Constraints, 8, 9–39, 2003

Current Approaches for Solving Over-Constrained Problems

PEDRO MESEGUER pedro@iiia.csic.es
IIIA-CSIC Campus UAB, 08193 Bellaterra, Spain

NOUREDDINE BOUHMALA noureddine.bouhmala@hive.no
Vesfold University College, N-3103 Tønsberg, Norway

TAOUFIK BOUZOUBAA tbo@treedalgo.co.ma
SGTM-TreeDAlgo 23 rue Massena, 20100 Casablanca, Maroc

MORTEN IRGENS morten.irgens@sintef.no
SINTEF Postboks 124 Blindern, N-0314 Oslo, Norway

MARTÍ SÁNCHEZ marti@iiia.csic.es
IIIA-CSIC Campus UAB, 08193 Bellaterra, Spain

Abstract. We summarize existing approaches to model and solve overconstrained problems. These problems are usually formulated as combinatorial optimization problems, and different specific and generic formalisms are discussed, including the special case of multi-objective optimization. Regarding solving methods, both systematic and local search approaches are considered. Finally we review a number of case studies on over-constrained problems taken from the specialized literature.

Keywords: over-constrained CSPs, soft constraints, multi-objective optimization

1. Over-Constrained CSPs

A *Constraint satisfaction problem* (CSP for short) is defined by a triple $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C})$ where,

— $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ is a set of variables.
— $\boldsymbol{D} = \{D_1, \ldots, D_n\}$ is a collection of finite domains, D_i is the set of possible values for X_i,
— $\boldsymbol{C} = \{C_1, \ldots, C_r\}$ is a set of constraints. A constraint C_i on the ordered set of variables $var(C_i)$ specifies the relation $rel(C_i)$ of the allowed combinations of values for the variables in $var(C_i)$; $rel(C_i)$ is a subset of the cartesian product of domains of variables in $var(C_i)$.

A *solution* of the CSP is a complete assignment of values to variables which satisfies every constraint. CSP solving is a NP-complete problem. A CSP can be seen as a *decision* problem, where the goal is to decide whether there exists a solution or not, by constructing such solution. Typically, a CSP is solved by a tree search procedure with backtracking. See [30, 52] for overviews of solving strategies.

While powerful, the CSP schema presents some limitations. In particular, all constraints are considered mandatory. In many real problems often appear constraints that could be violated in solutions without causing such solutions to be unacceptable. If these constraints are treated as mandatory, this often causes problems to be unsolvable. If these constraints are ignored, solutions of bad quality are found. This is the motivation to extend the CSP schema to include over-constrained problems. An *over-constrained* CSP (OCSP for short) is a CSP for that any complete assignment violates some constraint. A *solution* of the OCSP is the complete assignment that *best respects* the set of constraints. We differentiate among several constraint types. A constraint is *crisp* when it is either completely satisfied or completely violated. A constraint is *fuzzy* when it allows for intermediate satisfaction degrees. A constraint is *hard* when it should necessarily be satisfied by any solution. A constraint is *soft* or *relaxable* when it can be violated by some solution. In the CSP schema, all constraints are crisp and hard. In addition to these, the OCSP schema includes fuzzy and soft constraints. Several models for OCSP have been devised, each with different constraint representation and solution criterion. The most often used models in the literature are detailed in the following.

Fuzzy (also called *possibilistic*). Constraints are represented by fuzzy relations. The degree in which a tuple satisfies a fuzzy relation is given by a membership function on the interval [0, 1], where 1 means complete satisfaction and 0 complete violation. The aggregation of satisfaction degrees is made using the *min* operator. A solution is the complete assignment with maximum satisfaction degree on the least satisfied constraint [16, 19].

Lexicographical. The fuzzy model considers the degree of the least satisfied constraint only, no matter the number of constraints achieving this or higher levels of satisfaction. This approach selects, among equivalent solutions of the fuzzy model, the solution with the highest number of constraints satisfied at the level equal to (or higher than) the degree of the least satisfied constraint [16, 19].

Weighted. Each constraint is labelled with a weight or cost. The cost of a complete assignment is the addition of costs of unsatisfied constraints. A solution is a complete assignment with minima1 cost. A popular version of this model is the Max-CSP problem, where all constraints have the same weight [20].

Probabilistic. Each constraint is labelled with its probability of presence in the problem. A solution is the assignment that maximizes the probability to be a solution of the actual CSP [18].

Hierarchical. Constraints are divided in a hierarchy of levels, according to their relative importance. A complete assignment violates some constraints, causing some errors (consider for instance arithmetic errors) among which a partial order exists. A solution is the assignment that minimizes such errors, according to some selected comparator (local, regional or global) [8, 64].

Two generic algebraic frameworks have been proposed, to encompass most of the above models as particular cases. The *Valued CSP* model requires a valuation structure

$(E, \succ, \bot, \top, *)$, where E is the valuation set (the set of possible violation degrees for each constraint) totally ordered by $\succ$, $\bot$ and $\top$ are the minimum and maximum elements of E in the total order, and $*$ is the aggregation operation of valuations, closed on E [49]. The *Semiring CSP* model requires a c-semiring structure $(A, +, \times, \mathbf{0}, \mathbf{1})$, where A is the set of satisfaction degrees of a constraint tuple, $+$ and $\times$ are the semiring operations, and $\mathbf{0}, \mathbf{1}$ are the minimum and maximum values of A with the partial order [6]. Both models are closely related, being equivalent when the valuation set is totally ordered. Otherwise, the semiring model is more general [7].

Solving a OCSP is an *optimization* task. This problem is NP-hard, being harder to solve than classical CSP. If constraint violations can be aggregated into a sigle function, OCSP solving is formulated as single-objective optimization task. Otherwise, it becomes a multi-criteria optimization, for which specific strategies have been developed.

Considering OCSP solving methods, there are two main approaches. *Systematic search* methods perform an ordered traversal of the state space, visiting every state which could contain a solution. These methods are optimal and provide the best solution for the problem. *Local search* methods perform search in some subspaces of whole state space, looking for a good solution within some time limit. They are suboptimal, since they do not guarantee to find the best solution.

This paper is organized as follows. In Section 2, we present specific and generic models used to formulate an OCSP, including a multi-objective optimization formulation. In Section 3, we describe different methods for OCSP solving. In Section 4, we discuss some case studies of specific OCSP. Finally, in Section 5 we summarize the paper, pointing out promising directions for further research.

2. OCSP Modelling and Formulation

2.1. Specific Models For OCSPs

2.1.1 Fuzzy Model

A fuzzy CSP [16, 19] is defined as a CSP where constraints are represented by fuzzy relations. In this model constraint satisfaction becomes a matter of degree. The simplest constraint types in the fuzzy model are detailed in the following.

Fuzzy Constraints. A fuzzy constraint C is represented by the fuzzy relation R, defined by,

$$\mu_R : \prod_{x_i \in var(C)} D_i \mapsto [0, 1]$$

where μ_R is the membership function indicating to what extent a tuple v satisfies C,

— $\mu_R(v) = 1$ means v totally satisfies C,
— $\mu_R(v) = 0$ means v totally violates C,
— $0 < \mu_R(v) < 1$ means that v partially satisfies C.

Obviously, crisp constraints are included in the model, involving values 0 and 1 only.

Prioritized Constraints. The priority degree $pr(C)$ of a crisp constraint C indicates to what extent it is required for C to be satisfied,

— $pr(C) = 1$ means that C is hard (only assignments satisfying C could be part of a solution)

— $pr(C) = 0$ means that C can be ignored (any assignment satisfies C)

The priority of a constraint can also be interpreted as the extent to which the constraint can be violated. Assuming that any tuple satisfies the prioritized constraint from at least degree $1 - pr(C)$, a prioritized constraint can be represented by a fuzzy relation R,

— $\mu_R(v) = 1$, if v satisfies C

— $\mu_R(v) = 1 - pr(C)$, if v does not satisfy C

If R and R' are two fuzzy relations representing constraints C and C', the following operations hold.

Projection. The projection of a fuzzy relation R on a set of variables Y, $var(C) \supseteq Y$, is a new fuzzy relation $R^{\downarrow Y}$ defined as follows,

$$\mu_{R^{\downarrow Y}}(v) = sup_w(\mu_R(w)) \qquad v \in \prod_{i \in Y} D_i, w \in \prod_{i \in var(C_i)} D_i, w^{\downarrow Y} = v$$

Conjunctive Combination. The conjunctive combination of two fuzzy relations R and R' is a new fuzzy relation $R \otimes R'$ defined as,

$$\mu_{R \otimes R'}(v) = min(\mu_R(v^{\downarrow var(C)}), (\mu_{R'}(v^{\downarrow var(C')}))) \qquad v \in \prod_{i \in var(C) \cup var(C')} D_i$$

The consistency degree of an assignment is defined as the conjunctive combination of all fuzzy relations corresponding to those constraints with all its variables instantiated. A solution is a complete assignment with a maximal consistency degree, that is,

$$max_v(min_k(\mu_{R_k}(v^{\downarrow var(C_k)}))) \qquad v \in \prod_{i=1, \ldots, n} D_i$$

A fuzzy CSP with p different levels of satisfaction is equivalent to p CSPs: for each level α, there is one CSP with constraints $\{C_1^\alpha, \ldots, C_r^\alpha\}$, where C_i^α is the hard constraint satisfied by the tuples satisfying C_i with a degree higher or equal to α. From this view, it is easy to see that local consistency concepts and algorithms can be adapted to fuzzy CSPs. In particular, arc consistency for binary fuzzy CSPs is achieved when,

$$R_i \subseteq [R_{ij} \otimes R_j]^{\downarrow \{X_i\}}$$

where R_i, R_j and R_{ij} are the fuzzy relations modelling unary constraints and the binary constraint between X_i and X_j.

