# Algorithmic problems and solutions in the Valued Constraint Satisfaction Problem framework

Gérard Verfaillie and Simon de Givry
ONERA-CERT/DERI, 2 av. Edouard Belin, BP 4025, 31055 Toulouse Cedex 4, France
phone: +33 5 62 25 26 32, fax: +33 5 62 25 25 93, email: {verfaillie,degivry}@cert.fr

**Abstract**

In this paper, we explore the algorithmic problems which are induced by the extension of the *CSP* framework, for dealing with preferences between solutions and with over-constrained problems. We show that the optimization induced problem is incredibly more difficult than the classical satisfaction problem, and that the current algorithmic methods are particularly inefficient. We try to give reasons for such a situation, and to draw up an inventory of possible solutions.

# 1 Valued Constraint Satisfaction Problems

## 1.1 Motivation

Either alone, or included in *Constraint Programming* systems, the *Constraint Satisfaction Problem* framework (*CSP* [Mac92]) is now widely used for representing and solving *Artificial Intelligence* problems, like planning, scheduling, diagnosis, or design ...

But, if this framework allows imperative constraints (like physical constraints) to be represented, it does not allow very important notions, like preferences, priorities, gains, costs, uncertainties, or probabilities ...to be represented and handled. If these notions are expressed as constraints, the problems become over-constrained and have no solution. If they are not expressed, the quality of the resulting solutions may be very bad.

It is the reason why various extensions of the *CSP* framework have been proposed. The *Valued Constraint Satisfaction Problem* framework (*VCSP* [SFV95]) is a very general algebraic framework, which has the double advantage to allow all these notions to be expressed, whatever their particular semantics, and a general algorithmics to be defined.

## 1.2 VCSP framework

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 a consistent assignment (all the imperative constraints are satisfied), whose valuation is minimum.

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 $(E, \succ, \top, \perp, \otimes)$, where $E$ is a valuation set, $\succ$ a total order on $E$, $\top$ and $\perp$ 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 $\perp$ as identity,
- and a valuation function $\varphi$ 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 $\perp$ to express constraint satisfaction and complete consistency. The valuation function $\varphi$ 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 $C_{unsat}(A)$ be the set of the problem constraints unsatisfied by $A$. The valuation $\varphi(A)$ of $A$ is the aggregation of the valuations of all of the constraints in $C_{unsat}(A)$:

$$\varphi(A) = \bigotimes_{c \in C_{unsat}(A)} \varphi(c)$$

Then, the goal is to find an assignment, whose valuation is lower than $\top$ and minimum. The valuation of the problem is the valuation of such an assignment. The global valuation of a partial assignment is the minimum valuation of all its complete extensions (on all the problem variables).

Note that finding an optimal assignment or an optimal consistent relaxation are two equivalent problems. With each assignment, one can indeed associate a consistent relaxation, which involves the set of the satisfied constraints. Thus, any optimal assignment defines an optimal consistent relaxation and, inversely, any optimal consistent relaxation induces at least an optimal assignment.

## 1.3 Subframeworks

Specific subframeworks can be defined by instantiating the valuation structure $S$. Figure 1 shows 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 $\Sigma$-*VCSP* framework, the goal is to find an assignment which minimizes the sum of the valuations of the unsatisfied constraints.

| 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* | $\Sigma$-*VCSP* | $\overline{N}$ | $>$ | $+\infty$ | 0 | $+$ |

Figure 1: Several *VCSP* subframeworks

Defining a richer representation framework is a good thing. But, it may be useless if we do not develop efficient associated reasoning methods. In the sequel of this paper, we consider the methods, which are available for reasoning within the *VCSP* framework, and especially for producing provenly optimal solutions.

# 2 Extending Consistency Enforcing Methods

*Consistency Enforcing* methods [Mac77, Fre78], like *Arc Consistency*, *Path Consistency*, *i-Consistency*, *i-j-Consistency* ... are the key to the efficiency of the classical *CSP* methods. By deleting values or combinations of values, which cannot be involved in any solution, and by using polynomial algorithms for that, they allow, either problems to be simplified, or global inconsistencies to be quickly detected. So, it is natural to try to extend them to *VCSP* framework. But, to do that, it is necessary to distinguish idempotent from non idempotent aggregation operators[2].

