Optimum Anytime Bounding for Constraint Optimization Problems

Simon de Givry and **Gérard Verfaillie**
ONERA-CERT/DERI
2 av. Edouard Belin, BP 4025, 31055 Toulouse Cedex 4, France
{degivry,verfaillie}@cert.fr

Abstract

In this paper, we consider *Constraint Optimization Problems* in a *Resource-Bounded context*. We observe that both exact and approximate methods produce only an anytime upper bound of the optimum (in case of minimization). No lower bound, and thus no *quality* is available at run time. For a meta-reasoning system, it is difficult to reason on the basis of a so poor piece of information. Therefore, we discuss some ways of producing an *anytime lower bound*. In the *Valued Constraint Satisfaction Problem* framework, we develop some of them, based on the complete solving of problem *simplifications*, and we present experimental results.

Motivation

There are some difficulties in considering constraint optimization problems within a resource bounded framework.

Most of these problems are *NP-hard*. The worst-case time, which is needed to solve them optimally, grows exponentially with the size of the problem. The mean or the median time, which can be experimentally observed, is far lower than the worst-case time. The observed variance may be very large.

Approximate methods, like *Greedy*, *Hill Climbing*, *Tabu*, *Simulated Annealing*, or *Genetic* algorithms, generally allow good solutions to be rapidly produced. Their run time can fit any deadline. But they give no information about the distance between the best value of the criterion which has been obtained and the problem optimum. They can never prove the optimality of a solution and may waste a lot of time trying to improve an already optimal solution.

Exact methods, like *Best-First* or *Depth-First Branch and Bound*, are able to produce optimal solutions and to prove their optimality. But, due to the unknown and potentially extremely large amount of necessary backtracking, the time to get this result is unpredictable and may be huge. Moreover, it has been experimentally observed that, due to their strict search ordering, the quality of their intermediate solutions is often very poor (Wallace & Freuder 1995).

Both kinds of methods are interruptible: as soon as a first solution has been produced (what corresponds to the *mandatory part* of the task (Liu *et al.* 1991)), they can deliver the best solution found so far, but the value of the criterion for this solution is just an upper bound of the problem optimum (in case of minimization). Except when using a *Best First Branch and Bound* method, no lower bound is available.

In such a situation, an *Anytime* approach (Dean & Boddy 1988) seems promising. The reasoning tasks are interruptible. What is called a scheduler in the Real Time community or a meta-reasoning system in the Artificial Intelligence community (Horvitz 1990; Boddy 1991; Russel & Wefald 1991; Strosnider & Paul 1994; Musliner *et al.* 1995; Adelantado & de Givry 1995) is responsible for organizing the different tasks and trading time for quality. To do that, it uses information about the external environnement, the state and the capacity of the resources, and the current and the expected state of the tasks.

The problem is that, given the classical methods which are used to solve constraint optimization problems, the information the scheduling or meta-reasoning system can get from the reasoning tasks is very poor:

- concerning the time, which will be necessary to produce the optimum and to prove its optimality, the worst-case time is too far from the mean or the median observed time to be useful; a probabilistic information would need the framework of the considered problems to be precised and a lot of experiments to be carried out in this framework; it may be impossible or costly to do that; due to the generally large observed variance, a mean time is not more useful; off-line scheduling the tasks seems consequently unpracticable;
- concerning the value of the optimum, it is generally unknown; although an upper bound can be rapidly

produced, no lower bound is available, except when using a *Best First Branch and Bound* method; if the measure of the quality of a solution is the distance between the value of the criterion for this solution and the optimum, this quality is not available (not *recognizable* (Zilberstein 1996)) at run time, except when the problem has been completely solved by an exact method: the distance is then null and the quality is maximum;

- concerning eventual *Performance Profiles* (Boddy & Dean 1994), they are difficult to be produced, due to the fact that the value of the optimum and the time, which is necessary to obtain a given value of the criterion, are both unknown; exact performance profiles have no sense in this framework; mean profiles do not have much more sense due to the observed variance; probabilistic conditional or dynamic profiles (Zilberstein 1996; Hansen & Zilberstein 1996) might have more sense, but they need a lot of work to be off-line produced.

