Bounding the Optimum of Constraint Optimization Problems

Simon de Givry[1], Gérard Verfaillie[1], and Thomas Schiex[2]

[1] ONERA-CERT, 2 av. Edouard Belin, BP4025, 31055 Toulouse Cedex 4, France, {degivry,verfaillie}@cert.fr

[2] INRA, Chemin de Borde Rouge, Auzeville, BP26, 31326 Castanet Tolosan Cedex, France, tschiex@toulouse.inra.fr

Abstract. Solving constraint optimization problems is computationally so expensive that it is often impossible to provide a guaranteed optimal solution, either when the problem is too large, or when time is bounded. In these cases, local search algorithms usually provide *good* solutions. However, and even if an optimality proof is unreachable, it is often desirable to have some guarantee on the quality of the solution found, in order to decide if it is worthwile to spend more time on the problem.

This paper is dedicated to the production of intervals, that bound as precisely as possible the optimum of *Valued Constraint Satisfaction Problems* (*VCSP*). Such intervals provide an upper bound on the distance of the best available solution to the optimum *i.e.*, on the quality of the optimization performed. Experimental results on random *VCSPs* and real problems are given.

Motivations

The *Constraint Satisfaction Problem* framework is very convenient for representing and solving various problems, related to *Artificial Intelligence* and *Operations Research*: scheduling, assignment, design... But many real overconstrained problems are more faithfully translated to *Constraint Optimization Problems*, and, more precisely, to *Valued Constraint Satisfaction Problems* [16]. The classical objective of constraint satisfaction is replaced by an objective of constraint violation minimization.

Both theoretical and practical observations show that these optimization problems are much more difficult to tackle than satisfaction problems. The construction of a provenly optimal solution is often out of reach. It is the case, either when the problem is too large and difficult, or when time and resources are bounded.

Exact or complete methods, such as *Branch and Bound*, are able to produce both an optimal solution, and a proof of optimality. But, because of their exponential worst-case behavior, they may be extremely time consuming. Moreover, it has been experimentally observed that, due to their systematic way of exploring the search space, the quality of their intermediate solutions is usually very poor [21].

Due to their opportunist way of exploring the search space, approximate or incomplete methods, based on heuristic or stochastic *Local Search* mechanisms, usually provide *good* solutions within a reasonable time. Naturally, the value of the best solution found so far is an upper bound on the optimum. But these algorithms do not provide any information about the distance between this value and the optimum. By themselves, they cannot prove the optimality of a solution and may waste a lot of time trying to improve an already optimal solution: a wasteful behavior if several problems have to be tackled in a time or resource-bounded context.

This situation can be largely improved by computing non trivial lower bounds on the optimum: the distance between, on the one hand, the value of the best solution found so far, and, on the other hand, the best lower bound produced so far, provides an upper bound on the distance to the optimum. This information can be used for deciding, either to stop the optimization process, or to spend more time, in order to get a tighter bounding of the optimum.

This paper is organized as follows: in Section 1, we introduce the *Valued Constraint Satisfaction Problem* framework; in Section 2, we show how problem simplifications can be used for producing lower bounds; we finally present, in Section 3, the results of the experiments which have been performed, both on random *VCSPs* in a time-bounded context, and on large real problems.

1 Valued Constraint Satisfaction Problems

The *Valued Constraint Satisfaction Problem* framework (*VCSP*) [16] is an extension of the *Constraint Satisfaction Problem* framework (*CSP*), which allows overconstrained problems or preferences between solutions to be dealt with. In the *VCSP* framework, a mathematical object, called a *valuation*, is associated with each constraint. The valuation of an assignment is defined as the aggregation of the valuations of the constraints which are violated by this assignment. The goal is to find a complete assignment of minimum valuation.

More formally speaking, whereas a *CSP* is defined as a triple $P = (V, D, C)$, where V is a set of variables, D a set of associated domains, and C a set of constraints between the variables, a *VCSP* can be defined as a *CSP* which is extended with:

- a valuation structure S, which is itself a quintuple $(E, \succ, \top, \perp, \circledast)$, where E is a valuation set, $\succ$ a total order on E, $\top$ and $\perp$ the maximum and the minimum elements in E, and $\circledast$ a binary closed aggregation operator on E which satisfies the following properties: commutativity, associativity, monotonicity according to $\succ$, $\top$ as absorbing element and $\perp$ as identity,
- and a valuation function φ from C to E.

The valuation set E is used for defining 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 $\circledast$. The maximum element $\top$ is used for expressing imperative constraint violation and complete inconsistency, the minimum element $\perp$ to express constraint satisfaction and complete consistency. The valuation function φ associates with each constraint a valuation which denotes its importance (the valuation of any imperative constraint equals $\top$).

Let A be an assignment of all the variables and $C_{unsat}(A, P)$ be the set of the problem constraints unsatisfied by A. The valuation $\varphi(A, P)$ of A is the aggregation of the valuations of all the constraints in $C_{unsat}(A, P)$:

$$\varphi(A,P) = \underset{c \in C_{unsat}(A,P)}{\circledast} \varphi(c)$$

The goal is then to find an assignment whose valuation is minimum and lower than $\top$ (all the imperative constraints must be satisfied). The optimal valuation $\varphi(P)$ of a problem P is the valuation of such an assignment.

