Full Arc Consistency in WCSP and in Constraint Hierarchies with Finite Domains

Student: Josef Zlomek
Supervisor: Roman Barták

Department of Theoretical Computer Science
Faculty of Mathematics and Physics
Charles University
Prague, Czech Republic
zlomek@kti.mff.cuni.cz

Abstract. Consistency techniques proved to be an efficient method for reducing the search space of CSP. Several consistency techniques were proposed for soft constraint frameworks too. In this paper, we propose a *full arc consistency* (FAC) for weighted CSP (WCSP) and algorithms for enforcing FAC and maintaining it during Branch & Bound search. We also define the transformation of constraint hierarchies with finite domains to WCSP.

1 Introduction

Consistency techniques reduce the search space of CSP and thus make larger problems solvable. They reduce the variables' domains by removing values that can't be in any solution. Given a constraint, the value can be pruned if there are no values in the domains of other variables of the constraint such that the constraint is satisfied by these values.

Weighted CSP [2] extends classical CSP by assigning costs to tuples. The costs from different constraints are combined together. Solutions of WCSP are those complete assignments that have the lowest combined cost. Because of the costs, we can use the optimality reasoning in addition to feasibility reasoning. If all the complete assignments that contain a given value assigned to a given variable have a combined cost greater than the minimal cost, we can prune such a value because it can't be in a solution.

Several consistency techniques have been introduced for WCSP. The *star arc consistency* (AC*) [3] enforcing algorithm transforms the WCSP problem by sending costs from binary constraints to unary ones and from unary constraints to nullary constraint $C_\emptyset$. This enables an efficient way to detect values that would cause a too high combined cost. The values are pruned if their cost together with the cost of $C_\emptyset$ is too high.

The costs are never sent from unary constraints to binary constraints by AC* algorithm. The definition of a *full star directional arc consistency* [2] allows the cost sending process to be extended by sending desirable amount of cost from

unary constraints to binary ones, but only in one direction according to the order of variables. This extension makes it possible to send the cost from one variable to another, which causes more costs to be combined together and thus more values to be pruned.

In this paper, we sketch a possible direction for future research in consistency techniques for soft constraints. In particular, we extend the cost sending process from [2] even more. We define a *full star arc consistency* that enables the costs to be sent between variables in any direction. We also show that the WCSP algorithms can be used for constraint hierarchies with finite domains too. We consider the order of variables during propagation too because it might affect the effectivity.

The paper is structured as follows. Section 2 gives preliminary definitions. Section 3 defines full star arc consistency and proposes an enforcing algorithm. Section 4 defines transformation of constraint hierarchies to WCSP so that the proposed algorithm could be used for them too. Section 5 considers the order of variables when solving constraint hierarchies. Finally, Section 6 gives conclusions and directions for future work.

2 Preliminaries

2.1 Frameworks

A *constraint satisfaction problem* (CSP) is a triple $P = (V, D, C)$. V is a set of variables. Each variable $v_i \in V$ has a finite domain $D_i \in D$ of possible values. (i, a) denotes the assignment of value $a \in D_i$ to variable v_i. C is a set of constraints. Constraint c is a relation defined on a subset of variables, denoted as $vars(c)$. A unary constraint is a constraint $C_i \subseteq D_i$, a binary constraint is a constraint $C_{ij} \subseteq D_i \times D_j$. A tuple is an assignment to a set of variables. Tuple t is consistent if all constraints referring only to variables assigned by t are satisfied. A solution of P is a consistent complete assignment. A constraint graph of P is a graph whose vertices are the variables and edges are the (binary) constraints.

Following the paper [2], a *weighted CSP* (WCSP) is a tuple $P = (k, V, D, C)$. $k \in \{1, \ldots, \infty\}$ defines the valuation structure $S(k) = (\{0, \ldots, k\}, \oplus, \geq)$, where $\oplus$ is defined as $a \oplus b = \min(k, a + b)$ and $\geq$ is the standard order on natural numbers. V is a set of variables, D is a set of variables' domains, and C is a set of constraints. Constraints assign costs to assignments to variables. For instance, a binary constraint is a function $C_{ij} : D_i \times D_j \to \{0, \ldots, k\}$. $C_\emptyset$ is a nullary constraint. The cost of a tuple is the combined cost of the constraints. A solution is a tuple with the minimal cost.

