Benefits and Problems of Using Cycle-Cutset Within Iterative Improvement Algorithms

HARALD MEYER AUF'M HOFE
GWI-SIEDA GmbH
Richard-Wagner-Str. 91, 67655 Kaiserslautern
Email: hmeyer@sieda.com

March 2000

Abstract

Some experiments on randomly generated partial constraint satisfaction problems as well as an example from the domain of real world nurse rostering illustrate the advantage of the cycle-cutset method as a repair step in iterative search. These results motivate the integration of adopted algorithms on solving tree-structured constraint problems and the cycle-cutset method into modern constraint-based optimization with branch-and-bound and propagation of global constraints.

1 Motivation

The GWI Group is the number two provider of integrated software for hospitals in Germany. Process management in hospitals requires the generation of several schedules for instance on nurse rostering, transport of patients, and coordination of diagnosis processes. The availability of new therapies in combination with increasing cost pressure motivates further optimizations of these schedules by intelligent optimization systems. The GWI-SIEDA GmbH pioneered this approach by the development of a constraint-based nurse rostering system that is used in a growing number of hospitals [Meyer auf'm Hofe, 1997, Meyer auf'm Hofe, 2000, Meyer auf'm Hofe, 1999].

All these applications require the use of generic algorithms that work on declarative and easily maintainable problem representations since the exact scheduling problem differs typically from hospital to hospital. Thus, the above cited system uses very general methods: Extended *partial constraint satisfaction problems (PCSP)* [Freuder and Wallace, 1992] are used to represent the rostering problem. An iterative search where a branch-and-bound algorithm extended by constraint propagation conducts improvement steps is used to produce rosters. Soft constraints in combination with iterative search allows on-line modification of the problem specification. Additionally, iterative search converges quickly on rosters of sufficient quality.

Refer to Fig. 1 for a small and simple example. This figure presents a simplified roster. Each cell is labeled by a shift that is either a early-morning shift (MS), a late shift (LS), a night shift (NS), a longer day-turn (DT), or an idle shift (-). The corresponding constraint problem contains a constraint variable for each cell in the roster. The available types of shifts form the domain of these variables. Constraints concern either shift assignments within the same row of the roster, e.g. due to working

	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
nurse #1	NS	-	-	NS	NS	NS	NS	NS	NS	-	-	MS	MS	LS
nurse #2	MS	MS	MS	-	DT	DT	-	-	DT	NS	-	-	MS	MS
nurse #3	MS	MS	-	-	MS	MS	MS	MS	MS	-	NS	NS	NS	NS
nurse #4	-	NS	NS	-	-	MS	MS	MS	LS	LS	-	DT	-	DT
nurse #5	DT	DT	-	MS	MS	LS	LS	LS	-	-	MS	MS	LS	MS
nurse #6	LS	LS	LS	-	LS	-	DT	DT	MS	MS	-	-	DT	DT
nurse #7	LS	-	-	LS	LS	LS	LS	LS	-	-	LS	LS	LS	LS

Figure 1: *A roster and a subproblem representing a repair step.*

Algorithm 1 ITERATIVEIMPROVEMENT($\mathcal{P} = \langle X, D, C, \mathrm{var}, \varphi, \succ \rangle$)

1: Compute an initial assignment A to all variables in X.
2: **loop**
3: Set $X' \cup X_T \subseteq X$ to hold a region where A possibly is suboptimal and X_T forms a tree-structured subproblem.
4: **if** $X' = \emptyset$ **then break**, **end if**
5: Generate an entry $conflicts[x \leftarrow d]$ for each $x \in X' \cup X_T$ and $d \in D$ that holds all conflicts with $A{\downarrow}\, X \setminus (X' \cup X_T)$.
6: If BRANCHANDBOUND($\mathcal{P}, \beta\ \mathcal{V}_P(A{\downarrow}\, (X \setminus (X' \cup X_T)),\ A{\downarrow}\, (X \setminus (X' \cup X_T),\ X',\ X_T,\ conflicts$) leads to an improved solution then assign this improvement to A.
7: **end loop**
8: **return** A

time restrictions and compatibility of consecutive shifts. Or constraints refer to all shifts within a column of the roster to ensure a minimal crew at the ward and to state preferences on larger standard crews [Meyer auf'm Hofe, 1997].