In this paper, we propose a limited, but potentially interesting, improvement of the existing methods, which consists in producing an anytime bounding of the problem optimum, that is an upper bound and a lower bound of the optimum, which improve with time, in order to provide the user, a scheduling or meta-reasoning system with a more and more precise information. The upper bound is the value of the best solution found so far. The lower bound is the best-case value of the optimum. The difference is then the worst-case quality of the best solution.

Producing an upper bound is not a problem: existing methods produce one naturally. Producing an interesting lower bound is more difficult. In the *Valued Constraint Satisfaction Problem* framework (Schiex, Fargier, & Verfaillie 1995), which is an extension of the classical *Constraint Satisfaction Problem* framework (Mackworth 1992), we present several means to produce and improve a lower bound. We develop some of them and present some interesting experimental results.

Constraint Optimization Problems

The *Constraint Satisfaction Problem* framework (*CSP*) is widely used for representing and solving *Artificial Intelligence* problems, like planning, scheduling, diagnosis, design ... The *Valued Constraint Satisfaction Problem* framework (*VCSP*) is an extension of the previous framework, which allows overconstrained problems or preferences between solutions to be dealt with.

Valued Constraint Satisfaction Problems

Unformally speaking:

- in the *CSP* framework, all the constraints are imperative; an assignment of the problem variables is either consistent (all the constraints are satisfied), or inconsistent (at least one constraint is unsatisfied); the goal is to find a consistent assignment;

- in the *VCSP* framework, some constraints are imperative, the others are not; a valuation is associated with each constraint; the valuation of an assignment of the problem variables results from the aggregation of the valuations of the unsatisfied constraints; the goal is to find an assignment of optimal 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 variables, a *VCSP* can be defined as a *CSP*, which is extended with:

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

- and a valuation function φ from C to 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 $\otimes$. The maximum element $\top$ is used to express imperative constraint violation and complete inconsistency, the minimum element $\bot$ 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 the problem 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 of the constraints in $C_{unsat}(A, P)$: $\varphi(A, P) = \otimes_{c \in C_{unsat}(A,P)} \ \varphi(c)$

Then, the goal is 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 by assigning a valuation to each tuple of a constraint.

Specific subframeworks can also be defined by instantiating the valuation structure S. Figure 1 shows

Subframework	Notation	E	$\succ$	$\top$	$\perp$	$\otimes$
Classical *CSP*	$\wedge$-*VCSP*	{true,false}	false $\succ$ true	false	true	$\wedge$
Possibilistic *VCSP*	*max-VCSP*	$\overline{N}$	$>$	$+\infty$	0	max
Lexicographic *VCSP*	*lex-VCSP*	$\overline{N}^*$	$>^*$	$\{+\infty\}$	$\emptyset$	$\cup$
Additive *VCSP*	Σ-*VCSP*	$\overline{N}$	$>$	$+\infty$	0	$+$

Figure 1: Several *VCSP* subframeworks

the valuation structures, which are used by the *classical CSPs* and the *possibilistic*, *lexicographic*, and *additive VCSPs*[1]. In the $\wedge$-*VCSP* framework, the goal is to find an assignment which satisfies all of the constraints. In the *max-VCSP* framework, the goal is to find an assignment which minimizes the maximum valuation of the unsatisfied constraints. In the *lex-VCSP* framework, the goal is to find an assignment which minimizes the number of unsatisfied constraints of maximum valuation and, in case of equality, minimizes the number of unsatisfied constraints of lower valuation, and so on, until eventually considering the number of unsatisfied constraints of 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.

Solving Methods

Three kinds of methods are classically used for solving *CSPs*: *filtering*, exact *tree search*, and approximate *local search* methods.