A *constraint hierarchy* $H = H_0 \cup \ldots \cup H_l$ is a (finite) multiset of constraints. H_0 denotes the set of required constraints, H_1 is the set of the most preferred constraints etc. The constraints of the i-th level H_i are more preferred than the constraints in H_{i+1}. A comparator *better* is an irreflexive and transitive relation that compares two assignments with respect to the hierarchy. If there is an assignment that satisfies all constraints up to level k, then all better assignments

satisfy all constraints up to level k too. Comparators use error functions to find out how well the assignment satisfies a constraint. The error is a non-negative real number, the lower error the better. Global comparators aggregate errors for each level and compare the aggregated errors while local comparators compare the error for each constraint separately. The solution is such an assignment of variables that satisfies all the required constraints and satisfies the preferential constraints best.

2.2 Some Local Consistencies in WCSP

Definition. *[2] (i, a) is star node consistent (NC*) if $C_\emptyset \oplus C_i(a) < k$. Variable v_i is NC* if all its values are NC* and there exists a value $a \in D_i$ such that $C_i(a) = 0$. P in NC* if every variable is NC*.*

Definition. *[2] (i, a) is arc consistent (AC) with respect to constraint C_{ij} if there is a value $b \in D_j$ such that $C_{ij}(a, b) = 0$. Variable v_i is AC if all its values are AC with respect to every binary constraint affecting v_i. P is AC* if all variables are AC and NC*.*

Definition. *[2] (i, a) is directional arc consistent (DAC) with respect to constraint C_{ij}, $j > i$, if there is a value $b \in D_j$ such that $C_{ij}(a, b) \oplus C_j(b) = 0$. Variable v_i is DAC if all its values are DAC with respect to every constraint C_{ij}, $j > i$. P is DAC* if all variables are DAC and NC*.*

Definition. *[2] P is full star directional arc consistent (FDAC*) if it is DAC* and AC*.*

Definition. *[2] Subtraction $\ominus$ of b from a is defined as*

$$a \ominus b = \begin{cases} a - b & a \neq k \\ k & a = k \end{cases}$$

Definition. *[2] Projection of α cost units from $C_{ij} \in C$ to value (i, a) means subtracting α from $C_{ij}(a, b)$ $\forall b \in D_j$ and adding α to $C_i(a)$.*

Definition. *[2] Extension of β cost units from value (i, a) to $C_{ij} \in C$ means adding β to $C_{ij}(a, b)$ $\forall b \in D_j$ and subtracting β from $C_i(a)$.*

The projection and extension are used by the consistency algorithms [2, 3] to send the costs from one constraint to another.

3 Full Arc Consistency in WCSP

FDAC* is the strongest form of consistency defined above. However, it allows the costs to be propagated only in one direction with respect to the fixed order of variables. This is rather limiting so we propose a form of consistency that allows to propagate in any direction. Because it may not be always possible to send the cost along a directed edge, the proposed extension enables the costs to be sent along other paths and to be concentrated in different variables. Because we might combine more costs, we could prune more values from variables' domains.

Definition. *(i, a) is full arc consistent (FAC) with respect to constraint C_{ij} if there is a value $b \in D_j$ such that $C_{ij}(a, b) \oplus C_j(b) = 0$. Variable x is FAC with respect to variable y if there exists a path $x = x_0, \ldots, x_p = y$ such that $\forall i = 0, \ldots, p-1$ all values of x_i are FAC with respect to constraint $C_{x_i, x_{i+1}}$. Problem P is FAC if there exists a variable x such that x is FAC with respect to every variable $y \neq x$. P is FAC* if it is FAC and NC*.*

Remark. *If (i, a) is FAC with respect to constraint C_{ij} using support b, then (j, b) is AC with respect to constraint C_{ji} using support a.*

Remark. *The FAC relation can be seen as a rooted tree where $\forall v \in V$ parent(v) is FAC with respect to v.*

Now we will describe in more detail how to make the problem FAC* and maintain it FAC* during the Branch & Bound search.

