Soft CLP(FD)

Hana Rudová
Faculty of Informatics, Masaryk University
Botanická 68a
Brno 602 00, Czech Republic
hanka@fi.muni.cz

Abstract

Over-constrained problems can be solved with the help of soft constraints. Weighted constraints are a typical representation of soft constraints used to minimize weights of unsatisfied constraints. A natural extension of the CLP(FD) approach is presented which allows handling of weighted soft constraints. To achieve this goal, the costs associated with unsatisfied constraints is accumulated for each problem variable and its value. For the approach proposed, implementation of the soft constraint solver on top of the existing CLP(FD) library of SICStus Prolog is described. A large scale timetabling implementation demonstrates practical application of the approach presented.

Introduction

Various approaches have been proposed to find solutions of problems with uncertainties, ill-defined problems, optimization problems, and over-constrained problems where some kind of softness is needed to obtain a feasible solution. The simplest framework, the maximal constraint satisfaction (MAX-CSP) (Freuder & Wallace 1992), seeks a solution that satisfies as many constraints as possible. Weighted CSP considers weights for each constraint and minimizes the weighted sum of unsatisfied constraints. Fuzzy CSP (Dubois, Fargier, & Prade 1996) combines preference degrees with the help of fuzzy sets and possibility theory. Valued CSP (Schiex, Fargier, & Verfaillie 1995) and Semiring-based CSP (Bistarelli, Montanari, & Rossi 1997) represent meta-frameworks defining monoid and lattice structures, which specify type of used preferences and manipulations with them.

A variety of solving algorithms have also been studied. Extensions of the branch&bound algorithm are often applied for weighted CSP. They can be aimed at improvements to the cost from the local (e.g., partial forward checking (Freuder & Wallace 1992), reversible directed arc consistency (Larrosa & Meseguer 1996; Larossa, Meseguer, & Schiex 1999)) or global point of view (Russian Doll search (Verfaillie, Lemaître, & Schiex 1996)). These approaches for binary constraints have also been extended to non-binary constraints (Meseguer, Larrosa, & Sánchez 2001). Another class of algorithms is introduced by local search methods, e.g., tabu search (Galinier & Hao 1997). Soft constraint propagation has also been studied for general problem instances, representatives are fuzzy CSP (Bistarelli, Gennari, & Rossi 2000) and weighted CSP (Schiex 2000).

A great deal of effort has been devoted to the development of various frameworks for soft constraints. However, there are still few tools available for soft constraint solving. To this author's knowledge, no library can be used together with an existing CLP(FD) solver (Carlsson, Ottosson, & Carlson 1997; IC- 2002). Our aim is to propose a natural extension of the CLP(FD) approach to tackle weighted soft constraints. Based on this proposal, we have implemented a new solver built on top of the existing CLP(FD) library of SICStus Prolog (Carlsson, Ottosson, & Carlson 1997). An advantage of this approach is the ability to include both hard constraints from the CLP(FD) library and soft constraints from a new soft constraint solver. We will also give a brief description of a large scale timetabling problem from Purdue University which was solved with the help of this solver.

This paper will first describe the general method by which the CLP(FD) program can be extended to handle soft constraints. The next two sections define our extension of the domain variables and propose how the soft constraint propagation can be processed. Optimization criteria for soft constraints is specified in the next section, followed by a description of the search methods applied in our approach. The soft constraint solver implemented is then introduced, and the solution of a real-life timetabling problem by our solver is described in the section on Application. Other tools for soft constraints are noted in Related Works. The final section summarizes our work and looks to future extensions of the soft constraint solver proposed.

Soft CLP(FD) Program

We would like to extend the structure of the CLP(FD) program to handle soft constraints. The typical CLP(FD) program consists of the three steps:

1. declare the domains of the variables
2. post the (hard) constraints
3. look for a feasible (or an optimal) solution

We propose a new general solving method for soft constraints which includes, beside optimization, one additional

step.

```
solve( Variables ) :−
    declare-variables( Variables, CostVariables ),
    post-soft-constraints( Variables ),
    post-hard-constraints( Variables ),
    minimize( Variables, sum(CostVariables) ).
```

The traditional CLP optimization code was extended by a declarative statement for the soft constraints which occur in the problem. Soft constraints are defined over the problem variables in a manner similar to the hard constraints. All soft constraints must be stated before any hard constraint is activated though. This ordering is meaningful because the hard constraints may cause a violation of some soft constraint which would not be captured otherwise.

Each problem variable is associated with one cost variable. Cost variables reflect information about the accumulated costs of unsatisfied soft constraints. The proposed cost variable can be understood as the degree of satisfaction of soft constraints related to the given variable.