The nurse rostering system uses an iterative search according to Algorithm 1. An initial assignment to all variables is generated in line 1. The loop from line 2 to 7 performs improvement steps. These steps start with a heuristic detection of a part $X' \cup X_T$ of the current roster A that is considered to be responsible for some deficiencies of A (line 3). Fig. 1 presents a subproblem by grey shaded cells that can result from a heuristic to split the chain of night shifts of nurse No. 1. This chain is too long. Line 5 prepares an exhaustive search for better assignments to the variables in $X' \cup X_T$ of the chosen subproblem. A call to a branch-and-bound procedure completes the improvement step in line 6.

As mentioned above, Fig. 1 presents an example for a subproblem $X' \cup X_T$ that may be chosen in order to repair a highly relevant deficiency of a schedule. As mentioned above, all constraints are either oriented along the rows or the columns in the roster. As it will be shown later, the constraint graph of this subproblem is a hyper tree. This observation motivates a further investigation of efficient procedures for solving tree-structured constraint problems [Freuder, 1982, Freuder, 1990, Dechter *et al.*, 1990, Freuder and Wallace, 1992]. As a consequence, this paper addresses opportunities to extend branch-and-bound search by the cycle-cutset method [Dechter and Pearl, 1987, Dechter, 1990]. This improved branch-and-bound is then used to perform the the repair step in Algorithm 1. For this reason, line 3 of Algorithm 1 returns two sets of variables to indicate a subproblem: X' and X_T. The variables in X_T are supposed to form a *tree-structured subproblem* that can be solved by efficient algorithms. In contrast, X' is thought to be a cutset: A set of variables that need to have a certain value assigned in order to enable the use of efficient algorithms for tree-structured problems.

This paper suggests to apply efficient algorithms for special constraint problems — for instance of a tree-like structure — as a repair step in iterative search to solve real world problems like nurse rostering. Therefore, the next section briefly illustrates partial constraint satisfaction and tree-structured

constraint problems accompanied with the relevance of these definitions for nurse rostering. Starting with reformulations of standard algorithms, the next two sections develop algorithms for solving k-tree structured subproblems in real world applications. Concluding remarks sum up the results.

2 Nurse Rostering as Partial Constraint Satisfaction

Partial Constraint Satisfaction with Fuzzy Constraints

As it is well known, constraint problems consist of constraint variables and their domains which is a set of labels that can be assigned to the variable. Constraints post restrictions on consistent assignments of labels to two or more variables. The standard constraint problem is to assign labels to all variables in such a way that all constraints are satisfied. Things get a bit more complex on partial constraint satisfaction where solutions to a constraint problem are only required to satisfy the constraints as good as possible. Such forms of constraint problems are appropriate to represent all kinds of optimization problems. This section describes a notion of partial constraint satisfaction with fuzzy constraints [Meyer auf'm Hofe, 2000] that uses a partial ordering of fuzzy sets of constraints to distinguish more from less important conflicts. The standard PCSP [Freuder and Wallace, 1992] considers soft but crisp constraints, i.e. solutions are not necessarily required to satisfy all constraints but all constraints are either completely satisfied or completely violated. In contrast, fuzzy constraints may be *partially* satisfied by a solution.

Let X be the set of variables, D the domain of the variables, and C be the set of constraints that describe constraint problem $\mathcal{P}$. Each constraint $c \in C$ has a set of local variables var_c.

An assignment to the variables in set X' is a set of labelings $\{x \leftarrow d \mid x \in X', d \in D\}$. $A{\downarrow}\, X''$ selects from A the assignments to the variables in X''. Additionally, $A{\downarrow}\, x$ denotes the value that A assigns to variable x.

A function $\varphi_c : D^{\mathrm{var_c}} \rightarrow [0;1]$ maps a degree of constraint violation to each assignment A to the local variables var_c. A 0 degree of constraint violation means that A satisfies the constraint perfectly. A 1 degree of constraint violation indicates a complete violation. Degrees between these extremes represent a partial violation.

Since assignments may violate constraints to a degree from 0 to 1, the conflicts of each assignment A can be represented by a fuzzy set $\bar{C}(A)$ where $\varphi_c(A{\downarrow}\, \mathrm{var}_c)$ is the membership of constraint c in this set. This view makes it easy to sum up conflicts by use of fuzzy set union $C_1 = C_2 \sqcup C_3$ where the membership of constraint c to C_1 is the maximum of c's membership to C_2 and C_3.