## 2.1 With an idempotent operator

First, note that the only idempotent operator in the *VCSP* framework is the *max* operator, which is used in the $\wedge$-*VCSP* and *max-VCSP* frameworks. An extension of *Arc Consistency Enforcing* to *max-VCSPs* has been defined years ago in [RHZ76]. It consists in associating, with each value *val* of a variable $v$, a valuation, which

[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 $\forall a \in E, a \otimes a = a$.

is the maximum on all the variables $v'$, which are connected to $v$, of the minimum valuation of an extension of the assignment $(v, val)$ to $v'$. This valuation is a lower bound of the global valuation of $(v, val)$. The maximum, on all the variables, of its minimum, on all the variable values, is a lower bound of the problem valuation. More recently and generally, in [BMR95], a *Consistency Enforcing* scheme has been defined on *Semiring-based Constraint Satisfaction Problems*[3] (*SCSP*), using an idempotent operator.

### 2.2 With a non idempotent operator

But, with a non idempotent operator, like those which are used in the *lex-VCSP* and $\Sigma$-*VCSP* frameworks, this straightforward extension is not possible: adding an induced constraint to the problem may modify the valuation distribution on the set of the assignments and the resulting problem may not be equivalent to the original one.

It is why limited forms of *Consistency Enforcing*, which can be viewed as *Consistency Enforcing* without propagation, have been defined. For example, *Arc Consistency Counts* [FW92] consist in associating, with each value *val* of a variable $v$, a valuation, which is the aggregation of the valuations of all the constraints, which involve $v$ and cannot be satisfied when using the assignment $(v, val)$. As previously, this valuation is a lower bound of the global valuation of $(v, val)$.

Another way of extending *Consistency Enforcing* would consist in finding an optimal relaxation, whose closure by *Arc Consistency*, or any other level of local consistency, is not empty. Unfortunately, it has been established in [SFV95] that this problem is *NP-hard*, with a strictly monotonic operator, like those which are used in the *lex-VCSP* and $\Sigma$-*VCSP* frameworks.

## 3 Extending Solution Generation Methods

*Backtrack* is the basic algorithm in the *CSP* framework. Its natural extension to the *VCSP* framework, is the *Depth First Branch and Bound* algorithm [FW92, SFV95].

At any moment, the algorithm searches for a complete assignment whose valuation is lower than a current upper bound $ub$. This upper bound is initialized with $\top$ and decreases during the search: each time a complete assignment, whose valuation is lower than $ub$, is found, its valuation is used as a new upper bound. It maintains a lower bound $lb$ on the global valuation of the current partial assignment $A$, and backtracks each time $ub \preceq lb$, since there exists no complete extension of $A$ whose valuation is lower than $ub$. It ends when no complete assignment whose valuation is lower than $ub$ could be found. The lower bound on the global valuation of the current partial assignment $A$ is, by default, the result of *Backward Checking*, *i.e.* the aggregation of the valuations of the constraints, which are assigned and not satisfied by $A$.

As with the *Backtrack* algorithm, there are many ways of improving such an algorithm:

- to choose an adequate, static or dynamic, variable ordering in order to reduce the tree size, to backtrack earlier, or to make backtracking easier; the numerous efficient heuristics, which have been developed in the *CSP* framework [DM89], can be easily extended, by taking into account constraint valuations;
- to provide the algorithm with a low initial upper bound $ub_{init}$; approximate methods (see 6.3) can be used for that;
- to choose an adequate, static or dynamic, value ordering in order to decrease rapidly the upper bound $ub$; but, as within the *CSP* framework, the most efficient heuristics seem to be domain dependent;
- to compute a high lower bound on the global valuation of the current partial assignment, in order to cut early unproductive branches:
  - an extension of the *Forward Checking* mechanism [HE80] can be found in [FW92, SFV95]; it consists in associating with each value *val* of an unassigned variable $v$, the increase in valuation, which would be induced by the extension of the current assignment with $(v, val)$; the aggregation, on all the unassigned variables, of its minimum, on all the variable values, can be added to the result of *Backward Checking*;
  - since *Arc Consistency Enforcing* cannot be easily extended to *VCSPs*, which use a non idempotent operator (see 2.2), algorithms which enforce *Arc Consistency* at each node of the search, like *MAC* [SF94], are limited to $\wedge$-*VCSPs* and *max-VCSPs*;

[3] *SCSP* and *VCSP* are two close algebraic frameworks, which have been both defined for dealing with over-constrained problems and preferences between solutions. Whereas the former uses a totally ordered commutative monoïd, the latter uses a semiring structure.

