIU gives practically the same results as LSTt) when we vary the size of the problems. We can also notice that the size of the marriages tends to increase when the size of the problems increases. For instance, Figure 6.12 shows the results for LSTt and it easy to see that, for the same values of the other parameters, it finds larger marriages as $n$ increases. The same results are obtained by the other algorithms. We conjecture that the reason of this behavior is that, considering SMIs, the probability of having a certain person in at least one preference list, say $P_l$, is very high even with small sizes and a lot of incompleteness. More precisely, the event of having a person $p$ in at least one preference list in an SMI of size $n$, denoted by $P_l(n, p_1)$, is $1 - p_1^n$. Moreover, the probability to be in exactly $k$ lists is:

$$\left[(1-p_1)^k \cdot p_1^{n-k}\right] \binom{n}{k} \tag{6.1}$$

Then, the probability to be in at least $k$ lists is:

$$\sum_{i=k}^{n} \left\{ \left[(1-p_1)^i \cdot p_1^{n-i}\right] \binom{n}{i} \right\} \tag{6.2}$$

Finally, since our generator rejects problems with empty preference lists, in our test set each person is always in at least one preference list. Thus the probability to be in at least $k$ lists becomes:

$$P(n, p_1, k) = \frac{\sum_{i=k}^{n} \left\{ \left[ (1-p_1)^i \cdot p_1^{n-i} \right] \binom{n}{i} \right\}}{1 - p_1^n} \tag{6.3}$$

For example, Figure 6.13 shows how slowly $P(n, p_1, 5)$ decreases varying $p_1$ for different values of $n$. Thus, in general, the probability for a person to be in more than one preference list rises with the size of the problem. Therefore, having a perfect matching or a marriage with very high cardinality is more probable in bigger problems than in smaller ones.

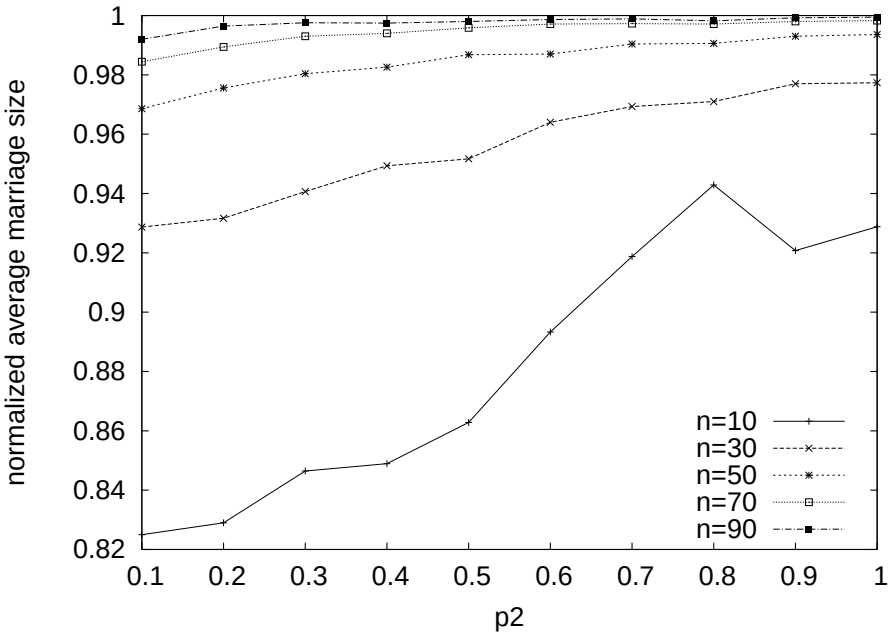



Figure 6.12: Normalized average size for LSTt varying $n$ and fixing $p_1$=0.8 and $t = 0.5$.

We then focus our attention on the average number of steps needed by the algorithms to finish their execution. As can be expected, the number of steps increases as the incompleteness $p_1$ rises. This happens for all algorithms and all problems sizes, and it is more clear as $n$ increases. This can be seen for example in Figure 6.14 that shows the average steps for LSTt on problems of size 10 and in Figure 6.15 that shows the results for $n$=30.

Figure 6.16 shows that the number of steps needed by LSTt for $p_1$=0.8 decreases as $n$ increases. Moreover, it decreases as the amount of ties ($p_2$)

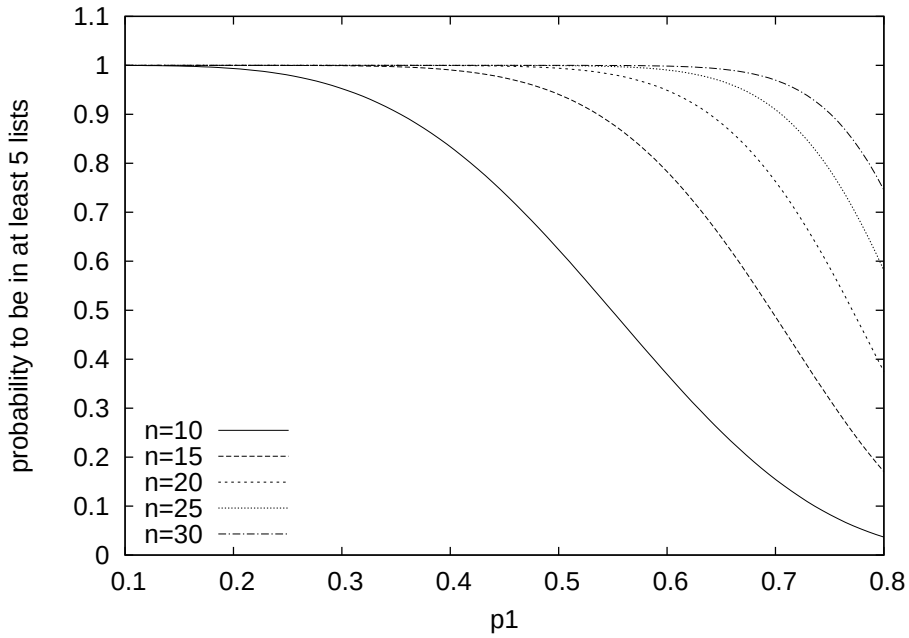



Figure 6.13: Probability for a person to be in at least 5 preference list varying $p_1$.

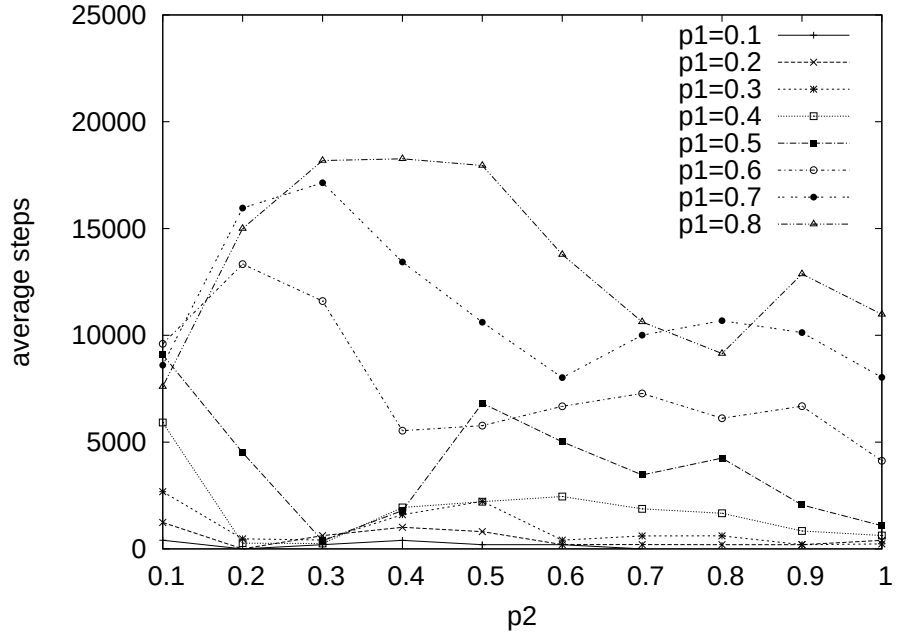



Figure 6.14: Average number of steps for LSTt for $n$=10.

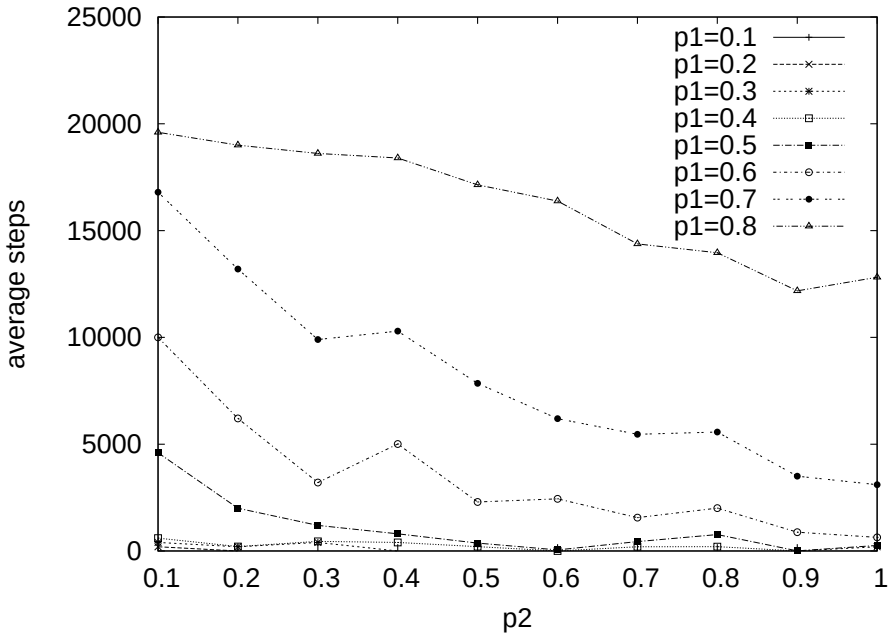



Figure 6.15: Average number of steps for LSTt for $n$=30.

increases. This behavior is the same for the other algorithms and is due to the increased probability of finding a perfect matching on larger problems. For instance, Figure 6.17 shows how steps vary considering problems of size $n$=30 and $n$=90.