Finally, a partial ordering $\succ$ among fuzzy sets of constraints describes which conflicts are more important than others. A fuzzy set of constraints C_1 is more important than another fuzzy set of constraints C_2 iff $C_1 \succ C_2$. An assignment A is a solution of constraint problem $\mathcal{P}$ iff an assignment A' with less important conflicts — i.e. $\bar{C}(A) \succ \bar{C}(A')$ — does not exist.

PCSPs according to this definition provide a very flexible formalism to represent problems like nurse rostering. Nurse rostering concerns classical binary constraints — for instance to represent compatibility of consecutive shifts — as well as global constraints which affect a large number of local variables. As an example for such constraints in nurse rostering, consider the APPROX constraint that is for instance used to prefer attendance of a standard crew at the ward and a balanced working time account. The degree of violating this constraint is defined with respect to two parameters f and g,

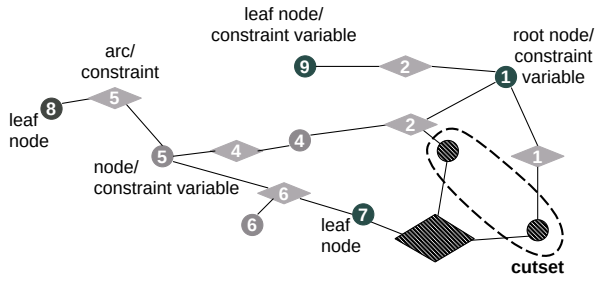

Figure 2: *Components of a tree-structured constraint network.*

where $f(d)$ maps a weight to each value in the domain D and g is a goal sum of this weights. The degree $\varphi_{f,g}(A)$ of violation a constraint c of this type grows with the distance between the goal sum g and the sum of weights of the values that A assigns to the local variables.

$$\varphi_{f,g}(A) = \frac{|g - \sum_{\{x \leftarrow d\} \in A} f(d)|}{|\mathrm{var}_c| \cdot \max\{f(d) \mid d \in D\}}$$

Let for instance f map a 1 to the morning shift and a 0 to all other shifts. Then an APPROX constraint is appropriate to state a preference on a standard crew of 2 nurses during the morning shift on day 2 in Fig. 1. The constraint has a goal sum of 2 and all constraint variables concerning shifts on day 2 as local variables. Consequence: The degree of constraint violation grows with the difference between the scheduled crew size and the preferred crew size during the time period of the morning shift.

Tree-structured Subproblems

Fig. 2 presents the constraint graph of a constraint problem. The nodes in this graph represent variables. The hyper-arcs represent constraints between the connected variables. The tree-structured subproblem consists of the numbered nodes and constraints. The striped constraint disturbs the tree-structure of the problem and the striped nodes represent a so-called cutset: If these variables are labeled with unique values then the striped constraint degenerates to a unary constraint. As a consequence, the rest of the constraint problem has a tree-structure [Dechter and Pearl, 1987, Dechter, 1990].

A constraint graph is *plain* iff two nodes are connected by at most one hyper-arc.

In Fig. 2, the numbers of nodes represent a total ordering $>$ of the nodes. The *width of a node* x_1 *of the constraint graph with respect to a particular ordering* $>$ is defined to be the number of nodes x_2 with $x_1 > x_2$ where x_1 and x_2 are connected by different arcs (or constraints). The *width of the constraint graph with respect to a particular ordering* $>$ is the largest width of a node. $>$ is called to be the *best ordering in a particular constraint graph* iff the width of the constraint graph with respect to $>$ is minimal. The *width* of a constraint graph (in general) is the width with respect to the best ordering [Freuder, 1982].

A constraint problem whose constraint graph is of width 1 is called a tree-structured problem. The smallest node according to $>$ is the root node. All nodes which are not connected to $>$-larger nodes

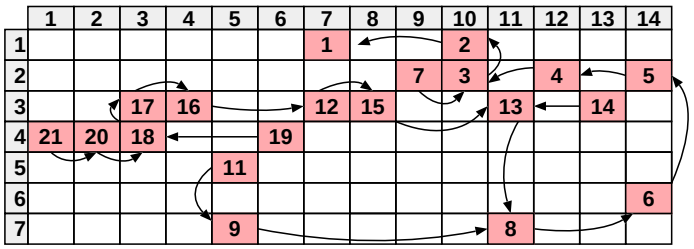

Figure 3: *Constraints connect only variables within the same row or the same column. Consequence: The subproblem's constraint graph has the form of a hyper-tree.*

are leaf nodes. An ordering $>$ proving width 1 is also called tree-ordering if for each constraint the local variable which is the smallest due to $>$ is connected with a smaller variable.

