Contents

This chapter is to appear as

W.-J. van Hoeve. Over-Constrained Problems. Chapter 6 of P. Van Hentenryck and M. Milano (eds.), *Hybrid Optimization: the 10 years of CPAIOR*, Springer.

Chapter 6
Over-Constrained Problems

Willem-Jan van Hoeve

Abstract Over-constrained problems are ubiquitous in real-world applications. In constraint programming, over-constrained problems can be modeled and solved using *soft constraints*. Soft constraints, as opposed to hard constraints, are allowed to be violated, and the goal is to find a solution that minimizes the total amount of violation. In this chapter, an overview of recent developments in solution methods for over-constrained problems using constraint programming is presented, with an emphasis on soft global constraints.

6.1 Introduction

In the context of constraint programming, combinatorial optimization problems are modeled using variables and constraints over subsets of these variables. When the constraints in a model do not allow any solution to the problem, we say that the problem is *over-constrained*. Unfortunately, most combinatorial problems found in real-world applications are essentially over-constrained. Practitioners typically circumvent this inherent difficulty when *modeling* the problem, by ignoring certain aspects of the problem. The resulting model, that hopefully allows a solution, then serves as a relaxation of the original problem.

Instead of removing constraints, one may wish to slightly modify (some of) the constraints, thereby maintaining a model that is as close as possible to the original problem description. A natural way to modify constraints in an over-constrained setting is to allow some constraints to be (partly) violated. In constraint programming, constraints that are allowed to be violated are called *soft constraints*. Solving the original problem then amounts to finding a solution that minimizes the overall cost

Willem-Jan van Hoeve
Tepper School of Business, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA
e-mail: vanhoeve@andrew.cmu.edu

of violation, or to optimize the original objective function given a threshold value on the total amount of violation that is acceptable.

This chapter gives an overview of techniques to handle over-constrained problems in the context of constraint programming. Following the nature of this collection, the focus will be on recent developments that are most relevant to CPAIOR, over (roughly) the last ten years. Interestingly, in 1998, the first paper appeared that marked the start of the recent research efforts that will be discussed in this chapter; that of *soft global constraints*.

6.1.1 A brief historical overview

We start by presenting a brief overview of soft constraints and over-constrained problems in constraint programming. The most influential early works on soft constraints are the framework for Constraint Hierarchies by Borning, Duisberg, Freeman-Benson, Kramer, and Woolf [1987], and the Partial-CSP framework by Freuder and Wallace [1992]. The latter includes the *Max-CSP* framework that aims to maximize the number of satisfied constraints. Since in this framework each constraint is either violated or satisfied, the objective is equivalent to minimizing the number of violated constraints. It has been extended to the *Weighted CSP* framework by Larrosa [2002] and Larrosa and Schiex [2003], associating a degree of violation (not just a Boolean value) to each constraint and minimizing the sum of all weighted violations. The *Possibilistic-CSP* framework in [Schiex, 1992] associates a preference to each constraint (a real value between 0 and 1) representing its importance. The objective of the framework is the hierarchical satisfaction of the most important constraints, i.e., the minimization of the highest preference level for a violated constraint. The *Fuzzy-CSP* framework in [Dubois et al., 1993], [Fargier et al., 1993] and [Ruttkay, 1994] is somewhat similar to the Possibilistic-CSP but here a preference is associated to each tuple of each constraint. A preference value of 0 means the constraint is highly violated and 1 stands for satisfaction. The objective is the maximization of the smallest preference value induced by a variable assignment. The last two frameworks are different from the previous ones since the aggregation operator is a min/max function instead of addition. With valued-CSPs [Schiex et al., 1995] and semi-rings [Bistarelli et al., 1997] it is possible to encode Max-CSP, weighted CSPs, Fuzzy CSPs, and Possibilistic CSPs.

Even though the above approaches allow to model a wide range of over-constrained problems, certain aspects arising in practical problems cannot be represented, as argued by Petit, Régin, and Bessière [2000]. First, it is important to distinguish hard constraints that must always be satisfied (for example due to physical restrictions) and soft constraints, that are allowed to be violated. All above frameworks, except for Max-CSP, allow to model this distinction. However, in most practical problems, not all soft constraints are equally important. Instead, they are usually subject to certain rules, such as "if constraint c_1 is violated, then c_2 cannot be violated", or "if constraint c_3 is violated, a new constraint c_4 becomes active". Rules of this nature cannot be modeled using the above frameworks, which was one of the main motivations to introduce the *meta-constraint* framework by Petit, Régin, and Bessière [2000]; see also Petit [2002]. In this framework, a cost variable is associated to each soft constraint, representing the degree of violation for that constraint. If the cost variable is 0, the constraint is satisfied. By posting meta-constraints on these cost variables, we can easily model additional rules and preferences among the soft constraints. For example, if z_i represents the cost variable of soft constraint c_i (for $i = 1,2,3,4$), the above rules can be modeled as $(z_1 > 0) \rightarrow (z_2 = 0)$, and $(z_3 > 0) \rightarrow c_4$, respectively. In addition, Petit, Régin, and Bessière [2000] show that the meta-constraint framework can be used to model the Max-CSP, Weighted CSP, Possibilistic CSP, and Fuzzy CSP frameworks in a straightforward manner.

An important aspect of the meta-constraint framework is that it allows to propagate information from one (soft) constraint to the other through the domains of the cost variables using domain filtering algorithms. This can be done even for *global constraints* (and soft global constraints) that encapsulate a particular combinatorial structure on an arbitrary number of variables. The first such filtering algorithm was given by Baptiste, Le Pape, and Péridy [1998], while Petit, Régin, and Bessière [2001] introduce soft global constraints in the context of their meta-constraint framework. Since then, several papers have appeared that present filtering algorithms for soft global constraints, many of which use methods from operations research (e.g., matchings and network flows), or computer science (e.g., formal languages). Therefore, the developments in the area of soft global constraints are an exemplary illustration for the successful integration of CP, AI, and OR over the last 10 years.

6.1.2 Outline

The main focus of this chapter will be on soft global constraints. We first introduce basic constraint programming concepts in Section 6.2. Then, in Section 6.3, we introduce soft constraints and show how they can be treated as hard optimization constraints using the meta-constraint framework of Petit, Régin, and Bessière [2000]. Section 6.4 presents soft global constraints: We will discuss in detail the soft `alldifferent` constraint, the soft *global cardinality* constraint, and the soft `regular` constraint. This section also provides a comprehensive overview of other soft global constraints that have appeared in the literature. Section 6.5 discusses constraint-based local search, and shows the parallel between soft global constraints and constraint-based local search. Finally, we present a conclusion and an outlook in Section 6.6.

6.2 Constraint Programming

We first introduce basic constraint programming concepts. For more information on constraint programming we refer to the books by Apt [2003], Dechter [2003], and Rossi et al. [2006]. For more information on global constraints we refer to [Régin, 2003], [van Hoeve and Katriel, 2006], and Chapter 3 of this collection.

Let x be a variable. The *domain* of x, denoted by $D(x)$, is a set of values that can be assigned to x. In this chapter we only consider variables with *finite* domains. For a set of variables X we denote $D(X) = \bigcup_{x \in X} D(x)$.

A *constraint* C on a set of variables $X = \{x_1, x_2, \dots, x_k\}$ is defined as a subset of the Cartesian product of the domains of the variables in X, i.e., $C \subseteq D(x_1) \times D(x_2) \times \cdots \times D(x_k)$. A tuple $(d_1, \dots, d_k) \in C$ is called a *solution* to C. We also say that the tuple *satisfies* C. A value $d \in D(x_i)$ for some $i = 1, \dots, k$ is *inconsistent* with respect to C if it does not belong to a tuple of C, otherwise it is *consistent*. C is *inconsistent* if it does not contain a solution. Otherwise, C is called *consistent*. A constraint is called a *binary constraint* if it is defined on two variables. If it is defined on an arbitrary number of variables, we call it a *global constraint*.

A *constraint satisfaction problem*, or a *CSP*, is defined by a finite set of variables $\mathscr{X} = \{x_1, x_2, \dots, x_n\}$ with respective domains $\mathscr{D} = \{D(x_1), D(x_2), \dots, D(x_n)\}$, together with a finite set of constraints $\mathscr{C}$, each on a subset of $\mathscr{X}$. This is written as $P = (\mathscr{X}, \mathscr{D}, \mathscr{C})$. The goal is to find an assignment $x_i = d_i$ with $d_i \in D(x_i)$ for $i = 1, \dots, n$, such that all constraints are satisfied. This assignment is called a *solution to the CSP*. A *constraint optimization problem*, or *COP*, is a CSP $(\mathscr{X}, \mathscr{D}, \mathscr{C})$ together with an objective function $f : D(x_1) \times \cdots \times D(x_n) \to \mathbb{R}$ that has to be optimized. This is written as $P = (\mathscr{X}, \mathscr{D}, \mathscr{C}, f)$. A variable assignment is a solution to a COP if it is a solution to its associated CSP. An *optimal* solution to a COP is a solution that optimizes the objective function. In this chapter, we assume that the objective function is to be minimized, unless stated otherwise.

The solution process of constraint programming interleaves *constraint propagation* and *search*. The search process essentially consists of enumerating all possible variable-value combinations, until we find a solution or prove that none exists. We say that this process constructs a *search tree*. To reduce the exponential number of combinations, *domain filtering* and *constraint propagation* is applied at each node of the search tree. A *domain filtering algorithm* operates on an individual constraint. Given a constraint, and the current domains of the variables in its scope, a domain filtering algorithm removes domain values that do not belong to a solution to the constraint. Since variables usually participate in several constraints, the updated domains are propagated to the other constraints, whose domain filtering algorithms in effect become active. This process of constraint propagation is repeated for all constraints until no more domain values can be removed, or a domain becomes empty.

In order to be effective, domain filtering algorithms should be computationally efficient, because they are applied many times during the solution process. Further, they should remove as many inconsistent values as possible. If a domain filtering algorithm for a constraint C removes *all* inconsistent values from the domains with respect to C, we say that it makes C *domain consistent*.[1] In other words, all remaining domain values participate in at least one solution to C. More formally:

Definition 1 (Domain consistency). A constraint C on the variables $x_1,\dots,x_k$ is called *domain consistent* if for each variable x_i and each value $d_i \in D(x_i)$ $(i = 1,\dots,k)$, there exist a value $d_j \in D(x_j)$ for all $j \neq i$ such that $(d_1,\dots,d_k) \in C$.

In practice, one usually tries to develop filtering algorithms that separate the check for consistency and the actual domain filtering. That is, we would like to avoid applying the algorithm that performs the consistency check for each individual variable-value pair. Moreover, one typically tries to design incremental algorithms that re-use data structures and partial solutions from one filtering event to the next, instead of applying the filtering algorithm from scratch every time it is invoked.

In the context of constraint optimization problems, we define *optimization constraints* in the following way. Let variable z represent the value of the objective function $f(X)$ to be minimized, where $X = \{x_1,x_2,\dots,x_n\}$ is a set of variables. The corresponding "optimization constraint" can then be defined as

$$C(X,z,f) = \{(d_1,\dots,d_n,d) | d_i \in D(x_i), d \in D(z), f(d_1,\dots,d_n) \leq d\}. \quad (6.1)$$

In other words, C allows only those tuples in the Cartesian product of variables in X that have an objective function smaller than the maximum value of z (we assume z is to be minimized). An optimization constraint is different from a standard inequality constraint mainly because its right-hand side (the value representing the current best solution) will change during the search for a solution. Note that in this definition, we add the function f as an argument to C for syntactical convenience.

It should be noted that we intentionally define z to be not *equal* to $f(X)$ in (6.1). The reason for this is that the relation $f(X) \leq z$ allows us to establish domain consistency on several optimization constraints efficiently. In particular, it implies that we can filter the domains of variables in X with respect to $\max D(z)$, and to potentially increase $\min D(z)$ with respect to X. If we would have used the relation $f(X) = z$ in (6.1) instead, the task of establishing domain consistency becomes NP-complete for general optimization constraints.