Note that an equivalent framework can be defined, without any major modification, by assigning a valuation to each tuple of a constraint. Note also that a very close algebraic framework, which only assumes a partial order on E, is defined in [3].

Specific subframeworks can be defined by instantiating the valuation structure S. Table 1 shows the valuation structures which are used by the $\wedge$-*VCSP*, *max-VCSP*, *lex-VCSP*, and Σ-*VCSP* subframeworks.

- in the $\wedge$-*VCSP* framework (equivalent to the classical *CSP* framework), the goal is to find an assignment which satisfies all the constraints;
- in the *max-VCSP* framework[1] (equivalent to the possibilistic *CSP* framework), the goal is to find an assignment which minimizes the maximum valuation of the unsatisfied constraints;
- in the *lex-VCSP* framework[2] (used in the *Hierarchical Constraint Logic Programming* framework [22]), the goal is to find an assignment which minimizes the number of unsatisfied constraints of maximum valuation and then minimizes the number of unsatisfied constraints of lower valuation, and so on, down to the lowest valuation;
- in the Σ-*VCSP* framework, the goal is to find an assignment which minimizes the sum of the valuations of the unsatisfied constraints.

From an application point of view, many real problems use an additive criterion and can be cast as Σ-*VCSPs*. In the sequel of this paper, we consequently assume that the target problem is a Σ-*VCSP*.

2 Bounding the optimum of a VCSP

Let P be a *VCSP*, $\varphi(P)$ be its optimal valuation, and $P(\alpha)$ be the following *NP-complete* decision problem: does there exist an assignment of the variables in P, whose

---

1 Traditionally, the set $E = [0,1]$ is used instead of $\overline{\mathbb{N}}$, the set $\mathbb{N}$ of the natural integers, extended with the element $+\infty$. But obviously any infinite ordered set with a maximum and a minimum element may be used. We have chosen to use $\overline{\mathbb{N}}$, to be homogeneous with the other subframeworks.

2 To be consistent with our definition of the *max-VCSPs*, *lex-VCSPs* use $E = \overline{\mathbb{N}^*}$, the set of the multi-sets of elements of $\mathbb{N}$, completed with a maximum element. This set is ordered by $>^*$, the lexicographic order induced on $\overline{\mathbb{N}^*}$ by the natural order $>$ on $\overline{\mathbb{N}}$. Let us recall that a multi-set is a set which may contain several times the same element. Let E be a set, totally ordered by $\succ$. Let E^* be the set of the multi-sets of elements of E. The lexicographic order $\succ^*$, which is induced on E^* by the order $\succ$ on E, can be defined as follows: let be $e_1, e_2 \in E^*$, let m be the maximum element of $e_1 \cup e_2$ according to $\succ$, let e_1^m and e_2^m the subsets of e_1 and e_2 whose elements equal m, $e_1 \succ^* e_2 \Leftrightarrow [|e_1^m| > |e_2^m|] \vee [(|e_1^m| = |e_2^m|) \wedge (e_1 - e_1^m \succ^* e_2 - e_2^m)]$.

Subframework	Notation	E	$\succ$	$\top$	$\perp$	$\circledast$
Classical *CSP*	$\wedge$-*VCSP*	$\{t, f\}$	$f \succ t$	f	t	$\wedge$
Possibilistic *VCSP*	*max-VCSP*	$\overline{\mathbb{N}}$	$>$	$+\infty$	0	max
Lexicographic *VCSP*	*lex-VCSP*	$\overline{\mathbb{N}^*}$	$>^*$	$\top$	$\varnothing$	$\cup$
Additive *VCSP*	Σ-*VCSP*	$\overline{\mathbb{N}}$	$>$	$+\infty$	0	$+$

Table 1. Several *VCSP* subframeworks

valuation is strictly lower than α ? Bounding the optimum of P implies solving two different problems:

- to find an upper bound *i.e.*, a valuation ub which is as low as possible and such that the answer to $P(ub)$ is *yes*;
- to find a lower bound *i.e.*, a valuation lb which is as high as possible and such that the answer to $P(lb)$ is *no*.

It is interesting to observe the behaviour of a standard *Depth First Branch and Bound* algorithm, using *Forward Checking* [8, 16], when solving $P(\alpha)$ with increasing values of α. Figure 1 shows the mean number of nodes in the tree search as a function of α. Each point represents the mean value on 44 different random Σ-*VCSPs*, whose characteristics are the following: they involve 15 variables and 10 values per variable; graph connectivity and constraint tightness equal 70%; all the constraint valuations equal 1; the optimum equals 20.

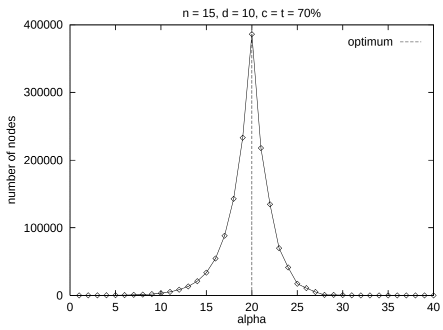


Fig. 1. Number of nodes when looking for an assignment whose valuation is strictly lower than α.