Fig. 2 presents such a tree-structured problem. The numbers of the nodes indicate a tree-ordering. Each node is at most connected with one node of a smaller index.

Consequently, a tree-ordering $>$ can be used also in the nurse rostering application to prove that a subproblem has a tree structure. Since each cell in a roster corresponds with a constraint variable in the constraint problem, Fig. 3 presents a tree-ordering for the subproblem of Fig. 1 numbering the nodes in the constraint graph from 1 (root node) to 21. The constraints on a roster either connect cells in the same row of a roster — maximal and minimal working time, preferred working time, constraints on minimal resting time, and so on — or the same column of a roster — for each required shift type a constraint on minimal and preferred crew size. Hence, all variables referring to cells of the same row or column are connected by constraints. The arrows in Fig. 3 point from each constraint variable x_1 to a variable x_2 iff both are local to the same constraint and $x_1 > x_2$. Since at most one arrow leaves from the same variable, the subproblem has width 1. This small example shows that more efficient algorithms for solving tree-structured subproblems are also relevant to improve nurse rostering if they are not restricted to subproblems of a plain constraint graph — in the nurse rostering application many constraints overlap in more than one variable.

3 Cycle-Cutset and Branch-and-Bound

This section introduces an extension of the branch-and-bound with cycle-cutset that is derived directly from the literature. Algorithm 2 presents a version of the branch-and-bound as it is referred by Algorithm 1. The branch-and-bound receives as arguments: The constraint problem $\mathcal{P}$, the bound β, the best yet found solution S, the currently explored assignment A, and the distance δ, that is the fuzzy set of constraints reflecting A's constraint violations. Furthermore, X' and X_T specify which part of $\mathcal{P}$ has to be searched. The variables in X_T are known to form a tree-structured subproblem. The branch-and-bound has to branch over the possible assignments to the variables in X' whereas the variables in X_T will be labeled by a special algorithm[1].

Line 1 leads to a backtracking if the conflicts δ of the currently explored assignment A exceed bound β. Then, line 2 determines whether the branch-and-bound has to perform a branching or not. If both X' and X_T are empty, then A is a new solution. Hence, line 4 returns this new solution

[1]The literature on cycle-cutset assumes an algorithm where X_T needs to be computed for any branch in the search tree.

Algorithm 2 BRANCHANDBOUND($\mathcal{P}, \beta, S, \delta, A, X', X_T, conflicts$)

$\mathcal{P}$ is the constraint problem. β is the bound: The fuzzy set of constraints representing the conflicts of the best yet found solution S. Initially, the bound maps full membership to all constraints and S is the empty set. A is the partial assignment describing the current branch of the search tree that is initially the empty set. δ is the fuzzy set of constraints that represents known conflicts concluding from assignment A. X' is the set of variables to be labeled. $X_T \cap X' = \emptyset$ is a tree-structured subproblem. $conflicts$ describes the known conflicts. The result of the algorithm is a tuple $\langle S, \beta \rangle$ representing either the old solution or a new improved one.

1: **if** not $\beta \succ \delta$ **then return** $\langle S, \beta \rangle$, **end if**.
2: **if** $X' = \emptyset$ **then**
3: **if** $X_T = \emptyset$ **then**
4: **return** $\langle A, \delta \rangle$.
5: **else**
6: **return** SOLVETREE($\mathcal{P}, \beta, \delta, A, X_T, conflicts$) .
7: **end if**
8: **else**
9: Choose an $x \in X'$.
10: **for all** $d \in D$ with $\beta \succ conflicts[x \leftarrow d]$ **do**
11: **for all** $c \in C$ with $x \in \mathrm{var}_c$ **do**
12: $conflicts \leftarrow$PROPAGATION($c, \beta\, \delta \sqcup conflicts[x \leftarrow d], A \cup \{x \leftarrow d\}, conflicts$), where Propagation is one of the procedures for constraint propagation described below.
13: **end for**
$\langle S, \beta \rangle \leftarrow$ BRANCHANDBOUND($\mathcal{P}, \beta, \delta \sqcup conflicts[x \leftarrow d],$
14: $A \cup \{x \leftarrow d\}, X' \setminus \{x\}, X_T,$
$conflicts$).
15: **end for**.
16: **return** $\langle S, \beta \rangle$.
17: **end if**

and its conflicts. If X_T represents a non-trivial tree-structured subproblem, line 6 calls Algorithm 4 SOLVETREE to solve the remaining problem. If X' is not empty, line 3 chooses a variable and the loop in line 10 loops over all admissible assignments to this variable. Then, line 12 calls constraint propagation to find out which conflicts arise on the new assignment $x \leftarrow d$. Line 14 implements the final step in branching: A recursive call of the branch-and-bound including the new assignment and a new distance.