In some cases, the objective function aggregates several sub-functions, e.g., $z = z_1 + z_2 + \cdots + z_k$, where $z_i = f_i(X_i)$, and X_i is a set of variables, for $i = 1,\dots,k$. We apply the concept of optimization constraint to each of these variables z_i and functions f_i correspondingly.

6.3 From Soft Constraints to Hard Optimization Constraints

So far, all constraints in a given CSP or COP are defined as hard constraints, that must always be satisfied. We next focus on *soft constraints*, that are allowed to

[1] In the literature, domain consistency is also referred to as *hyper-arc consistency* or *generalized arc consistency*.

be violated. When a soft constraint is violated, we assume that we can measure to what degree it is violated, and that we wish to minimize the overall amount of violation. As discussed in Section 6.1.1, there exist several frameworks to handle soft constraints, and we will focus on the meta-constraint framework introduced by Petit, Régin, and Bessière [2000]; Petit [2002].

The meta-constraint framework of Petit, Régin, and Bessière [2000] for over-constrained problems works as follows. With each soft constraint we associate a particular measure of violation, and a "cost" variable that represents this violation. As we will see later, the eventual effectiveness of a soft constraint depends heavily on the measure of violation that is applied. We then transform each soft constraint into a hard optimization constraint, and minimize an aggregation function on the cost variables. The aggregation function can for example be a weighted sum, or a weighted maximum, of the cost variables. In addition, we can post meta-constraints over the cost variables to model preferences among soft constraints, or more complex relationships, as indicated in Section 6.1.1.

Let us first consider a small motivating example, taken from Petit et al. [2001], to illustrate the application and potential of this framework.

Example 1. Consider the constraint $x \leq y$ where x and y are variables with respective domains specified by the intervals $D(x) = [9000, 10000]$ and $D(y) = [0, 20000]$. We soften this constraint by introducing a cost variable z, representing the amount of violation for the constraint. In this case, we let z represent the gap between x and y if the constraint is not satisfied, that is, z represents $\max\{0, x-y\}$. Suppose the maximum amount of violation is 5, i.e., $D(z) = [0, 5]$. This allows us to deduce that $D(y) = [8995, 20000]$, based on the relation $x - y \leq 5$. We can use the semantics of this constraint to obtain the updated domain efficiently by only comparing the bounds of the variables. If we would not exploit the semantics, but instead list and check all possible variable-value combinations, reducing $D(y)$ would take at least $|D(x)| \cdot 8995$ checks. $\square$

The example above demonstrates how we can exploit the semantics of a constraint to design efficient filtering algorithms for soft constraints. Moreover, it shows that we can perform "back-propagation" from the cost variable to filter the domains of the the other variables. This is crucial to make soft global constraints (and optimization constraints in general) effective in practice [Baptiste et al., 1998], [Focacci et al., 2002].

We next formally introduce violation measures and the transformation of soft constraints into hard optimization constraints, following the notation of van Hoeve, Pesant, and Rousseau [2006a].

Definition 2 (Violation measure). A *violation measure* of a constraint $C(x_1, \ldots, x_n)$ is a function $\mu : D(x_1) \times \cdots \times D(x_n) \to \mathbb{R}_+$ such that $\mu(d_1, \ldots, d_n) = 0$ if and only if $(d_1, \ldots, d_n) \in C$.

Definition 3 (Constraint softening). Let z be a variable with finite domain $D(z)$ and $C(x_1, \ldots, x_n)$ a constraint with a violation measure μ. Then

$$soft\text{-}C(x_1,\ldots,x_n,z,\mu) = \{(d_1,\ldots,d_n,d) \mid d_i \in D(x_i), d \in D(z),\ \mu(d_1,\ldots,d_n) \le d\}$$

is the soft version of C with respect to μ.

In the definition of *soft-C*, z is the cost variable that represents the measure of violation of C; $\max D(z)$ represents the maximum amount of violation that is allowed for C, given the current state of the solution process. Note that *soft-C* is an optimization constraint, since we assume that z is to be minimized.

In addition to definition 2, we usually require that the violation measure allows us to “back-propagate” the domain of the cost variable z to the domains of the other variables efficiently, when we apply definition 3. That is, we need to be able to remove inconsistent domain values from $D(x_1),\ldots,D(x_n)$, based on $D(z)$. The violation measures discussed in this chapter possess that property.

For most global constraints, there exist several natural ways to evaluate the degree to which it is violated, and these are usually not equivalent. Two general measures are the *variable-based* violation measure and the *decomposition-based* violation measure, both introduced by Petit et al. [2001].

Definition 4 (Variable-based violation measure). Let C be a constraint on the variables $x_1,\ldots,x_n$ and let $d_1,\ldots,d_n$ be an instantiation of variables such that $d_i \in D(x_i)$ for $i = 1,\ldots,n$. The *variable-based violation measure* μ_{var} of C is the minimum number of variables that need to change their value in order to satisfy C.

For the decomposition-based violation measure we make use of the binary decomposition of a constraint [Dechter, 1990].

Definition 5 (Binary decomposition). Let C be a constraint on the variables $x_1,\ldots,$ x_n. A *binary decomposition* of C is a minimal set of binary constraints $C_{\text{dec}} = \{C_1,\ldots,C_k\}$ (for integer $k > 0$) on the variables $x_1,\ldots,x_n$ such that the solution set of C equals the solution set of $\bigcap_{i=1}^{k} C_i$.

Note that we can extend the definition of binary decomposition by defining the constraints in C_{dec} on arbitrary variables, such that the solution set of $\bigwedge_{i=1}^{k} C_i$ is mapped to the solution set of C and vice versa, as proposed in [Rossi et al., 1990].

Definition 6 (Decomposition-based violation measure). [2] Let C be a constraint on the variables $x_1,\ldots,x_n$ for which a binary decomposition C_{dec} exists and let $d_1,\ldots,d_n$ be an instantiation of variables such that $d_i \in D(x_i)$ for $i = 1,\ldots,n$. The *decomposition-based violation measure* μ_{dec} of C is the number of violated constraints in C_{dec}.

Example 2. The `alldifferent` constraint specifies that a given set of variables take pairwise different values. Consider the following over-constrained CSP:

$$x_1 \in \{a,b\}, x_2 \in \{a,b\}, x_3 \in \{a,b\}, x_4 \in \{b,c\},$$
$$\texttt{alldifferent}(x_1,x_2,x_3,x_4).$$

[2] In [Petit et al., 2001], the decomposition-based violation measure is referred to as *primal graph based violation cost*.

The following table shows the value of μ_{var} and μ_{dec} for a number of different variable assignments:

(x_1,x_2,x_3,x_4)	μ_{var}	μ_{val}
(a,a,b,c)	1	1
(a,a,b,b)	2	2
(a,a,a,b)	2	3
(b,b,b,b)	3	6

The table shows that μ_{dec} can be more distinctive than μ_{var}. For example, the assignments (a,a,b,b) and (a,a,a,b) are equivalent with respect to μ_{var}, while μ_{dec} is able to distinguish them.

Next, we convert the `alldifferent` constraint into a `soft-alldifferent` constraint, and introduce a variable z that measures its violation. For the sake of this example, we assume that its domain is $D(z) = \{0,1,2\}$:

$$x_1 \in \{a,b\}, x_2 \in \{a,b\}, x_3 \in \{a,b\}, x_4 \in \{b,c\}, z \in \{0,1,2\}$$
$$\texttt{soft-alldifferent}(x_1,x_2,x_3,x_4,z,\mu).$$

We can choose μ to be any measure of violation, for example μ_{dec} or μ_{var}. This choice impacts the solution space; the assignment (a,a,a,b) is allowed by μ_{var} since its violation value is 2, but not by μ_{dec} because its violation value of 3 is higher than the maximum of $D(z)$. □

The variable-based and decomposition-based violation measures can be viewed as "combinatorial violation measures", as they are based on the combinatorial structure of the global constraint. Other violation measures were introduced by Beldiceanu and Petit [2004]. For example, they introduce the *refined* variable-based violation measure, that applies the variable-based violation measure to a specific subset of variables only. Furthermore, they introduce the *object-based* violation measure, that can be applied to high-level modeling objects such as activities in a scheduling context. Finally, they propose specific violation measures based on the *graph properties*-representation of global constraints [Beldiceanu, 2000].

In addition to these general violation measures, alternative measures exist for specific constraints. For example, van Hoeve et al. [2006a] introduce the *value-based* violation measure for the global cardinality constraint, and the *edit-based* violation measure for the `regular` constraint.

After we have assigned a violation measure to each soft constraint, we can recast our problem as follows. Consider a CSP of the form $P = (X,D,C)$. Suppose we partition the constraint set C into a subset of hard constraints C_{hard} and a subset of constraints to be softened C_{soft}. We soften each constraint $c_i \in C_{\text{soft}}$ using the violation measure it has been assigned and a cost variable z_i $(i = 1,\dots,|C_{\text{soft}}|)$ representing this measure. We choose an aggregation function $f : D(z_1) \times \cdots \times D(z_{|C_{\text{soft}}|}) \to \mathbb{R}$ over the cost variables to represent the overall violation to be minimized. Then we transform the CSP into the COP $\tilde{P} = (\tilde{X},\tilde{D},\tilde{C},f)$ where $\tilde{X} = X \cup \{z_1,\dots,z_{|C_{\text{soft}}|}\}$, $\tilde{D}$

constraint	violation measure	consistency check	domain consistency	reference
`alldifferent`		$O(m\sqrt{n})$	$O(m)$	[Régin, 1994]
`soft-alldifferent`	variable-based	$O(m\sqrt{n})$	$O(m)$	[Petit et al., 2001]
`soft-alldifferent`	decomposition-based	$O(mn)$	$O(m)$	[van Hoeve, 2004]
`gcc`		$O(m\sqrt{n})$	$O(m)$	[Quimper et al., 2004]
`soft-gcc`	variable-based	$O(m\sqrt{n})$	$O(m)$	[Zanarini et al., 2006]
`soft-gcc`	value-based	$O(m\sqrt{n})$	$O(m)$	[Zanarini et al., 2006]
`regular`		$O(m)$	$O(m)$	[Pesant, 2004]
`soft-regular`	variable-based	$O(m)$	$O(m)$	[van Hoeve et al., 2006a]
`soft-regular`	edit-based	$O(m)$	$O(m)$	[van Hoeve et al., 2006a]

Table 6.1 Best worst-case time complexity for three hard global constraints on n variables, and their soft counterparts. Here "consistency check" denotes the time complexity to verify that the constraint is consistent, while "domain consistency" denotes the additional time complexity to make the constraint domain consistent, given at least one solution. Each algorithm is based on a graph with m arcs.

contains their corresponding domains, and $\tilde{C}$ contains C_{hard} and the softened version of each constraint in C_{soft}. Note that if our initial problem P is a COP rather than a CSP, we need to define an objective function that balances the original objective and the aggregation of the cost variables.

6.4 Soft Global Constraints

In this section we present several soft global constraints, together with, for some of them, detailed filtering algorithms establishing domain consistency. We will consider in detail the soft `alldifferent` constraint in Section 6.4.1, the soft global cardinality constraint in Section 6.4.2, and the soft `regular` constraint in Section 6.4.3. An interesting observation for these soft global constraints is that the corresponding filtering algorithms establish domain consistency in the same worst-case time complexity as their hard counterparts, as shown in Table 6.1. Finally, in Section 6.4.4 an overview of other soft global constraints will be presented.

Some of the presented filtering algorithms rely on matching theory or network flow theory. We present below the basic definitions that we will use in this chapter. For more information we refer to Schrijver [2003] and Ahuja, Magnanti, and Orlin [1993].

Matchings

Let $G=(V,E)$ be a graph with vertex set V and edge set E. A *matching* $M \subseteq E$ is a subset of edges such that no two edges in M are incident to a common vertex. A vertex that is incident to an edge in M is said to be *covered* by M. A vertex that is not incident to any edge in M is called an *M-free* vertex. A *maximum matching* or *maximum-size matching* is a matching in G of maximum size.