2.1.2 Lexicographical Model

The fuzzy model causes a too coarse solution generation because it considers the least satisfied constraint only. The lexicographical model [16, 19] is a refinement of this, discriminating among assignments with the same value for their least satisfied constraint. The consistencỹ of a complete assignment v with respect to problem constraints is defined by a vector of satisfaction degrees $SD(v)$,

$$[SD(v)]_k = \mu_{R_k}(\mu(v^{\downarrow var(C_k)})$$

$SD(v)$ has r components, one for each constraint. Component k contains the satisfaction degree of C_k by the assignment v. Vectors with the same global satisfaction degree are ranked by increasing lexicographical order. Given two assignments, v and w, such that $SD(v) = (v_1, v_2, \ldots, v_r)$ and $SD(w) = (w_1, w_2, \ldots, w_r)$, its lexicographical ordering is defined as follows: (1) rearrange vectors in increasing order, say $v_{i_1} \leq v_{i_2} \leq \cdots \leq v_{i_r}$, and $w_{i_1} \leq w_{i_2} \leq \cdots \leq w_{i_r}$, (2) perform a lexicographical comparison starting from the leftmost component,

$$SD(v) > SD(w) \text{ iff } \exists k \leq r, \forall m < k, v_{i_m} = w_{i_m} \text{ and } v_{i_k} > w_{i_k}$$

The preferred solution is the complete assignment with a maximal vector of satisfaction degrees.

2.1.3 Weighted Model

The weighted model considers crisp constraints. Each constraint C_i is labelled with a weight w_i, which represents the cost (or penalty) that exists if C_i is violated. We define the cost of C_i under the assignment v as,

— $cost(C_i(v)) = 0$, if C_i is satisfied by v

— $cost(C_i(v)) = w_i$, if C_i is violated by v

The cost of an assignment is the addition of costs of all constraints instantiated by that assignment. The solution is the complete assignment with minimum cost, that is,

$$min_v \sum_i cost(C_i(v)), \qquad v \in \prod_{i=1,\ldots,n} D_i$$

A particular version of this model is the Max-CSP problem [20], where all constraints have the same weight, and the solution is the assignment satisfying as many constraints as possible.

2.1.4 Probabilistic Model

The probabilistic model [18] allows representing those situations where there is a partial knowledge of the constraints that will be present. Each constraint C is labelled with a

probability of presence, $p(C)$, assumed independent of the presence of other constraints. In this context, we talk about the real problem, the problem that actually occurs where a constraint either occurs or not, to differentiate from the model where constraints have probabilities associated. The solution is the assignment with maximum probability to be a solution of the real problem. The probability that an assignment v is solution is the probability that all constraints violated by v are not present in the real problem. This is product of all $\mathbf{1} - p(C)$, for all C violated by v. Then, the solution of the probabilistic model is the following maximization problem,

$$max_v \prod_i (1 - p(C_i)) \qquad v \in \prod_{j=1,\ldots,n} D_j, C_i \text{ is violated by } v$$

2.1.5 Hierarchical Model

The hierarchical model (also called constraint hierarchies) has been proposed to describe over-constrained systems with a hierarchy of constraint preferences [8, 64]. This model considers crisp constraints only. A constraint hierarchy H is a finite collection of constraints labelled with a level of strength or preference. H_0 is the set of hard constraints. H_1 is the set of constraints with the following preference level. In the same way, we define $H_2, \ldots, H_n$, constraints associated with decreasing preference levels $2, \ldots, n$, respectively. A solution of a constraint hierarchy H is an assignment of values to variables in H which satisfies all constraints in H_0 and best satisfies the other constraints respecting the hierarchy. We define the sets,

$$S_{H_0} = \{v | v \text{ satisfies all constraints in } H_0\}$$
$$S_H = \{v | v \in S_{H_0}, \forall w \in S_{H_0}, w \text{ is not better than } v \text{ respecting } H\}$$

The set S_{H_0} contains assignment satisfying hard constraints in H. The set S_H, called the solution set for H, is a subset of S_{H_0} containing all assignments which are not worse than other assignments in S_{H_0}. Given an assignment v and a constraint C, the error of v in C is a positive real number indicating the magnitude of violation of C by v. When C is totally satisfied, the error of v is zero. Different criteria for the predicate better in the above definition of solution set are called comparators, based on assignment errors. A comparator is an irreflexive and transitive relation over assignments of variables that respects the hierarchy, i.e., if there is some assignment in S_{H_0} that completely satisfies all the constraints through level k, then all assignments in S_H must satisfy all the constraints through level k. Note that the comparator defines a partial order over assignments. Currently, there are three widely used groups of comparators,

— *Locally-better.* A locally-better comparator considers each constraint individually. An assignment v is *locally-better* than another w if, for each of the constraints through some level $k-1$, the errors are equal for respective assignments, and at level k the error is strictly less for at least one constraint and less than or equal to for all the rest.

— *Regionally-better.* A regionally-better comparator allows the comparison of assignments which are incomparable by locally-better comparators. An assignment v is *regionally-better* than another assignment w if, for each level till $k-1$, the levels are incomparable by a locally-better comparator, and at level k the error is strictly less for at least one constraint and less than or equal to for all the rest.

— *Globally-better.* A globally-better comparator aggregates errors of individual constraints at each level. An assignment v is *globally-better* than another w if, the combined errors are equal till some level $k-1$ for both assignments, and at level k the combined error is strictly less for the valuation v. There is a number of reasonable candidates for the combining function that combines error of individual constraints at each level, namely weighted-sum, worst-case or least-squares methods.

2.2. Generic Models for OCSPs

2.2.1 Valued Constraint Satisfaction Problems

A *valuation structure* [49] is a quintuple $(E, \succ, \bot, \top, *)$, where E is the valuation set, $\succ$ is a total order on E, $\top$ and $\bot$ are the maximum and minimum elements in E, and $*$ is a binary closed operation on E satisfying the following properties,

— Associative: $(a * b) * c = a * (b * c)$

— Commutative: $a * b = b * a$

— Monotonicity: $a \succ b \Rightarrow (a * c \succ b * c)$

— Identity: $a * \bot = a$

— Absorbing element: $a * \top = \top$

where $a, b, c \in E$. A *valued* CSP is defined by a classical CSP $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C})$, a valuation structure $(E, \succ, \bot, \top, *)$, and an application $\phi : \boldsymbol{C} \mapsto E$. The valuation set E is used to define a gradual notion of constraint violation and inconsistency. The elements of E can be compared using the total order $\succ$ and aggregated using the operator $*$. The maximum element $\top$ is used to express complete inconsistency, while the minimum element $\bot$ expresses complete consistency. The valuation function ϕ associates with each constraint a valuation which denotes its importance. The valuation of hard constraints equals $\top$. If v is an assignment of problem variables and $C_{unsat}(v)$ is the set of constraints unsatisfied by v, the valuation of v is the aggregation of valuations of all constraints in $C_{unsat}(v)$,

$$\phi(v) = \underset{C \in C_{unsat}(v)}{*} \phi(C)$$

The goal is to find an assignment with valuation lower than $\top$ and minimum. The valuation of the problem is the valuation of such assignment. Specific models can be defined by instantiating the valuation structure. This approach is denoted as VCSP, and it includes the following specific models of OCSP.

Classical CSP Model. All constraints are hard, $E = \{true, false\}$, $false > true$, $\top = false$, $\bot = true$, $*$ is logical *and*. The goal is to find an assignment satisfying all constraints.

Fuzzy Model. $E = [0, 1]$, $1 > 0$, $\top = 1$, $\bot = 0$, $*$ is *max*. Constraint valuations are considered as constraint priorities. Valuations are aggregated according to the most important violated constraint.

Lexicographical Model. E is a multiset of elements of $[0, 1]$. The operation $*$ is simply multiset union, extended to treat $\top$ as absorbing element. The empty multiset is the identity $\bot$. The order $>$ is the lexicographic total order induced by the standard order on multisets and extended to give $\top$ its role of maximum element: let v and v' be two multisets and a and a' be the largest elements in v and v', $v > v'$ iff either $a > a'$ or $(a = a'$ and $v - \{a\} > v' - \{a'\})$. The recursion ends on Ø, the minimum multiset.

Weighted Model. E is $N \cup \{\infty\}$. The operation $*$ is $+$, $\top = \infty$, $\bot = 0$. The goal is to find an assignment with minimum addition of weights. When all weights are equal to 1, we get the Max-CSP problem.

Probabilistic Model. $E = [0, 1]$, $\top = 1$, $\bot = 0$. The valuation of a constraint is its probability of existence. The operation is $x * y = 1 - (1 - x)(1 - y)$ (x and y are supposed to be independent probabilities).

One of the mayor results of the VCSP framework relates the idempotency of the $*$ operator ($*$ is idempotent iff $a * a = a, \forall a \in E$) and the applicability of classical local consistency methods in CSP. In the VCSP framework, if the operator is idempotent, classical local consistency concepts and algorithms can be easily adapted to deal with a valuation structure. However, if the operator is not idempotent, adding induced constraints is not possible: this may modify the valuation of assignments, and the resulting problem may not be equivalent to the original one. For classical and fuzzy CSP models, their operators (*and* and *max*) are idempotent, so classical local consistency enforcing algorithms can be defined. For the lexicographic or weighted models, their operators are not idempotent so classical local consistency involving propagation cannot be used. In this case, local consistency methods which do not involve propagation can be used (fully or directed arc inconsistency counts, for instance [20, 60]).