This version of the branch-and-bound uses constraint propagation to detect conflicts which conclude from the currently explored assignments. Array $conflicts$ stores for each assignment to a yet unlabeled variable the corresponding detected conflicts. Algorithm 3 presents a generic procedure for the propagation of fuzzy constraints: The min-max-propagation [Snow and Freuder, 1990, Dubois *et al.*, 1993]. This procedure loops over all admissible assignments to the local variables of the constraint (line 4). For each of these "tuples", fuzzy set $tupleConflicts$ holds the propagated constraint with a membership according to the tuple's degree of violating the constraint. Additionally, $tupleConflicts$ is loaded with the conflicts that have been detected in advance to each of the assigned values (line 7). After this procedure, $tupleConflicts$ holds for each constraint the maximal degree of constraint violation that follows either from the propagated constraint or that has been detected in advance for one of the assigned values. Then, line 10 collects for each labeling of a single variable the conflicts according to the best tuple A' it appears in. Finally, $conflicts$ is loaded with the newly detected conflicts. Provided that φ_c can be computed efficiently, the effort for the min-max-propagation grows with $|D|^{|\mathrm{var}_c|}$ which is the number of tuples A'.

Algorithm 4 TREESOLVE finds optimal solutions to tree-structured problems with $|C|$ calls of al-

Algorithm 3 MINMAXPROPAGATION($c, \beta, \delta, A, conflicts$)

c is the constraint to be propagated. Argument A' is a partial assignment that may be used for instance to describe the branch of a search tree. $conflicts[x \leftarrow d]$ is a fuzzy set of constraints that holds the conflicts that follow from assigning label d to variable x.

1: **for all** $x \in \mathrm{var}_c$ and $d \in D$ **do**
2: Add all constraints with membership 1 to $bestConflicts[x \leftarrow d]$.
3: **end for**.
4: **for all** $A' \in D^{\mathrm{var}_c}$ composed of admissible labelings **do**
5: Set $tupleConflicts$ to be the fuzzy set mapping membership $\varphi_c(A')$ to constraint c.
6: **for all** $x \in \mathrm{var}_c$ **do**
7: $tupleConflicts \leftarrow tupleConflicts \sqcup conflicts[x \leftarrow A'\downarrow x]$.
8: **end for**.
9: **for all** $x \in \mathrm{var}_c$ **do**
10: **if** $bestConflicts[A'\downarrow \{x\}] \succ tupleConflicts$ **then**
11: $bestConflicts[A'\downarrow \{x\}] \leftarrow tupleConflicts$.
12: **end if**.
13: **end for**.
14: **end for**.
15: **for all** $x \in \mathrm{var}_c$ and $d \in D$ **do** $conflicts[x \leftarrow d] \leftarrow bestConflicts[x \leftarrow d]$ **end for**.

A labeling $x \leftarrow d$ is admissible iff either $(x \leftarrow d) \in A$ or A does not assign a value to variable x and $\beta \succ \delta \sqcup conflicts[x \leftarrow d]$.

gorithms for constraint propagation. For now assume, that PROPAGATE is synonymous with the max-min-propagation according to Algorithm 3. The basic idea is to propagate the conflicts from the leave nodes to the root node. Consider again Fig. 2 as an example. The loop over line 2 considers at first leaf node 7. Algorithm 5 DIRECTEDPROP is called to propagate the constraints linking node 7 as a root node of a subtree to the branches of the subtree. However, node 7 is a leaf node and, consequently, DIRECTEDPROP has nothing to do. This situation changes on node 5 representing variable x_5. After DIRECTEDPROP on this variable, the $conflicts$ of the labels of variable x_5 are set according to the best opportunity to label the variables x_5 to x_8 forming the subtree below x_5. It can be shown that $conflicts[x_5 \leftarrow d]$ comprises under these circumstances exactly the conflicts of the optimal assignment to the variables in the subtree that also assigns d to x_5. The condition for constraint propagation in DIRECTEDPROP guarantees that constraints of deeper subtrees will be considered before constraints of higher subtrees.