Let $c : V \to \mathbb{N}$ be a "capacity" function on the vertices of G. A *capacitated matching* $M \subseteq E$ is a subset of edges such that each vertex $v \in V$ is incident to at most $c(v)$ edges in M. Note that a capacitated matching is equivalent to a "normal" matching if $c(v) = 1$ for all $v \in V$. A *maximum (capacitated) matching* in a vertex-capacitated graph is a capacitated matching of maximum size.

Network Flows

Let $D = (V,A)$ be a directed graph (or network) and let $s,t \in V$ represent the "source" and the "sink" respectively. An arc $a \in A$ from u to v will also be represented as (u,v).

A function $f : A \to \mathbb{R}$ is called a *flow from s to t*, or an *s-t flow*, if

$$\begin{array}{lll} (i) & f(u,v) \geq 0 & \text{for each } (u,v) \in A, \\ (ii) & \displaystyle\sum_{u:(u,v)\in A} f(u,v) = \sum_{w:(v,w)\in A} f(v,w) & \text{for each } v \in V \setminus \{s,t\}. \end{array} \tag{6.2}$$

Property (6.2)(ii) ensures *flow conservation*, i.e., for a vertex $v \neq s,t$, the amount of flow entering v is equal to the amount of flow leaving v.

The *value* of an s-t flow f is defined as

$$\text{value}(f) = \sum_{v:(s,v)\in A} f(s,v) - \sum_{u:(u,s)\in A} f(u,s).$$

In other words, the value of a flow is the net amount of flow leaving s, which by flow conservation must be equal to the net amount of flow entering t.

In a flow network, each arc $a \in A$ has an associated "demand" $d(a)$ and "capacity" $c(a)$, such that $0 \leq d(a) \leq c(a)$. We say that a flow f is *feasible* in the network if $d(a) \leq f(a) \leq c(a)$ for every $a \in A$. If the demand d and capacity c are integer-valued, it can be shown that if there exists a feasible flow, there also exists an *integer* feasible flow in D.

Let $w : A \to \mathbb{R}$ be a "weight" (or "cost") function on the arcs. We define the weight of a directed path P as weight$(P) = \sum_{a\in P} w(a)$. Similarly for a directed circuit. The weight of a flow f is defined as

$$\text{weight}(f) = \sum_{a\in A} w(a)f(a).$$

A feasible flow f is called a *minimum-weight flow* if weight$(f) \leq$ weight(f') for any feasible flow f'.

Let f be an s-t flow in G. The *residual graph* of f (with respect to c and d) is defined as $D_f = (V, A_f)$, where the arc set A_f is defined as follows. For all arcs $a = (u,v) \in A$:

- if $f(a) < c(a)$ then $(u,v) \in A_f$ with residual demand $\max\{d(a) - f(a), 0\}$, residual capacity $c(a) - f(a)$, and residual weight $w(a)$,
- if $f(a) > d(a)$ then $(v,u) \in A_f$ with residual demand 0, residual capacity $f(a) - d(a)$, and residual weight $-w(a)$.

6.4.1 Soft Alldifferent Constraint

The `alldifferent` constraint on a set of variables specifies that all variables should take pairwise different values. Here we consider two measures of violation to soften the `alldifferent` constraint: The variable-based violation measure μ_{var} and the decomposition-based violation measure μ_{dec}. For `alldifferent`$(x_1,\dots,x_n)$ we have

$$\mu_{\text{var}}(x_1,\dots,x_n) = \textstyle\sum_{d \in D(X)} \max\left(|\{i \mid x_i = d\}| - 1, 0\right),$$
$$\mu_{\text{dec}}(x_1,\dots,x_n) = \left|\left\{(i,j) \mid x_i = x_j, \text{ for } i < j\right\}\right|.$$

If we apply Definition 3 to the `alldifferent` constraint using the measures μ_{var} and μ_{dec}, we obtain `soft-alldifferent`$(x_1,\dots,x_n,z,\mu_{\text{var}})$ and `soft-alldifferent`$(x_1,\dots,x_n,z,\mu_{\text{dec}})$. Each of the violation measures μ_{var} and μ_{dec} gives rise to a different domain consistency filtering algorithm for `soft-alldifferent`.

Variable-Based Violation Measure

A domain consistency filtering algorithm for the variable-based `soft-alldifferent` constraint was presented by Petit, Régin, and Bessière [2001]. It makes use of bipartite matchings.

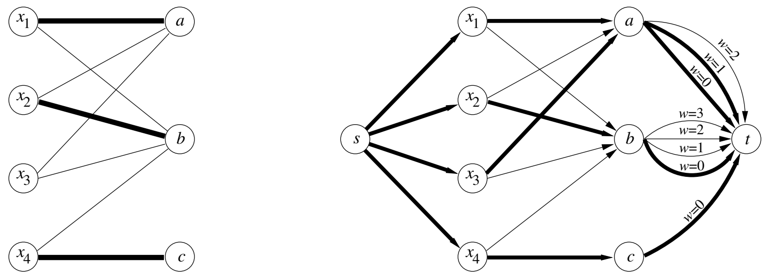
Throughout this section, let X be a set of variables. The *value graph* of X is a bipartite graph $\mathscr{G}(X) = (V,E)$ where $V = X \cup D(X)$ and $E = \{(x,d) \mid x \in X, d \in D(x)\}$ [Lauriere, 1978]. It was first observed by Régin [1994] that a solution to `alldifferent`(X) is equivalent to a matching covering X in the corresponding value graph. For the variable-based `soft-alldifferent` constraint, we can exploit the correspondence with bipartite matchings in a similar way.

Lemma 1. [Petit et al., 2001] *Let M be a maximum-size matching in the value graph $\mathscr{G}(X)$. For* `alldifferent`(X)*, the minimum value of $\mu_{\text{var}}(X)$ is equal to $|X| - |M|$.*

Theorem 1. [Petit et al., 2001] *The constraint* `soft-alldifferent`(X,z,μ_{var}) *is domain consistent if and only if*

i) all edges in the value graph $\mathscr{G}(X)$ belong to a matching M in $\mathscr{G}(X)$ with $|X| - |M| \leq \max D(z)$, and

ii) $\min D(z) \geq |X| - |M|$, where M is a maximum-size matching in $\mathscr{G}(X)$.



a. Maximum matching in the value graph b. Minimum-weight flow in the value network

Fig. 6.1 Graph representation for the `soft-alldifferent` constraint. In figure a., the value graph for the variable-based `soft-alldifferent` is depicted; bold edges form a maximum matching. In figure b., the extended value network for the decomposition-based `soft-alldifferent` is presented. For all arcs the capacity is 1. For some arcs the weight w is given, for all other arcs the weight is 0.

We can apply Theorem 1 to establish domain consistency for `soft-alldifferent`$(x_1,\dots,x_n,z,\mu_{\text{var}})$ as follows. First, we compute a maximum matching M in the value graph. This can be done in $O(m\sqrt{n})$ time [Hopcroft and Karp, 1973], where m is the number of edges in the graph. We then distinguish the following cases:

- If $n-|M| > \max D(z)$, the constraint is inconsistent.
- If $n-|M| < \max D(z)$, the constraint is consistent, and moreover all domain values are consistent. Namely, if we change the value of any variable, the violation increases with at most 1 unit.
- If $n-|M| = \max D(z)$, the constraint is consistent, and only those domain values $d \in D(x)$ whose corresponding edge (x,d) belongs to a maximum matching are consistent. We can identify all consistent domain values in the same way as for the hard `alldifferent` constraint. That is, we direct the edges in M from X to $D(X)$, and edges not in M from $D(X)$ to X. Then, an edge belongs to any maximum matching if and only if it belongs to M, or it belongs to a path starting from an M-free vertex, or it belongs to a strongly connected component. All these edges can be identified, and the corresponding domain values can be removed, in $O(m)$ time [Régin, 1994; Tarjan, 1972].

Finally, we can update $\min D(z)$ to be the maximum of its current value and $n-|M|$.

The algorithm above separates the check for consistency and the actual domain filtering. Moreover, it can be implemented to behave incrementally; after k domain changes, a new matching can be found in $O(\min\{km,\sqrt{n}m\})$ time, by re-using the previous matching.

Example 3. Consider the following CSP:

$$x_1 \in \{a,b\}, x_2 \in \{a,b\}, x_3 \in \{a,b\}, x_4 \in \{b,c\}, z \in \{0,1\},$$
$$\texttt{soft-alldifferent}(x_1,x_2,x_3,x_4,z,\mu_{\text{var}}).$$

The corresponding value graph is depicted in Figure 6.1.a. The bold edges indicate a maximum-size matching, covering three variables. Hence, the minimum value of μ_{var} is $4-3=1$, which is equal to $\max D(z)$. This allows us to remove edge (x_4,b), as it does not belong to a matching of size 3. Also, note that we can remove value 0 from $D(z)$, since it does not belong to any solution. □

An alternative domain consistency algorithm for the variable-based `soft-all-different` constraint was given by van Hoeve et al. [2006a], based on the correspondence to a minimum-weight network flow. In that work, additional arcs are introduced to the network whose weights reflect the violation measure. A similar approach is presented in the next section, for the decomposition-based `soft-all-different` constraint.

Decomposition-Based Violation Measure

A first filtering algorithm for the decomposition-based `soft-alldifferent` constraint was given by Petit et al. [2001]. It does not necessarily establish domain consistency, and runs in $O(m^2 n\sqrt{n})$ time, where n is the number of variables and m is the sum of the cardinalities of their domains. A domain consistency filtering algorithm was given in van Hoeve [2004], running in $O(mn)$ time. Here we present the latter algorithm.

The filtering algorithm for the decomposition-based `soft-alldifferent` constraint by van Hoeve [2004] exploits the correspondence with a minimum-weight network flow. Let us first introduce the network representation of the hard `alldiff-erent` constraint, which can be viewed as an extension of the value graph. For a set of variables X, we define the *value network* of X as a directed graph $\mathscr{D}(X)=(V,A)$, with vertex set $V=X\cup D(X)\cup\{s,t\}$, and arc set $A=A_s\cup A_X\cup A_t$, where

$$\begin{aligned}A_s &= \{(s,x)\mid x\in X\},\\ A_X &= \{(x,d)\mid x\in X, d\in D(x)\},\\ A_t &= \{(d,t)\mid d\in D(X)\},\end{aligned}$$

with "capacity" function $c(a)=1$ for all $a\in A$. An integer flow f of value $|X|$ in $\mathscr{D}(X)$ corresponds to a solution to the constraint `alldifferent`(X); the solution is formed by assigning $x=d$ for all arcs $a=(x,d)\in A_X$ with $f(a)=1$. Moreover, those arcs form a maximum-size matching in the graph induced by A_X (i.e., the value graph).

If the `alldifferent` constraint cannot be satisfied, there does not exist a flow of value $|X|$ in the value network. Therefore, for the `soft-alldifferent` constraint, we adapt the value network in such a way that a flow of value $|X|$ becomes possible, and moreover represents a variable assignment whose violation measure is exactly the cost of the network flow. This is done as follows.

In the graph $\mathscr{D}(X)=(V,A)$, we replace the arc set A_t by $\tilde{A}_t=\{(d,t)\mid d\in D(x), x\in X\}$, with capacity $c(a)=1$ for all arcs $a\in\tilde{A}_t$. Note that $\tilde{A}_t$ contains parallel arcs if two or more variables share a domain value. If there are k parallel arcs

(d,t) between some $d \in D(X)$ and t, we distinguish them by numbering the arcs as $(d,t)_0, (d,t)_1, \dots, (d,t)_{k-1}$ in a fixed but arbitrary way. One can view the arcs $(d,t)_0$ to be the original arc set A_t.

We next apply a "cost" function $w : A \to \mathbb{N}$ as follows. If $a \in \tilde{A}_t$, i.e., $a = (d,t)_i$ for some $d \in D(X)$ and integer i, we define $w(a) = i$. Otherwise $w(a) = 0$. Let the resulting digraph be denoted by $\mathscr{D}_{\text{dec}}(X)$. We have the following result.

