MASARYK UNIVERSITY BRNO
FACULTY OF INFORMATICS

Constraint Satisfaction with Preferences

Ph.D. Thesis

Brno, January 2001 Hana Rudová

Acknowledgements

I would like to thank my supervisor, Luděk Matyska, for his continuous support, guidance, and encouragement. I am very grateful for his help and advice, which have allowed me to develop both as a person and as the avid student I am today.

I want to thank to my husband and my family for their support, patience, and love during my PhD study and especially during writing of this thesis.

This research was supported by the Universities Development Fund of the Czech Republic under contracts # 0748/1998 and # 0407/1999.

Declaration

I declare that this thesis was composed by myself, and all presented results are my own, unless otherwise stated.

Hana Rudová

Contents

Chapter 1

Introduction

First works on constraint satisfaction problems (CSPs) can be traced in the 70'th [Mon74, Mac77, Fre78], but their widespread study and application started since the end of 80'th when the extension of unification algorithm of logic programming [O'K90, MW88, GHR93] led to the proposal of constraint logic programming (CLP) scheme [JL87, VH89]. CSP and CLP allow to express properties of the problem declaratively by means of constraints and to search for its solution via specialized algorithms. The most studied problem instances range over variables with finite domains which probably belong to more than 95 % of all industrial applications of constraints [Sim99, Bar99a]. Constraint programming is applied to solve wide range of problems, among them scheduling, configuration, hardware verification, graphical interfaces, or molecular biology [Sim99, AS99, Wal92, MCF98, FB98, KB99].

In most real-life situations we need to express fuzziness, possibilities, probabilities, costs, weights, . . . In general, human specifications are difficult to express faithfully via hard constraints only. While problems may become over-constrained it does not make sense to say there is no solution. Many problems require finding of best or optimal solution wrt. one or even more optimization criteria. Others may be ill-defined and we need to include uncertainty of the problem definition into final solution. All these problems need to apply some type of *preferences* which have to be included into both declarative and control part of the solution. That led us to the study of *CSPs with preferences* from both theoretical and practical points of view within this thesis.

1.1 Thesis Outline

Chapter 2 is aimed to give an overview of CSP and CLP paradigms as a background for understanding both theoretical and practical parts of thesis. All theoretically oriented Chapters 3–5 follow the same structure we have proposed to give a unifying view to particular approaches for solving CSPs with preferences. That way, notions of constraint, problem, satisfaction degree, solution, and consistency degree create a basic outline of all presented frameworks. Both existing and proposed approaches are applied in the practical part of thesis represented by solution of timetabling problem in Chapter 6.

Chapter 3 summarizes existing approaches for solving CSPs with preferences. Starting from the basic frameworks over particular types of preferences (weighted, probabilistic, possibilistic, fuzzy CSPs) we continue through meta-frameworks (partial, valued, semiring-based CSPs — SCSPs). Basic frameworks may be obtained by specification of a general algebraic structure of meta-framework. In Sect. 3.7.4, we have proposed new relations of equivalence and refinement for SCSPs to be able compare its instances with partial ordering of preferences.

Theory of constraint hierarchies (CHs) is studied in Chapter 4. After basic description of the framework with its comparators for selection of solution, we continue with proposal of new ordered-better and lexicographic-better comparators and their comparison with traditional comparators of CHs in Sect. 4.3. Section 4.4 proposes an algorithm for solving CH with ordered-better comparator. As a main contribution of this chapter, we define instances of algebraic structure of SCSP for certain comparators of CHs (see Sect. 4.5) and classify them into complexity classes with help of proposed equivalence for SCSPs (details in Sect. 4.6). We also show that for remaining comparators any correspondence with SCSP does not exist.

An original contribution of the thesis concerns an idea of assigning preferences to particular variables in constraint. That led us to study of constraints with the so called variables' annotations in Chapter 5. Solutions of constraint system with variables' annotations is defined via fuzzy and hierarchical annotations inspired by existing frameworks fuzzy CSP and CHs. Annotations are also applied to compute variable orderings in CSPs with preferences.

Practical application of discussed approaches for CSPs with preferences is described for timetabling problem in Chapter 6. After description of course timetabling problem and current constraint-based approaches for its solution, we study and propose strategies for solving classroom allocation problem in Sect. 6.3. Section 6.4 is devoted to construction of student-oriented schedules representing problem which still was not solved via CP methodology. Correspondence of problem with weighted and fuzzy CSPs is also presented. Possible application of variables' annotations for timetabling problems is discussed in Sect. 6.5. Proposed methods are applied for solving Faculty of Informatics timetabling problem including construction of individual student schedules for more than 1 000 students. Achieved results are compared with other solution methods and comparable problem instances in last section.

Chapter 7 concludes the thesis discussing possible directions for future research, presenting main contributions of the thesis, and giving summary of the thesis.

Chapter 2

Constraint Satisfaction

This chapter will give a basic overview of constraint satisfaction approach emphasizing those topics we will later concentrate on within the thesis. For further details, we refer to resent book of Marriott & Stuckey [MS98] or Van Henteryck's book [VH89]. Surveys or tutorials on constraint satisfaction my be also found in papers [Rut98, BM95, Kum92, Bar99b, Pug98a] or on web page [Bar98]. For references on constraint logic programming see [JM94, F$^+$93, Mat93].

2.1 Constraint Satisfaction Problem

A constraint satisfaction problem prescribes some requirements for a finite number of variables in the form of constraints. The set of possible values — the domains — for each variable is finite. A constraint tells which value tuples are allowed for a certain subset of all the variables. A constraint can be given either explicitly, by enumerating the tuples allowed, or implicitly, e.g., by an algebraic expression.

Definition 2.1 (constraint, problem) A *constraint satisfaction problem (CSP)* is a triple $P = (V, D, C)$, where

- $V = \{v_1, \ldots, v_n\}$ is the set of variables called *domain variables*;
- $D = \{D_1, \ldots, D_n\}$ is the set of *domains*. Each domain is a finite set containing the possible values for the corresponding variable;
- $C = \{c_1, \ldots, c_n\}$ is the set of *constraints*. A constraint c_i is a relation defined on a subset $\{v_{i_1}, \ldots, v_{i_{k_i}}\}$ of all the variables, that is $\{D_{i_1} \times \cdots \times D_{i_{k_i}}\} \supseteq c_i$.

Set of variables in constraint c will be denoted by V_c. If the set V_c has only one or two elements, we speak about *unary* or *binary* constraints, resp. Remaining constraints are called *non-binary*. A CSP is a *binary CSP*, if all its constraints are unary or binary. Binary CSPs play a special role, as any CSP can be transformed into an equivalent binary CSP by the so called *constraint binarization*. In practice, nevertheless, this transformation sometimes introduces too many additional variables with large domains which may considerably complicate solution of a new problem.

The structure of CSP may be represented by a *constraint graph*, the most simple constraint graph handles binary CSP — vertices correspond to the variables which are connected by an edge iff there is a constraint referring to both variables.

Definition 2.2 *Assignment* is a mapping θ from set of domain variables $Y \subseteq V$ to their corresponding domains, i.e., $\theta(v_i) \in D_i$ for $v_i \in Y$. Set of all assignments on Y will be denoted by Θ_Y.

Let us take variables in Y in any fixed order $(v_1, \ldots, v_m)$. Assignment θ corresponding to $\{(v_1, d_{v_1}), \ldots, (v_m, d_{v_m})\}$ will be written by $\theta = (d_{v_1}, \ldots, d_{v_m})$. Sometimes is spoken about *(value) tuple* $(d_{v_1}, \ldots, d_{v_m})$.

An assignment defined on the set of domain variables V is *complete*, otherwise it is called *partial*. The set of all possible complete assignments $\Theta_V \equiv D_1 \times \cdots \times D_n$ is called *solution space*, in the sense that the solution should be searched within this space.

Definition 2.3 Constraint c is *satisfied* in assignment θ (noted $\theta \vDash c$) if all its variables got a value such that corresponding value tuple belongs to c. Dually constraint is called *unsatisfied*.

For any given constraint $c_i(v_{i_1}, \ldots, v_{i_m})$, we will note $\neg c_i(v_{i_1}, \ldots, v_{i_m})$ which is unsatisfied when c_i is satisfied.

Definition 2.4 (solution) A partial assignment θ is *consistent* if all the constraints referring only to variables assigned by θ are satisfied.

Solution of CSP (V, D, C) is a consistent complete assignment, i.e., all constraints in the set C have to be satisfied.

If a CSP does not have any solution, the problem is called *over-constrained* or *inconsistent* and a CSP with more than one solution is called *under-constrained*. A constraint is *relaxed* if there are further element(s) added to the relation. If elements are removed from the relation, then the constraint is *tightened*.

2.2 Optimization Problem

Within the search for solution we may decide for different kinds of *exploration* of the solution space. We may search for one solutions, all solution, or often for some best solution wrt. given criteria. Such solution is searched within the so called *optimization problem* (or more precisely *constraint satisfaction optimization problem*).

Definition 2.5 *Objective function* F is a function defined from variables V to an ordered set W. Minimal values of W wrt. ordering $\leq_W$ are expected to be more preferred.

An *optimization problem* is a CSP with an *objective function* F.

Considering an optimization problem, an assignment θ is *preferred* to the assignment δ, if the value of the objective function under θ is less than the value under δ, i.e., $F\theta < F\delta$.

An *optimal solution* is such a solution θ that none preferred solution to θ exists.

Often objective function is an expression which evaluates to a real number and whose value should be minimized. Maximization problem is easily transformed into an equivalent minimization problem by simply negating the value of objective function F. However, our general definition also includes multiple criteria optimization problem. In such problem, ordering of possibly multi-dimensional set W should be carefully defined.

Definition of a *feasible* solution allows us to consider still interesting sub-optimal solutions. Such solutions may have the value of objective function constrained by some *threshold* *value*. Selection of appropriate threshold value then becomes critical part during a search for such sub-optimal solutions.

Sometimes particular requirements (constraints) may be even contradictory which complicates selection of any solution. After relaxing some constraints in the over-constrained problem, it can be transformed into an optimization problem where the objective function may express understanding of the “best” possible solution.

2.3 Solution Methods

The basic search algorithm which variations are often applied to find a solution of CSP is a *backtracking*. Basically it assigns values to particular variables extending a partial assignment step by step. Each time a value is instantiated, satisfaction of constraint with already assigned variables is tested. If some of the constraints is violated other of remaining values are considered. Different heuristics define more or less *intelligent backtracking* strategies to recover from dead ends.

Other basic methods are the so called *consistency* (or *constraint propagation*) *techniques* eliminating those values from domains of variables which are inconsistent with any constraint. Starting from *node-consistency* for unary constraints through *arc-consistency* considering binary constraints, we may continue up to *k-consistency* which removes inconsistent values for any system of k variables. Unfortunately pure k-consistency leads to an inefficient search of overall solution space.

The most common way of solving CSPs combines both techniques. Consistency techniques are interwoven with steps of basic search algorithm to boots its performance. The so called *lookahead algorithms* accept a value for some variable after having look ahead whether its assignment would not lead to a dead-end.

For all algorithms based on tree-search, it becomes critical which variables and values are assigned first to avoid deep backtracking. It means that a big part of the variable assignments have to be undone, often repeatedly many times (the so called *trashing*). The reason for this is that the variable, which can not be instantiated properly for a big set of partial assignments, is dealt with too late. *Variable ordering* heuristics are used to judge which variables are the most *critical* to instantiate them first. They may be based on the number of possible values in current domain of variables and on already satisfied and remaining constraints. An example is a *first fail strategy* assigning first variables with the least possible values. Variable ordering may be computed a priori before any search in the solution space starts. Such variable ordering is called *static*. More often the so called *dynamic* variable ordering is applied (e.g., first fail strategy). Such variable ordering selects variable for instantiation during search of solution space each time when some variable should be instantiated. *Value ordering* heuristics decide order of assigning values to a variable. Generally one should take a value which will not have to be reconsidered later on, due to backtracking. Hence, the most *promising* values should be tried first, i.e., those extending partial assignments to a solution.

Besides tree-search algorithms, *structure-driven* or *local* algorithms may be applied. The structure-driven algorithms exploits graph structure of the problem, common methods includes its decomposition. Hill climbing or genetic algorithms belong to a class of local methods stepping from one complete instantiation to another one.

Optimization problems are often solved by the so called *branch-and-bound* algorithm. It needs a heuristic function that associates each partial assignment with some value estimating its quality. The algorithm behaves like backtracking except that as soon as a value is assigned to the variable, the value of heuristic function for the assignment is computed (e.g., current partially evaluated value of objective function). If this value exceeds the bound (e.g., currently best computed value of objective function), then the sub-tree under the current partial assignment is pruned to avoid its useless exploration.

2.4 Constraint Programming

The most natural programming paradigm for combining constraints represents *logic programming* [O'K90, MW88, GHR93]. Logic programming provides an elegant way allowing to separate the logic/declarative and control/search parts of a program. Its best known representative is the logic programming language Prolog (e.g., SICStus Prolog [COC97, Int00], ECL^{i}PS^e [WNS97], CHIP [AB91]). A *constraint logic programming (CLP)* extends standard unification of logic programming by constraint satisfaction having equality as only one of the constraints. However, logic programming is not an essential basis for CLP as this paradigm was inherited by other languages like C++ library ILOG [Pap94, Pug94] or concurrent object-oriented language Oz [Smo95].

The basic structure of constraint logic programs for solving constraint satisfaction problems remains always the same. The first part consists in definition of all the problem variables with their domains. The domains of variables are reduced by constraints which are stated in the next step. The method defining search of solution space is included via *labelling* (or *enumeration*) — the process of generating values of particular domain variables. Described structure may be rewritten into the following CLP code.

```
solve(Variables) :-
     define_variables(Variables),
     state_constraints(Variables),
     labelling(Variables).
```

Tree search can be easily expressed via the following code where variable and value ordering heuristics are processed before a value assignment evoking the constraint propagation.

```
labelling(Variables) :-
     select_variable(Variables, Variable, Rest),
     select_value(Variable, Value),
     Variable #= Value,
     labelling(Rest).
```

Often # symbol is used to distinguish operators of logic programming and constraint logic programming.

Labelling procedure may be also intended to find a solution of optimization problem with some objective function `F`, i.e.,

```
state_constraints(Variables,F), labelling([minimize(F)], Variables)
```

where objective function `F` over variables in `Variables` is defined in the first step and its value is minimized by the second step.

2.4.1 Global Constraints

Constraint binarization may lead to a large increase of solution space such that solution may not be even found in any reasonable time. To avoid this drawback, special constraint propagation algorithms were proposed to solve distinguished sub-problems defined on some subset of variables by the so called *global constraints*. Modelling of problem via suitable global constraints belongs to critical decisions of constraint programming greatly influencing computational efficiency.

Let us introduce global constraints in detail as they are extensively applied within last chapter on timetabling. The description will start from basic global constraints with almost "trivial" propagation and conclude with more special constraints applied for scheduling.

The `element(N, List, Value)` constraint specifies that the `N`-th element of the `List` must have the `Value`. Elements of list `List`, `Value`, and `N` are either domain variables or integers.

The constraint `count(Value, List, RelationalOperator, Count)` is true if `N` is the number of elements of the `List` that are equal to `Value` (only integer) and the statement `N RelationalOperator Count` holds for some of

```
RelationalOperator -> { #= | #\= | #< | #=< | #> | #>= }.
```

This definition generalizes a family of simpler constraints

```
exactly/atmost/atleast(Value, List, Count)
```

which state that exactly/at most/at least `Count` variables of the `List` have the `Value`, resp. Compared with `count` constraint, the meaning of the parameters `Value`, `List`, and `Count` remains the same and value `RelationalOperator` defines its semantics. Combinatorial constraints `exactly`, `atmost`, and `atleast` correspond to `#=`, `#=<`, and `#>=`, respectively.

The family of `among` constraints was introduced in CHIP for solving sequencing problems especially. Let us describe one of the variants of this constraint which helps us to solve timetabling problems.

```
among([Low, Up], [X1,...,Xs], [C1,...,Cs], [V1,...,Vm])
```

constraint has integer parameters `Low` and `Up`, list of domain variables `[X1,...,Xs]`, and lists of integers `[C1,...,Cs]` and `[V1,...,Vm]` with increasing values in `[V1,...,Vm]`. The `among` constraint is true if the following condition holds: at least `Low` and at most `Up` terms among `X1+C1,...,Xs+Cs` take their value in the list of values `[V1,...,Vm]`.

The `among` constraint is another extension of above mentioned `atmost`, `atleast`, and `exactly` constraints.

```
atmost(Value,List,Count) ≡ among([0,Count],List,Zeros,[Value])
atleast(Value,List,Count) ≡ among([Count,0],List,Zeros,[Value])
exactly(Value,List,Count) ≡ among([Count,Count],List,Zeros,[Value])
```

The constraint `alldifferent(List)` (or `alldistinct`) [Rég94, Pug98b] states that a set of variables in `List` must have pairwise distinct n values. While a naive implementation has running time complexity $O(2^n)$, a graph theoretic approach [Rég94] leads to the resulting complexity $O(n^2d^2)$ with $d = \mathrm{card}(D)$. The recent implementation by Puget [Pug98b] based on bound consistency algorithm even achieves complexity $O(n \log n)$.

The `alldifferent` constraint may be also understood as a requirement on scheduling of different *tasks (activities)* of unit length while all of them need one exclusive resource (the so called *unary resource*). The `disjunctive` (or `serialized`) constraint extends applicability of `alldifferent` towards the tasks of variable duration. Generally such constraints belongs to a class of *disjunctive scheduling problems* [CL94, BLP95, Bap98, Nui94, BLP96, PB98] constraining it such a way that it does not allow interruption of particular tasks (*non-preemptive* case). Tasks are specified by a start time `StartJ` and a duration `DurationJ`.

```
disjunctive([Start1,...,StartN], [Duration1,...,DurationN]) ,
```

where each `StartJ` and `DurationJ` are domain variables with finite bounds or integers.

Cumulative scheduling [CL96b, NA96, Bap98, Nui94, BLP96, PB98] constraint has been introduced in CHIP in order to solve scheduling and placement problems [AB93]. It ensures that a resource can run several tasks in parallel, provided that the *discrete resource* capacity is not exceeded. If there are `N` tasks, each starting at a certain start time (`StartI`), having a certain duration (`DurationI`) and consuming a certain amount of resource (`ResourceI`), then the sum of resource usage of all the tasks must not exceed `ResourceLimit` at any time. The constraint syntax follows

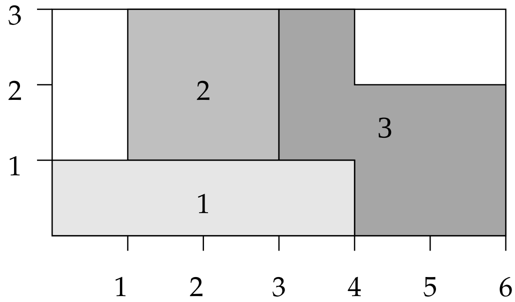
```
cumulative([Start1,...,StartN], [Duration1,...,DurationN],
           [Resource1,...,ResourceN], ResourceLimit) ,
```

where elements of all list together with `ResourceLimit` are domain variables with finite bounds or integers.

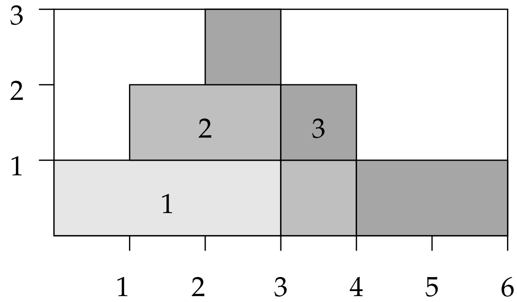
Let us consider major uses of `cumulative` constraint [AB93] in Fig. 2.1. The first example in Fig. 2.1(a) demonstrates full capabilities of `cumulative` constraint and while the example in Fig. 2.1(d) simplifies `cumulative` constraint to the disjunctive scheduling problem with the same application as the `disjunctive` constraint above. Figures 2.1(b) and 2.1(c) present typical application of `cumulative` constraint in timetabling problems. Their special instance may be semantically represented by a set of `atmost` constraints. It corresponds to tasks of unit durations and unit resource capacities (see Fig. 2.1(e)) for which we obtain the following equivalence

```
cumulative(Starts,ListOf1,ListOf1,ResourceLimit) ≡
          {atmost(TimeSlice,Starts,ResourceLimit) | ∀TimeSlice}  .
```

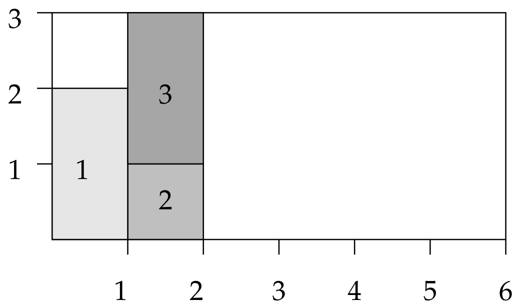
Let us note that all above global constraints express a condition which has to be necessary satisfied. Global constraints which would be able to take into account possibly overconstrained problem definition were efficiently implemented for highly specialized subproblems [BLPP98, Rég99] only.



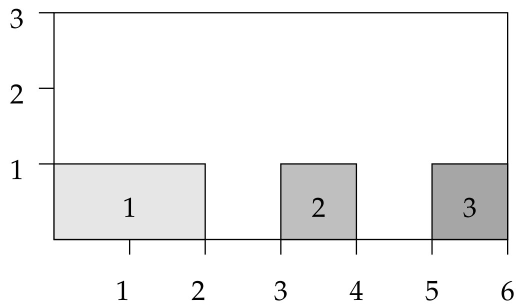
(a) `cumulative`([0,1,3],[4,2,3],[1,2,2],3)



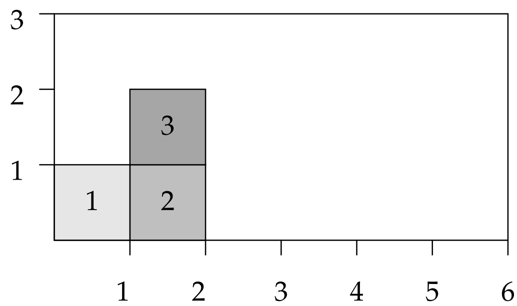
(b) `cumulative`([0,1,2],[3,3,4],**[1,1,1]**,3)



(c) `cumulative`([0,1,1],**[1,1,1]**,[2,1,2],3)



(d) `cumulative`([0,3,5],[2,1,1],**[1,1,1]**,**1**)



(e) `cumulative`([0,1,1],**[1,1,1]**,**[1,1,1]**,3)

Figure 2.1: Major uses of `cumulative` constraint

Chapter 3

Frameworks

Each section of this chapter tries to give a uniform view to particular approaches for solving constraint satisfaction problems with preferences. Definitions of following terms[1] are crucial for understanding of each framework

– constraint, problem, satisfaction degree, solution, consistency degree.

Constraint defines some extension of classical constraint from Def. 2.1. Definition of *problem* naturally extends the notion of CSP wrt. the constraint definition. *Satisfaction degree* evaluates how every assignment satisfies constraints in the problem, i.e., it is aimed to compare particular assignments. The main idea behind definitions of *solution* slightly differs for particular frameworks. Some of them express a solution as a *sufficient* assignment while most of them require its *optimality*. Unifying view to different definitions introduces a *consistency degree* evaluating to which extent optimal assignments satisfy initial conditions. Meta-frameworks which are able to handle basic frameworks will also elucidate the notion of a *structure*. Its specification defines particular frameworks as *classes* (or *instances*) of meta-framework.

3.1 Weighted Constraint Satisfaction

Weighted, penalty, cost-based, or optimization constraint solving refers to the same approach firstly considered in [SH81]. The most general idea behind this framework consists in minimization of weights (costs, penalties) for tuples of values in all constraints. Different definitions consider weights of constraints with the aim to minimize them for unsatisfied constraints. The most studied instance of weighted CSP, the so called *MAX-CSP* (or *maximal* CSP) is specialized to satisfaction of maximal number of constraints [FW92].

Our description will introduce *weight* for each constraint — possible transformation of this approach to preferences of particular tuples is discussed in Sect. 3.8.1.

Definition 3.1 (constraint, problem) A *weighted constraint* is a pair $(c, w(c))$ with c as a classical constraint defined by Def. 2.1 and $w(c) \in W$ giving the weight of constraint c as an element of totally ordered set W. W is ordered by an ordering $\leq_W$ such that smaller weights represent weights of less important constraints.

[1] Particular names are inspired by terminology taken from [DFP94] introducing fuzzy constraint satisfaction.

Weighted constraint satisfaction problem P_ω consists from a set of weighted constraints C restricting possible values of variables from the set V each ranging on a domain D.

The weight of constraint is usually natural number. Already mentioned MAX-CSP may be introduced by constraining the cardinality of the set of all weights W to one.

Definition of assignment together with satisfied and unsatisfied constraint may be taken from the section about constraint satisfaction (see Definitions 2.2 and 2.3).

Definition 3.2 (satisfaction degree) Let us consider weighted CSP $P_\omega = (V, D, C)$. A *satisfaction degree* ω of assignment $\theta \in \Theta_V$ is given by the sum of weights of all constraints which are unsatisfied by assignment θ, i.e.,

$$\forall \theta \in \Theta_V : \omega(\theta) = \sum_{\theta \models \neg c} w(c) \enspace .$$

Definition 3.3 (solution, consistency degree) A *solution of weighted CSP* P_ω is such assignments θ that its satisfaction degree $\omega(\theta)$ is minimal wrt. $\leq_W$.

Satisfaction degree of solution gives us *consistency degree* of weighted CSP.

Example 3.1 Let us consider a trivial example from temporal scheduling. We have to schedule three events into three different time periods. The first one should be scheduled before the second one with weight 10. The third event should be placed right after the first one with smaller weight equal to 5. And if it would be possible we would like to schedule all events in subsequent order, i.e., the first event as the first one with weight 2, the second as a second one with weight 2 and the last event as a last one with the weight 2 too.

The first requirement with the most important weight would require scheduling of first event into the first or second time period while the second event has to be scheduled within the second or third time period. This decision ensures that the first constraint will be satisfied and its weight will not be subsumed into weight of solution. The second constraint requires placement of the third event between the first and second event. This would result into assignment of times in order first, third, and second event. Such assignment satisfies the first and second requirement together with constraint on first event. However, we have switched the second and third events and violated two constraints each having weight 2. This would result into assignment having weight 4. Any other assignment has to violate either first or second requirement which would result into worse satisfaction degree of assignment (at least equal to 5), i.e., we have obtained solution with consistency degree 4.

3.2 Probabilistic Constraint Satisfaction

While the preferences in previous approach express necessity of satisfaction of given constraint in form of its weight, probabilistic approach [FL93] is aimed to consider an ambiguity about the relevance of some constraint to the *real* (in the sense of real-world) problem. As an example we may consider a robot moving in an *ill-known* environment, which knows that there *could* be an obstacle at point (x, y). A preference attached to the constraint then expresses probability degree on its relevance to the ill-known real problem.

This section expects basic knowledge on probabilistic reasoning, for references see introductory books about probability theory [Š87, LM82].

3.2.1 Problem Definition

Definition 3.4 (constraint, problem) The *probabilistic constraint satisfaction problem* P_{pr} consists from a set of variables V, their domains D, and a set of *probability constraints* $C_{\mathrm{pr}} = \{(c_1, p_1), \ldots, (c_m, p_m)\}$ where $C = \bigcup_i c_i$ is a set of classical constraints and p_i is the probability that the constraint c_i is a constraint of the *real* CSP $\mathcal{P} = (\mathcal{C}, V, D)$ having $\mathcal{C} \subseteq C$, i.e.,

$$\Pr(c_i \in \mathcal{C}) = p_i \qquad\qquad \Pr(c_i \notin \mathcal{C}) = 1 - p_i \ .$$

It is assumed that each p_i is greater than 0 (constraints being certainly not relevant are not significant at all) and that the relevance of two different constraints are two *independent* events. Such constraints should be relaxed independently of the others, i.e., the probability that both c_i and c_j belong to $\mathcal{C}$ is

$$\Pr\big((c_i \in \mathcal{C}) \wedge (c_j \in \mathcal{C})\big) = \Pr(c_i \in \mathcal{C}) \times \Pr(c_j \in \mathcal{C}) = p_i \times p_j \ .$$

Probabilistic constraint satisfaction is aimed to find real constraint satisfaction problem $\mathcal{P}$ together with assignment(s) which may be acceptable to this problem. It should be mentioned that we couldn't search directly for solution of CSP $\mathcal{P}$ because there is no guarantee for consistency of problem $\mathcal{P}$.

3.2.2 Problems' Lattice

Let us consider probabilistic CSP $P_{\mathrm{pr}} = (V, D, C_{\mathrm{pr}})$ with $C_{\mathrm{pr}} = \{(c_1, p_1), \ldots, (c_m, p_m)\}$ and denote $P = (V, D, C)$ having $C = \{c_1, \ldots, c_m\}$. The set of possible constraints C defines a lattice of possible classical CSPs, namely 2^P, in which only one problem is the real one $\mathcal{P}$. Having defined this lattice, the notion of *sub-problem* and *super-problem* follows from relations inside it. For an example of problem's lattice, you may see Fig. 3.1 in Example 3.2.

Definition 3.5 Probability distribution pr on 2^P is defined as follows: $P_j \in 2^P$ is the real CSP iff each constraint of P_j is relevant and all other constraints are not[2].

This definition together with independence assumption gives

$$\mathrm{pr}(P_j) = \Pr\big(P_j = \mathcal{P}\big) = \prod\{p_i \,|\, c_i \in C_j\} \times \prod\{1 - p_i \,|\, c_i \in (C - C_j)\}$$

where C_j and C are the set of constraints in P_j and P, respectively.

Example 3.2 Let us consider probabilistic CSP with two variables `A` and `B` having domains $D_{\mathtt{A}} = \{1, 2\}$ and $D_{\mathtt{B}} = \{1, 2, 3\}$. Three probability constraints are given

```
(c1,0.5): (A,B) = {(1,2),(1,3),(2,1)}
(c2,0.6):  A=1
(c3,0.7): (A=2) ∧ (B>1)
```

Lattice of possible problems for this probabilistic CSP is shown in Fig. 3.1 where the number associated with each problem is its probability distribution. As an example, computing probability distribution for problem represented by $\{\mathtt{c2}, \mathtt{c3}\}$ is shown

$$\Pr\big((V, D, \{\mathtt{c2}, \mathtt{c3}\}) = \mathcal{P}\big) = (1 - p_1) \times p_2 \times p_3 = (1 - 0.5) \times 0.6 \times 0.7 = 0.21 \ .$$

[2] Let us remind that definition of probability distribution enforces the following property $\sum_{P_j \in 2^P} \mathrm{pr}(P_j) = 1$.

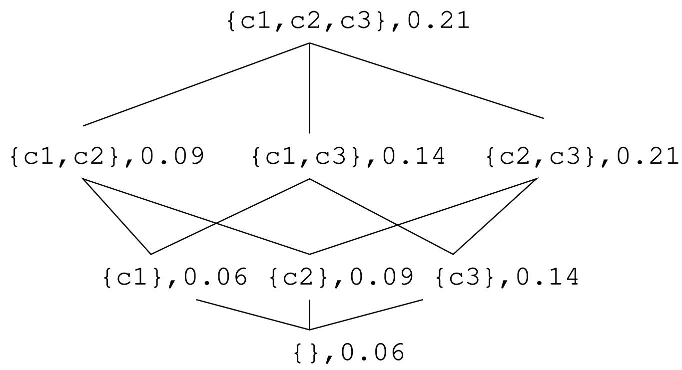


Figure 3.1: Lattice of possible CSPs $P_j = (V, D, C_j)$ each represented by C_j

We should note that the set of constraints $\{\mathtt{c2}, \mathtt{c3}\}$ is inconsistent. An example of consistent[3] problem may be given by the set of constraints $\{\mathtt{c1}, \mathtt{c2}\}$.

As we have seen in our example, some problems in a lattice of problems may be inconsistent while some others are consistent.

Definition 3.6 A consistent sub-problem of P such that each of its strict super-problems is inconsistent is *maximal consistent* in P.

Trivially, a solution of a CSP is also a solution of all its sub-problems. If C is consistent, then classical CSP containing all constraints in C is obviously the maximal consistent sub-problem by itself since in this case, all problems of 2^P are consistent and so is the real problem. Otherwise, the real problem may be inconsistent; the probability that $\mathcal{P}$ is consistent is equal to

$$\mathrm{Pr}(\mathcal{P} \text{ consistent}) = \sum \left\{ \mathrm{pr}(P_j) \,|\, P_j \in 2^P, P_j \text{ consistent} \right\} \ .$$

Let us recall that $\mathrm{Pr}(\mathcal{P} \text{ consistent}) \leq 1$ due to the properties of probability distribution as noted in Footnote 2 on page 13.

Example 3.2 (continuation) Set of constraints $\{\mathtt{c1}, \mathtt{c2}, \mathtt{c3}\}$, $\{\mathtt{c1}, \mathtt{c3}\}$, and $\{\mathtt{c2}, \mathtt{c3}\}$ representing particular problems are inconsistent. The two maximal consistent sub-problems are $\{\mathtt{c1}, \mathtt{c2}\}$ and $\{\mathtt{c3}\}$. Since only $\{\}$, $\{\mathtt{c1}\}$, $\{\mathtt{c2}\}$, $\{\mathtt{c3}\}$, and $\{\mathtt{c1}, \mathtt{c2}\}$ are consistent, the probability that $\mathcal{P}$ is consistent corresponds to

$$\mathrm{Pr}(\mathcal{P} \text{ consistent}) = \underset{\{\}}{0.06} + \underset{\{\mathtt{c1}\}}{0.06} + \underset{\{\mathtt{c2}\}}{0.09} + \underset{\{\mathtt{c3}\}}{0.14} + \underset{\{\mathtt{c1},\mathtt{c2}\}}{0.09} = 0.44 \ .$$

3.2.3 Solution

A naive kind of request would be to solve only the most probable CSPs (using classical CSP techniques), or, since these problems may be inconsistent, to solve one of the most probable consistent problems. However, these problems may have a very low probability and we will also show that their solutions do not necessarily have the highest probability to solve $\mathcal{P}$. It

[3] Let us recall that a problem (V, D, C) is consistent iff $\theta \in \Theta_V$ exits such that $\theta \models C$ (Def. 2.4).

seems to be more reasonable to search for an assignment with a maximal probability to be a solution of $\mathcal{P}$.

Assignment $\theta \in \Theta_V$ is the solution of the real problem $\mathcal{P}$ iff it doesn't violate any constraint included in $\mathcal{P}$.

The probability that θ is a solution of $\mathcal{P} = (V, D, \mathcal{C})$ is the probability that the constraints it violates are not relevant in $\mathcal{P}$, i.e.,

$$\Pr(\theta \vDash \mathcal{C}) = \prod_i \{1 - p_i \,|\, \theta \vDash \neg c_i\} = \sum_j \{\mathrm{pr}(P_j) \,|\, P_j \in 2^P, P_j = (V, D, C_j), \theta \vDash C_j\} \ .$$

Definition 3.7 (satisfaction degree) *Satisfaction degree of probabilistic CSP* P_{pr} corresponds to the probability that θ is a solution of $\mathcal{P}$, i.e., $\Pr(\theta \vDash \mathcal{C})$.

Definition 3.8 (solution, consistency degree) *Solution of probabilistic CSP* P_{pr} is an assignment θ^* which maximizes the probability to be a solution of real problem $\mathcal{P}$, i.e., θ^* is such that

$$\Pr(\theta^* \vDash \mathcal{C}) = \max_{\theta \in \Theta_V} \Pr(\theta \vDash \mathcal{C}) = \max_{\theta \in \Theta_V} \Big(\prod_i \{1 - p_i \,|\, \theta \vDash \neg c_i\}\Big) \ .$$

$\Pr(\theta^* \vDash \mathcal{C})$ gives us *consistency degree of probabilistic CSP* P_{pr}.