We also measured the execution time of our algorithms. The execution time is mainly influenced by the size and nature of the neighborhood that has to be explored at each search step. The neighborhood used by LTIU depends on blocking pairs and so it is larger in problems with few ties. On the other hand, the neighborhoods defined for LSTt and LSTk are bigger as the number of ties arises. For these reasons, the execution time of LTIU tends to slightly decrease as $p_2$ increases no matter the size of the problem for fixed values of $p_1$ (see Figure 6.18). Figures 6.19 and 6.20 show respectively the execution time of LSTk and LSTt. In both cases the execution is longer as $p_2$ becomes larger. The difference is that the size of the neighborhood in LSTt varies dynamically according the weights of the ties and the threshold $t$. This speeds up drastically the algorithm and, as we can see, the execution time of LSTt is about half of the execution time of LSTk.

Summarizing, both LSTk and LSTt are effective in terms of the size of the returned marriages but, when we take into account also the execution time, LSTt has to be preferred.

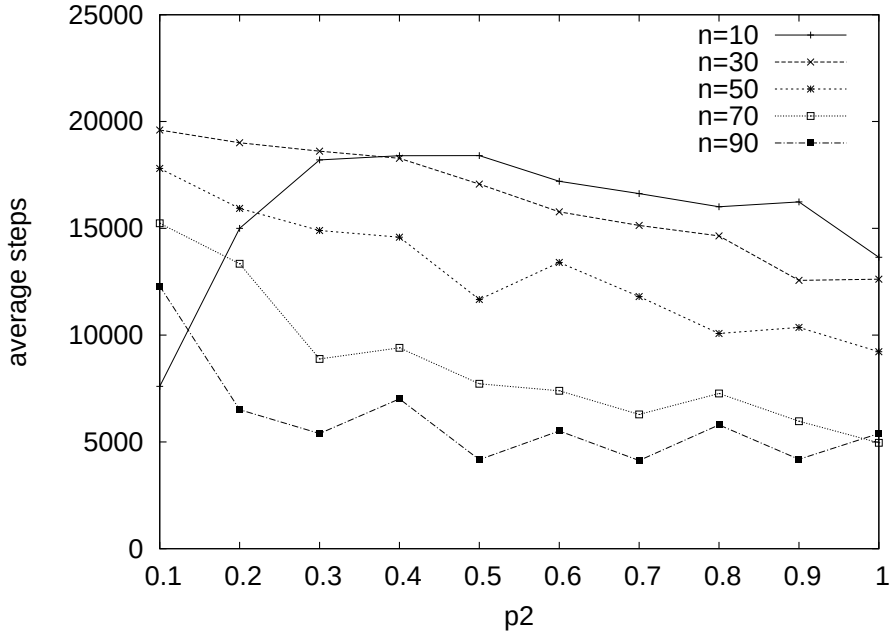



Figure 6.16: Average number of steps for LSTk varying $n$ and fixing $p_1$=0.8 and $k$=50%.

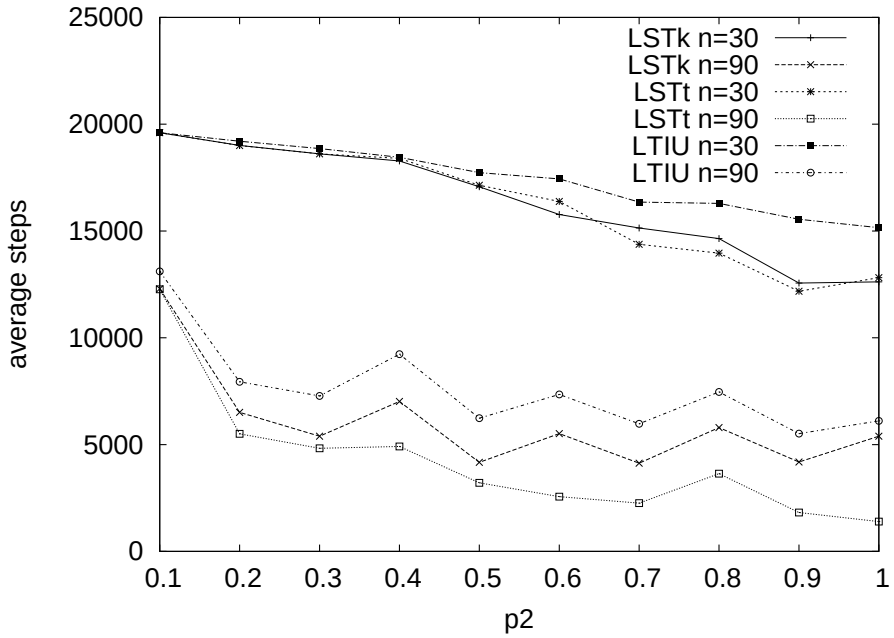



Figure 6.17: Average number of steps for LSTk, LSTt and LTIU on problems of size 10 and 30. Fixing $p_1$=0.8, $k = 50\%$ and $t = 0.5$.

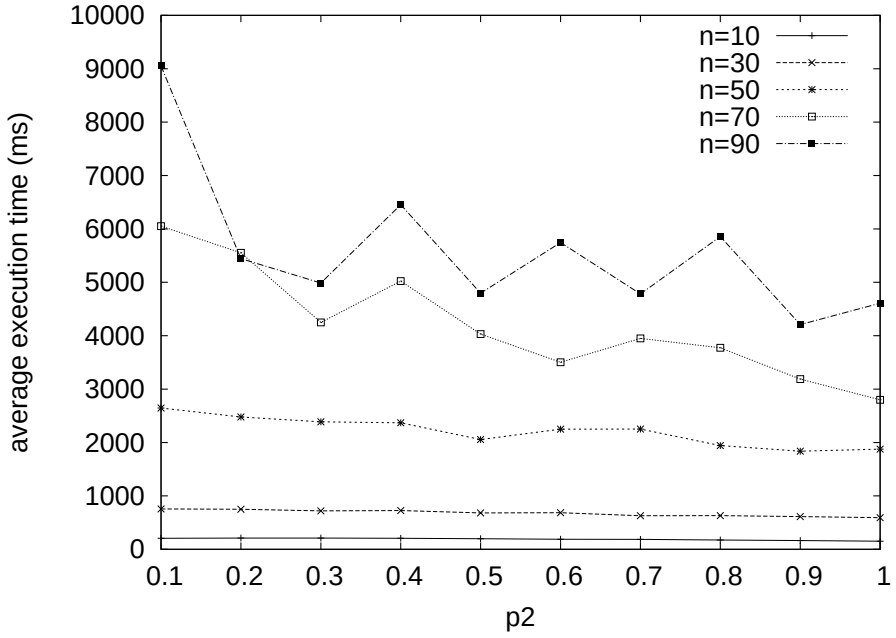



Figure 6.18: Average execution time for LTIU varying $n$ for $p_1$=0.8.

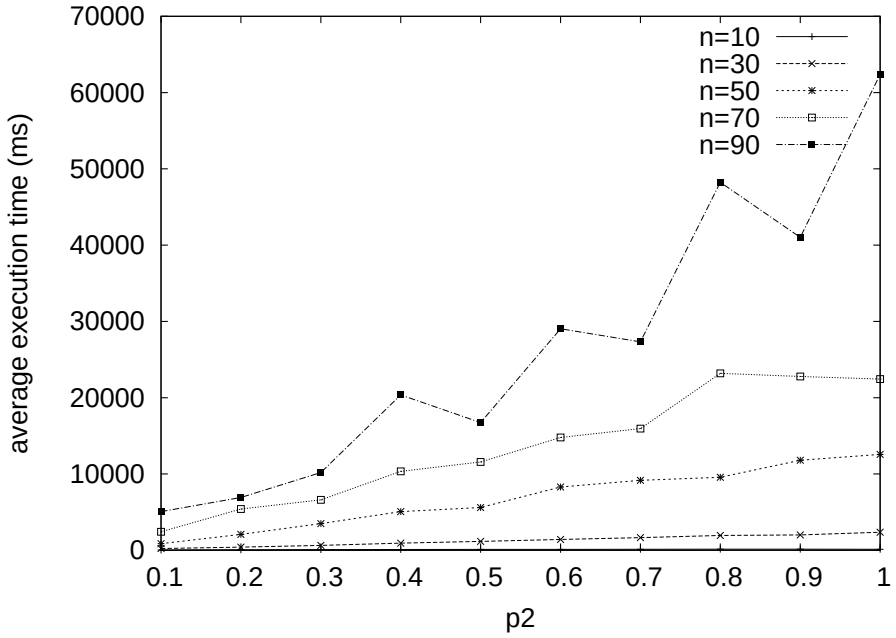



Figure 6.19: Average execution time for LSTk varying $n$ for $p_1$=0.8 and $k$=50%.