Traditionally, the parameter which appears on the x axis is one of the parameters which have been used for generating the considered instances. Here, it is a parameter of the question. But, the same traditional behavior of the *NP-complete* problems can be observed: a phase transition, which is located round the optimum, separates problems whose answer is *no*, on the left (problems whose solving provides lower bounds on the optimum), from problems whose answer is *yes* on the right (problems whose solving provides upper bounds on the optimum). This curve is enlightning on several points:

- the hardest problem to solve is the problem $P(\varphi(P))$, which consists in proving that there is no assignment whose valuation is strictly lower than the optimum; it is harder than the problem $P(\varphi(P)+1)$, which consists in producing an optimal assignment;
- a complete algorithm, like a *Branch and Bound*, whose objective is to provide an optimal solution and an optimality proof, has to solve at least both these problems;
- at the same distance from the optimum, producing an upper bound seems easier than producing a lower bound, when using a complete algorithm for both problems; this difference may become more important, when incomplete *Local Search* algorithms are used for producing quickly *good* upper bounds;
- the difficulty of bounding the optimum seems to grow exponentially, when the size of the desired interval decreases.

Incomplete methods seem well suited to the production of upper bounds. On the other hand, complete methods seem well suited to the production of lower bounds. In this paper, we show that optimally solving *simplifications* of an additive target problem can produce both interesting upper and lower bounds. More precisely, we show that simplifications are easier to solve than the target problem, and that solving them allows the optimum of the target problem to be bounded.

2.1 Simplifying a VCSP

The different methods, that will be used for transforming a *VCSP* into a hopefully simpler *VCSP*, are covered by the following definition:

Definition 1. Given two valuation structures $S = (E, \succ, \top, \perp, \circledast)$ and $S' = (E', \succ', \top', \perp', \circledast')$, a simplification function is a function Φ from E to E', such that:

$$\Phi(\perp) = \perp', \Phi(\top) = \top'$$
$$\forall e_1, e_2 \in E, e_1 \succcurlyeq e_2 \Rightarrow \Phi(e_1) \succcurlyeq' \Phi(e_2)$$

In other words, a simplification function is an order morphism. The last condition can be equivalently written $\forall e_1, e_2 \in E, e_1 \succ e_2 \Rightarrow \Phi(e_1) \succcurlyeq' \Phi(e_2)$. This shows that a simplification function may suppress some differences between valuations (two different valuations may map to the same element), but preserves the order, and the maximum and minimum elements. A simplification function between S and S' can be applied to a *VCSP* $P = (X, D, C, S, \varphi)$ and yields the *VCSP* $P' = (X, D, C, S', \Phi \circ \varphi)$, abusively noted $\Phi(P)$. Simplification functions can be composed together and the result is again a simplification function.

The following simplifications will be used in the sequel:

- given any valuation structure S and e any element of E, Φ_e is a simplification from S to S which maps to $\perp$ any valuation strictly lower than e, all the other valuations remaining unchanged; a *VCSP* simplified in this way simply ignores all the constraints whose valuation is strictly lower than e;

– from any valuation structure to $\wedge$-*VCSP*, $\Phi^{\wedge}$ is a simplification such that, $\forall e, e \neq \perp, \Phi^{\wedge}(e) = f$; it maps any *VCSP* to a classical *CSP*, where all the soft constraints are made hard;
– from Σ-*VCSP* to *max-VCSP*, Φ^{max} is simply the identity; from *lex-VCSP* to *max-VCSP*, Φ^{max} is a simplification such that, $\forall e, e \neq \emptyset, e \neq \top, \Phi^{max}(e)$ is the maximum element in the multi-set e;
– from Σ-*VCSP* to *lex-VCSP*, Φ^{lex} is a simplification such that $\forall e, e \neq 0, e \neq +\infty, \Phi^{lex}(e) = \{e\}$.

In some cases, when the simplification function is compatible with the aggregation operators which are used in S and S', it is possible to prove that the simplified *VCSP* is effectively simpler to solve.

Definition 2. A simplification function Φ between S and S' is aggregation-compatible iff $\forall E_1, E_2$, finite subsets of E:

$$\Phi(\underset{e \in E_1}{\circledast} e) \succcurlyeq' \Phi(\underset{e \in E_2}{\circledast} e) \Rightarrow \underset{e \in E_1}{\circledast'} \Phi(e) \succcurlyeq' \underset{e \in E_2}{\circledast'} \Phi(e)$$

Theorem 3. *If Φ is an aggregation-compatible simplification function, then the set of the optimal solutions of $\Phi(P)$ is a superset of the set of the optimal solutions of P. Moreover, the search tree of $\Phi(P)$ is included in the search tree of P, when using a* Depth First Branch and Bound *algorithm, the same static variable and value orderings, and the same way of computing a lower bound on the global valuation*[3] *of a partial assignment*[4].