Lemma 2. [van Hoeve, 2004] *Let X be a set of variables, and let f be an integer s-t flow of value $|X|$ in $\mathscr{D}_{\text{dec}}(X)$. Let $\bar{X}$ be the variable assignment $\{x = d \mid (x,d) \in A_X, f(x,d) = 1\}$. For* `alldifferent`$(X)$, $\mu_{\text{dec}}(\bar{X}) = \text{weight}(f)$.

Example 4. For the problem in Example 2, the extended value network $\mathscr{D}_{\text{dec}}(X)$ is presented in Figure 6.1.b. Bold arcs indicate a minimum-weight flow of weight 1, corresponding to the variable assignment $x_1 = a, x_2 = b, x_3 = a, x_4 = c$. Indeed, this assignment violates one not-equal constraint, $x_1 \neq x_3$.

To illustrate how the cost structure of $\mathscr{D}_{\text{dec}}$ represents the decomposition-based violation measure, suppose we were to assign all variables to value b. Then there are three units of flow that need to use an arc in $\tilde{A}_t$ with positive cost, while one unit of flow can use the arc in $\tilde{A}_t$ without violation cost. Indeed, for the first variable assigned to b, say x_1, there is no violated binary constraint and the corresponding unit of flow may use the arc without violation cost. The second variable assigned to b, say x_2, violates one binary constraint, namely $x_1 \neq x_2$. Indeed it uses the arc with the next lowest possible cost, i.e., 1. The following variable assigned to b, say x_3, violates two binary constraints (involving x_1 and x_2), which corresponds to using the arc with cost 2. Finally, the fourth variable assigned to b, x_4, violates three binary constraints and uses the arc with cost 3. Together, they exactly constitute the decomposition-based violation of value 6. □

Theorem 2. [van Hoeve, 2004] *The constraint* `soft-alldifferent`(X, z, μ_{dec}) *is domain consistent if and only if*

i) *for every arc $a \in A_X$ there exists an integer feasible s-t flow f of value $|X|$ in $\mathscr{D}_{\text{dec}}(X)$ with $f(a) = 1$ and* $\text{weight}(f) \leq \max D(z)$*, and*
ii) $\min D(z) \geq \text{weight}(f)$ *for a feasible minimum-weight s-t flow f of value $|X|$ in $\mathscr{D}_{\text{dec}}$.*

We can apply Theorem 2 to establish domain consistency for `soft-alldifferent`(X, z, μ_{dec}) as follows. We first compute a minimum-weight flow f in $\mathscr{D}_{\text{dec}}$. Since the only positive costs are on arcs in $\tilde{A}_t$, this can be done in $O(mn)$ time, where m is the number of arcs in the graph, and n is the number of variables in X [van Hoeve, 2004]. If $\text{weight}(f) > \max D(z)$ we know that the constraint is inconsistent.

Consistent domain values $d \in D(x)$ for $x \in X$ correspond to arcs $a = (x,d) \in A_X$ for which there exists a flow g with $g(a) = 1$, $\text{value}(g) = |X|$ and $\text{weight}(g) \leq \max D(z)$. To identify these arcs we apply a theorem from flow theory stating that a minimum-weight flow g with $g(a) = 1$ can be found by "re-routing" the flow f through a shortest directed cycle C containing the arc a in the residual graph of f. Then $\text{weight}(g) = \text{weight}(f) + \text{weight}(C)$. In other words, for each arc $a = (x,d)$

with $f(a) = 0$, we need to compute a shortest d-x path in the residual graph. If the weight of this path exceeds $\max D(z) - \text{weight}(f)$, the value $d \in D(x)$ is inconsistent.

In order to find the shortest d-x paths, we first consider the strongly connected components in the graph induced by A_X. For all arcs (x,d) in these components, the shortest d-x path will remain within the component and has cost 0; indeed, if the path would visit t the cost cannot decrease since f is a minimum-weight flow.

We next consider all arcs (x,d) between two strongly connected components. Observe that we can assume that the shortest d-x path must visit t *exactly once*. Therefore we can split the path into two parts: The shortest d-t path and the shortest t-x path. Now, all vertices inside a strongly connected component have the same shortest distance to t, and also the same shortest distance from t (possibly visiting other strongly connected components). Therefore, we can contract the strongly connected components in the graph induced by A_X, and use the resulting acyclic "component graph". Since the algorithm to compute the strongly connected components also provides the topological order and inverse topological order of the component graph, we can apply these to efficiently compute the shortest distance to and from t for every component. Hence, a shortest d-x path is the shortest path from the component to which d belongs to t, plus the path from t to the component to which x belongs. All these computations can be done in $O(m)$ time [Tarjan, 1972; Cormen et al., 2001].

Finally, we update $\min D(z) = \text{weight}(f)$ if $\min D(z) < \text{weight}(f)$. Again, this algorithm separates the consistency check from the actual domain filtering. Moreover, the algorithm can be implemented to behave incrementally. After k domain changes, we can re-compute a minimum-weight flow in $O(km)$ time.

6.4.2 *Soft Global Cardinality Constraint*

The *global cardinality constraint* (`gcc`) was introduced by Régin [1996]. It is defined on a set of variables and specifies for each value in the union of their domains an upper and lower bound to the number of variables that are assigned to this value.

Throughout this section, let $X = \{x_1, \dots, x_n\}$ be a set of variables and let $l_d, u_d \in \mathbb{N}$ with $l_d \le u_d$ for all $d \in D(X)$.

Definition 7 (Global cardinality constraint).

$$\begin{aligned}\texttt{gcc}(X,l,u) = \{(d_1,\dots,d_n) \mid {} & d_i \in D(x_i)\ \forall i \in \{1,\dots,n\}, \\ & l_d \le |\{i \mid d_i = d\}| \le u_d\ \forall d \in D(X)\}.\end{aligned}$$

The `gcc` is a generalization of the `alldifferent` constraint; if we set $l_d = 0$ and $u_d = 1$ for all $d \in D(X)$, the `gcc` is equal to the `alldifferent` constraint.

In order to define measures of violation for the `gcc`, it is convenient to introduce for each domain value a "shortage" function $s : D(x_1) \times \cdots \times D(x_n) \times D(X) \to \mathbb{N}$

and an “excess” function $e : D(x_1) \times \cdots \times D(x_n) \times D(X) \to \mathbb{N}$ as follows [van Hoeve et al., 2006a]:

$$s(X,d) = \begin{cases} l_d - |\{x \mid x \in X, x = d\}| & \text{if } |\{x \mid x \in X, x = d\}| \le l_d, \\ 0 & \text{otherwise,} \end{cases}$$

$$e(X,d) = \begin{cases} |\{x \mid x \in X, x = d\}| - u_d & \text{if } |\{x \mid x \in X, x = d\}| \ge u_d, \\ 0 & \text{otherwise.} \end{cases}$$

For $\texttt{gcc}(X,l,u)$, the variable-based violation measure μ_{var} can then be expressed in terms of the shortage and excess functions:

$$\mu_{\text{var}}(X) = \max \left(\sum_{d \in D(X)} s(X,d), \sum_{d \in D(X)} e(X,d) \right)$$

provided that

$$\sum_{d \in D(X)} l_d \le |X| \le \sum_{d \in D(X)} u_d. \tag{6.3}$$

Note that if condition (6.3) does not hold, there is no variable assignment that satisfies the $\texttt{gcc}$, and μ_{var} cannot be applied. Therefore, van Hoeve et al. [2006a] introduced the following violation measure for the $\texttt{gcc}$, which can also be applied when assumption (6.3) does not hold.

Definition 8 (Value-based violation measure). For $\texttt{gcc}(X,l,u)$ the *value-based violation measure* is

$$\mu_{\text{val}}(X) = \sum_{d \in D(X)} (s(X,d) + e(X,d)).$$

Example 5. Consider the over-constrained CSP

$$x_1 \in \{1,2\}, x_2 \in \{1\}, x_3 \in \{1,2\}, x_4 \in \{1\},$$
$$\texttt{gcc}(x_1,x_2,x_3,x_4,[1,3],[2,5]).$$

That is, value 1 must be taken between 1 and 2 times, while value 2 must be taken between 3 and 5 times. The violation measures for all possible tuples are:

(x_1,x_2,x_3,x_4)	$\sum_{d \in D(X)} s(X,d)$	$\sum_{d \in D(X)} e(X,d)$	μ_{var}	μ_{val}
$(1,1,1,1)$	3	2	3	5
$(2,1,1,1)$	2	1	2	3
$(1,1,2,1)$	2	1	2	3
$(2,1,2,1)$	1	0	1	1

□

For both the variable-based and value-based violation measures for the `soft-gcc` constraint, van Hoeve et al. [2006a] present domain consistency filtering algorithms, running in $O(n(m+n\log n))$ and $O((n+k)(m+n\log n))$ time respectively, where n is the number of variables, m is the sum of the cardinalities of the variable domains, and k is the cardinality of the union of the variable domains. Their algorithms are based on an extension of the value network for the decomposition-based `soft-all-different` constraint. They apply the same concept of adding "violation arcs" to allow feasible flows with cost equal to the corresponding variable assignment. A more efficient approach based on matching theory, running in $O(m\sqrt{n})$ time, was proposed by Zanarini, Milano, and Pesant [2006], and we describe their method below.

Two Capacitated Matchings

Similar to the method proposed by Petit, Régin, and Bessière [2001], the approach taken by Zanarini, Milano, and Pesant [2006] for the `soft-gcc` uses the value graph representation. Recall from section 6.4.1 that for a set of variables X, the value graph of X is a bipartite graph $\mathscr{G}(X)=(V,E)$ where $V=X\cup D(X)$ and $E=\{(x,d)\mid x\in X, d\in D(x)\}$. For the `soft-gcc`, the goal is to find two *capacitated* maximum matchings, one minimizing the shortage function and one minimizing the excess function. These matchings can then be used to measure the overall violation cost.

For a constraint $\texttt{gcc}(X,l,u)$, we define two vertex-capacitated value graphs $\mathscr{G}_e(X)$ and $\mathscr{G}_s(X)$, by extending the value graph with a "capacity" function $c: V\to\mathbb{N}$ on its vertices. For both $\mathscr{G}_e(X)$ and $\mathscr{G}_s(X)$, we define $c(x)=1$ for each vertex $x\in X$. For the vertices $d\in D(X)$, we define $c(d)=l_d$ for $\mathscr{G}_s(X)$ and $c(d)=u_d$ $\mathscr{G}_e(X)$. We will slightly abuse terminology and refer to a capacitated matching as simply a matching.

We first focus on minimizing the *excess* function. Let M_e be a maximum matching in the value graph $\mathscr{G}_e$. If $|M_e|=|X|$, the edges in M_e correspond to a partial assignment satisfying the upper capacities u of the values in the `gcc`. If $|M_e|<|X|$, exactly $|X|-|M_e|$ variables must be assigned to a saturated value, which equals the total excess for all domain values, i.e., $\sum_{d\in D(X)} e(X,d)=|X|-|M_e|$.

Analogously for the *shortage* function, let M_s be a maximum matching in the value graph $\mathscr{G}_s$. Edges in M_s correspond to a partial assignment satisfying the lower capacities l of the values in the `gcc`. If $|M_s|<\sum_{d\in D(X)} l_d$, one or more values have not enough variables assigned to them. In fact, the difference corresponds to the total shortage of all domain values, i.e., $\sum_{d\in D(X)} s(X,d)=\sum_{d\in D(X)} l_d-|M_s|$.

Variable-based violation measure

We can characterize domain consistency for the variable-based `soft-gcc` as follows.