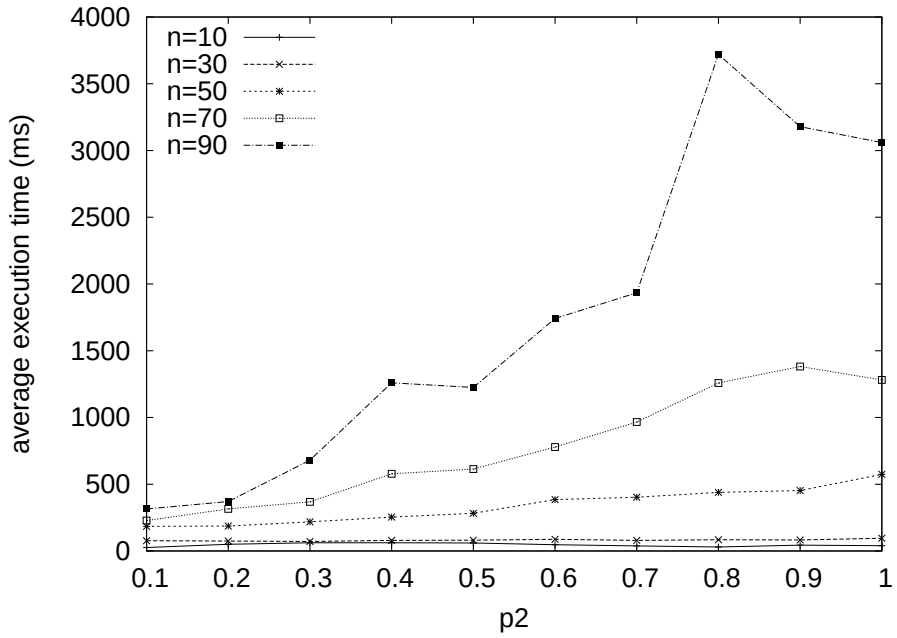



Figure 6.20: Average execution time for LSTt varying $n$ for $p_1$=0.8 and $t$=0.5.

## 6.6 Male optimality

In the classical stable marriage problem, both men and women express a strict order over the members of the other sex. We now consider a potentially more realistic case, where men and women express their preferences via *partial orders*, i.e., given a pair of men (resp., women), the women (resp., the men) can strictly order the elements of the pair, they may say that these elements are in a tie, or that they are incomparable. This is useful in practical applications when a person may not wish (or be able) to choose between alternatives, thus allowing ties in the preference list (or more generally, allowing each preference list to be a partial order) [39]. For example, in the context of centralized matching scheme, some participating hospitals with many applicants have found the task of producing a strictly ordered preference list difficult, and have expressed a desire to use ties [44]. Ties also naturally occur when assigning students to schools, since many students are indistinguishable from the point of view of a given school. Another situation where partial orders are useful is when preferences are elicited with a compact preference representation formalism like soft constraints [7] or CP-nets [9] that give partial orders. Another context where partial orders naturally occur is when preferences are obtained via multi-criteria reasoning.

We study male optimality and uniqueness of solution in this more general

context. *Male optimality* can be a useful property since it allows us to give priority to one gender over the other. For example, in matching residents to hospitals in the US, priority is given to the residents. We present an algorithm, based on an extended version of the Gale-Shapely algorithm, to find a male optimal solution in stable marriage problems with partially ordered preferences (SMPs). This algorithm is sound but not complete: it may fail to find a male optimal solution even when one exists. We conjecture, however, that the incompleteness is rare. We also give a sufficient condition on the preference profile that guarantees to find a male optimal solution, and we show how to find it. *Uniqueness* is another interesting concept. For instance, it guarantees that the solution is optimal since it is as good as possible for all the participating agents. Uniqueness is also a barrier against manipulation. This is important as *all* stable marriage procedures can be manipulated. In [18] sufficient conditions on the preferences are given, that guarantee uniqueness of stable marriages when only strictly ordered preferences are allowed. Such conditions identify classes of preferences that are broad and of particular interest in many real-life scenarios [14]. In particular, a class of preference orderings that satisfy one of these conditions requires that all the agents of the same sex have identical preferences over the mates of the opposite sex, i.e., there is a common ordering over the mates. Another class of preference orderings that satisfy one of these conditions of uniqueness requires that each agent has a different most preferred mate, i.e., there is a subjective ranking over the mates. We show that it is possible to generalize these sufficient conditions for uniqueness to SMs with partially ordered preferences, by considering in some cases uniqueness up to indifference and incomparability.

## Stable marriage problems with partial orders

We assume now that men and women express their preferences via partial orders. The notions given in Section 2.2 can be generalized as follows.

**Definition 49** (partially ordered profile)**.** *Given $n$ men and $n$ women, a profile is a sequence of $2n$ partial orders (i.e., reflexive, antisymmetric and transitive binary relations), $n$ over the men and $n$ over the women.*

**Definition 50** (SMP)**.** *A stable marriage problem with partial orders (SMP) is just a SM where men's preferences and women's preference are partially ordered.*

**Definition 51** (linearization of an SMP)**.** *A linearization of an SMP is an SM that is obtained by giving a strict ordering to all the pairs that are not strictly ordered such that the resulting ordering is transitive.*

**Definition 52** (weakly stable marriage in SMP)**.** *A marriage in an SMP is weakly stable if there is no pair $(x, y)$ such that each one strictly prefers the other to his/her current partner.*

**Definition 53** (feasible partner in SMP)**.** *Given an SMP $P$, a feasible partner for a man $m$ (resp., woman $w$) is a woman $w$ (resp., man $m$) such that there is a weakly stable marriage for $P$ where $m$ and $w$ are married.*

A weakly stable marriage is male optimal if there is no man that can get a strictly better partner in some other weakly stable marriage.

**Definition 54** (male optimal weakly stable marriage)**.** *Given an SMP $P$, a weakly stable marriage of $P$ is male optimal iff there is no man that prefers to be married with another feasible partner of $P$.*

In SMs there is always exactly one male optimal stable marriage. In SMPs, however, we can have zero or more male optimal weakly stable marriages. Moreover, given an SMP $P$, all the stable marriages of the linearizations of $P$ are weakly stable marriages. However, not all these marriages are male optimal.

**Example 5.** *In a setting with 2 men and 2 women, consider the profile $P$: $\{m_1 : w_1 \bowtie w_2$ ($\bowtie$ means incomparable); $m_2 : w_2 > w_1$ ($a > b$ means that $a$ is preferred to $b$) ;$\}$ $\{w_1 : m_1 \bowtie m_2;\ w_2 : m_1 \bowtie m_2;\}$. Then consider the following linearization of $P$, say $Q$: $\{m_1 : w_2 > w_1;\ m_2 : w_2 > w_1;\}$ $\{w_1 : m_2 > m_1;\ w_2 : m_1 > m_2;\}$. If we apply the extended GS algorithm to $Q$, we obtain the weakly stable marriage $\mu_1$ where $m_1$ marries $w_2$ and $m_2$ marries $w_1$. However, $w_1$ is not the most preferred woman for $m_2$ amongst all weakly stable marriages. In fact, if we consider the linearization $Q'$, obtained from $Q$, by changing $m_1$'s preferences as follows: $m_1 : w_1 > w_2$, and if we apply the extended GS algorithm, we obtain the weakly stable marriage $\mu_2$, where $m_1$ is married with $w_1$ and $m_2$ is married with $w_2$, i.e., $m_2$ is married with a woman that $m_2$ prefers more than $w_1$. Notice that $\mu_2$ is male optimal, while $\mu_1$ is not. Also, $\mu_1$ and $\mu_2$ are the only weakly stable marriages for this example.* □

## Finding male optimal weakly stable marriages

We now present an algorithm, called *MaleWeaklyStable*, that takes as input an SMP $P$ and, either returns a male optimal weakly stable marriage for $P$, or the string 'I don't know'. This algorithm is sound but not complete: if the algorithm returns a marriage, then it is weakly stable and male optimal; however, it may fail to return a male optimal marriage even if there is one.

**Algorithm 9**: *MaleWeaklyStable*

```
1  Input: p: a profile;
2  Output: μ: a weakly stable marriage or the string 'I don't
   know';
3  μ ← ∅;
4  L ← list of the men of p;
5  L ← ComputeOrderedList(L);
6  while Top((first(L)) contains exactly one unmarried woman)
   or (first(L) has a single top choice already married) do
7      m ← first(L);
8      if Top(m) contains exactly one unmarried woman then
9          w ← UnmarriedTop(m);
10         Add the pair (m, w) to μ;
11         foreach strict successor m* of m on w's preferences do
12             delete m* from w's preferences and w from m's
               preferences ;
13         L ← L \ {m};
14         L ← ComputeOrderedList(L);
15     if m has a single top choice already married then
16         w ← Top(m);
17         m' ← μ(w);
18         Remove the pair (m', w) from μ;
19         Add the pair (m, w) to μ;
20         foreach strict successor m* of m on w's preferences do
21             delete m* from w's preferences and w from m's
               preferences;
22         L ← L ∪ {m'} \ {m};
23         L ← ComputeOrderedList(L);
24 if (L = ∅) or (AllDiffUnmarried(L)=true) then
25     Add to μ AllDiffUnmarriedMatching(L);
26     return μ
27 else
28     return 'I don't know'
```

We assume that the women express strict total orders over the men. If they don't, we simply pick any linearization.

The algorithm exploits the extended GS algorithm [36], and at every step orders the free men by increasing number of their current top choices (i.e., the alternatives that are undominated). List $L$ contains the current ordered sequence of free men.

More precisely, our algorithm works as follows. It takes in input an SMP $P$, and it computes the list $L$ of free men. At the beginning all the men are unmarried, and thus $L$ contains them all. Then, we continue to check the following cases on the man $m$ which is the first element of $L$, until they do not occur any longer:

- If the set of top choices of $m$ contains exactly one unmarried woman, say $w$, $m$ proposes to $w$ and, since we are using the extended GS algorithm, the proposal is accepted. Then, all men that are strictly worse than $m$ in $w$'s preferences are removed from $w$'s preference list, and $w$ is removed from the preference lists of these men. Then, $m$ is removed from $L$ and $L$ is ordered again, since the top choices of some men may now be smaller.

- If $m$ has a single top choice, say $w$, that is already married, $m$ proposes to $w$, $w$ accepts the proposal, and she breaks the engagement with her current partner, say $m'$. Then, $m$ is removed from $L$, $m'$ becomes free and is put back in $L$, and $L$ is ordered again.