2.2.2 *Semiring-Based Constraint Satisfaction Problems*

A c-semiring [6] is a tuple $(A, +, \times, \mathbf{0}, \mathbf{1})$ with the following properties,

— A is a set, $\mathbf{0}, \mathbf{1} \in A$;

— $+$, called the additive operation, is closed in A, commutative, associative, $\mathbf{0}$ is the identity, idempotent and $\mathbf{1}$ is its absorbing element;

— $\times$, called the multiplicative operation, is a closed in A, associative, $\mathbf{1}$ is the identity, $\mathbf{0}$ is the absorbing element, and commutative;

— $\times$ distributes over $+$.

In this framework, a value of A is assigned to each tuple satisfying (partially) a constraint. This value represents the tuple weight, cost or level of confidence. There is a partial order among the elements of A: $a \leq_S b$ iff $a+b=b$. Intuitively, this means that b is better than a. Element $\mathbf{1}$ is the maximum in the partial ordering, representing complete satisfaction, and $\mathbf{0}$ is the minimum, representing complete violation ($\mathbf{0} \leq_S a \leq_S \mathbf{1}, \forall a \in A$). The addition operation is used to select the best solution among different assignments, while the product is used to combine several constraints. It should be noted that $a \times b \leq_S a$, that is, combining more constraints leads to worse results. Problems are represented as constraint systems. This approach is denoted as SCSP. This framework includes the following specific models for OCSP.

Classical CSP Model. Two levels of satisfaction, 1 or true (allowed tuples) and 0 or false (not allowed tuples). The additive operation is logical *or*, while the multiplicative operation is logical *and*. The c-semiring is $(\{0, 1\}, or, and, 0, 1)$.

Fuzzy Model. The set A is the real interval $[0, 1]$, with the usual meaning (0 complete violation, 1 complete satisfaction). The order is the natural order on reals. The additive operation is *max*, while the multiplicative operation is *min*. The c-semiring is $(\{0, 1\}, max, min, 0, 1)$.

Lexicographical Model. As in the VCSP case, the lexicographical model can be included in the semiring approach. The resulting c-semiring is $(\{(l, k)|l,\ k \in [0, 1]\}, max, min,\ (0, 0), (1, 1))$, where the first element of a value is used to reason as in the fuzzy model, and the second is used to discriminate among equivalent fuzzy solutions. Operations *max* and *min* are defined appropriately (see [6] for further details).

Weighted Model. Each constraint tuple has an associated cost. The goal is to find the assignment that minimizes the total cost. A suitable c-semiring is $(R^-, max, +, -\infty, 0)$, where costs are represented as negative real numbers, and the ordering is the usual order on reals.

Probabilistic Model. In the probabilistic model, a probability of presence is attached to each constraint. This probability is attached to each tuple in the following form. If a tuple t satisfies constraint $C, pr(t)$ is 1; otherwise, $pr(t)$ is $1-pr(C)$ (a similar transformation was made in the fuzzy model with prioritized constraints). With this interpretation, the c-semiring is $([0, 1], max, \times, 0, 1)$, with the usual order on reals.

Informally, significant results of the SCSP framework are the following,

— *Local and global consistency*: the best level of consistency of a subproblem is higher than the best level of consistency of the whole problem.

— *Equivalence*: after applying local consistency to a problem, it is equivalent to the original problem if $\times$ is idempotent.

— *Termination*: a local consistency procedure terminates in a finite number of steps, if the values of A used in the problem constraints form a finite set I, and operations $+$ and $\times$ are closed in I.

— *Order independence*: two different applications of local consistency procedures to the same problem produce equal results if $\times$ is idempotent.

These results on local consistency are in agreement with VCSP result: only if the operator combining constraint valuations is idempotent, classical local consistency methods make sense for OCSP. Only the classical CSP and the fuzzy models present idempotent operators, and therefore, classical local consistency methods can be applied on them. For the other models, local consistency can be enforced by methods not involving constraint propagation.

Comparing SCSP and VCSP [6], if the valuation set is totally ordered both approaches are equivalent (with the same expressive power), and differences on modelling can be solved by a syntactical transformation. If the valuation set is partially ordered, the SCSP approach is the only applicable, being more general than VCSP. Considering a partial order on valuations is not a purely academic feature: this capacity seems to be very adequate for multi-objective optimization.

2.3. Multi-Objective Optimization

In most of the specific OCSP models and the VCSP generic model, it is assumed that there exists a total order in the valuation set. Given two assignments, they are always comparable. However, this is not always the case for real-life problems: when constraints are of very different nature, violations of different constraints cannot be easily aggregated. In this case, an OCSP problem can be formulated as a multi-objective optimization problem, as opposite to single-objective optimization where a single scalar function has to be optimized.

A multi-objective optimization problem is defined by a pair $(S, \boldsymbol{F})$, where S is a finite set of feasible states and $\boldsymbol{F}$ denotes the vector $[f_1, \ldots, f_k]$ of k cost functions to be minimized simultaneously. Each scalar function f_i maps the state of feasible states S in the set of reals R. Given a state s, $\boldsymbol{F}(s)$ denotes the vector $[f_1(s), \ldots, f_k(s)] = [z_1, \ldots, z_k] = z$. We will call this vector the objective point. The set of objective points is called the *objective space* $\boldsymbol{Z}$. An objective point is said to be *attainable* if there exists a state s in S so that $\boldsymbol{F}(s) = z$. An *ideal point* in the objective state is a point where each objective function is minimized independently of the others,

$$z* = [min(f_1), \ldots, min(f_k)] \text{ over all } s \in S$$

Ideal points are however usually not attainable. Without a total ordering in $\boldsymbol{Z}$, given two feasible alternatives s and s', there may be no answer as to whether $\boldsymbol{F}(s)$ is greater to, less than or equal to $\boldsymbol{F}(s')$. This is sometimes emphasized by quoting the word 'Minimize'.

Given two feasible states $s, s' \in S$, we say that s *dominates* s' if and only if for all f_i of $\boldsymbol{F}$, $f_i(s) \leq f_i(s')$ and there exists at least one f_j such that $f_j(s) < f_j(s')$. A feasible state s is said to be *efficient*, also called *Pareto optimal*, non-inferior, non-dominated or Pareto-admissible, if and only if there does not exist a state $s' \in S$ which dominates s. This definition requires that the objective functions are monotone [45]. Efficiency is the multi-objective extension of optimality. Given a problem P, $E(P)$ denotes the set of

efficient states; they are the "best" states for the multi-objective optimization problem. In practice, the set $E(P)$ is approximated, and the user selects one of these states as problem solution.

Multi-objective optimization problems are typically NP-hard. Determining whether an attainable point is dominated or non-dominated is an NP-complete task even for problems where the corresponding single-objective optimization is not. For instance, both the shortest path problem and the minimal spanning tree problem are NP-complete in their corresponding multi-objective formulations, even for only two objectives. However, increasing complexity through adding objective dimensions is especially troubling when even the one-dimensional case is NP-hard, as it is often true for combinatorial optimization problems.

3. Solving Methods for OCSPs

3.1. Systematic Search Methods

3.1.1 Branch and Bound

All specific OCSP models can be solved by methods based on depth-first branch and bound, an optimization procedure for finite combinatorial optimization. In the following, we provide a detailed explanation of these methods specialized for the binary Max-CSP problem, an instance of the weighted model where all constraints are binary and all constraints have the same weight. A solution of the Max-CSP problem is an assignment satisfying as many constraints as possible. Many of the strategies developed below can be adapted to other models.

Depth-first branch and bound (BnB) performs a depth-first traversal on the search tree defined by the problem, where internal nodes represent incomplete assignments and leaf nodes stand for complete ones. Assigned variables are called *past* (P), while unassigned variables are called *future* (F). The *distance* of a node is the number of constraints violated by its assignment. At each node, BnB computes the *upper bound* (UB) as the distance of the best solution found so far, and the *lower bound* (LB) as an underestimation of the distance of any leaf node descendant from the current one. When $UB \leq LB$, we know that the current best solution cannot be improved below the current node. In that case, the algorithm prunes all its successors and performs backtracking.

The efficiency of BnB-based algorithms largely depends on the quality of the lower bound, which should be both as large and as cheap to compute as possible. At the current node, the simplest lower bound is the number of inconsistencies among past variables,

$$LB(P) = distance(P)$$

The BnB algorithm with this lower bound appears in Figure 1. Procedure BB receives the following arguments: P is the set of past variables, F is the set of future variables, and FD is the collection of future domains. First, it checks if a leaf node has been reached

```
procedure BB(P, F, FD)
1 if (F = Ø) then
2    BestS ← assignment(P);
3    UB ← distance(Best S);
4 else
5    X_i ← select-variable(F);
6    while FD_i ≠ Ø do
7       a ← select-value(FD_i);
8       if (LB(P, X_i, a) < UB) then BB(P ∪ {X_i, a}, F − {X_i}, FD − {FD_i})
9       FD_i ← FD_i − {a};
endprocedure

function LB(P, F, X_i, a, FD)
10 return distance(P ∪ {X_i, a});
```

Figure 1. Depth-first branch and bound algorithm.

(line 1). If so, it updates the best solution *BestS* (line 2) and the upper bound UB (line 3). Otherwise, it selects X_i as the current variable (line 5) and performs a loop checking every value of X_i (lines 6 to 9). If the lower bound computed with value a of X_i (line 10) reaches the upper bound, this branch is pruned. Otherwise, search continues along this branch (recursive call of line 8). It is assumed the existence of a function `distance`, which takes a set of assigned variables and returns the number of constraints among variables in the set unsatisfied by their current assignment.