Theorem 3. [Zanarini et al., 2006] *The constraint* `soft-gcc`$(X,l,u,z,\mu_{\text{var}})$ *is domain consistent if and only if* $\min D(z) \geq \max\left\{|X|-|M_e|, \sum_{d\in D(X)} l_d - |M_s|\right\}$*, and either*

i) $\max\left\{|X|-|M_e|, \sum_{d\in D(X)} l_d - |M_s|\right\} < \max D(z)$*, or*

ii) $|X|-|M_e| = \max D(z)$ *and* $\sum_{d\in D(X)} l_d - |M_s| < \max D(z)$*, and all edges in* G_e *belong to a maximum matching, or*

iii) $|X|-|M_e| < \max D(z)$ *and* $\sum_{d\in D(X)} l_d - |M_s| = \max D(z)$*, and all edges in* G_s *belong to a maximum matching, or*

iv) $|X|-|M_e| = \sum_{d\in D(X)} l_d - |M_s| = \max D(z)$*, and all edges in* $\mathscr{G}_e$ *and* $\mathscr{G}_s$ *belong to a maximum matching.*

To establish domain consistency algorithmically, we first compute maximum matchings M_e and M_s in the value graphs $\mathscr{G}_e$ and $\mathscr{G}_s$ respectively. An algorithm to compute such capacitated matchings was given by Quimper et al. [2004]. It is a generalization of the Hopcroft-Karp algorithm and, similar to the Hopcroft-Karp algorithm, runs in $O(m\sqrt{n})$, where $n = |X|$ and m is the number edges in the value graph. If $\max\left\{|X|-|M_e|, \sum_{d\in D(X)} l_d - |M_s|\right\} > \max D(z)$, we know the constraint is inconsistent.

Next, we filter the inconsistent edges and corresponding domain values. Using the cardinalities of M_e and M_s, we can easily determine which of the four cases of Theorem 3 applies. In case *ii*), *iii*), or *iv*), we can identify all edges that do not belong to a maximum matching in $O(m)$ time, similar to the approach for the variable-based `soft-alldifferent` constraint in Section 6.4.1.

Once again, the algorithm separates the check for consistency and the actual domain filtering, and it can be implemented to behave incrementally.

Notice that Theorem 3 is an extension of Theorem 1 for the variable-based `soft-alldifferent` constraint by Petit et al. [2001]. In fact, when the upper bounds u_d are 1 for all $d \in D(X)$, the two filtering algorithms are equivalent.

Beldiceanu and Petit [2004] discuss the variable-based violation measure for a different version of the `soft-gcc`. Their version considers the parameters l and u to be variables instead of constants. Hence, the variable-based violation measure becomes a rather poor measure, as we trivially can change l and u to satisfy the `gcc`. For this reason they introduce the refined variable-based violation measure, and apply it to their version of the `soft-gcc` by restricting the violation measure to the set of variables X, which corresponds to the `soft-gcc` described above. Beldiceanu and Petit [2004] do not provide a filtering algorithm, however.

Value-Based Violation Measure

For the value-based `soft-gcc`, domain consistency can be characterized as follows.

Theorem 4. [Zanarini et al., 2006] *The constraint* `soft-gcc`$(X,l,u,z,\mu_{\text{val}})$ *is domain consistent if and only if* $\min D(z) \geq |X|-|M_e| + \sum_{d\in D(X)} l_d - |M_s|$*, and either* *i)* $|X| - |M_e| + \sum_{d \in D(X)} l_d - |M_s| < \max D(z)$*, or*

ii) $|X| + |M_e| = \max D(z) - 1$ *and all edges belong to a maximum matching in at least one of* G_e *or* G_s*, or*

iii)$|X| + |M_e| = \max D(z)$ *and all edges belong to a maximum matching in both* G_e *and* G_s*.*

The filtering algorithm for the value-based soft-gcc proceeds similar to the algorithm for the variable-based soft-gcc. We first need to compute maximum matchings M_e and M_s, again in $O(m\sqrt{n})$ time, which allows us to perform the consistency check. We then remove all edges and corresponding domain values in $O(m)$ time, if we are in cases *ii*) and *iii*).

6.4.3 Soft Regular Constraint

The regular constraint was introduced by Pesant [2004] (related concepts were introduced by Beldiceanu, Carlsson, and Petit [2004]). It is defined on a fixed-length sequence of finite-domain variables and it states that the corresponding sequence of values taken by these variables belongs to a given regular language. Particular instances of the regular constraint can for example be applied in rostering problems or sequencing problems.

Before we introduce the regular constraint we need the following definitions [Hopcroft and Ullman, 1979]. A *deterministic finite automaton* (DFA) is described by a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states. Given an input string, the automaton starts in the initial state q_0 and processes the string one symbol at the time, applying the transition function δ at each step to update the current state. The string is *accepted* if and only if the last state reached belongs to the set of final states F. Strings processed by M that are accepted are said to belong to the language defined by M, denoted by $L(M)$. For example with M depicted in Figure 6.2, strings *aaabaa* and *cc* belong to $L(M)$ but not *aacbba*. The languages recognized by DFAs are precisely regular languages.

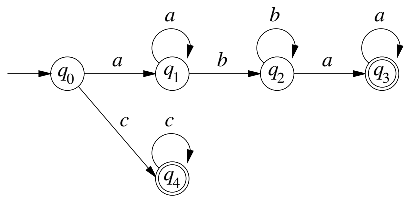


Fig. 6.2 A representation of a DFA with each state shown as a circle, final states as a double circle, and transitions as arcs.

Given an ordered sequence of variables $X = x_1, x_2, \ldots, x_n$ with respective finite domains $D(x_1), D(x_2), \ldots, D(x_n) \subseteq \Sigma$, there is a natural interpretation of the set of possible instantiations of X, i.e., $D(x_1) \times D(x_2) \times \cdots \times D(x_n)$, as a subset of all strings of length n over Σ.

Definition 9 (Regular language membership constraint). Let $M = (Q, \Sigma, \delta, q_0, F)$ denote a DFA and let $X = x_1, x_2, \ldots, x_n$ be a sequence of variables with respective finite domains $D(x_1), D(x_2), \ldots, D(x_n) \subseteq \Sigma$. Then

$$\texttt{regular}(X, M) = \{(d_1, \ldots, d_n) \mid d_i \in D(x_i), d_1 d_2 \cdots d_n \in L(M)\}.$$

Here we consider two measures of violation for the `regular` constraint: The variable-based violation measure μ_{var} and the *edit-based* violation measure μ_{edit} that was introduced by van Hoeve et al. [2006a].

Let s_1 and s_2 be two strings of the same length. The *Hamming distance* $H(s_1, s_2)$ is the number of positions in which they differ. Associating with a tuple $(d_1, d_2, \ldots, d_n)$ the string $d_1 d_2 \cdots d_n$, the variable-based violation measure can be expressed in terms of the Hamming distance:

$$\mu_{\text{var}}(X) = \min\{H(D, X) \mid D = d_1 \cdots d_n \in L(M)\}.$$

Another distance function that is often used for comparing two strings is the following. Again, let s_1 and s_2 be two strings of the same length. The *edit distance* $E(s_1, s_2)$ is the smallest number of insertions, deletions, and substitutions required to change one string into another. It captures the fact that two strings that are identical except for one extra or missing symbol should be considered close to one another. The edit distance is probably a better way to measure violations of a `regular` constraint than the Hamming distance. Consider for example a regular language in which strings alternate between pairs of a's and b's, e.g., "*aabbaabbaa*" belongs to this language. The string "*abbaabbaab*" does not belong to the language, and the minimum Hamming distance, i.e., to any string of the same length that belongs to the language, is 5 (that is, the length of the string divided by 2) since changing either the first a to a b or the first b to an a has a domino effect. On the other hand, the minimum edit distance of the same string is 2, since we can insert an a at the beginning and remove a b at the end. In this case, the edit distance reflects the number of incomplete pairs whereas the Hamming distance is proportional to the length of the string rather than to the amount of violation.

Definition 10 (Edit-based violation measure). For $\texttt{regular}(X, M)$ the *edit-based violation measure* is

$$\mu_{\text{edit}}(X) = \min\{E(D, X) \mid D = d_1 \cdots d_n \in L(M)\}.$$

Example 6. Consider the CSP

$$\begin{array}{l} x_1 \in \{a,b,c\}, x_2 \in \{a,b,c\}, x_3 \in \{a,b,c\}, x_4 \in \{a,b,c\}, \\ \texttt{regular}(x_1, x_2, x_3, x_4, M) \end{array}$$

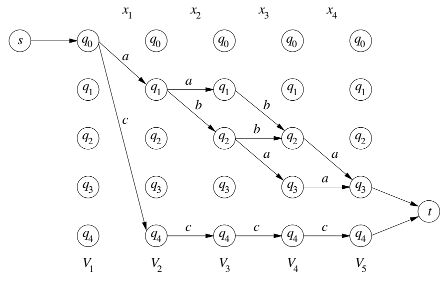


Fig. 6.3 Graph representation for the `regular` constraint of Example 6, after filtering inconsistent arcs.

with DFA M as in Figure 6.2. We have $\mu_{\text{var}}(c,a,a,b) = 3$, because we need to change the value of at least 3 variables; corresponding valid strings with Hamming distance 3 are for example *aaba* or *cccc*. On the other hand, we have $\mu_{\text{edit}}(c,a,a,b) = 2$, because we can delete the value c at the front and add the value a at the end, thus obtaining the valid string *aaba*. □

A graph representation for the `regular` constraint was presented by Pesant [2004]. Recall that $M = (Q,\Sigma,\delta,q_0,F)$.

Theorem 5. [Pesant, 2004] *A solution to* `regular`(X,M) *corresponds to an s-t path in the digraph* $\mathscr{R} = (V,A)$ *with vertex set*

$$V = V_1 \cup V_2 \cup \cdots \cup V_{n+1} \cup \{s,t\}$$
$$\textit{and arc set} \quad A = A_s \cup A_1 \cup A_2 \cup \cdots \cup A_n \cup A_t,$$

$$\begin{aligned} \textit{where} \quad & V_i = \{q_k^i \mid q_k \in Q\} \;\; \textit{for } i = 1,\ldots,n+1, \\ \textit{and} \quad & A_s = \{(s,q_0^1)\}, \\ & A_i = \{(q_k^i,q_l^{i+1}) \mid \delta(q_k,d) = q_l \, \textit{for } d \in D(x_i)\} \;\; \textit{for } i = 1,\ldots,n, \\ & A_t = \{(q_k^{n+1},t) \mid q_k \in F\}. \end{aligned}$$

Theorem 5 can be applied to filter the `regular` constraint to domain consistency, by removing all arcs (and corresponding domain values) that do not belong to an *s-t* path in $\mathscr{R}$. For the `regular` constraint in Example 6, Figure 6.3 gives the corresponding graph representation, after filtering inconsistent arcs. Observe that the filtering algorithm has correctly removed domain value b from $D(x_1)$ and $D(x_4)$.

Whenever the `regular` constraint cannot be satisfied there does not exist an *s-t* path in $\mathscr{R}$. Therefore, for the `soft-regular` constraint, van Hoeve et al. [2006a] extend the digraph $\mathscr{R}$ in such a way that an *s-t* path always exist, and has a cost corresponding to the respective measure of violation. For both the variable-based and the edit-based `soft-regular` constraint, again particular weighted "violation arcs" are added to $\mathscr{R}$ to make this possible.

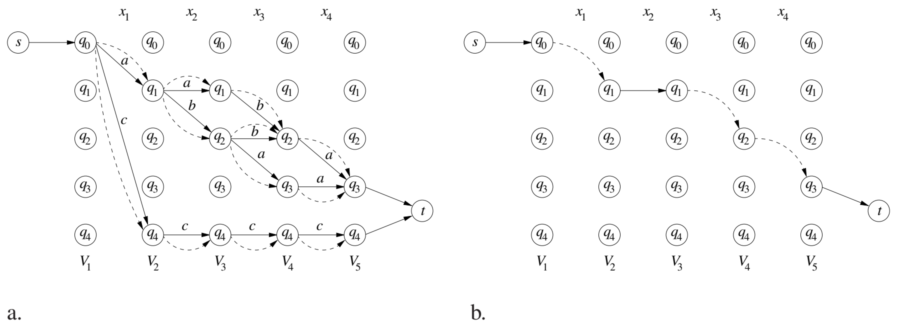


Fig. 6.4 a. Graph representation for the variable-based `soft-regular` constraint. Dashed arcs indicate the inserted weighted arcs with weight 1. b. Example: arcs and associated path used in solution $x_1 = c, x_2 = a, x_3 = a, x_4 = b$ of weight 3, corresponding to three substitutions from valid string *aaba*.