When we exit from this cycle, we check if $L$ is empty or not:

- if $L$ is empty, the algorithm returns the current marriage. Notice that the current marriage, say $(m_i, w_i)$, for $i = 1, \ldots, n$, is weakly stable, since it is the solution of a linearization of $P$ where, for every $m_i$ with ties or incomparability in current set of top choices, we have set $w_i$ strictly better than all the other women in the top choice. Also, the returned marriage is male optimal since we have applied the extended GS algorithm.

- If $L$ is not empty, it means that the next free man in $L$ has several current top choices and more than one is unmarried.

  - If there is a way to assign to the men currently in $L$ different unmarried women from their current top choices then these men make these proposals, that are certainly accepted by the women, since every woman receives a proposal from a different man. Therefore, we add to the current marriage these new pairs and we return

the resulting marriage. Such a marriage is weakly stable and male optimal by construction.

– If it is not possible to make the above assignment, stops returning the string 'I don't know'.

**Example 6.** *Consider the profile* $\{m_1 : w_1 \bowtie w_2 > w_3;\ m_2 : w_1 \bowtie w_2 > w_3;\ m_3 : w_1 \bowtie w_2 > w_3;\}$ $\{w_1 : m_1 > m_2 > m_3;\ w_2 : m_1 > m_2 > m_3;\ w_3 : m_1 > m_2 > m_3;\}$. *The algorithm first computes the ordered list* $L = [m_1, m_2, m_3]$. *The elements of* $L$ *are men with more than one top choice and all these top choices are unmarried, but there is no way to assign them with different women from their top choices, since they are three men and the union of their top choices contains only two women. However, in every linearization,* $m_3$ *will not be matched with* $w_1$ *or* $w_2$, *due to* $w_1$ *and* $w_2$*'s preferences. In fact,* $m_1$ *and* $m_2$ *will choose between* $\{w_1, w_2\}$, *while* $m_3$ *will always propose to his next best choice, i.e.,* $w_3$. *Hence, the considered profile is one of the profiles where only two of the three men with multiple top choices are feasible with* $w_1$ *and* $w_2$, *i.e.* $m_1$ *and* $m_2$, *and there is a way to assign to these men different unmarried women in their top choices. In such a case there are two male optimal weakly stable solutions, i.e.,* $\{(m_1, w_1)(m_2, w_2)(m_3, w_3)\}$ *and* $\{(m_1, w_2)(m_2, w_1)(m_3, w_3)\}$. *Our algorithm returns the first one.* □

**Example 7.** *Consider the profile obtained from the profile shown in Example 6 by changing the preferences of* $w_1$ *as follows:* $m_1 > m_3 > m_2$. *We now show that there is no male optimal solution. It is easy to see that in any weakly stable marriage* $m_1$ *is married with* $w_1$ *or* $w_2$. *In the weakly stable marriage where* $m_1$ *is married with* $w_1$, $m_2$ *must be married with* $w_2$ *and* $m_3$ *must be married with* $w_3$, *while in the weakly stable marriage where* $m_1$ *is married with* $w_2$, $m_2$ *must be married with* $w_3$ *and* $m_3$ *must be married with* $w_1$. *Therefore, in any weakly stable marriage, exactly one of these conditions holds: either* $m_2$ *prefers to be married with* $w_2$, *or* $m_3$ *prefers to be married with* $w_2$. *Therefore, there is no male optimal solution. Our algorithm works as follows. Since AllDiffUnmarried(L)=false and since we cannot remove any unfeasible woman from the top choices of* $m_1$, $m_2$, *and* $m_3$, *the algorithm returns the string 'I don't know'.* □

The *MaleWeaklyStable* algorithm has a time complexity which is $O(n^{\frac{5}{2}})$. In fact, the first part has the same complexity of the extended GS algorithm, which is $O(n^2)$. The second part requires performing an all-different check between the current set of free men and the union of their top choices. Since there are at most $n$ free men and $n$ top choices for each man, we can build a bipartite graph where nodes are men and women, and each arc connects a man with one of his unmarried top choices. Performing the all-different check

means finding a subset of the arcs which forms a matching in this graph and involves all men. This can be done in $O(m\sqrt{n})$ where $m$ is the number of edges, which is $O(n^2)$.

The *MaleWeaklyStable Algorithm* is sound, but not complete, i.e., if it returns a marriage, then such a marriage is male optimal and weakly stable, but if it returns the string 'I don't know', we don't know if there is a weakly stable marriage that is male optimal. A case where our algorithm returns the string 'I don't know' is when $L$ is *not empty* and there is a free man with more than one top choice and *all his top choices are already married*. We conjecture that in this case there is a male optimal weakly stable marriage a few times, since it seems there are some very specific circumstances for our algorithm to mot return a male optimal weakly stable marriage (i.e., it has to pass through all the conditions we test) when one exists.

As we noticed above, there are SMPs with no male optimal weakly stable marriages. We now want to identify a class of SMPs where it is always possible to find a linearization which has a male optimal stable marriage.

**Definition 55** (male-alldifference property)**.** *An SMP $P$ satisfies the male-alldifference property iff men's preferences satisfy the following conditions:*

- *all the men with a single top choice have top choices that are different;*
- *it is possible to assign to all men with multiple top choices an alternative in their top choices that is different from the one of all the other men of $P$.*

**Theorem 23.** *If an SMP is male-alldifferent, then there is a weakly stable marriage that is male optimal and we can find it in polynomial time.*

*Proof.* If an SMP satisfies the male-alldifference property, then, by Definition 55, we can easily build the marriage $\mu$ where all the men with a single top choice are married with their first top choice and all the men with more than one top choice are married with that alternative in their top choices that satisfies the second hypothesis of Definition 55. This marriage is both weakly stable and male optimal. It is is weakly stable, since it can be obtained by applying men-proposing GS on one of the linearizations where, for all the men $m_i$ with more than one top choice, we put $\mu(m_i)$ strictly better than all the other alternatives. Moreover, by construction, $\mu$ is male optimal, i.e., there is no other weakly stable marriage where a man can obtain a strictly better partner. In fact, all the men with a single top choice are already married in $\mu$ with their best woman, and thus they cannot obtain a better result, and, by construction, all the men with more than one alternative in their top are married with one of these alternatives, that is better than, in

a ties with or incomparable to all the other alternatives, and thus also these men cannot obtain a strictly better woman in any other marriage. □

The *MaleWeaklyStable Algorithm* exploits this same sufficient condition, plus some other sufficient condition. Notice that if an SMP satisfies the male-alldifference property, then, not only is there at least one weakly stable marriage that is male optimal, but there is an unique stable marriage up to ties and incomparability.

## A complete algorithm for male-optimality in SMPs

We have shown an incomplete algorithm to find a male-optimal weakly stable marriage in stable marriage problems where men may express partially ordered preferences.

To measure the incompleteness of this algorithm, we need to know when a male optimal weakly stable marriage exists. To do this, we define now a complete algorithm, that we call *GSTiesComplete*, to find, when it exists, a male-optimal weakly-stable marriage for these problems. It takes as input an SMP $P$ with $n$ men and $n$ women, where the women's preferences are totally ordered, and it returns a set $R$ of weakly-stable marriages that are male-optimal, i.e., such that no man prefers to be married with another feasible woman.

First, it sets $R$ to the empty set. Then, it instantiates an array $C$ of size $n$ where every element is zero. During the execution, for each man $m_i$, $C$ associates the current woman $w_j$ ($C[m_i] = w_j$) that has been matched with $m_i$, if any, otherwise it associates zero. Then, it calls algorithm *GSties* with $P$, $L$, $C$, and $R$, as input, and it returns the set $R$ returned by *GSties*. The returned set $R$ contains the weakly-stable marriages of $P$ that are male-optimal. Notice that at the end $R$ may also be empty. This happens when no male-optimal weakly-stable marriage exists.

**Algorithm 10**: *GStiesComplete*

```
1 Input: P: an SMP where the women's preferences are totally
  ordered, Output: R: a set of marriages;
2 L ← the list of the men of P;
3 R ← ∅;
4 C ← an array [0, ..., 0] of size n;
5 GSties(P, L, C, R);
6 return R;
```

Algorithm *GSties*$(P, L, C, R)$ is a recursive algorithm that works as follows. First, it instantiates $SW$, that is, a set of women, to the empty set.

Then, while the list $L$, that is the list of the free men, is not empty, it performs the following sequence of steps. It considers $m$, i.e., the first man of $L$, and it instantiates $SW$ with the next set of women (that are in a tie or incomparable), that are strictly worse than $C[m]$ in $m$'s preferences in $P$. $C[m]$ is zero as $m$ is a free man and so $SW$ contains the top choices of $m$. Then, there are two possible cases: $|SW| = 1$ or $|SW| > 1$.

- If $|SW| = 1$, there is only one woman in $SW$, i.e., the next woman worse than $C[m]$ in $m$'s preferences is unique. Then, we consider such a woman, say $w$, and we marry $m$ with $w$, i.e., we put $w$ in $C[m]$. Then, we remove $m$ from the list of the free men, we remove from the $w$'s preferences in $P$ every man that is strictly worse than $m$ for $w$, and we remove $w$ from the preferences of these men. If there is a man, say $m_k$ already married with $w$ (i.e., $C[m_k] = w$), then we put $m_k$ again in the list of the free men $L$ and so we put 0 in $C[m_k]$.

- If $|SW| > 1$, we build a new SMP $P'$, obtained from $P$ by putting $w$ strictly better than all the other women in $SW$ in $m$'s preferences and by removing every man that is worse than $m$ in $w$'s preferences as well as removing $w$ from the preferences of these men. Moreover, we compute a new array $C'$, that is obtained from $C$ by putting $w$ in $C'(m)$.