3.1.2 Partial Forward Checking

The *partial forward checking* (PFC) algorithm combines the branch and bound schema enhanced with lookahead [20] (in that paper this algorithm was called P-EFC3). It is a direct descendent of the popular forward checking algorithm [23]. PFC keeps for all feasible values of future variables the number of inconsistencies with previous assignments. The *inconsistency count* associated with value a of variable X_i, ic_{ia}, is the number of inconsistencies that value a of X_i has with the assignments of past variables. The sum $\sum_{j \in F} min_b(ic_{jb})$ is a lower bound of the number of inconsistencies that will necessarily occur between variables of P and F if the current partial assignment is extended into a total one. This term can be added to the distance of past variables to compute the lower bound of the current partial assignment, because both terms record different inconsistencies. The new lower bound is,

$$LB(P, F) = distance(P) + \sum_{j \in F} min_b(ic_{jb})$$

In addition, a value b of a future variable X_j can be pruned if the lower bound, where the minimum contribution of X_j is substituted by ic_{jb}, is not lower than the upper bound. The PFC algorithm appears in Figure 2.

```
procedure PCF(P, F, FD)
1 if (F = Ø) then
2   BestS ← assignment(P);
3   UB ← distance(BestS);
4 else
5   X_i ← select-variable(F);
6   while FD_i ≠ Ø do
7     a ← select-value(FD_i);
8     if (LB(P, X_i, a, FD) < UB) then
9       NewFD ← look − ahead(P, F, X_i, a, FD, UB);
10      if (not empty-domain and LB(P, F, X_i, a, NewFD) < UB) then
11        PFC(P ∪ {X_i, a}, F − {X_i}, NewFD);
12    FD_i ← FD_i − {a};
endprocedure

function look-ahead(P, F, X_i, a, FD, UB)
13 forall j ∈ F − {X_i} do
14   forall b ∈ FD_j do
15     if (LB_jb(P, F, X_i, a, FD) ≥ UB) then FD_j ← FD_j − {b}
16     elseif (inconsistent(X_i, a, X_j, b)) then
17       ic_jb ← ic_jb + 1;
18       if (LB_jb(P, F, X_i, a, FD) ≥ UB) then FD_j ← FD_j − {b};
19 return FD;

function LB(P, F, X_i, a, FD)
20 Newd ← distance(P) + ic_ia;
21 return Newd + Σ_{j∈F, j≠i} min_b ic_jb;

function LB_jb(P, F, X_i, a, FD)
22 Newd ← distance(P) + ic_ia;
23 return Newd + ic_jb + Σ_{k∈F, k≠i, j} min_c ic_kc;
```

Figure 2. Partial forward checking algorithm.

The main procedure presents three new lines with respect to the BB algorithm of Figure 1, lines 9, 10 and 11. The look-ahead function updates future domains (line 9). If no empty domains have been found and the new lower bound (possibly updated by the look-ahead function) does not reach the upper bound (line 10), search continues along this branch (recursive call of line 11). The look-ahead function considers all feasible future values (double loop in lines 13 and 14), checking if they can be pruned before (line 15) or after (line 18) updating their inconsistency counts (line 17). It returns the new set of future domains. Functions LB and LB_{jb} compute, respectively, the lower bound at the current node and the lower bound for a future variable X_j taking value b.

Directed Arc-Inconsistency Counts. While distance of P records inconsistencies among past variables, and inconsistency counts record inconsistencies between past and future variables, *directed arc-inconsistency courts* record inconsistencies among future variables. Given that each term records violations of different constraints, they can be added (with some care) to form a new lower bound. Different versions of DAC have produced three different lower bounds for the PFC algorithm. In the following we present these versions in chronological order.

Static DAC. Given a static variable ordering, the DAC associated to value a of variable X_i, dac_{ia}, is the number of variables which are arc-inconsistent with value a for X_i and appear after X_i in the ordering [60]. A variable is arc inconsistent with value a for X_i if all its values are incompatible with a. The counter dac_{ia}, is a lower bound of the number of inconsistencies that X_i will have with variables after X_i in the ordering if a is assigned to X_i. It is worth noting that each arc inconsistency is recorded in one DAC only, because they are directed. The sum $\sum_{j\in F} min_b(dac_{jb})$ is a lower bound of the number of inconsistencies that will necessarily occur among variables of F if the current partial assignment is extended into a total one. This term can be added to the distance of past variables plus the sum of ICs, to compute the lower bound of the current partial assignment, because they record different inconsistencies. The new lower bound is [60],

$$LB(P, F) = distance(P) + \sum_{j\in F} min_b(ic_{jb}) + \sum_{j\in F} min_b(dac_{jb})$$

which can be substituted advantageously by [33],

$$LB(P, F) = distance(P) + \sum_{j\in F} min_b(ic_{jb} + dac_{jb})$$

since $ic_{ia} + dac_{ia}$ is the minimum number of inconsistencies that will necessarily occur among pairs of non past variables if value a is assigned to X_i. With this lower bound, the PFC algorithm remains the same (now it is called PFC-DAC), changing the functions computing the lower bound, which are detailed in Figure 3. Notice that in function $\texttt{LB}_{jb}$, the dac_{ia}, is not added to the result because these inconsistencies have been recorded as ICs on future variables by the function `look-ahead`. Using static DAC, PFC must follow for variable instantiation the same ordering used in DAC computation. DACs are computed in a preprocessing step, before PFC starts.

function $\texttt{LB}(P, F, X_i, a, FD)$
20 $Newd \leftarrow \texttt{distance}(P) + ic_{ia}$;
21 **return** $Newd + dac_{ia} + \sum_{j\in F,\, j\neq i} min_b(ic_{jb} + dac_{jb})$;

function $\texttt{LB}_{jb}(P, F, X_i, a, FD)$
22 $Newd \leftarrow \texttt{distance}(P) + ic_{ia}$;
23 **return** $Newd + ic_{jb} + dac_{jb} + \sum_{k\in F,\, k\neq i,\, j} min_c(ic_{kc} + dac_{kc})$;

Figure 3. Lower bound functions for static DAC.

Inferred and Cascaded DAC. If values a, b with minimum DAC of two constrained variables X_i and X_j, X_j after X_i in the static ordering, are incompatible and dac_{ia} does not include an inconsistency from C_{ij}, dac_{ia} can be incremented in 1. In the best case values a, b will be selected for their variables, and they are incompatible. Otherwise, other values with higer DAC contribution will be selected. In both cases, the increment of dac_{ia} is justified. This approach is called *inferred* DAC [61]. If inferred DAC are used as before, we talk about *cascaded* DAC, which have to be computed with care, recording the contribution of each constraint to avoid adding twice the same inconsistency. When using cascaded DAC in lower bound computation, only contributions of variables not connected with the current one can be added (otherwise, duplications may occur). The author suggests to compute two lower bounds, one from static DAC and other from cascaded DAC, using the largest for pruning purposes.

Reversible DAC. A new approach uses reversible DAC, relaxing the condition of a static variable ordering. Its only requirement is that constraints among future variables must be directed: if C_{ij} is a constraint between future variables, it is given a direction, for instance from j to i. Arc-inconsistencies of C_{ij} are recorded in the DAC of X_i, In this way, the same inconsistency cannot be recorded in the DAC of two different variables. Directed constraints among future variables induce a directed constraint graph G^F, where $Nodes(G^F) = F$ and $Edges(G^F) = \{(j, i) | C_{ij}$ between future variables, direction of C_{ij} from j to $i\}$. Given a directed constraint graph G^F, the graph-based DAC of value a of variable X_i, $dac_{ia}(G^F)$, is the number of predecesors of X_i in G^F which are arc-inconsistent with value a for X_i. The minimum number of inconsistencies recorded in variable X_j, $MNI(X_j, G^F)$, is as follows,

$$MNI(X_j, G^F) = min_b(ic_{jb} + dac_{jb}(G^F))$$

and the graph-based lower bound (based on G^F) is,

$$LB(P, F, G^F) = distance(P) + \sum_{j \in F} MNI(X_j, G^F)$$

With graph-based DAC any dynamic variable ordering can be used. This approach is complemented with the dynamic selection of G^F. Given that any G^F is suitable for DAC computation, a local optimization process looks for a good G^F with respect to current IC values at each node. Since the only possible change in G^F is reversing the orientation of its edges (by reversing the direction of its constraints), this approach is called *reversible* DAC. These features are included in the PFC-MRDAC algorithm [35], where in addition, DAC are updated during search considering future value pruning.

The algorithm implementing these features appears in Figure 4 and it is relatively complex for a detailed description. The differences with PFC (Figure 2) start at line 10. If the current branch is not pruned, a better directed graph G^F is searched by a greedy optimization method (line 11), and the test for pruning the current branch is repeated (line 12). If the search continues, future values are tested for pruning by function *delete* (line 13), where G^F is specialized for each future value to maximize value removal. If no future domain becomes empty, DAC are maintained with respect to previous value removal in function *prop-del*, which again test future domains (line 15). If no empty domain is found, search continues along the current branch with the recursive call (line 17).

```
procedure PFC-MRDAC (P, F, FD, G^F)
1  if(F = ∅) then
2    BestS ← assignment(P);
3    UB ← distance(BestS);
4  else
5    X_i ← select-variable(F);
6    while FD_i ≠ ∅ do
7     a ← select-value(FD_i);
8     if(LB(P, F, X_i, a, FD, G^F) < UB) then
9       NewFD ← look-ahead(P, F, X_i, a, FD, UB, G^F);
10      if(not empty-domain)
11        NewFD ← greedy-opt (G^F, F, NewFD);
12        if(LB(P, F, X_i, a, NewFD, NewG^F) < UB) then
13          NewFD ← delete(F, NewFD, NewG^F);
14          if(not empty-domain)
15            NewFD ← prop-del(F, NewFD);
16            if(not empty-domain)
17              PFC-MRDAC(P ∪ {X_i, a}, F − {X_i}, NewFD, NewG^F)
18    FD_i ← FD_i − {a};
endprocedure
```

Figure 4. PFC with maintaining reversible DAC.