Variable-Based Violation Measure

For the variable-based `soft-regular` constraint, we add the following violation arcs to the graph $\mathscr{R}$ of Theorem 5:

$$A_{\text{sub}} = \{(q_k^i, q_l^{i+1}) \mid \delta(q_k, d) = q_l \text{ for some } d \in \Sigma,\ i = 1, \ldots, n\}.$$

We next apply a "cost" function $w : A \to \mathbb{N}$ as follows. For all arcs $a \in A$, $w(a) = 1$ if $a \in A_{\text{sub}}$ and $w(a) = 0$ otherwise. Let the resulting digraph be denoted by $\mathscr{R}_{\text{var}}$ (see Figure 6.4 for an illustration on Example 6).

The input automaton of this constraint specifies the allowed transitions from state to state according to different values. The objective here, in counting the minimum number of substitutions, is to make these transitions value independent. Therefore, the violation arcs in A_{sub} are added between two states (q_k^i, q_l^{i+1}) if there already exists at least one valid arc between them. This means that an *s*-*t* path using a violation arc is in fact a solution where a variable takes a value outside of its domain. The number of such variables thus constitutes a minimum on the number of variables which need to change value.

Theorem 6. [van Hoeve et al., 2006a] *The constraint* `soft-regular`$(X, M, z, \mu_{\text{var}})$ *is domain consistent if and only if*

i) *every arc* $a \in A_1 \cup \cdots \cup A_n$ *belongs to an s-t path P in* $\mathscr{R}_{\text{var}}$ *with* $\text{weight}(P) \leq \max D(z)$*, and*

ii) $\min D(z) \geq \text{weight}(P)$ *for a minimum-weight s-t path in* $\mathscr{R}_{\text{var}}$.

The filtering algorithm must ensure that all arcs corresponding to a variable-value assignment are on an *s*-*t* path with cost smaller than $\max D(z)$. Computing shortest paths from the initial state in the first layer to every other node and from every node to a final state in the last layer can be done in $O(n|\delta|)$ time through topological sorts because of the special structure of the graph (it is acyclic), as observed by

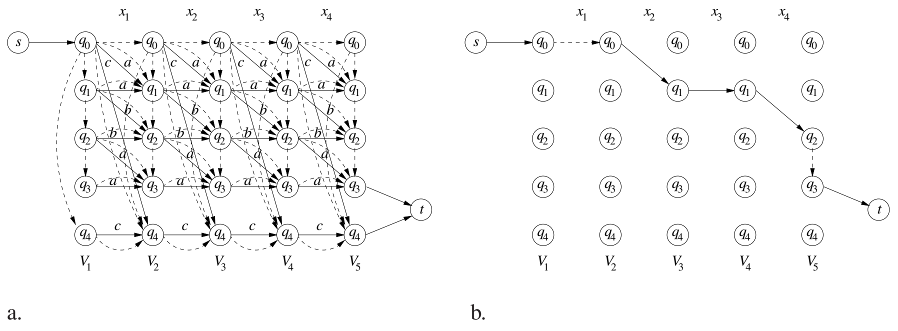


Fig. 6.5 a. Graph representation for the edit-based `soft-regular` constraint. Dashed arcs indicate the inserted weighted arcs with weight 1. b. Example: arcs and associated path used in solution $x_1 = c, x_2 = a, x_3 = a, x_4 = b$ of weight 2, corresponding to one deletion (c in position 1) and one insertion (a in position 4) from valid string *aaba*.

Pesant [2004]. Here $|\delta|$ denotes the number of transitions in the corresponding DFA. Hence, the algorithm runs in $O(m)$ time, where m is the number of arcs in the graph. The computation can also be made incremental in the same way as proposed by Pesant [2004].

A similar filtering algorithm for the variable-based `soft-regular` constraint was proposed by Beldiceanu, Carlsson, and Petit [2004]. That filtering algorithm does not necessarily achieve domain consistency, however.

Edit-Based Violation Measure

For the edit-based `soft-regular` constraint, we add the following violation arcs to the graph $\mathscr{R}$ representing the `regular` constraint. As in the previous section, we add A_{sub} to allow the substitution of a value. To allow deletions and insertions, we add violation arcs

$$A_{\text{del}} = \{(q_k^i, q_k^{i+1}) \mid i = 1, \dots, n\}\} \setminus A$$

$$\text{and} \quad A_{\text{ins}} = \{(q_k^i, q_l^i) \mid \delta(q_k, d) = q_l \text{ for some } d \in \Sigma, k \neq l, i = 1, \dots, n+1\}.$$

We extend the cost function w of the previous section such that $w(a) = 1$ if $a \in A_{\text{del}}$ or $a \in A_{\text{ins}}$. Let the resulting digraph be denoted by $\mathscr{R}_{\text{edit}}$ (see Figure 6.5 for an illustration on Example 6).

Deletions are modeled with the arcs introduced in A_{del}, which link equivalent states of successive layers. The intuition is that by using such an arc it is possible to remain at a given state and simply ignore the value taken by the corresponding variable. The arcs in A_{ins} allow a path to make more than one transition at any given layer. Since a layer corresponds to a variable and a transition is made on a symbol of the string, this is equivalent to *inserting* one or more symbols. Of course one has to make sure only to allow transitions defined by the automaton.

Theorem 7. [van Hoeve et al., 2006a] *The constraint* `soft-regular`$(X, M, z, \mu_{\text{edit}})$ *is domain consistent if and only if*

i) *every arc* $a \in A_1 \cup \cdots \cup A_n$ *belongs to an s-t path* P *in* $\mathscr{R}_{\text{edit}}$ *with* $\text{weight}(P) \leq \max D(z)$*, and*

ii) $\min D(z) \geq \text{weight}(P)$ *for a minimum-weight s-t path* P *in* $\mathscr{R}_{\text{edit}}$.

For the filtering algorithm, we proceed slightly different from the variable-based `soft-regular` constraint because the structure of the graph is not the same: Arcs within a layer may form (positive weight) directed circuits. We compute once and for all the smallest cumulative weight to go from q_k^i to q_l^i for every pair of nodes and record it in a table. This can be done through breadth-first-search from each node since every arc considered has unit weight. Notice that every layer has the same "insertion" arcs — we may preprocess one layer and use the result for all of them. In all, this initial step requires $\Theta(|Q|\,|\delta|)$ time. Then we can proceed as before through topological sort with table lookups, in $O(n\,|\delta|)$ time. The overall time complexity is therefore $O((n+|Q|)\,|\delta|) = O(m)$, where m is the number of arcs in the graph. The last step follows from $|Q| \leq n$, because otherwise some states would be unreachable.

6.4.4 *Other Soft Global Constraints*

We next present, in brief, a comprehensive[3] overview of other soft global constraints that have appeared in the literature.

Soft Cumulative Constraint

The `cumulative` constraint can be applied to model and solve resource constraints that appear for example in scheduling and packing problems [Aggoun and Beldiceanu, 1993]. It is defined on a set of 'activities', each of which has an associated variable representing the starting time, a given fixed duration, a given fixed time window in which it can be executed, and a given fixed amount of resource consumption. For example, when scheduling jobs on machines such that any two jobs cannot overlap, jobs correspond to activities, machines represent the (unary) resource, and each job has a unary resource consumption.

For the restricted version of the `cumulative` constraint on unary resources, Baptiste, Le Pape, and Péridy [1998] consider the soft version in which the number of late activities (i.e., that are completed after their associated deadline) is to be minimized. They provide a filtering algorithm that is able to identify that some activities must be on time, while others must be late. It is the first soft global constraint with an associated filtering algorithm reported in the literature.

[3] Comprehensive to the best of our knowledge.

Petit and Poder [2008] propose a version of the `soft-cumulative` constraint that aims to minimize the amount of over-load of the resource, while enforcing the time windows for the activities as hard constraints. They present a filtering algorithm for the variable-based violation measure on this constraint. Petit and Poder [2008] also provide an experimental comparison between their soft global `cumulative` constraint and the Valued-CSP approach (see Section 6.1.1) on over-constrained scheduling problems, showing the computational advantage of the soft `cumulative` constraint.

Soft Precedence Constraint

Lesaint, Mehta, O'Sullivan, Quesada, and Wilson [2009] introduce the `soft-precedence` constraint. It groups together hard precedence constraints and (weighted) soft precedence constraints among certain objects. In the telecommunication application that motivates their work, the objects correspond to features in a call-control feature subscription configuration problem. The `soft-precedence` constraint states that all hard precedence constraints be respected, while the total weight of respected soft precedence constraints is equal to a given value. Achieving domain consistency on `soft-precedence` is NP-hard, and therefore Lesaint et al. [2009] propose filtering rules based on lower and upper bounds to the problem.

Soft Constraints for a Timetabling Application

Cambazard, Hebrard, O'Sullivan, and Papadopoulos [2008] present three soft global constraints that are applied to solve a particular problem class from the 2007 International Timetabling competition. The three soft global constraints are problem-specific; their purpose is to derive and exploit good bounds for this particular problem class.

Soft Balancing Constraints

Balancing constraints appear in many combinatorial problems, such as fairly distributing workloads (or shifts) over employees, or generating spatially balanced experimental designs. Because a perfect balance is generally not possible, it is natural to soften the balancing constraint and minimize the induced cost of violation, as proposed by [Schaus, 2009], following earlier work by Pesant and Régin [2005].

For a set of variables $X = \{x_1, \ldots, x_n\}$ and a given fixed sum s, Schaus [2009] defines as a measure of violation for the balancing constraint the L_p-norm of $(X - s/n)$, assuming that $\sum_{i=1}^{n} x_i = s$. The L_p-norm of $(X - s/n)$ is defined as

$$\|X - s/n\|_p = \left(\sum_{i=1}^{n} |x_i - s/n| \right)^{\frac{1}{p}},$$

with $p \geq 0$. Schaus [2009] then introduces the constraint soft-balance(X,s,z,L_p) that holds if and only if $\sum_{i=1}^{n} x_i = s$ and $\|X - s/n\|_p \leq z$.

Different values of p lead to different realizations of the soft-balance constraint. For example, for L_0, we measure the number of different values from the mean, while for L_1, we sum the deviations from the mean. For L_2, we sum the squared deviations from the mean (which is equivalent to the variance). Finally, for L_∞, we measure the maximum deviation from the mean.

When the L_1-norm is applied, the resulting soft-balance constraint corresponds to the deviation constraint introduced by Schaus, Deville, Dupont, and Régin [2007b]. Bound consistency filtering algorithms for the deviation constraint were given by Schaus, Deville, and Dupont [2007a].

When the L_2-norm is applied, the resulting soft-balance constraint corresponds to the spread constraint, introduced by Pesant and Régin [2005]. The spread constraint is more general however, as it allows to represent the mean and standard deviation as (continuous) variables. Pesant and Régin [2005] also provide filtering algorithms for the spread constraint.

Soft Same Constraint

The same constraint is defined on two sequences of variables of equal length and states that the variables in one sequence use the same values as the variables in the other sequence. It can be applied to timetabling problems and pairing problems. van Hoeve, Pesant, and Rousseau [2006a] present a domain consistency filtering algorithm for the variable-based soft same constraint. Similar to the algorithms for the decomposition-based soft-alldifferent and soft-regular constraints presented before, it is based on the addition of "violation arcs" to a network flow representation of the problem.

Soft All-Equal Constraint

The ALLEQUAL constraint states that a given set of variables should all be assigned an equal value. The soft-ALLEQUAL constraint was introduced by Hebrard, O'Sullivan, and Razgon [2008] as the inverse of the (decomposition-based) soft-alldifferent constraint. Hebrard et al. [2008] show that finding a solution to the decomposition-based soft-ALLEQUAL constraint is NP-complete. Therefore, they propose to filter the constraint using an approximation algorithm, which can be implemented to run in linear amortized time.