- however, *Directed Arc Consistency Counts* [Wal94, LM96] and *Russian Doll Search* [VLS96] are two ways of improving *Forward Checking*; both need a static variable ordering; the former is a preprocessing mechanism, which consists in initializing the increase in valuation of each value $val$ of a variable $v$ with an *Arc Consistency Count*, which takes into account only the variables, which are after $v$ in the assignment order; the latter solves successively subproblems, which involve more and more variables, and adds the valuation of the subproblem, which involves all the unassigned variables, to the results of *Backward* and *Forward Checking*;

- to use improved backtracking mechanisms; an extension of the classical *Backmarking* [Gas77] and *Backjumping* [Dec90] mechanisms can be found in [FW92];
- to use learning mechanisms; an extension of such mechanisms [Dec90, SV94] is proposed in [DV96].

# 4 Experimental Results

Figure 2 shows the evolution of the mean complexity (in terms of number of nodes) according to the constraint tightness, when solving randomly generated $\wedge$-*VCSPs* (satisfaction) and $\Sigma$-*VCSPs* (optimization). These problems have been generated according to the classical parameters: number of variables ($n$), domain size ($d$), graph connectivity ($c$), and constraint tightness ($t$). For the $\Sigma$-*VCSPs*, the valuation 1 is associated with each constraint. A *Backtrack* algorithm is used for solving the $\wedge$-*VCSPs*, a *Depth First Branch and Bound* for solving the $\Sigma$-*VCSPs*. Both algorithms use *Forward Checking*. Each point is a mean value on 100 problems.

The first graph shows the classical complexity peak, at the frontier between consistent and inconsistent $\wedge$-*VCSPs*. Solving under and over-constrained problems is easy. It becomes suddenly hard with the intermediate problems.

The second graph shows no complexity peak when solving $\Sigma$-*VCSPs*. Complexity steadily increases with constraint tightness[4]. It shows also how much $\Sigma$-*VCSPs* are harder than $\wedge$-*VCSPs*. Whereas the mean number of nodes goes up to 60000 with the $\Sigma$-*VCSPs*, it does not go beyond 50 with the corresponding $\wedge$-*VCSPs*. Optimization is incredibly more difficult than satisfaction, and the complexity peak of the $\wedge$-*VCSPs* is ridiculous, when compared with the complexity mountain of the $\Sigma$-*VCSPs*.

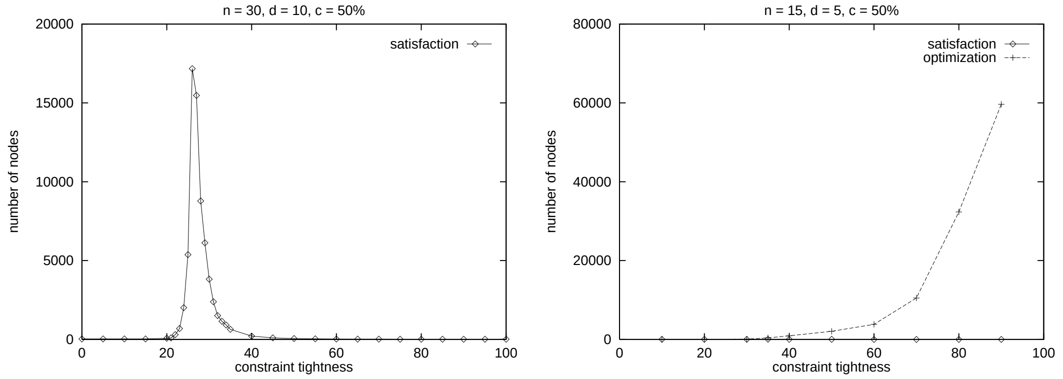


Figure 2: Mean complexity on random $\wedge$-*VCSPs* and $\Sigma$-*VCSPs*

# 5 Why is it so hard ?

The graph in Figure 3 may be a good way of understanding why optimally solving *VCSPs* is so hard. It shows the mean complexity for either finding an assignment whose valuation is lower than or equal to $ub$, or proving there is none. The upper bound $ub$ appears on the $x$-axis, the mean number of nodes on the $y$-axis. Each point is a mean value on 44 $\Sigma$-*VCSPs*, whose optimum equals 20. A complexity peak clearly appears round the optimum and the hardest problem consists in proving that there is no assignment whose valuation is lower than this optimum.

