AbsconPseudo 2006

Fred Hemery and Christophe Lecoutre

CRIL-CNRS FRE 2499,
Université d'Artois
Lens, France
{*hemery*, *lecoutre*}@cril.univ-artois.fr

Abstract. This document succinctly presents the solver *AbsconPseudo* submitted to the Pseudo Boolean Evaluation 2006. Knowing that each pseudo-Boolean instance corresponds to a CSP (Constraint Satisfaction Problem) instance or a COP (Constraint Optimization Problem) instance when an optimization function is introduced, we used a Constraint Programming (CP) approach to write this solver. More precisely, our solver integrates a specific filtering algorithm (propagator) to enforce Generalized Arc Consistency (GAC) and a search heuristic combining constraint weighting and coefficients of the objective function. Our solver suffers, on some series of instances, from the absence of cutting plane inferences, not introduced due to lack of time.

1 Introduction

A Constraint Network (CN) P is a pair $(\mathscr{X}, \mathscr{C})$ where $\mathscr{X}$ is a finite set of variables and $\mathscr{C}$ a finite set of constraints. Each variable $X \in \mathscr{X}$ has an associated domain, denoted $dom(X)$, which contains the set of values allowed for X. Each constraint $C \in \mathscr{C}$ involves a subset of variables of $\mathscr{X}$, called the scope of C and denoted by $vars(C)$, and has an associated relation, denoted $rel(C)$, which contains the set of tuples allowed for the variables of its scope. We will respectively denote the number of variables and constraints of a CN by n and e. The arity of a constraint corresponds to the size of its scope. A solution to a CN is an assignment of values to all the variables such that all the constraints are satisfied. A CN is said to be satisfiable iff it admits at least one solution.

The Constraint Satisfaction Problem (CSP) is the task to determine whether or not a given constraint network, also called CSP instance, is satisfiable (i.e. admits a solution). When, in addition, an objective function is considered, we face a Constraint Optimization Problem (COP). The aim is then to find a solution of the underlying constraint network which minimizes (or maximizes) the objective.

A pseudo-Boolean instance corresponds to either a CSP (when no objective function is present) or a COP (when an objective function is present) instance. Each variable of the network is defined on $\{0, 1\}$ and each constraint is an inequality of the form:

$\sum_{i=1}^{r} a_i x_i \geq b$

where r denotes the arity of the constraint (i.e. the number of variables involved by the constraint) and coefficients a_i and b are integers. In case of optimisation, an objective function of the form $\sum_{i=1}^{k} a_i x_i$ must be minimized, where k is the number of variables involved in the objective function.

To solve a pseudo-Boolean instance, we can use a complete CSP/COP solver based on depth-first search with backtracking. This is the approach we followed with our solver *AbsconPseudo*. It has been developed in Java (J2SE 5.0) from the Constraint Programming platform *Abscon* [9].

2 Constraint Propagation

Many works in the area of Constraint Programming have focused on inference, and more precisely, on filtering methods based on properties of constraint networks. The idea is to exploit such properties in order to identify some no-goods where a no-good corresponds to a partial assignment (i.e. a set of variable assignments) that can not lead to any solution. Properties that allow identifying no-goods of size 1 are called domain filtering consistencies [6]. The interest of exploiting domain filtering consistencies is that it is quite easy to deal with no-goods of size 1. Indeed, as such no-goods correspond to inconsistent values, it suffices to remove them from domains of variables.

Generalized Arc consistency (GAC) is one domain filtering consistency which guarantees that each value admits at least a support in each constraint. GAC remains a fundamental property of constraint networks as MGAC, i.e. the algorithm which maintains GAC during search (and called MAC [13] when constraints are binary), is still considered as one of the most efficient complete approach to solve CSP instances.

Definition 1. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN. A pair (X,a), with $X \in \mathscr{X}$ and $a \in dom(X)$, is Generalized Arc Consistent (GAC) iff $\forall C \in \mathscr{C} | X \in vars(C)$, there exists a support of (X, a) in C, i.e. a tuple $t \in rel(C)$ such that $t[X] = a$ and $t[Y] \in dom(Y) \forall Y \in vars(C)$[1]. P is GAC iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and $\forall a \in dom(X)$, (X, a) is GAC.*

To establish GAC on a given CN, one can uses a generic algorithm or a specific one. The interest of a generic algorithm such as GAC3 [10], GAC2001 [3] or GAC3.2 [7] is that it can be used in any situation, i.e. for any constraint. However, the worst-case time complexity of GAC3 and GAC2001/GAC3.2 are $O(er^3d^{r+1})$ [1] and $O(er^2d^r)$ [3], respectively. Here, e denotes the number of constraints, d the size of the greatest domain and r the greatest constraint arity. It means that they can be exploited in practice only when the arity of the constraints is small.