Hebrard, Marx, O'Sullivan, and Razgon [2009] study the relationship between the soft-ALLEQUAL constraint and the soft-alldifferent constraint in more detail. They consider variants of the two constraints by combining the variable-based violation measure and the decomposition-based violation measure with the minimization objective and the maximization objective, respectively. In particular, they show that bounds consistency on the minimization-version of the decomposition-based `soft-`ALLEQUAL constraint can be established in polynomial time.

A related soft global constraint, named SIMILAR, was proposed by Hebrard, O'Sullivan, and Walsh [2007] to bound similarities between (partial) solutions, for example based on the Hamming distance.

Soft Sequence Constraint

The `sequence` constraint was introduced as a global constraint by Beldiceanu and Contejean [1994]. It is defined on an ordered sequence of variables X, a fixed number q, a fixed set of domain values S, and fixed lower and upper bounds l and u. It states that for every subsequence of q consecutive variables, the number of variables taking a value from S must be between l and u. The `sequence` constraint can be applied to model problems such as car sequencing or nurse rostering [van Hoeve, Pesant, Rousseau, and Sabharwal, 2006b, 2009].

The `soft-sequence` constraint was studied by Maher, Narodytska, Quimper, and Walsh [2008]. For each subsequence of q consecutive variables, they apply a violation measure that represents the deviation from the lower bound l or upper bound u. The violation measure for the `soft-sequence` is the sum of the violations for all subsequences. Maher et al. [2008] present a domain consistency filtering algorithm for this `soft-sequence` constraint, based on a particular minimum-weight network flow representation.

Soft Slide Constraint

The `slide` constraint was introduced by Bessiere, Hebrard, Hnich, Kiziltan, and Walsh [2008]. It is an extension of the `sequence` constraint, as well as a special case of the `cardinality_path` constraint [Beldiceanu and Carlsson, 2001]. The `slide` constraint allows to "slide" any constraint over an ordered sequence of variables, similar to the `sequence` constraint. Additionally, it allows to slide the particular constraint over more than one sequence of variables. Bessiere et al. [2007] show how the edit-based and variable-based `soft-slide` constraints can be reformulated in terms of hard `slide` constraints using sequences of additional variables. The `slide` constraint can similarly be applied to encode the variable-based and edit-based `soft-regular` constraints.

Soft Context-Free Grammar Constraint

The context-free grammar constraint (CFG) is an extension of the `regular` constraint; it restricts an ordered sequence of variables to belong to a context-free grammar [Sellmann, 2006; Quimper and Walsh, 2006]. The `soft-CFG` constraint was presented by Katsirelos, Narodytska, and Walsh [2008] as a special case of the weighted context-free grammar constraint. They propose domain consistency filtering algorithms for the variable-based (or Hamming-based) and edit-based versions of the `soft-CFG` constraint.

Σ-Alldifferent, Σ-Gcc, and Σ-Regular Constraints

The Σ-`alldifferent` constraint was introduced by Métivier, Boizumault, and Loudni [2007] as a variation of the `soft-alldifferent` constraint. In the `soft-alldifferent` constraint as discussed in Section 6.4.1, all variables and all not-equal constraints are equally important. In order to be able to model preferences among variables and constraints, the Σ-`alldifferent` constraint allows to associate a weight to variables and not-equal constraints. These weights have to be taken into account when evaluating the amount of violation of the constraint.

For the variable-based Σ-`alldifferent` constraint, a weight is associated to each variable, and the goal is to find an assignment whose total weighted violation is within the allowed bound defined by the cost variable. Métivier et al. [2007] present a domain consistency filtering algorithm based on a weighted network flow representation.

Similarly, for the decomposition-based Σ-`alldifferent` constraint, a weight is associated to each not-equal constraint. For this constraint, achieving domain consistency is NP-hard, however. Therefore, Métivier et al. [2007] propose filtering algorithms based on relaxations of the constraint.

Métivier, Boizumault, and Loudni [2009a] present a filtering algorithm for the decomposition-based `soft-gcc` with preferences (the Σ-`gcc` constraint), and for a distance-based `soft-regular` constraint with preferences (the Σ-`regular` constraint). The Σ-`gcc` and other (soft) global constraints are applied by Métivier, Boizumault, and Loudni [2009b] to model and solve nurse rostering problems.

Soft Global Constraints for Weighted CSPs

Lee and Leung [2009] consider soft global constraints in the context of the weighted CSP framework, where costs are associated to the tuples of variable assignments (see Section 6.1.1). In particular, Lee and Leung [2009] study the extension of the flow-based soft global constraints of van Hoeve et al. [2006a] in a weighted CSP setting. They show that the direct application of the flow-based filtering algorithms of van Hoeve et al. [2006a] can enforce so-called $\varnothing$-inverse consistency in weighted

CSPs. Lee and Leung [2009] further show how the modify the flow-based algorithms to achieve stronger forms of consistency in weighted CSPs.

Soft Global Constraints for Preference Modeling

Joseph, Chan, Hiroux, and Weil [2007] study soft global constraints for preference modeling in the context of multi-criteria decision support and social choice theory. Their underlying model applies several objective functions and binary preference relations. They apply soft global constraints to build hierarchical preference models.

Global Constraint for Max-CSP

The Max-CSP framework aims to maximize the number of satisfied constraints, or equivalently minimize the number of violated constraints (see also Section 6.1.1). Because Max-CSP problems can occur as a subproblems of real-world applications, Régin, Petit, Bessière, and Puget [2000, 2001] propose to encapsulate the Max-CSP problem as a single global constraint, which can be applied as a soft global constraint. Régin et al. [2000] propose a filtering algorithm based on a lower bound on the number of constraint violations, for example using 'conflict sets'. A conflict set is a set of constraints that leads to a contradiction. For example, the set of constraints $\{x < y, y < z, z < x\}$ is a conflict set, and we can infer that at least one constraint in this set must be violated in any solution. Régin et al. [2001] provide new lower bounds based on conflict-sets, where the constraints in the Max-CSP subproblem can be of any arity.

Soft Open Global Constraints

Traditionally, a (global) constraint has a fixed scope of variables on which it is defined. Many practical applications require the scope of a constraint to be less rigid however. For example, suppose we need to execute a set of activities on different machines, such that on each machine no two activities overlap. Assuming unit processing times, we can model the non-overlapping requirement using an `alldifferent` constraint on the starting time variables of the activities for each machine. However, the scope of each such `alldifferent` constraint is unknown until we have assigned the activities to the machines. Constraints of this nature are called *open* constraints [Faltings and Macho-Gonzalez, 2002; Barták, 2003; van Hoeve and Régin, 2006]. During the search for a solution, variables can be added to, or removed from, the scope of an open constraint dynamically.

Maher [2009a] considers soft open global constraints, and investigates when a filtering algorithm for the closed version of a constraint is sound for the open version. The property of *contractibility* introduced by Maher [2009b] can be used for this purpose. Maher [2009a] shows that the contractibility of a soft constraint is independent on the contractibility of the associated hard constraint, and relies solely on the violation measure that is applied. He further shows that the decomposition-based violation measure and various versions of the edit-based violation measure lead to contractible soft open global constraints. For such soft open global constraints, one can therefore safely apply the existing filtering algorithm for the closed version of the constraint in an open setting. Maher [2009a] presents a corresponding filtering algorithm for the open `soft-regular` constraint under a weighted edit-based violation measure, building on the existing algorithm for the `soft-regular` constraint by van Hoeve et al. [2006a].

6.5 Constraint-Based Local Search

Local search methods provide an alternative to complete systematic search methods (such as constraint programming) for solving combinatorial problems [Aarts and Lenstra, 2003; Van Hentenryck and Michel, 2005]. Conceptually, local search iteratively moves from one solution to a neighboring one, with the aim of improving the objective function. Therefore, local search algorithm are based on a definition of a neighborhood and cost evaluation functions. In many cases, local search can quickly find solutions of good quality, but in general it is not able to prove optimality of a solution. Local search is a natural approach to solve over-constrained problems, where the objective is to minimize some specified measure of violation of the problem.

In the literature, local search algorithms have been largely described using low-level concepts close to the actual computer implementation. The first modeling language for local search was Localizer [Michel and Van Hentenryck, 1997, 2000], which offered a generic and re-usable way of implementing different local search methods. In *constraint-based* local search, the aim is to model the problem at hand using constraints and objectives, to which then any (suitable) local search can be applied. One of the earliest of such general approaches was developed by Galinier and Hao [2000], see also [Galinier and Hao, 2004]. In that work, a library of constraints is presented that can be used to model a problem. To each constraint, a penalty function is associated that is used in the evaluation function of the Tabu Search engine underlying the system. A similar approach was taken by Michel and Van Hentenryck [2002] for the system *Comet*, and by Bohlin [2004, 2005] for the system *Composer*.

Essential to constraint-based local search is that the solution method can be derived from the constraints of the problem. That is, the definition of neighborhoods as well as the evaluation functions can be based on the combinatorial properties of the constraints. Also global constraints can be used for this purpose. For example, Nareyek [2001] applies global constraints to define improvement heuristics for scheduling problems.

The evaluation functions (or penalty functions) for constraints in local search are closely related to the violation measures for soft global constraints in constraint programming. For example, in the system *Comet*, to each constraint a measure of violation is associated similar to those that are used to define soft global constraints. In constraint-based local search, the violation measures play a different role, however. Instead of filtering variable domains, they are applied to compute a "gradient" with respect to the violation measure. That is, for each variable-value pair, we can define the additional amount of violation if we were to assign this variable to the value.

In the context of constraint-based local search, Van Hentenryck and Michel [2005] present violation measures for several global constraints, including `all-different`, `atmost`, `atleast`, multi-knapsack, `sequence`, systems of not-equal constraints, and arbitrary (weighted) constraint systems.

6.6 Conclusion and Outlook

In this chapter we have presented an overview of techniques to handle over-constrained problems in constraint programming. The main focus has been on recent developments in the area of soft global constraints. Starting from initial works by Baptiste, Le Pape, and Péridy [1998], Petit, Régin, and Bessière [2000], and especially Petit, Régin, and Bessière [2001], the field of soft global constraint has developed into a mature and established approach to modeling and solving over-constrained problems in constraint programming.

We have presented detailed filtering algorithms for the `soft-alldifferent` constraint, the `soft-gcc` constraint, and the `soft-regular` constraint. In addition, we have given a comprehensive overview of other soft global constraints that have been studied in the literature. The techniques for handling soft global constraints are often based on methods from graph theory, network flows, and regular languages, which reflect the synergy between constraint programming, operations research, and artificial intelligence; the focus of this collection.

Several soft global constraints that appeared in the literature have been applied successfully to solve practical (over-constrained) problems, as we have seen in Section 6.4.4. Nevertheless, so far no commercial constraint programming solver offers soft global constraints as part of their product. Given the increasing interest of the research community as well as the growing number of successful applications, it would be highly desirable if soft global constraints were added to these commercial solvers.

Many research challenges remain in this area. Perhaps the most important one is the issue of aggregating effectively different soft global constraints. It is likely that a weighted sum of the associated cost variables is not the most effective aggregation. Other approaches, such as minimizing the maximum over all cost variables, or applying a (soft) balancing constraint to the cost variables [Schaus, 2009], appear to be more promising.

Finally, as was discussed in Section 6.5, the violation measures for soft global constraints are closely related to constraint-based evaluation functions in local search. Therefore, integrating local search and constraint programming based on soft global constraints appears to be an interesting and promising avenue for future research.

Acknowledgements As parts of this chapter are based on the paper [van Hoeve, Pesant, and Rousseau, 2006a], I wish to thank Gilles Pesant and Louis-Martin Rousseau.