A *Depth First Branch and Bound* algorithm solves a sequence of such problems, with a decreasing value for $ub$. The precise sequence it solves depends on the value of $ub_{init}$, and on the way $ub$ decreases during the search. But, whatever the sequence is, the last problem to be solved consists in proving that there is no assignment

[4]Note however that, with better lower bounds, like those which are provided by *Directed Arc Consistency Counts* or *Russian Doll Search*, complexity decreases with high values of graph connectivity or constraint tightness.

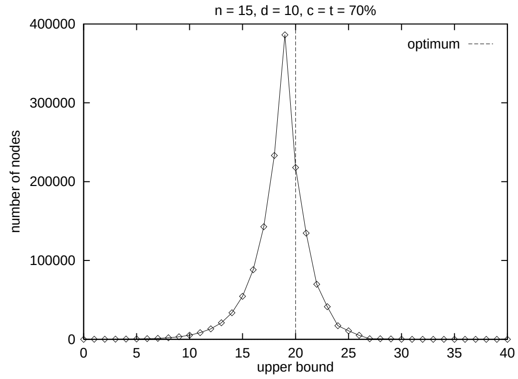

Figure 3: Why is it so hard ?

whose valuation is lower than the optimum. It is the hardest. Optimizing always implies working round the complexity peak.

Moreover, the highest graph connectivity or constraint tightness are, the highest optimum is, and the hardest the last problem of the sequence is to be solved: it is necessary to go deeper and deeper in the search tree, before discovering that it is not possible to get better than the optimum. This explains why complexity steadily increases with graph connectivity or constraint tightness.

# 6 Proposals

## 6.1 Upper and lower bounds

Providing the *Depth First Branch and Bound* algorithm with a low $ub_{init}$ seems to be a good idea. But, if it may exempt from solving some intermediate easy problems, it does not exempt from solving the hardest. Consequently and as it is confirmed by experiments, the resulting gain is not dramatic.

In the *CSP* framework, the combination of *Backtrack* and *Consistency Enforcing* is the main way of efficiently solving problems round the complexity peak. Analogously, computing high lower bounds on the global valuation of the current partial assignment, should be the best way of improving the *Depth First Branch and Bound* algorithm. In spite of first results [Wal94, LM96, VLS96], efficient methods for computing higher lower bounds are still to be developed.

## 6.2 Exploring other methods

Although *Best First Branch and Bound* is widely used for solving close problems, like *Integer Linear Programming* problems, it has been rarely used in the *Constraint Programming* community, because of its potentially huge space requirements. But less space consuming versions, like [Kor93], deserve to be explored, even if, as with *Depth First*, high lower bounds are the key to efficiency.

On the other hand, the graph in Figure 3 shows that *Iterative Deepening* [Kor85] may not be a good idea: solving successive problems with an increasing $ub$ seems harder than with a decreasing $ub$; in both cases, the hardest problem has to be solved; but, with an increasing $ub$, intermediate problems are harder.

Another way to be explored lies on the fact that tree-structured *VCSPs* (problems whose constraint graph is a tree) can be polynomially solved, using a *Dynamic Programming* algorithm [BB72], which consists in going up the tree from the leaves to the root, and in computing, for each value $val$ of a variable $v$, the global valuation of $(v, val)$ in the subproblem, whose root is $v$. It may be consequently profitable to revisite *CSP* methods, like *Cycle Cutset* [Dec90] or *Tree Clustering* [DP89], which both use tree structures: whereas the former exploits a tree forest, which results from the assignment of the cycle-cutset variables, the latter exploits a tree of subproblems.