It can be proved that θ^* is solution of one of the maximal consistent sub-problems, say $P^* = (V, D, \{c_i \in C \,|\, \theta^* \vDash c_i\})$. It is also a solution of all sub-problems of P^*, but does not satisfy any problem outside this sub-part of the lattice. Besides inequality $\Pr(\mathcal{P} \subseteq P^*) \geq \Pr(\mathcal{P} = P^*)$ holds because $\mathcal{P}$ may also correspond to any of sub-problems of P^*.

It should be outlined that P^* is not necessarily the most probable consistent problem in the lattice; it may even not be one of its super-problems.

Example 3.2 (continuation) As we may see in Fig. 3.1, the most probable problems are given by $\{\mathtt{c1}, \mathtt{c2}, \mathtt{c3}\}$ and $\{\mathtt{c2}, \mathtt{c3}\}$ having their probability equal to 0.21. They have maximal consistent sub-problem $\{\mathtt{c3}\}$ with probability to be a real problem $\mathrm{pr}\big(V, D, \{\mathtt{c3}\}\big) = 0.14$.

On the other hand, the most probable assignments are $\theta^* \in \{(1,2), (1,3)\}$ corresponding to $P^* = \{\mathtt{c1}, \mathtt{c2}\}$. Their probability to be a solution of $\mathcal{P}$ is $\Pr(\theta^* \vDash \mathcal{C}) = 1 - 0.7 = 1 - p_3 = 0.3$ because only the constraint $\mathtt{c3}$ is violated.

3.3 Possibilistic Constraint Satisfaction

This section will formalize the notion of possibilistic constraint satisfaction firstly considered by Thomas Schiex [Sch92]. Behind the proposal of this framework lies Zadeh's possibility theory together with the so called possibilistic logic [DP93].

3.3.1 Possibility Distribution

The main idea is to encapsulate preferences (or respective certainty degree) among labelings in a "possibility distribution" over assignments. Such distribution induces possibility and necessity measures over constraints.

Definition 3.9 Let us consider set of domain variables V together with the set of all possible assignments Θ_V. A *possibility distribution* on Θ_V is a function π from Θ_V to $\langle 0, 1\rangle$.

Assignments with possibility distribution equal to 1 may be compared with assignments of classical constraint satisfaction in the sense that they don't exclude the best possible problem satisfaction while remaining assignments with lower possibility distribution are not able of such full satisfaction. A degree of this inconsistency will be compared with help of the so called *sub-normalization degree*.

Definition 3.10 π is said to be *normalized* iff assignment θ exists such that $\pi(\theta) = 1$ holds. *Sub-normalization degree* of π is defined as the quantity $SN(\pi) = 1 - \max_{\theta \in \Theta_V} \pi(\theta)$.

As you may intuitively expect we will try to find a possibility distribution with the smallest sub-normalization degree. Computing such possibility distribution will lead us to those assignments having value $\pi(\theta)$ maximal.

3.3.2 Problem Definition

The following definition will formalize the best Π_π and worst N_π possible "evaluation" of constraint wrt. selected possibility distribution π.

Definition 3.11 Let C be a set of every possible constraints on any non empty subset of V. *Possibility measure* Π_π and *necessity measure* N_π of possibility distribution π are functions from C to $\langle 0, 1\rangle$ defined by expressions

$$\Pi_\pi(c) = \max_{(\theta \in \Theta_V) \wedge (\theta \models c)} (\pi(\theta), 0) \ \ , \ \ N_\pi(c) = \min_{(\theta \in \Theta_V) \wedge (\theta \models \neg c)} (1 - \pi(\theta), 1) \ \ .$$

Let us remark that from the definition of $\neg c$ follows $N_\pi(c) = 1 - \Pi_\pi(\neg c)$.

Because of $\min$ and $\max$ operators used, the precise values of necessity or possibility are not so important. It is the total pre-order induced by them which is essential. Thus, necessity measure of constraint expresses *preference degree*, $N_\pi(c_1) > N_\pi(c_2)$ expressing that the satisfaction of constraint c_1 is preferred to the satisfaction of c_2.

Definition 3.12 (constraint, problem) *Necessity-valued constraint* is a pair (c, w) where c is defined as a classical constraint by Def. 2.1 and $w \in \langle 0, 1\rangle$ as a *preference degree*.

Possibilistic constraint satisfaction problem P_π consists from set of domain variables V, set of domains D, and set of necessity valued constraints C.

The necessity-valued constraint (c, w) expresses that $N_\pi(c) \geq w$, i.e., that the satisfaction of c is at least w-necessary. The constraint $(c, 1)$ should be absolutely satisfied while the constraint $(c, 0)$ is totally redundant as it expresses that the necessity measure of c should be at least 0, which is always true.

Definition 3.13 (solution) Necessity-valued constraint (c, w) is *satisfied by possibility distribution* π (noted $\pi \models (c, w)$) iff the necessity measure N_π induced by π on C verifies $N_\pi(c) \geq w$.

Solution of possibilistic CSP (V, D, C) is each possibility distribution π that all necessity-valued constraints are satisfied. We will say that π satisfies C.

Compared with the solution of CSP, possibilistic CSP has not a set of consistent assignments on V but a set of consistent possibility distributions on the set of all assignments on V.

Solution of possibilistic CSP doesn't require any optimal satisfaction of all constraints, it is just *sufficient* to find such possibility distribution satisfying all constraints.

Example 3.3 Let us consider possibilistic CSP P with variables `{A,B,C}` having initial domains in `{1,2,3}` and with necessity-valued constraints

```
(a,0.8), a: A < B
(b,0.6), b: C = 1+B
(c,0.4), c: A = 3
(d,0.3), d: A = C
```

Let us take trivial possibility distribution π equal to 1 for all $\theta \in \Theta_V$. Such possibility distribution has minimal sub-normalization degree but it doesn't satisfy any of the constraints because $N_\pi(c) < w$ holds for all (c, w). This is given by the fact that an assignment θ exists for each constraint with the property $\theta \vDash \neg c$, i.e., $N_\pi(c) = \min_{(\theta \in \Theta_V) \wedge (\theta \vDash \neg c)}(1 - \pi(\theta), 1) = 0$. Because the necessity degree of all constraints is higher than 0, this trivial possibility distribution may be classified as an "unsuccessful" solution.

Now we will consider possibility distribution π equal to 0.2 for all $\theta \in \Theta_V$. Then all constraints are satisfied because all necessity degrees are equal to 0.8 and because this value is sufficient wrt. preference degrees of all constraints. It means that this possibility distribution is a solution as all the constraints are satisfied to necessary extent. Even if such possibility distribution is classified to be a solution, its quality seems to be doubtful as its sub-normalization degree $SN(\pi)$ correspond to 0.8.

3.3.3 Consistency

This part will concentrate of selection of such possibility distribution which is not only sufficient but may be considered as the optimal one.

Definition 3.14 (consistency degree) *Consistency degree* $\mathbb{C}$ *of the possibilistic constraint satisfaction problem* (V, D, C) is defined as the maximum of $1 - SN(\pi)$ for every π which satisfies C. *Inconsistency degree* $\mathbb{I}$ is its complement to 1.

Inconsistency degree of possibilistic CSP may be estimated by determining the more important constraint which is not satisfied in any assignment $\theta \in \Theta_V$. It may be shown that inconsistency degree is equal to the smallest necessity degree of the unsatisfiable constraint c_{false} for all possibility distributions satisfying C.

The computation of the inconsistency degree of a possibilistic CSP is made easier by the fact that one can define a maximal possibility distribution satisfying set of constraints C.

Theorem 3.1 (satisfaction degree) Let $P_\pi = (V, D, C)$ be a possibilistic CSP, we define the *maximal possibility distribution* π^*_P on Θ_V by

$$\forall \theta \in \Theta_V : \pi^*_P(\theta) = \min_{((c_i, w_i) \in C) \wedge (\theta \vDash \neg c_i)} (1 - w_i, 1) \ .$$

Then for any possibility distribution π on Θ_V, π satisfies P_π iff $\pi \leq \pi^*_P$.

Proof: $\forall (c_i, w_i) \in C : \pi \vDash (c_i, w_i) \Rightarrow N_\pi(c_i) \geq w_i \Rightarrow \min_{(\theta \in \Theta_V) \wedge (\theta \vDash \neg c)}(1 - \pi(\theta), 1) \geq w_i$
$\forall (c_i, w_i) \in C \ \forall \theta \in \Theta_V$ such that $\theta \vDash \neg c_i : \pi(\theta) \leq 1 - w_i$
$\forall \theta \in \Theta_V : \pi(\theta) \leq \min_{((c_i, w_i) \in C) \wedge (\theta \vDash \neg c)}(1 - w_i, 1) \Rightarrow \pi(\theta) \leq \pi^*_P(\theta)$ □

Informally maximal possibility distribution evaluates to which extent each assignment may be considered as an optimal assignment, i.e., it corresponds to the notion of satisfaction degree as it was introduced at the beginning of the chapter.

Example 3.4 Let us consider possibilistic CSP P from Example 3.3. We will compute possibility distribution π^*_P for all $\theta \in \Theta_V$. First, $\pi^*_P((1,2,3)) = 0.6$ holds due to unsatisfied constraint `c`. Second, $\pi^*_P(\delta) = 0.4$ holds for all assignments satisfying `a` and unsatisfying `b`, e.g., $\delta = (1,2,2)$. All remaining assignments β (these assignments violate `a`) have their possibility distribution $\pi^*_P(\beta)$ equal to 0.2.

Necessity degrees of constraints $N_{\pi^*_P}$ are equal $0.8, 0.6, 0.4, 0.4$ for constraints `a`, `b`, `c`, `d`, respectively. As you may see $\pi^*_P \models C$ because $N_{\pi^*_P}(c) \geq w$ holds for all constraints (c, w) in the problem.

Corollary 3.2 For possibilistic CSP P, we conclude that

- $\mathbb{C}(P) = 1 - SN(\pi^*_P) = \max_{\theta\in\Theta_V} \pi^*_P(\theta)$
- $\mathbb{I}(P) = SN(\pi^*_P) = 1 - \max_{\theta\in\Theta_V} \pi^*_P(\theta)$

Proof: $\forall\pi \models P \Rightarrow \pi \leq \pi^*_P$
$\forall\pi \models P, \forall\theta \in \Theta_V \Rightarrow (1 - \pi(\theta)) \geq (1 - \pi^*_P(\theta))$
$\forall\pi \models P \Rightarrow SN(\pi) \geq SN(\pi^*_P)$
$\mathbb{C}(P) = \max_{\pi\models C}(1 - SN(\pi)) = 1 - SN(\pi^*_P)$ □

Theorem 3.3 Sub-normalization degree of π^*_P is minimal among all possibility distributions π satisfying constraints in P.

Proof: $\forall\pi : \pi \leq \pi^*_P$
$\forall\pi\, \forall\theta \in \Theta_V : \pi(\theta) \leq \pi^*_P(\theta)$
$\forall\pi : 1 - \max_{\theta\in\Theta_V} \pi(\theta) \geq 1 - \max_{\theta\in\Theta_V} \pi^*_P(\theta) \Rightarrow SN(\pi) \geq SN(\pi^*_P)$ □

We have shown in Theorem 3.1 that π^*_P is a maximal possibility distribution, i.e., assignment $\theta \in \Theta_V$ having $\pi^*_P(\theta)$ maximal belongs to those assignments with the highest value of possibility distribution, i.e., $\pi(\delta) \leq \pi^*_P(\theta)$ holds for all possibility distribution π and for all $\delta \in \Theta_V$.

Definition 3.15 (optimal assignment) *Optimal assignment of possibilistic CSP P is each assignment $\theta \in \Theta_V$ such that $\pi^*_P(\theta)$ is maximal, i.e., $\pi^*_P(\theta) = \max_{\delta\in\Theta_V} \pi^*_P(\delta)$ holds.*

The problem of finding optimal assignment then consists in solving any of the following equivalent min-max optimization problems

- $\mathbb{C}(V, D, C) = \max_{\theta\in\Theta_V} \min_{((c_i,w_i)\in C)\wedge(\theta\models\neg c_i)}(1 - w_i, 1)$
- $\mathbb{I}(V, D, C) = \min_{\theta\in\Theta_V} \max_{((c_i,w_i)\in C)\wedge(\theta\models\neg c_i)}(w_i, 0)$

Example 3.4 (continuation) Sub-normalization degree of π^*_P is computed as $SN(\pi^*_P) = 0.2$, which means that consistency $\mathbb{C}(P)$ and inconsistency $\mathbb{I}(P)$ degrees are equal to 0.8 and 0.2, respectively.

There is only one optimal assignment θ having possibility distribution $\pi^*_P(\theta)$ maximal which is the assignment $\theta = (1,2,3)$. Even if this assignment doesn't satisfy constraints `A=3` and `A=C`, it is able to satisfy more important constraints `A<B` and `C=1+B` which excludes satisfaction of any above less important constraints.

Let us note that such definition of optimality leads to the so called *drowning effect*: if a constraint with preference degree w has to be necessarily violated then any constraint with preference degree lower than w is simply ignored and its satisfaction or violation doesn't change possibility degree of final optimal assignment. Possible solutions of this problem will be discussed in the following section as it considers framework having close relationships with possibilistic CSPs.

3.4 Fuzzy Constraint Satisfaction

Possibility theory and calculus of fuzzy restrictions [DP93, MG81] belong to the main areas lying behind the proposal of the *fuzzy constraint satisfaction* framework studied in papers [DFP94, Rut94, DFP96, FLS93].

While possibilistic CSP was meant to express statements like: "It is 0.7 necessary that the product be delivered before the 21th", fuzzy CSP also encapsulates statements such as: "The product should be delivered not too late after 21th". Both these statements are modeled within fuzzy CSP with help of fuzzy relations. Let us shortly introduce their notion (for further details we refer to basic literature about fuzzy sets and fuzzy reasoning [DP93, MG81, Nov86]). *Fuzzy relation* may be seen as a membership degree of k-tuple $(d_1, \ldots, d_k)$ to a cartesian product $D_1 \times \cdots \times D_k$ expressed by membership function μ from $D_1 \times \cdots \times D_k$ to unit interval $\langle 0, 1 \rangle$.

3.4.1 Problem Definition

Definition 3.16 (constraint) Let us consider values $d_1 \in D_1, \ldots, d_k \in D_k$. *(Fuzzy) constraint* c is defined by a fuzzy relation, that assigns to each k-tuple $(d_1, \ldots, d_k)$ its *level of preference* $\mu_c(d_1, \ldots, d_k)$ from unit interval $\langle 0, 1 \rangle$.

Inequality $\mu_c(d_1, \ldots, d_k) > \mu_c(d'_1, \ldots, d'_k)$ means that $(d_1, \ldots, d_k)$ is preferred to $(d'_1, \ldots, d'_k)$ as values for variables $v_1, \ldots, v_k \in V$. $(d_1, \ldots, d_k)$ is a forbidden k tuple if $\mu_c(d_1, \ldots, d_k) = 0$ holds and $\mu_c(d_1, \ldots, d_k) = 1$ means that $(d_1, \ldots, d_k)$ *totally satisfies* the constraint. More generally, $\mu_c(d_1, \ldots, d_k)$ can also be interpreted as degree of satisfaction of the *soft* constraint c.

Constraint c and its fuzzy relation are said to be *normalized* if at least one tuple $(d_1, \ldots, d_k)$ that totally satisfies the constraint c exists, i.e., $\exists d_1, \ldots, d_k \in D$ such that $\mu_c(d_1, \ldots, d_k) = 1$ holds.

Levels of preference within fuzzy relation of fuzzy constraint may express *priorities* with the same interpretation as the preference degree of necessity-valued constraint in possibilistic CSP framework (see Sect. 3.3). If a constraint c should be satisfied with a priority w, we express it by a fuzzy relation

$$\begin{aligned} \mu_c(d_1, \ldots, d_k) &= \quad 1 \quad \Leftrightarrow \quad (d_1, \ldots, d_k) \models c \\ &= \; 1 - w \quad \Leftrightarrow \quad (d_1, \ldots, d_k) \models \neg c \end{aligned} \tag{3.1}$$

We may even change definition of any underlying fuzzy relation of constraint c such that a new constraint c_w is able to model given fuzzy relation together with priority w.

$$\mu_{c_w}(d_1, \ldots, d_k) = \max(1 - w, \mu_c(d_1, \ldots, d_k)) \ . \tag{3.2}$$

Definition 3.17 (problem) *Fuzzy constraint satisfaction problem P_μ* consists from a set of fuzzy constraints $C = \{c_1, \ldots, c_m\}$ restricting the possible values of variables from the set V each ranging on a domain D_i which is supposed to be finite.

Example 3.5 Let us state fuzzy constraint satisfaction problem P having variables `A,B`, domains in $D_1 = D_2 =$ `{1,2,3}` and constraints

```
c1: A = 1 @(1,0.2)
c2: min( abs(A - B) ), abs(A - B) = 0 => @1
                                  = 1 => @0.6
                                  = 2 => @0.3
c3: max (A + B) @(A + B)/10
```

If `c1` is satisfied its preference degree is equal to 1 otherwise it corresponds to 0.2. Preference of constraint may be given by function which is shown by the constraints `c2` and `c3` where the preference of each tuple `(A,B)` depends on the value `abs(A - B)` for `c2` or just corresponds to `(A + B)/10` for `c3`.

Let us expect that the first constraint has a priority 0.3 (for definition of priorities see Eqn. 3.1), the second and the third have the same priority 0.7. In correspondence with definition in Eqn. 3.2, we obtain a new CSP P'

```
c1': A = 1 @(1,0.7)   ⇐ max(1-0.3,1)=1, max(1-0.3,0.2)=0.7
c2': min( abs(A - B) ), abs(A - B)=0 => @1     ⇐ max(1-0.7,1)
                                  =1 => @0.6   ⇐ max(1-0.7,0.6)
                                  =2 => @0.3   ⇐ max(1-0.7,0.3)
c3': max (A + B), A + B > 3 @(A+B)/10          ⇐ max(1-0.7,(A+B)/10)
                  A + B < 4 @0.3
```

3.4.2 Operations

Definition 3.18 (tuple projection) Let us consider two sets of variables $X = \{v'_1, \ldots, v'_k\}$ and $Y = \{v_1, \ldots, v_l\}$ such that $X \subseteq Y \subseteq V$ holds, and any l-tuple $(d_1, \ldots, d_l)$ of values for variables from Y. *Tuple projection* of $(d_1, \ldots, d_l)$ from Y to X written $(d_1, \ldots, d_l) \downarrow^Y_X$, is defined as the tuple $(d'_1, \ldots, d'_k)$ with $d'_i = d_j$ if $v'_i = v_j$.

Example 3.6 Let us consider tuple $(1, 2, 3, 4, 5)$ corresponding to variables `(A,B,C,D,E)`, then $(1, 2, 3, 4, 5) \downarrow^{(\mathtt{A,B,C,D,E})}_{(\mathtt{D,A,E})} = (4, 1, 5)$.

Next definition introduces fuzzy relation for constraint c which estimates to what extent the assignment $(d_1, \ldots, d_k)$ of variables in Z satisfies two fuzzy constraints c_X, c_Y concurrently.

Definition 3.19 (conjunctive combination) The *conjunctive combination* of two constraints c_X, c_Y restricting the possible values of two sets of variables X and Y is a constraint c given by fuzzy relation over the possible values of $Z = X \cup Y$. It is denoted by $c = c_X \oplus c_Y$ and defined by

$$\mu_c(d_1, \ldots, d_k) = \min(\mu_{c_X}((d_1, \ldots, d_k) \downarrow^Z_X), \mu_{c_Y}((d_1, \ldots, d_k) \downarrow^Z_Y)) \ .$$

Definition 3.20 (productive combination) Having the same c_X, c_Y as above, a constraint c defined over $Z = X \cup Y$ is their *productive combination* $c = c_X \otimes c_Y$ iff

$$\mu_c(d_1, \ldots, d_k) = \mu_{c_X}((d_1, \ldots, d_k) \downarrow^Z_X) \times \mu_{c_Y}((d_1, \ldots, d_k) \downarrow^Z_Y) \ .$$

Definition 3.21 (average combination) Let $c_1, \ldots, c_m$ be constraints restricting values from $X_1, \ldots, X_m$ subsequently. Their *average combination* $c = \mathrm{avg}^m_{i=1} c_i$ defined over $X = X_1 \cup \cdots \cup X_m$ is given by

$$\mu_c(d_1, \ldots, d_k) = \frac{1}{m} \sum_{i=1}^{m} \mu_{c_i}((d_1, \ldots, d_k) \downarrow^X_{X_i}) \ .$$

Wrt. the commutativity and associativity of $\oplus$ and $\otimes$ operations, we may denote $c_1 \oplus \cdots \oplus c_m = \bigoplus C$ and $c_1 \otimes \cdots \otimes c_m = \bigotimes C$ for $C = \{c_1, \ldots, c_m\}$. We will also denote by $\bigodot$ the general operation of *combination* with possible substitution $\bigoplus$, $\bigotimes$, and avg.

Note that the productive combination does not differentiate among assignments which fully violate at least one constraint, i.e.,

$$(\exists c_i \in C : c_i(d_1, \ldots, d_k) = 0) \Rightarrow (\mu_{\bigotimes C}(d_1, \ldots, d_k) = 0) \ .$$

The use of the combination rules underlies an assumption of *commensurability* between preference levels pertaining to different constraints, i.e., user who specifies the constraints must describe them by means of a unique preference scale.

Other operation we need is the so called *projection* of constraint c_Y defined over variables from Y *on* a set of variables $X = \{v_{x1}, \ldots v_{xk}\}$. Such operation estimates to what extent the tuple $(d_{x1}, \ldots, d_{xk})$, which is a partial assignment of Y, can be extended to a complete assignment of Y that satisfies c_Y.

Definition 3.22 (projection) Given $X = \{v_{x1}, \ldots, v_{xk}\}$ and $Y = \{v_{y1}, \ldots, v_{yl}\}$ two sets of variables such that $X \subseteq Y \subseteq V$, and a fuzzy constraint c_Y restricting the possible values of Y, the *projection* of c_Y on X is a fuzzy constraint $c = c_Y \Downarrow_X$ restricting the possible values of X. It is defined on $D_{x1} \times \cdots \times D_{xk}$ by

$$\mu_c(d_{x1}, \ldots, d_{xk}) = \max_{((d_{y1}, \ldots, d_{yl}) \in D_{y1} \times \cdots \times D_{yl}) \wedge ((d_{y1}, \ldots, d_{yl}) \downarrow^Y_X = (d_{x1}, \ldots, d_{xk}))} \mu_{c_Y}(d_{y1}, \ldots, d_{yl}) \ .$$

Example 3.5 (continuation) Conjunctive combination of all three constraints $\mathtt{c1} \oplus \mathtt{c2} \oplus \mathtt{c3}$ for tuple $(1, 3)$ corresponds to

$$\mu_{\mathtt{c1} \oplus \mathtt{c2} \oplus \mathtt{c3}}(1, 3) = \min(\mu_{\mathtt{c1}}((1)), \mu_{\mathtt{c2}}((1, 3)), \mu_{\mathtt{c3}}((1, 3))) = \min(1, 0.3, 0.4) = 0.3 \ .$$

Projection of the third constraint on $(\mathtt{B})$ for tuple (1) corresponds to

$$\mu_{\mathtt{c3} \Downarrow_{(\mathtt{B})}}(1) = \max(\mu_{\mathtt{c3}}(1, 1), \mu_{\mathtt{c3}}(2, 1), \mu_{\mathtt{c3}}(3, 1)) = \max(0.2, 0.3, 0.4) = 0.4 \ .$$

3.4.3 Consistency

Given a fuzzy constraint satisfaction problem P_μ with constraints C restricting the possible values of variables, combination $\mu_{\bigodot C}(d_1, \dots, d_n)$ estimates to which extent the variable assignment $(d_1, \dots, d_n)$ satisfies all constraints.

Definition 3.23 (satisfaction degree) Let us consider fuzzy CSP P_μ and a complete value assignment $(d_1, \dots, d_n)$ of variables in P_μ. *Satisfaction degree* of tuple $(d_1, \dots, d_n)$ in P_μ is given by $\mu_{\bigodot C}(d_1, \dots, d_n)$.

Satisfaction degree of $(d_1, \dots, d_n)$ may be also seen as the membership degree to the set $D_1 \times \cdots \times D_n$. The membership degrees discriminate the potential solutions since they induce a complete order over the assignments. Similarly to possibility CSP, this order does not depend on a numerical scale, i.e., it is more qualitative than quantitative.

Considering conjunctive combination, satisfaction degree of tuple $(d_1, \dots, d_n)$ in problem P_μ is equal to the satisfaction degree of the constraint that is the least satisfied by $(d_1, \dots, d_n)$. Among all these tuples (complete assignments) we will search for those with the maximal satisfaction degree.

Definition 3.24 (solution) *Solution of fuzzy CSP P_μ* is such assignment of variables which satisfaction degree is maximal, i.e.,

$$\max_{(d_1,\dots,d_n)\in D_1\times\cdots\times D_n} \mu_{\bigodot C}(d_1, \dots, d_n) \ .$$

Constraints in a fuzzy CSP P_μ are not required to be normalized, which means that the overall problem P_μ may not be normalized and no value tuple may exist with satisfaction degree equal to 1. Having satisfaction degree of each value tuple for variables in a fuzzy CSP we may define its consistency degree.

Definition 3.25 (consistency degree) Let us consider fuzzy CSP P_μ with constraints C, variables $V = \{v_1, \dots, v_n\}$ and their domains $D = D_1 \cup \cdots \cup D_n$. *Consistency degree* $\mathbb{C}$ for the fuzzy CSP P_μ corresponds to

$$\mathbb{C}(P_\mu) = \max_{(d_1,\dots,d_n)\in D_1\times\cdots\times D_n} \mu_{\bigodot C}(d_1, \dots, d_n)$$

Inconsistency degree $\mathbb{I}(P_\mu)$ is defined as its complement $\mathbb{I}(P_\mu) = 1 - \mathbb{C}(P_\mu)$.

As you may see, the consistency degree corresponds to satisfaction degree of solution of fuzzy CSP. Consistency degree may be also defined with help of projection on empty set of variables, i.e.,

$$\mathbb{C}(P_\mu) = \bigodot C \Downarrow_\emptyset \ .$$

Such projection estimates to what extent an empty tuple can be extended to a complete assignment of V that satisfies combination of all constraints. That is exactly computed by consistency degree.

Example 3.5 (continuation) Let us compute solution for the problem P'. First we should know satisfaction degrees for all tuples: $(3,3)$ have degree 0.6, both tuples $(2,3)$ and $(3,2)$ correspond to degree 0.5, tuple $(2,2)$ to 0.4, and all remaining tuples have their satisfaction degree equal to 0.3 wrt. preferences of either second or third constraint. Solution has the maximal satisfaction degree, i.e., assignment $(3, 3)$ is the single solution of P'. Consistency degree of P' is then equal to 0.6.

Conjunctive combination never discriminates solutions which satisfy fuzzy CSP to the same degree, even if some of them satisfy more constraints than other. More precisely, all constraints having preference degree lower then inconsistency degree $\mathbb{I}(P_\mu)$ do not have any effect on how solutions are ranked since only the preference degree of the most important violated constraint is relevant. This already mentioned drowning effect[4] doesn't occur for average and productive combination. In [FLS93], two refinements of the conjunctive principle are discussed based on inclusion and lexicographic ordering to avoid drowning effect without excluding the conjunctive combination.

Let us note that the fuzzy CSP with conjunctive combination has the same semantics as possibilistic CSP taking into account preferences defined over constraints instead over each tuple of constraint (in Sect. 3.3). While their definitions are different, semantic behavior remains the same: assignments with maximal preference degree of the least satisfied constraints are the best one.

3.5 Partial Constraint Satisfaction

Freuder & Wallace [FW92] formalize the notion of partial constraint satisfaction which tries to find optimal solution of over-constrained problem with help of metrics over set of CSPs. Definition of this approach was the first attempt to take into account various frameworks via one common meta-framework.

Definition 3.26 A *problem space* is a partially ordered set $(PS, \leq)$ with PS as a set of CSPs and ordering $\leq$ over these problems. $P_1 \leq P_2$ holds if the set of solutions to P_2 is a subset of the set of solutions to P_1. If $P_1 \leq P_2$ holds and sets of solutions P_1 and P_2 are not equal, we will write $P_1 < P_2$ and say that P_1 is *weaker than* P_2.

The problem may be relaxed by enlarging the domain of a variable, enlarging the domain of constraint, removing a variable, or removing a constraint.

Definition 3.27 (problem) *Partial constraint satisfaction problem* $\langle P, (PS, \leq), M, (N, S)\rangle$ consists from initial problem P, a problem space PS containing P, a metric M on that space, and necessary N and sufficient S solution distances ($S < N$ holds).

Variants of the distance function may compare how many solutions were added relaxing P, count of the P's augmentations needed to get from P to P', or maximal count of the satisfied constraints.

Definition 3.28 (solution) *Solution of partial CSP* $\langle P, (PS, \leq), M, (N, S)\rangle$, is a CSP P' from the problem space PS along with solutions to that problem where the metric distance of P' from P is less than N, i.e., $M(P, P') < N$. A solution is *sufficient* if the distance is less than or equal to S. An *optimal solution* is one where the metric distance of P' from P is minimal over the problem space.

[4] This property was firstly discussed for possibilistic CSPs in Sect. 3.3.

As a satisfaction degree of partial CSP may be considered the value of distance function $M(P, P')$. The value of consistency degree corresponds to the satisfaction degree of optimal solution, i.e., $\min_M M(P, P')$.

All approaches described to solve constraint satisfaction problems with preferences may be seen as a class of partial CSP. For description of this correspondence, you may see Sections 3.6.3 and 3.8.

3.6 Valued Constraint Satisfaction

Valued constraint satisfaction [SFV95, BFM+99, BFM+96, Sch00b] defines valuations over constraints as a basic preferences of this meta-framework. It proposes a general structure based on totally ordered monoid over valuations. Frameworks introduced within this chapter may be defined as a special classes of valued constraint satisfaction problem.

3.6.1 Valuation Structure

To express the fact that a constraint may eventually be violated, each constraint is associated with the so called *valuation* introducing general constraint preference. Valuations are given by the following structure.

Definition 3.29 (structure) A *valuation structure* is defined by $(E, \circledast, \succ, \top, \bot)$ where

- E is a set whose elements are called *valuations*;
- $\succ$ is a total ordering over E;
- $\top$ and $\bot$ are maximum and minimum elements of E given by $\succ$;
- $\circledast$ is a commutative, associative binary operation on E that satisfies
  - identity: $\forall a \in E : a \circledast \bot = a$;
  - monotonicity: $\forall a, a', b \in E : (a \succeq a') \Rightarrow \big((a \circledast b) \succeq (a' \circledast b)\big)$.

From these axioms, it may be also inferred that the element $\top$ is an absorbing element, i.e., $\forall a \in E : (a \circledast \top) = \top$.

The ordered set E allows different levels of violations to be expressed. Commutativity and associativity guarantee that the valuation of an assignment depends only on the set of valuation of the violated constraints, and not on the way they are aggregated. Monotonicity guarantees that the valuation of an assignment that satisfies a set C' of constraints will always be as good as the valuation of any assignment which satisfies a subset of C'.

Two additional properties will be also considered for particular specifications of valuation structure as they significantly influence classification of frameworks defined as classes of valued CSP. The first one characterize the following fact: if something is locally improved, it shouldn't be globally ignored. Valuation structure satisfying such property is *strictly monotonous*, i.e.,

$$\text{strict monotonicity: } \forall a, b, c \in E : \big((a \succ c) \wedge (b \neq \top)\big) \Rightarrow \big((a \circledast b) \succ (c \circledast b)\big) \quad . \tag{3.3}$$

The second important property is the *idempotency* of $\circledast$, i.e.,

$$\text{idempotency: } \forall a \in E : a \circledast a = a \ . \tag{3.4}$$

Such property guarantees that a constraint that is satisfied by all the solutions of a CSP can be added to the CSP without changing its meaning.

Lemma 3.4 Idempotency is incompatible with strict monotonicity as soon as E has more than two elements.

Proof: $\forall a \in E : \bot \succ a \succ \top \overset{\text{identity}}{\Rightarrow} (\bot \circledast a) \succ (a \circledast a) \overset{\text{strict monotonicity}}{\Rightarrow} a \succ (a \circledast a)$ and this is contradictory with idempotency. □

3.6.2 Problem Definition

Definition 3.30 (constraint) A *valued constraint* is a tuple $(c, \varphi(c))$ where c is a classical constraint and φ is a function from set of constraints C to set of valuations E called *valuation of constraint*.

Definition 3.31 (problem) A *valued CSP* P_E is defined by a classical CSP (V, D, C), a valuation structure $S = (E, \circledast, \succ, \top, \bot)$, and by a valuation of all constraints φ, i.e., $P_E = (V, D, C, S, \varphi)$.

Each assignment will be valuated by combining the valuations of all violated constraints using $\circledast$.

Definition 3.32 (satisfaction degree) Given a valued CSP $P_E = (V, D, C, S, \varphi)$ and an assignment $\theta \in \Theta_X, X \subseteq V$, the *valuation of assignment* θ *wrt.* P_E is defined by

$$\nu_{P_E}(\theta) = \underset{(c \in C) \wedge (V_c \subseteq X) \wedge (\theta \models \neg c)}{\circledast} \varphi(c) \ .$$

Let us notice entailment of this definition for valuation structure with idempotent $\circledast$. Once an assignment violates a constraint with valuation w, it doesn't matter how many constraints with valuation w are violated — valuation of assignment is not changed at all.

ν_{P_E} is used to distribute valuations to particular assignments introducing potential solution, i.e., it gives us satisfaction degree for each assignment. With this we are able to define solution of valued CSP.

Definition 3.33 (solution, consistency degree) A *solution of valued CSP* $P_E = (V, D, C, S, \varphi)$ is an assignment θ^* having minimal valuation wrt. to ordering $\succ$. This minimal valuation will be called *valuation of problem* P_E.

Valuation of the problem P_E corresponds to its consistency degree.

As we search for assignment with minimal valuation computed by combining violated constraints by $\circledast$, you may see that the element $\top$ corresponds to unacceptable violation and is used to express hard constraints while $\bot$ element corresponds to complete satisfaction.

3.6.3 Relaxation

Valued constraint satisfaction may be seen as a partial constraint satisfaction (see Sect. 3.5) as it defines relaxation lattice equipped with a distance measure.

Definition 3.34 Let $P_E = (V, D, C, S, \varphi)$ be a valued CSP. A *relaxation of valued CSP P_E* is a classical CSP (V, D, C') where $C' \subset C$.

Relaxations are naturally ordered by inclusion of constraint sets. Let us consider consistent relaxation having the set of constraint C' maximal. This relaxation is a solvable problem with minimal distance from original problem. It is also possible to order relaxations with help of a satisfaction degree of *valuation to particular relaxations*.

Definition 3.35 Given a valued CSP $P_E = (V, D, C, S, \varphi)$ and its relaxation (V, D, C'), the *valuation of relaxation* (V, D, C') is

$$\nu_{P_E}(V, D, C') = \underset{c \in (C - C')}{\circledast} \varphi(c) \ .$$

The valuation of the top of the relaxation lattice, CSP (V, D, C), is obviously $\perp$. The valuations of other relaxations can be understood as a distance to this ideal problem. The best assignments of V are the solutions of the closest consistent problems of the lattice. The following corollary ensures that the order on problems defined by this distribution of valuations is consistent with the inclusion order on relaxations.