References

E. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

A. Aggoun and N. Beldiceanu. Extending CHIP in order to Solve Complex Scheduling and Placement Problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

K.R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

P. Baptiste, C. Le Pape, and L. Péridy. Global Constraints for Partial CSPs: A Case-Study of Resource and Due Date Constraints. In M.J. Maher and J.-F. Puget, editors, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1520 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 1998.

R. Barták. Dynamic Global Constraints in Backtracking Based Environments. *Annals of Operations Research*, 118(1–4):101–119, 2003.

N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. In R. Dechter, editor, *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1894 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2000.

N. Beldiceanu and M. Carlsson. Revisiting the Cardinality Operator and Introducing the Cardinality-Path Constraint Family. In P. Codognet, editor, *Proceedings of the 17th International Conference on Logic Programming (ICLP)*, volume 2237 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2001.

N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994.

N. Beldiceanu and T. Petit. Cost Evaluation of Soft Global Constraints. In J.-C. Régin and M. Rueher, editors, *Proceedings of the First International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 3011 of *Lecture Notes in Computer Science*, pages 80–95. Springer, 2004.

N. Beldiceanu, M. Carlsson, and T. Petit. Deriving Filtering Algorithms from Constraint Checkers. In M. Wallace, editor, *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3258 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2004.

C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, C.-G. Quimper, and T. Walsh. Reformulating Global Constraints: The Slide and Regular Constraints. In I. Miguel and W. Ruml, editors, *Proceedings of 7th International Symposium on Abstraction, Reformulation, and Approximation (SARA)*, volume 4612 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2007.

C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. SLIDE: A Useful Special Case of the CARDPATH Constraint. In M. Ghallab, C.D. Spyropoulos, N. Fakotakis, and N.M. Avouris, editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, pages 475–479. IOS Press, 2008.

S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Satisfaction and Optimization. *Journal of the ACM*, 44(2):201–236, 1997.

M. Bohlin. *Design and Implementation of a Graph-Based Constraint Constraint Model for Local Search*. PhD thesis, Mälardalen University, 2004. Licentiate Thesis No.27.

M. Bohlin. A Local Search System for Solving Constraint Problems. In D. Seipel, M. Hanus, U. Geske, and O. Bartenstein, editors, *Applications of Declarative Programming and Knowledge Management*, volume 3392 of *Lecture Notes in Artificial Intelligence*, pages 166–184. Springer, 2005.

A. Borning, R. Duisberg, B. Freeman-Benson, A. Kramer, and M. Woolf. Constraint hierarchies. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 48–60, 1987.

H. Cambazard, E. Hebrard, B. O'Sullivan, and A. Papadopoulos. Local Search and Constraint Programming for the Post Enrolment-based Course Timetabling Problem. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 2008.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms - Second edition*. The MIT Press, 2001.

R. Dechter. On the expressiveness of networks with hidden variables. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI)*, pages 555–562. AAAI Press/The MIT Press, 1990.

R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, volume 2, pages 1131–1136, 1993.

B. Faltings and S. Macho-Gonzalez. Open Constraint Satisfaction. In P. Van Hentenryck, editor, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, volume 2470 of *Lecture Notes in Computer Science*, pages 356–370. Springer, 2002.

H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proceedings of the first European Congress on Fuzzy and Intelligent Technologies*, 1993.

F. Focacci, A. Lodi, and M. Milano. Optimization-oriented global constraints. *Constraints*, 7(3): 351–365, 2002.

E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3): 21–70, 1992.

P. Galinier and J.K. Hao. A General Approach for Constraint Solving by Local Search. In *Proceedings of the Second International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, 2000.

P. Galinier and J.K. Hao. A general approach for constraint solving by local search. *Journal of Mathematical Modelling and Algorithms*, 3(1):73–88, 2004.

E. Hebrard, B. O'Sullivan, and T. Walsh. Distance Constraints in Constraint Satisfaction. In M.M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 106–111, 2007. Available online at http://ijcai.org/.

E. Hebrard, B. O'Sullivan, and I. Razgon. A Soft Constraint of Equality: Complexity and Approximability. In P.J. Stuckey, editor, *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5202 of *Lecture Notes in Computer Science*, pages 358–371. Springer, 2008.

E. Hebrard, D. Marx, B. O'Sullivan, and I. Razgon. Constraints of Difference and Equality: A Complete Taxonomic Characterization. In I.P. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5732 of *Lecture Notes in Computer Science*, pages 424–438. Springer, 2009.

J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.

R.-R. Joseph, P. Chan, M. Hiroux, and G. Weil. Decision-support with preference constraints. *European Journal of Operational Research*, 177(3):1469–1494, 2007.

G. Katsirelos, N. Narodytska, and T. Walsh. The Weighted CFG Constraint. In L. Perron and M.A. Trick, editors, *Proceedings of the 5th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 5015 of *Lecture Notes in Computer Science*, pages 323–327. Springer, 2008.

J. Larrosa. Node and Arc Consistency in Weighted CSP. In R. Dechter, M. Kearns, and R. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 48–53. AAAI Press, 2002.

J. Larrosa and T. Schiex. In the quest of the best form of local consistency for Weighted CSP. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 239–244. Morgan Kaufmann, 2003.

J.-L. Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1):29–127, 1978.

J.H.M. Lee and K.L. Leung. Towards Efficient Consistency Enforcement for Global Constraints in Weighted Constraint Satisfaction. In C. Boutilier, editor, *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI)*, pages 559–565, 2009. Available online at http://ijcai.org/.

D. Lesaint, D. Mehta, B. O'Sullivan, L. Quesada, and N. Wilson. A Soft Global Precedence Constraint. In C. Boutilier, editor, *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI)*, pages 566–571, 2009. Available online at http://ijcai.org/.

M.J. Maher. SOGgy Constraints: Soft Open Global Constraints. In I.P. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5732 of *Lecture Notes in Computer Science*, pages 584–591. Springer, 2009a.

M.J. Maher. Open Contractible Global Constraints. In C. Boutilier, editor, *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI)*, pages 578–583, 2009b. Available online at http://ijcai.org/.

M.J. Maher, N. Narodytska, C.-G. Quimper, and T. Walsh. Flow-Based Propagators for the SEQUENCE and Related Global Constraints. In P.J. Stuckey, editor, *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5202 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2008.

J.-P. Métivier, P. Boizumault, and S. Loudni. Σ-AllDifferent: Softening AllDifferent in Weighted CSPs. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 223–230. IEEE, 2007.

J.-P. Métivier, P. Boizumault, and S. Loudni. Softening Gcc and Regular with preferences. In *Proceedings of the 2009 ACM symposium on Applied Computing (SAC)*, pages 1392–1396. ACM, 2009a.

J.-P. Métivier, P. Boizumault, and S. Loudni. Solving Nurse Rostering Problems Using Soft Global Constraints. In I.P. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5732 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2009b.

L. Michel and P. Van Hentenryck. A Constraint-Based Architecture for Local Search. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 101–110, 2002.

L. Michel and P. Van Hentenryck. Localizer. *Constraints*, 5:43–84, 2000.

L. Michel and P. Van Hentenryck. Localizer: A Modeling Language for Local Search. In G. Smolka, editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1330 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 1997.

A. Nareyek. Using global constraints for local search. In *Constraint Programming and Large Scale Discrete Optimization: DIMACS Workshop Constraint Programming and Large Scale Discrete Optimization, September 14-17, 1998, DIMACS Center*, volume 54 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 9–28. American Mathematical Society, 2001.

G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In M. Wallace, editor, *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.

G. Pesant and J.-C. Régin. Spread: A balancing constraint based on statistics. In P. van Beek, editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3709 of *Lecture Notes in Computer Science*, pages 460–474. Springer, 2005.

T. Petit. *Modélisation et Algorithmes de Résolution de Problèmes Sur-Contraints*. PhD thesis, Université Montpellier II, 2002. In French.

T. Petit and E. Poder. Global Propagation of Practicability Constraints. In L. Perron and M.A. Trick, editors, *Proceedings of the 5th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 5015 of *Lecture Notes in Computer Science*, pages 361–366. Springer, 2008.

T. Petit, J.-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. In *Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 358–365. IEEE, 2000.

T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithms for Over-Constrained Problems. In T. Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2239 of *Lecture Notes in Computer Science*, pages 451–463. Springer, 2001.

C.-G. Quimper and T. Walsh. Decomposing Global Grammar Constraints. In F. Benhamou, editor, *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4204 of *Lecture Notes in Computer Science*, pages 751–755, 2006.

C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. Improved Algorithms for the Global Cardinality Constraint. In M. Wallace, editor, *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3258 of *Lecture Notes in Computer Science*, pages 542–556. Springer, 2004.

J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 362–367. AAAI Press, 1994.

J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference (AAAI / IAAI)*, volume 1, pages 209–215. AAAI Press/The MIT Press, 1996.

J.-C. Régin. Global Constraints and Filtering Algorithms. In M. Milano, editor, *Constraint and Integer Programming - Toward a Unified Methodology*, volume 27 of *Operations Research/Computer Science Interfaces*, chapter 4. Kluwer Academic Publishers, 2003.

J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. An Original Constraint Based Approach for Solving over Constrained Problems. In R. Dechter, editor, *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1894 of *Lecture Notes in Computer Science*, pages 543–548. Springer, 2000.

J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. New Lower Bounds of Constraint Violations for Over-Constrained Problems. In T. Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2239 of *Lecture Notes in Computer Science*, pages 332–345. Springer, 2001.

F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI)*, pages 550–556, 1990.

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

Z. Ruttkay. Fuzzy constraint satisfaction. In *Proceedings of the First IEEE Conference on Evolutionary Computing*, pages 542–547, 1994.

P. Schaus. *Solving Balancing and Bin-Packing problems with Constraint Programming*. PhD thesis, Université catholique de Louvain, 2009.

P. Schaus, Y. Deville, and P. Dupont. Bound-Consistent Deviation Constraint. In C. Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4741 of *Lecture Notes in Computer Science*, pages 620–634. Springer, 2007a.

P. Schaus, Y. Deville, P. Dupont, and J.-C. Régin. The Deviation Constraint. In P. Van Hentenryck and L.A. Wolsey, editors, *Proceedings of the 4th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 4510 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2007b.

T. Schiex. Possibilistic Constraint Satisfaction Problems or "How to handle soft constraints?". In D. Dubois and M.P. Wellman, editors, *Proceedings of the Eighth Annual Conference on Uncertainty in Artificial Intelligence*, pages 268–275. Morgan Kaufmann, 1992.

T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–639. Morgan Kaufmann, 1995.

A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

M. Sellmann. The Theory of Grammar Constraints. In F. Benhamou, editor, *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4204 of *Lecture Notes in Computer Science*, pages 530–544, 2006.

R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. The MIT Press, 2005.

W.-J. van Hoeve. A Hyper-Arc Consistency Algorithm for the Soft Alldifferent Constraint. In M. Wallace, editor, *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3258 of *Lecture Notes in Computer Science*, pages 679–689. Springer, 2004.

W.-J. van Hoeve and I. Katriel. Global constraints. In F. Rossi, P. Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 6. Elsevier, 2006.

W-J. van Hoeve and J-C. Régin. Open constraints in a closed world. In J.C. Beck and B.M. Smith, editors, *Proceedings of the Third International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 3990 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2006.

W.-J. van Hoeve, G. Pesant, and L.-M. Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4):347–373, 2006a.

W.-J. van Hoeve, G. Pesant, L.-M. Rousseau, and A. Sabharwal. Revisiting the Sequence Constraint. In F. Benhamou, editor, *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4204 of *Lecture Notes in Computer Science*, pages 620–634, 2006b.

W.-J. van Hoeve, G. Pesant, L.-M. Rousseau, and A. Sabharwal. New Filtering Algorithms for Combinations of Among Constraints. *Constraints*, 14:273–292, 2009.

A. Zanarini, M. Milano, and G. Pesant. Improved Algorithm for the Soft Global Cardinality Constraint. In J.C. Beck and B.M. Smith, editors, *Proceedings of the Third International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 3990 of *Lecture Notes in Computer Science*, pages 288–299. Springer, 2006.

Index