Proof If Φ is an aggregation-compatible simplification function, it follows from Definitions 1 and 2 that, $\forall E_1, E_2$, finite subsets of E:

$$\underset{e \in E_1}{\circledast} e \succcurlyeq \underset{e \in E_2}{\circledast} e \Rightarrow \Phi(\underset{e \in E_1}{\circledast} e) \succcurlyeq' \Phi(\underset{e \in E_2}{\circledast} e) \Rightarrow \underset{e \in E_1}{\circledast'} \Phi(e) \succcurlyeq' \underset{e \in E_2}{\circledast'} \Phi(e) \quad (1)$$

Let $lb(A, P)$ be any lower bound on the global valuation of a partial assignment A in a problem P, which is computed by aggregating constraint valuations, as it is done with *Backward* or *Forward Checking*. From 1, it follows that, $\forall A, A'$, partial assignments:

$$lb(A, P) \succcurlyeq lb(A', P) \Rightarrow lb(A, \Phi(P)) \succcurlyeq' lb(A', \Phi(P)) \quad (2)$$

or equivalently:

$$lb(A', \Phi(P)) \succ' lb(A, \Phi(P)) \Rightarrow lb(A', P) \succ lb(A, P) \quad (3)$$

If A is an optimal complete assignment of P, then $\forall A'$, complete assignment, $\varphi(A', P) \succcurlyeq \varphi(A, P)$. From 2, it follows that, $\forall A'$, complete assignment, $\varphi(A', \Phi(P)) \succcurlyeq' \varphi(A, \Phi(P))$, and that A is an optimal complete assignment of $\Phi(P)$. Thus, the set of the optimal solutions of $\Phi(P)$ is a superset of the set of the optimal solutions of P.

[3] The global valuation of a partial assignment is the optimal valuation of all its complete extensions.

[4] Note that, if Φ is an aggregation-compatible simplification function, then P is a *strong refinement* of $\Phi(P)$ [16].

If A is a partial assignment, associated with a node which is expanded, when using on $\Phi(P)$ a *Depth First Branch and Bound* algorithm, static variable and value orderings, and any way of computing a lower bound on the global valuation of a partial assignment, then $\forall A'$, complete assignment, preceding A according to the search order, $\varphi(A', \Phi(P)) \succ' lb(A, \Phi(P))$: if this condition was not satisfied, A would not be expanded. From 3, it follows that, $\forall A'$, complete assignment, preceding A according to the search order, $\varphi(A', P) \succ lb(A, P)$, and that the associated node is expanded, when using on P the same algorithm, the same variable and value orderings, and the same way of computing a lower bound on the global valuation of a partial assignment. Thus, the search tree of $\Phi(P)$ is, under these conditions, included in the search tree of P $\square$

Among the previous simplification functions, it is easy to establish that:

- Φ_e is aggregation-compatible from *max-VCSP* and *lex-VCSP*, but not from Σ-*VCSP*;
- $\Phi^{\wedge}$ is aggregation-compatible from any valuation structure;
- Φ^{max} is aggregation-compatible from *lex-VCSP*, but not from Σ-*VCSP*;
- Φ^{lex} is not aggregation-compatible from Σ-*VCSP*.

From Σ-*VCSP*, even if Φ_e is not aggregation-compatible, it has been experimentally observed that complexity globally decreases with the number of constraints[5]. In the same way, even if Φ^{max} and Φ^{lex} are not aggregation-compatible, it has been experimentally observed that solving *max-VCSPs* and *lex-VCSPs* is generally simpler than solving Σ-*VCSPs*.

2.2 Building lower bounds from simplifications

Given a problem P and a simplification function Φ, the valuation in P of any optimal solution of $\Phi(P)$ is obviously an upper bound on the optimum of P. Interestingly, the optimum of $\Phi(P)$, or any lower bound on this optimum, can also be used for producing a lower bound on the optimum of P. This is captured by the notion of *transfer function*.

Definition 4. Given a simplification function Φ between S and S', a function Ψ from E' to E is a transfer function for Φ iff:

$$\forall E_1, \text{ finite subset of } E, \forall e' \in E', \ \underset{e \in E_1}{\circledast'} \Phi(e) \succcurlyeq' e' \Rightarrow \underset{e \in E_1}{\circledast} e \succcurlyeq \Psi(e')$$

Theorem 5. *If Ψ is a transfer function for the simplification function Φ between S and S', and if e' is a lower bound on the optimum of $\Phi(P)$, then $\Psi(e')$ is a lower bound on the optimum of P. Particularly:*

$$\varphi(P) \succcurlyeq \Psi(\varphi(\Phi(P))$$

[5] It remains true with algorithms such as *Russian Doll Search* [19] or *Forward Checking* enhanced with *Directed Arc Consistency* [20, 10], for which what [11] call a phase transition occurs: complexity decreases when constraint tightness increases beyond a point; but, even with these algorithms, complexity continues to increase regularly with graph connectivity.

The proof is straightforward. Note that a trivial transfer function always exists: $\forall e' \in E', \Psi(e') = \bot$. Fortunately, it is possible to define more useful transfer functions. For example, for any valuation structure, identity is a transfer function for the simplification Φ_e.