Corollary 3.5 Given a valued CSP $P_E = (V, D, C, S, \varphi)$ and $P' = (V, D, C')$ with $P'' = (V, D, C'')$ its two relaxations. Then proposition

$$C' \subsetneq C'' \Rightarrow \nu_{P_E}(V, D, C') \succeq \nu_{P_E}(V, D, C'')$$

holds. If $\circledast$ is strictly monotonic, the inequality becomes strict if the valuation of P_E is not $\top$.

Proof: Let us consider $a = \circledast_{c \in (C'' - C')} \varphi(c)$. Due to monotonicity we obtain $\big(\perp \preceq a\big) \Rightarrow \Big(\big(\perp \circledast \nu_{P_E}(P'')\big) \preceq \big(a \circledast \nu_{P_E}(P'')\big)\Big)$. Finally, conclusion $\nu_{P_E}(P'') \preceq \nu_{P_E}(P')$ is inferred from identity and Def. 3.35. Strict inequality is obtained by application of strict monotonicity instead of monotonicity. □

This corollary shows that strict monotonicity is indeed very desirable property. Its existence guarantees that by removing a violated constraint from constraint set, valuation of corresponding relaxation will be increased. In this case, optimal consistent relaxations are selected to be consistent and to have minimal valuation *together with* maximal constraint set.

Theorem 3.6 Assignment $\theta \in \Theta_V$ is solution of valued CSP P_E iff it is solution of consistent relaxation P' with minimal valuation. At the same time, $\nu_{P_E}(\theta) = \nu_{P_E}(P')$ holds.

Proof: Solution of P_E is an assignment with minimal combination of valuations of constraints which have to be removed from constraint set to obtain consistent solution. Relaxation with minimal valuation has to combine valuations of the "same" constraints. Signs of quotation stands for constraints with identical valuations in case of idempotent $\circledast$. But then their valuations change neither $\nu_{P_E}(\theta)$ nor $\nu_{P_E}(P')$. □

3.6.4 Classes of Valued CSP

By specification of evaluation structure, particular classes of valued CSP are obtained which give us a unifying view to already discussed CSPs with preferences. These specifications are summarized in Table 3.1. We should note that we have excluded fuzzy CSP applying tuple

Framework	E	$\succ$	$\circledast$	$\top$	$\perp$
CSP	$\{0,1\}$	$<$	$\wedge$	0	1
Weighted CSP	$\mathbb{N} \cup \{+\infty\}$	$>$	$+$	$+\infty$	0
Probabilistic CSP	$\langle 0,1 \rangle$	$<$	$\times$	0	1
Possibilistic CSP	$\langle 0,1 \rangle$	$>$	$\max$	1	0

Table 3.1: Summary of specifications $(E, \succ, \circledast, \top, \perp)$

preferences instead of constraint preference — fuzzy CSP may be easily and naturally associated with valued CSP via their straightforward relation with semiring CSP (see Sections 3.7.5 and 3.8).

Classical CSP

General definition of valued CSP includes classical constraint satisfaction (see Sect. 2.1) as the most simple framework. Valuation structure is given by trivial boolean lattice $E = \{0,1\}, 1 \equiv \perp \prec 0 \equiv \top, \circledast \equiv \wedge$ (or $\min$). All constraints have valuations $\top$, i.e., function ν_{P_E} returns for each assignment with at least one violated constraint $\top$ as the maximal element. Valuation of CSP corresponds to $\perp$ (complete satisfaction) only if at least one assignment with all satisfied constraints and valuation $\perp$ exists and this would be selected upon minimization. The operation $\wedge$ is both idempotent and strictly monotonic — set of valuations has only two elements, i.e., their coexistence is not contradictory.

Weighted CSP

Specification of valuation structure corresponds to $(\mathbb{N} \cup \{+\infty\}, >, +, +\infty, 0)$ for weighted CSP (in Sect. 3.1). For MAX-CSP, valuations of all constraints correspond to 1. Weights of violated constraints are cumulated by $+$ for each assignment, valuation of CSP is given by assignment having this sum minimal. The operation $+$ is strictly monotonic.

Probabilistic CSP

This framework assigns to each constraint its probability of existence in the real problem which is ill known. ν_{P_E} has to compute probability that constraints violated by assignment are not relevant (don't exist) in the real problem. Because all constraints are independent we obtain $\nu_{P_E}(\theta) = \prod_{(c \in C) \wedge (\theta \models \neg c)} (1-p)$ where p is the probability of existence of constraint c in the real problem. Each constraint has valuation $\varphi(c) = 1 - p$. Valuations are combined by $\times$ operation as $\circledast$. Constraints with probability 1 may not be violated, i.e., $\top \equiv 0$. Resulting ordering $\succ$ of $E = \langle 0,1 \rangle$ has to correspond to $<$. Among all valuations of assignments, the one with maximal probability (wrt. <) is selected which is in correspondence with minimum computed by ordering $\succ$. Overall valuation structure is then given by $(\langle 0,1\rangle, <, \times, 0, 1)$. Operation $\times$ is again strictly monotonic.

This definition is dual to the one presented in original paper [SFV95, BFM+96] about valued CSP with $\varphi(c) = p$ and $S = (\langle 0,1\rangle, >, \circledast, 1, 0)$. However, this specification expects more complicated definition of operation: $x \circledast y = 1 - (1-x) \times (1-y)$.

Possibilistic CSP

Necessity-valued constraint in possibilistic CSP (see Sect. 3.3) corresponds to constraint with its valuation in valued CSP. Valuation structure is given by tuple $(\langle 0,1\rangle, >, \max, 1, 0)$ which conforms with expected semantics of possibilistic constraint satisfaction: an optimal assignment minimizes preference degree of the most important violated constraint. This class is the only one having idempotent operation together with cardinality of valuation set higher than two.

3.6.5 Variable Valued Constraint Satisfaction

Variable VCSP is the only one framework taking into account preferences over variables showing possible impact of such preferences via a special application. This approach was only briefly described in [LV97, VLS96] and applied for the daily management of an earth observation satellite. There, each variable is associated with some weight expressing its importance. The objective is to produce a partial assignment of the problem variables which satisfies all imperative constraints and minimizes sum of weights of un-assigned variables.

This situation may be expressed with help of general valued CSP framework (in which search for a complete assignment is included): just add to each original domain a special *rejection value*, and associate to each variable a soft unary constraint expressing the cost of assigning it the rejection value. It means that such preferences of variables may be fully handled via preferences of constraints.

3.7 Semiring-based Constraint Satisfaction

Semiring-based constraint satisfaction [BMR97b, BFM+99, BMR95, BMR97a] belongs to the meta-frameworks for solving constraint satisfaction problems with preferences. The framework is based on a semiring structure, where the set of semiring specifies the values to be associated with each tuple of values of the variable domain, and the two semiring operations ($+$ and $\times$) to model constraint projection and combination, resp.

3.7.1 Semirings and SCSP

Definition 3.36 (structure) A *semiring* is a tuple $\langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that

- $\mathcal{A}$ is a semiring set;
- $\mathbf{0}, \mathbf{1} \in \mathcal{A}$;
- $+$ is commutative, associative and $\mathbf{0}$ is its unit element;
- $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element.

A *c-semiring* (constraint-based semiring) is a semiring $\langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that $+$ is idempotent (see Eqn. 3.4) with $\mathbf{1}$ as its absorbing element and $\times$ is commutative.

Let us consider the relation $\leq_S$ over $\mathcal{A}$ such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that: $\leq_S$ is a *partial order*, $\mathbf{0}$ is its minimum, and $\mathbf{1}$ its maximum. The relation $\leq_S$ is used to compare tuples and constraints: if $a \leq_S b$ it intuitively means that b is better than a. Let us emphasize that $\leq_S$ may not be total ordering which is a significant difference from the ordering applied within valued CSP (see Sect. 3.6) having all preferences totally ordered. We say that semiring values $a, b \in \mathcal{A}$ are *incomparable* and denote it by $a <>_S b$ iff $(a + b \neq a) \wedge (a + b \neq b)$.

Other property which may be derived from attributes of c-semiring is the monotonicity of both semiring operations, i.e., $a \leq_S a'$ implies $a + b \leq_S a' + b$ and $a \times b \leq_S a' \times b$ for all $a, a', b \in \mathcal{A}$.

Definition 3.37 A *constraint system* is a tuple $CS = (S, D, V)$ where S is a c-semiring, D is a finite set (the domain of the variables) and V is an ordered set of variables.

Definition 3.38 (constraint) Given a semiring $S = \langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and a constraint system $CS = (S, D, V)$, a *constraint* is a pair (def, con) where $con \subseteq V$ and $def : D^{|con|} \rightarrow \mathcal{A}$.

Therefore, a constraint specifies a set of variables (the ones in con), and assigns to each tuple of values over these variables an element of the semiring.

Definition 3.39 (problem) *SCSP (Soft Constraint Satisfaction Problem)* is a pair (C, con) over a constraint system (S, D, V) where $con \subseteq V$ and C is a set of constraints.

con is the set of variables of interest for the constraint set C, which however may concern also variables not in con.

3.7.2 Operations

Let us recall that tuple projection $t \downarrow^X_Y$ projects value tuple t defined on variables in set X to value tuple over variables in set Y (for details see Def 3.18).

Definition 3.40 (combination) Given $c_1 = (def_1, con_1)$ and $c_2 = (def_2, con_2)$, their *combination* $c_1 \otimes c_2$ is the constraint (def, con) defined by $con = con_1 \cup con_2$ and

$$def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2}) \ .$$

Due to commutativity and associativity of multiplicative operation, we may denote $c_1 \otimes \cdots \otimes c_n$ by $\bigotimes_{i=1}^n c_i = \bigotimes C$ for $C = \{c_1, \ldots, c_n\}$.

In other words, combining two constraints means building a new constraint involving all the variables of the original ones associating to each value tuple over such variables certain semiring element. This semiring element is obtained by multiplying the elements associated by the original constraints to the appropriate sub-tuples.

Definition 3.41 (projection) Given a constraint $C = (def, con)$ and a subset I of V, the *projection* of c over I, written $c \Downarrow_I$ is the constraint (def', con') where $con' = con \cap I$ and

$$def'(t') = \sum_{t / t \downarrow^{con}_{I \cap con} = t'} def(t) \ .$$

Informally, projecting means elimination of some variables. This is done by association to each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables.

Summarizing, combination is performed via the multiplicative operation of the semiring, and projection via the additive operation.

3.7.3 Solution

Definition 3.42 (satisfaction degree, solution) The *solution* of an SCSP $P = (C, con)$ is the constraint

$$Sol(P) = (\bigotimes C) \Downarrow_{con} \quad .$$

That is, all constraints are combined, and then projected over the variables in con. That way, we get the constraint over con which is "induced" by the entire SCSP. Generally this constraint distributes by its def function evaluation on each tuple (assignment), i.e., $(\bigotimes C) \Downarrow_{con}$ may be seen as a satisfaction degree in accordance with its usual meaning.

Definition 3.43 (consistency degree) Semiring value of the optimal solutions is called the *best level of consistency* of an SCSP P and it is defined by $blevel(P) = Sol(P) \Downarrow_{\emptyset}$.

Informally, maximal (wrt. $\leq_S$) semiring value of tuples in solution $Sol(P)$ corresponds to the best level of consistency. Set of those tuples will be referred as a *set of optimal tuples*. In contrast to the Def. 3.42 as a solution, just optimal tuples are conforming with the usual definition of solution as it was introduced at the beginning of this chapter.

3.7.4 Equivalence and Refinement

First let us introduce the notion of equivalence on problems as it is usually presented for semiring-based constraint satisfaction.

Definition 3.44 (constraint ordering) Let us consider two constraints $c_1 = (def, con_1)$, $c_2 = (def, con_2)$ over (S, D, V). Then we define the *constraint ordering* $\sqsubseteq_S$ as the following partial ordering: $c_1 \sqsubseteq_S c_2$ iff $def_1(t) \leq_S def_2(t)$ holds for all tuples t of values from D.

Notice, that if $c_1 \sqsubseteq_S c_2$ and $c_1 \sqsupseteq_S c_2$ then $c_1 = c_2$.

Definition 3.45 (equivalence I.) Let us consider SCSPs $P_1 = (C_1, con)$ and $P_2 = (C_2, con)$. Then we define *problem preorder* $\sqsubseteq_P$ as: if $P_1 \sqsubseteq_P P_2$ if $Sol(P_1) \sqsubseteq_S Sol(P_2)$. If $P_1 \sqsubseteq_P P_2$ and $P_2 \sqsubseteq_P P_1$, then they have the same solutions. Thus we say that P_1 and P_2 are *problem equivalent* and write $P_1 \equiv P_2$.

This notion of equivalence may be useful to show that SCSP framework is monotone $(C \cup C', con) \sqsubseteq_P (C \cup C', con)$.

Another notion of equivalence and refinement may be introduced to compare problems having different semiring structures. We will propose new definitions based on equivalence defined within valued CSP framework [BFM+99, SFV95] comparing problems with total ordering of valuations. However, our extension is able to compare all classes of SCSP including those with partial semiring ordering.

In the following, we will consider two constraint systems $CS = (S, D, V)$ and $CS' = (S', D, V)$ together with SCSPs $P = (C, con)$ and $P' = (C', con)$ having solutions $Sol(P) = (def, con)$ and $Sol(P') = (def', con)$.

Definition 3.46 (refinement) The SCSP P is a *refinement* of P' if for any pair of tuples $t = (t_1, \ldots, t_{|con|})$, $r = (r_1, \ldots, r_{|con|})$ having $t_i, r_i \in D$ such that the proposition $def'(t) <_{S'} def'(r)$ holds then the total ordering $def(t) <_S def(r)$ is implied.

The SCSP P is a *strong refinement* of P' if this property holds for all def_1 and def_2 defined by $(def_1, con_\subseteq) \in C$ and $(def_2, con_\subseteq) \in C'$ having $con_\subseteq \subseteq con$.

The main point is that if P is a refinement of P', then the set of corresponding optimal tuples of P is included in the set of optimal tuples of P'; the problem of finding optimal tuples of P can be reduced to the same problem in P'.

VCSP framework which originally defines refinement doesn't include any partial ordering. Let us study consequences of SCSP definition wrt. existence of incomparable semiring values ($a <>_S b$). First we should note that incomparable semiring values $def'(t)$ and $def'(r)$ in original SCSP P' doesn't entail any relation within refined problem: they may become either comparable or incomparable in P. This conclusion is in correspondence with standard definition of refinement, e.g., within set theory (if something is incomparable within the original partially ordered set, just refinement may be able or even aimed to order it). However, opposite statement has a strict consequence, as demonstrated by Corollary 3.7.

Corollary 3.7 Let us expect that SCSP P is the refinement of P' and consider a pair of tuples $t = (t_1, \ldots, t_{|con|})$, $r = (r_1, \ldots, r_{|con|})$ having $t_i, r_i \in D$ such that $def(t) <>_S def(r)$ then $def'(t) <>_{S'} def'(r)$ is implied (see Fig. 3.2).

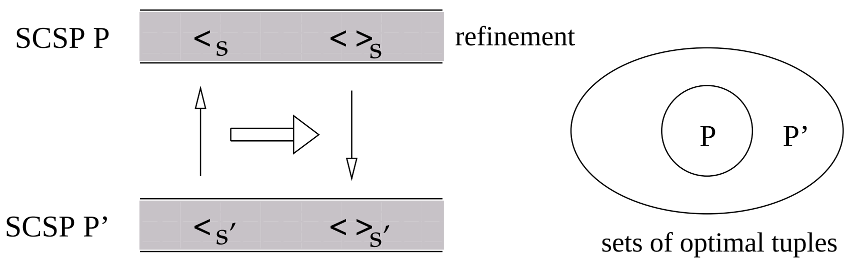


Figure 3.2: SCSP P is a refinement of SCSP P'

Proof: Let us expect that $def'(t) <>_{S'} def'(r)$ doesn't hold. This means that $def'(t)$ and $def'(r)$ are ordered. Having any ordering between these semiring values necessitate the corresponding ordering between $def(t)$ and $def(r)$ due to Def. 3.46. As these elements are incomparable wrt. corollary's presumption we have obtained a contradiction compelling the proof. □

Definition 3.47 (equivalence II.) Two SCSPs P and P' will be said *(strongly) equivalent* iff each one is a (strong) refinement of the other.

Equivalent SCSPs define the same ordering on tuples of variables in V and have the same corresponding sets of optimal tuples: the problem of finding optimal tuples is equivalent in both SCSPs. Note that two equivalent SCSPs are not required to give the same values on tuples but only that they order them similarly, i.e., this definition is weaker than Def. 3.45.

As an entailment of Corollary 3.7 we obtain that if something is incomparable in one of the equivalent problems, it has to be incomparable also in the related problem.

In order to be able to compare SCSP classes that relies on different semiring structures, the notion of polynomial refinement is introduced.

Definition 3.48 (polynomial time refinement) Given S and S', two c-semirings, a *polynomial time refinement* from S' to S is a function ϕ that

- transforms any SCSP $P' = (C', con)$ having set of constraints $C' = \bigcup_i (def'_i, con_i)$ over constraint system (S', D, con) in a $P = (C, con)$ having $C = \bigcup_i (def_i, con_i)$ over (S, D, con) where $def_i = \phi \circ def'_i$ holds for all i such that P is a refinement of P';
- is deterministic polynomial time computable.

If polynomial time refinement from S' to S exists then any SCSP P' over S' can be solved by first applying this polynomial time refinement to P' and then solving the resulting problem over S, i.e., problems defined over S' are not harder than problems defined over S.

3.7.5 Classes of SCSP

By specification of semiring structure, particular classes of SCSP are obtained. These specifications are summarized in Table 3.2.

Framework	$\mathcal{A}$	$+$	$\times$	$\mathbf{0}$	$\mathbf{1}$
CSP	$\{0,1\}$	$\vee$	$\wedge$	0	1
Weighted CSP	$\mathbb{N} \cup \{+\infty\}$	min	$+$	$+\infty$	0
Probabilistic CSP	$\langle 0,1 \rangle$	max	$\times$	0	1
Possibilistic CSP	$\langle 0,1 \rangle$	min	max	1	0
Fuzzy CSP	$\langle 0,1 \rangle$	max	min	0	1
Lexicographic CSP	$\mathbb{N}^{\langle 0,1 \rangle} \cup \{\perp\}$	$\max_{lex}$	$\sqcup$	$\perp$	$\emptyset$

Table 3.2: Summary of specifications $\langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$

Having look on this table and comparing it with the table of valued CSP's classes, a correspondence between semiring-based CSP and valued CSP may be observed easily. More detail justification of this relationship will be described in the Sect. 3.8. Also description of particular classes is similar to those presented in section about valued CSP (Sect. 3.6.4). In comparison with valued CSP's classes we will describe fuzzy CSP together with another framework, the so called lexicographic CSP.

Fuzzy CSP

Introducing fuzzy CSP (see Sect. 3.4) as a class of SCSP is given by straightforward correspondence of both combination and projection operations. Semiring set is defined by interval $\langle 0, 1 \rangle$ having the best element equal to 1 and the worst one to 0. As a result we have obtained semiring $(\langle 0,1\rangle, \max, \min, 0, 1)$ for conjunctive combination. To consider productive combination, $\min$ operation is replaced by $\times$ operation. However, average combination is not possible to include as an instance of SCSP as avg is not associative, which is a basic requirement to semiring structure. Non-existence of this property excludes independence on the order of application of semiring values.

Lexicographic CSP

Lexicographic CSPs [FLS93] were proposed as an extension of fuzzy CSP in order to suppress their drowning effect (see page 23). This extension is also significant for possibilistic CSP where this effect was also observed (see page 19).

Semiring values for class of lexicographic CSPs correspond to multi-sets (see Appendix A for description of multi-sets). An element of semiring set is either an element $\bot$ or a multi-set of elements over $\langle 0,1)$, i.e., $\mathcal{A} = \mathbb{N}^{\langle 0,1)} \cup \{\bot\}$. $\bot$ element is intended for those tuples of constraints which are completely forbidden while the empty multi-set $\emptyset$ represents the best value. Multi-set union $\sqcup$ extended to treat $\bot$ as an absorbing element is used for constraint combination. Projection of constraints is realized with help of lexicographic maximum $\max_{lex}$ (see Eqn. A.5) over multi-sets.

3.8 Relationships between Frameworks

3.8.1 Preferences for Constraints and Tuples

Frameworks studied within this chapter may be divided into two parts. Weighted, probabilistic, possibilistic, and valued CSPs associate a preference with each constraint, i.e., constraint is given as a pair (c, w) consisting from a classical constraint c (see Def. 2.1) and some kind of preference w depending on the type of framework. Fuzzy, lexicographic, and semiring-based CSPs take constraint as a function defined on tuple of values and giving as a result some level of preference, i.e.,

$$c : D_1 \times \cdots \times D_k \to W$$

with $D_1, \ldots, D_k$ as underlying domains of variables in constraint c and W as a set of possible preferences.

Correspondence between those approaches was already observed for fuzzy CSP which is able to model constraints with priorities having the same interpretation as necessity-valued constraints of possibilistic CSP (see Eqns. 3.1 and 3.2).

Generally tuple preferences are able to express constraint preferences easily, e.g., the best possible preference is assigned to all tuples which satisfy given constraint and remaining tuples obtain required (complement of) constraint preference.

Opposite transformation replaces constraint $c : D_1 \times \cdots \times D_k \to W$ by a set of constraints $c_1, \ldots, c_k$, where k is the cardinality of the range of c, i.e.,

$$k = \mathrm{card}\{w \in W \mid \exists \theta \in D_1 \times \cdots \times D_k : c(\theta) = w\} \enspace .$$

However, this transformation may be used if k is finite for all constraints. As an example where this condition is not satisfied we may consider preference given by distance of two real numbers. Sufficient condition to obtain finiteness of k consists in taking only finite domains of variables into considerations. With this we obtain a finite number of assignments, i.e., upper bound of k corresponds to the number of assignments.

3.8.2 Meta-Frameworks

In Sect. 3.7.5, we have already mentioned existing relations between two main presented meta-frameworks, valued and semiring-based CSPs. 3.7.5. We will shortly summarize their formal relation which was studied by the papers [BFM$^+$99, BFM$^+$96] in detail.

The most striking difference lies in the SCSP framework ability to represent partial orders whereas valued CSP exploits a totally ordered sets of the valuations. Partial orders are interesting for multi-criteria optimization since the product of two c-semirings yields a c-semiring defining a partial order.

As the domain of variables within SCSP framework is expected to be finite, transformations of constraints between SCSP and valued CSP frameworks are feasible. The assumption of total order even gives the two frameworks the same theoretical expressive power. The previous section already discussed correspondence between tuple preferences and constraint preferences. Both semiring and valuation structure are related together through replacement of additive semiring operation by $\min$ operation which is applied in valued CSP to compare valuations on different assignments.

The last presented meta-framework, partial constraint satisfaction defines problem space of relaxed problems together with a metric on this space. We have show in Sect. 3.6.3 that partial CSP corresponds to valued CSP defining relaxation lattice together with distance measure.

3.8.3 Basic Frameworks

Particular basic frameworks (weighted, probabilistic, possibilistic, fuzzy, lexicographic CSP) were described as classes of valued and semiring-based CSP. Section 3.7.4 presents the notion of equivalence, refinement, and polynomial time refinement within semiring-based constraint satisfaction. With these definitions in mind, we may shortly summarize known relationships between particular classes as they have been described in [BFM$^+$99, SFV95].

All classes of SCSPs may be partitioned according to the idempotency of $\times$ semiring operator: classical, possibilistic, and fuzzy CSP on one side and weighted, probabilistic, and lexicographic CSP on the other. This partition is an agreement that polynomial transformations between instances of different classes exist for the problem of finding an optimal tuple (or corresponding decision problem of existence of a tuple with a semiring value higher than a given $a \in \mathcal{A}$ can be defined). An isthm between idempotent and non-idempotent classes is provided by lexicographic and possibilistic CSP as polynomial time refinement from possibilistic CSP to lexicographic CSP may be defined.

An exception should be taken into consideration for probabilistic CSP which is related by simple isomorphism with weighted CSP taking values in $\mathbb{R}$ instead of $\mathbb{N}$. Because real numbers are not countable this isomorphism is not a true polynomial refinement. This complication can be fixed by efficient (but approximate) transformation a probabilistic CSP in a weighted CSP, real numbers being approximated by floating point numbers.

Chapter 4

Constraint Hierarchies

Constraint hierarchies [BFBW92, BMMW89] belong to first and favorite approaches which try to handle preferences in constraint systems. In many situations, it is desirable to be able to state both *required* (or *hard*) and *preferential* (or *soft*) constraints. The required constraints must hold. Since other constraints are merely preferences, the system should try to satisfy them if possible, but a solution is found nevertheless if they cannot be all satisfied.

Example 4.1 Let us have two variables `A` and `B` and constraint hierarchy with two preferential levels (`strong` and `weak`)

```
required A + B = 0
strong A = 1
weak B = 0
```

The first required constraint has to be satisfied. The second constraint at the `strong` level can be also satisfied and the solution `{A = 1, B = −1}` is derived. The third weakest constraint is violated due to existing stronger inconsistent constraints.

The theory of constraint hierarchies was studied at the University of Washington by Borning, et al. [WB93, BFBW92, BMMW89, Wil93, WB89]. Logical definition of the soft hierarchical constraints based on the second-order (and later first-order) logic and interpretation ordering were described in [Sat90, SA91]. Later theoretical studies include fuzzifying of hierarchies [KL98], partial ordering of constraints [CL98b] to achieve higher expressivity of the framework, or redefinition of comparators [HMY96] and compositional models [JJGH96, Jam96, Jam95] to improve efficiency of hierarchies.

4.1 Problem Definition

Definition 4.1 (constraint) *Labeled constraint* is a pair $c@l$ having c as a classical constraint defined by Def. 2.1 which is labeled by l defining *level* of constraint in hierarchy. The levels are totally ordered and symbolic names could be assigned to particular levels of constraints. The names are mapped to the integers $0 \dots n$. The level 0, with the symbolic name *required*, is always reserved for constraints which have to be satisfied.

Definition 4.2 (problem) A *constraint hierarchy (CH)* is a finite set C of labeled constraints over variables in a set V having their domains in D. C_0 denotes the set of required constraints in C, with their labels removed. C_1 is a set of the constraints at the strongest non-required level, an so forth through the weakest constraints in C_n, where n is the number of non-required levels in the CH. The set C_k for each $k > n$ is empty.

Such CH will be denoted by $P_{CH} = (V, D, \bigcup_{i=0}^{n} C_i)$. Often we will refer to CH by its set of constraints C only.

Definition of an *assignment* θ and a *set of assignments* θ_V of CH $(V, D, \bigcup_{i=0}^{n} C_i)$ together with *satisfied* and *unsatisfied* constraint in CH corresponds with Defs. 2.2 and 2.3 taking as a set of constraints all constraints from $\bigcup_{i=0}^{n} C_i$.

Definition 4.3 (solution) A *solution* to a CH C is such assignment θ for the variables V in C that satisfies at least required constraints. In addition, the solution satisfies non-required constraints at least as well as any other assignment that also satisfies required constraints, i.e., *solution set* S corresponds to

$$\begin{aligned} S_0 &= \{\theta \mid \forall c \in C_0 \; c\theta \text{ holds}\} \\ S &= \{\theta \mid \theta \in S_0 \wedge \forall \delta \in S_0 \; \neg \mathit{better}(\delta, \theta, C)\} \end{aligned}$$

where *better* is the so called *comparator* which is an irreflexive and transitive relation comparing two assignments of hierarchy.

We also say that solution θ is *better for CH C* if no assignment better than θ exist.

However, in general, *better* will not provide a total ordering — there may exist θ and δ such that θ is not better than δ and δ is not better than θ.

To compare assignments, we will need some measure of how well a particular assignment satisfies a given constraint. We will define the so called *error function* indicating satisfaction of the constraint wrt. assignment $\theta \in \Theta_V$.

Definition 4.4 *Error function* e is a function $e : C \times \Theta_V \to \mathbb{R}_+$ with the property

$$\forall \theta \in \Theta_V \; \forall c \in C : e(c\theta) = 0 \Leftrightarrow \theta \models c \; .$$

We will refer $e(c\theta)$ as an *error of constraint* c wrt. the assignment θ.

Trivial (or *predicate*) *error function* may be applied for all domains D. This function defines value for $e(c\theta)$ to be equal 1 for remaining undefined function values, i.e., $e(c\theta) = 1$ for unsatisfied constraint c in assignment θ. When the domain D is a metric space with distance function dist, *metric error function* may be defined. We can specify such function by

$$\forall d_1, d_2 \in D : e(d_1 = d_2) = \mathrm{dist}(d_1, d_2) \; .$$

The following definition allows us to describe satisfaction degree of each assignment of CH[1].

Definition 4.5 Let us consider set of constraints C with its constraints taken in any fixed order $(c_1, \ldots, c_k)$. An *error of constraint set* $C = (c_1, \ldots, c_k)$ wrt. assignment $\theta \in \Theta_V$ is defined by tuple $E(C\theta) = [e(c_1\theta), \ldots, e(c_k\theta)]$.

[1]General definition of satisfaction degree for all frameworks is described on page 11.

4.2 Traditional Comparators

A number of comparators was defined, each of which gives rise to a different way of defining the set of solutions to a CH. We can classify types of comparators as global, local, or regional. For a *local comparator*, each constraint is considered individually. *Global comparators* aggregate the errors of all constraints at a given level. *Regional comparator* still considers each constraint individually but unlike local comparators, two solutions that are incomparable at higher levels may be still compared at lower levels.

Local Comparator

Definition 4.6 An assignment θ *locally-better* than another assignment δ must do exactly as well as δ for all constraints in levels $1 \dots k-1$, and at level k, θ must do at least as well as δ for all constraints, and strictly better for at least one.

$$
\begin{aligned}
&\text{locally-better}(\theta, \delta, C) \equiv \\
&\quad \exists k \in 1 \dots n \text{ such that} \\
&\qquad \forall i \in 1 \dots k-1\ \forall c \in C_i\colon e(c\theta) = e(c\delta) \\
&\qquad \wedge\ \exists c \in C_k\colon e(c\theta) < e(c\delta) \\
&\qquad \wedge\ \forall d \in C_k\colon e(d\theta) \leq e(d\delta)
\end{aligned}
$$

A *locally-predicate-better* is the locally-better comparator using the trivial error function and a *locally-metric-better* is the locally-better comparator using a domain metric for computing the error of constraints.

To obtain comparison with satisfaction degree of other approaches discussed in Chap. 3, we will define an error over assignments of CH by an array of values of error functions over particular levels of CH and over their constraints.

Definition 4.7 (satisfaction degree I.) An *error* $\mathbb{E}$ *of CH* $P_{CH} = (V, D, \bigcup_{i=0}^{n} C_i) = (V, D, C)$ *for locally-better comparator* is a function defined over assignments $\theta \in \theta_V$ such that

$$
\mathbb{E}(C\theta) = \begin{cases} [E(C_1\theta), \dots, E(C_n\theta)] & \text{if } \forall c \in C_0 : \theta \models c \\ +\infty & \text{if } \exists c \in C_0 : \theta \models \neg c \end{cases}
$$

Having definition of error for CH, we may consider also a dual definition of CH solution corresponding to an assignment with minimal error $\mathbb{E}$ where the notion of minimization is in accordance with comparator definition. The error of solution would give us consistency degree for CH with locally-better comparator. Similar dual definitions may be taken into account for all remaining comparators.

Regional Comparator

Regionally-better comparator extends locally-better comparator to enable comparison of assignments which are incomparable by locally-better comparators.

Definition 4.8 An assignment θ is *regionally-better* than another assignment δ if, for each level through some level $k-1$, the levels are incomparable, and at the level k the value of error function is strictly less for at least one constraint and less than or equal for all the rest.

$$
\begin{aligned}
&\text{regionally-better}(\theta, \delta, C) \equiv \\
&\quad \exists k \ \in 1 \ldots n \text{ such that} \\
&\qquad \forall i \in 1 \ldots k-1 \\
&\qquad\quad \forall c \in C_i \colon e(c\theta) = e(c\delta) \\
&\qquad\quad \vee \ \exists c, d \in C_i \colon e(c\theta) < e(c\delta) \wedge e(d\delta) < e(d\theta) \\
&\qquad \wedge \exists c \in C_k \colon e(c\theta) < e(c\delta) \\
&\qquad \wedge \forall d \in C_k \colon e(d\theta) \leq e(d\delta)
\end{aligned}
$$

Regionally-metric-better and *regionally-predicate-better* comparators apply corresponding error functions on regionally-better comparator.

An *error of CH for regionally-better comparator* remains the same as for locally-better comparator (see Def. 4.7).

Global Comparators

Finally we will define a schema globally-better for all global comparators. This schema is parametrized by a function g that combines the errors of all the constraints at a given level.

Definition 4.9 An assignment θ is *globally-better* than another assignment δ if, for each level through some level $k-1$, the combined errors $g(C_i, \theta)$ of the constraints after applying θ are equal to that after applying δ, and at the level k it is strictly less.

$$
\begin{aligned}
&\text{globally-better}(\theta, \delta, C, g) \equiv \\
&\quad \exists k \in 1 \ldots n \text{ such that} \\
&\qquad \forall i \in 1 \ldots k-1 \colon g(C_i, \theta) = g(C_i, \delta) \\
&\qquad \wedge \ g(C_k, \theta) < g(C_k, \delta)
\end{aligned}
$$

Particular global comparators are given by specification of function g. Such general definition allows us to define satisfaction degree for all global comparators by the following.

Definition 4.10 (satisfaction degree II.) An *error* $\mathbb{E}$ *of CH* $P_{CH} = (V, D, \bigcup_{i=0}^{n} C_i) = (V, D, C)$ *for global comparator* defined by a function g is given for each assignment $\theta \in \theta_V$ such that

$$
\mathbb{E}(C\theta) = \begin{cases} [g(C_1, \theta), \ldots, g(C_n, \theta)] & \text{if } \forall c \in C_0 : \theta \models c \\ +\infty & \text{if } \exists c \in C_0 : \theta \models \neg c \end{cases}
$$

Many global comparators introduce weights for constraints expressing how constraints at the same level are comparable each other.

Definition 4.11 A *weight* of each constraint is defined by function $w : C \to W$ where W is a totally ordered set. Let $\leq_W$ be an ordering of the set W then $w(c) \leq_W w(d)$ means that the weight of d is more preferred than the weight of the constraint c.

A weight can be represented by the set of natural numbers $\mathbb{N}$ or positive real numbers $\mathbb{R}_+$.

Following g functions are usually applied to define particular global comparators

$$\textit{weighted-sum-better}: \; g(C_i, \theta) \equiv \sum_{\{c \in C_i\}} w(c)\, e(c\theta) \quad ; \tag{4.1}$$

$$\textit{worst-case-better}: \; g(C_i, \theta) \equiv \max_{\{c \in C_i\}} w(c)\, e(c\theta) \quad ; \tag{4.2}$$

$$\textit{least-squares-better}: \; g(C_i, \theta) \equiv \sum_{\{c \in C_i\}} w(c)\, e(c\theta)^2 \quad . \tag{4.3}$$

Orthogonal to the choice of particular global comparator, we can select an appropriate error function for the constraints. This way comparators *weighted-sum-predicate-better*, *weighted-sum-metric-better*, and so forth, may be defined.

The so called *unsatisfied-count-better* is a special case of weighted-sum-predicate-better comparator, using weights of 1 on each constraint; it counts the number of unsatisfied constraints in making comparisons.

4.3 New Comparators

This section will concentrate on the proposal of a new global comparator and the description of a local comparator we have firstly introduced in [Rud98a] and discussed its properties in [Rud98c].

4.3.1 Lexicographic-Better Comparator

The disadvantage of the worst-case better comparator is the so called *drowning effect*: if a constraint c with weight $w(c)$ has to be necessarily violated then any constraint at the same level with weight lower than w is simply ignored and its satisfaction or violation doesn't change satisfaction degree of final solution.