The *filtering* methods, also called *constraint propagation* or *consistency enforcing* methods (*arc*, *path*, *i*, *i-j-consistency* (Freuder 1978)) can be extended to the *VCSP* framework, but only when the aggregation operator is idempotent[2]. It is the case with the *max-VCSPs*, but not with the *lex-VCSPs* and Σ-*VCSPs* (Schiex, Fargier, & Verfaillie 1995; Bistarelli, Montanari, & Rossi 1995).

Concerning the exact *tree search* methods, the natural extension of the *Backtrack* algorithm is a *Depth First Branch and Bound* algorithm, which can use variable and value heuristics, and limited filtering methods, like *Forward Checking* (Freuder & Wallace 1992). Due to the limited amount of memory, *Best First Branch and Bound* and *Dynamic Programming* are more rarely used: the former because of the potentially exponential number of nodes to be managed, the latter because of the potentially exponential number of subproblems to be considered.

Concerning the approximate *local search* methods, there is no fundamental problem to adapt general methods like *Greedy*, *Hill Climbing*, *Tabu*, *Simulated Annealing*, or *Genetic* algorithms, to this particular framework.

From an application point of view, many real problems use an additive criterion and can be cast as Σ-*VCSPs*. But, from an algorithmic point of view, it has been observed that they are the most difficult to be optimally solved (Schiex, Fargier, & Verfaillie 1995). They are generally more difficult than the *lex-VCSPs* and far more difficult than the *max-VCSPs* and $\wedge$-*VCSP*. In the sequel of this paper, we consequently assume that the target problem is a Σ-*VCSP*.

Optimum Anytime Bounding

As we already said, producing a better and better upper bound is easy. The existing exact or approximate methods produce one naturally. What is difficult is to produce a more and more accurate lower bound. To do that, our approach consists in considering modifications of the target problem, which are *a priori* easier to be solved and whose complete solving allows lower bounds of the target problem to be deduced. We use the term *simplifications* to refer to such problem modifications.

Possible Simplifications

Here are some possible simplifications:

- modifying the constraint graph: it is always possible to remove some constraints and to exploit the property that the optimal valuation of the resulting problem is a lower bound; in some cases, it is possible to do that in order to obtain a particular constraint graph, which is easier to be managed (for example, a tree); 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 also a lower bound; by considering increasing subsets of variables, the *Russian Doll Search* algorithm (Verfaillie, Lemaître, & Schiex 1996) exploits the same idea;

- modifying the constraints: it is always 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; it may be interesting to do that in order to obtain particular

[1] $\overline{N}$ is the set N of the natural integers, extented with the element $+\infty$; $\overline{N}^*$ is the set of the multi-sets of elements of $\overline{N}$; $>^*$ is the lexicographic order, which is induced on $\overline{N}^*$ by the natural order $>$ on $\overline{N}$

[2] An operator $\otimes$ is said idempotent iff $\forall a \in E, a \otimes a = a$.

constraints; another way would consist in exploiting the notion of *neighborhood substitutability* (Freuder 1991) or a weaker notion of *similar values*, in order to aggregate some domain values;

- modifying the valuation structure: following the experimental observation that the *max-VCSPs* and $\wedge$-*VCSPs* are easier to be solved than the *lex-VCSPs* and that the latter are themselves easier to be solved than the Σ-*VCSPs*, it may be interesting to use a simpler valuation structure, without modifying variables, domains, constraints, and valuation function; we show further how to deduce a lower bound from the optimal valuation of the problem resulting from such a modification;

- modifying the initial upper bound: at the beginning of the search, it is always possible to define an initial upper bound ub_{init} (any assignment whose valuation is greater than or equal to ub_{init} is considered as unacceptable); the *Iterative Deepening* method (Korf 1985) solves successive problems with an increasing ub_{init} ; when a problem is inconsistent, the corresponding ub_{init} is a lower bound;