For any target *Σ-VCSP* P, let $(e_1, \ldots, e_k)$ be the different valuations which are associated with the constraints in P, from the highest to the lowest, let n_i be the number of constraints in P whose valuation equals e_i, let m_i the number of constraints in P which are violated by an assignment A and whose valuation equals e_i, and let $(e_1^{m_1}, \ldots, e_k^{m_k})$ be the notation we use for representing the lexicographic valuation of A. The following transfer functions can be established[6]:

- $\Psi^{\wedge}(t) = 0, \Psi^{\wedge}(f) = e_k$ is a transfer function for the simplification $\Phi_{\wedge}$;
- $\Psi^{max}(e) = e$ (identity) is a transfer function for the simplification Φ_{max};
- $\Psi^{lex}(e_1^{m_1}, \ldots, e_k^{m_k}) = lex(e_1^{m_1}, \ldots, e_k^{m_k})$ is a transfer function for the simplification Φ^{lex}, where lex is defined as follows:

  $lex() = 0$
  $lex(e_i^{m_i}, \ldots, e_k^{m_k}) =$
  if $m_i = n_i$, $(n_i.e_i) + lex(e_{i+1}^{m_{i+1}}, \ldots, e_k^{m_k})$
  else $\min[(m_i + 1).e_i, (m_i.e_i) + lex(e_{i+1}^{m_{i+1}}, \ldots, e_k^{m_k})]$

Note that the optimum of a simplified problem is not directly a lower bound on the optimum of the target problem. We need a transfer function to get a lower bound. The notion of simplification is therefore more general than the notion of relaxation widely used in *Operations Research*.

2.3 Other approaches

Other ways of computing lower bounds could be considered:

- we saw that removing some constraints and solving the resulting problem produces a lower bound; it is possible to do that in order to obtain, either a particular constraint graph (for example a tree, which can be polynomially solved), or a problem decomposition; it is also possible to produce a partition of the problem constraints and to use the property that the combination of the optimal valuations of the resulting subproblems is a lower bound;
- it is possible to remove some forbidden tuples from the constraints and to exploit the property that the optimal valuation of the resulting problem is a lower bound; as previously with the constraints, it may be interesting to do that in order to obtain particular constraints; a related approach would consist in merging similar values (values with a similar neighborhood) in a single value and using it as an abstract value as in [18];
- *Iterative Deepening* [9], which solves successive problems with an increasing value for the initial upper bound α, is a natural way of producing lower bounds: when no solution is found, α is a lower bound;

[6] Transfer functions can be also established for *lex-VCSP* (from *∧-VCSP* and *max-VCSP*) and for *max-VCSP* (from *∧-VCSP*).

- with a *Best First Branch and Bound* algorithm, at any time, the minimum of the lower bounds of the pending nodes is a lower bound, and, in the same way, any *Depth-Bounded Branch and Bound* algorithm produces a lower bound;
- another approach consists in using an ϵ-*Optimal Branch and Bound* algorithm [14], which guarantees to find a solution whose valuation is less than or equal to $\varphi(P)/\epsilon, 0 < \epsilon < 1$: if e is the valuation of the solution found by such an algorithm, e is an upper bound and $\epsilon.e$ a lower bound;
- *Local Consistency* properties can be used; two directions seem possible; one can use the fact the optimum valuation of a problem is also the valuation of an optimal consistent relaxation, as shown in [16]; it is therefore possible to look for an optimal locally consistent relaxation, whose valuation provides a lower bound; another approach consists in using the *Directional Local Consistency* properties which are exploited in [20, 10];
- finally, any Σ-*VCSP* can be translated into the $0/1$ *Integer Linear Programming* framework [13], which offers several ways of computing lower bounds.

Except with a *Tree-structured* simplification and with *Directional Local Consistency*, computing such lower bounds implies solving one or several *NP-hard* problems. The hope is that, even if these problems remain *NP-hard*, they are simpler to solve than the target problem.

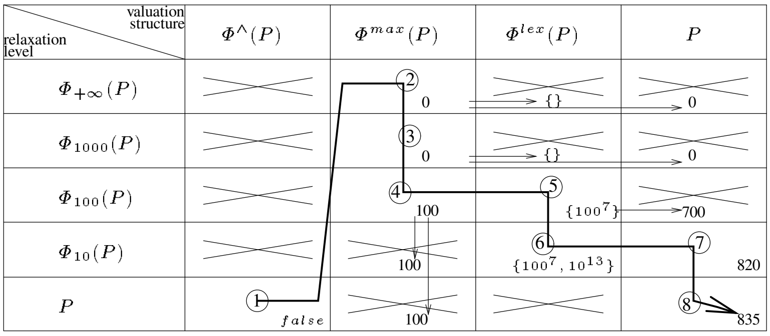
Obviously, it is not possible to assess all these approaches in the framework of this paper. In the next section, we show how it is possible to exploit the simplifications which have been presented in Section 2.1: modification of the valuation structure and relaxation of the less important constraints.

2.4 How to use simplifications

In this section, we describe how the simplifications presented in 2.1 and 2.2 can be used to produce tighter and tighter lower bounds.