Example 4.2 Let us expect that we need to organize two meetings during weekdays of one week (variables `A,B`, domains `{mon..fri}`). Each meeting has to be organized in a different day (if `A` is considered as a first of them then `A<B` holds). Several strong requirements are given by organizers and invited speakers. Organizers would prefer at least two free days between meetings (`B-A≥3@strong,w=20`). Two of invited speakers would like to meet on Wednesday (`A,B=wed@strong,w=10` where `A,B=wed` corresponds to `(A=wed)`$\vee$`(B=wed)`) and the other one prefers Monday (`A,B=mon@strong,w=5`). The last one prefers Thursday (`A,B=thu@strong,w=5`). Requirements of remaining participants are not so important but they may be considered as a secondary constraints: some people would appreciate Tuesday (`A,B=tue@weak,w=10`) and few others would welcome Monday and Thursday, resp. (`A,B=mon@weak,w=5` and `A,B=thu@weak,w=5`).

Constraint hierarchy having two variables `A,B` with domains `{mon..fri}` looks like

```
required: c0: A<B
strong:   c1: B-A≥3        w=20
          c2: A,B=wed      w=10
          c3: A,B=mon      w= 5
```

```
          c4: A,B=thu      w= 5
weak:     c5: A,B=tue      w=10
          c6: A,B=mon      w= 5
          c7: A,B=thu      w= 5
```

Worst-case-predicate-better comparator selects as a solution assignment $\theta = (\texttt{tue}, \texttt{fri})$ satisfying only constraints `c0`, `c1`, and `c5` with $\mathbb{E}(C, \theta) = [10, 5]$. It means that requirements of all invited speakers are violated even if there exist assignments partially satisfying their requirements. This undesirable behavior of worst-case-better comparator is influenced by the high weight of `c2` which has to be violated wrt. `c0` and `c1` (`[c0` $\wedge$ `c1]` $\Rightarrow$ `[(A in mon,tue)` $\wedge$ `(B in thu,fri)]` $\Rightarrow$ `[A,B`$\neq$`wed]`). As a consequence, violation or satisfaction of `c3` and `c4` is not able to change the value of $g(C_1, \theta)$. Subsequently the most important constraint at the weak level (`c5`) outweighs solution of the hierarchy towards $(\texttt{tue}, \texttt{fri})$.

Drowning effect was already observed for fuzzy and possibilistic constraint satisfaction problems (see sections 3.4 and 3.3). Within fuzzy constraint satisfaction framework, refinements of an order of assignments were proposed by the paper [FLS93] and discussed in section 3.7.5. This solution is based on set inclusion and lexicographic ordering. We will apply similar method and propose *lexicographic-better* comparator. This comparator belongs to the class of global comparators, application of g to any assignment θ of level C_i will correspond to multi-set, however. It means that equality and inequality in globally-better schema must be replaced by operations over multi-sets.

Definition 4.12 A *lexicographic-better comparator* is defined by the schema for global comparators (see Def. 4.9) such that for each assignment θ and level $C_i = \{c_1, \ldots, c_{m_i}\}$ of CH C the value of $g(C_i, \theta)$ corresponds to multi-set $M = \{w(c_1)e(c_1\theta), \ldots, w(c_{m_i})e(c_{m_i}\theta)\}$, i.e., $g(C_i, \theta) \in \mathbb{N}^{\mathbb{N}}$ holds for $e(c\theta), w(c) \in \mathbb{N}$. Accordingly equality $=$ together with total ordering $<$ for globally-better schema is replaced by equality and inclusion $\sqsubset$ over multi-sets (Eqn. A.2).

It means that lexicographic-predicate-better comparator adds to multi-set $g(C_i, \theta)$ either $w(c)$ ($\theta \models \neg c$) or 0 ($\theta \models c$) for each constraint $c \in C_i$.

Because the definition of lexicographic-better comparator applies globally-better schema, the satisfaction degree for this comparator is given by Def. 4.10.

Example 4.1 (continuation) The assignment $\delta = (\texttt{tue}, \texttt{fri})$ which was selected by worst-case-predicate-better comparator has the value $g(C_1, \delta)$ equal to $\{0, 10, 5, 5\}$ wrt. violated `c2`, `c3`, and `c4`. $g(C_2, \delta)$ corresponds to $\{0, 5, 5\}$ as `c6` and `c7` are unsatisfied. However, assignment $\theta = (\texttt{mon}, \texttt{thu})$ which lexicographic-predicate-better comparator selects as a solution is able to satisfy strong constraints `c1`,`c3`, and `c4` and also weak constraints `c6` and `c7`. This solution evaluates particular $g(C_i, \theta)$ to $\{0, 10, 0, 0\}$ and $\{10, 0, 0\}$ and it is selected due to the best satisfaction of constraints at the higher level (`c2` has to be violated in each solution due to required `c0` and strong `c1` with higher weight). Each assignment satisfying strong constraints `c1`, `c3`, `c4` has to violate `c5` ([`c1` $\wedge$ `c3`] $\Rightarrow$ $\neg$`c5`) and satisfy `c6` (`c3` $\Rightarrow$ `c6`) and `c7` (`c4` $\Rightarrow$ `c7`).

4.3.2 Ordered-Better Comparator

Another possibility considering local approach to errors of constraints is introduced by a comparator, which uses an ordering (weights) of constraints at every level.

Definition 4.13 An assignment θ is *ordered-better* than another assignment δ if, for each of the constraints through some level $k-1$, the error after applying θ is equal to that after applying δ, and at the level k the errors are compared with help of constraint weights $w(c)$ from Def. 4.11.

$$\begin{array}{l} \text{ordered-better}(\theta, \delta, C) \equiv \\ \quad \exists k \in 1 \ldots n \text{ such that} \\ \qquad \forall l \in 1 \ldots k-1 \; \forall c \in C_l : \; e(c\theta) = e(c\delta) \\ \qquad \wedge \; \exists c \in C_k : \; e(c\theta) < e(c\delta) \\ \qquad \wedge \; \forall d \in C_k \text{ such that } w(c) \leq_W w(d) : \; e(d\theta) \leq e(d\delta) \; . \end{array}$$

The following part clarifies relations between ordered-better and locally-better comparators. In the following, we suppose that $C = \{c_1, c_2, \ldots, c_m\}$ is a CH with levels $\bigcup_{i=0}^{n} C_i$ having weights given by W, $\leq_W$, and w.

Lemma 4.1 Every ordered-better solution θ of hierarchy C is locally-better.

Proof: Let us assume that ordered-better assignment θ is not locally-better. Then an assignment ω exists which is locally-better than θ. Next let C_k be the first level, where assignments ω and θ have different values of error function on some constraints. Because the assignment ω is locally-better than assignment θ:

$$\forall d \in C_k : \; e(d\omega) \leq e(d\theta) \; . \tag{4.4}$$

The level C_k is the first level where any error functions differ, so the next proposition follows from ordered-better(θ, ω, C) (see Definition 4.13)

$$\exists d \in C_k : e(d\theta) < e(d\omega) \tag{4.5}$$

which is contradictory with the proposition (4.4). As the result, we obtain that no solution ω locally-better than θ exists and the assignment θ has to be the locally-better solution. □

There are locally-better solutions, which are not ordered-better. For example, let us consider hierarchy $C = C_1 = \{c, d\}$ where $w(d) <_W w(c)$ holds. Let there exist solutions ω and θ such that $e(c\omega) > e(c\theta)$, $e(d\omega) < e(d\theta)$. Both solutions could be locally-better but only θ could be ordered-better because it is ordered-better than ω.

The next part concentrates on exact specification of relation between locally-better and ordered-better comparator.

Definition 4.14 Let $C = \bigcup\limits_{i=0}^{n} C_i$ be hierarchy and $w : C \rightarrow W$ weight function. *Hierarchy refinement* C/w is defined by $\bigcup\limits_{i=0}^{n} C_i/w$, $C_i/w = \bigcup\limits_{j=1}^{n_i} C_{ij}$ if the proposition $C_0/w = C_0$ holds and C_{ij} is given for $\forall i \in 1 \ldots n$, $\forall j \in 1 \ldots n_i$ by a formula $(\forall c \in C_i \forall d \in C_i : (c \in C_{ij}, d \in C_{il}, l \in 1 \ldots n_i, j < l) \Leftrightarrow (w(d) <_W w(c)))$.

Because the weights have no meaning in required level we can suppose the same weights for every $c \in C_0$. So $C_0/w = C_0 = C_{01}$ is justified. The level C_0 is required and all constraints have to be satisfied for every solution. Therefore, we may restrict ourselves to levels $1 \dots n$ when comparing potential valuations.

Hierarchy refinement may be seen as a hierarchy where the level C_{ij} is more important than C_{kl}, iff $(i < k) \vee ((i = k) \wedge (j < l))$ holds. Let us also note that constraints $c, d \in C_i j$ have the same weight due to totally ordered set of weights W.

Lemma 4.2 For a given hierarchy C, weight function w, and assignments θ and δ the proposition ordered-better$(\theta, \delta, C) \Leftrightarrow$ locally-better$(\theta, \delta, C/w)$ holds.

Proof: ($\Rightarrow$): Let θ and δ be assignments of hierarchy C and let θ be ordered-better assignment than δ. Let k be the first level, where the error function on assignments θ and δ differs, and let $c \in C_k$ be a constraint with maximal weight $w(c)$ such that $e(c\theta) \neq e(c\delta)$ holds. Next let $c \in C_{kl}$ holds for some $l \in 1 \dots n_k$ in hierarchy C/w. We show that θ is locally-better than δ in C/w.

1. $e(d\theta) = e(d\delta)$ holds for every $d \in C_{ij}, i \in 1 \dots (k-1), j \in 1 \dots n_i$ because the same holds for every $d \in C_i, i \in 1 \dots (k-1)$ (error function on θ and δ differs in the level k for the first time).

2. $e(d\theta) = e(d\delta)$ holds for every $d \in C_{kj}, j \in 1 \dots (l-1)$. Firstly $d \in C_k$ and $w(c) <_W w(d)$ is obtained from Definition 4.14 and $j < l$. The constraint c has maximal weight in C_k such that error function on θ and δ differs. This entails $e(d\theta) = e(d\delta)$ for $d \in C_{kj}$.

3. $e(c\theta) < e(c\delta)$ holds because c is the first by level and weight with distinct values of error function on θ and δ, and θ is ordered-better than δ.

4. $e(d\theta) \leq e(d\delta)$ holds for every $d \in C_{kl}$ because θ is ordered-better than δ, $d \in C_k$ and $w(c) =_W w(d)$.

We have shown that proposition $e(d\theta) = e(d\delta)$ holds for $\forall d \in C_{ij}$, $(i < k) \vee ((i = k) \wedge (j < l))$, next $e(c\theta) < e(c\delta)$ and $\forall d \in C_{kl} : e(d\theta) \leq e(d\delta)$ hold. This means that θ is locally-better than δ in C/w.

($\Leftarrow$): This proof is very similar to opposite direction. Let θ be locally-better than δ in C/w. Let error function differ for $c \in C_{kl}$ firstly. So $e(c\theta) < e(c\delta)$ stands. The statement $\forall i < k \, \forall d \in C_i : e(d\theta) = e(d\delta)$ is implied from locally-better comparator definition ($\forall i \, \forall j$ such that $(i < k) \vee ((i = k) \wedge (j < l)) \forall d \in C_{ij} : e(d\theta) = e(d\delta)$). For the same reason, $e(d\theta) = e(d\delta)$ holds for $\forall d \in C_k$ such that $w(c) <_W w(d)$. The proposition $e(d\theta) \leq e(d\delta)$ holds for $\forall d \in C_k : w(d) =_W w(c)$ because $d \in C_{kl}$ holds, error functions on θ and δ differ on C_{kl} firstly and θ is locally-better than δ in C/w. So, all necessary conditions are satisfied and θ is ordered-better than δ in C. □

Theorem 4.3 The assignment θ is ordered-better solution of hierarchy C with weight function w, iff θ is locally-better solution of hierarchy refinement C/w.

Proof: Entailment of Lemma 4.2. □

This theorem allows us to define an *error of CH* (V, D, C) *for ordered-better comparator* (and its satisfaction degree) as an error of CH C/w for locally-better comparator (see Def. 4.7).

4.4 Algorithm for Solving the Hierarchy

This part gives tools which allow to propose an algorithm for solving constraint hierarchy with ordered-better comparator.

Definition 4.15 A sequence $SC = \langle c_1, \ldots, c_m \rangle$ is *hierarchy-ordering* of hierarchy C with m constraints if all constraints of SC are sorted by the level of hierarchy ($c_i \in C_k$, $c_j \in C_l$, $k < l$ implies $i < j$) and by the ordering $\leq_W$ (for $c_i, c_j \in C_k$ such that $w(c_j) <_W w(c_i)$ implies $i < j$). A sequence $\langle c_1, c_2, \ldots, c_i \rangle$ is denoted SC_i for $i \leq m$.

Definition 4.16 Let $SC = \langle c_1, c_2, \ldots, c_m \rangle$ be a hierarchy-ordering of hierarchy C. Recursively defined set $S = S_m$ is called *ordering-solution-set* of hierarchy-ordering SC for

$$\begin{aligned} S_0 &= \{\theta \,|\, \theta \text{ is an assignment of } SC\} \ , \\ S_i &= \{\theta \,|\, \theta \in S_{i-1} \land e(c_i\theta) = \min_{\omega \in S_{i-1}} e(c_i\omega)\} \quad \text{for } i \in 1 \ldots m \ . \end{aligned}$$

Theorem 4.4 Let SC be a hierarchy-ordering of C and a set S be the ordering-solution-set of SC. Then S is the set of ordered-better solutions.

Proof: The proof is by induction on the number of constraints m. The base case is for $m = 1$. The hierarchy is $C = \{c_1\}$ and only one $SC = \langle c_1 \rangle$ exists. We obtain $S = S_1 = \{\theta \,|\, \forall \omega : e(c_1\theta) \leq e(c_1\omega)\}$ and so every assignment $\theta \in S$ is an ordered-better solution.

Suppose that the proposition holds for a hierarchy with m constraints and now describe the case with $m + 1$ constraint. Let us suppose $\theta \in S_{m+1}$ and show for every assignment δ that either θ is ordered-better than δ for SC_{m+1} or δ is not ordered-better than θ for SC_{m+1} (θ and δ are not comparable for SC_{m+1}).

1. $\delta \notin S_{m+1} \land \delta \in S_m$: The error function for every c_i ($i \in 1 \ldots m$) is defined uniquely which follows from the assumption $\delta \in S_m$ and the definition of ordering-solution-set. Inequality $e(c_{m+1}\theta) < e(c_{m+1}\delta)$ is implied from the assumptions $\delta \notin S_{m+1}$ and minimal value for c_{m+1}'s error function. Together both these properties induce that θ is ordered-better than δ.

2. $\delta \in S_{m+1}$: The error function for every constraint is the same again, so no constraint c_i ($i \in 1 \ldots m + 1$) exists such that $e(c\theta) > e(c\delta)$ (or <) and neither δ nor θ is ordered-better than second assignment for SC_{m+1}.

3. $\delta \notin S_m$: $\theta \in S_m$ and so δ can not be ordered-better than θ for SC_m from induction's assumptions. We show that adding of c_{m+1} does not change this situation for SC_{m+1}. The value of error function for some $i \in 1 \ldots m$ differs for θ and δ (from $\delta \notin S_m$). Let i be the first of them. Then $e(c_i\theta) < e(c_i\delta)$ holds. Because c_i is contained in the most important level having the most important weight with such property, assignment δ can not be ordered-better than θ.

Therefore every $\theta \in S$ is an ordered-better solution. □

However, there are ordered-better solutions which can not be obtained via any hierarchy-ordering as its ordering-solution-set. We will demonstrate this proposition by the following example.

Example 4.3 Let us consider constraint hierarchy having only one level $C = C_1 = \{c_1, c_2\} = \{B >= 10, B =< 8\}$ where $w(c_1) = w(c_2)$ holds. An assignment {B = 10} is obtained for hierarchy-ordering $\langle c_1, c_2\rangle$ and {B = 8} for $\langle c_2, c_1\rangle$. Both assignments are ordered-better but for example an assignment {B = 9} is ordered-better, too.

The algorithm for solving constraint hierarchies with ordered-better comparator is based on the Theorem 4.4 and Indigo algorithm [BAFB96a, BAFB96b] for local propagation. The key idea in Indigo is that lower and upper bounds on variables (i.e., intervals) are propagated, and the constraints are processed from strongest to weakest, tightening the bounds on variables using interval arithmetic [Ben95] step by step.

The overall solution for CH with ordered-better comparator is divided into two steps:

1. sorting constraints in every level C_i of hierarchy $\{C_0, C_1, \ldots, C_n\}$ using weights given by w and their ordering $\leq_W$ to an output sequence of constraints OC_i,

2. the application of the Indigo algorithm with sorted input constraints by the sequence $\langle OC_0, OC_1, \ldots, OC_n\rangle$.

Theorem 4.5 Given an acyclic set of constraints, the algorithm computes ordered-metric-better solution.

Proof: Input constraints for the Indigo algorithm define hierarchy-ordering SC using OC_0, OC_1, ..., OC_n. Indigo algorithm minimizes error function in the order given by hierarchy-ordering SC. Those are requirements of Theorem 4.4 and so we obtain an ordered-metric-better solution as a result of the algorithm. □

Let us denote $m = |C|, k = |V|$ and consider the complexity of algorithm. Sorting particular disjoint sets of altogether m constraints takes $O(m \log m)$ steps. The complexity of the last step is $O(mk)$ [BAFB96b]. As a result, we get the total complexity $O(m(k + \log m))$.

4.5 Relationships with Semiring-based CSP

This section introduces the most comparators of CHs as the classes of SCSP. Because the semiring-based constraint satisfaction represents problems defined over finite domains, we will consider CHs over finite domains in the following. The basic semiring structures are given for all existing classes in Table 4.1, their details are discussed in the following part of the section. The remaining comparators (worst-case-better, regionally-better) will be shown being incompatible with properties of c-semiring (monotonicity of $\times$ over $\leq_S$, associativity of $+$, resp.).

Each CH consists from several levels of soft constraints and one level of hard constraints. To model required satisfaction of hard constraints, $\perp$ element representing **0** is introduced and operation $\times$ is extended to handle $\perp$ as an absorbing element for all proposed $\times$ operation. Each hard constraint has its semiring value equal to $\perp$ for all prohibited tuples which ensures its unacceptable violation. The semiring value of remaining acceptable tuples doesn't play any role for all hard constraints, the most valid semiring value for these tuples is maximum element **1** however. Particular levels $C_1, \ldots, C_h$ of soft constraints are modeled with help of h-tuples denoted either by $\vec{a} = (a_1, \ldots, a_h)$ or by $\vec{A} = (A_1, \ldots, A_h)$ wrt. the meaning of particular element of h-tuple.

SCSP Class	comparator	$\mathcal{A}$	$+$	$\times$	$\mathbf{0}$	$\mathbf{1}$
Weighted-SCSP	unsatisfied-c.-b.					
	weighted-sum-b.	$\mathbb{N}^h \cup \{\perp\}$	$\vec{\min}$	$\vec{+}$	$\perp$	$\vec{0}$
	least-squares-b.					
Lexicographic-SCSP	lexicographic-b.	$(\mathbb{N}^{\mathbb{N}})^h \cup \{\perp\}$	$\vec{\min}_{lex}$	$\vec{\sqcup}$	$\perp$	$\vec{\emptyset}$
Local-SCSP	locally-pred.-b.	$(\mathbb{N}^{\mathcal{U}})^h \cup \{\perp\}$	$\vec{\min}_{\diamond}$	$\vec{\sqcup}$	$\perp$	$\vec{\emptyset}$
	locally-metr.-b.	$(\mathbb{N}^{\mathbb{N}\times\mathcal{U}})^h \cup \{\perp\}$	$\vec{\min}_{\diamond lex}$	$\vec{\sqcup}$	$\perp$	$\vec{\emptyset}$
	ordered-b.	$(\mathbb{N}^{\mathbb{N}\times\mathcal{U}})^{h'} \cup \{\perp\}$	$\vec{\min}_{\diamond lex}$	$\vec{\sqcup}$	$\perp$	$\vec{\emptyset}$

Table 4.1: Particular comparators of CHs with specification of $\langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$

4.5.1 Global Comparators

Global comparators will define two basic SCSP classes: *weighted-SCSP* for weighted-sum-better, unsatisfied-count-better, and least-squares-better comparators and *lexicographic-SCSP* for lexicographic-better comparator.

Weighted-sum-better comparator. Let us consider constraint $c@l$ in CH C having weight $w(c)$ defined over variables $(v_1, \ldots, v_k)$. Then c corresponds to semiring constraint $(\mathit{def}, \mathit{con})$ such that $\mathit{con} \equiv (v_1, \ldots, v_k)$ and

$$\forall \theta \equiv [v_1 = d_1, \ldots, v_k = d_k] : \mathit{def}(d_1, \ldots, d_k) = (\underset{1}{0}, \ldots, \underset{l-1}{0}, \underset{l}{w(c) \times e(c\theta)}, \underset{l+1}{0}, \ldots, \underset{h}{0}) \ . \quad (4.6)$$

Operation $\times$ is equivalent to classical vector sum denoted by $\vec{+}$, i.e.,

$$\vec{a} \,\vec{+}\, \vec{b} \equiv (a_1, \ldots, a_h) \vec{+} (b_1, \ldots, b_h) = (a_1 + b_1, \ldots, a_h + b_h) \ .$$

Semiring set corresponds to set of h-tuples extended by $\perp$ element as proposed above, i.e., $\mathcal{A} = \mathbb{N}^h \cup \{\perp\}$ for $e(c\theta), w(c) \in \mathbb{N}$. Additive operation is performed via lexicographic minimum over h-tuples, i.e.,

$$[\vec{\min}(\vec{a}, \vec{b}) = \vec{a}] \equiv [\exists i \, (a_i < b_i) \wedge (\forall j > i : a_i \leq b_i)] \ .$$

Minimum $\mathbf{1}$ corresponds to h-tuple of zeros $\vec{0}$ representing complete satisfaction. Finally, c-semiring corresponds to $S = (\mathbb{N}^h \cup \{\perp\}, \vec{\min}, \vec{+}, \perp, \vec{0})$ for weighted-sum-better comparator.

Unsatisfied-count-better comparator. This comparator is a special version of weighted-sum-predicate-better comparator having all constraint weights equal to 1. Corresponding weighted-SCSP class simplifies definition of the semiring constraint to $\mathit{def}(d_1, \ldots, d_k) = (0, \ldots, 0, e(c\theta), 0, \ldots, 0)$ having value $e(c\theta) \in \{0, 1\}$ on l-th position for $c@l$.

Least-squares-better comparator. The distinction from weighted-sum-better-comparator is given by change of semiring constraint reflecting replacement of error function by its square root, i.e., $\mathit{def}(d_1, \ldots, d_k) = (0, \ldots, 0, w(c) \times e(c\theta)^2, 0, \ldots, 0)$ with non-zero value on l-th position for $c@l$.

Worst-case-better comparator. The first idea leading to the definition of SCSP class for worst-case-better comparator is a replacement of multiplicative operation: for each level of CH, the worst satisfied constraint is searched, i.e.,

$$\vec{\max}(\vec{a}, \vec{b}) \equiv \big(\max(a_1, b_1), \ldots, \max(a_h, b_h)\big) \ .$$

The trouble is that such multiplicative operation is non-monotonous over semiring ordering $\leq_S$[2], i.e., we are in contradiction with basic attributes of c-semiring. Let us remind that $[a \leq_S b] \equiv [\vec{\min}(a, b) = b]$ and show an example demonstrating non-monotonous behavior of $\times$ for this definition of semiring.

Example 4.4 Let $(0, 1)$ and $(1, 0)$ belong to semiring set $\mathcal{A} = \mathbb{N}^2 \cup \{\perp\}$. Then $(0, 1) \geq_S (1, 0)$ holds. After multiplication of both sides by $(1, 0)$ we should obtain $\vec{\max}((0, 1), (1, 0)) \geq_S \vec{\max}((1, 0), (1, 0))$ from monotonicity. However, we compute as a result inverse ordering $(1, 1) <_S (1, 0)$ which gives us a contradiction with monotonicity.

The aim of the following part is to show that each multiplicative operation of c-semiring for worst-case-better comparator has to be non-monotonous over semiring ordering, i.e., no SCSP class computing worst-case-better solution of CH exists.

Let us take a CH $C = C_1 \cup C_2 = \{c_1, c_2\} \cup \{c_3\}$ such that $w(c_1)e(c_1\theta) \leq w(c_2)e(c_2\theta)$ holds for each assignment θ. Corresponding SCSP class may be defined by $(C, con) = (\{(def_i, con_i) \,|\, c_i = (def_i, con_i), i = 1 \ldots 3\}, con_1 \cup con_2 \cup con_3)$. This SCSP class has to fulfill the inequality concerning presumption for c_1 and c_2. Because both constraints belong to the same level of CH and c_2 is always more important than c_1 (from presumption)

$$def_1(t \downarrow^{con}_{con_1}) \leq_S def_2(t \downarrow^{con}_{con_2}) \tag{4.7}$$

holds for all tuples t defined on variables in con. Constraint c_3 is contained in a lower level than c_1, i.e., $\forall t : def_3(t \downarrow^{con}_{con_3}) \leq_S def_1(t \downarrow^{con}_{con_1})$ holds. Now let us combine both sides of this inequality with c_2, i.e., we obtain

$$def_3(t \downarrow^{con}_{con_3}) \times def_2(t \downarrow^{con}_{con_2}) \leq_S def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2}) \tag{4.8}$$

from monotonicity of $\times$ over $\leq_S$. Both constraints c_1 and c_2 belong to the same level of CH and c_2 outweighs c_1 due to Eqn 4.7. As the worst-case-better comparator has to have worst-case behavior on each level, the equality $def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2}) =_S def_2(t \downarrow^{con}_{con_2})$ holds for all tuples t defined on variables in con wrt. Eqn 4.7. Finally we derive

$$def_3(t \downarrow^{con}_{con_3}) \times def_2(t \downarrow^{con}_{con_2}) \leq_S def_2(t \downarrow^{con}_{con_2})$$

from Eqn. 4.8. However, inverse ordering is implied from hierarchical behavior of comparator (c_3 together with c_2 is more important than c_2 alone) which is a contradiction compelling validity of monotonicity condition from Eqn. 4.8.

Lexicographic-better comparator. We have defined this comparator in order to avoid the drowning effect of worst-case-better comparator and to eliminate its contra-intuitive behavior (details in Example 4.2). This change also removes undesirable non-monotonicity and it leads to the existence of a SCSP class.

[2] Operation $\times$ is non-monotonous over ordering $\leq_S$ iff $\exists a, a', b \in S : [a \leq_S a'] \Rightarrow [(a \times b) >_S (a' \times b)]$.

Lexicographic-better comparator differs from the above mentioned comparators by considering multi-sets as the elements A_i of h-tuple $\vec{A} = (A_1, \ldots, A_h)$. The best semiring value corresponds to empty multi-set $\vec{\emptyset}$ (tuple of $\emptyset$'s) and the worst one is $\perp$ element again. Considering a constraint in CH, corresponding semiring constraint is distinguished from semiring constraint of weighted-sum-better comparator (see Eqn. 4.6) by the definition of def function as follows

$$\forall\theta \equiv [v_1 = d_1, \ldots, v_k = d_k] : def(d_1, \ldots, d_k) = (\underset{1}{\emptyset}, \ldots, \underset{l-1}{\emptyset}, \underset{l}{\{w(c) \times e(c\theta)\}}, \underset{l+1}{\emptyset}, \ldots, \underset{h}{\emptyset})$$

having $\{w(c) \times e(c\theta)\}$ as a multi-set. Semiring set is given by h-tuple of multi-sets given by relation $\mathbb{N}$ *to* $\mathbb{N}$, i.e., $(\mathbb{N}^{\mathbb{N}})^h$. Additive operation $\vec{\sqcup}$ is given by a union of multi-sets $\sqcup$ (Eqn. A.1) applied to each element of h-tuple, i.e.,

$$\vec{A} \,\vec{\sqcup}\, \vec{B} = (A_1, \ldots, A_h) \,\vec{\sqcup}\, (B_1, \ldots, B_h) = (A_1 \sqcup B_1, \ldots, A_h \sqcup B_h) \ . \tag{4.9}$$

Multiplicative operation $\vec{\min}_{lex}$ is a lexicographic extension of minimum $\min_{lex}$ operation (defined in Eqn. A.3)

$$\begin{aligned}\vec{A} = \vec{\min}_{lex}(\vec{A}, \vec{B}) &= \vec{\min}_{lex}((A_1, \ldots, A_h), (B_1, \ldots, B_h)) \equiv \\ &[(A_1 \neq B_1) \wedge (A_1 = \min_{lex}(A_1, B_1))] \ \vee \\ &[(A_1 = B_1) \wedge ((A_2, \ldots, A_h) = \vec{\min}_{lex}((A_2, \ldots, A_h), (B_2, \ldots, B_h)))] \ .\end{aligned} \tag{4.10}$$

4.5.2 Local Comparators

Situation for local comparators is a bit complicated as they compare error of each constraint separately and don't subsume them together level by level. This fact is reflected via multi-sets over incomparable elements each corresponding to one constraint. For CH with m constraints we will denote by $\mathcal{U} = \{u_1, \ldots, u_m\}$ the set of elements which are pairwise incomparable. The idea behind inclusion of multi-sets (instead of sets) into the proposal of local-SCSP class lies in preserving of monotonicity of multiplicative operation.

As we include multi-sets into our consideration, maximum element **1** is replaced by a tuple of empty multi-sets denoted by $\vec{\emptyset}$. Meaning and handling of $\perp$ element remains unchanged.

Locally-predicate-better comparator. Each semiring value corresponds to h-tuple $\vec{A} = (A_1, \ldots, A_h)$ having A_i as a multi-set of elements from the set $\mathcal{U}$, i.e., $\mathcal{A} = (\mathbb{N}^{\mathcal{U}})^h \cup \{\perp\}$.

Let us consider CH $C = \{c_1, \ldots, c_m\}$ and its constraint $c_i@l$ over variables $(v_1, \ldots, v_k)$. Then c_i corresponds to semiring constraint (def_i, con_i) such that $con_i \equiv (v_1, \ldots, v_k)$ and

$$\forall\theta \equiv [v_1 = d_1, \ldots, v_k = d_k] : def_i(d_1, \ldots, d_k) = \begin{cases} (\underset{1}{\emptyset}, \ldots, \underset{l-1}{\emptyset}, \underset{l}{\{u_i\}}, \underset{l+1}{\emptyset}, \ldots, \underset{h}{\emptyset}) & \text{if } \theta \models \neg c_i \\ \vec{\emptyset} & \text{if } \theta \models c_i \end{cases}$$

Multiplicative operation corresponds to union of multi-sets extended to tuples (Eqn. 4.9). Additive operation $\vec{\min}_\diamond$ is a lexicographic extension of $\min_\diamond$ operation (as in Eqn 4.10 for $\min_{lex}$).

Locally-metric-better comparator. Compared with predicate version of comparator, we must consider the value of error function and compare it for individual constraints. Elements of multi-set have to be replaced by pair $ru \equiv (r, u) \in \mathbb{N} \times \mathcal{U}$ for $e(c\theta) \in \mathbb{N}$. As a result we obtain semiring set $\mathcal{A} = (\mathbb{N}^{\mathbb{N}\times\mathcal{U}})^h \cup \{\perp\}$. This change is reflected in the definition of semiring constraint for which $def_i(d_1, \dots, d_k) = (\emptyset, \dots, \emptyset, \{e(c_i\theta)u_i\}, \emptyset, \dots, \emptyset)$ holds for $\theta \vDash \neg c_i$. Multiplicative operation again corresponds to union over tuples of multi-sets (Eqn. 4.9).

From these definitions we may derive that the following property

$$[(A_1, \dots, A_h) \in \mathcal{A}] \Rightarrow [\forall i \, \forall ru, r'u \in A_i : r = r']$$

holds for each valid semiring value. This is influenced by the fact that each element $u \in \mathcal{U}$ corresponds to one constraint which is always evaluated by the same value of error function $e(c\theta)$ for given assignment θ. As we need to compare values of error function for individual constraints we apply the following minimum operation for comparison of particular levels

$$[M = \min_{\diamond lex}(M, M')] \equiv [M = \min_{\diamond}(U_M, U_{M'}) \wedge (\forall ru \in M \; \exists r'u \in M' : r \leq r')] \;\; ,$$

where multiset $U_S = \{u \,|\, ru \in S\}$ parametrized by S is such that $\mathrm{card}(u, U_S) = \mathrm{card}(ru, S)$ holds. Operation $\min_{\diamond lex}$ is extended to handle tuples (all levels) as above in Eqn. 4.10.

Let us note that this definition may be directly applied for locally-predicate-better comparator instead of simplified SCSP class from above paragraph.

Ordered-better comparator. SCSP class can be defined for ordered-better comparator with help of hierarchy refinement (see Def. 4.14). For each CH C with weight function w, we construct a new CH by the hierarchy refinement C/w. SCSP class for CH C/w with locally-better comparator corresponds to SCSP class for CH C with ordered-better comparator due to Theorem 4.3.

4.5.3 Regional Comparator

Regionally-better comparator enables comparison of assignments which are incomparable by local comparator. Assignments may be incomparable up to some level and on this level error functions of constraints are compared individually. This incomparability has to be included into additive operation of c-semiring which is used to compare particular semiring values ($a \leq_S b$ iff $a + b = b$). We will show that any additive operation including this kind of incomparability wouldn't be associative[3]. Due to this property we are not able to define CH with regional comparator as a SCSP class.

We will show that any additive operation wouldn't be associative by the following (negative) example. Let us consider constraint hierarchy $C = C_1 \cup C_2 = \{c_1, c_2, c_3\} \cup \{c_4, c_5, c_6\}$. For assignment θ with values of error function $e(c_i\theta)$ for $i = 1 \dots 6$, we will denote its satisfaction degree $\mathbb{E}(C\theta)$ by

$$\begin{pmatrix} e(c_1\theta) & e(c_2\theta) & e(c_3\theta) \\ e(c_4\theta) & e(c_5\theta) & e(c_6\theta) \end{pmatrix}$$

Let us take three possible assignments α, β, γ of CH C with the following values of error functions resulting in the following satisfaction degrees

$$\mathbb{E}(C\alpha) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \longrightarrow a \qquad \mathbb{E}(C\beta) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \longrightarrow b \qquad \mathbb{E}(C\gamma) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \longrightarrow c$$

[3]Operation $+$ is associative iff $(a + b) + c = a + (b + c)$.

each of them corresponding to semiring values $a, b, c \in \mathcal{A}$, resp. Due to the definition of regional comparator, α must have preferred semiring value than β ($a + b = a$) and at the same time γ is better than α ($a + c = c$). As a conclusion we have $(a + b) + c = a + c = c$. Similarly it is derived $a+(b+c) = a+b = a$ because assignment β must have higher semiring value than γ.

$$\left[\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}\right] + \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad [a+b]+c = c$$

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} + \left[\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right] = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \qquad a+[b+c] = a$$

Summarizing it we obtain that associativity is not satisfied for this hierarchy and any c-semiring for regionally-better comparator may not have associative additive operation. Because the associativity of $+$ is one of its basic properties due to the Def. 3.36, no SCSP class for CH with regionally-better comparator exists.

4.6 Categorization of Hierarchies over Finite Domains

We have defined SCSP classes for lexicographic-better comparator (lexicographic-SCSP) and comparators dedicated to weighted-sum-better comparator (weighted-SCSP) and locally-better comparators (local-SCSP), resp. For all defined classes multiplicative operation of semiring is non-idempotent which associates these classes in relation with weighted CSPs. We will show that lexicographic-better and weighted-sum-better comparators are equivalent to weighted CSP wrt. polynomial transformation. CHs with local comparators belong to the different class of problems as their semiring set is partially ordered. Local-SCSP class may be transformed into weighted CSP via polynomial time refinement. However, opposite refinement doesn't exist due to partial ordering over set of CH's assignments. Summarization of all relationships between basic SCSP classes wrt. their properties (idempotency versus non-idempotency of multiplicative semiring operation, total versus partial semiring ordering) is shown in Fig. 4.1.