- modifying the optimality objective: another method consists in using an ϵ-*optimal Branch and Bound* algorithm which guarantees to find a solution whose valuation is less than or equal to $\varphi(P)/\epsilon, 0 < \epsilon < 1$: let ub be the valuation of the solution found by such an algorithm; $ub \times \epsilon$ is a lower bound;

- modifying the consistency property: as it has been shown in (Schiex, Fargier, & Verfaillie 1995), finding an optimal assigment and finding an optimal consistent problem relaxation are two equivalent problems; in the latter, it is possible to use a weakened notion of consistency, like, for example, *arc-consistency*; the valuation of an optimal more weakly consistent relaxation is a lower bound;

- using the 0-1 *Integer Linear Programming* framework (*ILP*) and relaxing the integrity constraint: any Σ-*VCSP* can be cast as an 0-1 *ILP* problem, by associating a 0-1 *ILP* variable with each *VCSP* value; the result of the relaxation of the integrity constraint on the *ILP* variables is a lower bound.

Another approach would consist in using a *Best First Branch and Bound* algorithm on the target problem or on any simplification: at any step of the search, the minimum of the lower bounds of the pending nodes is a lower bound of the considered problem.

Selected Simplifications

We have chosen to use and to combine two types of simplification: modification of the constraint graph and modification of the valuation structure.

Let P be the target Σ-*VCSP* problem, and $\{l_1 \ldots l_i \ldots l_k\}$ be the set E of the constraint valuations (ordered from the lowest to the highest). The simplification P_i^s denotes the problem P without the constraints whose valuation is lower than l_i (relaxation level) and with the valuation structure $s, s \in \{\wedge\text{-}VCSP, max\text{-}VCSP, lex\text{-}VCSP, \Sigma\text{-}VCSP\}$. P_1^{Σ} is the problem P.

Simplified Solving

Experimental results show that the complexity of a complete solving increases with the number of constraints and with the valuation structure, following the order $\wedge$-*VCSP*, *max-VCSP*, *lex-VCSP*, Σ-*VCSP*.

It is possible to obtain a partial theoretical confirmation of these results, by using the notion of *strong refinement* (Schiex, Fargier, & Verfaillie 1995):

Definition 1 *P is a strong refinement of P' iff*
$\forall A, \forall A'$, *partial assignments*[3],
$$\varphi(A, P') \prec_{P'} \varphi(A', P') \Rightarrow \varphi(A, P) \prec_P \varphi(A', P)$$

Theorem 1 *If P is a strong refinement of P', then the set of the optimal solutions of P is a subset of the set of the optimal solutions of P', and the tree search of P' is included in the tree search of P*[4].

This theorem induces a partial order on the considered simplifications. In Figure 2, an arrow from P' to P means that P is a strong refinement of P', and that solving P' is easier than solving P.

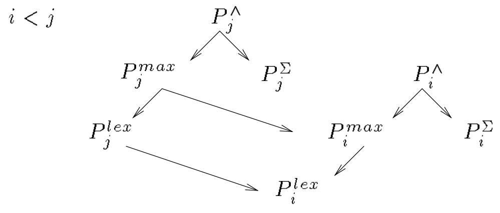


Figure 2: Partial order on selected simplifications

Producing an Upper and a Lower Bound

The result of the evaluation, in the target problem, of any solution of a simplification is an upper bound.

To deduce a lower bound from the optimal valuation of a simplification, we use the notion of *transfer function*:

[3] A partial assignment is an assignment of any subset of the problem variables. A complete assignment is an assignment of all of the problem variables.

[4] At least, with a *Branch and Bound* algorithm using *Backward Checking* and static variable and value orderings.