To schedule the successive simplifications, we follow an order of increasing complexity, which provides us with an increasing lower bound: from the $\wedge$-*VCSP* to the Σ-*VCSP* valuation structure and, for each valuation structure except for the first, from the maximum to the minimum relaxation level. Each time a simplification is solved, existing transfer functions are used for updating the lower bounds on the target problem P and on the next simplifications. Moreover, each time a simplification finds a better solution, the upper bounds on P and on the next simplifications may be updated. Therefore, some simplifications can be short circuited, either if their optimal valuation can be deduced from the previous simplifications, or if it can be established that solving them will not improve the lower bound on P. This process stops as soon as, on P, the upper bound meets the lower bound. Each simplification is solved by a *Depth First Branch and Bound* algorithm, using *Forward Checking*, and dynamic variable and value orderings. The detection of connected components allows us to speed up each solving, especially with the highest relaxations levels.

Figure 2 illustrates this scheduling on a Σ-*VCSP* which involves constraints whose valuation equals $+\infty$, 1000, 100, 10 or 1, and whose optimal valuation equals 835. Each box is associated with a particular simplification, which is a combination of a constraint relaxation and of a modification of the valuation structure (for example, at the step 5, the simplification $\Phi^{lex}_{100}(P)$ is the result of the application to P of the simplifications Φ_{100} and Φ^{lex}). It contains the optimum of the associated simplification which has been obtained, either by a complete solving (denoted by an encircled step number), or by a deduction from a previous simplification (denoted by an arrow).



step	0	1	4	5	6	7	8
lower bound	0	1	100	700	800	820	835

Fig. 2. An example of simplification scheduling

The successive simplification steps are:

- step 1 the simplification $\Phi^{\wedge}(P)$ is solved; as it is inconsistent, a first lower bound on the optimum of P, equal to 1, can be deduced, using the transfer function $\Psi^{\wedge}$;
- steps 2-3 the simplifications $\Phi^{max}_{+\infty}(P)$ and $\Phi^{max}_{1000}(P)$ are solved; as they are consistent, we can deduce that the simplifications $\Phi^{lex}_{+\infty}(P)$, $\Phi^{lex}_{1000}(P)$, $\Phi^{\Sigma}_{+\infty}(P)$, and $\Phi^{\Sigma}_{1000}(P)$ are also consistent; they will not be solved;
- step 4 the simplification $\Phi^{max}_{100}(P)$ is solved; as it is inconsistent, a second lower bound, equal to 100, can be deduced, using the transfer functions Ψ^{max} and Ψ_{100}; we can also deduce that the simplifications $\Phi^{max}_{10}(P)$ and $\Phi^{max}(P)$ have the same optimal valuation (100); we can avoid solving them;
- step 5 the simplification $\Phi^{lex}_{100}(P)$ is solved; as its optimal valuation is $\{100^7\}$ (any optimal solution violates 7 constraints, whose valuation equals 100), a third lower bound, equal to 700, can be deduced, using the transfer functions Ψ^{lex} and Ψ_{100}; we can also deduce that the optimal valuation of the simplification $\Phi^{\Sigma}_{100}(P)$ equals 700; it will not be solved;
- step 6 the simplification $\Phi^{lex}_{10}(P)$ is solved; as its optimal valuation is $\{100^7, 10^{13}\}$, a fourth lower bound, equal to 800, can be deduced, using the transfer functions Ψ^{lex} and Ψ_{10}; it is also possible to establish that solving the following lexicographic simplification $\Phi^{lex}(P)$ cannot improve the lower bound on the optimum of P; we can avoid solving it;
- step 7 the simplification $\Phi^{\Sigma}_{10}(P)$ is solved; as its optimal valuation equals 820, a fifth lower bound, also equal to 820, can be deduced, using the transfer function Ψ_{10};

step 8 we finally solve the target problem P, whose optimal valuation equals 835.

This scheduling and the resolution of the successive simplifications could be further improved:

- by using a dichotomic order, instead of the sequential order, to process the possibilistic simplifications;
- by applying *Local Consistency Enforcing* methods [12, 7], like *Arc Consistency*, to the imperative constraints, either statically as a pre-processing, or dynamically during the search [15, 2];
- by exploiting the fact that constraints, whose valuation is greater than or equal to the initial upper bound or becomes, during the search, greater than or equal to the current upper bound, can be considered as imperative constraints;
- by using more efficient optimization algorithms, like *Russian Doll Search* [19] or *Forward Checking* enhanced with *Directed Arc Consistency* [20, 10];
- by using *Learning* mechanisms ([6, 17] for *CSP*, [5] for *VCSP*), in order to reuse the previous searches, when solving the current simplification;
- and finally, by using *Local Search* algorithms for producing quickly *good* upper bounds.

3 Experiments

3.1 Random VCSPs

The scheduling, which is described above, has been experimented on randomly generated binary Σ-*VCSPs*. Classical parameters were used for generating the problems: number of variables (n), domain size (d), graph connectivity (c) and constraint tightness (t). k different valuations were used for the constraint valuations. Each valuation was associated with the same number of constraints. The constraints which are associated with each valuation were randomly chosen.

We used $n = 15, d = 10, c = 80\%, t$ varying from 10% to 90% in steps of 10%, $k = 4$, with the different valuations in $\{1000, 100, 10, 1\}$, and 100 instances for each value of the parameters.

Figure 3 shows, for a particular problem, the typical evolution of the upper bound which is produced by a classical *Depth First Branch and Bound* algorithm, a *Simulated Annealing* algorithm[7], and of the upper and lower bounds which are produced by solving successive simplifications.