The overall cost for failure to satisfy soft constraints can be easily defined as the sum of all cost variables. Finally, the classical CLP(FD) optimization can be processed by the minimize procedure. Its aim is to find an assignment of the problem variables such that the criterion *sum(CostVariables)* is minimized.

Let us emphasize that the described structure is very different from a typical approach for solving soft constraints in the CLP(FD) environment. Here a *variable is associated with each "soft constraint"* and its value defines the degree of satisfaction of this constraint. Optimization is processed as a minimization of sum of these variables. In contrast, we associate a (cost) variable with each domain variable. Even more, *we do not bind any additional variable with a soft constraint*. The most important is that the classical approach has a poor propagation to improve optimization process and it is not feasible for most applications. Our aim is to propose a method which would allow inclusion of additional propagation algorithms and improve efficiency of optimization.

Preference Variables

Each problem variable that occurs in any soft constraint is a *preference variable*. The values of the preference variables are associated with a *cost*. The smaller the cost is, the highly preferred preferred the corresponding value is.

The *initial cost* of all values can be set by an unary soft constraint pref used to initialize all preference variables.

> The unary soft constraint pref(PA, [7-5, 8-0, 10-0]) creates the preference variable PA with initial domain containing values 7, 8, and 10 and costs 5, 0, and 0, resp. It means that the value 7 is discouraged wrt. other values.

Zero cost means complete satisfaction of the constraint for the corresponding value in the domain of the preference variable. Any higher cost expresses a degree of violation that would result from the assignment of this value to the variable. All values which are not present in the domain of the preference variable have infinite cost.

Let us note that the preference variable may not be handled as a domain variable only. The domain variable do not have any mechanism for inclusion of the cost for each value.

Soft Constraint Propagation

During computation with soft constraints, the cost associated with a value can be increased. Increasing any cost is called a *soft constraint propagation*. If it is detected that a soft constraint is inconsistent with a value of the preference variable, the soft constraint propagation is processed. The *current cost* of the value indicates the degree to which any soft constraint dependent on this value is currently violated.

> The soft-different(P1, P2, Weight) constraint expresses that the two preference variables P1, P2 should have different values. The constant Weight gives us the cost for violation of this constraint.
>
> If the preference variable P1 is instantiated to some value X, the current cost for the value X of the variable P2 is increased by Weight, i.e., the soft constraint propagation is processed.

There are various algorithms which can be applied to compute the current cost, including algorithms for binary soft constraints as inconsistency counts (Freuder & Wallace 1992), directed arc consistency (Larrosa & Meseguer 1996; Wallace 1995), reversible directed arc consistency (Larossa, Meseguer, & Schiex 1999), or arc inconsistency counts (Affane & Bennaceur 1998). There are also extensions of these algorithms for non-binary constraints described in (Meseguer, Larrosa, & Sánchez 2001).

All of the algorithms mentioned associate each value X of a variable PJ with an inconsistency counter IC_{XJ} (Meseguer, Larrosa, & Sánchez 2001). Definition of this inconsistency counter allows other algorithms to be applied within our soft constraint propagation scheme. The cost of any value for the preference variable is related to the inconsistency counter, i.e., the current cost directly corresponds to the value of the inconsistency counter.

Let us give an example with the algorithm for inconsistency counts (Freuder & Wallace 1992) which is implemented in our solver. We will consider binary soft constraints c_{KJ} over preference variables PK and PJ with a weight W_{KJ}. Let tuples (X,Y) define *invalid* combinations of values of PK and PJ derived from c_{KJ}. Any time some value X is assigned to a variable PK, all inconsistency counters IC_{YJ} (and corresponding current costs) are increased by W_{KJ}. This means that the current cost corresponds to the number of inconsistencies which the value Y of PJ has with the assignments of instantiated variables.

Note the difference with (hard) constraint propagation, which removes values from the domain of the domain variable during the computation of hard constraints. Soft constraint propagation is characterized by increasing costs. However, it may lead to a removal of values during optimization (see the section Labeling and Optimization for details). This may also happen as a consequence of constraint propagation. In this case, it reflects violation of some hard

constraint by this domain value. It is also easy to set the degree of acceptable violation for any preference variable. If the current cost associated with a value in the domain of the preference variable exceeds this limit, it is removed from the domain.

Cost Function

For each preference variable, the soft constraint solver maintains an additional domain variable (called *cost variable*). The sum of all cost variables gives the total cost of the solution (*cost function*). This cost function can then be applied during labeling and optimization.