  If there is a man, say $m_k$, which is already married with $w$, we put 0 in $C[m_k]$ and we call recursively algorithm *GSties* with inputs $P'$, $L - m + m_k$, $C'$, and $R$, otherwise we call recursively algorithm *GSties* with inputs $P'$, $L - m$, $C'$, and $R$.

When we exit from the while loop, the list of the free men $L$ is empty, and thus all men have been married. We call $M_{new}$ such a marriage. We put $M_{new}$ in $R$ and we denote with $R^*$ the new set. Then, for every marriage $M$ in $R^*$, we check if there is a man, say $m$, such that he prefers his partner in $M$ (resp., $M_{new}$) more than his partner in $M_{new}$ (resp., $M$). If this happens, we remove $M_{new}$ (resp., $M$) from $R$. Finally, we return $R$.

Algorithm *GStiesComplete* is sound and complete. In the worst case all women are in tie and each man proposes to every woman so the while loop is executed $n^2$ times. In every loop we have $n!$ ways to break ties in the current top choice. Thus, the main loop complexity is $O(n^2 * (n!)^n)$. Then, the last steps of the algorithm compare each marriage with the other found so far and so, in the worst case, it is done $(n!)^2$ times. Hence, the overall complexity of the algorithm is $O(n^2 * (n!)^n + (n!)^2)$.

**Proposition 13.** *Given an SMP $P$ where the women's preferences are totally ordered, Algorithm* GStiesComplete *applied to $P$ always return a set of*

**Algorithm 11**: *GSties*

**Input**: $P$: an SMP where the women's preferences are totally ordered, $L$: a list of $n$ men, $C$: an array of $n$ women, $R$: a set of weakly-stable marriages of $P$;
**Output**:$R$: a set of weakly-stable marriages of $P$;
$SW \leftarrow \emptyset$ ;
**while** $L \neq \emptyset$ **do**
  $m \leftarrow \text{first}(L)$;
  $SW \leftarrow \text{next}(m, P, C[m])$;
  **if** $|SW| = 1$ **then**
    $w \leftarrow \text{el}(SW)$;
    $C[m] \leftarrow w$;
    $L \leftarrow L - m$;
    **foreach** *man* $m'$ *s.t.* $m >_w m'$ **do**
      remove $m'$ from $P(w)$ and $w$ from $P(m')$;
    **if** $\exists$ $m_k$ *such that* $C[m_k] = w$ **then**
      $C[m_k] = 0$;
      add $m_k$ to $L$;
  **else**
    **foreach** *woman* $w \in SW$ **do**
      **foreach** *man* $a \neq m$ **do**
        $P'(a) \leftarrow P(a)$;
        $C'(a) \leftarrow C(a)$;
      **foreach** *woman* $b$ **do**
        $P'(b) \leftarrow P(b)$;
      $P'(m) \leftarrow P(m) \cup \{w > \text{rest}(SW, w)\}$;
      $C'(m) \leftarrow w$;
      **foreach** *man* $m'$ *s.t.* $m >_w m'$ **do**
        remove $m'$ from $P'(w)$ and $w$ from $P'(m')$;
      **if** $\exists$ $m_k$ *such that* $C'[m_k] = w$ **then**
        $C'[m_k] = 0$;
        $\text{GSties}(P', L - m + m_k, C', R)$;
      **else**
        $\text{GSties}(P', L - m, C', R)$;
    break;
$M_{new} \leftarrow \text{Marriage}(P, C)$;
**if** $M_{new}$ *is not a complete marriage* **then**
  **return**
$R \leftarrow R \cup M_{new}$;
$R^* \leftarrow R \cup M_{new}$;
**foreach** *marriage* $M$ *in* $R^* - M_{new}$ **do**
  **if** *there is a man* $m$ *s.t.* $M(m) < M_{new}(m)$ **then**
    $R \leftarrow R - M$;
  **if** *there is a man* $m$ *s.t.* $M_{new}(m) < M(m)$ **then**
    $R \leftarrow R - M_{new}$;
**return** $R$;

*marriages that are weakly stable and male-optimal for $P$. When such a set is empty, the SMP $P$ does not have any weakly stable marriage that is male optimal.*

Notice that *GStiesComplete* algorithm is a more sophisticated version of a naive algorithm that takes in input an SMP $P$ where the women are strictly ordered, and it applies the GS algorithm to every linearization of $P$, thus obtaining various weakly stable marriages, and among these marriages it returns only the weakly stable marriages that are male optimal. Therefore, it is easy to adapt Algorithm *GStiesComplete* to handle also generic SMPs, where also the women can express partially ordered preferences.

## Experimental evaluation

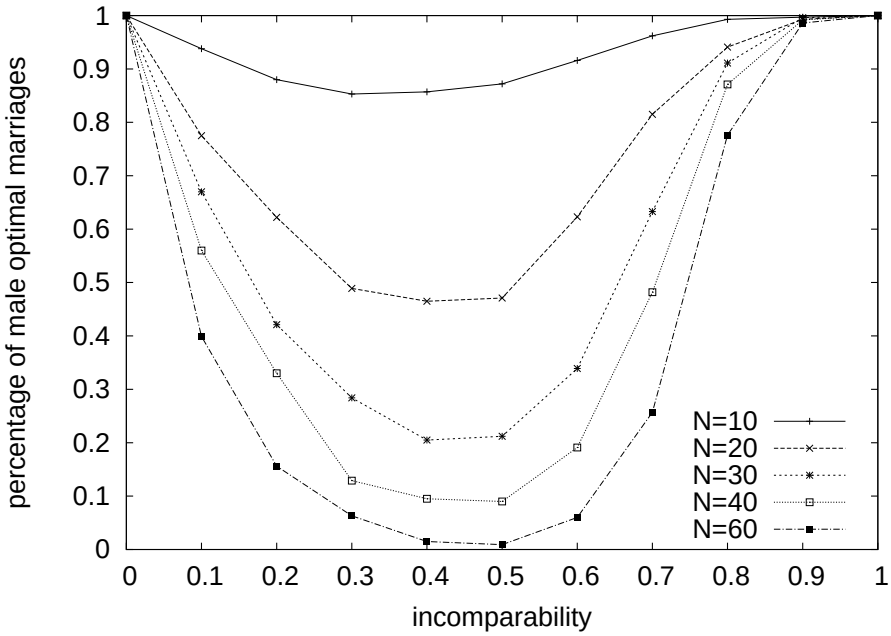



Figure 6.21: Percentage of male optimal stable marriages returned as we vary the amount of incomparability.

In Figure 6.21, we show the behaviour of the *MaleWeaklyStable* algorithm on some SMPs generated randomly: each woman totally orders the men uniformly at random; each man totally orders the women uniformly at random; with probability $p$, each neighboring pair in this total order is incomparable. Hence, $p = 0$ means no incomparability, whilst $p = 1$ means that all woman

are ranked incomparable by all men. In Figure 6.21, N is the total number of agents. We tested 1000 stable marriage problems for N=10 to 60 in steps of 10, and $p$=0 to 1 in steps of 0.1. It can be seen that, for small instances, the *MaleWeaklyStable* algorithm returns a male optimal weakly stable marriage very often. For larger instances, it returns a male optimal weakly stable marriage in a smaller percentage of cases. However, the percentage grows for large $p$, as the chance that the top choices are all different increases.

We intend now to investigate the incompleteness of the *MaleWeaklyStable* algorithm by comparing its behaviour to the one of the complete algorithm *GStiesComplete* on randomly generated instances of SMPs where the women's preferences are totally ordered. In Table 6.2 we compare the MaleWeaklyStable algorithm with the complete algorithm GStiesComplete showing how many times MaleWeaklyStable does not find a male optimal weakly stable marriage while GStiesComplete finds one. The comparison is performed on randomly generated SMPs computing the average over 50 instances, and varying $N$ from 10 to 60 and $p$ in $\{0.1, 0.2, 0.5, 0.7, 0.9\}$. We set a timeout for GStiesComplete and '-' means that the algorithm did not give a result. It is easy to notice that the incompleteness of the MaleWeaklyStable algorithm is not very large and it is always less than 25%.

| N | incomparability | | | | |
|---|---|---|---|---|---|
| | 0.1 | 0.2 | 0.5 | 0.7 | 0.9 |
| 10 | 2% | 10% | 10% | 2% | 0% |
| 20 | 6% | 20% | 8% | 4% | 0% |
| 30 | 16% | 18% | 14% | - | 0% |
| 40 | 20% | 24% | - | - | 0% |
| 50 | 12% | 24% | - | - | 0% |
| 60 | 24% | 16% | - | - | 0% |

Table 6.2: Percentage of times MaleWeaklyStable does not found a male optimal marriage when GStiesComplete does.

## 6.7 Uniqueness of weakly stable marriages

For strict total orders, [18] gives sufficient conditions on preference for the uniqueness of the stable marriage. We now extend these results to partial orders. Notice that, if there is an unique stable marriage, then it is clearly male optimal. A class of preference profiles in [18] giving an unique stable marriage, when the preferences are strict total orders, is defined as follows. The set of the men and the set of the women are ordered sets, the preferences

require that no man or woman prefers the mate of the opposite sex with the same rank order below his/her own order. Given such a preference ordering, by a recursive argument starting at the highest ranked mates, any other stable marriage would be blocked by the identity marriage, i.e., the marriage in which we match mates of the same rank.

**Theorem 24.** [18] *Consider two ordered sets $M = (m_i)$ and $W = (w_i)$. If the profile satisfies the following conditions:*

$$\forall w_i \in W : \ m_i >_{w_i} m_j, \ \forall j > i \tag{6.4}$$

$$\forall m_i \in M : \ w_i >_{m_i} w_j, \ \forall j > i \tag{6.5}$$

*then there is a unique stable marriage $\mu^*(w_i) = m_i$, $\forall i \in \{1, 2, \ldots, \frac{N}{2}\}$.*

Notice that the condition above is also necessary when the economies are small, i.e., $N = 4$ and $N = 6$.