Another way to establish GAC is to exploit the semantics of the constraints. The interest is that it is possible to conceive efficient filtering algorithms (but the drawback is that we have to develop as many algorithms as types of constraints).

[1] $t[X]$ denotes the value assigned to X in t

For a pseudo-Boolean instance, it is possible to conceive a filtering algorithm (that establishes GAC) whose worst-case time complexity is O(er). Indeed, for each constraint, it is possible to deal with each involved variable only once, in order to determine if a value can be removed. It simply requires that variables of the constraint scope are initially ordered according to their respective coefficients and that, at any time, the maximum sum be known (it can be incrementally updated). Either this sum is less than the limit b of the constraint, and then the current network is unsatisfiable, or it is used to make some inferences (value removals). We can always resume the inference process from the last checked involved variable. This is what we included in *AbsconPseudo*.

3 Search Heuristics

The order in which variables are assigned by a backtracking search algorithm such as MGAC has been recognized as a key issue for a long time. Using different variable ordering heuristics to solve a CSP/COP can lead to drastically different results in terms of efficiency. Traditional dynamic variable ordering heuristics benefit from information about the current state of the search such as current domain sizes and current variable degrees. For instance, $dom/ddeg$ [2] involves selecting first the variable with the smallest ratio current domain size to current dynamic degree. One limitation of this approach is that no information about previous states of the search is exploited.

In [4], inspired from [12, 14, 11, 15, 5], it is proposed to record such information by associating a counter with any constraint of the problem. These counters are used as constraint weighting. Whenever a constraint is shown to be unsatisfied (during the constraint propagation process), its weight is incremented by 1. Using these counters, it is possible to define a new variable ordering heuristic, denoted $wdeg$, that gives an evaluation called weighted degree of any variable. The weighted degree of a variable V corresponds to the sum of the weights of the constraints involving V and at least another uninstantiated variable. In order to benefit, at the beginning of the search, from relevant information about current variable degrees, all counters are initially set to 1. Finally, combining weighted degrees and domain sizes yields $dom/wdeg$, an heuristic that selects first the variable with the smallest ratio current domain size to current weighted degree. The experimental results of [4, 8] show that MAC-$wdeg$ and MAC-$dom/wdeg$, i.e., MAC combined with $wdeg$ or $dom/wdeg$ (called conflict-directed heuristics), is a generic backtracking approach which is quite stable to solve static constraint networks.

The search heuristic that we have used in *AbsconPseudo* is defined as follows. Variables involved in the objective function (if present) are selected in priority in decreasing order according to the absolute value of their respective coefficients. For such variables, the value (0 or 1) is selected in order to minimize the objective function. When there are no more unassigned variables involved in the objective function, variables are selected according to $wdeg$.

4 Conclusion

In this paper, we have succinctly presented the solver $AbsconPseudo$, developed in Java, that we have submitted to the Pseudo Boolean Evaluation 2006. On some series of instances, this solver has a good behaviour. For example, for the instance $normalized-fast0507$, $AbsconPseudo$ finds a solution with a cost equal to 235 in less than 10 seconds whereas the best solution found by a solver (within 1200 minutes) submitted to the Pseudo Boolean Evaluation 2005 was only 251. On some other series (e.g. the *chnl* series), $AbsconPseudo$ is subject to thrashing (the fact of always rediscovering the same inconsistencies) as, unfortunately due to lack of time, no cutting planes technique has been introduced.

References

1. C. Bessiere. Constraint propagation. Technical report, LIRMM, Montpellier, 2006.
2. C. Bessière and J. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of CP'96*, pages 61–75, 1996.
3. C. Bessière, J.C. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.
4. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.
5. R. Bruni and A. Sassano. Detecting minimaly unsatisfiable subformulae in unsatisfiable SAT instances by means of adaptative core search. In *Proceedings of SAT'00*, 2000.
6. R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.
7. C. Lecoutre, F. Boussemart, and F. Hemery. Exploiting multidirectionality in coarse-grained arc consistency algorithms. In *Proceedings of CP'03*, pages 480–494, 2003.
8. C. Lecoutre, F. Boussemart, and F. Hemery. Backjump-based techniques versus conflict-directed heuristics. In *Proceedings of ICTAI'04*, pages 549–557, 2004.
9. C. Lecoutre, F. Boussemart, and F. Hemery. Abscon 2005. In *Proceedings of CPAI'05*, volume II, pages 67–72, 2005.
10. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
11. B. Mazure, L. Sais, and E. Gregoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.
12. P. Morris. The breakout method for escaping from local minima. In *Proceedings of AAAI'93*, pages 40–45, 1993.
13. D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of CP'94*, pages 10–20, 1994.
14. B. Selman and H. Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. In *Proceedings of IJCAI'93*, pages 290–295, 1993.
15. J.R. Thornton. *Constraint weighting local search for constraint satisfaction*. PhD thesis, Griffith University, Australia, 2000.