Figure 4 shows the same kind of result for a set of problems: each point corresponds to the mean value, on 100 different problems, of the quality of the bound[8].

Concerning the lower bound which is produced by the successive simplifications, one can observe that it regularly increases and gets very close to the optimum, usually

[7] *Simulated Annealing* is not the fastest *Local Search* algorithm we tried, but it is able to get very close to the optimum for a wide range of problems [1].

[8] Let e be the sum of the valuations of the problem constraints. For the upper bound, the measure of the quality is the ratio $(e - ub)/(e - \varphi(P))$. For the lower bound, it is the ratio $lb/\varphi(P)$. These ratios equal 0 at the beginning of the search, and 1 at the end.

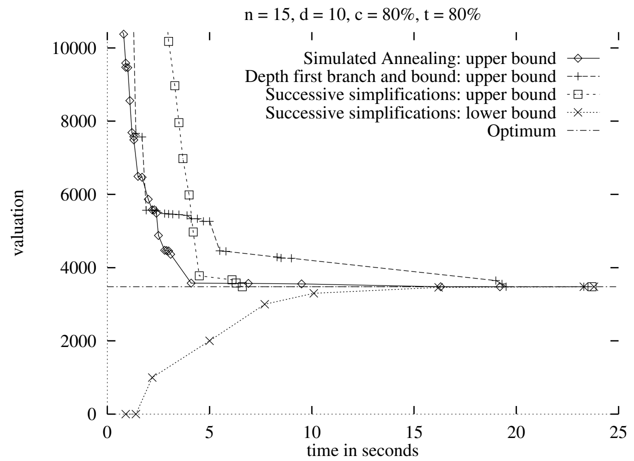


Fig. 3. Upper and lower bound evolutions on a particular problem

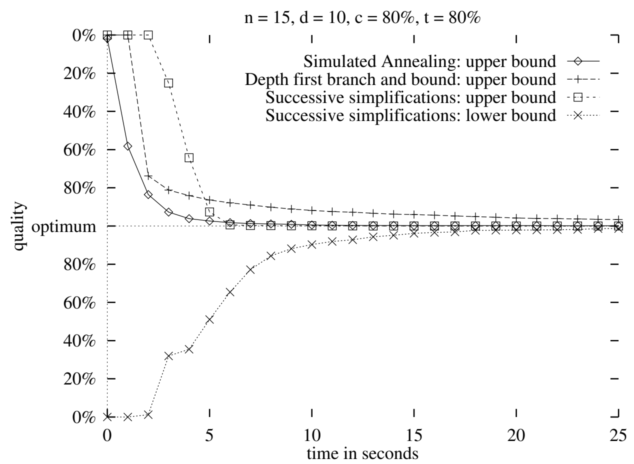


Fig. 4. Mean upper and lower bound evolutions on a set of problems

far before the problem is completely solved by the classical *Depth First Branch and Bound* algorithm. This confirms the interest of the approach: no lower bound is produced by the *Depth First Branch and Bound* algorithm, before it completely solves the problem; within a more limited time, solving the successive simplifications produces very relevant lower bounds. If we look at Fig. 4, we can see that only 10 seconds are needed to be at less than 10% of the optimum with the simplifications, while the mean solving time is about 46 seconds with the *Depth First Branch and Bound* algorithm.

Concerning the upper bounds which are produced, either by the *Depth First Branch and Bound* algorithm, or by the successive simplifications, it is surprising to observe that the latter generally decreases more rapidly than the former. This phenomenon can be explained by the fact that the search trees associated with the successive simplifications are much smaller than the search tree associated with the target problem, and that the search can be consequently much more diversified in the space of the possible assignments. Note that a local search algorithm generally produces even better results, but may be unable to get the optimum.

By varying constraint tightness, Figure 5 shows the mean profiles of the upper and lower bounds which are produced by the successive simplifications on a larger spectrum of problems. This shows that the approach is meaningful on a wide range of problems. However one can observe that the interval which is defined by both the bounds converges more slowly, when constraint tightness increases. This can be explained by the fact that the successive simplifications are more and more difficult to solve.

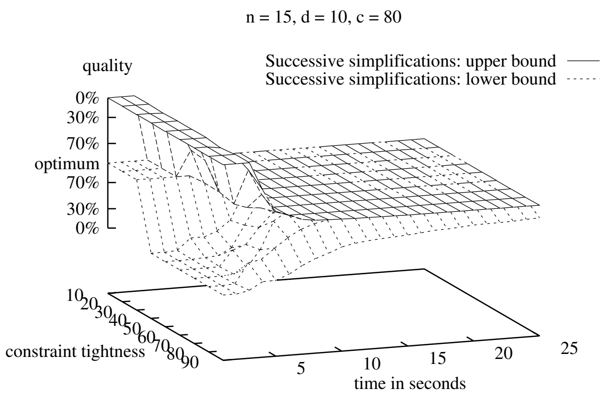


Fig. 5. Mean upper and lower bound evolutions with a varying constraint tigthness

3.2 Real size problems

The French *Centre d'Electronique de l'Armement* (*CELAR*) has made available a set of 11 instances of the *Radio Link Frequency Assignment* problem [4]. These instances have been tackled by several teams, in the framework of the European *EUCLID CALMA* project. Among them, the instances 6,7,8,9 and 10 can be naturally cast as Σ-*VCSPs*.