It is not possible to apply preference variable for optimization directly. The *domain of preference variable is created from values* and not costs. However, the summarized information about the costs related with each preference variable is stored in a cost variable, i.e., the *domain of the cost variable consists of costs*.

Bound consistency is processed for all cost variables. This means that only changes to the lower and upper bounds are maintained. The lower bound corresponds to the best possible satisfaction of soft constraints dependent on the corresponding preference variable, the upper bound is related with their worst possible satisfaction.

A *best current cost* of the preference variable corresponds to the smallest current cost among all costs of its values. Soft constraint propagation may result in an increase of the best current cost. This change is reflected via the lower bound of the cost variable.

The initial upper bound is set to infinity. It can be decreased during optimization. Computing a solution with a defined cost allows the upper bound to be reduced to the actual cost achieved. This is a very weak method however. Generally any method available for computation of the upper bound of the solution cost can be applied to improve constraint propagation for the cost variables.

Labeling and Optimization

Labeling and optimization are processed as is usual for hard constraints and any search method suitable for (sub)optimization can be applied. Preference variables are labeled as the traditional domain variables and optimization criteria are defined as is usual through the domain variables (cost variables).

We can apply the backtracking algorithm for labeling of the preference variables. The current cost of values for the preference variables should be applied to compute value and variable ordering heuristics. Variables can be ordered either by the largest mean of the current cost in their domains or by minimum domain size, value ordering may consider values by increasing current cost (Freuder & Wallace 1992).

The branch and bound algorithm explores the search tree and keeps the cost of the best solution found so far. This cost is an upper bound of the problem solution (UB). At each node, branch and bound estimates the global cost of any leaf node descendant from the current node. This cost, considered as a lower bound of the problem solution (LB), corresponds to the best possible cost of the solution which can be found in the current path. If LB $\geq$ UB holds, the current best solution can not be improved along the explored path and the current branch can be pruned.

Quality of the LB is improved with the help of the best current cost of the preference variable. Let us recall that the best current cost for each preference variable is stored as a lower bound of its cost variable, i.e., the smallest possible sum of the cost variables corresponds to a new LB.

During optimization, the current cost of some value for the preference variable can become higher than the UB. This possibility leads to a removal of this value from the domain of the preference variable.

Let us give an example of a non-classical search algorithm. We have proposed an iterative repair search algorithm based on chronological backtracking — limited assignment number (LAN) search algorithm (Veřmiřovský & Rudová 2002). Each iteration is of linear complexity because the number of attempts to assign a value to each variable is limited. LAN search attempts to find some initial partial assignment of the variables and subsequently repair it such that all, or at least most, of the variables are assigned a value. This is done by developing variable and value heuristics based on the results of previous iterations. In the first iteration of the algorithm, the variable and value ordering heuristics proposed above for backtracking can be applied (called initial heuristics). In each of the following iterations, heuristics based on the previous iteration are used primarily. Any ties are broken by using the initial heuristics. Initial heuristics are aimed at improving the quality of the cost function for soft constraints.

This algorithm can be applied for problems where it is not easy to find a solution in a reasonable time. A complete assignment may not even exist due to conflicts among the hard constraints. While contradictory sets of hard constraints are handled by the LAN search, predefined soft constraints can be included with the help of our approach.

Implementation

The soft constraint solver was implemented as an extension of the CLP(FD) library of SICStus Prolog (Carlsson, Ottosson, & Carlson 1997).

The preference variable is implemented using the attributed variable (Holzbaur 1990). The preference variable remains a domain variable. The attribute of each preference variable stores the cost variable and the current cost for each value present in the domain of the domain variable. The costs are stored in association lists (library assoc at SICStus Prolog). Their implementation is done with the help of AVL trees[1]. Associative lists were chosen as the most suitable library among others available at SICStus Prolog.

The unary soft constraint pref must be called to initialize any preference variable. Initial costs are created based on the "domain value-cost" list and the input domain variable. The keys of the AVL tree are the domain values and their values are the costs. Any time some domain value of

[1] AVL tree is an approximately balanced tree — for every node, the heights of its two subtrees differ by at most one. See (Wirth 1976) for algorithms.

the domain variable is deleted, the AVL tree must be updated accordingly. Any time soft constraint propagation for some domain value is processed, the corresponding cost in the AVL tree is increased. If the best current cost for the preference variable is increased, the lower bound of the cost variable is increased.

The implementation of soft constraint propagation is based on inconsistency counts (Freuder & Wallace 1992). The pref soft constraint for initialization of the preference variables is implemented. Other soft constraints include the soft-different constraint (see example in the section Soft Constraint Propagation) and constraints for non-overlapping of two tasks (soft version of SICStus Prolog serialized constraint). For the labeling of preference variables, backtracking search, branch and bound, and LAN search are implemented.