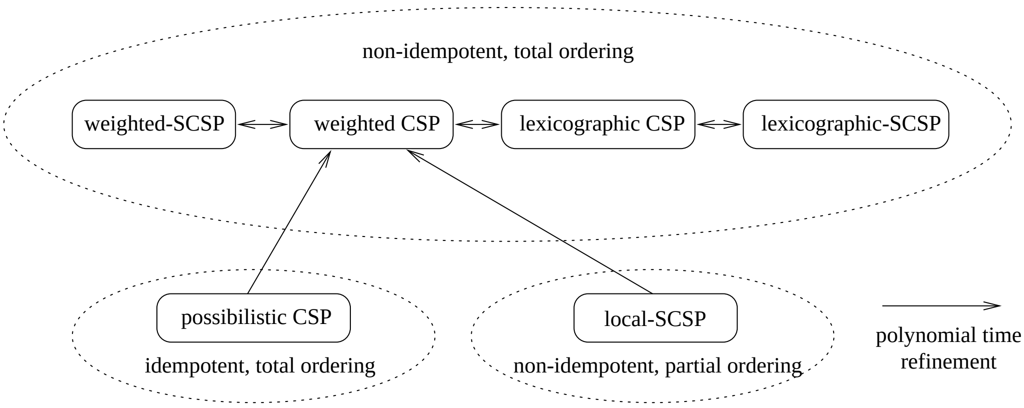


Figure 4.1: Relationships between frameworks

Weighted-SCSP and lexicographic-SCSP classes may be defined as valued CSP classes with help of direct relation between SCSP and VSCP described in Sect. 3.8.2. The only one difference between VCSP and SCSP consists in required total ordering for valuations of VCSP. Because this property is not satisfied for local-SCSP class, we may not define VCSP class for CH with any local comparator.

Weighted-SCSP

Weighted CSP is nothing but the first non-required level of CH with weighted-sum-better comparator. A simple polynomial time refinement exists from CH with weighted-sum-better comparator into the weighted CSP. Let us consider the highest value p among $p_1, \dots, p_h$ where each p_i is the highest value on i-th position of all semiring values $(a_1, \dots, a_h)$ in definition of any semiring constraint. Each semiring value $(a_1, \dots, a_h)$ of any tuple that appears in CH with weighted-sum-better comparator is transformed into $a_1 p^{(h-1)} + \cdots + a_{h-1} p + a_h$. Semiring value $\perp$ is transformed into $+\infty$.

The transformations for the unsatisfied-count-better and least-squares-better are defined in the same way as for the above mentioned weighted-sum-better comparator.

Lexicographic-SCSP

Lexicographic CSP corresponds to the first non-required level of CH with lexicographic-better comparator. For the second part of equivalence, let us consider the highest possible value p from multi-set $\bigsqcup_j A_j$ of all semiring values $(A_1, \dots, A_h)$ occurring in any definition of semiring constraint. Then the desired transformation f of every semiring element $\vec{A} = (A_1, \dots, A_h)$ is given by

$$f(\vec{A}) = (A'_1 \sqcup \cdots \sqcup A'_h) \text{ such that } \forall i\ A'_i = \{a' \mid (a' = ap^{h-i}) \wedge (a \in A_i)\}$$

Semiring value $\perp$ of lexicographic-SCSP corresponds with its counterpart in lexicographic CSP $\perp$.

Local-SCSP

Local-SCSP class may be transformed into the problem which is strongly equivalent with weighted CSP via polynomial time refinement. Let $\{u_1, \dots, u_m\}$ be a set of incomparable elements $\mathcal{U}$ in local-SCSP with h levels. Let n_i be the maximal number of occurence of pair ru_i for any r in multi-set $\bigsqcup_j A_j$ in any semiring value $\vec{A} = (A_1, \dots, A_h)$ of local-SCSP. Let e define the highest value among all numbers r of any pair ru_i from multi-set $\bigsqcup_j A_j$ in any semiring value $\vec{A} = (A_1, \dots, A_h)$ occurring in definition of any semiring constraint in local-SCSP. Then each semiring value $\vec{A} = (A_1, \dots, A_h)$ may be transformed into $a_1 p^{(h-1)} + \cdots + a_{h-1} p + a_h$ where $p = \sum_{i=1}^{m} n_i e$ and $a_i = \sum_{ru \in A_i} [r \times \mathrm{card}(ru, A_i)]$. In this case, an optimal tuple of local-SCSP not only minimizes error function of individual constraint at each level, but also, the weighted sum of violated constraints.

The opposite transformation doesn't exist as an entailment of Corollary 3.7: local-SCSP may not be defined as a refinement of lexicographic-SCSP because incomparable semiring elements in local-SCSP must be incomparable in SCSP class for weighted CSP which semiring set is totally ordered.

Chapter 5

Variables' Annotations

Over-constrained problems are usually solved by giving some level of preference to individual constraints [FW92, BFBW92, SFV95] or to each combination of values for every constraint [DFP96, BMR97b] and defining the solution using these preferences. Still not investigated possibility offers assigning preferences to variables, which seems to be interesting for over-constrained problems or optimization problems with partially or even completely ordered variables. Constraints with variables' annotations is a constraint solving environment where preferences (or annotations) are assigned to individual variables instead of to the constraints themselves [Rud98c, Rud98a, Rud99, RM99b]. Moreover, the annotations are local to variable occurrences, i.e., any variable may have different annotations in different constraints (in fact, even different occurrences in the same constraint are allowed).

Different levels of importance for particular variables may become interesting in areas like temporal scheduling where time of events are classified, e.g., by participant's interest, in job scheduling to influence an order of job via job owner's priority, or in placement problems where a better placement of an object depends on its properties.

The most prominent example is the timetabling problem. The problem is stated in variables which represent teachers (dean, professors, assistants...), rooms (more and less occupied) and different groups of students. All these variables have their own preferences, which could be applied directly helping to find appropriate solution.

Let us consider job-shop scheduling problem [Bru98]. Such problem is represented by tasks to be completed and resources to be used by particular tasks. There annotations may represent preferences over particular tasks or resources derived from their importance within the problem.

Other example is the problem of the search for the optimal sequence of aircraft departures from a runway. Because practically all the constraints are safety regulations, we can only change aircraft allocation to different time slots. In an over-constrained situation the only allowed action is the removal of an aircraft from further consideration. We can assign preferences to individual variables (planes) and define such a comparator which simply abandons the aircraft with the smallest preferences.

Example 5.1 The small timetabling example illustrates a possible interpretation of variables' annotations in constraints over natural numbers. Also, we show by this example that assigning preferences to variables could be more natural than defining preferences for constraints. There is a lecture `L` and its associated practice `P`. The practice should be preferably taught at least one day after the lecture. We would like to express by the following constraint that the scheduling of lecture (taught by a professor) is more important than scheduling (time placement) of the practice (taught by an assistant).

```
L@strong + 1 #=< P@medium  % c1
```

There are two weaker constraints: the lecture has to be taught on Thursday or Friday and the practice anytime between Monday and Thursday:

```
L@weak in 4..5             % c2
P@weak in 1..4             % c3
```

These constraints form a kind of hierarchy: the constraint c_1 with the highest preferences must be satisfied first and then we may try to satisfy constraints c_2 and c_3. It is possible to satisfy c_1 but not c_2 and c_3 taken together. The constraint c_2 influences the variable with higher annotations (look at c_1), so this constraint is also satisfied. Then, trying to minimize the overall constraint violation, we get (a kind of) optimal solution `L=4, P=5`. By classical hierarchy (see Chapter 4) where c_1 is annotated by `strong` or `medium`, the solution `L=3, P=4` is also obtained. But this solution is not optimal from our point of view. The different requirements towards the lecture and practice must be stated by assigning different preferences to c_2 and c_3 and so these constraints must be ordered. But this could be wrong with respect to other constraints in a more complex problem. Also, the exact location of the two appropriate constraints need not be easy to find in this context. Assuming we are not satisfied with the solution found and would like to reformulate constraints in such a way as to reflect more precisely our expectations, it is easier to change values of variables' annotations than to find constraints whose preference changes will lead to better solution.

5.1 Annotation Triple

Definition 5.1 *Combining function* $\bigoplus$ on a set A is a function defined on each finite tuple of values from A such that the value of combining function on tuple $(a_1, \ldots, a_n)$ is independent on permutation of values in this tuple, i.e.,

$$\begin{aligned}&\forall i_1, \ldots, i_n \in \{1, \ldots, n\} \text{ such that}\\ &\quad \forall k, l \in \{1, \ldots, n\} : i_k \neq i_l \text{ for } k \neq l\\ &\qquad \Rightarrow \bigoplus(a_1, \ldots, a_n) = \bigoplus(a_{i_1}, \ldots, a_{i_n}) \ .\end{aligned}$$

Because of allowed permutation of values in tuple $(a_1, \ldots, a_n)$ we may denote an expression $\bigoplus(a_{i_1}, \ldots, a_{i_n})$ also by $\bigoplus\limits_{i=1}^{n} a_i$.

Let us note that combining function may not be defined only on a subset of set A instead of on a tuple $(a_1, \ldots, a_n)$ of values from the set A because we don't require pairwise different values a_i.

We should also remark that $\bigoplus$ may be easily defined as an extension of a commutative and associative closed binary operation $+$ on S. Examples of combining functions are arithmetic average or n-ary sum $\sum$ which is an extension of binary operation $+$.

Definition 5.2 (structure) *Annotation triple* is defined by $(\mathcal{A}, \preceq, \bigoplus)$, where

- $\mathcal{A}$ is a set denoted as *annotation set* with elements $a \in \mathcal{A}$ called *annotations*;

- $\preceq$ is an ordering on $\mathcal{A}$ denoted as *annotation ordering*;
- $\bigoplus$ is a combining function on $\mathcal{A}$.

For elements $a, b \in \mathcal{A}$, we say that annotation a is *more preferred* annotation than b if proposition $a \preceq b$ holds. The most preferred annotation will be denoted **0**, the least preferred **1**. For $a_i \in \mathcal{A}$, we say $\bigoplus$ *combines* annotations a_i.

Definition 5.3 (problem) *Constraint system with (variables') annotations* $P_{\mathcal{A}}$ is composed from $(V, D, C, (\mathcal{A}, \preceq, \bigoplus), a)$, where

- $V = \{v_1, \ldots, v_n\}$ is the set of variables;
- $D = \{D_1, \ldots, D_n\}$ is the set of domains. Each domain is a finite set containing the possible values for the corresponding variable;
- $C = \{c_1, \ldots, c_n\}$ is the set of constraints. A constraint c_i is a relation defined on a subset $\{v_{i_1}, \ldots, v_{i_{k_i}}\}$ of all the variables, that is $\{D_{i_1} \times \cdots \times D_{i_{k_i}}\} \supseteq c_i$;
- $(\mathcal{A}, \preceq, \bigoplus)$ is the annotation triple;
- $a : C \times V \to \mathcal{A}$ is a function which determines the annotation of variable in constraint, we call such annotation *variable annotation*, i.e., annotation of variable is *local* to a constraint.

Set of variables in constraint c will be referenced by V_c as usually.

Definition 5.4 *Assignment* is a function θ defined from set of variables to the domains of variables $\theta : V \to D$. Θ_V denotes set of all possible assignments of variables in V.

Local variable's annotations are used for computing several characteristics of underlying constraint system, the so called *global annotations*.

Definition 5.5 *Global variable annotation* combines variable's annotations in all constraints where this variable occurs.

$$av\colon V \to \mathcal{A} \;\;,\;\; av(v) = \bigoplus_{(c \in C) \wedge (v \in V_c)} a(c, v)$$

Definition 5.6 *Constraint annotation* combines variable's annotations of all variables in constraint.

$$ac\colon C \to \mathcal{A} \;\;,\;\; ac(c) = \bigoplus_{v \in V_c} a(c, v)$$

Definition 5.7 *Global constraint annotation* combines global variable annotations of all variables in constraint.

$$acv\colon C \to \mathcal{A} \;\;,\;\; acv(c) = \bigoplus_{v \in V_c} av(v)$$

5.1.1 Instances of Annotation Triple

Let us consider representatives of annotation triple $(\mathcal{A}, \preceq, \bigoplus)$. First two instances are concentrated on weighted approach to annotations.

- $\mathcal{A}_{sum} = (\mathbb{N}, \geq_{\mathbb{N}}, +)$
- $\mathcal{A}_{vect} = (\mathbb{N}^k, \geq_{lex}, +_{vect})$
  - fixed $k \in \mathbb{N}$
  - $\geq_{lex}$ is standard lexicographic ordering
  - $+_{vect}$ is standard vector sum
  - $a(c, v) = (0, \ldots, 0, w, 0, \ldots, 0)$ where $w \in \mathbb{N}$

While $\mathcal{A}_{sum}$ allows only one level of weighted annotations, $\mathcal{A}_{vect}$ as its generalization considers k weighted levels of preferences.

$\mathcal{A}_{avg}$ and $\mathcal{A}_{gavg}$ select the variable with the highest average annotation.

- $\mathcal{A}_{avg} = (\langle 0, 1\rangle \subseteq \mathbb{R}, \geq_{\mathbb{R}}$, arithmetic average$)$
- $\mathcal{A}_{gavg} = (\langle 0, 1\rangle \subseteq \mathbb{R}, \geq_{\mathbb{R}}$, geometric average$)$

Choice of another "non-average" operation is suitable for combination of annotations when the dependency on the number of variables wouldn't be overestimated.

A selection of arbitrary closed interval in $\mathcal{A}_{avg}$ and $\mathcal{A}_{gavg}$ does not change ordering of computed global annotations. Also the selected $\geq$ ordering is dual to the ordering $\leq$. However, arithmetic and geometric average are not interchangeable with respect to the proposition

$$(\frac{a+b}{2} \leq \frac{c+d}{2}) \not\Leftrightarrow (\sqrt{ab} \leq \sqrt{cd}) \ .$$

Example 5.2 Let us consider $a = b = 0.5$ and $c = 0.2$ with $d = 0.9$. Then following inequalities hold

$$(0.5 + 0.5)/2 \leq (0.2 + 0.9)/2 \ , \ \sqrt{0.2 \times 0.9} \leq \sqrt{0.5 \times 0.5} \ .$$

5.1.2 Solution

Annotations will be applied for computing solution in over-constrained and optimization problems. Different methods for selection of solution from the set of all assignments will reflect possible semantics of variables' annotations. Annotations will define variable ordering in constraint satisfaction problems with preferences and in optimization problems. Mappings of annotations to the fuzzy CSP (see Sect. 3.4) and to the constraint hierarchies (in Chapter 4) are aimed to solve over-constrained problems based on specific classical frameworks.

5.2 Fuzzy Annotations

Fuzzy annotations express importance of variables via fuzzy membership degree [DP93, MG81]. Annotations of variables are combined to compute global constraint annotations defining preferences for selection a solution via min-max optimization. This basic view on annotations allows construction of corresponding fuzzy CSP or in a simpler case a possibilistic CSP which qualifies a complexity of possible solution methods (see Sect. 3.8.3).

Definition 5.8 (class) *Constraint system with fuzzy annotations* $P_{\mathcal{A}\pi}$ is a constraint system with annotations defined by tuple $(V, D, C, \mathcal{A}_{\mathbb{R}}, a)$ having $\mathcal{A}_{\mathbb{R}} = (\langle 0, 1\rangle \subseteq \mathbb{R}, \geq_{\mathbb{R}}, \bigoplus_{\mathbb{R}})$ where $\bigoplus_{\mathbb{R}}$ is a combining function defined over real numbers.

Selection of interval, its ordering, and decision for combining function over real numbers will be discussed in the next part which is intended to the define a solution of $P_{\mathcal{A}\pi}$.

An *error function* indicates how the given constraint is satisfied by some assignment $\theta \in \Theta_V$. Its definition together with the notion of *metric* and *trivial* error function may be taken from Def. 4.4 and the accompanied description.

Definition 5.9 (satisfaction degree) *Error of constraint set* C wrt. assignment $\theta \in \Theta_V$ is a function defined by

$$E(C\theta) = \max_{c \in C} acv(c)e(c\theta)$$

where acv is a global constraint annotation and e is an error function such that $e(c\alpha) \leq 1$ holds for all $c \in C$ and $\alpha \in \Theta_V$.

Definition 5.10 (solution) *Solution of constraint system with fuzzy annotations* $P_{\mathcal{A}\pi} = (V, D, C, \mathcal{A}_{\mathbb{R}}, a)$ is each assignment $\theta \in \Theta_V$ with minimal error of constraint set C, i.e.,

$$E(C\theta) = \min_{\delta \in \Theta_V} E(C\delta) \ .$$

Now we are able to discuss selection of annotation triple. Ordering $\geq_{\mathbb{R}}$ was chosen due to the usual interpretation of error function $e(c\theta)$ which is applied for computing solution together with annotations: the more important annotation is, the more an error of constraint is multiplied. Interval $\langle 0, 1\rangle$ was selected due to the fact that membership degree is expressed with help of unit interval usually. Wrt. min-max behavior of the problem solution, we will prefer combining function as an "average" operation rather than over-estimation of the number of variables because sufficient number of even very non-important annotations could be able outweigh even very important annotation.

Example 5.3 Let us consider that the Example 5.1 is solved with trivial error function and arithmetic average as combining function.

```
L@0.9 + 1 #=< P@0.6    % c1
L@0.3 in 4..5          % c2
P@0.3 in 1..4          % c3
```

Global variable annotations necessary for computing global constraint annotations are

$$av(\mathtt{L}) = \frac{0.9 + 0.3}{2} = 0.6 \ , \quad av(\mathtt{P}) = \frac{0.6 + 0.3}{2} = 0.45 \ .$$

Global constraint annotations represent the weight of constraint

$$acv(\texttt{c1}) = \frac{av(\texttt{L}) + av(\texttt{P})}{2} = 0.525, \quad acv(\texttt{c2}) = av(\texttt{L}) = 0.6, \quad acv(\texttt{c3}) = av(\texttt{P}) = 0.45 \ .$$

As a result we obtain solution $\theta = \{(\texttt{L}, 4), (\texttt{P}, 5)\}$ with unsatisfied constraint c3 and error

$$\begin{aligned} E(C\theta) &= \min(acv(\texttt{c1}) \times e(\texttt{c1}\theta),\, acv(\texttt{c2}) \times e(\texttt{c2}\theta),\, acv(\texttt{c3}) \times e(\texttt{c3}\theta)) = \\ &= \max(0.525 \times 0,\, 0.6 \times 0,\, 0.45 \times 1) = 0.45 \ . \end{aligned}$$

Preferences for the trivial error function $e(c\theta)$ are expressed only by the global constraint annotation. When the metric error function is used the situation is very different. Then, the preferences of constraints are changed with respect to the chosen assignment θ and a sufficiently large value of error function can change the solution drastically. This combination of the metric error function and global constraint annotation can be used only when the metric error function is normalized as required in Def. 5.10.

Example 5.3 (continuation) If a metric error function would be applied to this problem, a normalization of domain $D = \{1, \dots, \|days\|\}$ has to be done to obtain metric space with suitable distance function, i.e.,

$$\forall d_1, d_2 \in D \; e(d_1 = d_2) = \text{abs}(d_1 - d_2)/\|days\| \ .$$

In this example, a solution with such metric comparator is the same as above.

Even if the above example returns the same solution for both metric and trivial comparators, an important difference may be seen between these approaches. The metric error function should be used when the decrease of large values of the error function is desirable. The trivial error function is advantageous when the meaning of annotations does not rely on the shape of the error function so the size of error (any different from zero) is not important.

Example 5.4 Let us consider the example with constraints $C = \{c_1, c_2\}$ and two assignments θ_0, θ_1, and describe the selection of a better assignment with a metric comparator. We suppose global constraint annotations:

$$acv(c_1) = 0.9,\; acv(c_2) = 0.1$$

and value for error function (in this case, the normalization is done by division by maximal expectable value of the error function):

$$\begin{aligned} e(c_1\theta_0) &= 0/100 \ , & e(c_2\theta_0) &= 10/100 \ , \\ e(c_1\theta_1) &= 1/100 \ , & e(c_2\theta_1) &= 9/100 \ . \end{aligned}$$

The errors $E(C\theta_i)$ for both assignments are

$$\begin{aligned} E(C\theta_0) &= \max\{acv(c_1) \times e(c_1\theta_0),\, acv(c_2) \times e(c_2\theta_0)\} = \\ &\ \max\{0.9 \times 0,\, 0.1 \times 0.1\} = 0.01 \ , \\ E(C\theta_1) &= \max\{acv(c_1) \times e(c_1\theta_1),\, acv(c_2) \times e(c_2\theta_1)\} = \\ &\ \max\{0.9 \times 0.01,\, 0.1 \times 0.09\} = 0.009 \end{aligned}$$

and the better assignment is θ_1 with minimal error 0.009. This assignment violates the constraint c_1 with the highest global constraint's annotation but the large value of $e(c_2\theta_0)$ causes this selection. But the trivial comparator selects as a solution the assignment θ_0, because the assignment θ_0 does not violate the strong constraint c_1 and the combination with a value of the error function is trivial.

5.2.1 Mappings

The following theorems relate constraint system with fuzzy annotations with possibilistic CSP (see Sect. 3.3) and fuzzy CSP (described in Sect. 3.4).

Theorem 5.1 (possibilistic CSP) Let us consider a constraint system with fuzzy annotations $P_{\mathcal{A}\pi} = (V, D, C, \mathcal{A}_{\mathbb{R}}, a)$. We define possibilistic CSP P_π as a triple $(V, D, C_\mathcal{A})$ where $C_\mathcal{A} = \{(c, acv(c)) \mid c \in C\}$ is a set of necessity-valued constraints having preference degrees equal to global constraint annotation. Then $\theta \in \Theta_V$ is an optimal assignment of P_π iff θ is a solution of $P_{\mathcal{A}\pi}$ with trivial error function and $E(C\theta)$ corresponds to inconsistency degree of P_π.

Proof: Set S of optimal assignments in possibilistic CSP P_π corresponds to assignments selected by

$$\min\nolimits_{\theta\in\Theta_V} \max\nolimits_{((c_i,acv(c_i))\in C_\mathcal{A})\wedge(\theta\models\neg c_i)} (acv(c_i), 0) \enspace .$$

Because the value of trivial error function $e(c\theta)$ is equal 0 for all satisfied constraints and 1 for any violated constraint, we may rewrite it into

$$\min\nolimits_{\theta\in\Theta_V} \max\nolimits_{(c_i,acv(c_i))\in C_\mathcal{A}} acv(c_i)e(c_i\theta)$$

which defines exactly the set of all solutions in $P_{\mathcal{A}\pi}$. At the same time we have computed that error of any solution in $P_{\mathcal{A}\pi}$ is equal to the inconsistency degree of P_π. □

Possibilistic constraint satisfaction was applied to describe behavior of fuzzy annotation with trivial error function. The metric error function associates a preference to each tuple of values for constraint which may be easily modeled with help of fuzzy constraint satisfaction.

Theorem 5.2 (fuzzy CSP) Let $P_{\mathcal{A}\pi} = (V, D, C, \mathcal{A}_{\mathbb{R}}, a)$ be a constraint system with fuzzy annotations having $C = \{c_1, \ldots, c_m\}$. We will define fuzzy CSP P_μ as a triple $(V, D, C_\mathcal{A})$ where $C_\mathcal{A} = \{c_{\mathcal{A}1}, \ldots, c_{\mathcal{A}m}\}$ is a set of fuzzy constraints. Each fuzzy constraint $c_{\mathcal{A}i}$ is defined over the same set of variables as c_i, i.e., $\{v_{i_1}, \ldots, v_{i_k}\}$. Its fuzzy relation is given by $\mu_{c_{\mathcal{A}i}}(d_{i_1}, \ldots, d_{i_k}) = 1 - e(c_i(d_{i_1}, \ldots, d_{i_k})) \times acv(c_i)$ where all d_{i_j} are possible values from domain of corresponding variable v_{i_j}. Then $\theta \in \Theta_V$ is a solution of P_μ computed by conjunctive combination iff θ is a solution of $P_{\mathcal{A}\pi}$ and $E(C\theta)$ corresponds to inconsistency degree of P_μ.

Proof: Set of solutions in P_μ corresponds to tuples selected by

$$\max_{(d_1,\ldots,d_n)\in D^n} \min_{c_{\mathcal{A}i}\in C_\mathcal{A}} \mu_{c_i}(d_1, \ldots, d_n) \downarrow^V_{(v_{i_1},\ldots,v_{i_k})} = \max_{\theta\in\Theta_V} \min_{c_i\in C} \big(1 - e(c_i\theta) \times acv(c_i)\big) =$$

$$= \max_{\theta\in\Theta_V} \Big(1 - \max_{c_i\in C}\big(e(c_i\theta)) \times acv(c_i)\big)\Big) = 1 - \min_{\theta\in\Theta_V} \max_{c_i\in C}\big(e(c_i\theta) \times acv(c_i)\big) \enspace .$$

Set of assignments selected by $\min_{\theta\in\Theta_V} \max_{c_i\in C_\mathcal{A}} \big(e(c_i\theta) \times acv(c_i)\big)$ corresponds to the set of solution in $P_{\mathcal{A}\pi}$. We also see that consistency degree of P_μ is the complement to $E(C\theta)$, i.e., desired relation for inconsistency degree was also derived. □

5.3 Hierarchical Annotations

An opposite view of annotations compared with fuzzy approach demonstrates their interpretation through constraint hierarchies (for details about constraint hierarchies see Chapter 4). Basically such hierarchy will be constructed over constraint annotations, with additional order imposed by global constraint annotations within each level. Because hierarchical annotations applies variables' annotations in a more extended way than fuzzy annotations they are able to catch semantics of annotations more extensively. However, we need to take into account a more complete solution strategy than for fuzzy annotations wrt. shown correspondences of particular annotations with fuzzy CSP and later with constraint hierarchies (see Sect. 3.8.3 and 4.6).

Definition 5.11 (class) *Constraint system with hierarchical annotations* $P_{\mathcal{A}H}$ is a constraint system with annotations defined by tuple $(V, D, C, \mathcal{A}_{\mathbb{R}}, a)$ having $\mathcal{A}_{\mathbb{R}} = (\langle 0, 1\rangle \subseteq \mathbb{R}, \geq_{\mathbb{R}}, \bigoplus_{\mathbb{R}})$ where $\bigoplus_{\mathbb{R}}$ is a combining function defined over real numbers.

The reasons for such selection of annotations are basically the same as those discussed for fuzzy annotations in Sect. 5.2. Combining function $\bigoplus_{\mathbb{R}}$ should be again replaced by "average" operation which influence selection of annotation set towards an interval of real numbers. Due to the generality of unit interval as a typical representative of closed continuous domain, we have restricted ourself to this selection. Annotation set could be also defined by any other closed interval $\langle 0, r\rangle, r \in \mathbb{R}$ of real numbers having r as the most important annotation. The least important annotation has to correspond to 0 due to later combination of the annotations with error function as above for fuzzy annotations.

Definition 5.12 (ACH) Let us consider $P_{\mathcal{A}H} = (V, D, C, \mathcal{A}_{\mathbb{R}}, a)$ and split the set of constraints C into a union of disjoint sets C_i such that constraints having more important constraint annotations ac are contained in a set with the smaller index i, i.e.,

$$c, d \in C : ac(c) \geq_{\mathbb{R}} ac(d) \equiv (c \in C_i) \wedge (d \in C_j) \wedge (i \leq j) \ .$$

Annotated constraint hierarchy (ACH) is such division of the constraint set C that $C = C_0 \cup C_1 \cup \ldots \cup C_n$ holds and sets C_j are empty for all $j > n$, all sets C_k for $k \in 1 \ldots n$ are non-empty, and C_0 denotes the set $\{c \in C \mid ac(c) =_{\mathbb{R}} 1\}$. Particular sets C_i will be called *levels of constraints* in ACH.

From the definition of ACH follows that levels with smaller indices contain constraints with higher and more important constraint annotations. Such level with smaller index will be called *more important* than another level having a higher index.

In order to evaluate each assignment by its satisfaction degree, we will check satisfaction of each constraint with help of *error functions (trivial or metric)* applied for fuzzy annotations in Sect. 5.2 and defined by Def. 4.4. Within this framework, error of each constraint combined together with global constraints annotation was a source for computing error of overall constraint set giving us the satisfaction degree of each assignment. Here we will define an error of constraint set as a general function to be further applied on every level of constraints in ACH separately.

Definition 5.13 Let us consider $P_{\mathcal{A}H} = (V, D, C, \mathcal{A}_{\mathbb{R}}, a)$. An *error of constraint set* $C' \subseteq C$ wrt. assignment $\theta \in \Theta_V$ is defined by function $E : 2^C \times \Theta_V \rightarrow W$ where W is ordered by partial ordering $\leq_W$. For each C', the set W contains a minimal element $\mathbf{0}_{C'}$.

The smaller elements of W correspond to the less important errors. Minimal element $\mathbf{0}_{C'}$ will express that all constraints in C' are satisfied. Usually the one common minimal element exists for all sets C'. However, different minimal elements may exist (e.g., $\mathbf{0}_{C'}$ depends on the number of constraints in C'). An example of W could be a set of real positive numbers $\mathbb{R}_+$ with minimal element corresponding to 0.

Definition 5.14 (satisfaction degree, solution) Having constraint system with hierarchical annotations $P_{\mathcal{A}H} = (V, D, C, \mathcal{A}_{\mathbb{R}}, a)$ and corresponding ACH $C = C_0 \cup C_1 \cup \ldots \cup C_n$, an *error of ACH* wrt. assignment θ is a tuple $[E(C_1\theta), \ldots, E(C_n\theta)]$. This tuple will be denoted by $\mathbb{E}(C\theta)$.

A *solution of constraint system with hierarchical annotations $P_{\mathcal{A}H}$* is such assignment θ that $E(C_0\theta) = \mathbf{0}_{C_0}$ holds and the error $\mathbb{E}(C\theta)$ is minimal wrt. lexicographic ordering, i.e.,

$$\forall \delta \in \Theta_V \; \exists k \text{ such that } \forall i < k : [E(C_i\theta) \leq_W E(C_i\delta)] \wedge [E(C_k\theta) <_W E(C_k\delta)]$$

Requirement on the best possible satisfaction of level C_0 is intended to express requirement of complete satisfaction of constraints in C_0 ($\forall c \in C_0 : \theta \models c$) . Having such definition of solution, we can see that error of assignments are compared by error of their levels from the most to the less important level. Smaller error on any more important level is able to determine selection of better assignment independently on the value of error on less important levels.

We have defined error of constraint set as a general function $E : 2^C \times \Theta_V \rightarrow W$. Let us introduce possible instances of this error for $W = \mathbb{R}_+$ with standard ordering over real numbers having minimum element equal to 0

$$\textit{weighted-sum}\text{ error: } E(C'\theta) = \sum_{c \in C'} acv(c)\, e(c\theta) \tag{5.1}$$

$$\textit{worst-case}\text{ error: } E(C'\theta) = \max_{c \in C'} acv(c)\, e(c\theta) \tag{5.2}$$

$$\textit{least-squares}\text{ error: } E(C'\theta) = \sum_{c \in C'} acv(c)\, e(c\theta)^2 \tag{5.3}$$

where $C' \subseteq C$ and $\theta \in \Theta_V$ hold.

Example 5.5 Let us take constraint system with hierarchical annotations and arithmetic average as a combining function given by constraints from Example 5.3. We will show how to define its annotated constraint hierarchy and find its solution via weighted-sum error and trivial error function. Constraint annotations are

$$ac(c_1) = \tfrac{a(c_1,\mathtt{L}) + a(c_1,\mathtt{P})}{2} = 0.75 \ ,$$
$$ac(c_2) = a(c_2, \mathtt{L}) = 0.3 \ , \quad ac(c_3) = a(c_3, \mathtt{P}) = 0.3 \ .$$

As a consequence we obtain an ACH $C = C_1 \cup C_2 = \{c_1\} \cup \{c_2, c_3\}$. Global variable annotations and global constraint annotations have the same values as in Example 5.3. Constraint c_1 from the first level can be satisfied but c_2 and c_3 taken together can not. With respect to the smaller weight of c_3, we get solution θ=$\{[\mathtt{L}, 4], [\mathtt{P}, 5]\}$ with error

$$\begin{aligned} E(C\theta) &= [0, (acv(c_2) \times e(c_2\,\theta) + acv(c_3) \times e(c_3\,\theta))] = \\ & [0, (0.6 \times 0 + 0.45 \times 1)] = [0, 0.45] \ . \end{aligned}$$

Let us consider a change in annotation of variable P in c_3 to 0.6. Three levels of hierarchy $C = [\{c_1\}, \{c_3\}, \{c_2\}]$ arise as a consequence of $ac(c_3)$ increase. The values $av(\texttt{P}) = 0.6$ and $acv(c_3) = 0.6$ increase, too. Final solution $\theta = \{[\texttt{L}, 3], [\texttt{P}, 4]\}$ has error $E(C\theta) \doteq [0, 0, 0.6]$. This value of error reflects that we violate a more important constraint than in the example above.

Other possibility can be introduced by comparison of *individual* errors for particular constraints again with help of global constraint annotations acv. The so called *local* error will be defined by tuple of pairs $(acv(c), e(c\theta))$

$$E(C'\theta) = [(acv(c_1), e(c_1\theta)), \dots, (acv(c_k), e(c_k\theta))] \quad \text{for } C' = \{c_1, \dots, c_k\} \subseteq C \tag{5.4}$$

where constraints in C' are taken in any fixed order. Now let us define ordering $<_l$ over local errors of constraint sets which has to be proposed to define solution of $P_{\mathcal{A}H}$ via local errors (see Def. 5.14).

$$\begin{aligned} [(a_1, e_1), \dots, (a_k, e_k)] <_l [(a'_1, e'_1), \dots, (a'_k, e'_k)] \equiv \\ (\exists i : e_i < e'_i) \wedge (\forall j : a_j \geq_{\mathbb{R}} a'_j \Rightarrow e_j \leq e'_j) \end{aligned} \tag{5.5}$$

For constraint set $C' = \{c_1, \dots, c_k\}$ with global constraint annotations $acv(c_1), \dots, acv(c_k)$, a minimal element of ordering $<_l$ correspond to such tuple that second element of each pair $(acv(c_i), e(c_i\theta))$ is equal to 0, i.e., all constraints in C' are satisfied by assignment θ. Values of global constraint annotations are unique in $P_{\mathcal{A}H}$, i.e., only one minimal element exists for each constraint set C'.

5.3.1 Mappings

In a similar way we have related fuzzy annotations with fuzzy CSP we would like to define correspondence between hierarchical annotations and constraint hierarchies (see Chapter 4).

Theorem 5.3 (constraint hierarchy) Let $P_{\mathcal{A}H} = (V, D, C, \mathcal{A}_{\mathbb{R}}, a)$ be a constraint system with hierarchical annotations and $\bigcup_{i=1}^{n} C$ its ACH. We define CH P_{CH} as a triple $(V, D, \bigcup_{i=1}^{n} C)$. Then $\theta \in \Theta_V$ is a solution of $P_{\mathcal{A}H}$ with weighted-sum error (or one of the following errors: worst-case, least-squares, local) iff θ is a solution of P_{CH} with weighted-sum-better (or worst-case-better, least-squares-better, ordered-better, resp.) comparator where $acv(c)$ is taken as a weight for any constraint c.