Partition-Based LB. So far, the lower bound has recorded contributions of individual future variables. This approach [33] considers the partition $\mathcal{P}(F) = \{F_q\}$ of the set F, and the lower bound is computed as the aggregation of contributions from each element of the partition. The minimum number of inconsistencies provided by a subset $F_q = \{X_i, \ldots, X_p\}$ is,

$$MNI(F_q, G^F) = min_{b_1, \ldots, b_p} \sum_{j \in F_q} [ic_{jb_j} + dac_{jb_j}(G^F)] + cost'((X_i, b_i), \ldots, (X_p, b_p))$$

where value b_j corresponds to variable X_j. The expression $cost'((X_i, b_i), \ldots, (X_p, b_p))$ accounts for the number of constraints among variables in F_q which are violated by the values $\{b_i, \ldots, b_p\}$ and this violation is not recorded as directed arc inconsistency. The new lower bound, called *partition-based* because it depends on the selected partition, is defined as,

$$LB(P, \mathcal{P}(F), G^F) = distance(P) + \sum_{F_q \in \mathcal{P}(F)} MNI(F_q, G^F)$$

A central aspect of this approach is finding easily good partitions. For efficiency reasons, authors restrict the search to partitions with subsets of up two elements, which are computed following a greedy approach. If merging two future variables into a partition element increases their contribution in 1 (the respective minima of IC + DAC are incompatible, but this inconsistency is not recorded in any DAC), they are merged. In this way,

the partition is computed, and the associated lower bound is obtained. It is easy to prove that the partition-based lower bound is always higher than or equal to the reversible DAC lower bound.

Arc-Inconsistency Counts. The *arc-inconsistency count* (AC) associated to value a of variable X_i, ac_{ia}, is the number of future variables which are arc-inconsistent with value a for X_i [20]. A variable is arc inconsistent with value a for X_i if all its values are incompatible with a. Using AC instead of DAC for lower bound computation may cause to count twice an inconsistency (suppose that C_{ij} forbids all value pairs between X_i and X_j). A way to overcome this fact is to weight AC contributions with 1/2, in such a way that the contribution of an inconsistency cannot be counted twice. The lower bound proposed is [2],

$$LB(P, F) = distance(P) + \sum_{j \in F} min_b(ic_{jb} + 1/2ac_{jb})$$

This approach can be generalized to consider any weights such that $w_{ij} + w_{ji} = 1$, where w_{ij} is the weight that multiplies AC contributions of C_{ij} in variable X_i. Weights can also change dynamically during search. Determining how weights evolve is an open question of this method.

Russian Doll Search. The idea of *Russian Doll Search* (RDS) [58], is to replace one search by n successive searches on nested subproblems. Given a static variable ordering, the first subproblem involves the last variable only, the ith subproblem involves all the variables from the $n - i + 1$ to the last, and the nth subproblem involves all variables. Each subproblem is optimally solved using a PFC algorithm, following an increasing variable order: the first variable has the lowest number in the static ordering, and the last is always the n variable. The central point of this technique is that, when solving the whole problem, subproblem solutions obtained before can be used in the lower bound in the following form,

$$LB(P, F) = distance(P) + \sum_{j \in F} min_b(ic_{jb}) + distance(Bestsolution(F))$$

Let us suppose that we are solving the whole problem and P involves the first $n - i$ variables. The set F involves from $n - i + 1$ to n variables; that is, F is composed by the variables of the ith subproblem. Then, the distance of the best solution found in that problem can be safely added to the contribution of P plus ICs to form the lower bound. This strategy is also used when solving the subproblem sequence. Solving subproblem i involves reusing all solutions from previously solved subproblems.

3.1.3 Systematic Search Heuristics

As in CSP solving, heuristics for variable and value ordering are used. Regarding static variable ordering, required by PFC-DAC or RDS algorithms, the following heuristics have been reported.

Decreasing Backward Degree (BD) [62]. It considers first variables most constrained with past variables. It is expected that these variables will have high IC in their values, so after variable assignment the current distance is likely to increase. Its main disadvantage is the lack of information at the first levels of the tree.

Decreasing Forward Degree (FD) [33]. It considers first variables most constrained with future variables. This heuristic tries to increase the propagation of IC towards future variables, increasing the IC contribution to the lower bound.

Decreasing Degree (DG) [62]. It can be seen as a combination of the two previous, BD and FD. At first levels of the tree FD dominates, while at deep levels BD dominates.

Decreasing AC Mean (AC) [62]. It consider first variables with high AC. Variables with high AC will probably have also high DAC, so it tries to increase the DAC contribution to the lower bound.

These heuristics and their combinations have been tested for PFC-DAC algorithms [33]. Their conclusion on random problems is that the combination FD/BD (FD as first criterion, breaking ties with BD) is the most effective combination and, in addition, it is quite chip to compute. For the RDS algorithm, variable orderings with limited bandwith seem to be more adequate.

Regarding dynamic variable ordering, required by PFC, PFC-RDAC or PFC-PRDAC algorithms, the classical heuristic of *minimum domain* (DOM) is preferred, selecting first the variable with the minimum number of values in its domain. Regarding PFC [61], provided a set of experiments on problems with low connectivity, concluding that combinations BD/AC and BD/DOM/AC were good choices. Considering PFC-RDAC and PRDAC, both use DOM divided by FD, a popular combination in classical CSPs.

Regarding heuristics for dynamic value selection, in BnB-based algorithms most promising values (violating less constraints) are selected first. The goal it to decrease the upper bound, which will improve search efficiency. Therefore, in all described algorithms values are ordered by either (i) increasing IC, (ii) increasing IC + DAC, or (iii) increasing IC + AC. No comparative studies among the three criteria have been reported.

3.2. Local Search Methods

Greedy local search algorithms have been successfully applied to different classes of OCSP, mainly due to their efficiency. Local search algorithms start from an initial configuration and move from the current configuration to neighborhood configurations until a good solution is reached. These algorithms can produce suboptimal solutions, since there is no guarantee for finding the optimal one.

3.2.1 Min-Conflicts

Min-Conflicts [39, 40] have been widely used to solve OCSP. Several versions of the Min-Conflicts procedures have been developed in recent years. They all differ from the

basic ones in that they incorporate clever search techniques [61, 63]. The Min-Conflicts algorithm chooses randomly any conflicting variable, i.e., the variable that is involved in any unsatisfied constraint, and then picks a value which minimizes the number of violated constraints.

Because the pure Min-Conflicts algorithm is unable to move beyond a local-minimum, the Min-Conflicts algorithm has been combined with the random-walk strategy. For a given conflicting variable, the random-walk strategy picks randomly a value with a probability p, and then applies the Min-Conflicts algorithm with a probability $1-p$. The value of the parameter p has a big influence on the performance of the algorithm. Another version of the Min-Conflicts algorithm called the Steepest-Descent-Random-Walk has been proposed. This version explores the whole neighborhood of the current configuration and selects the best neighbour according to the evaluation value.

One factor that limits the efficiency of local search algorithms is the size of the neighborhood. If there are many neighbors to consider, then the search will be very costly. To cope with this problem, Fast Local Search (FLS) has been introduced [53]. The main goal is, guided by heuristics, to ignore neighbours that are unlikely to lead to fruitful hill-climbs in order to improve the efficiency of a search. Each neighborhood move is associated with an activation bit. Only those neighbors whose bits are switched on will be considered in a hill-climbing step. All activation bits are switched on at the beginning. If a move has been examined in a hill-climbing step without leading to a better candidate solution, then its activation bit will be switched Off.

A connectionist approach called GENET [15] has been proposed for solving CSP. In this approach, the problem is represented by a network with inhibitory links. GENET solves CSP by hill-climbing using a variation of the Min-Conflicts heuristic. GENET uses a constraint weighting scheme to escape from local minima. When GENET encounters a local minimum, the weights of all the constraints which are violated in the minimum are increased. This increases the cost of violating constraints which are violated in the minimum, thus increasing the value of the cost function in the minimum and enabling the network to escape to other states. The idea of incrementing the weight of violated constraints at a local minima is also present in the breakout method [41].

3.2.2 Genetic Algorithms

Genetic Algorithms (GA) [22] have been shown quite successful in a wide range of applications. GA borrow their ideas from evolution [24]. The idea is to maintain a population of candidate solutions. The candidate solutions are given individual chances to produce offspring depending on their fitness. Fitness is measured by the objective function in optimization. GA have been applied to constraint satisfaction [12, 17, 46]. The use of GA in constrained optimization problems raises several issues to which a considerable amount of research has been devoted in the last years. One of the most important issues is how to incorporate constraints into the fitness function in order to guide the search properly. To incorporate constraints, one approach focussed on the use of penalty functions [43]. Several researchers have attempted to derive good techniques to build

penalty functions. The work reported in [25] proposed a technique in which the user defines several levels of violations, and a penalty coefficient is chosen for each in such a way that the penalty coefficient increases as one reaches higher level of violations.

3.2.3 Simulated Annealing

Simulated annealing (SA) [28] is another stochastic search technique that has proved to be very effective for large scale optimization problems. In the SA technique, the temperature that is a variable is started at a high value and gradually reduces during the search. At high temperatures, moves are accepted in a random fashion regardless of whether they are uphill or down. As the temperature is lowered, the probability of accepting downhill moves rises and the probability of accepting uphill moves drops. A variant of SA [38] has been applied to CSPs that proved to be better than standard SA. This approach relies on dividing the search space into disjoints subspaces which are then processed by a localized SA strategy. Different temperatures and annealing speeds can be maintained in the different subspaces depending upon certain evaluation criteria. Another approach [59] combines SA with the theory of Lagrange multipliers, providing good results in discrete and continuous problems.