After leaving the propagation phase, the loop starting at line 5 collects labelings to build an optimal solution in the same manner as the branch-and-bound algorithm: Select a new labeling for a variable according to the yet known $conflicts$ and propagate the consequences of this new labeling by the same constraint propagation as it is used in the branch-and-bound to find new conflicts.

Hence, tree-structured problems can be solved efficiently, if the constraint graph is plain and propagation of the constraints is efficient. Academic papers often concentrate on binary constraints since these constraints guarantee an efficient max-min-propagation.

Fig. 4 shows the effect of introducing cycle-cutset into the branch-and-bound (bb+cycle-cutset) and of using this algorithm for steps of repair (iterative cycle-cutset) compared to the branch-and-bound with constraint propagation but without using TREESOLVE (bb). The diagrams show an average performance on randomly generated PCSPs of a plain constraint graph and with binary and crisp constraints of low satisfiability. The constraints have a randomly chosen weight and the preference $\succ$ is defined according to the weight sum of violated constraints [Freuder and Wallace, 1992]. The curves

Algorithm 4 TREESOLVE($\mathcal{P}, \beta, \delta, A, X_T, conflicts$)

Let $>$ be a tree-ordering on the variables of the subproblem. Let $C_T \leftarrow \{c \mid |\mathrm{var}_c \cap X_T| \geq 2\}$ be the constraints connecting variables of the tree-structured subproblem, i.e. the set of constraints oft he tree-structured subproblem.

1: **for all** $x \in X_T$ ordered by $>$ starting with the largest variable **do**
2: DIRECTEDPROP($x, C_T, \beta, \delta, A, conflicts$).
3: **end for**.
4: **if** not $\beta \succ$ **then return** $\langle \emptyset, \beta \rangle$, **end if**
5: **for all** $x \in X$ ordered by $>$ starting with the smallest variable (root node) **do**
6: Choose a value $d \in D$ with minimal $\delta \sqcup conflicts[x \leftarrow d]$.
7: **for all** $c \in C$ with $x \in \mathrm{var}_c$ **do** PROPAGATION($c, \beta, \delta \sqcup conflicts[x \leftarrow d], A, conflicts$) **end for**.
8: $\delta \leftarrow \delta \sqcup conflicts[x \leftarrow d]$.
9: **end for**.
10: **return** $\langle A, \delta \rangle$.

Algorithm 5 DIRECTEDPROP($x', C_T, \beta, \delta, A, conflicts$)

1: **for all** $c \in C_T$ where x' is minimal local variable of c due to $>$ **do**
2: MINMAXPROPAGATION($c, \beta, \delta, A, conflicts$).
3: **end for**.
4: **return** $conflicts$.

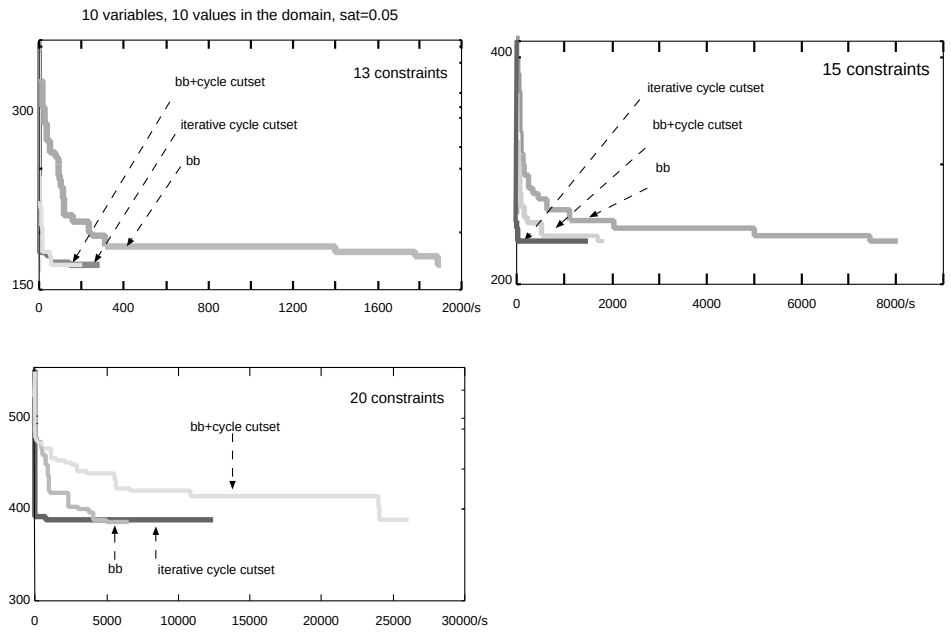

Figure 4: The basic effect of cycle-cutset as step of repair within iterative improvement.