There are two particular classes of preference profiles that generate a unique stable marriage, and that are commonly assumed in economic applications [18]. The first assumes that all the women have identical preferences over the men, and that all the men have identical preferences over the women. In such a case there is a common (objective) ranking over the other sex.

**Definition 56** (vertical heterogeneity)**.** [18] *Consider two ordered sets $M = (m_i)$ and $W = (w_i)$. A profile satisfies the vertical heterogeneity property iff it satisfies the following conditions:*

- $\forall w_i \in W : m_k >_{w_i} m_j, \ \forall k < j$
- $\forall m_i \in M : w_k >_{m_i} w_j, \ \forall k < j$

**Example 8.** *An example of a profile that satisfies vertical heterogeneity for $N = 6$ is the following.* $\{m_1 : w_1 > w_2 > w_3; \ m_2 : w_1 > w_2 > w_3; \ m_3 : w_1 > w_2 > w_3;\}$ $\{w_1 : m_2 > m_3 > m_1; \ w_2 : m_2 > m_3 > m_1; \ w_3 : m_2 > m_3 > m_1.\}$ □

**Corollary 24.1.** [18] *Consider two ordered sets $M = (m_i)$ and $W = (w_i)$ and a profile $P$. If $P$ satisfies the vertical heterogeneity property, then there is a unique stable marriage $\mu^*(w_i) = m_i$.*

When agents have different preferences over the other sex, but each agent has a different most preferred mate and in addition is the most preferred by the mate, then the preference profile satisfies horizontal heterogeneity. In this situation there is a subjective ranking over the other sex.

**Definition 57** (horizontal heterogeneity)**.** [18] *Consider two ordered sets* $M = (m_i)$ *and* $W = (w_i)$*. A profile satisfies the horizontal heterogeneity property iff it satisfies the following conditions:*

- $\forall w_i \in W : m_i >_{w_i} m_j, \ \forall j$
- $\forall m_i \in M : w_i >_{m_i} w_j, \ \forall j$

**Example 9.** *The following profile over 3 men and 3 women satisfies horizontal heterogeneity.* $\{m_1 : w_1 > \ldots;\ m_2 : w_2 > \ldots;\ m_3 : w_3 > \ldots\}$ $\{w_1 : m_1 > \ldots;\ w_2 : m_2 > \ldots;\ w_3 : m_3 > \ldots.\}$ □

**Corollary 24.2.** [18] *Consider two ordered sets* $M = (m_i)$ *and* $W = (w_i)$ *and a profile* $P$*. If* $P$ *satisfies the horizontal heterogeneity property, then there is a unique stable marriage* $\mu^*(w_i) = m_i$*.*

We now check if the results given above for strictly ordered preferences can be generalized to the case of partially ordered preferences. Theorem 24 holds also when the men's preferences and/or women's preferences are partially ordered.

**Theorem 25.** *In SMPs, if there is an ordering of men and women such that the preference profile satisfies the conditions described in Theorem 24, then there is a unique weakly stable marriage* $\mu(w_i) = m_i$*,* $\forall i \in \{1, 2, \ldots, n\}$*.*

*Proof.* The proof is similar to the one used to show Theorem 26. More precisely, to show that the marriage $\mu$ is unique, we show that in any linearization the male optimal and the female optimal marriage coincide with $\mu$. For any linearization of $p$, we can compute the male optimal marriage by using the men-proposing extending GS. First, $m_1$ makes the proposal to $w_1$ that accepts, since $m_1$ is her best man in her preferences all the other men are removed from $w$'s preference ranking and $w$ is removed from the list of these men. Therefore, $w$ will not receive any other proposal and thus she remains with $m_1$ until the end of the algorithm. Hence, $\mu(m_1) = w_1$. Similarly, we can show that $\mu(m_2) = w_2$ and so on. Hence, the male optimal stable marriage in every linearization of $p$ is $\mu(m_i) = w_i$, $\forall i \in \{1, 2, \ldots, \frac{N}{2}\}$. To conclude that $\mu$ is unique, we can show with a reasoning similar to the one performed above, but using the women-proposing extended GS, instead of the men-proposing extended GS, that for every linearization of $p$ also the female optimal stable marriage is $\mu$. □

Notice that the condition above is also necessary when the economies are small. For example, this holds when $N = 6$ (that is, three men and three women).

We now check if the *vertical heterogeneity* result (Corollary 24.1) holds also when the preferences are partially ordered. We recall that vertical heterogeneity assumes that all the agents of the same sex have the same strict preference ordering over the mates of the opposite sex. It is possible to see that, even if there is only one incomparable element in the ordering given by the men (or the women), then vertical heterogeneity does not hold and there may be more than one weakly stable marriage, as shown in the following example.

**Example 10.** *Consider the following profile:* $\{m_1 : w_1 > w_2 \bowtie w_3;\ \ m_2 : w_1 > w_2 \bowtie w_3;\ \ m_3 : w_1 > w_2 \bowtie w_3;\}$ $\{w_1 : m_1 > m_2 > m_3;\ \ w_2 : m_1 > m_2 > m_3;\ \ w_3 : m_1 > m_2 > m_3\}$. *In this profile all the agents of the same sex have the same preference ordering over the mates of the opposite sex, however, there are two weakly stable marriages, i.e.,* $\mu_1 = \{(m_1, w_1), (m_2, w_2), (m_3, w_3)\}$ *and* $\mu_2 = \{(m_1, w_1), (m_2, w_3), (m_3, w_2)\}$. *Notice however that these two weakly stable marriages differ only for incomparable or tied partners.* □

It is possible to show that if all the agents of the same sex have the same preference ordering over the mates of the opposite sex and there is at least one incomparable or tied pair, then there is a unique weakly stable marriage up to ties and incomparability.

Let us consider now Corollary 24.2 regarding the *horizontal heterogeneity* property. From Theorem 25, it follows immediately that Corollary 24.2 holds also when partially ordered preferences are allowed.

**Corollary 25.1.** *In SMPs, if there is an ordering of men and women such that the preference profile satisfies horizontal heterogeneity, there is a unique weakly stable marriage* $\mu(w_i) = m_i$, $\forall i \in \{1, 2, \ldots, n\}$.

*Proof.* It follows immediately from Theorem 25. □

For partially ordered preferences, we can also guarantee uniqueness of weakly stable marriages by relaxing the horizontal heterogeneity property as follows.

**Theorem 26.** *In an SMP, let us denote with* $m_k$ *is the first man with more than one top choice, if he exists. If there is an ordering of men and women in increasing number of their top choices such that the preference profile satisfies the following conditions:*

- $\forall m_i \in M$ *with* $m_i < m_k$, $w_i >_{m_i} w_j$, $\forall j$;
- $\forall m_i \in M$ *with* $m_i \geq m_k$,

  - *($w_i >_{m_i}$ (or $\bowtie_{m_i}$) $w_j$), $\forall j < i$, and*
  - *($w_i >_{m_i} w_j$), $\forall j > i$;*

- *$\forall w_i \in W$, with $w_i < w_k$, $m_i >_{w_i} m_j$, $\forall j$;*
- *$\forall w_i \in W$, with $w_i \geq w_k$,*
  - *($m_i >_{w_i}$ (or $\bowtie_{w_i}$) $m_j$), $\forall j < i$, and*
  - *($m_i >_{w_i} m_j$), $\forall j > i$,*

*there is a unique weakly stable marriage $\mu(w_i) = m_i$, $\forall i \in \{1, 2, \ldots, n\}$.*

*Proof.* In order to show that that the marriage $\mu(w_i) = m_i$, $\forall i$, is the unique weakly stable marriage that can be obtained for any linearization of the given profile, say $p$, we will show that in every linearization of $p$, the male optimal marriage and the female optimal marriage coincide with the marriage $\mu$.

To show that in every linearization of $p$ the male optimal marriage is $\mu$, we apply the extended version of the men-proposing GS to $p$ and we show that the result is $\mu$ independently of how the non-ordered pairs are ordered. If we apply the extended men-proposing GS algorithm, every man $m_i$, for $i < k$, makes a proposal to his best woman $w_i$, that accepts since for her $m_i$ is her best man, and all the other men of $w_i$ are deleted from her preference list and $w_i$ is removed from these men's preference ranking. This means, that when $w_i$ accepts the proposal from $m_i$, then all the remaining men cannot propose to $w_i$ and thus $w_i$ remains with $m_i$, and so, for every men $m_i$, with $i < k$, $\mu(m_i) = w_i$. Since we are using the extended version of GS and the sets of men and women are ordered, every man $m_i$, for $i \geq k$, will not have in his preference ranking any woman $w_j$ for $j < i$, and thus, by hypothesis, $w_i >_{m_i} w_j$, $\forall j > i$. Therefore, every man $m_i$, for $i \geq k$, has in his top choice only the woman $w_i$ and similarly every woman $w_i$ has in his top choice $m_i$. Hence, for every linearization of the $p$, the marriage returned by the men-proposing GS, that, as we know, is male optimal, is $\mu(m_i) = w_i$, $\forall i$.

Similarly, we can show that for every linearization of $p$ the marriage returned by the women-proposing GS, that, as we know, is female optimal, is $\mu(w_i) = m_i$, $\forall i$. Hence, we can conclude that $\mu$ is the unique weakly stable marriage. □

In words, the conditions above require that every man $m_i$ (resp., woman $w_i$) with a single alternative, i.e., $w_i$ (resp., $m_i$) has as unique top choice $w_i$ (resp., $m_i$), and every $m_i$ (resp., $w_i$) with more than one top choice has

exactly one alternative that must be chosen in every weakly stable marriage, that is, $w_i$ (resp., $m_i$).