We can see that the two data structures must be maintained to store domain values (values in domain variable, values together with costs in the AVL tree). A unique data structure would be better for efficiency reasons. Such extension of the SICStus CLP(FD) library would not be as simple as the method described. It would lead to extensive changes in the internal data structures of the CLP(FD) library which was not our aim. In addition, these changes may not be as general as we need. It could lead to a reduced set of available hard constraints. We would like to explore such possibility in the future though.

Application

The soft constraint solver was originally developed to solve the timetabling problem for Purdue University (see `http://www.purdue.edu`). Let us describe this application to show that the current version of the solver is able to solve a large scale real-life problem. A full description of the problem together with the presented results can be found in (Rudová & Murray 2002).

The timetabling problem from Purdue University consists of timetabling (assigning time and classroom to classes) approximately 750 classes attended by 30,000 students into 41 large lecture rooms with capacities up to 474 students. The classes are taught several times a week resulting in 1,600 meetings to be timetabled. The space covered by all meetings fills approximately 85 % of the total available space. Special meeting patterns defined for each class direct possible time and location placement. Classroom allocation must respect instructional requirements and *preferences* of faculty. All instructors may have specific time requirements and *preferences* for each class. A major objective is to *minimize the number of potential student course conflicts*.

There are two types of the preference variables — time and classroom preference variables (about 1500 preference variables together). Both of them have given initial costs corresponding to preferred or discouraged times or classrooms. The binary soft constraints are applied to include the requirements of students (the largest set includes about 23.000 such constraints). For each possible pair of classes, the number of students who want to attend both is defined. An extended version of the soft-disjunctive constraint is implemented to discourage overlaps of multi-meeting classes. The cost of this soft constraint corresponds to the number of students in common.

Independent optimization criteria are defined for the time and classroom variables. The cost function is equal to the sum of cost variables corresponding to time (classroom) variables. It reflects satisfaction of time (classroom) requirements. Each cost variable has its own importance. Since all classes should be relatively equal in importance, the cost variables help to maintain information about the satisfaction of the requirements for each class.

The LAN search algorithm was applied as a search procedure. About ten iterations were needed to compute a complete timetable.

Let us summarize achieved satisfaction of soft constraints given in percentage. The final solution was able to satisfy 97.7 % of the student requirements from course preenrollment, 80.7 % of classes were assigned at the preferred times while 4.5 % classes must be taught at discouraged times. A secondary requirement on selection of preferred classroom was satisfied up to 52.3 %.

Related Work

The clp(FD,*S*) (Georget & Codognet 1998) is an implementation of semiring-based constraint logic programming. It needed a new implementation and abstract machine which results in its good efficiency. However, it inherited neither the broad class of constraint propagation algorithms nor the search methods from any parent CLP(FD) solver. The extensive support of other tools important for the development of real-life applications is also missing (e.g., available from SICStus Prolog).

A different approach was chosen for soft constraint propagation with the help of Constraint Handling Rules (Bistarelli *et al.* 2002). They also relay on the support of the constraint environment for hard constraints — on CHR and on SICStus Prolog programming environment. Comparison of the generality and efficiency of both approaches is an interesting issue for our future work. It seems that both approaches should be complementary as are CLP(FD) and CHR now.

Conclusion

We have proposed a simple extension of the CLP(FD) approach which allows the handling of weighted soft constraints. Each problem variable was associated with an additional domain variable reflecting violations of soft constraints. These variables directly define optimization criterion for soft constraints. The soft constraint propagation described is based on the idea of inconsistency counters (Meseguer, Larrosa, & Sánchez 2001). It can be easily extended by any algorithm cumulating violations of soft constraints with their help. Our approach allows inclusion of both hard constraints from the parent CLP(FD) solver and soft constraints from our solver. Application using backtracking, branch and bound, and a new LAN search algorithms show how it is possible to define search methods with the help of the parent CLP(FD) solver. Based on our proposal, we have implemented a soft constraint solver on top of the CLP(FD) library of SICStus Prolog. The solution of

a large scale timetabling application Purdue University verifies validity and vitality of our proposal and implementation.

In the future, we would like to study more general properties of the algorithm which can be applied within our soft constraint propagation scheme. We also plan to extend the soft constraint propagation used in our problem solution to achieve a more complete propagation for cost variables. Comparison with other solvers could show how to extend our proposal and implementation. A possibility to improve cooperation with the CLP(FD) library will be explored. Extensions of the set of implemented constraints would allow application of the solver for a broader class of problems. Last but not least, the development of the timetabling system continues. The main concentration is devoted to the minimal perturbation problem to allow minimal changes of the generated timetable in case of a redefinition of the original problem.