Definition 2 *A function $f_{P'P}$ from E' to E is called* transfer function *iff*

$$\forall A, \textit{complete assignment}, \forall e' \in E',$$
$$\varphi(A, P') \succeq_{P'} e' \Rightarrow \varphi(A, P) \succeq_P f_{P'P}(e')$$

Theorem 2 *If $f_{P'P}$ is a* transfer function, *then $\varphi(P) \succeq_P f_{P'P}(\varphi(P'))$.*

The following *transfer functions* can be established:

$$\begin{array}{rcl} \forall i, f_{P_i^{\wedge} P}(true) & = & 0 \\ f_{P_i^{\wedge} P}(false) & = & l_i \\ \forall i, f_{P_i^{max} P}(l_j) & = & l_j \\ \forall i, f_{P_i^{lex} P}(m_i, \ldots, m_k) & = & lexadd(m_i, \ldots, m_k) \\ \forall i, f_{P_i^{\Sigma} P}(e) & = & e \end{array}$$

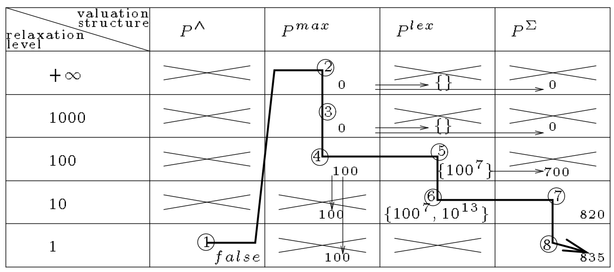
If n_j is the number of problem constraints and m_j the number of unsatisfied constraints at the level j:

$$\begin{array}{lcl} m_j = n_j, & & \\ lexadd(m_i, \ldots, m_j) & = & n_j \times l_j + lexadd(m_i, \ldots, m_{j-1}) \\ m_j < n_j, & & \\ lexadd(m_i, \ldots, m_j) & = & \min[(m_j + 1) \times l_j, \\ & & \quad m_j \times l_j + lexadd(m_i, \ldots, m_{j-1})] \\ lexadd() & = & 0 \end{array}$$

Scheduling the Simplifications

To schedule the successive simplifications, we follow an order of increasing complexity: 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. Some simplifications can be short circuited 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.

Figure 3 shows an example of simplification scheduling and of lower bound evolution, on a Σ-*VCSP*, whose constraint valuations equal 1, 10, 100, 1000 or $+\infty$ and whose optimal valuation equals 835.



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

Figure 3: An example of simplification scheduling

Experimental Results

Figures 4 and 5 are examples of the results which can be obtained by using such a method on randomly generated binary Σ-*VCSPs*. The first is related to one problem, the second to a set of one hundred problems. In each figure, we show the evolution of the upper bound, by using a classical *Depth First Branch and Bound*, and the evolution of the upper and lower bounds, by using the successive simplifications.

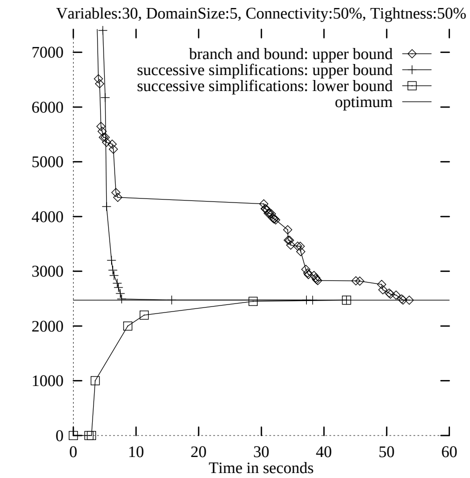


Figure 4: On a particular problem

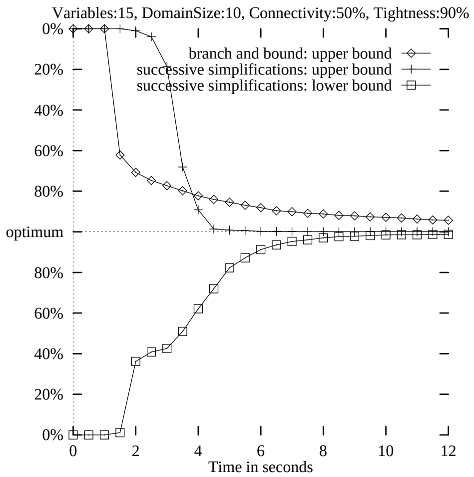


Figure 5: On a set of problems (mean percentages)