Proof: First let us consider correspondence with global comparators of CHs. Any solution of $P_{\mathcal{A}H}$ must satisfy all constraints in C_0 and minimizes error $\mathbb{E}(C\theta)$ by subsequent comparison of $E(C_i\theta)$ from the smallest to the highest index i (see Def. 5.14). The same minimization performs globally-better comparator of CHs (see Def. 4.9) comparing values $g(C_i, \theta)$ subsequently. As the values $E(C_i\theta)$ and $g(C_i, \theta)$ are defined by particular errors of $P_{\mathcal{A}H}$ and comparators of P_{CH} by the same expressions, we obtain the same set of solutions for both problems $P_{\mathcal{A}H}$ and P_{CH}.

Ordered-better comparator takes into account subsequent levels of CH P_{CH} like errors $\mathbb{E}$ of $P_{\mathcal{A}H}$ are compared from the smallest to the highest index of these tuples. If the values $e(c\theta)$ and $e(c\delta)$ are equal for all $c \in C_i$ neither error of $P_{\mathcal{A}H}$ nor ordered-better comparator of P_{CH} doesn't differ which of assignments θ and δ is better. Let us consider the first level j

where error function e is different for assignments θ and δ. Then θ is ordered-better than δ if $c \in C_j$ exists such that $e(c\theta) < e(c\delta)$ holds, and if $acv(c) \geq_{\mathbb{R}} acv(d)$ holds for some $d \in C_j$ then $e(d\theta) \leq e(d\delta)$ is entailed. However, this comparison corresponds to the ordering $<_l$ defined by Eqn. 5.5. As a consequence we obtain that the selection of better assignments is the same for both ACH given by $P_{\mathcal{A}H}$ and CH P_{CH}, i.e., they have the same set of solutions. $\square$

Within the section 4.4, we have proposed an algorithm for solving CH with ordered-better comparator. As the solutions of CH with this comparator correspond to solutions of ACH with local error, we may apply it on ACH to compute solution of underlying problem with annotations. This algorithm solving constraint system with variables' annotations via local error was implemented in SICStus Prolog [COC97, Int00] with attributed variables and mutable terms [Rud98c].

5.4 Variable Ordering

Variable ordering (VO) heuristics in CSPs tend to select those variables which are the most critical ones to be first instantiated (see Sect. 2.3). Our idea is to consider as the most critical ones exactly those variables of CSPs with preferences which are the most important.

By an early instantiation of more important variables it may be easier to assign them the *more acceptable values* as it is shown in the following example.

Example 5.6 Let us consider a binary CSP (for definition see Sect. 2.1) with unary constraints on all variables as soft constraints having associated a cost for each value in the domain of variable and with set of hard binary constraints. If we apply trivial value ordering heuristics selecting values in order given by costs of unary constraints then the more important variables would have a better possibility to be instantiated on values having higher cost.

Such preferences of variables (variables' annotations) may be able to express *user's preferences* like the earlier approaches together with his/her *expectations about difficulty of variable's assignments* (e.g., variable occurence corresponds to an activity asking a scarce resource, order of job expressed by variable is highly synchronized with other jobs, placement of an object is strongly constrained wrt. some of its coordinates — variables) and they are applied to overcome difficulties given by solving over-constrained problems or problems with large solution spaces. The most difficult variables would become a source of constraint propagation and their early instantiation may prune the solution space or avoid unsuccessful search bathes, both at the beginning and during the search.

Example 5.7 Let us imagine a simple constraint expressing that one place (`T1`) should be visited before the another (`T2`). An agent responsible for many places should attend the first place while the second place should be attended by an agent with much lower load. This relation may express the following constraint with annotations.

```
T1@strong < T2@weak
```

The annotation `strong` becomes a source of early instantiation of the variable `T1`. If it occurs, the constraint is satisfied and a reduction of domain of `T2` is less critical than would be a reduction for the variable `T1` wrt. stronger requirement on `T1`.

The first place may be accommodated by a more important customer than the second place. Annotations in the above constraint express such degree of importance. They become very interesting when considered together with preferential constraints for value ordering (e.g., each customer prefers certain hours of day to be visited by an agent). Any later instantiation of place visiting time would probably worse cost of value selected from domain of variable. This in general is not a problem (backtracking etc. would overcome this), but the efficiency is completely different (not only in practice, but even theoretical complexity may differ).

5.4.1 Computing Variable Ordering

Static Variable Ordering. Having a constraint system with variables' annotations $P_{\mathcal{A}} = (V, D, C,(\mathcal{A}, \preceq, \bigoplus), a)$, variable ordering will be computed via *global variable annotation* av. This VO belongs to a class of static VOs. Final computing VO depends on selection of one of annotation triples $\mathcal{A}_{sum}$, $\mathcal{A}_{avg}$, $\mathcal{A}_{gavg}$, or $\mathcal{A}_{vect}$. Instantiation of this triple should consider properties of given problem. If it is desirable to strictly differ among particular variables then taking into account level by level assignment via $\mathcal{A}_{vect}$ would be the best choice. Selection between average $\mathcal{A}_{avg}$, $\mathcal{A}_{gavg}$ and additive $\mathcal{A}_{sum}$ triples is related with possible overestimation of the number of variables as discussed in Sect. 5.1.1.

As a constraint with annotations due to $\mathcal{A}_{sum}$, $\mathcal{A}_{avg}$, or $\mathcal{A}_{gavg}$instances inequality `T1@10 < T2@1` from Example 5.7 may be considered having symbolic names replaced by corresponding numbers. The following example is concentrated on application of $\mathcal{A}_{vect}$ triple.

Example 5.8 Let us expect that we have several sets S_i of tasks each of them requiring a resource. Ideally each task is assigned to a different resource but such assignment may not be feasible (e.g., wrt. the number of resources). Any task from set S_i should get a distinct resource preferably than another task from set S_j such that $j > i$. Such requirements may be expressed by the following soft constraint with annotations from $\mathcal{A}_{vect}$.

```
different_times(s_11@(1,0,...,0), ..., s_1n_1@(1,0,...,0),
                s_21@(0,1,...,0), ..., s_2n_2@(0,1,...,0), ...
                s_m1@(0,0,...,1), ..., s_mn_m@(0,0,...,1)) ,
```

where $S_i = \{s_{i1}, \ldots , s_{in_i}\}$ for $i = 1 \ldots m$.

Now let us imagine that each task s_{ij} has also associated a cost w_{ij} expressing preferences among tasks of the same set S_i. Such requirement would be solved by replacement of the value `1` by w_{ij} in annotation of variable for task s_{ij}.

Let us consider a semantics of such constraint. Annotations of tasks influence variable ordering, tasks having more important annotations would be preferably assigned first. As early as it occurs some distinct resource remains still free.

Dynamic Variable Ordering. We would like to reflect by annotations also a dynamic behavior of constraint system during evaluation, i.e., domain variables are transformed to constants due to their value assignments, constraints become passive due to their satisfaction, they may be removed in over-constrained problems wrt. their relaxation. To take this consideration into account, we select a variable to be assigned via *dynamic global variable annotation* which corresponds to global variable annotation computed after applying current *partial* assignment.

Before presenting a definition let us shortly mention how sets $C\theta$, $V\theta$ differ from sets C, V for some partial assignment $\theta \in \Theta_W, W \subseteq V$. The set $V\theta$ is a subset of the original set V including only those variables which does not have assigned their values by partial assignment θ. For the set $C\theta = \{c\theta \,|\, c \in C\}$ holds. Some constraints $c\theta$ represent empty relation only as their set of variables $V\theta_{c\theta}$ becomes empty. Other constraints $c\theta$ have reduced their set of variables by the already assigned variables $V\theta_{c\theta} = V_c - (W \cap V_c)$. Due to constraint propagation, particular relations for constraints $c\theta$ contain possibly smaller number of elements than corresponding c after applying tuple projection (see Def. 3.18) on remaining variables in $c\theta$, i.e., we have $c\theta \subseteq c \downarrow^{V_c}_{V\theta_{c\theta}}$.

Definition 5.15 Let us consider $P_{\mathcal{A}} = (V, D, C, (\mathcal{A}, \preceq, \bigoplus), a)$ and some partial assignment $\theta \in \Theta_W$, i.e., $W \subseteq V$ holds. *Dynamic global variable annotation* dav_W combines variable's annotations in all constraints $C\theta$ where variables from the set $V\theta = V - W$ occurs.

$$dav_W \colon [V - W] \times \Theta_W \rightarrow \mathcal{A} \ \ , \ \ dav(v, \theta) = \bigoplus_{(c \in C\theta) \wedge (v \in V\theta_{c\theta})} a(c, v)$$

Such dynamic global variable annotation allows ordering of all remaining (unassigned) variables. The next variable $v \in V\theta$ to be assigned is such that its dynamic global variable annotation is the most important (or, more exactly, no variable having its dav more important exists), i.e., $\forall v' \in V\theta : dav(v, \theta) \preceq dav(v', \theta)$ holds.

Chapter 6

Timetabling

Educational, transport, sport, or employee timetabling are just examples of problems covered by the timetabling research area. Timetabling determines what time and place each course/exam will be given; when train/bus/aeroplane will depart/arrive and from which station/airport; what time, date, and place each match will be played; or designs each employee's work timetable. Anthony Wren [Wre96] defines timetabling as a special case of scheduling:

> *Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives.*

An intent of this thesis is to study educational course timetabling with special emphasis to university-based timetabling as a classical application area where various types of preferences need to be involved to obtain some acceptable solution.

Timetabling problems may be solved by different methods inherited from operational research such as graph coloring and mathematical programming, from local search procedures such as tabu search and simulated annealing, or from genetic algorithms. Our concentration will be devoted to constraint programming approach, references including also other solution methods may be found in surveys on educational timetabling [Sch95, CL98a] or in proceedings from the timetabling conference PATAT [BR96, BC98, BE00].

6.1 Problem Description

Course timetabling can be viewed as a multi-dimensional assignment problem in which students and teachers are assigned to courses, course sections, classes, or exams, and those meetings between teachers and students are assigned to classrooms and times. Let us describe particular components.

Course is taught one or more times a week during part of a year. Sometimes courses are split to multiple *course sections* due to the large number of students subscribed to a course.

Teacher is assigned to each course or course section.

Classroom of suitable size, equipment (laboratory, computer room, classroom with data projector, etc.), and location (part of building, building, campus, etc.) has to be assigned to each course or course section.

Student attends a set of courses. His/her selection is usually predefined by subscription either in a *class* taking an identical set of courses (usually at high schools) or in some *program* containing compulsory and optional courses (universities). In some universities, students are also allowed to subscribe almost any arbitrary selection of courses within *course pre-enrollment* process.

The timetabling problem consists of set of different tasks which differ with respect to each school. Common properties depends on a type of school. High and secondary schools create their timetables for classes with a small number of free choices for students, teachers and students are busy all day, classrooms have the same location and the timetable has to be created without any conflicts. Universities allow more choices for individual students, the programs are loosely structured, teachers with smaller number of teaching units are more flexible resource, classrooms have different size, equipment, and location, and resulting timetable is generated with minimal overlaps of courses having common students. High school timetabling problem is usually very tight, the complexity of university timetables worsens optimization requirement.

The sequence of particular tasks (e.g., time assignment or classroom allocation) or their interleaved solution depends on the complexity and tightness of the problem, decision criteria with their required sub-optimal or optimal satisfaction. High and secondary schools usually need to consider teacher and class-teacher assignment sub-problems only. At universities, we may describe two basic schemas distinguished by the sequence of tasks.

Master timetabling

1. Determination of number of sections
2. Teacher assignment
3. Time (and classroom) assignment
4. Student enrollment for courses wrt. published timetables
5. Student scheduling
6. Incremental changes of constructed timetable

Demand-driven timetabling

1. Student enrollment for courses
2. Determination of number of sections
3. Teacher assignment
4. Time (and classroom) assignment
5. Student scheduling
6. Incremental changes of constructed timetable

Let us study particular sub-problems which has to be solved via such automated timetabling, i.e., time assignment, classroom allocation, and student scheduling. We will also discuss possible inclusion of teacher assignment into the problem structure.

6.1.1 Time Assignment

Each course or course section must be scheduled during some time period, e.g., week. In high and secondary school, the problem is usually called *class-teacher* timetabling [CL98a] with course understood as a class-teacher meeting. Class-teacher meetings are scheduled without creating conflicts and satisfying some side constraints on the spread and sequencing of courses. While the high school timetabling schedules contiguous blocks of courses for particular classes, university timetabling allows more loosed timetables (even during evenings) for particular groups of students having the same *program*. University-structured programs include significant number of optional courses which are scheduled to satisfy different student requirements to the largest possible extent. *Student-oriented* timetables are required at universities where open credit system with just few strict requirements plays substantial role. Such timetables for individual students solve the problem of maximal participant satisfaction [SFW95].

The complexity of time assignment problems influences required time model. The simplified version considers only courses of fixed duration starting in non-overlapping time slices but the realistic model should include variable duration of courses at least. Increase of the solution space brings shorter time slots than possible duration of course, e.g., courses with duration from one to four hours starting each quarter of an hour; or different time slices for particular administrative units (faculty, department).

Let us present basic type of constraints which are often considered when the time assignment sub-problem is taken into account.

(C1.1) Consistency constraints: each teacher can only give one course at a time and each classroom can only host one course at a time.

(C1.2) Student time constraints: induced by school organization of teaching into class-teacher meetings, programs, or student-oriented meetings.

(C1.3) Teacher time constraints: requirements and preferences of teachers towards their courses.

(C1.4) General time constraints: time preferences or requirements (e.g., restrained or preferred time slices, maximal number of time slices per teacher per day)

6.1.2 Classroom Allocation

Classroom assignment plays substantial role at universities with classrooms of different size, equipment, and location. Courses should be scheduled into classrooms whose size is large enough (but not too large) to contain them. Special time constraints must be posted at schools with distant buildings, students need sufficient time between two consecutive courses due to transportation, teachers prefer closest buildings, and every such location changes should be minimized.

To obtain optimal solution, the problem should be solved together with time assignment but it is also solved separately to simplify the solution.

Such problem description gives us the basic types of constraints.

(C1.5) Required or preferred equipment (e.g., laboratory, computer room) is available at classroom for each course.

(C1.6) Classroom of suitable size must be assigned to each course.

(C1.7) Each course may have specified allowed or preferred set of its locations.

6.1.3 Section Assignment

Some schools having large groups of students for particular courses split them to course sections (groups). The courses are taught in multiple sections with possibly different teacher, time slice, and classroom. The aim of section assignment is to find suitable timetables for students with minimal conflicts for particular students while balancing section sizes and respecting room capacities.

The problem is often solved with pre-assigned times, classrooms, and teachers for course sections [LD86] and such problem instance is called *student scheduling*.

6.1.4 Teacher Assignment

The problem consists in assigning teachers to courses while maximizing satisfaction of preference function. For university-based timetabling, it is normally assumed that courses have assigned their teachers a priori and this assignment does not take the part in automatic timetabling. This fact is mainly due to the specific orientation of particular teachers at universities. Any choices among teachers are rather solved by discussion within faculty or departments. However, teacher assignment could become interesting for problems with large number of identical course sections where possible changes of assignment of particular teachers to course sections may substantially improve overall solution.

6.2 Current Constraint-based Approaches

Constraint satisfaction together with constraint logic programming is applied to a wide range of timetabling problems despite the fact that it belongs to quite new solving techniques with first implementations and papers starting in the nineties [BDP93, AB94, YKNW94, BGJ94]. Constraint logic programming is particularly well suited for timetabling problems. It allows the formulation of all the constraints of the problem in a more declarative way than other approaches [Laj96, GJBP96], and in some cases, CLP programs are more efficient than other kinds of programs [BDP96].

Aim of this section is to concentrate on solving of time assignment and classroom allocation problems for schools with programs or classes which are often solved via constraint programming approach. Student-oriented schedules which would require solving of section assignment problem were not commonly constructed via constraint programming approach — this task will be discussed as a part of our work within Sect. 6.4 separately. Not solving section assignment problem there wasn't a substantial need to solve teacher assignment problem (see discussion in Sect. 6.1.4).

6.2.1 Problem Modelling

Suitable representation of the problem by the variables and their domains can greatly influence the way constraints can be formulated and overall computational effectiveness. Time assignment and classroom allocation problems prescribe two basic types of domain variables — *time* and *classroom variables*.

Time variables express which *time slice* of the *day* each course is taught. The most common time model represents time by consecutive numbering of all time slices [Laj96, GJBP96, HW96, AB94, BKMQC97], usually running from Monday morning to Friday evening. Every time variable denoted by `Time` has the domain

```
Time in 0..(NbTimeSliceDay * NbDays),
```

where `NbTimeSliceDay` is a number of time slices per day and `NbDays` is a number of days. Variable for day of course is then derived with help of constraint for each course if some constraints on day are posted. Slightly different representation may define for each variable two basic domain variables corresponding to time slice within day `TimeOfDay` and day variable `Day` [FHS95, BGJ94]. Differences between these models are seen namely within labelling process where variable and value ordering heuristics are computed wrt. either to `Time` or `TimeOfDay`, `Day` variables. Additional problem dimension requiring another variable `Week` may introduce teaching *week* of course [GKM98, GM99].

Some schools allows interruption of particular courses which is usually included into problem solution via predefinition of *course blocks* each having separate time variables [AB94, FHS95, Laj96]. It means that it is always supposed that basic timetabling units may not be interrupted.

Generally courses considered within course timetabling have constant durations (none domain variable for course duration need to be included). Particular models may be distinguished by taking into account all courses of the same *fixed duration* or the different *variable duration*. Model with courses of fixed duration [BKMQC97] significantly simplifies solution of overall problem as all requirements for unary resource (e.g., courses of the same program or sharing the same teacher ask for non-overlapping via sharing of unary resource) may be modeled using `alldifferent` constraint and requirements on cumulative resource via set of `atmost` constraints (see Sect. 2.4.1 and Fig. 2.1(e)). Unfortunately more realistic problem solutions must consider courses of variable duration [HW96, GJBP96, GKM98, AB94, Laj96]. Mostly such courses are modeled with help of domain variable expressing starting time of course and constant for duration of course. Then `disjunctive` constraint expresses basic relation of courses non-overlapping and `cumulative` constraint asks for cumulative resources. A different solution for courses of variable duration proposes [Laj96], where starting time with duration is replaced by a set of the so called *class variables*, each representing one time slice of a course. Additionally, requirement on continuity of these time slices is stated. Simpler constraints may be applied (`alldifferent` and `atmost`) but they may be outweighed by the increased size of the solution space.

Some schools may require the so called *refined time slicing* [GJBP96, HW96, GKM98, GM99] which means that time slices (e.g., 15, 30 minutes, 1 hour) may be even shorter than any course duration (e.g., 1, 2, 3 hours). Such time slicing may ensure appropriate time for breaks between courses in different buildings, appropriate time for lunches, proper time coordination with other timetables, etc. This requirement increases size of the solution space due to enlarged domain of variables. It also necessitates application of more complex global constraints like `disjunctive` as courses may partially overlap even if they have the same fixed duration.

Representation of classroom variables together with selection of appropriate global constraints are tightly linked with selected time model and with requirements on classroom allocation. Courses of fixed duration are usually represented by one classroom variable for each course — trivial time model is followed by simplified classroom representation. Courses of variable duration may need either one or more classroom variables for each time slice, depending on allowed or prohibited changes of classrooms during one course. Basically classrooms may be handled via global constraints for unary and cumulative resources wrt. selected representation of variables as it was described for time variables. Further consideration are discussed in Sect. 6.3 presenting our approaches for solving classroom allocation problem.

6.2.2 Preferences & Search

Constraint-based methods allow to separate definition of the problem which is given by constraints from search of the solution space. For timetabling problems, search within such space is often derived from preferences within the problem to find such solution which could be accepted by both faculty and students.

Basically, search within solution space is performed without any global optimization criteria via labelling heuristic or other non-systematic constraint relaxation methods. More sophisticated approaches define some objective function over preferences within the problem with aim to optimize their value. More specifically such definition often leads to the weighted CSP (see Sect. 3.1) and relaxing the constraints wrt. their costs.

Labelling Heuristics

Within the traditional CSP approach, preferences for selecting feasible solution are implemented with the help of some labelling heuristics [AB94, GKM98]. Goltz et al. [GKM98] applies the typical solutions — given unary soft constraints with priority are integrated into the solution search through value and variable ordering heuristics. Azavedo et al. [AB94] describe decision steps leading to computing final labelling heuristics which instantiate hard variables first, e.g., courses with demanded classrooms, or longer durations.

Non-Systematic Constraint Relaxation

An expensive search of overall solution space may be replaced by relaxation of difficult constraints. Solutions described within this section try to relax constraints in order given by some preferences but any global criteria considering overall number or quality of relaxed constraints is not taken into account.

Papers [GJBP96, BGJ94] propose automatic relaxation systems but they don't apply them within their timetabling systems where all relaxations are done manually. The reason is that they are not able to handle global constraints because the system manipulates with only linear constraints.

Semi-automatic constraint relaxation is proposed and implemented by the so called *virtual time slice* [Laj96] such that each infeasible course is placed into such virtual time slice. Course becomes infeasible if it may not be scheduled to any non-virtual time slice. All such infeasible courses are timetabled sooner with help of appropriate variable ordering in subsequent runs. Courses which remain infeasible even after this treatment are sectioned (split to several courses) and solved separately reducing the scope of conflicts between old and newly sectioned courses.

[CKLW96, WZ98] introduce *equivalent reversing* to relax suitable constraints. When some course is not successfully timetabled, soft constraints collected for this course are relaxed with help of their hierarchical ordering (see Chapter 4). When a course becomes unschedulable even with all relaxed soft constraints, the equivalent reversing tries to timetable it with help of rescheduling of equivalent courses (enrollments in the same size range, same timezone, etc.). If this approach does not succeed to timetable some course, it is placed in a list of unschedulable courses to be manually placed later.

Cost-based Constraint Relaxation

A formulation of an over-constrained part of the problem may be included into the objective function of constraint optimization problem. Associating costs to constraints or to values in domain of variables for particular constraints, assignments with optimal costs are searched. As a consequence we obtain a weighted constraint satisfaction problem. However, all mentioned approaches find some sub-optimal solution due to the size of the solution space for real-life problems.

Timetabling system described in [FHS95] searches for optimal solution by branch and bound algorithm but the constraints defining costs don't play a role of active constraints — they are implemented through value and variable ordering only. System searches either for solution with optimal cost or for solution which cost is constrained by threshold.

Similarly branch and bound algorithm with selection of the preferred values from domains of variables (preferred times are tried earlier) is applied by [HW96]. They construct any-time algorithm such that user can interrupt the optimization at any time and request currently best solution. Henz et al. recognize that going for the globally best solution is not feasible. When the size (120/90 courses for [FHS95]/[HW96], resp.) and type (classical course timetabling without student schedules) of the solved problems are considered, it remains doubtful whether this approach would scale up.

Yoshikawa et al. [YKNW94] solves high school timetabling problem via general-purpose constrained solver COASTOOL able to handle costs of particular constraints. Within the first step, they apply constraint propagation while any consistent values remain in domain of variables and then they assign values with the least cost (look-ahead greedy algorithm). The cost of constructed solution is improved via hill climbing [MJPL92] algorithm up to falling into local minimum. Later [KYN99] propose a new heuristic repair method to escape from this local minimum. They also improved initialization method which applies a heuristic repair method whenever inconsistencies are found during construction of solution.

Weighted CSPs with costs for values in domain of particular variables was applied in course time assignment problem [AM00, AM98] where cost (they call it *assessment*) of each value in the domain of time variable is derived during computation. Final value with the best cost is selected during value selection. They found the first solution satisfactory but costs included in the problem could be extended to search by branch and bound for some better solution. This implementation is the first approach with cost propagation within CLP approach. However, described solution deals with department-sized problem only.

Similar method is implemented in classroom allocation problem [ASW00] with costs computed for values in the domain of classroom variables. All soft constraints may be expressed by unary function returning how some classroom is desired for given course date. Optimization requirement summarizes costs for all course dates and it searches for sufficient sub-optimal solution. They have shown that cost propagation works efficiently for university-sized classroom allocation problem (the biggest building consists of 40 classrooms where about 1000 courses is held).

6.3 Classroom Allocation

Within Sect. 6.2.1 we have discussed general models for classroom allocation problems giving description of variable representation together with applied global constraints. In the following, we would like to concentrate on specific problems of classroom allocation and their solution we have proposed and applied within [RM00, RM99a].

6.3.1 Identical Classrooms

Let us consider a typical situation arising when courses are scheduled to several classrooms of the same characteristics (e.g., size, equipment). Instead of posting `disjunctive` constraint for each classroom, these classrooms are considered together by introducing one cumulative resource of capacity `ResourceLimit`. Each course represented by `StartJ` and `DurationJ` would require unit capacity of cumulative resource, i.e. `ResourceJ` is equal to `1` for each `J`. Such requirements allows application of global constraint `cumulative` from Figure 2.1(b) on page 9. Allocation of particular classrooms may be postponed after time assignment which significantly reduces the size of the solution space. This approach also decreases the number of symmetries in CSP.

Exact allocation of classrooms takes $O(\texttt{ResourceLimit} \times \texttt{TimeSlices})$ steps where `TimeSlices` denotes the overall number of time slices. A result of exact allocation for Fig. 2.1(b) is shown on Fig 6.1. Let us note that such allocation assigns just the one classroom

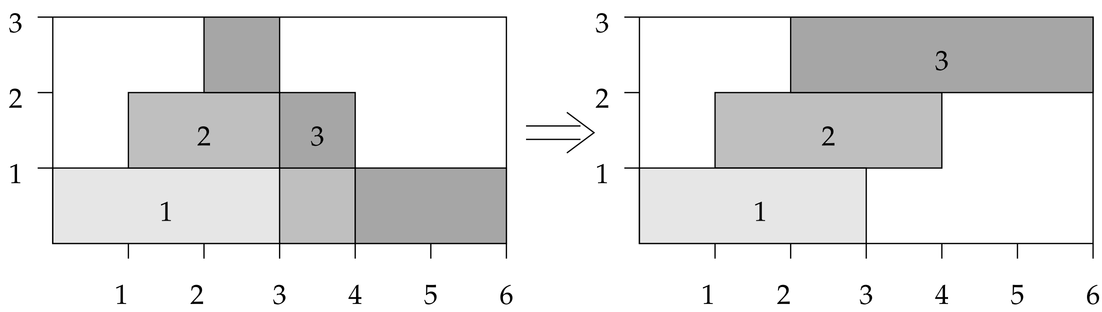


Figure 6.1: Exact allocation of classrooms for `cumulative`([0,1,2],[3,3,4],[1,1,1],3)

for each course during all time slices of the course. This fact may become important when courses are not allowed to change classrooms during their duration.

The following part shows construction of the desired solution together with its complexity. The `cumulative` constraint ensures that smaller or equal number of courses than capacity `ResourceLimit` is placed in every time slice. Now let us take classrooms in any fixed order. Classrooms are assigned to courses subsequently. Each classroom is assigned to courses in increasing time without any gaps if possible. We have proceed through all classrooms and all time slices which gives the resulting complexity $O(\texttt{ResourceLimit} \times \texttt{TimeSlices})$. When some course have remained without assigned classroom at the end, then some gap beginning at the starting time of course has to exist wrt. existing cumulative resource. But all courses were assigned in order given by starting time and any such gap may not exist.

6.3.2 Diverse Classrooms

Attention of this section is concentrated on allocation of classrooms having different capacities (C1.6). Having set of classrooms of various capacities it is not suitable to assign courses to classrooms of exact capacity they require due to resulting unbalanced room occupation which could later lead to computing of much worse solution or no solution may even exists. Often it is expected that courses may be placed into the classrooms of larger size than they require and this possibility will be exhibited within this section. An extension of presented solutions towards the requirement of suitable equipment (C1.5) is presented at the end of the section.

Classroom Allocation After Time Assignment

First we will exhibit a solution which would allow to postpone any decision about classroom assignment after completion of time assignment.

Let us suppose that each classroom `Id` is represented by its `Capacity`

```
classroom(Id,Capacity)
```

and each course by its starting time `Start`, `Duration`, and number of subscribed students `Students`.

```
course(Start,Duration,Students)
```

One `cumulative` constraint

```
cumulative(StartList,DurationList,ListOf1,Limit)
```

is posted for each possible capacity of classroom denoted by `Size` and constant `ListOf1` denotes a list of 1, which has the same length as `StartList`. `DurationList` has to include durations of courses which are contained in `StartList`. Variables `StartList` and `Limit` should satisfy following properties

$$\texttt{StartList} = \{\texttt{Start} \mid \texttt{course(Start,Duration,Students)} \wedge \texttt{Students} \geq \texttt{Size}\}$$
$$\texttt{Limit} = \mathrm{card}\{\texttt{Id} \mid \texttt{classroom(Id,Capacity)} \wedge \texttt{Capacity} \geq \texttt{Size}\}$$

Example 6.1 Let us imagine small example with 2 rooms for 40 students, 3 rooms for 20 students, and 1 room for 10 students. Set of `cumulative` constraints follow

```
cumulative(Starts_of_courses_with_size_40,...,2),
cumulative(Starts_of_courses_with_size_20_40,...,5),
cumulative(Starts_of_all_courses,...,6).
```

The first constraint ensures that the largest courses are accommodated into the largest classrooms, the second constraint allows to place medium-sized courses into classrooms for 20 students and also to classrooms for 40 students if they are not already asked for by the first constraint. The third constraint may move small courses between all classrooms in condition that they are not occupied by any larger courses at the same time.

Presented solution restrains importance of classroom variables for many problems and it allows splitting of the problem into primary time assignment and secondary classroom assignment. Separated sub-problems have significantly decreased size of solution space and a solution of the classroom assignment with known starting times of courses also becomes easier. Unfortunately this solution does not exclude changes of classrooms during teaching of particular course as it was shown for identical classrooms in Sect. 6.3.1.

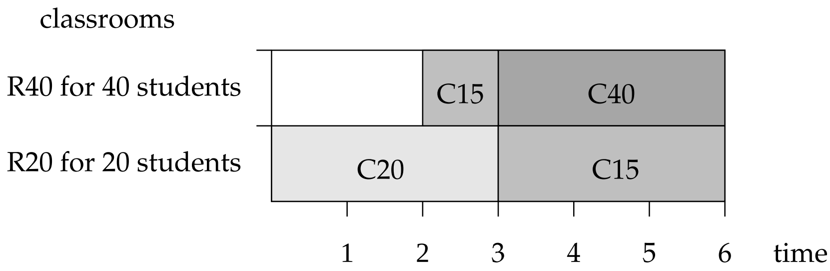


Figure 6.2: Course C15 for 15 students has to be taught in two classrooms.

Example 6.2 Let us imagine that course with small number of students (see course C15 for 15 students in Fig. 6.2) is assigned to large classroom (R40) during time slice t (2) because all smaller classrooms (R20) are already occupied. During time slice $t+1$ (3), some smaller classroom (R20) becomes empty, large classroom (R40) is required by some larger course (C40) which may not be moved to the small one wrt. its number of students, and the first course (C15) has to change its classroom.

Final assignment of courses into classrooms must include one classroom variable for *each time slice* of course. These variables may be instantiated with help of `alldifferent` constraint posted for each time slice on corresponding classroom variables of all courses scheduled during this time slice. Overall number of classroom changes for one course may be decreased by minimization. Such minimization is run for classroom variables only and it may be decomposed into several parts each including variables for courses `Cs` taught in the same teaching day. Let us denote classroom variables for one course `C` of duration `N` by `[Room1,...,RoomN]`. `RoomI` (for `I=1 ... N`) is a domain variable over natural numbers (each number corresponds to classroom). Objective function which value will be minimized corresponds to

$$\sum_{(\mathtt{C},\mathtt{N})\in\mathtt{Cs}} \sum_{\mathtt{I}=1}^{\mathtt{N}-1} \mathrm{abs}(\mathrm{sign}\,(\mathtt{RoomI}\ -\ \mathtt{RoomIpp})) \tag{6.1}$$

where `RoomIpp` is the `(I+1)`-th element of the list `[Room1,...,RoomN]` and sign returns -1, 0, or 1 for positive, zero, or negative value of the argument, resp.

Partition of Classroom Allocation

Having requirement on prohibited classroom changes during teaching of one course, problem definition should be changed not to hold on generate-test principle, i.e., time assignment is generated and than tested if classroom assignment without classroom changes exists (the value of objective function from Eqn. 6.1 has to be equal to 0). In the following, we will study possible solution of a new problem instance.

During time assignment we need to ensure that just one proper classroom is allocated for each course during its duration. To achieve this aim, classroom allocation could be performed before any time assignment. During time assignment, each classroom would be represented via one unary resource. However, such problem solution is rather restricting as it allows lack of flexibility during time assignment. Having the classrooms of identical capacity, we may gather them into one cumulative resource within time assignment and postpone exact allocation of classrooms after time assignments. Due to proof for classrooms of identical capacity in Sect. 6.3.1, such assignment of classrooms has to exist.

Let us consider particular steps of proposed solution in detail. Each classroom is identified by natural number corresponding to its order in the sequence of all classrooms sorted by increasing capacity. Classrooms of the same capacity are taken in any fixed order. Initial domain of each classroom variable for course of certain capacity corresponds to identifiers ($\in \mathbb{N}$) of all classrooms having the required or any larger capacity. Initial classroom allocation may be solved via constraint for cumulative resource, e.g.,

```
cumulative(RoomList,ListOf1,DurationList,NbTimeSlices).
```

`RoomList` contains proposed classroom variables for all courses and `DurationList` is a list of their corresponding durations. Each course duration represents a capacity of cumulative resource required by each course from overall capacity of cumulative resource `NbTimeSlices`. The constant `NbTimeSlices` is a number of time slices during the whole scheduled period, e.g., week. `ListOf1` corresponds to list of 1 as each course asks for one classroom. Let us note that such shape of `cumulative` constraint corresponds to the example from Figure 2.1(c) on page 9.

Example 6.3 Let us imagine small example with 2 classrooms `r40_1`, `r40_2` for 40 students, 3 classrooms `r20_1`, `r20_2`, `r20_3` for 20 students, and 1 classroom `r10_1` for 10 students. Such classrooms may be sorted in the following way

```
r10_1≡1  r20_1≡2 r20_2≡3 r20_3≡4 r40_1≡5 r40_2≡6
```

Let us consider courses requiring classroom of capacity 10, 10, 20, 20, and 40 having durations 1, 2, 3, 1, and 4, resp. As a consequence we get

```
R1,R2 in 1..6, R3,R4 in 2..6, R5 in 5..6,
cumulative([R1,R2,R3,R4,R5],[1,1,1,1,1],[1,2,3,1,4],NbTimeSlices),
```

where `R1`, `R2`, `R3`, `R4`, `R5` represent classroom variables for particular courses.

After labelling of classroom variables, each course have assigned one particular classroom. Instead of posting disjunctive constraint for each classroom during time assignment, one cumulative constraint may be posted for all courses asking for classrooms of the same capacity. As we have already mentioned existence of final classroom assignment after processed time assignment was proven in Sect. 6.3.1.

Unfortunately presented solution still allows a lower degree of flexibility than the one presented in previous section for courses with allowed classroom changes. During time assignment it could occur that classroom of sufficient capacity remains to be free within certain time slice but we have already restricted ourselves to classrooms of different capacity which may be all already engaged within this time slice. Such behavior may be partially eliminated via balanced occupation of classrooms but it still does not achieve the same degree of flexibility. It should be carefully decided whether possible changes of classrooms which may be for certain problem instances minimal outweigh this disadvantage.

Possible Extensions

Proposed sets of overlapping `cumulative` constraints for first solution and the discussed `cumulative` constraint for second solution may introduce only one part of the problem representing courses of the same equipment. To include constraint (C1.5) for classrooms with different equipment (e.g., lecture hall, computer room), one overlapping set of `cumulative` constraints (or one `cumulative` constraint for second solution) is added for each type of room. However, such straightforward solution is only applicable for pairwise disjoint sets of classrooms, where one `cumulative` set contains classrooms of the same equipment.