Acknowledgements

This work is partially supported by the Grant Agency of Czech Republic under the contract 201/01/0942 and by Purdue University. We would like to thank our students Vlastimil Křápek, Aleš Prokopec, and Kamil Veřmiřovský, who are assisting with the solution of the timetabling problem. Our thanks also go to the Supercomputer Center Brno where experiments with the search algorithm are conducted.

References

Affane, M. S., and Bennaceur, H. 1998. A weighted arc consistency technique for MAX-CSP. In Prade, H., ed., *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, 209–213. John Wiley & Sons.

Bistarelli, S.; Frühwirth, T.; Martel, M.; and Rossi, F. 2002. Soft constraint propagation and solving in constraint handling rules. In *Proceedings of the ACM Symposium on Applied Computing*.

Bistarelli, S.; Gennari, R.; and Rossi, F. 2000. Constraint propagation for soft constraint satisfaction problems: Generalization and termination conditions. In *Principles and Practice of Constraint Programming — CP'00*, 83–97. Springer-Verlag LNCS 1894.

Bistarelli, S.; Montanari, U.; and Rossi, F. 1997. Semiring-based constraint solving and optimization. *Journal of ACM* 44(2):201–236.

Carlsson, M.; Ottosson, G.; and Carlson, B. 1997. An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programming*. Springer-Verlag LNCS 1292.

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

Freuder, E. C., and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58:21–70.

Galinier, P., and Hao, J.-K. 1997. Tabu search for maximal constraint satisfaction problems. In Smolka, G., ed., *Proceedings Third International Conference on Principles and Practice of Constraint Programming*, 196–208. Springer-Verlag LNCS 1330.

Georget, Y., and Codognet, P. 1998. Compiling Semiring-based constraints with `clp(FD,`S`)`. In Maher, M., and Puget, J.-F., eds., *Principles and Practice of Constraint Programming — CP98*, 205–219. Springer-Verlag LNCS 1520.

Holzbaur, C. 1990. *Specification of Constraint Based Inference Mechanism through Extended Unification*. Ph.D. Dissertation, University of Vienna.

IC-Park. 2002. *ECL^{i}PS^e Constraint Library Manual, Release 5.5*. `http://www.icparc.ic.ac.uk/eclipse`.

Larossa, J.; Meseguer, P.; and Schiex, T. 1999. Maintaining reversible DAC for MAX-CSP. *Artificial Intelligence* 107(1):149–163.

Larrosa, J., and Meseguer, P. 1996. Exploiting the use of DAC in MAX-CSP. In Freuder, E. C., ed., *Principles and Practice of Constraint Programming – CP96*, volume 1118 of *Lecture Notes in Computer Science*, 308–322. Springer.

Meseguer, P.; Larrosa, J.; and Sánchez, M. 2001. Lower bounds for non-binary constraint optimization problems. In Walsh, T., ed., *Principles and Practice of Constraint Programming – CP 2001*, volume 2239 of *Lecture Notes in Computer Science*, 317–331. Springer.

Rudová, H., and Murray, K. 2002. University course timetabling with soft constraints. In Burke, E., and Causmaecker, P. D., eds., *PATAT 2002 — Proceedings of the 4th international conference on the Practice And Theory of Automated Timetabling*, 73–89.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In Mellish, C. S., ed., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 631–639. San Mateo: Morgan Kaufmann.

Schiex, T. 2000. Arc consistency for soft constraints. In *Principles and Practice of Constraint Programming — CP'00*, 411–424. Springer-Verlag LNCS 1894.

Verfaillie, G.; Lemaître, M.; and Schiex, T. 1996. Russian doll search for solving constraint optimization problems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96) and Eighth Conference on Innovative Applications of Artificial Intelligence (IAAI-96)*, 181–187.

Veřmiřovský, K., and Rudová, H. 2002. Limited assignment number search algorithm. In *Student Research Forum of the 29th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'2002)*. See `http://www.fi.muni.cz/~hanka/publications`.

Wallace, R. J. 1995. Directed Arc Consistency Preprocessing. In Meyer, M., ed., *Constraint Processing, Selected Papers*, volume 923 of *Lecture Notes in Computer Science*. Springer-Verlag. 121–137.

Wirth, N. 1976. *Algorithms + Data Structures = Programs*. Prentice-Hall.