Discussion

What we briefly described is just an example of what it is possible to do for producing an anytime bounding of the optimum of a constraint optimization problem. A lot of choices are arguable. It might be interesting to consider other simplifications and schedulings and, above all, to discuss the way a meta-reasoning system could reason about the possible simplifications and their scheduling, according to the nature of the problem to be solved, the current bounds, the other tasks, the capacity of the resources, and the pressure from the external environment.

References

Adelantado, M., and de Givry, S. 1995. Reactive/Anytime Agents: Towards Intelligent Agents with Real-Time Performance. In *IJCAI-95 Workshop on Anytime Algorithms and Deliberation Scheduling*.

Bistarelli, S.; Montanari, U.; and Rossi, F. 1995. Constraint Solving over Semirings. In *Proc. of IJCAI-95*, 624–630.

Boddy, M., and Dean, T. 1994. Deliberation Scheduling for Problem Solving in Time-Constrained Environments. *Artificial Intelligence* 67(2):245–285.

Boddy, M. 1991. *Solving Time-Dependent Problems: A Decision-Theoritic Approach to Planning in Dynamic Environments*. Ph.D. Dissertation, Brown University, Department of Computer Science, Providence, RI.

Dean, T., and Boddy, M. 1988. An Analysis of Time-Dependent Planning. In *Proc. of AAAI-88*, 49–54.

Freuder, E., and Wallace, R. 1992. Partial Constraint Satisfaction. *Artificial Intelligence* 58:21–70.

Freuder, E. 1978. Synthesizing Constraint Expressions. *Communications of the ACM* 21:958–966.

Freuder, E. 1991. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proc. of AAAI-91*, 227–233.

Hansen, E., and Zilberstein, S. 1996. Monitoring the Progress of Anytime Problem-Solving. In *Proc. of AAAI-96*, 1229–1234.

Horvitz, E. 1990. *Computation and Action under Bounded Resources*. Ph.D. Dissertation, Stanford University, Departments of Computer Science and Medecine, Stanford, CA.

Korf, R. 1985. Depth-First Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence* 27:97–109.

Liu, J.; Lin, K.; Shih, W.; Yu, A.; Chung, J.; and Zhao, W. 1991. Algorithms for Scheduling Imprecise Computations. *IEEE Computer* 24(5):58–68.

Mackworth, A. 1992. Constraint Satisfaction. In Shapiro, S., ed., *The Encyclopedia of Artificial Intelligence*. John Wiley & Sons. 285–293.

Musliner, D.; Hendler, J.; Agrawala, A.; Durfee, E.; Strosnider, J.; and Paul, C. 1995. The Challenges of Real-Time AI. *IEEE Computer* 28(1):58–66.

Russel, S., and Wefald, E. 1991. *Do the Right Thing*. MIT Press.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of IJCAI-95*, 631–637.

Strosnider, J., and Paul, C. 1994. A Structured View of Real-Time Problem Solving. *AI Magazine* 15(2):45–66.

Verfaillie, G.; Lemaître, M.; and Schiex, T. 1996. Russian Doll Search for Solving Constraint Optimization Problems. In *Proc. of AAAI-96*, 181–187.

Wallace, R., and Freuder, E. 1995. Anytime Algorithms for Constraint Satisfaction and SAT problems. In *IJCAI-95 Workshop on Anytime Algorithms and Deliberation Scheduling*, 99–103.

Zilberstein, S. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17(3):73–83.