Both implementations of the constraint (C1.6) expect that size of particular courses is known before constraints are stated. Sometimes it may occur that the size of courses decreases (e.g., pre-enrollments of students into courses may not be fully satisfied) and it may seem that described solution is not very suitable wrt. possible relaxations of required size of classroom. However, the complexity of proposed model may increase such that possible profit would be outweighed. Fortunately the number of students which are not able to attend pre-enrolled course may be neglected — scheduling students into classroom is able to satisfy more than 94 % course pre-enrollment requirements [RM00, BvBM98, Car00, SFW95].

6.4 Student-Oriented Timetables

We have seen that student time requirements (C1.2) for class-teacher meetings or programs may be efficiently represented via global constraints as a set of strict requirements (see Sect. 6.2.1). Constraints within student-oriented timetabling would have to express that courses of each student may not overlap in time. For schools with diverse student timetables, straightforward addition of such disjunctive constraint for each student as a hard constraint would make the problem over-constrained. The aim of this section is to present our approach allowing inclusion of this sub-problem into the CLP paradigm.

6.4.1 Student Conflict Minimization

Our intent will be devoted to the definition of suitable objective function which value could be optimized in final solution. Such optimization criteria for student-oriented schedules should minimize the sum of course sections, which student is not able to attend wrt. concurrent schedule of other his (her) course sections, for all students. Each generated timetable may be evaluated by the so called *StudentsConflict* reflecting solution quality wrt. the value of objective function. The following part defines *StudentsConflict* and its approximation and describes the process of computing sub-optimal solution of time assignment sub-problem with help of *StudentsConflict* approximation.

Each student pre-enrolls some set of courses. We will suppose that section assignment gives as a result fixed course section(s) for each his (her) course, i.e., we may optimize overall number of the conflicting sections for any student. Because we don't know which of conflicting course sections will be selected by particular student, we need to consider some selection criteria c to express attendance of course sections by each student. $\mathit{StudentsConflict}(c)$ will summarize the number of course sections which students are not able to attend due to selection criteria c.

$$StudentsConflict(c) = \sum_{s} ConflictingSection(c, s) \tag{6.2}$$

$ConflictingSection(c, s)$ denotes the number of course sections which student s is not able to attend wrt. selection criteria c. To obtain correct interpretation of solution quality, we have to consider $StudentsConflict(c_{max})$ with worst possible selection of course sections by students as our cost estimation.

$$\exists c_{max} \forall c \; StudentsConflict(c_{max}) \geq StudentsConflict(c)$$

Example 6.4 Let us demonstrate above statements by a small example. We will consider contribution of one student to the sum *StudentsConflict*. Selected student is assigned to course sections {i004, i005, i002} within section assignment. Part of generated timetable including these course sections is shown at Fig. 6.3. Student s may decide to cancel ei-

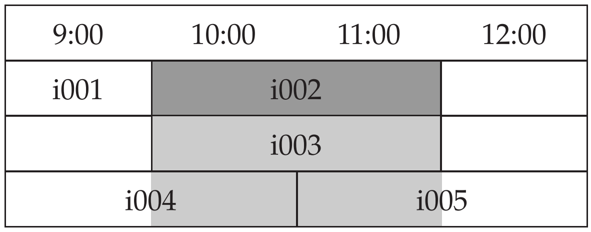


Figure 6.3: Small example of generated timetable

ther course section i002 ($ConflictingSection(c_1, s) = 1$) or course sections i004 and i005 ($ConflictingSection(c_2, s) = 2$). The value $StudentsConflict(c_{max})$ has to subsume the worst case contribution for each student, which means $c_{max} = c_2$.

Let us define $TwoConflict(i, j)$ as a number of students which pre-enrolls both course sections i and j. For course section i, $TimeOverlap(i)$ is a set of all course sections having overlap with i, $VarOrder(i)$ gives instantiation order of time assignment variable for course section i in the sequence of time assignment variables.

Let us denote $StudentsConflict_{inc}$ as our approximation of $StudentsConflict(c_{max})$ which may be incrementally computed and optimized during time assignment of particular course sections, i.e., during labelling of time variables.

$$StudentsConflict_{inc} = \sum_{i} SectionConflict(i) \tag{6.3}$$

Each value of $SectionConflict(i)$ contributes to the resulting $StudentsConflict_{inc}$ by sum of $TwoConflict(i, j)$ for all course section j having already assigned their starting time during labelling of time variables (($VarOrder(j) < VarOrder(i)$) and overlapping with i ($j \in TimeOverlap(i)$)).

$$SectionConflict(i) = \sum_{\substack{j \in TimeOverlap(i) \\ VarOrder(j) < VarOrder(i)}} TwoConflict(i, j) \tag{6.4}$$

Example 6.5 Let us go back to the timetable at Fig. 6.3 and suppose that times of course sections were assigned in order i004, i005, i003, i001, and i002. For example, the value *SectionConflict*(i005) is equal to 0 because none of previously assigned course sections (only i004) overlaps with i005. Last assigned course section i002 overlaps with i004, i005, and i003. Its *SectionConflict* is equal to the sum *TwoConflict*(i002,i004) + *TwoConflict*(i002,i005) + *TwoConflict*(i002,i003).

Corollary 6.1 The value $StudentsConflict_{inc}$ is an upper bound of *StudentsConflict*(c) for any selection criteria *c*, i.e.,

$$StudentsConflict_{inc} \geq StudentsConflict(c_{max}) \ .$$

Proof: $StudentsConflict(c_{max})$ summarizes $ConflictingSection(c_{max}, s)$ for all students s (see Eqn. 6.2). $ConflictingSection(c_{max}, s)$ includes some selection of overlapping course sections for student s which we will denote cs_i for i going from 1 to $ConflictingSection(c_{max}, s)$. Each cs_i is subsumed into $ConflictingSection(c_{max}, s)$ if it overlaps with some course section cs_{io} also pre-enrolled by student *s*. This overlap is either included in $SectionConflict(cs_i)$ for $VarOrder(cs_{io}) < VarOrder(cs_i)$ or contributes to $SectionConflict(cs_{io})$ for $VarOrder(cs_i) < VarOrder(cs_{io})$. □

$StudentsConflict_{inc}$ also cumulates some additional fail contributions. These contributions are related to multiple course sections of one student overlapping at the same time and it may occur if at least three course sections are cumulated into subsequent time slices.

Example 6.6 Let us consider a student attending courses sections {i002, i003, i004} and the timetable from the Fig. 6.3. A contribution to the objective function $StudentsConflict(c_{max})$ corresponds to 2 but its approximation $StudentsConflict_{inc}$ is increased by 3 for any ordering of course sections during time assignment.

The resulting quality of our upper bound depends both on selection criteria *c* and on above described cumulative error. However, these inaccuracies decrease with increasing quality of generated timetable and may be even abandoned for sufficiently fair solutions[1].

6.4.2 Soft Disjunctive Scheduling Problem

Within the last section we have shown how the problem may be seen as an optimization problem and defined its objective function via Eqns. 6.3 and 6.4. Here we will propose the so called *soft disjunctive scheduling problem* which instance is also discussed student conflict minimization problem.

First let us introduce *soft disjunctive (scheduling) constraint*. Disjunctive scheduling will be understood in its standard interpretation as a scheduling of disjunctive activities (see Sect. 2.4.1) and the attribute soft will express that some of the activities may overlap as their disjunctive scheduling is an over-constrained problem. Different criteria can be defined expressing the most desirable non-overlapping of activities. Examples are maximization of the number of non-overlapping activities or the preferred non-overlapping of activities of

[1] Let us expect that at least 90 % requirements in course pre-enrollment would be satisfied. Having average number of courses for each student equal to 10, it would result in approximately one course per student which he/she is not able to attend. Taking into account number of time slices within a week, such average overlapping doesn't become substantial for discussed fail contributions

higher rank. The second criteria requires some ordering or ranking of particular activities within problem definition.

Example 6.7 Let us consider a student having pre-enrolled 6 course sections.

```
C1@required, C2@required, C3@strong, C4@strong, C5@medium, C6@weak
```

Such ordering expresses that student has to visit courses C1 and C2 as they are compulsory, he would like to attend optional courses C3, C4 and prefers them over C5, C6, and his least preferred course is C6.

Let us note that such ordering of activities may be introduced within the problem definition with help of variables' annotations (see Chapter 5).

Soft disjunctive scheduling problem (SDSP) is given by a set of soft disjunctive scheduling constraints

$$\texttt{softDisjunctive([Start}_{i1}\texttt{, ... ,Start}_{im_i}\texttt{],}\quad \texttt{[Duration}_{i1}\texttt{, ... ,Duration}_{im_i}\texttt{])} \text{ for } i = 1, \ldots, m_i$$

where variables $\texttt{Start}_{ij}$, $\texttt{Duration}_{ij}$ belong to a set of variables

$$\texttt{S} = \{\texttt{Start}_l \mid l = 1, \ldots, n\} \ , \ \texttt{D} = \{\texttt{Duration}_l \mid l = 1, \ldots, n\} \ , \tag{6.5}$$

resp. Two soft disjunctive constraints may have common variables.

With help of standard semantics of disjunctive constraint, each SDSP P may be rewritten into a *binary SDSP* P_2 such that all soft disjunctive constraints have only two activities.

$$\texttt{softDisjunctive([S1,S2], [D1,D2])} \qquad \texttt{S1,S2} \in \texttt{S},\ \texttt{D1,D2} \in \texttt{D}$$

Because the original constraints may share some variables each soft disjunctive constraint may possibly occur in a new SDSP w times. Cardinality w of such soft disjunctive constraint in SDSP P_2 certainly expresses how important non-overlapping of its activities is and it may be called a *cost* of soft disjunctive constraint. We may even construct *conflict matrix* having $n \times n$ elements with each cell corresponding to the cost of soft disjunctive constraints between activities from the set of activities given by S, D (see Eqn. 6.5). Non-existence of soft disjunctive constraint for some activities would naturally infer a zero value in corresponding cell of conflict matrix.

If all activities in SDSP P have a constant duration then resulting binary SDSP P_2 is a binary constraint satisfaction problem. Another simplification may be resolved when all activities have the same unit duration[2]. Then resulting binary CSP contains only binary constraints representing inequalities $\texttt{S1} \neq \texttt{S2}$.

Solution Approaches

Transformation of the general SDSP into the SDSP P_2 allows us to handle original problem with help of frameworks for solving constraint satisfaction problems with preferences (see Chapter 3). Preferences in the problem P_2 can be expressed by costs within the conflict

[2]Within timetabling problem, we would require fixed duration of courses and model without refined time slicing (see Sect. 6.2.1).

matrix. The SDSP P_2 can be handled via weighted constraint satisfaction (see Sect. 3.1) and fuzzy constraint satisfaction (see Sect. 3.4), the basic representatives of different complexity classes of CSPs with preferences (see Sect. 3.8.3).

In *weighted CSP* approach, cost of each soft disjunctive constraint in SDSP P_2 may represent a weight of constraint. Solution of SDSP P_2 is then such assignment of variables in `S`, `D` that sum of costs of violated soft disjunctive constraint is minimized (see corresponding Defs. 3.2 and 3.3 for weighted CSPs).

Fuzzy CSP interpretation will try to find solution of SDSP via min-max optimization. First we will define fuzzy relation corresponding to each soft disjunctive constraint with cost w via Eqn. 3.1 and normalization of costs into unit interval. Having n activities, maximal cost correspond to $\frac{n^2-n}{2}$ which is the maximal cardinality of any soft disjunctive constraint in SDSP P_2. Each cost w may be transformed into $w' = \frac{2w}{n^2-n}$ to achieve the normalization. If a soft disjunctive constraint is satisfied then its level of preference corresponds to 1 and if it is not then it corresponds to $1 - w'$ (definition of fuzzy relation for soft disjunctive constraint). Solution of such fuzzy CSP is obtained via search of assignment of variables in `S`, `D` (see Eqn. 6.5) having the maximal satisfaction degree where satisfaction degree of assignment is given by minimal level of preference among all soft disjunctive constraints.

Relation with Student Conflict Minimization

Soft disjunctive constraint can be seen as requirements of one student within course pre-enrollment. Summarized course pre-enrollment informations define SDSP over time variables.

Objective function proposed in Sect. 6.4.1

$$\mathit{StudentsConflict}_{inc} = \sum_{i} \sum_{\substack{j \in \mathit{TimeOverlap}(i) \\ \mathit{VarOrder}(j) < \mathit{VarOrder}(i)}} \mathit{TwoConflict}(i, j)$$

corresponds to satisfaction degree of weighted CSP over soft disjunctive constraints between course sections i and j having weight equal to cost of this constraint, i.e., $\mathit{TwoConflict}(i, j)$. The condition $j \in \mathit{TimeOverlap}(i)$ expresses that soft disjunctive constraint is not satisfied. By the second condition $\mathit{VarOrder}(j) < \mathit{VarOrder}(i)$, we take into account each cost of unsatisfied soft disjunctive constraint only once.

Having proposed interpretation through fuzzy CSP, we could also perform search of optimal solution for student conflict minimization problem via fuzzy min-max optimization.

6.5 Variables' Annotations in Timetabling

In Sect. 6.2.1, we have written about two basic types of variables in timetabling problem, about classroom variables and time variables describing each course. Annotations of such variables may express users' preferences given by facts like how many students are enrolled to the course, seniority of the course teacher (professor, assistant, ...), or required classroom for course (more or less occupied). Annotations expressing expectation about difficulty of variable's assignment may describe how demanded some resource is (classrooms or teachers) or how critical (constraining) some requirements on variable is (e.g., highly restricted time for course expressed by constraint).

These preferences are combined via variable ordering method (see Sect. 5.4) to achieve that *both* more important and more critical variables will be instantiated first. This instantiation order also allows us to assign more preferred values to such variables prior to others which may not be so important. That way, soft unary constraints on particular variables may be applied in order influenced by annotations. Remaining non-unary soft constraints are posted (if possible) to prune the solution space as a consequence of assignment of variable which occurs in the constraint.

6.5.1 Timetabling Constraints with Annotations

Semantics of annotations depends on the context of constraint where variable with annotations occurs. We will give several examples to demonstrate their typical semantics.

Hard Constraints. Each course section requires suitable teacher (C1.1). This constraint adds annotation for time variable of course section corresponding to the seniority of the teacher to promote teaching times of course wrt. given teacher.

To make timetables acceptable for teachers, we have to restrain the number of time slices for teacher per day (C1.4). The value of time slices is related with annotation — more important annotation corresponds to the smaller number of time slices. Such annotation reflects following relation: the lower the number of time slices is the more difficult this condition becomes.

Some teachers may prohibit or order particular teaching times (C1.3) and annotation expresses how strong such requirement is to enforce possible earlier assignment of highly constrained time variables. While full-time teachers are available in a non-restricted way, requirements of partial-time teachers may have strong impact on possible teaching time of their courses. That way, annotations reflect possible flexibility of particular teachers.

Soft Constraints. Unary constraints may prefer certain time slices for courses (e.g., times from 9 a.m. to 5 p.m.) or restrain others like Friday's afternoon (C1.4). Particular value ordering may be specific wrt. teacher's requirement (C1.3). Annotation of variables in all these constraints may stress more frequent times, smaller required domains, and/or who and why is asking for special time assignment.

Similarly unary constraints may express within classroom allocation which classrooms are the most or the least preferred by the teacher (C1.5)+(C1.7). Such requirement may again stress how demanding such requirement is and who/why is asking for it.

Time dependency constraint (C1.3) includes non-standard requirements of particular teachers. Set of course sections should be taught in the same day, some lectures should be taught before their corresponding seminars. Let us consider a constraint expressing that lecture (`T1`) should precede all seminars (`T2,T3,T4`).

```
before(T1@strong,[T2@medium,T3@weak,T4@weak])
```

Annotations in these constraints represent seniority of teacher.

Most of teachers would like to attend special colloquy taught by invited lecturers. Annotation of starting time of the colloquy in this constraint belongs to the most important annotations. Later assignment of this variable would be very problematic. This constraint also includes times of all course sections taught by those teachers. Their annotation is again given by the seniority of each teacher to express which violation would be less important. Semantics of such constraint makes promising an application of hierarchical annotations which would emphasize the time variable for colloquy, e.g.,

```
differTimes(Colloquy@(1,0),CourseI@(0,1),CourseJ@(0,2),...)
```

Teaching (PhD.) students may prefer non-overlapping of several courses. Some non-overlapping courses are specified as required constraints but most of them include student priority to satisfy such requirement (see also Example 6.7). This priority is directly translated into annotation of particular course section.

```
differTimes(Teach@required,Study1@strong,Colloquy@medium,
            Study2@weak,Study3@weak)
```

Section 6.4.1 proposes an objective function representing requirement on minimization of conflicts between course sections of particular students. This objective function was shown as a special type of soft constraint (see Sect. 6.4.2) and we also define preferences on its variables which are all time variables included in the problem. They could express that course section should be scheduled in such a way that the number of conflicts with other course sections is minimal. This semantics leads us to the definition of annotation of time variable for course section related with the maximal possible number of student conflicts with other course sections (for course section c_i: $\sum_j \mathit{TwoConflict}(c_i, c_j)$). Such annotations are aimed to promote earlier assignment of time variables for course sections with possibly greater contribution to the value of objective function.

6.6 Faculty of Informatics Timetabling Problem

Faculty of Informatics timetabling problem represents large scale highly constrained scheduling problem as individual timetable for every student from more than thousand students has to be scheduled wrt. course pre-enrollment information. The number of prescribed courses is small, majority of courses are optional and so the sets of enrolled courses for each student can be very different. The probability of any two courses conflicting is about 30 % (which is equivalent to more than 10 000 constraints).

Having diverse requirements within course pre-enrollment, special section assignments has to be processed. Each course may consists from lecture or seminar or lecture+seminar, where the number of lectures and/or seminars is determined by the number of students pre-enrolled on a course, and with respect to teacher's requirements. Computing an ideal timetable should solve the problem of section assignment such that each student is able to visit one lecture and/or one seminar for each his (her) specified course. Each course section is then given by a tuple ⟨course, lecture or seminar identifier/order, set of students⟩. Basically our task consists in instantiation of the set of tuples ⟨course section, classroom, time⟩, i.e., each lecture or seminar has assigned its set of students, classroom, and time. Teachers of particular courses are determined as a part of our problem definition, so we don't need to solve this kind of resource allocation — assign teachers to courses.

Let us describe particular problem components in detail.

Section assignment assigns students to course sections, where an input of this problem are tuples ⟨number of lectures, number of seminars, students⟩ given for each course, and an output is a set of course sections defined by tuple ⟨course, lecture or seminar identifier, set of students⟩.

Classroom allocation for particular course sections while satisfying following hard and soft constraints

(C6.1) only one course section can take place in a classroom at a time;
(C6.2) course section must not change classroom during its duration;
(C6.3) each classroom has to have sufficient capacity wrt. expected number of students in course section;
(C6.4) each classroom has to have suitable equipment due to teacher's requirements (lecture hall, computer room, classroom with data projector);
(C6.5) exception — one teacher can teach two course sections in two specific classrooms at the same time either with use of camera and video projection or to ensure arbitrary exchange between computer room and lecture hall;
(C6.6) teachers may specify preferences on classroom selection (e.g., selection of classrooms in certain part of building).

Time assignment of starting times to course sections while minimizing total number of course sections which overlap for any student and satisfying following hard and soft constraints: (C6.1), (C6.5), and

(C6.7) each teacher gives only one course section at a time;
(C6.8) interruption of course section wrt. time is not allowed;
(C6.9) maximal number of time slices for teacher per day has to be respected;
(C6.10) requested time of course sections wrt. strict unavailability of teacher has to be respected;
(C6.11) various hard time dependencies are given between two or more course sections;
(C6.12) Friday afternoon is restrained;
(C6.13) early morning and late evening times are generally restrained;
(C6.14) special time preferences for course section are specified by teacher;
(C6.15) various soft time dependencies between two or more course sections are given by teachers.

Problem Size

With respect to growing size of our school we can use input data of different scope. Each instance is timetabled for 5 days with 13 teaching time slices. Each course section has variable duration from 1 to 4 time slices. Currently our implementation includes two different data sets with courses scheduled at the main faculty building. The number of our course sections is 234 and 269, resp. We have 93 (113) teachers, 12 (18) classrooms, 948 (1 255) pre-enrolled students[3], and 10 360 (13 289) student's requirements in course pre-enrollment, resp. Currently our problem definition includes 454 and 483 preferential requirements, resp. The probability of any two course sections conflicting is about 30 % (which is equivalent to 7 440 and 11 635 constraints, resp.).

[3] Actual number of students is slightly higher — about 1 000 and 1 400 students.

6.6.1 Solution Structure

Generally all sub-problems should be solved together to obtain such assignment of students, classrooms, and times to course sections that the hard constraints are satisfied, the objective function is minimized, and all soft constraints are satisfied to the largest possible extent. Wrt. the size of such general problem, we separated the most complex section assignment problem and constructed the feasible division of students as an input for other problems. This approach allows us to compute for every two course sections i, j a number *TwoConflict*(i, j) of students which want to visit both course sections. With this we may generate student-oriented schedules with help of objective function proposed in Sect. 6.4.

Following part of the section describes application of particular approaches proposed within this chapter to solve Faculty of Informatics timetabling problem. We concentrate on the structure and principles of the problem solution and refer to corresponding parts of this chapter including details on sub-problem solution. Representation of particular constraints was realized with help of built-in constraints applying standard approaches as summarized in Sect. 6.2.

Section Assignment. The output of section assignment problem should be some feasible splitting of students into particular course sections. Set of registered students for each course is basically sorted in lexicographic order and then split wrt. the required number of groups. The lexicographic order is advantageous wrt. the splitting of the same student group to lectures and to seminars, i.e., useless conflicts for seminars and lectures of the same course are not imposed.

Classroom Allocation. The basic requirement of the classroom allocation is prohibited change of classrooms during duration of each course section (C6.2). Such restriction does not allow to postpone decisions on assignment of classrooms after time assignment and we are required to apply solution which pre-assigns course section to set of classrooms of the same capacity (see Sect. 6.3.2). Specified equipment of classroom is included into problem definition via three types of cumulative resources corresponding to lecture halls, classrooms with data projector, and computer rooms. Within this initial classroom assignment, annotations are defined for particular constraints as discussed in Sect. 6.5. Labelling of initial classroom allocation applies standard backtracking with variable ordering computed by global variable annotations, and with value ordering reflecting requirements of soft constraint (C6.6). Exact allocation of identical classrooms is completed after time assignment as discussed in Sect 6.3.1.

Time Assignment. Each course section is represented by the one time variable reflecting constraint (C6.8). Basic representation of hard constraints is in correspondence with description in Sect. 6.2.

Both hard and soft constraints include annotations as it was shown by examples in Sect. 6.5 which also propose inclusion of soft constraints with help of annotations. The two possible instances of annotation framework are considered in our problem. Basically $\mathcal{A}_{vect}$ approach puts variables with teacher's requirements on higher level while student's interests or classroom properties are deferred. $\mathcal{A}_{sum}$ approach summarizes all annotations in one level.

Generation of student-oriented schedules is ensured with help of objective function we have proposed in Sect. 6.4.1. Incremental behavior of this function (Eqn. 6.3) allows its direct application within labelling which is processed via standard backtracking. Variable ordering is defined with help of global variable annotations, special value ordering is described in the following paragraph.

Generally, value ordering heuristics select the most promising values from more preferred values. By the promising value for time variable of course section i we mean those with their potential optimal contribution $SectionConflict(i)$ to the value of objective function $StudentsConflict_{inc}$. The preferred values may be defined for each variable separately (C6.14) or generally (C6.13)+(C6.12). Several levels of preferences (preferred values) may be given by soft constraints. The preferred values from the first level are selected in increasing order and current value of overlaps $SectionConflict(i)$ is compared with two kinds of thresholds. Absolute threshold is a maximal number of students in one course section for whom some overlapping with other course sections may occur and relative threshold is a ratio between the number of students with any overlap and the number of students pre-enrolled to the course section $NumberStudents(i)$

$$\text{absolute threshold:} \quad SectionConflict(i) \leq AbsThreshold; \tag{6.6}$$

$$\text{relative threshold:} \quad \frac{SectionConflict(i)}{NumberStudents(i)} \leq RelThreshold. \tag{6.7}$$

If $SectionConflict(i)$ is not sufficient the next remaining value is tried. Both thresholds may be also combined — if one of the thresholds is satisfied, value may be selected for instantiation. If values of current preference level are not sufficient to find feasible solution, values from lower level are tried in increasing order again. Possible faulty instantiations are undone by backtracking.

Values of objective function for different input data are compared via

$$Ratio = 1 - \frac{StudentsConflict_{inc}}{NbStudentRequirements} \tag{6.8}$$

with *NbStudentRequirements* as a number of student's requirements in course pre-enrollment. This gives us a percentage of satisfied students' requirements in course pre-enrollment.

6.6.2 Implementation & Computational Results

Described timetabling system was implemented in ILOG software [Pap94, Pug94], an object oriented library for constraint logic programming in C++. Computing the first solution takes 2–3, and 4–5 seconds on a Pentium II/400 MHz PC for the first and the second problem instance (particular data sets were described in Sect. 6.6), resp. Table 6.1 shows achieved results for particular data sets and annotation frameworks (general description of annotations may be found in Sect. 6.5; for further information about the concrete instances $\mathcal{A}_{sum}$ and $\mathcal{A}_{vect}$ see Sect. 6.6.1). The best achieved values of thresholds *AbsThreshold* (Eqn. 6.6) and *RelThreshold* (percentage from Eqn. 6.7) follow in the third and fourth columns. For the presented thresholds, generated solutions are evaluated also with help of the percentage of successfully satisfied soft constraints and with help of the value *Ratio* (percentage from Eqn. 6.8) giving us quality of solution wrt. the objective function.

$\mathcal{A}_{sum}$ framework computes better solution for absolute thresholds while $\mathcal{A}_{vect}$ annotations output more interesting results with relative thresholds. This is a consequence of higher

Data set	$\mathcal{A}$	*AbsThreshold*	*RelThreshold*	Soft constr.	*Ratio*
I.	$\mathcal{A}_{sum}$	4	–	92.7	95.4
I.	$\mathcal{A}_{vect}$	–	9	91.9	94.8
I.	$\mathcal{A}_{sum}$	3	6	90.3	96.4
I.	$\mathcal{A}_{vect}$	1	7	91.2	95.6
II.	$\mathcal{A}_{sum}$	8	–	95.0	92.1
II.	$\mathcal{A}_{vect}$	–	14	91.9	93.7
II.	$\mathcal{A}_{sum}$	3	13	90.7	94.5
II.	$\mathcal{A}_{vect}$	2	13	91.9	94.0

Table 6.1: Computational results for timetabling problem at FI

support of student's interests expressed by the number of students in $\mathcal{A}_{sum}$. While the large course sections (by the number of students) were satisfied to the high extent, importance of smaller course sections was deferred. If only one of the both thresholds have to be satisfied, results for $\mathcal{A}_{sum}$ and $\mathcal{A}_{vect}$ were comparable. Results also show that both optimization criteria are antagonistic — if student's conflicts are minimized, the quality of satisfaction of preferential constraints decreases, and vice versa.

6.7 Comparison with Other Approaches

Constraint Programming. Constraint programming was often applied for solving time assignment and classroom allocation problems as we have discussed in Sect. 6.2. Solution of section assignment sub-problem is included in [BvBM98] via iterative addition of constraints into a CSP representation. This implementation is able to find timetable satisfying 99 % student's requirements. However, this solution of high school timetabling problem doesn't include any additional requirements which are critical for solution of university-based timetabling problems. Their inclusion would be even problematic as authors implemented their own solution procedure as a CSP not applying constraint programming approach. To our knowledge, constraint programming was not applied for search of student-oriented schedules at all.

Other Solution Methods. To compare our results with other solutions, let us mention implementation of timetabling systems including other methods solving problems of comparable type.

Comprehensive university-sized timetabling system [Car00] may be characterized by problem decomposition wrt. both type and size of final sub-problems. They were able to solve the problem for 20 000 students with the largest problem component with 2 500 conflicts (inequalities) between course sections. On the other hand, our problem included more than 10 000 conflicts. With this comparison, it looks promisingly when our approach would be included as a component solver instead of applied greedy heuristics for time assignment.

Local search heuristic procedure [SFW95] solves the problem with comparable results wrt. satisfaction of course enrollment having smaller solution space with 85 course sections decomposed to part of weeks.

Aubin&Ferland [AF89] propose an iterative method to solve the problem which alternately assign times and students to course sections. However, proposal of special heuristic procedure for solution of this large scale system handling 3 300 students makes problematic extension of the problem definition by additional constraints.

Robert&Hertz [RH96] decompose the problem into a series of easier sub-problems corresponding to time, section, and classroom assignments and solve them via tabu search methods. Their problem solution allows addition of various side constraints but they need to be all included into one constraint function. Presented method is able to generate initial solution which can be incrementally improved after problem redefinition (negotiation on initial constraints with teachers and students). Achieved results looks promisingly even for initial solution without any negotiation as they were able to find schedules for about 500 students with approximately 10 % of unsatisfied student requirements. However, diversity of student requirements may not be so meaningful as both courses and students may be split into levels consisting of compulsory and optional courses. Possible extension of the problem size also remains open as computing of the solution takes several hours.

Examination Timetabling. Different kind of comparison may be obtained for solutions where constraint programming is applied to solve examination timetabling problem [CL96a, BDP96]. For this type of problem, section assignment problem plays also a substantial role as students must be distributed to their corresponding exams. Examination problem emphasizes side constraints as precedence constraints or spread constraints, objective functions may be also different (e.g., conflict-free timetable, examinations as early as possible), but the basic problem is the same — assign exams (courses) to a time slices in such a way that students are able to attend some sections of their exams (courses). Nature of the problem doesn't allow to consider variations of class-teacher oriented timetabling and as a consequence student-oriented schedules are constructed. The main disadvantage of solutions applying CLP methods is the search of conflict-free schedules for students [BDP96, LCKC00, BDP93, RO00] which may not be valid for course timetabling problem at all. Unfortunately the same disadvantage holds for all other implementations of examination problem solved via CSP mentioned in recent survey of Carter&Laporte [CL96a].

Chapter 7

Conclusion

Within this chapter we would like to mention possible directions of our future research, summarize the main contributions of this thesis, and give a global view to its particular parts.

7.1 Future Work

Let us study further directions of our research wrt. discussed problem areas within this thesis.

Hierarchies. Proposed SCSP classes for particular comparators of CH and their categorization based on properties of each instance allow further study of this framework from the point of view of general semiring-based and valued CSPs approaches. Showing correspondence with non-idempotent classes, last results from the area of algorithms [Sch00a, BGR00] for such classes can be taken into account also for CHs. We would like to study possible impact of this research on CHs and compare these results with extensive study in the area of algorithms for solving particular comparators of CHs (see [Bar97, Bar98] for surveys). Such research would be aimed towards a proposal of new algorithms for solving both comparators of CHs and general problem classes of meta-frameworks.

Local comparators defines non-idempotent SCSP class with partial ordering while the most studied instances include only total ordering [Sch00a]. Such conclusion opens new areas for study of the properties and algorithms for this type of classes.

Lexicographic-better comparator may become interesting as it eliminates contra-intuitive behavior of worst-case-better comparator and allows application of general SCSP approach. We intend to study possible algorithms for finding lexicographic-better solutions and further application of such comparator for solving real-life problems.

Annotations. We would like to study potential of annotations for solving problems with help of constraint-based methods with special intent devoted to over-constrained and constraint optimization problems where preferences play a critical role to define and find feasible solution.

We plan to propose and carry out experiments (e.g., on random CSP) with annotations as a source for computing variable ordering aimed to compare them with other methods and show their general usefulness which is currently demonstrated from the point of view of a practical application only.

Annotations expressing preferences of variables have shown to be interesting for declarative definition of global soft constraints. However, efficient algorithms taking into account over-constrainedness of the problem were studied for minor sub-problems only [BLPP98, Rég99]. Our future research will include extensions of constraint propagation algorithms for this type of constraints. We should mention especially `alldifferent` and `disjunctive` constraints which extension towards handling over-constrained problems would be helpful, e.g., in our timetabling application.

Timetabling. Our study of timetabling problem was oriented towards generation of schedules for individual students via constraint programming approach. Current solution may profit from application of constraint propagation methods for cost-based constraint relaxation applied in [AM00, AM98]. Based on our proposed instance of soft disjunctive scheduling problem we also plan to extend our solution strategy for student conflict minimization problem with help of the research in the area of algorithms for solving weighted and fuzzy CSPs [Sch00a, BGR00].

We intend to study further combination of classroom allocation and time assignment problems. It still remains open the possibility of a more interleaved solutions of individual subproblems which may substantially improve quality of generated solution. With increasing size of the problem, it may be more promising to study solution of separated subproblems while taking into consideration specialized solution methods from the areas of scheduling and resource allocation.

Further improvements of implemented timetabling system may include more sophisticated solutions of initial section assignment problem as it was proposed in [Car00] or take into account teacher assignment problem for courses with large number of equivalent courses sections.

Our solution approach was able to solve faculty-sized problem. For university-sized problems, it may become interesting to consider a proper problem decomposition as the one discussed in [Car00] and apply constraint-based approach to solve obtained sub-problems with help of the solver component, e.g., based on our implementation of Faculty of Informatics timetabling problem.

7.2 Contributions

Let us shortly summarize main contributions of the thesis.

- Unifying view to particular approaches for solving CSPs with preferences within theoretical part of the thesis.

- New ordered-better and lexicographic-better comparators for constraint hierarchies and proposal of an algorithm for ordered-better comparator [Rud98c].

- Categorization of comparators of constraint hierarchies over finite domains via proposed classes of semiring-based CSPs with help of a new equivalence over general SCSP classes.

- Proposal of a new framework for handling preferences over variables including approaches for fuzzy and hierarchical annotations [Rud98a, Rud99, Rud98b] and computing variable ordering [RM99b].
- Proposal of solution strategies for construction of student-oriented schedules applied for Faculty of Informatics timetabling problem [RM00, RM99a].

More detail description may be found as a part of the following section.

7.3 Summary & Discussion

We would like to give a complete overview of the thesis within this section by summarization and discussion of its contents.

After short introduction of CSP & CLP schemes in Chapter 2, we present theoretical foundations of thesis including description of particular frameworks for solving CSPs with preferences (Chapters 3–5). As we intended to give a unifying view to particular approaches, we have proposed a uniform structure which is followed by all frameworks. Each of them elucidates notions of constraint (natural extension of constraint definition by a specific preference), problem (how it is extended to handle given preferences), satisfaction degree (aimed to evaluate particular assignments), solution (defined with help of satisfaction degree via its optimal value), and consistency degree (up to which extent may be given problem satisfied).

Practical part of thesis contains the study of timetabling problem (Chapter 6) applying approaches for solving constraint satisfaction problems with preferences.

Frameworks. Chapter 3 gives a background of existing frameworks for solving CSPs with preferences. Starting from the basic frameworks over particular types of preferences (weighted, probabilistic, possibilistic, fuzzy CSPs), we continue to meta-frameworks (partial, valued, semiring-based CSPs). Basic frameworks may be obtained by specification of a general algebraic structure of meta-framework. Weighted CSP applies costs to optimize satisfaction of particular constraints (e.g., minimize sum of weights of unsatisfied constraints). Probabilities allow to express the degrees how much each constraint is valid in some ill-defined problem. Here we need to search for a definition of the real problem together with its best solution. Possibilistic CSP assigns a possibility degree to each constraint expressing how desirable its satisfaction is. This problem together with fuzzy CSP may be handled via min-max optimization. Fuzzy CSP associates the same type of preference to each tuple of values of constraint, however.