3.2.4 Local Search Preprocess for Systematic Search

To increase branch and bound efficiency, the lower bound should increase as fast as possible during search. In the same way, the upper bound should decrease as fast as possible, to enable the pruning condition, $LB \leq UB$. In systematic search, most efforts have been devoted to get large lower bounds, taking as starting upper bound the leftmost leave in the search tree. However, some local search can be done to get a better starting upper bound. In this sense [61], reported some experiments using three local search techniques to find an upper bound: (i) min-conflicts [40] with a random walk component, (ii) the breakout procedure [41], and (iii) weak commitment search [65]. They were used as any time procedures, getting in the first seconds of execution, a quite significant improvement in the upper bound. In the same line, the decrement of the initial upper bound by mean field annealing [10] causes visible improvements in branch and bound efficiency.

3.3. Approximation Methods

Systematic search methods compute the optimal solution, but they are often too expensive when solving real problems. Local search methods compute suboptimal solutions, but they cannot determine how far they are from the optimal one. Approximation methods provide an alternative approach. Instead of looking for one solution, they look for an interval [lower bound, upper bound] enclosing the optimal solution. If the lower bound equals the upper bound, the optimum has been found. Otherwise, the distance between

the upper and lower bounds can be used to assess the quality of the current solution. With this information, the user may decide when to stop the search, taking a suboptimal solution of reasonable quality.

Approximation methods combine previous solving approaches. Solution candidates producing the upper bound can be easily computed by local search methods. Systematic search strategies can be adapted for anytime lower bound computation. Three different methods for anytime lower bounds are presented in [11].

Problem Simplification. A lower bound is computed by solving completely a simplified problem. The optimum of the simplified problem either is a lower bound or allows such a bound to be computed. In [21], a simplification is produced by modifying the violation function.

Objective Simplification. A lower bound is computed by aiming at a simpler objective, like local consistency. For example, the sum of DAC at the root of the tree search [35], is a problem lower bound. More generally, the set of constraints which have to be removed to achieve any kind of classical local consistency property in the problem, can be used to compute a problem lower bound.

Search Simplification. The idea is performing a limited tree search and exploiting subproblem lower bounds in order to produce a problem lower bound. A number of algorithms have been explored in [11]. The combination iterative deepening + russian doll search produces the best results in both random and real benchmarks. It follows the Russian Doll approach, substituting one search by n nested searches of subproblems. However, each subproblem is not solved to optimality but a subproblem lower bound is computed using an iterative deepening algorithm. Interestingly, at the jth subproblem, the contribution of lookahead can be safely added with the lower bound of the $j+1$th subproblem, to compute a lower bound for the jth subproblem. The last subproblem is the whole problem, and then the problem lower bound is computed.

Following a different approach, an approximation of the optimal solution of a Weighted CSP is given in [36]. The method is inspired in randomized algorithms and semidefinite program relaxation. The approximation has polynomial time complexity and its cost (in terms of the added weight of unsatisfied constraints) is assured to be between the optimal and a worst case bound computable also in polynomial time. Experimental results show that the algorithm performs well on random instances.

3.4. Multi-Objective Optimization

In the following, we outline some methods to solve OCSPs when formulated as multi-objective optimization problems. For more details, the reader is addressed to specific bibliography [45]. The goal in multi-objective optimization is to construct the Pareto set $E(P)$. Assuming that all the states in $E(P)$ are of equal value, any can be taken as the problem solution. A good approach would be to generate a representative sampling of $E(P)$, on which a decision support system could make a final choice. If this is too costly, the sampling is approximated.

3.4.1 Parametric Scalarization

An approach to multi-objective optimization, $\boldsymbol{F} = (f_1, \ldots, f_k)$, is to specify a weight vector, $\boldsymbol{w} = (w_1, \ldots, w_k)$, such that $w_i \geq 0$ and $\sum w_i = 1$. Each w_i gives a weight to each objective function f_i, representing our preferences among objectives. Given a suitable weight vector and an scalarizing function g: $R^k \mapsto R$, the multi-objective optimization problem '*Minimize*' $\boldsymbol{F}(s)$ *over all* $s \in S$ is replaced by the following single-objective optimization problem,

$$\textit{Minimize}\, g(\boldsymbol{F}(s), \boldsymbol{w})\ \textit{over all}\ s \in S$$

Numerous researchers have discovered a connection between Pareto optimality and weights. Given suitable conditions on the objective functions and the state space we have that Pareto optimal points correspond to solutions of the optimization problem over the scalarizing function. Points on the Pareto frontier can be generated by solving the problem P_w for different weight vectors $\boldsymbol{w}$, i.e. minimizing the aggregation function for different weight settings. Common functions are the following.

The Weighted Sum. The weighted sum is probably the most common scalarizing function. Sometimes it makes use of a fixed reference point $\boldsymbol{z}'$, for instance an ideal point $\boldsymbol{z}^*$,

$$g(\boldsymbol{z}, \boldsymbol{w}) = \sum w_i z_i \quad \text{or} \quad g(\boldsymbol{z}, \boldsymbol{w}) = \sum w_i (z_i - z_i')$$

The Chebychev function. $g(\boldsymbol{z}, \boldsymbol{w}) = max_i\{w_i(z_i - z_i')\}$

The l_p-norm function. Given $p \geq 1$, $g(\boldsymbol{z}, \boldsymbol{w}) = (\sum w_i (z_i - z_i')^p)^{1/p}$

Other Scalarizing Functions. $g(\boldsymbol{z}, \boldsymbol{w}) = (1 - \alpha) g_c(\boldsymbol{z}, \boldsymbol{w}) + \alpha / k g_w(\boldsymbol{z}, \boldsymbol{w})$ where g_c is the Chebychev function, and g_w is the weighted sum.

We can now formulate a series single-objective optimization problems by varying these parameters, and solve them using traditional single-objective optimization methods. This will give us points on the Pareto set.

3.4.2 Local Search

As in the single-optimization case, local search is usually divided among a neighbourhood function that generates successors from the current state, a selection function that determines which state will be selected as the next one, and meta-strategies to escape from local minima. These issues, hard enough in the single-objective case, become more difficult in multi-objective optimization.

Selection among neighbours becomes more complicated because they may be non-comparable. Objective functions are usually scalarized, following the approaches mentioned in Section 3.4.1. When searching in the multi-objective case, we may find many states which are neither worse nor better than the "best" found so far; there is no dominance among them. A cache or archive is a way of collecting and handling non-dominant

best states. This cache will contain, at any time, a subset of the best approximation found to the Pareto set. The criterion to include a new state in the cache is as follows. If the new state dominates any state in the cache, the dominated states are removed from the cache and the new state is added. If the new state is dominated by any other state of the cache, it is not added. If there is no dominance relation between the new state and any state of the cache, the new state is added to the cache. It belongs to the approximation of the Pareto set.

Regarding SA procedures, the probability function is determined by using a probability scalarizing approach, where acceptance probabilities for each objective are aggregated. If p_k is the acceptance probability for the kth objective function, the aggregated acceptance probability p is a function $g(p_1, \ldots, p_k)$. A complete multi-objective Simulated Annealing algorithm (popularly called MOSA) was developed in [54]. The MOSA algorithm has been improved and tested in [55]. The paper illustrates the different options in the implementation of MOSA, and some typical behaviours of MOSA are explained by the possible complexity jump from solving uni-objective to multi-objective problems. The Mosa algorithm has been advocated for interactive use in an industrial setting [56].

Regarding GA procedures, those states non-dominated by other states in the population are selected to move the population toward the Pareto set. These states are then assigned the highest rank and eliminated from further contention. Another set of Pareto non-dominated states are determined from the remaining population and are assigned the next high rank. This process continues until the population is suitably ranked. Finally, the rank of an individual determines its fitness value. Remarkable here is the fact that fitness is related to the whole population, while with aggregation techniques an individual's raw fitness value is calculated independently of other individuals. Some popular algorithms are: Pareto ranking-based Genetic Algorithm (MOGA) [4], Non-dominated Sorting Genetic Algorithm (NSGA) [51], and Niched Pareto Genetic Algorithm (NPGA) [26].

4. Case Studies

4.1. Earth Observation Satellite Management

The problem consists of planning a daily management of a satellite to take a set of images from at least one of three instruments. The problem includes unary soft constraints, plus binary, ternary an one n-ary hard constraints [5, 37]. Problem instances can be downloaded from ftp://ftp.cert.fr/pub/lemaitre/LVCSP/Pbs/SPOT5.tgz.

Problem Formulation

Variables. A variable is associated with each image. A positive integer weights each variable.

Domain. It is formed by the possible assignment of the different instruments to take the image: three possible values for a mono image and only one value for a stereo image.

Constraints. A set of hard binary constraints expresses the non overlapping and minimum transition time constraints. A set of hard binary or ternary constraints expresses the limitation of the instantaneous data flow through the satellite telemetry. An n-ary hard constraint involving all the variables expresses the limitation of the on-board recording capacity.

Optimization Criteria. The weight of a partial assignment is defined as the sum of the weights of the assigned variables. A partial assignment is said feasible if and only if it satisfies all the hard constraints (with the n-ary constraint restricted to the assigned variables). The problem consists in finding a partial feasible assignment whose weight is maximum.