## 6.3 Relaxing the objective

Another option consists in relaxing the objective. It is what approximate methods, like *Greedy*, *Hill Climbing*, *Tabu Search* [GL93], *Simulated Annealing* [KGV83], or *Genetic Algorithms* [Gol89], do. All of them can be easily adapted to *Constraint Reasoning* and, more particularly, to the *VCSP* framework (see, for example, [MJPL92, Wal96, HD96]). Their opportunistic way of exploring the search space allows them to handle large problems, and generally to produce, within a limited time, good quality solutions, whose valuation is an upper bound of the optimum. On the other hand, they give no information about the distance to the optimum, and may lose a lot of time trying to improve optimal solutions. So, it may be interesting to combine the computation of an upper bound and of a lower bound, in order to have at any time a bounding of the optimum. Some proposals in that direction can be found in [dGV97].

# References


[BB72] U. Bertelé and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, 1972.

[BMR95] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. of IJCAI-95*, pages 624–630, Montréal, Canada, 1995.

[Dec90] R. Dechter. Enhancement Schemes for Constraint Processing : Backjumping, Learning and Cutset Decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[dGV97] S. de Givry and G. Verfaillie. Optimum Anytime Bounding for Constraint Optimization Problems. In *Proc. of the AAAI97 workshop on "Building Resource-Bounded Reasoning Systems"*, Providence, RI, 1997.

[DM89] R. Dechter and I. Meiri. Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems. In *Proc. of IJCAI-89*, pages 271–277, Detroit, MI, 1989.

[DP89] R. Dechter and J. Pearl. Tree Clustering for Constraint Networks. *Artificial Intelligence*, 38(3):353–366, 1989.

[DV96] P. Dago and G. Verfaillie. Nogood Recording for Valued Constraint Satisfaction Problems. In *Proc. of ICTAI-96*, Toulouse, France, 1996.

[Fre78] E. Freuder. Synthesizing Constraint Expressions. *Communications of the ACM*, 21:958–966, 1978.

[FW92] E. Freuder and R. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[Gas77] J. Gaschnig. A General Backtrack Algorithm that eliminates most Redundant Checks. In *Proc. of IJCAI-77*, page 457, Cambridge, MA, 1977.

[GL93] F. Glover and M. Laguna. Tabu Search. In *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–141. Blackwell Scientific Publishing, Oxford, 1993.

[Gol89] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, 1989.

[HD96] J. Hao and R. Dorne. Empirical Studies of Heuristic Local Search for Constraint Solving. In *Proc. of CP-96*, pages 194–208, Cambridge, MA, USA, 1996.

[HE80] R. Haralick and G. Elliot. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[KGV83] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220, 1983.

[Kor85] R. Korf. Depth-First Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27:97–109, 1985.

[Kor93] R. Korf. Linear-space Best-first Search. *Artificial Intelligence*, 62:41–78, 1993.

[LM96] J. Larrosa and P. Meseguer. Expoiting the Use of DAC in MAX-CSP. In *Proc. of CP-96*, pages 308–322, Cambridge, MA, USA, 1996.

[Mac77] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[Mac92] A. Mackworth. Constraint Satisfaction. In S. Shapiro, editor, *The Encyclopedia of Artificial Intelligence*, pages 285–293. John Wiley & Sons, 1992.

[MJPL92] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58:160–205, 1992.

[RHZ76] A. Rosenfeld, R. Hummel, and S. Zucker. Scene Labeling by Relaxation Operations. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(6):173–184, 1976.

[SF94] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI-94*, pages 125–129, Amsterdam, The Netherlands, 1994.

[SFV95] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of IJCAI-95*, pages 631–637, Montréal, Canada, 1995.

[SV94] 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.

[VLS96] G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proc. of AAAI-96*, pages 181–187, Portland, OR, 1996.

[Wal94] R. Wallace. Directed Arc Consistency Preprocessing. In *Proc. of the ECAI-94* Constraint Processing *workshop (LNCS 923)*, pages 121–137. Springer, 1994.

[Wal96] R. Wallace. Analysis of Heuristic Methods for Partial Constraint Satisfaction Problems. In *Proc. of CP-96*, pages 482–496, Cambridge, MA, USA, 1996.