## 6.8 Related work

In [30] Gent and Prosser give an exhaustive empirical study of the stable marriage problem with ties and incomplete lists, using a constraint programming encoding of the problem. Then, the encoded problem can be solved using on the shelf CP technology. They present results for the decision problem "Is there a stable matching of size $n$?" and for the optimization problem of finding a maximum or minimum cardinality stable marriage. In particular, regarding the optimization problem of finding the largest stable marriage, their complete method (based on the solution of the CP encoding of the problem using Choco constraint programming toolkit [49]) finds stable marriages of size 9.3 (in average) considering problems of size 10 with no ties. When the amount of ties increases the size increases as well. Our local search approach obtains very similar results using a test set generated in the same way.

Gent and Prosser in [31] give a SAT encoding of the stable marriage problem with ties and incomplete lists. Using such an encoding they obtain very good results in the decision problem of whether there is a perfect matching. Even though in our experiments we often find a perfect matching we consider a different problem from the one solved in [31].

In [11] Brito and Meseguer, propose a distributed approach to the stable marriage problem with ties and incomplete lists with the aim of keeping preference lists private for privacy reasons. They extend some specialized centralized algorithms (such as the Extended Gale Shapley algorithm) to the distributed case. Moreover, they provide a generic distributed constraint programming model. In their experimental evaluation, they consider the communication effort and the computational cost (in terms of constraint checks) which are not applicable to our centralized approach. However, they show also the maximum cardinality of the marriages found by their algorithms considering SMTIs. Considering problems of the same size, probability of ties and, incompleteness they used, we obtain marriages of very similar cardinality.

In [43] Irving and Manlove present two heuristic approaches to find the largest stable matching in the context of the hospital resident (HR) problems with incomplete lists and ties only in the hospitals' preference lists. One of the algorithms is based on the hospital-oriented version of Gale-Shapley algorithm and the other one is based on the resident version. Heuristics are

used to decide how breaking ties in order to maximize the size of the returned marriage. In fact, the ways in which ties are broken can significantly affect the size of the stable matching found and, in the extreme case, there may be two matchings of sizes one the double of the other [54]. When hospitals have capacity equal to 1, the problem becomes an SMTI with ties on one side only, thus the algorithms proposed in [43] can also be used to solve such restricted SMTIs. Moreover, we do not have to deal with tie breaking issues since in algorithm LTIU ties are implicitly handled via the notion of blocking pair.

In [5] the authors give complexity and approximation results regarding the problem of finding a maximum cardinality matching that admits the smallest number of blocking pairs in an SMI. They show that such a problem is NP-hard. Our experimental results show that our local search approach is able to find marriages of large size and with a very small number of blocking pairs within a small number of steps.

As in [39, 44], we permit non-strictly ordered preferences (i.e., preferences may contain ties and incomparable pairs) and we focus on weakly stable marriages. However, while in [39, 44] an algorithm is given that finds a weakly stable marriage by solving a specific linearization obtained by breaking arbitrarily the ties, we present an algorithm that looks for weakly stable marriages that are male optimal, i.e., we look for those linearizations that favor one gender over the other one. Moreover, since there is no guarantee that a male optimal weakly stable marriage exists, we give a sufficient condition on the preference profile that guarantees to find a weakly stable marriage that is male optimal, and we show how to find such a marriage. Other work focuses on providing sufficient conditions when a certain property is not assured for all marriages. For example, in [13] a sufficient condition is given for the existence of a stable roommate marriage when we have preferences with ties.

## 6.9 Conclusions

We have presented two local search approaches for solving stable marriage problems with ties and incomplete lists. Experimental results show that our algorithms are both fast and effective at finding large stable marriages for problems of sizes not considered before in the literature. Moreover, the runtime behavior of the algorithms is not greatly influenced by the amount of incompleteness or ties in the problem. The algorithms were usually able to obtain a very good solution after a very small amount of time.

We have given an algorithm to find male-optimal weakly-stable marriages

when the men's preferences are partially ordered. This may seems in contrast with what we do in Chapter 5, where we aim to develop a fair method to find a stable marriage. What we do in Section 6.6 and in Chapter 5 are different aspect of the general problem of matching elements of two sets. Sometimes we need to have a method that does not favour one group at the expense of the other to generate a stable marriage (which also may be the male or female optimal). In other context we need to clearly favour one group, for example, in the hospital resident problem, the goal is to find the best matching for residents.

The algorithm is sound but not complete. We conjecture, however, that incompleteness is rare, since very specific circumstances are required for our algorithm not to return a male optimal weakly-stable matching when one exists. We have then provided a sufficient condition, which is polynomial to check, for the existence of male optimal weakly-stable matchings. We have also analyzed the issue of uniqueness of weakly-stable matchings, providing sufficient conditions, which are likely to occur in real life problems, that are also necessary in special cases.

# Chapter 7

# Summary and future directions

In this chapter we summarize the results of this thesis and we propose a number of research issues that could be considered in the future.

## 7.1 Summary

In this thesis we have considered how incompleteness and imprecision in preference specification, which are aspect that are very common in real life scenarios, can be handled in an automated reasoning environment. In particular, we have considered various contexts of AI where preferences have a crucial role and we have provided a number of solutions that take into account the inherent difficulty in modeling and handling imprecise or missing preferences.

We have started by extending the soft constraint framework in order to model missing preferences. More precisely, we have modified the notion of constraints to allow for an unknown preference value. In this context, we have considered two notions of optimality: the possibly optimal solutions, that are optimal in at least one way in which preferences can be revealed, and the necessarily optimal solutions, that are optimal in all ways in which preferences can be revealed. Additionally, we have designed a modified version of the classical branch and bound algorithm which interleaves search with preference elicitation in order to find a necessarily optimal solution while eliciting as few preferences as possible. In particular, we have defined a general algorithm scheme that is based on three parameters: when to elicit, what to elicit, and who chooses the value to be assigned to the next variable. We have tested and compared several instances of this general algorithm, by measuring the percentage of the elicited preferences and the user's effort (i.e., the amount of missing preferences that the user has to consider in order to

respond to elicitation queries). The experimental studies have shown that a necessarily optimal solution can be found by eliciting very few preferences (from 5% to 30% of the missing preferences, depending on which c-semiring is used).

We have also considered a local search approach to solve this same problem. Also in this case, we have developed an algorithmic schema where different elicitation strategies can be plugged in. Our empirical results on fuzzy and weighted soft constraint problems show that our local search approach returns solution of high quality when compared with our complete algorithms, while showing better scaling properties.

We have then focus our attention on the extension of the concept of soft constraints to allow for imprecise preferences. We have provided a new formalism based on the modeling of preferences via a preference interval instead of a single value. We have defined new optimality notions based on intervals and we have developed algorithms to find such optimal solutions. Furthermore, we have characterized possibly and necessarily optimal solutions in terms of the notions of optimality based on intervals. Finally, we have provided algorithms to find also such kinds of solutions.

We have then considered stable marriage problems, a class of problems where preferences, often incomplete and/or imprecise, are crucial. We first have considered the classical version of the problem and we have provided a procedurally fair method to generate a stable marriage. We have used a local search approach, based on the removal of undominated blocking pairs, that has shown to have good sampling capabilities over the stable marriage lattice. Furthermore, our local search approach has shown a size independent behavior, which appears to scale well. It is, in fact, able to find a solution in a number of steps which grows as little as $O(nlog(n))$.

Starting from the encouraging results of our local search approach on classical stable marriage problem, we have then focused our attention to the optimization problem of finding a stable marriage with maximum cardinality in the context of stable marriage problems with ties and incomplete preference lists. We have adapted our local search approach to the new setting and experimental results have shown that our algorithm is both fast and effective at finding large stable marriages for problems of large size. Moreover, we developed an alternative local search approach based on tie swaps. However, since both our algorithms reach a solution with very few singles in very few steps, we can conjecture that we could stop the algorithms very soon guaranteeing a solution closer to the optimal.

Finally, we have studied male optimality in the context where men and women express their preferences via partial orders. In particular, we have provided an algorithm to find male optimal weakly stable marriages. Such an

algorithm is based on an extended version of the Gale-Shapely algorithm able to deal with partially ordered preferences. Our algorithm is sound but not complete: it may fail to find a male optimal solution even when one exists. We conjecture, however, that the incompleteness is negligible. Moreover, we have given a sufficient condition on the preference profiles that guarantees to find a male optimal solution, and we have shown how to find it.

## 7.2 Future directions

Starting from the results in this thesis, we have identified several lines of work that we intend to pursue in the future.

We aim to extend the incomplete soft constraint framework in order to model situations where different agents may have different computational or transmission costs. In this context, it would be useful to add a cost function specifying the cost of each preference elicitation query. This may lead to other optimality notions based on the expected cost, that have to be combined with the ones we already know.

We also would like to extend the interval-valued CSP framework with probabilistic information. This will enhance the expressiveness of the framework and will introduce other optimality notions that take into account the probability of a solution to be optimal. For example, we could aim at finding the best solutions among the most probable ones or, conversely, the most probable solutions among the optimal ones. Moreover, we would like to extend this framework to allow for an elicitation process that can be used to elicit one or both of the bounds of an interval. From the algorithmic point of view, we plan to develop incremental algorithms for imprecise soft constraints, that could revise the solution after some of the imprecision is resolved. An incremental algorithm may be applied when information is refined over time, when an interval is elicited to a precise value, or when one (or both) of its bounds becomes stricter. An application of these algorithms may be in interactive systems, where the user may refine intervals with subsequent feedback. Both incomplete and imprecise soft constraints could be applied to the specific problem of recommender systems and/or search in structured catalogs, to minimize the amount of queries to the end user while, at the same time, to give him the best product while allowing for more freedom in preference specification.

As far as stable marriage problems, we would like to use compact representation of imprecision in preference specifications. Also, in the context of stable marriage problems with ties and incomplete preference lists, it would be interesting to allow for unknown preferences. In this setting, we will

also need to understand the impact of different elicitation strategies which may be applied during search. In this context, different scenarios may arise, and therefore notions of necessarily stable or possibly stable marriage should be considered. Finally, there are many other variants of the stable marriage problem, such as the college admissions problem (also known as hospitals/residents problem) where "women" can be married to more than one "man", or the stable roommates problem, in which all participants belong to a single pool. We plan to extend our approaches also to these problems.