show the decrease in the weight of constraints violated by the best yet found solution over time. Diagram a) shows the performance on nearly tree-structured problems: All plain but connected graphs with 10 nodes and 9 arcs form a tree. As a consequence, both algorithms using cycle cutset converge very quickly on the optimal solution. With increasing number of the constraints, the branch-and-bound with cycle-cutset becomes worse than branch-and-bound without cycle-cutset since the solved problems loose their tree-like structure. In contrast, an iterative search exploiting cycle-cutset con-

Algorithm 6 SHALLOWPROPAPPROX($c, \beta, \delta, A, \mathit{conflicts}$)

As mentioned in the text, this constraint has a goal sum g and a function f as a special attribute that maps each value in D to a real number. Let A be an assignment to the local variables. Then, A's degree of violating the constraint is defined as follows:

$$\varphi_{f,g}(A) = \frac{|g - \sum_{\{x \leftarrow d\} \in A} f(d)|}{|\mathrm{var}_c| \cdot \max\{f(d) \mid d \in D\}}$$

1: $f_\Sigma^{\max} \leftarrow 0$. $f_\Sigma^{\min} \leftarrow 0$.
2: **for all** $x \in \mathrm{var}_c$ **do**
3: $\quad f_\Sigma^{\max} \leftarrow f_\Sigma^{\max} + \max\{f(d) \mid x \leftarrow d \text{ is admissible}\}$.
4: $\quad f_\Sigma^{\min} \leftarrow f_\Sigma^{\min} + \min\{f(d) \mid x \leftarrow d \text{ is admissible}\}$.
5: **end for**.
6: **for all** $x \in \mathrm{var}_c$ and $x \leftarrow d$ is admissibe **do**
7: $\quad f_{x \leftarrow d}^{\max} \leftarrow f_\Sigma^{\max} - \max\{f(d') \mid x \leftarrow d' \text{ is admissible}\} + f(d)$.
8: $\quad f_{x \leftarrow d}^{\min} \leftarrow f_\Sigma^{\min} - \min\{f(d') \mid x \leftarrow d' \text{ is admissible}\} + f(d)$.
9: $\quad$ **if** $f_{x \leftarrow d}^{\max} < g$ **then**
10: $\quad\quad$ Add c with a degree of $\frac{g - f_{x \leftarrow d}^{\max}}{|\mathrm{var}_c| \cdot \max\{f(d') \mid d' \in D\}}$ to $\mathit{conflicts}[x \leftarrow d]$.
11: $\quad$ **else if** $f_{x \leftarrow d}^{\min} > g$ **then**
12: $\quad\quad$ Add c with a degree of $\frac{f_{x \leftarrow d}^{\min} - g}{|\mathrm{var}_c| \cdot \max\{f(d') \mid d' \in D\}}$ to $\mathit{conflicts}[x \leftarrow d]$.
13: $\quad$ **end if**.
14: **end for**.

verges still far more quickly on optimal solutions than the branch-and-bound. Apparently, especially the integration into iterative search makes cycle-cutset a really attractive method for solving constraint problems.

However, algorithm TREESOLVE is only applicable to subproblems of a *plain* constraint graph. Real world problems imply only in very rare cases plain constraint graphs since these applications depend very often on constraints of large arity (global constraints) that often overlap in more than one variable. The APPROX constraint of section 2 provides an example for such large arity constraints.