At first, we make the problem FAC* by moving the costs towards one variable and to $C_\emptyset$. We move the costs from one variable to another by extending and projecting certain cost units. While making the problem FAC*, we remember the maximal cost $\forall v_i \in V \ \forall a \in D_i \ m(i, a)$ that value a of variable v_i had. So $m(i, a)$ is equal to $C_\emptyset + C_i(a)$ at some moment during the FAC* enforcing process. Because $m(i, a)$ is the cost moved from other variables, it is the lower bound of the cost of the solution that contains (i, a).

Then, we move the costs from one variable to another to update $m(i, a)$ where necessary. Since $m(i, a)$ is used as a lower bound of the cost of solution that contains (i, a), we do not have to visit all edges (constraints) but we would like to visit all variables. Nevertheless, the lower bounds would be more accurate if we visited all edges.

When we are going to assign a value to variable v_i during Branch & Bound search, we make variable v_i FAC* with respect to all other variables. Thus, the costs move to v_i. The updated maximal costs $m(i, a)$ can be used as a heuristic to choose a value of the variable. If $m(i, a) \geq k$, we can prune the value a because $m(i, a)$ is the lower bound and the current best solution can't be improved by extending the current assignment. When we find a solution, we update k to the cost of the solution.

After we choose an assignment (i, a), we actually delete all other values of variable v_i. This enables us to move costs $C_i(a)$ to $C_\emptyset$ and $C_{ij}(a, b)$ to $C_j(b)$ for every constraint C_{ij} similarly to [3]. These costs are propagated along the rest of the constraint graph and the values $m(i, a)$ are updated during the propagation. The values $m(i, a)$ also limit the propagation – if $m(j, b)$ will not increase when moving cost to (j, b), is it not necessary to move cost to (j, b).

4 Consistency on Constraint Hierarchies

Constraint hierarchies [1] are another popular soft constraint framework, which provides a very simple way for a user to specify preferences. We would like to use the ideas from the previous section for the constraint hierarchies too. We

describe how a constraint hierarchy with unsatisfied-count-better comparator can be transformed to WCSP. Other global comparators may be transformed in a very similar way. After the transformation, we can use the WCSP algorithm.

Definition. *Unsatisfied-count-better comparator [1] is a global comparator, its aggregated error for level H_i is the number of unsatisfied constraints from H_i.*

Constraint hierarchy with the unsatisfied-count-better comparator can be transformed to WCSP as follows.

$$HC_l = 1 \tag{1}$$

$$HC_i = (1 + |H_{i+1}|) \cdot HC_{i+1} \quad \forall i = 0, \ldots, l-1 \tag{2}$$

$$k = HC_0 \tag{3}$$

$$\forall c_j \in H \quad c_j(\boldsymbol{t}) = \begin{cases} 0 & c_j(\boldsymbol{t}) \text{ holds} \\ HC_i & \neg c_j(\boldsymbol{t}) \wedge c_j \in H_i \end{cases} \tag{4}$$

$$C_{\boldsymbol{v}}(\boldsymbol{t}) = \sum_{\substack{c_i \in H \\ vars(c_i) = \boldsymbol{v}}} c_i(\boldsymbol{t}) \tag{5}$$

(1) means that the cost of the weakest constraints is equal to 1. (2) makes the cost of a stronger constraint to be higher than the sum of costs of all weaker constraints. (3) enforces that the solution must satisfy all required constraints. (4) sets the appropriate costs to tuples of a constraint of the constraint hierarchy. (5) combines the costs of constraints that are defined over the same variables.