# Bibliography


[1] A. Anglada, P. Codognet, and L. Zimmer. An adaptive search for the nscsps. In *Proc. CSCLP 2004*, 2004.

[2] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[3] B. A. Berg. *Markov Chain Monte Carlo Simulations And Their Statistical Analysis: With Web-based Fortran Code*. World Scientific Publishing Company, October 2004.

[4] N. Bhatnagar, S. Greenberg, and D. Randall. Sampling stable marriages: why spouse-swapping won't work. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1223–1232. Society for Industrial and Applied Mathematics, 2008.

[5] P. Biró, D. F. Manlove, and S. Mittal. Size versus stability in the marriage problem. *Theoretical Computer Science*, 411:1828–1841, March 2010.

[6] S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95*, pages 624–630, Montreal, Quebec, Canada, August 20-25, 1995. Morgan Kaufmann.

[7] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, mar 1997.

[8] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3), 1999.

[9] 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. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[10] I. Brito and P. Meseguer. Distributed stable matching problems. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *LNCS*. Springer, 2005.

[11] I. Brito and P. Meseguer. Distributed stable matching problems with ties and incomplete lists. In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006*, volume 4204 of *LNCS*. Springer, 2006.

[12] M. Ceberio and F. Modave. An interval-valued, 2-additive choquet integral for multi-criteria decision making. In *10th Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU'04)*, 2004.

[13] K. Chung. On the existence of stable roommate matching. *Games and economic behavior*, 33:206–230, 2000.

[14] H. L. Cole, G. J. Mailath, and A. Postlewaite. Social norms, savings behavior, and growth. *Journal of Political Economy*, 100(6):1092–1125, December 1992.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[16] R. Cucchiara, E. Lamma, P. Mello, and M. Milano. Interactive constraint satisfaction, technical report deislia-97-00. Technical report, University of Bologna, 1997.

[17] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[18] J. Eeckhout. On the uniqueness of stable marriage matchings. *Economics Letters*, 69(1):1 – 8, 2000.

[19] B. Faltings. Distributed constraint programming. In F. Rossi, P Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2006.

[20] B. Faltings and S. Macho-Gonzalez. Open constraint satisfaction. In *CP2002*, volume 2470 of *LNCS*, pages 356–370. Springer, 2002.

[21] B. Faltings and S. Macho-Gonzalez. Open constraint optimization. In *CP2003*, volume 2833 of *LNCS*, pages 303–317. Springer, 2003.

[22] B. Faltings and S. Macho-Gonzalez. Open constraint programming. *AI Journal*, 161(1-2):181–208, 2005.

[23] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probalistic approach. In M. Clarke, R. Kruse, and S. Moral, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty, European Conference*, volume 747 of *LNCS*, pages 97–104. Springer, 1993.

[24] H. Fargier, T. Schiex, and G. Verfaille. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI-95*, pages 631–637. Morgan Kaufmann, 1995.

[25] E. C. Freuder and A. K. Mackworth. Constraint satisfaction: An emerging paradigm. In F. Rossi, P Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2006.

[26] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[27] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

[28] I. P. Gent, R. W. Irving, D. F. Manlove, P. Prosser, and B. M. Smith. A constraint programming approach to the stable marriage problem. In *Proceedings of CP 2001: The 7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *LNCS*. Springer, 2001.

[29] I. P. Gent, E. Macintyre, P. Prosser, B. M. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6(4):345–372, 2001.

[30] I. P. Gent and P. Prosser. An empirical study of the stable marriage problem with ties and incomplete lists. In *Proc. ECAI 2002*, pages 141–145, 2002.

[31] I. P. Gent, P. Prosser, B. M. Smith, and T. Walsh. Sat encodings of the stable marriage problem with ties and incomplete lists. In *In SAT 2002*, pages 133–140, 2002.

[32] F. Glover. *Tabu search and adaptive memory programming - advances, applications and challenges*. Kluwer Academic Publishers, 1996.

[33] S. Macho González, C. Ansótegui, and P. Meseguer. On the relation among open, interactive and dynamic CSP. In *Proc. of Fifth Workshop on Modelling and Solving Problems with Constraints (IJCAI'05)*, 2005.

[34] G.T. Guilbauld. Les théories de l'intérêt général et le problème logique de l'agrégation. *Économie Appliquée*, 5(4):501–584, 1952.

[35] D. Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM Journal on Computing*, 16(1):111–128, 1987.

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

[37] H. Hoos. On the run-time behaviour of stochastic local search algorithms for sat. In *Proc. AAAI'99*, pages 661–666, 1999.

[38] H. H. Hoos and E. Tsang. Local search methods. In F. Rossi, P Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2006.

[39] R. W. Irving. Stable marriage and indifference. In *CO89: Selected papers of the conference on Combinatorial Optimization*, pages 261–272, Amsterdam, The Netherlands, The Netherlands, 1994. North-Holland Publishing Co.

[40] R. W. Irving. The man-exchange stable marriage problem. Technical report, University of Glasgow, Department of Computing Science, 2004.

[41] R. W. Irving and P. Leather. The complexity of counting stable marriages. *SIAM Journal on Computing*, 15(3):655–667, 1986.

[42] R. W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the "optimal" stable marriage. *J. ACM*, 34(3):532–543, 1987.

[43] R. W. Irving and D. F. Manlove. Finding large stable matchings. *J. Exp. Algorithmics*, 14:2:1.2–2:1.30, January 2010.

[44] R. W. Irving, D. F. Manlove, and S. Scott. The hospitals/residents problem with ties. In *Algorithm Theory - SWAT 2000*, volume 1851 of *Lecture Notes in Computer Science*, pages 515–520. Springer Berlin / Heidelberg, 2000.

[45] K. Iwama, D. F. Manlove, S. Miyazaki, and Y. Morita. Stable marriage with incomplete lists and ties. In *Proc. ICALP*, pages 443–452, 1999.

[46] L. Khatib, P. Morris, R. Morris, F. Rossi, A. Sperduti, and K. B. Venable. Solving and learning a tractable class of soft temporal problems: theoretical and experimental results. *AI Communications*, 20(3), 2007.

[47] D.E. Knuth. *Marriages Stables*. Les Presses du l'Universitè de Montrèal, 1976.

[48] M. Koubarakis. Temporal csps. In F. Rossi, P Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2006.

[49] F. Laburthe. Choco, a constraint programming kernel for solving combinatorial optimization problems. http://choco-solver.net.

[50] E. Lamma, P. Mello, M. Milano, R. Cucchiara, M. Gavanelli, and M. Piccardi. Constraint propagation and value acquisition: Why we should do it interactively. In *Proc. IJCAI'99*, pages 468–477, 1999.

[51] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, pages 497–520, 1960.

[52] J. Lang, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Winner determination in sequential majority voting. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1372–1377, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[53] D. G. Luenberger. *Linear and nonlinear programming*. Addison-Wesley Publishing Co., 1984.

[54] D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002.

[55] P. Meseguer, F. Rossi, and T. Schiex. Soft constraints. In F. Rossi, P Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2006.

[56] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proc. AAAI 1990*, pages 25–32, 1990.

[57] M. S. Pini, F. Rossi, K. B. Venable, and R. Dechter. Robust solutions in unstable optimization problems. In *Recent Advances in Constraints*, LNAI 5783. Springer, 2009.

[58] M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Computing possible and necessary winners from incomplete partially-ordered preferences. In *Proceeding of the 2006 conference on ECAI 2006*, pages 767–768, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.

[59] M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Incompleteness and incomparability in preference aggregation. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1464–1469, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[60] P. Prosser. Private communication. 2010.

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

[62] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proc. ECAI '90*, pages 550–556, 1990.

[63] A. E. Roth and J. H. V. Vate. Random paths to stability in two-sided matching. *Econometrica*, 58(6):1475–1480, 1990.

[64] Z. Ruttkay. Fuzzy constraint satisfaction. In *Proceedings 1st IEEE Conference on Evolutionary Computing*, pages 542–547, Orlando, 1994.

[65] U. Shapen, R. Zivan, and A. Meisels. Meeting scheduling problem (MSP). Available at *http://www.cs.st-andrews.ac.uk/∼ianm/CSPLib/prob/prob046/index.html* , 2010.

[66] K. Stergiou and T. Walsh. Encodings of non-binary constraint satisfaction problems. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 163–168, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[67] T. G. Stutzle. *Local Search Algorithms for Combinatorial Problems - Analysis, Improvements, and New Applications*. PhD thesis, Am Fachbereich Informatik der Technischen Universitat Darmstadt, 1998.

[68] C. Unsworth and P. Prosser. An n-ary constraint for the stable marriage problem. In *The Fifth Workshop on Modelling and Solving Problems with Constraints, held at the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 2005.

[69] C. Unsworth and P. Prosser. A specialised binary constraint for the stable marriage problem. In *Symposium on Abstraction, Reformulation and Approximation (SARA 2005)*, volume 3607 of *LNCS*. Springer, 2005.

[70] T. Walsh. Uncertainty in preference elicitation and aggregation. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 3–8. AAAI Press, 2007.

[71] T. Walsh. Complexity of terminating preference elicitation. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 967–974, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[72] N. Wilson, D. Grimes, and E. C. Freuder. A cost-based model and algorithms for interleaving solving and elicitation of CSPs. In *Proc. CP'07*, LNCS 4741, pages 666–680. Springer, 2007.

[73] N. Yorke-Smith and C. Gervet. Certainty closure: A framework for reliable constraint reasoning with uncertainty. In *Proc. CP'03*, volume 2833 of *LNCS*, pages 769–783. Springer, 2003.