An *ad hoc* technique was designed during the project, in order to produce lower bounds. It yielded interesting lower bounds on instances 9 and 10, but provided nothing on instances 6, 7 and 8. The instance 6 involves 200 variables, at most 44 values per variable, and 1322 constraints. Like all the *CELAR* instances, it can be preprocessed in order to divide the number of variables by 2. Constraint valuations equal 1000, 100, 10 or 1. It is, with the instance 7, one of the hardest. By now, nobody had been able to produce a guaranteed optimal solution.

Since real problems are usually more structured than random *VCSPs*, an *ad hoc* simplification process, instead of the previously described successive simplifications, has been used, for producing a lower bound on the optimum of this instance: rather than removing constraints whose valuation is lower than a threshold, several strongly inter-connected subproblems[9] have been extracted and solved to optimality. Since all these subproblems involve disjunct subsets of constraints, it is possible to combine their optimal valuations to get a global lower bound of 3389, which is exactly the same as the best known upper bound of 3389.

Conclusion

When a problem is too large and too difficult to be solved to optimality, the traditional approach is to forget optimality and to use *Local Search* algorithms for finding a *good* solution, within a reasonable time. It is however well known, that these algorithms may have a *pathological* behavior on structured instances, that they do not provide any information about the distance to the optimum, and may waste consequently a lot of time trying to improve an optimal solution.

This paper shows that, rather than giving up the idea of getting information about the optimum, one can build lower bounds on this optimum, by solving simplifications of the target problem. We have shown, both on random and real instances, that this approach is effective, even when the optimum is not reachable within the available time.

Finally, we have not considered an important possible use of these lower bounds: at each node, during the search, a high lower bound on the global valuation of the partial associated assignment *i.e.*, on the valuation of the problem restricted by this assignment, is the best means to backtrack early. This may look unreasonnable since nearly all the problems associated with the simplifications we considered are *NP-hard*. However, the *Operations Research* community has shown that such a way may be effective.

References

1. E. Aarts and J. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.
2. C. Bessière and J.C. Régin. MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?). In *Proc. of the 2nd International Conference on Principles and Practice of Constraint Programming (CP-96, LNCS 1118)*, pages 61–75, Cambridge, MA, USA, 1996.

[9] The larger sub-problem involves 22 variables. Globally, about three days were needed to solve all the subproblems, using the *Russian Doll Search* algorithm [19], running on a PVM network of 40 *SPARCstations 4*.

3. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 624–630, Montréal, Canada, 1995.
4. CELAR. Radio Link Frequency Assignment Problem Benchmark. URL ftp://ftp.cs.unh.edu/pub/csp/archive/code/benchmarks, 1994. See also http://www-bia.inra.fr/T/schiex or http://www.cert.fr/francais/deri/verfaillie/oc.html.
5. P. Dago and G. Verfaillie. Nogood Recording for Valued Constraint Satisfaction Problems. In *Proc. of the 8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-96)*, pages 132–139, Toulouse, France, 1996.
6. R. Dechter. Enhancement Schemes for Constraint Processing : Backjumping, Learning and Cutset Decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.
7. E. Freuder. Synthesizing Constraint Expressions. *Communications of the ACM*, 21:958–966, 1978.
8. E. Freuder and R. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
9. R. Korf. Depth-First Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27:97–109, 1985.
10. J. Larrosa and P. Meseguer. Expoiting the Use of DAC in MAX-CSP. In *Proc. of the 2nd International Conference on Principles and Practice of Constraint Programming (CP-96, LNCS 1118)*, pages 308–322, Cambridge, MA, USA, 1996.
11. J. Larrosa and P. Meseguer. Phase Transition in MAX-CSP. In *Proc. of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 190–194, Budapest, Hungary, 1996.
12. A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.
13. G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.
14. J. Pearl. *HEURISTICS, Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, 1984.
15. D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 125–129, Amsterdam, The Netherlands, 1994.
16. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 631–637, Montréal, Canada, 1995.
17. T. Schiex and G. Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools*, 3(2):187–207, 1994.
18. R. Shrag and D. Miranker. Abstraction and the CSP Phase Transition Boundary. In *Proc. of the 4th International Symposium on Artificial Intelligence and Mathematics*, pages 138–141, Fort Lauderdale, FL, USA, 1996.
19. G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 181–187, Portland, OR, USA, 1996.
20. R. Wallace. Directed Arc Consistency Preprocessing. In *Proc. of the ECAI-94 Workshop on Constraint Processing (LNCS 923)*, pages 121–137. Springer, 1994.
21. R. Wallace. Analysis of Heuristic Methods for Partial Constraint Satisfaction Problems. In *Proc. of the 2nd International Conference on Principles and Practice of Constraint Programming (CP-96, LNCS 1118)*, pages 482–496, Cambridge, MA, USA, 1996.
22. M. Wilson and A. Borning. Hierarchical Constraint Logic Programming. *Journal of Logic Programming*, 16(3):277–318, 1993.