5 Order of Variables

When enforcing FDAC*, a "good" order of variables is needed so that the consistency could eliminate as many values as possible. The initial step of FAC enforcing algorithm may find more accurate lower bounds too when the "good" order of variables is used.

Definition. *A weighted constraint graph for WCSP is a constraint graph with weights assigned to edges. The weight of the edge is the maximal cost of the corresponding constraint.*

Because the costs are being sent along directed paths, it seems reasonable to order the variables so that the constraint graph would contain many paths directed according to the variables. Path $P = (v_{p_1}, \ldots, v_{p_m})$ is directed according to the variables if $p_i < p_{i+1}$ $\forall i \in \{1, \ldots, m-1\}$.

For example, we could order the variables as follows. At first, we create a weighted constraint graph and find its maximum spanning tree. Then, we choose one leaf of the spanning tree and make all edges directed from the chosen leaf. Finally, we topologically sort the spanning tree while preferring the variables of the stronger constraints if there are more options. The desired order of the variables is the topological order.

Example. *Figure 1.a shows a WCSP with variables x, y and z, the domain of every variable is $\{1, 2, 3\}$. The WCSP was transformed from a constraint hierarchy with a strong constraint $C_{xy}(a, b) = true$ iff $a < b$ and a weak constraint $C_{yz}(a, b) = true$ iff $a < b$. Hence, the cost of unsatisfaction of the strong constraint is 2 and the cost of unsatisfaction of the weak constraint is 1. Unary costs are illustrated inside the domain value. The edges illustrate the binary costs – thin edges represent a cost of 1 while strong edges represent a cost of 2.*

Figure 1 demonstrates how the costs are being moved from one constraint to another when the order of variables is (x, y, z). Problem 1.a is the original situation. Figure 1.b is the equivalent problem after projecting 2 cost units from C_{xy} to $C_x(3)$ and 1 cost unit from C_{yz} to $C_y(3)$. Figure 1.c shows the state after extending 1 cost unit from $C_y(3)$ to C_{xy} and projecting 1 cost unit from C_{xy} to $C_x(2)$. The problem 1.c is FAC with respect to variable z.*

If the order of variables was (x, z, y), only projections from Cxy to $C_x(3)$ and from C_{zy} to $C_z(1)$ could be performed. $C_x(2)$ would be 0, which is less accurate lower bound than the lower bound for the variable order (x, y, z).

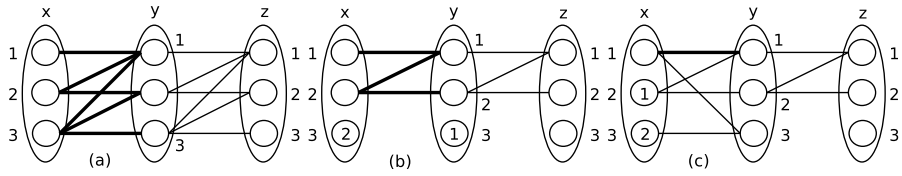

Fig. 1. Example of propagation

6 Conclusions

This paper suggests possible directions for research in soft constraints propagation. We have strengthen the cost propagation by allowing to send costs in any direction which should cause propagation of more costs and stronger domain pruning. In future, we would like to define the algorithm formally and compare the results. We would also like to use the ideas from WCSP to solve constraint hierarchies directly. We believe that it is promising to order the variables. We would like to study possible orderings more deeply.

References

1. A. Borning, M. Maher, A. Martindale, M. Wilson. *Constraint Hierarchies and Logic Programming.* In Proceedings of 6th International Conference on Logic Programming, pages 149–164, MIT Press, 1989.
2. J. Larrosa, T. Schiex. *In the quest of the best form of local consistency for Weighted CSP.* In Proceedings of IJCAI–03, Acapulco, Mexico, 2003.
3. J. Larrosa, T. Schiex. *Solving Weighted CSP by Maintaining Arc Consistency.* Artificial Intelligence, Volume 159 (1–2), 1–26, November 2004.