Solving Methods and Results

Systematic methods are able to optimally solve problem instances without any n-ary recording capacity constraint. However, they fail to do so when this type of constraint is included. All the proven optimal results have been obtained either using an ILP problem formalization and the CPLEX commercial software, or by using a Valued CSP formalization and Russian Doll Search. The original article of RDS [58] implemented in LISP reports very good results compared to Depth First Branch and Bound with backward and forward checking on several instances, and notices the negative influence of increasing graph bandwith to the cpu solving time. The ILOG Solver has been experimented on smaller instances, and a comparison report exists [37]. All the approximating results have been obtained with Tabu Search algorithms. In an operational context, an hour is currently considered as a maximum time to decide which images will be taken the next day and how to take them. Although this constraint cannot be considered as hard in the context of this benchmark, a program taking more than one day would not be very useful.

4.2. Timetabling

This case study presents a possible over-constrained problem plus a multi-objective optimization problem [48]. It includes constraints with high arity. The aim is to assign teachers to classes in a period of time in order to accomplish the requirement matrix and the teacher, and class availability matrices.

Problem Formulation

— $c_1, \ldots, c_m$ be m classes,
— $t_1, \ldots, t_n$ be n teachers,
— $1, \ldots, p$ be p periods,

— $R_{m\times n}$ requirements matrix: r_{ij} is the number of lectures given by teacher t_j to class c_i,

— $T_{m\times p}$: t_{ik} is 1 if teacher t_i is available at period k, otherwise it is 0,

— $C_{n\times p}$: c_{ik} is 1 if class c_i is available at period k, otherwise it is 0.

Variables. Find $x_{ijk} = 1$ if class c_i and teacher t_j meet at period k, $x_{ijk} = 0$ otherwise $(i = 1..m; j = 1..n; k = 1..p)$.

Constraints

1. x_{ijk} is either 0 or 1 $(i = 1..m; j = 1..n; k = 1..p)$.
2. $\sum_{k=1}^{p} x_{ijk} (i = 1..m; j = 1..n)$ is the right number of lectures to each class.
3. $\sum_{j=1}^{n} x_{ijk} \leq t_{ik} (i = 1..m; k = 1..p)$ each teacher is involved in at most 1 lecture for each period.
4. $\sum_{i=1}^{m} x_{ijk} \leq c_{jk} (j = 1..n; k = 1..p)$ each class is involved in at most 1 lecture for each period.

Preassignments can be added $x_{ijk} \geq p_{ijk}$ where $p_{ijk} = 0$ if there is no preassignment and $p_{ijk} = 1$ when a lecture of teacher t_j to class c_i is preassigned to period k.

Optimization Problem. To convert this into a optimization problem the following objective function has been proposed,

$$min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{p} d_{ijk} x_{ijk}$$

where a large d_{ijk} is assigned to periods k in which a lecture of teacher t_j to a class c_i is less desirable.

This problem has also a multicriteria version, considering the following aspects,

— The didactic cost: spreading the lectures over the whole week.

— The organizational cost: having a teacher available for temporary teaching.

— The personal cost: a specific day-off for each teacher.

Solving Methods and Results

The problem is NP-complete. Without considering availability matrices, the problem is polynomial time solvable. Reductions to graph coloring [42] have proven results on the existence of solution. Local search techniques have been applied for large amount of data like: tabu search [3, 14, 47], simulated annealing [1], genetic algorithms [13]. Logic programming approaches [27] have shown advantatges on the expressivness of constraints. A constraint relaxation problem solver COASTOOL [66] has also been used, casting the problem as a Partial CSP.

4.3. The Radio Link Frequency Assignment Problems

The problem consists in assigning frequencies to a set of radio links defined between pairs of sites in order to avoid interferences. Each radio link is represented by a variable whose domain is the set of all frequences that are available for this link. All the constraints are binary, non linear, and have finite domains. All problem instances have been built from a unique real instance with 916 links and 5744 constraints, and include both feasible (a solution exists satisfing all the constraints) and unfeasible (a solution exists satisfing all hard constraints and minimizing the cost of violating some of the soft constraints) [9]. Instances can be downloaded from: ftp://ftp.cert.fr/pub/lemaitre/FullRLFAP.tgz, and from http://www-bia.inra.fr/T/schiex/Doc/CELARE.html.

Problem Formulation

Variables. The links $i \in X$.

Domains. The finite set of frequencies available for each link D_i.

Constraints.

1. For each link $i \in X$ a frequency f_i has to be chosen from a finite set $D_i : f_i \in D_i$.
2. Two links can define a duplex link $|f_i - f_j| = d_{ij}$; d stands for distance.
3. Some links may already have a pre-assigned frequency $f_i = p_i$. There is a mobility cost for violating this soft constraint specified by m_i.
4. Two links may interfere together $|f_i - f_j| > d_{ij}$. There is a interference cost for violating this soft constraint specified by c_i.

Constraints (1) and (2) are hard, while (3) and (4) are soft. Several problems can be defined:

Feasibility. Find an assignment of frequencies to each link such that all constraints are satisfied. NP-complete since by constraints (1) and (4) it is possible to express the k-coloring problem.

Minimum Span. Minimize the largest frequency used in the assignment; it can be reduced to a short sequence of Feasibility problems using dichotomic search, and can simply be cast as Possibilistic/Fuzzy CSP by adding soft unary constraints on the domain values $min\, max_i\, f_i$.

Minimum Cardinality. Minimize the number of different frequencies used in the assignment; more difficult than the min-span $min\,|\cup_i f_i|$.

Maximum Feasibility. If all the constraints cannot be satisfied simultaneously, find the assignment that minimizes the sum of all the violation cost. It can be cast as Partial CSP $min(\sum c_{ij}\, violation(|f_i - f_j| > d_{ij}) + \sum m_{ij}\, violation(f_i = p_i))$.

Solving Methods and Results

Approximation techniques are applied to the large data sets to find good solutions without giving an optimality proof. The classical methods have been applied: local search including tabu search, simulated annealing, genetic algorithms, potential reduction. Some aproximations of local search techniques are very close to the optimum. For the calculation of lower bounds on the minimum number of used frequencies the following techniques are applied:

— *Branch and cut*: it can be applied in case that a linear programming (LP) formulation is used as a model. Then a branch and bound algorithm is applied that at every node of the search tree attempts to strengthen the LP bound of a LP relaxation version of the problem.

— *Constraint satisfaction*: several specialized branch-and-bound algorithms have been applied. And lately algorithms for overconstrained instances proving optimality have been provided.

— *Graph coloring techniques*: Chromatic number approximations on some modified version of the constraint graph gives good lower bound on the minimum number of used frequencies [31].

Several systematic methods have been applied to prove optimality. In [21] RDS was used to solve one of the hardest FAP instances in 32 days with a Spare 5 workstation. Five subinstances of this total instance were extracted, and the total accumulated cost of the subinstances was proven equal to best upperbound found at that moment for the global instance, so it was directly proven optimal. In [11] combinations of RDS with some iterative deeping techniques are shown to be very effective in the calculation of anytime lower bounds.

Four subinstances are solved using directed arc-inconsistency techniques in [34]. The CPU time given correspond to the time to prove optimality, that means the DAC algorithm is initialized with the optimal cost as upperbound. Recently in [29] the whole instance 6 is solved to optimality in no more than 3 hours using graph decomposition techniques combined with a dynamic programing algorithm.

5. Summary and Perspectives

In this paper we have given a short overview of the existing methods for modelling and solving OCSPs. The different models for OCSP formulation have been summarized, with special attention to the generic models on which several interesting properties on local consistency have been proved. Regarding solving methods, both systematic and local search families have been reviewed, devoting special attention to specific techniques to deal with over-constrained problems. Since some OCSP can be seen as multi-objective optimization problems, some notions on this topic have been presented. Finally, we have described three case studies to illustrate the usage of all previous techniques in real world problems.

This field is currently in full development. The last years have lead to important developments in OCSP solving algorithms, and new advances can be expected in the next years. Among the promising directions for further research, we mention the improvement of lower bounds used in BnB-based algorithms, and the exploitation of the constraint graph topology to speed up search. A good example of the first direction is very interesting work on arc consistency for soft constraints [50], where arc consistency is successfully redefined in the soft constraint context. This may cause better lower bounds to appear, as well as new redefinitions of other local consistency concepts. In the second direction, the work on variable elimination [32] has been shown very effective when applied to OCSP. Finally, we mention the reformulation of soft constraints as a single hard global constraint [44], to facilitate its inclusion into existing constraint programming frameworks. These positive results indicate that the solving capacity of existing technology will be substantially enhanced in the near future.

Acknowledgments

This work was carried out inside the ECSPLAIN project (IST-99-11969). Authors thank the other members of the project for their support, as well as project reviewers for their constructive criticisms. This work was supported by the IST Programme of the Commission of the European Union through the ECSPLAIN project (IST-1999-I1969).

References

1. Abramson, D. A. (1991). Constructing school timetables using simulated annealing: Sequential and parallel algorithms. *Management Science*, 37(1): 98–113.
2. Affane, M.-S., & Bennaceu, H. (1998). A weight are consistency technique for Max-CSP. In *Proc. of ECAI-98*, pages 209–213.
3. Alvarez-Valdes, R., Martin, G., & Tamarit, J. M. (1996). Constructing good solutions for the Spanish school timetabling problem. *Journal of the Operational Research Society*, 1203–1215.
4. Belegundu, A. D., Murthy, D. V., Salagame, & Constant, E. W. (1994). Multi-objective optimization of laminated ceramic composites using genetic algorithms. In *Fifth AIAA/USAF/NASA Symposium on Multidisciplinary Analysis and Optimization*, Paper 84-4363-CP, pages 1015–1022.
5. Bensana, E., Lemaitre, M., & Verfaillie, G. (1999). Earth observation satellite management. *Constraints*, 4(3): 293–299.
6. Bistarelli, S., Montanari, U., & Rossi, F. (1995). Constraint solving over semirings. In *Proc. IJCAI-95*, pages 624–630.
7. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaille, G., & Fargier, H. (1999). Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4: 199–240.
8. Borning, A., Freeman-Benson, B., & Wilson, M. (1992). Constraint hierarchies. *Lisp and Symbolic Computation*, 5: 223–270.
9. Cabon, B., De Givry, S., Lobjois, L., Schiex, T., & Warners, J. (1999). Radio link frequency assignment. *Constraints*, 4(1): 79–89.