Partial CSP introduces the first attempt towards the generalization of basic approaches via problem relaxation. Later proposed valued CSP (VCSP) and semiring-based CSP (SCSP) build axiomatic theories which enable the construction of non trivial generic properties and algorithms. Particular formalisms define monoid and lattice structures to catch various preferences. While the monoid structure is able to handle only totally ordered preferences, SCSP with its lattice structure may represent partial ordering also. Multi-criteria optimization belongs to a typical representatives which require this property.

Even if SCSP is able to handle partial ordering of preferences, relation of equivalence between different SCSPs is not able to consider them. This led us to definition of a new equivalence and refinement which are able to compare SCSP classes with partial ordering of preferences, too (in Sect. 3.7.4). This definition remains in accordance with basic idea behind the refinement: if some problem is a refinement of another one, then its set of solutions is included in the set of solutions of the later problem.

Together with the comparison of meta-frameworks, final part of this chapter summarized relationships between particular basic frameworks which may be partitioned according to the idempotency of unifiable operator of both algebraic structures. Classical CSP, possibilistic and fuzzy CSP may be transformed via polynomial refinement into weighted CSP but the opposite transformation doesn't exist. It means that problem of finding optimal solution of weighted CSP can not be reduced to fuzzy CSP.

Constraint Hierarchies. Constraint hierarchies (CHs) and namely hierarchical constraint logic programming belong to traditional frameworks for handling of over-constrained problems. They allow to express hard constraints which has to be satisfied and several preference levels of soft constraints which violations are minimized level by level subsequently. CHs define the so called comparators aimed to select solutions (the best assignment of values to particular variables) via minimizing errors of violated constraints. Global comparators aggregate errors of violated constraints at each level, basically we may compare weighted sum of errors or error of worst satisfied constraint (weighted-sum-better and worst-case-better comparators). Local (locally-better) comparator considers each constraint individually, regional comparator is able to select among assignments by individual comparison of constraints at some lower level even if they are incomparable at higher levels.

Starting from Sect. 4.3, this chapter describes our contributions within the area of CHs. We have enhanced classical local comparator such that it is able to apply weights similar to those existing for weighted-sum-better comparator and proposed the so called ordered-better comparator [Rud98c]. We have studied its relationship with locally-better comparator — based on it and on existing algorithm for locally-better comparator we have proposed a new extended algorithm which is able to find ordered-better solution of CH. Our further study was concentrated on a more accurate proposal of the worst-case comparator which evokes following contra-intuitive behavior: violating a constraint with large weight we are not able to differ assignments violating or satisfying constraints with any smaller weight. This disadvantage may be eliminated with help of lexicographic ordering which led us to the definition of the lexicographic-better comparator.

Even if the CHs are extensively studied and diverse algorithms were discussed to solve particular comparators, any correspondence of CH's comparators with existing algebraic meta-frameworks wasn't shown with exclusion of vague statements about their relationship with enhanced versions of fuzzy CSPs:

> *Fuzzy CSPs are able to model prioritized constraints, that is, constraints with different levels of importance [BMMW89].* *Taken from [BMR97b].*

This opens not only a theory gap, but also practical problems which disable the applicability of any general algorithm already proposed. We have even shown that comparators of CHs either belong to more complex classes of SCSP or don't have counterpart in any SCSP class.

Our effort was devoted to the proposal of instances of particular algebraic structures which would capture most of the discussed comparators. Remaining of them are shown to be incompatible with some of the required properties of algebraic structures. As the second step, we have applied our extended definitions of equivalence and refinement between classes of SCSPs and we have classified all comparators of CHs which have an existing counterpart in some class of SCSP.

Basically we have constructed three classes of SCSP: weighted-SCSP corresponding to comparators derived from weighted-sum-better comparator, lexicographic-SCSP for lexicographic-better comparator, and local-SCSP for locally-better and ordered-better comparators. Worst-case-better comparator doesn't have its SCSP's counterpart due to above described contra-intuitive behavior — this makes lexicographic-better comparator even more important as its equivalent SCSP class does exist. The property which forbids a proposal of SCSP class for regional comparator is just its ability to compare assignments which are incomparable at higher levels.

Weighted-SCSP and lexicographic-SCSP are shown to be equivalent with the weighted CSP and lexicographic CSP via polynomial time refinement, resp. This relationship might be expected because all of them have non-idempotent operator and total ordering of preferences. The total ordering implies that all comparators dedicated to weighted-SCSP and lexicographic-SCSP classes may be defined as classes of VCSP. However, this doesn't hold for local-SCSP having partial ordering of preferences. This class may be polynomially transformed into the weighted CSP but the opposite transformation may not be constructed as the preferences within local-SCSP are partially ordered.

Variables' Annotations. Existing frameworks associate preferences either with constraints or with each tuple of values of constraint. An original contribution of the thesis concerns another still not investigated possibility, the idea of assigning preferences to particular variables in constraint. Such preferences may express different levels of importance for particular variables (e.g., time of events within temporal scheduling may be classified by participant's interest, order of job within jobs' sequence is influenced by owner's priority, prior placement of objects may depend on their properties).

We have proposed an annotation triple, a structure over annotations defining ordering over annotations, and function for combining particular annotations. Local annotations of variables within constraints are combined to compute global annotations of variables and constraints via specified combining function. Examples of its instances are discussed and applied for computing solution via fuzzy annotations, hierarchical annotations, and variable ordering [RM99b].

Fuzzy and hierarchical annotations [Rud98a, Rud99, Rud98b] present interpretation of variable's annotations based on existing frameworks with preferences, on fuzzy CSP and constraint hierarchies. For fuzzy annotations, global constraint annotations define preferences for selection of solutions via min-max optimization. A correspondence of constraint system with fuzzy annotation with both fuzzy and possibilistic CSPs is proven. For hierarchical annotations [Rud98c], annotated constraint hierarchy (ACH) is constructed with help of global annotations which define weights of particular constraints. Solution of ACH is computed by different methods inherited by weighted-sum, worst-case, and least-square principles. Another method is presented via individual comparison of particular global annotations at each level. Mapping of applied methods to particular comparators of CH is presented in accordance with selection method for global comparators. Solutions obtained via individual comparison are compatible with the proposed ordered-better comparator of CHs. Based on the algorithm for computing the ordered-better solution of CH, we have implemented a solver for systems of inequalities with annotations.

Another interpretation of annotations considers them as a source for computing variable ordering (VO) in CSPs with preferences. Several methods are proposed including different annotation triples and static versus dynamic VO. While static VO is computed with help of global annotations within initial solving step, dynamic VO recomputes global annotation each time a variable should be instantiated. Such global annotations take into account those variables only which remain to be free wrt. current partial assignment of variables in given computation step.

Timetabling. The final part of the thesis concerns practical application of constraint-based approach with preferences in the area of timetabling [RM00, RM99a]. Our intent was devoted to the course timetabling problem which is a typical representative of scheduling and resource allocation application where various type of preferences need to be involved to obtain any acceptable solution.

First we have introduced general problem considering particular courses, teachers, classrooms, and students as the basic objects. Having courses with too large number of students, they may be split to course sections. The main tasks to be solved are section assignment (assign students to particular course sections), classroom allocation (assign classrooms to course sections), and time assignment (assign time to course sections).

Subsequently our attention was devoted to current solution methods of the classroom allocation and time assignment problems via constrained-based approach. We have distinguished possible models of problem solution including classroom and time assignment variables wrt. problem properties. We have also studied methods for search of solution space wrt. preferences within the problem. Such search can be performed with no global optimization criteria via labelling heuristics or non-systematic constraint relaxation methods. Labelling heuristics influence solution search by an ordering of variables and values in their domains. Both orderings may reflect preferences within the problem. Constraint relaxation methods select some of the problem constraints to be violated. Here selection of violated constraints depends on their preferences. More sophisticated approaches define some objective function over preferences within the problem with aim to optimize its value. This definition often leads to the weighted CSP and relaxing the constraints wrt. their weights or costs.

Starting from Sect. 6.3, this chapter contains our works within the area of timetabling. First we have studied classroom allocation problem and proposed representation of problem with help of global constraints for distinguished problem instances. The most simple problem definition allows to allocate classrooms of the same capacity to courses such that all requirements towards classrooms can be specified over time variables and overall classroom allocation can be postponed after completing the time assignment. We have defined sufficient property of the problem together with its representation by global constraints to allow such separation of classroom allocation and time assignment also for classrooms of various capacities and types. To solve general problem instance, we partitioned classroom allocation to two steps which are processed before and after time assignments.

We have studied construction of student-oriented schedules representing problem which still was not solved via CP methodology. The first part defines an objective function over generated timetable minimizing the number of courses which student is not able to attend wrt. parallel schedule of his (her) other course(s). Incremental definition of approximation of this objective function allows its inclusion into labelling. The second part defines soft disjunctive scheduling problem (SDSP) which instance is also designated student conflict minimization problem. Solution of such SDSP is defined via weighted and fuzzy constraint satisfaction as the basic representative of different complexity classes of CSPs with preferences.

Preferences are also included into the problem with help of variables' annotations as overall problem is stated in variables having their own natural preferences. Annotations are derived from seniority of teachers (dean, professors, assistants, ...), type of classrooms, or interest of students expressed by their requirements in course pre-enrollment. These annotations are applied via variable ordering methods. Examples of hard and soft constraints with annotations are described finally.

Proposed methods are applied for Faculty of Informatics timetabling problem which solution was implemented in CLP library of ILOG Scheduler. Overall problem is characterized by requirements of scheduling courses for individual students (more than 1 000 students) and taking into account preferential requirements of teachers towards any acceptable timetable. Achieved computational results conclude this part, within them the most interesting one is a quality of generated timetable satisfying more than 94 % students' requirements.

Finally we have compared our solution methods and achieved results with other approaches and solutions of comparable problem instances. We have shown that our problem solution is comparable with other solution methods while preserving advantages of declarative definition and easier extension of the problem definition.

Bibliography

[AB91] Abderrahmane Aggoun and Nicolas Beldiceanu. Overview of the CHIP compiler system. In Koichi Furukawa, editor, *Logic Programming, Proceedings of the Eighth International Conference*, pages 775–789. MIT Press, 1991.

[AB93] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993.

[AB94] Francisco Azevedo and Pedro Manuel Barahona. Timetabling in constraint logic programming. In *Proceedings of the 2nd World Congress on Expert Systems*, 1994.

[AF89] Jean Aubin and Jacques A. Ferland. A large scale timetabling problem. *Computers & Operations Research*, 16(1):67–77, 1989.

[AM98] Slim Abdennadher and Michael Marte. University timetabling using constraint handling rules. In *Actes des Journées Francophones de Programmation en Logique et Programmation par Contraintes*, pages 39–49, 1998.

[AM00] Slim Abdennadher and Michael Marte. University course timetabling using constraint handling rules. *Journal of Applied Artificial Intelligence*, 14(4):311–326, 2000.

[AS99] Slim Abdennadher and Hans Schlenker. Nurse scheduling using constraint logic programming. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-99); Proceedings of the 11th Conference on Innovative Applications of Artificial Intelligence*, pages 838–843, Menlo Park, Cal., 1999. AAAI/MIT Press.

[ASW00] Slim Abdennadher, Matthias Saft, and Sebastian Will. Classroom assignment using constraint logic programming. In *Proceedings of the Practical Application of Constraint Technology and Logic Programming*, pages 179–191. The Practical Application Company Ltd, 2000.

[BAFB96a] Alan Borning, Richard Anderson, and Bjorn Freeman-Benson. Indigo: A local propagation algorithm for inequality constraints. In *Proceedings of the 1996 ACM Symposium on User Interface Software and Technology*, pages 129–136, 1996.

[BAFB96b] Alan Borning, Richard Anderson, and Bjorn Freeman-Benson. The Indigo algorithm. Technical Report TR-96-05-01, Department of Computer Science and Engineering University of Washington, July 1996.

[Bap98] Philippe Baptiste. *A Theoretical and Experimental Study of Resource Constraint Propagation*. PhD thesis, University of Compiègne, 1998.

[Bar97] Roman Barták. *Expertní systémy založené na omezujících podmínkách*. PhD thesis, Katedra teoretické informatiky, Matematicko-fyzikální fakulta Univerzity Karlovy, 1997.

[Bar98] Roman Barták. On-line guide to constraint programming. In `http://kti.mff.cuni.cz/~abartak/constraints`, 1998.

[Bar99a] Roman Barták. Constraint programming: a survey of solving technology. *AIROnews*, IV(4):7–11, 1999.

[Bar99b] Roman Barták. Constraint programming: In pursuit of the holy grail. In *Proceedings of Workshop of Doctoral Students'99*, 1999.

[BC98] Edmund Burke and Michael Carter, editors. *Practice and Theory of Automated Timetabling II*. Springer-Verlag LNCS 1408, 1998.

[BDP93] Patrice Boizumault, Yan Delon, and Laurent Péridy. Solving a real-life planning exams problem using constraint logic programming. In Manfred Meyer, editor, *Constraint Processing: Proceedings of the International Workshop at CSAM'93*, Research Report RR-93-39, pages 107–112, DFKI Kaiserslautern, August 1993.

[BDP96] Patrice Boizumault, Yan Delon, and Laurent Péridy. Constraint logic programming for examination timetabling. *Journal of Logic Programming*, 26(2):217–233, 1996.

[BE00] Edmund Burke and Wilhelm Erben, editors. *PATAT 2000 — Proceedings of the 3rd international conference on the Practice And Theory of Automated Timetabling*. Fachhochschule Konstanz, University of Applied Science, Germany, 2000.

[Ben95] Frédéric Benhamou. Interval constraint logic programming. In Andreas Podelski, editor, *Constraint Programming: Basics and Trends*, pages 1–21. Springer-Verlag LNCS 910, 1995.

[BFBW92] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992.

[BFM+96] Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gérard Verfaillie. Semiring-based CSPs and Valued CSPs: Basic properties and comparison. In Michael Jampel, Eugene Freuder, and Michael Maher, editors, *Over-Constrained Systems*, pages 111–150. Springer-Verlag LNCS 1106, August 1996.

[BFM+99] Stefano Bistarelli, Helene Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gerard Verfaillie. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *CONSTRAINTS: An international journal*, 4(3), 1999.

[BGJ94] Patrice Boizumault, Christelle Guéret, and Narendra Jussien. Efficient labelling and constraint relaxation for solving time tabling problems. In Pierre Lim and Jean Jourdan, editors, *Proceeding of the 1994 ILPS post-conference workshop on Constraint Languages/Systems and their Use in Problem Modelling*, volume 1 (Application and Modelling), pages 116–130, November 1994.

[BGR00] Stephano Bistarelli, Rosella Gennari, and Francesca Rossi. Constraint propagation for soft constraint satisfaction problems: Generalization and termination conditions. In *Principles and Practice of Constraint Programming — CP'00*, pages 83–97. Springer-Verlag LNCS 1894, 2000.

[BKMQC97] Michael Baumgart, Hans Peter Kunz, Sascha Meyer, and Klaus Quibeldey-Cirkel. Priority-driven constraints used for scheduling at univesities. In Mark Wallace, editor, *Practical Application of Constraint Technology*, pages 65–73. The Practical Application Company Ltd, 1997.

[BLP95] Philippe Baptiste and Claude Le Pape. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 600–606, San Mateo, 1995. Morgan Kaufmann.

[BLP96] Philippe Baptiste and Claude Le Pape. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group*, 1996.

[BLPP98] Philippe Baptiste, Claude Le Pape, and Laurent Péridy. Global constraints for partials CSPs: A case-study of resource and due date constraints. In Michael Maher and Jean-François Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 87–101. Springer-Verlag LNCS 1520, 1998.

[BM95] Smith Barbara M.˙ A tutorial on constraint programming. Technical Report 95.14, University of Leeds, School of Computer Studies, 1995.

[BMMW89] Alan Borning, Michael Maher, Amy Martindale, and Molly Wilson. Constraint hierarchies and logic programming. In Giorgio Levi and Maurizio Martelli, editors, *ICLP'89: Proceedings 6th International Conference on Logic Programming*, pages 149–164, Lisbon, Portugal, June 1989. MIT Press.

[BMR95] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Constraint solving over semirings. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 624–630. Morgan Kaufmann, 1995.

[BMR97a] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint logic programming. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 352–357. Morgan Kaufmann, 1997.

[BMR97b] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, March 1997.

[BR96] Edmund Burke and Peter Ross, editors. *Practice and Theory of Automated Timetabling*. Springer-Verlag LNCS 1153, 1996.

[Bru98] Peter Brucker. *Scheduling Algorithms*. Springer Verlag, 1998.

[BvBM98] Don Banks, Peter van Beek, and Amnon Meisels. A heuristic incremental modeling approach to course timetabling. In *Proceedings of Artificial Intelligence'98, Canada*, 1998.

[Car00] Michael W. Carter. A comprihensive course timetabling and student scheduling system at the University of Waterloo. In Edmund Burke and Wilhelm Erben, editors, *PATAT 2000 — Proceedings of the 3rd international conference on the Practice And Theory of Automated Timetabling*, pages 48–59, 2000.

[CKLW96] Czarina Cheng, Le Kang, Norrus Leung, and George M. White. Investigations of a constraint logic programming approach to university timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 112–129. Springer-Verlag LNCS 1153, 1996.

[CL94] Yves Caseau and François Laburthe. Improved CLP scheduling with task intervals. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conference Logic Programming*, pages 369–383. MIT Press, 1994.

[CL96a] Michael W. Carter and Gilbert Laporte. Recent developments in practical examination timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 3–21. Springer-Verlag LNCS 1153, 1996.

[CL96b] Yves Caseau and François Laburthe. Cumulative scheduling with task intervals. In Michael Maher, editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 363–377. MIT Press, 1996.

[CL98a] Michael W. Carter and Gilbert Laporte. Resent developments in practical course timetabling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, pages 3–19. Springer-Verlag LNCS 1408, 1998.

[CL98b] C. K. Chiu and Jimmy Ho-man Lee. Extending HCLP with partially ordered hierarchies and composite constraints. *Journal of Experimental and Theoretical Artificial Intelligence*, 10:5–24, 1998.

[COC97] Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programming*. Springer-Verlag LNCS 1292, 1997.

[DFP94] Didier Dubois, Hélène Fargier, and Henri Prade. Propagation and satisfaction of flexible constraints. In *Fuzzy Sets, Neural Networks and Soft Computing*, pages 166–187. Van Nostrand Reinhold, New York, 1994.

[DFP96] Didier Dubois, Hélène Fargier, and Henri Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[DP93] Dedier Dubois and Henri Prade, editors. *Readings in Fuzzy Sets for Intelligent Systems*. Morgan Kaufmann, 1993.

[F^+93] Thom Frühwirth et al. Constraint logic programming – an informal introduction. Technical Report ECRC-93-5, ECRC, 1993.

[FB98] Thom Früwirth and Pascal Brisset. Optimal placement of base stations in wireless indoor telecommunication. In Michael Maher and Jean-François Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 476–480. Springer-Verlag LNCS 1520, 1998.

[FHS95] Harikleia Frangouli, Vassilis Harmandas, and Panagiotis Stamatopoulos. UTSE: Construction of optimum timetables for university courses — A CLP based approach. In *Proceedings of the Third International Conference on the Practical Applications of Prolog (PAP'95)*, pages 225–243, Paris, France, April 1995. Alinmead Software Ltd.

[FL93] Hélène Fargier and Jérôme Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty, European Conference ECSQARU'93*, pages 97–104. Springer-Verlag LNCS 747, 1993.

[FLS93] Hélène Fargier, Jérôme Lang, and Thomas Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *EUFIT'93, First European Congress on Fuzzy and Intelligent Technologies*, volume 3, pages 1128–1134. Augustinus Buchhandlung, 1993.

[Fre78] Eugene C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21(11):958–966, 1978.

[FW92] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[GHR93] Dov M. Gabbay, C.J. Hogger, and J.A. Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1 Logical Foundations. Clarendon Press, Oxford, 1993.

[GJBP96] Christelle Guéret, Narendra Jussien, Patrice Boizumault, and Christian Prins. Building university timetables using constraint logic programming. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 130–145. Springer-Verlag LNCS 1153, 1996.

[GKM98] Hans-Joachim Goltz, Georg Küchler, and Dirk Matzke. Constraint-based timetabling for universities. In *Proceedings INAP'98, 11th International Conference on Applications of Prolog*, pages 75–80, 1998.

[GM99] Hans-Joachim Goltz and Dirk Matzke. Combined interactive and automatic timetabling. In *Proceedings of the Practical Application of Constraint Technology and Logic Programming*, pages 529–535. The Practical Application Company Ltd, 1999.

[HMY96] Hiroshi Hosobe, Satoshi Matsuoka, and Akinori Yonezawa. Generalized local propagation: A framework for solving constraint hierarchies. In Eugene C. Freuder, editor, *Principles and Practice of Constraint Programming – CP96*, pages 237–251. Springer-Verlag LNCS 1118, 1996.

[HW96] Martin Henz and Jörg Würtz. Using Oz for college timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 162–177. Springer-Verlag LNCS 1153, 1996.

[Int00] Intelligent Systems Laboratory, Swedish Institute of Computer Science. *SICStus Prolog User's Manual*, 2000.

[Jam95] Michael Jampel. A compositional theory of constraint hierarchies (operational semantics). In Michael Jampel, Eugene Freduer, and Michael Maher, editors, *Procceding of Workshop on Over-Constrained Systems at CP'95*, 1995.

[Jam96] Michael Benjamin Jampel. *Over-Constrained Systems in CLP and CSP*. PhD thesis, Department of Computer Science City University, September 1996.

[JJGH96] Michael Jampel, Jean-Marie Jacquet, David Gilbert, and Sebastian Hunt. Transformation between HCLP and PCSP. In Eugene C. Freuder, editor, *Principles and Practice of Constraint Programming'– CP96*, pages 237–251. Springer-Verlag LNCS 1118, 1996.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *14th Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19:503–581, 1994.

[KB99] Ludwig Krippahl and Pedro Barahona. Applying constraint programming to protein structure determination. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming — CP'99*, pages 289–302. Springer-Verlag LNCS 1713, 1999.

[KL98] R. W. L. Kam and Jimmy Ho-man Lee. Fuzzifying the constraint hierarchies framework. In Michael Maher and Jean-François Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 280–294. Springer-Verlag LNCS 1520, 1998.

[Kum92] Vipin Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[KYN99] Kazuya Kaneko, Masazumi Yoshikawa, and Yoichiro Nakakuki. Improving a heuristic repair method for large-scale school timetabling problems. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming — CP'99*, pages 275–288. Springer-Verlag LNCS 1713, 1999.

[Laj96] Gyuri Lajos. Complete university modular timetabling using constraint logic programming. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 146–161. Springer-Verlag LNCS 1153, 1996.

[LCKC00] Andrew Lim, Ang Juay Chin, Ho Wee Kit, and Oon Wee Chong. A campus-wide university examination timetabling application. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-00); Proceedings of the 12th*

Conference on Innovative Applications of Artificial Intelligence, pages 1020–1025, Menlo Park, Cal., 2000. AAAI/MIT Press.

[LD86] Gilbert Laporte and Sylvain Desroches. The problem of assigning students to course sections in a large engineering school. *Computers & Operations Research*, 13(4):387–394, 1986.

[LM82] Jiří Likeš and Josef Machek. *Počet pravděpodobnosti*. Nakladatelství technické literatury, 1982.

[LV97] M. Lemaître and G. Verfaillie. Daily management of an earth observation satellite : comparison of ILOG solver with dedicated algorithms for Valued constraint satisfaction problems. In *Proceedings of Third ILOG International Users Meeting*, Paris, France, July 1997.

[Mac77] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[Mat93] Luděk Matyska. Constraint logic programming, an overview. In *SOFSEM'93*, pages 133–164, 1993.

[MCF98] Kim Marriott, Sitt Sen Chok, and Alan Finlay. A tableau based constraint solving toolkit for interactive graphical applications. In Michael Maher and Jean-François Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 340–354. Springer-Verlag LNCS 1520, 1998.

[MG81] E. H. Mamdani and B. R. Ganies, editors. *Fuzzy reasoning and its applications*. Academic Press, 1981.

[MJPL92] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[Mon74] Ugo Monatanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7(2):95–132, 1974. Also Tech.Rep., Carnegie Mellon University, 1970.

[MS98] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.

[MW88] David Maier and David S. Warren. *Computing with Logic, Logic Programming with Prolog*. The Benjamin/Cummings Publishing Company, Inc., 1988.

[NA96] Wim P. M. Nuijten and Emile H. L. Aarts. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996.

[Nov86] Vilém Novák. *Fuzzy množiny a jejich aplikace*. Státní nakladatelství technické literatury, 1986.

[Nui94] Wilhelmus P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Eindhoven University of Technology, 1994.

[O'K90] Richard A. O'Keefe. *The Craft of Prolog*. MIT Press, 1990.

[Pap94] Claude Le Pape. Implementation of resource constraints in ILOG SCHEDULE: a library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.

[PB98] Claude La Pape and Phillippe Baptiste. Constraint-based scheduling: a theoretical comparison of resource constraint propagation rules. In Wim Nuijten Jean-Charle Régin, editor, *ECAI-98 Workshop on Non Binary Constraints*, pages 1–12, 1998.

[Pug94] Jean-François Puget. A C++ implementation of CLP. Technical report, ILOG S.A., 1994.

[Pug98a] Jean-François Puget. Constraint programming: A great AI success. In Henri Prade, editor, *13th European Conference on Artificial Intelligence Proceedings*, pages 698–705. John Wiley & Sons, Ltd., 1998.

[Pug98b] Jean-François Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 359–366, Menlo Park, 1998. AAAI Press.

[Rég94] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Eleventh National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, volume 1, pages 362–367, Menlo Park, 1994. AAAI Press/MIT Press.

[Rég99] Jean-Charles Régin. Arc consistency for global cardinality constraints with costs. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming — CP'99*, pages 390–404. Springer-Verlag LNCS 1713, 1999.

[RH96] Vincent Robert and Alain Hertz. How to decompose constrained course scheduling problems into easier assignment type subproblems. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 364–373. Springer-Verlag LNCS 1153, 1996.

[RM99a] Hana Rudová and Luděk Matyska. Timetabling with annotations. Technical Report FIMU-RS-99-09, Faculty of Informatics Masaryk University, 1999.

[RM99b] Hana Rudová and Luděk Matyska. Uniform framework for solving overconstrained and optimization problems. In *CP'99 Post-Conference Workshop on Modelling and Solving Soft Constraints*, 1999.

[RM00] Hana Rudová and Luděk Matyska. Constraint-based timetabling with student schedules. In Edmund Burke and Wilhelm Erben, editors, *PATAT 2000 — Proceedings of the 3rd international conference on the Practice And Theory of Automated Timetabling*, pages 109–123, 2000.

[RO00] Luís Paulo Reis and Eugénio Oliveira. Examination timetabling using set variables. In Edmund Burke and Wilhelm Erben, editors, *PATAT 2000 — Proceedings of the 3rd international conference on the Practice And Theory of Automated Timetabling*, pages 181–183, 2000.

[Rud98a] Hana Rudová. Constraints with variables' annotations. In Henri Prade, editor, *13th European Conference on Artificial Intelligence Proceedings*, pages 261–262. John Wiley & Sons, Ltd., 1998.

[Rud98b] Hana Rudová. Constraints with variables' annotations. Technical Report FIMU-RS-98-04, Faculty of Informatics Masaryk University, 1998.

[Rud98c] Hana Rudová. Constraints with variables' annotations and constraint hierarchies. In Branislav Rovan, editor, *SOFSEM'98: Theory and Practice of Informatics*, pages 409–418. Springer-Verlag LNCS 1521, 1998.

[Rud99] Hana Rudová. Over-constrained systems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference*, page 954. AAAI Press/MIT Press, 1999. Presented in 1999 SIGART/AAAI Doctoral Consortium.

[Rut94] Zsófia Ruttkay. Fuzzy constraint satisfaction. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1263–1268. IEEE Press, 1994.

[Rut98] Zsófia Ruttkay. Constraint satisfaction — a survey. *CWI Quaterly*, 11(2&3):123–161, 1998.

[SA91] Ken Satoh and Akira Aiba. The hierarchical constraint logic language CHAL. Technical Report TR-0592, Institute for New Generation Computer Technology, Tokyo, 1991.

[Sat90] Ken Satoh. Formalizing soft constraints by interpretation ordering. Technical Report TR-0513, Institute for New Generation Computer Technology, Tokyo, May 1990.

[Sch92] Thomas Schiex. Possibilistic constraint satisfaction problems or "How to handle soft constraints ?". In *8^{th} International Conference on Uncertainty in Artificial Intelligence*, pages 268–275, Stanford, CA, July 1992.

[Sch95] Andrea Schaerf. A survey of automated timetabling. Technical Report CS-R9567, CWI, Amsterdam, NL, 1995.

[Sch00a] Thomas Schiex. Arc consistency for soft constraints. In *Principles and Practice of Constraint Programming — CP'00*, pages 411–424. Springer-Verlag LNCS 1894, 2000.

[Sch00b] Thomas Schiex. Valued CSPs. In *CP'99 Post-Conference Workshop on Soft Constraints: Theory and Practice*, 2000.

[SFV95] Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–639, San Mateo, August 1995. Morgan Kaufmann.

[SFW95] Scott E. Sampson, James R. Freeland, and Elliot N. Weiss. Class scheduling to maximize participant satisfaction. *Interfaces*, 25(3):30–41, 1995.

[SH81] Linda G. Shapiro and Robert M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions Pattern Analysis Machine Intelligence*, 3:504–519, 1981.

[Sim99] Helmut Simonis. Building industrial applications. In *International Summer School on Constraints in Computational Logics*. Esprit Working Group CCL, 1999.

[Smo95] Gert Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, pages 324–343. Springer-Verlag LNCS 1000, Berlin, 1995.

[VH89] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

[VLS96] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search for solving constraint optimization problems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96) and Eighth Conference on Innovative Applications of Artificial Intelligence (IAAI-96)*, pages 181–187, Portland, OR, USA, 1996.

[Š87] Josef Štěpán. *Teorie pravděpodobnosti : Matematické základy*. Československá akademie věd, Academia, 1987.

[Wal92] Mark Wallace. Applying constraints for scheduling. In Brian Mayoh, Enn Tyugu, and Jaan Penjam, editors, *Constraint Programming*, volume F: Computer and Systems Sciences 131 of *NATO Advanced Science Institut Series*, pages 153–171. Springer-Verlag, 1992.

[WB89] Molly Wilson and Alan Borning. Extending hierarchical constraint logic programming: Nonmonotonicity and inter-hierarchy comparison. In Ewing Lusk and Ross Overbeek, editors, *NACLP'89: Proceedings North American Conference on Logic Programming*, pages 3–19, Cleveland, Ohio, October 1989. MIT Press.

[WB93] Molly Wilson and Alan Borning. Hierarchical constraint logic programming. *Journal of Logic Programming*, 16(3,4):277–318, 1993.

[Wil93] Molly Ann Wilson. *Hierachical Constraint Logic Programming*. PhD thesis, Dept. of Computer Science and Engeneering University of Washington, May 1993.

[WNS97] Mark Wallace, Stefano Novello, and Joachim Schimpf. ECL^{i}PS^e: A platform for constraint logic programming. Technical report, IC-Parc, Imperial College, 1997.

[Wre96] Anthony Wren. Scheduling, timetabling and rostering – a special relationship? In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 46–75. Springer-Verlag LNCS 1153, 1996.

[WZ98] George M. White and Junhan Zhang. Generating complete university timetables by combining tabu search with constraint logic. In Edmund Burke and

Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, pages 187–198. Springer-Verlag LNCS 1408, 1998.

[YKNW94] Masazumi Yoshikawa, Kazuya Kaneko, Yuriko Nomura, and Masanobu Watanabe. A constraint-based approach to high-school timetabling problems: A case study. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference*, pages 1111–1116. AAAI Press/MIT Press, 1994.

Appendix A

Multi-sets

Multi-sets are extended sets with allowed repetition of elements. Set of all multi-sets over elements belonging to set W may be be defined with help of a relation W *to* $\mathbb{N}$. Such relation will be denoted $\mathbb{N}^W$ in the rest[1].

Let us define *cardinality* $\mathrm{card}(x, M)$ of element x in a multiset M giving the number of repetition of element x in the multi-set M. $\mathrm{card}(x, M) = 0$ holds if x doesn't occur in M.

A *union* $\sqcup$ of multisets M_1, M_2 has to satisfy following property

$$[M = M_1 \sqcup M_2] \equiv [\forall x \in M : \mathrm{card}(x, M) = \mathrm{card}(x, M_1) + \mathrm{card}(x, M_2)] \ . \qquad \text{(A.1)}$$

A multiset M_1 is *included* in a multiset M_2 iff each element of M_1 occurs in M_2 with the same or greater cardinality

$$[M_1 \sqsubseteq M_2] \equiv [\forall x \in M_1 : \mathrm{card}(x, M_1) \leq \mathrm{card}(x, M_2)] \ . \qquad \text{(A.2)}$$

As multi-set inclusion defines an ordering we say that M_1 is *minimum* $\min_\diamond$ from multi-sets M_1, M_2 while M_2 is their *maximum* $\max_\diamond$

$$[M_1 \sqsubseteq M_2] \equiv [M_1 = \min_\diamond(M_1, M_2) \ \vee \ M_2 = \max_\diamond(M_1, M_2)] \ . \qquad \text{(A.3)}$$

Another (lexicographic) ordering over multi-sets is induced by the totally ordered set W of elements in multi-set. A *lexicographic inclusion* $\sqsubset_{lex}$ of multi-sets M_1, M_2 is defined by

$$\begin{aligned}[M_1 \sqsubset_{lex} M_2] \ \equiv \ & [w_1 = \max_{x \in M_1} x, w_2 = \max_{x \in M_2} x : \\ & (w_1 <_W w_2) \vee ((w_1 =_W w_2) \wedge ((M_1 - \{w_1\}) \sqsubset_{lex} (M_2 - \{w_2\})))] \end{aligned} \qquad \text{(A.4)}$$

The recursion ends to $\emptyset$ which is the minimal multi-set. Let us note that this ordering is total while classical inclusion $\sqsubseteq$ is a partial ordering only.

Lexicographic minimum and maximum are defined by

$$[M_1 \sqsubseteq_{lex} M_2] \equiv [M_1 = \min_{lex}(M_1, M_2) \ \vee \ M_2 = \max_{lex}(M_1, M_2)] \ . \qquad \text{(A.5)}$$

[1]As an example let us recall the notion of set of all subsets of A which is a relation A *to* 2 denoted 2^A.

Abstract

Various types of preferences were proposed to find solutions of over-constrained problems, optimization problems, problems with uncertainties, or ill-defined problems where some kind of softness have to be involved to get feasible solutions. Studying these problems from the point of view of constraint satisfaction, we concentrate on description of constraint satisfaction problems (CSPs) with preferences. An original contribution consists in assigning preferences to particular variables in constraint which lead to study of constraints with the so called variables' annotations. Local annotations of variables in constraints are introduced as a method for computing static and dynamic variable ordering in CSPs with preferences. Interpretations based on existing frameworks with preferences, on fuzzy CSP and constraint hierarchies, use fuzzy and hierarchical annotations. Our intent is also devoted to constraint hierarchies (CHs) having several preference levels of constraints and defining solutions via the so called comparators. New ordered-better and lexicographic-better comparators are proposed and their comparison with traditional comparators of CHs is discussed. Subsequently an algorithm for solving CH with ordered-better comparator is presented. Certain comparators of CHs are introduced as instances of general framework for CSPs with preferences, semiring-based CSP, and classified into complexity classes with help of a new equivalence for semiring-based CSPs. It is also shown that any correspondence does not exist for remaining comparators. Practical application of constraint-based approaches with preferences is described for timetabling problem having special emphasis on variables' annotations. Concentration is devoted to construction of student-oriented schedules representing problem which still was not solved via constraint programming methodology. Proposed methods are applied for solving Faculty of Informatics timetabling problem including construction of individual student schedules for more than 1 000 students. Achieved results are compared with other solution methods and solutions of comparable problem instances.