10. Cabon, B., Verfaillie, G., Martinez, D., & Bourret, P. (1996). Using mean field methods for boosting backtrack search in constraint satisfaction problems. In *Proc. of ECAI-96*, pages 165–169.

11. Cabon, B., Givry, S., & Verfaillie, G. (1998). Anytime lower bounds for constraint violation minimization problems. In *Proc. of CP-98*, pages 117–131.

12. Chu, P., & Beasley, J. E. (1997). Genetic algorithms for the generalized assignment problem. *Computers and Operations Research*, 24: 17–23.

13. Colorni, A., Dorigo, & Maniezzo (1992). A genetic algorithm, to solve the timetable problem. Technical Report 90.060, Politecnico di Milano, Italy.

14. Costa, D. (1994). A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research*, 76: 98–110.

15. Davenport, A., Tsang, E., Wang, C., & Zhu, K. (1994). GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proc. of AAAI-94*, pages 325–330.

16. Dubois, D., Fargier, H., & Prade, H. (1996). Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6: 287–309.

17. Eiben, A. E., Raua, P.-E., & Ruttkay, Zs. (1994). Solving constraint satisfaction problem using genetic algorithms. In *Proc. 1st IEEE Conference on Evolutionary Computing*, pages 543–547.

18. Fargier, H., & Lang, J. (1993). Uncertainty in constraint satisfaction problems: A probabilistic approach. In *Proc. ECSQARU-93*, In Vol. 747 of LNCS, pages 97–104.

19. Fargier, H. (1994). *Problemes de satisfaction de constraintes flexibles: Application a l'ordonnancement de production*. Ph.D. Thesis, Univ. Paul Sabatier, Toulouse, France.

20. Freuder, E., & Wallace, R. (1992). Partial constraint satisfaction. *Artificial Intelligence*, 58: 21–71.

21. Givry, S., Verfaillie, G., & Schiex, T. (1997). Bounding the optimum of constraint optimization problems. In *Proc. of CP-97*, pages 405–419.

22. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addition-Wesley Publishing Co., Reading, MA.

23. Haralick, R. M., & Elliot, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14: 263–313.

24. Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*, 2nd ed. MIT Press.

25. Homaifar, A., Lai, S. H. Y., & Qi, X. (1994). Constrained optimization via genetic algorithms. *Simulation*, 62(4): 242–254.

26. Horn, J., & Nafpliotis, N. (1993). Multi-objective optimization using the niched pareto genetic algorithm. Technical Report Illi GAI Report 93005, University of Illinois at Urbana Champaign.

27. Kang, L., & White, G. M. (1992). A logic approach to the resolution of constraints in timetabling. *European Journal of Operational Research*, 61: 306–317.

28. Kirkpatrick, S., Gelat, C., & Vicci, M. (1983). Optimization by simulated annealing. *Science*, 220(4598): 671–680.

29. Koster, A. M., Hoesel, C. P., & Kolen, A. W. Optimal solutions for a frequency assignment problem via tree-decomposition. In Vol. 1665 of LNCS, pages 338–349.

30. Kumar, V. (1992). Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, Spring 1992, pages 32–44.

31. Lanfear, T. A. (1989). Graph theory and radio link frequency assignment problems. Technical Report, NATO, Allied Radio Frequency.

32. Larrosa, J. (2000). Boosting search with variable elimination. In *Proc. CP-00*, pages 291–305.

33. Larrosa, J., & Meseguer, P. (1996). Exploiting the use of DAC in Max-CSP. In *Proc. of CP-96*, pages 308–322.

34. Larrosa, J., & Meseguer, P. (1999). Partition-based lower bound for max-CSP. In *Proc. of CP-99*, pages 303–315.

35. Larrosa, J., Meseguer, P., & Schiex, T. (1999). Maintaining reversible DAC for max-CSP. *Artificial Intelligence*, 107(1): 149–163.

36. Lau, H. C. (1996). A new approach for weighted constraint satisfaction: Theoretical and computational results. In *Proc. of CP-96*, pages 323–337.

37. Lemaitre, M., & Verfaillie, G. (1997). Daily management of an earth observation satellite: comparairon of ILOG solver with dedicated algorithms for Valued CSP. In *Proc. of the Third ILOG International Users Meeting*. Paris, France.

38. Li, Y. H. (1997). *Directed annealing search in constraint satisfaction and optimization*. Ph.D. Thesis, Imperial College of Science, Department of Computing.

39. Minton, S., Johnson, M., Philips, A., & Laird, P. (1990). Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proc. of AAAI-90*, pages 17–24.

40. Minton, S., Johnson, M., Philips, A., & Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58: 161–205.

41. Morris, P. (1993). The breakout method for escaping from local minima. In *Proc. of AAAI-93*, pages 40–45.

42. Neufeld, G. A., & Tartar, J. (1974). Graph coloring conditions for the existence of solutions to the timetable problem. *Communications of the ACM*, 17(8): 450–453.

43. Parmee, I. (1989). The integration of evolutionary and adaptive computing technologies with product/system design and realization. Springer-Verlag, Plymouth, UK.

44. Régin, J. C., Petit, T., Bessiére, C., & Puget, J. F., (2000). An original constraint based approach for solving over constrained problems. In *Proc. CP-00*, pages 543–548.

45. Rosenthal, R. (1985). Principles of multiobjective optimization. *Decision Sciences*, 16: 133–152.

46. Ruttkay, Z., Eiben, A. E., & Raue, P. E. (1995). Improving the performance of GAs on a GA-hard CSP. In *Proceedings, CP95 Workshop on Studying and Solving Really Hard Problems*, pages 157–171.

47. Schaerf, A. (1996). Tabu search techniques for large high-school timetabling problems. In *Proc. of ECAI-96*, pages 634–639.

48. Schaerf, A. (1996). A survey of automated timetabling. *Artificial Intelligence Review*, 13: 87–127.

49. Schiex, T., Verfaille, G., & Fargier, H. (1995). Valued constraint satisfaction problems: Hard and easy problems. In *Proc. IJCAI-95*, pages 631–637.

50. Schiex, T. (2000). Arc consistency for soft constraints. In *Proc. CP-2000*, pages 411–424.

51. Srinivas, N., & Deb, K. (1993). Multi-objective optimization using nondominated sorting in genetic algorithms, Technical Report, Department of Mechanical Engineering, Indian Institute of Technology, Kanpur.

52. Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press.

53. Tsang, E., Wang, C., Davenport, A., Voudouris, C., & Lau, T. (1999). A family of stochastic methods for constraint satisfaction and optimization. In *The First International Conference on the Practical Application of Constraint Technologies and Logic Programming*, London.

54. Ulungu, E. L., Teghem, J., & Fortemps, P. H. (1997). Heuristics for multi-objective combinatorial optimization by simulated annealing. In Gu, J., Chen, G., Wei, Q., & Wang, S. (eds.), *Multicriteria Analysis*, pages 269–278. Springer-Verlag, Berlin.

55. Ulungu, E. L., Teghem, J., Fortemps, P. H., & Tuyttens, D. (1999). MOSA method: A tool for solving multi-objective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 8(4): 221–236.

56. Ulungu, E. L., Teghem, J., & Ost, C. (1998). Interactive simulate annealing in a multi-objective framework: Application to an industrial problem. *J. Oper. Res. Soc.*, 49: 1044–1050.

57. Verfaille, G., & Schiex, T. (1994). Solution reuse in dynamic constraint satisfaction problems. In *Proc. of AAAI-94*, pages 307–312.

58. Verfaillie, G., Lemaitre, M., & Schiex, T. (1996). Russian doll search for solving constraint satisfaction problems. In *Proc. of AAAI-96*, pages 181–187.

59. Wah, B., & Wang, T. (1999). Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. In *Proc. of CP-99*, pages 461–475.

60. Wallace, R. (1995). Directed arc consistency preprocessing. In Meyer, M. (ed.), *Selected Papers from the ECAI-94 Workshop on Constraint Processing*, Vol. 923 of LNCS, pages 121–137.

61. Wallace, R. (1996). Enhancements of branch and bound methods for the maximal constraint satisfaction problem. In *Proc. of AAAI-96*, pages 188–195.

62. Wallace, R., & Freuder, E. C. (1993). Conjunctive width heuristics for maximal constraint satisfaction. In *Proc. of AAAI-93*, pages 762–768.

63. Wallace R., & Freuder, E. C. (1995). Heuristics methods for over-constrained constraint satisfaction problems. In *Over-Constrained Systems*, Vol. 1106 of LNCS. (Also Workshop in CP-95.)

64. Wilson, M., & Borning, A. (1993). Hierarchical constraint logic programming. *Journal of Logic Programming*, 16: 227–318.

65. Yokoo, M. (1994). Weak-commitment search for solving constraint satisfaction problems. In *Proc. of AAAI-94*, pages 313–318.

66. Yoshikawa, M., Kaneko, K., Yamanouchi, T., & Watanabe M. (1996). A constraint-based high school scheduling system. *IEEE Expert*, 11(1): 63–72.