4 Solving Tree-structured Problems with Shallow Propagation

Algorithm MINMAXPROPAGATION looks for optimal support with respect to the propagated constraint *and* previously recognized conflicts. Both requirements in combination make it hardly possible to implement this kind of propagation efficiently for special types of large arity constraints — but efficient propagation of large arity constraints is one of the key factors for the success of constraint-based problem solving. Fortunately, algorithm *2* BRANCHANDBOUND can make use of a reduced form of constraint propagation that distinguishes only admissible from non-admissible labelings. A labeling $x \leftarrow d$ is called to be admissible iff it is an element of assignment A describing the currently explored branch of the search tree or x is a yet unlabeled variable and the currently known conflicts of $x \leftarrow d$ do not exceed the bound β [Meyer auf'm Hofe, 1999, Meyer auf'm Hofe, 2000]. This *shallow propagation* determines conflicts with a constraint c according to possible admissible assignments to the local variables of c. On the one hand, such *shallow propagation* is useful to detect conflicts early in a kind of extended forward checking as performed in line 12 of Algorithm 2. On the other hand, efficient algorithms can be found for many relevant constraints. As an example consider Algorithm 6 SHALLOWPROPAPPROX presenting a shallow propagation for the constraints of type APPROX that has been defined in section 2.

Algorithm 7 DIRECTEDCLUSTERPROP(x', C_T, β, δ, A, *conflicts*)

1: $C' \leftarrow \{c \in C_T \mid x$ is minimal local variable of c due to $>\}$. $X' \leftarrow \bigcup_{c \in C'} \mathrm{var}_c$.
2: **if** $C' = \emptyset$ **then return** *conflicts*, **end if**
3: **for all** $c \in C_T$ **do**
4: **if** $|X' \cap \mathrm{var}_c| \geq 2$ **then**
5: **if** x' is not minimal local variable due to $>$ **then return** *conflicts*, **end if**
6: Add all variables in var_c to X'. Insert c into C'.
7: **end if**
8: **end for**
9: **for all** $d \in D$ **do** Add all constraints with membership 1 to $bestConflicts[x' \leftarrow d]$. **end for**.
10: **for all** $A' \in D^{X' \setminus \{x'\}}$ of admissible labelings **do**
11: Initialize $tupleConflicts$ to hold an empty set for each $x \leftarrow d$ with $x \in \mathrm{var}_c \setminus X'$ and d $\in D$.
12: **for all** x $\in X'$ and $d \in D$ **do** $tupleConflicts[x' \leftarrow d] \leftarrow tupleConflicts[x' \leftarrow d] \sqcup conflicts[A'\!\downarrow \{x\}]$. **end for**.
13: **for all** $c \in C'$ **do** $tupleConflicts \leftarrow$ SHALLOWPROP($c, \beta, \delta, A \cup A', tupleConflicts$). **end for**
14: **for all** $d \in D$ **do**
15: **if** $bestConflicts[x' \leftarrow d] \succ tupleConflicts[x' \leftarrow d]$ **then** $bestConflicts[x' \leftarrow d] \leftarrow tupleConflicts[x' \leftarrow d]$. **end if**.
16: **end for**.
17: **end for**.
18: **for all** $d \in D$ **do** $conflicts[x' \leftarrow d] \leftarrow bestConflicts[x' \leftarrow d]$ **end for**.

Let A be the partial assignment describing the currently valid branch in the search tree. A's degree of violating a constraint of type APPROX is determined according to the distance between $\sum_{x \in \mathrm{var}_c} f(A\!\downarrow x)$ and a goal sum g. Algorithm 6 determines the conflicts for currently unlabeled variables by a simple procedure that at first sums up the minimal sum $f_\Sigma^{\min}$ and the maximal sum $f_\Sigma^{\max}$ of weights resulting from f on admissible values (in the loop starting at line 2). The second loop starting at line 6 uses these results to determine for each admissible labeling $x \leftarrow d$ the minimal and maximal sum of weights $f_{x \leftarrow d}^{\min}$ and $f_{x \leftarrow d}^{\max}$ that can result from extending $x \leftarrow d$ with admissible labelings. A complete satisfaction of the constraint is considered to be possible if the goal sum g lies between these values. Otherwise, an optimistic estimate on the degree of constraint violation that results from assigning d to x is computed from $f_{x \leftarrow d}^{\min}$ or $f_{x \leftarrow d}^{\max}$. The effort for this algorithm grows linear with the size of the domain D and the arity $|\mathrm{var}_c|$ and is, thus, very efficient.

Such algorithms detect conflicts with partial assignments as required in line 12 of Algorithm 2 and in line 7 of Algorithm 4. However, shallow propagation is not appropriate for the task of collecting all conflicts in a subproblem as required in line 2 of Algorithm 5 since shallow propagation only distinguishes between admissible and non-admissible labelings. These algorithms fail, thus, to propagate conflicts detected in deeper subtrees to higher nodes of a tree-structured problem. As a consequence, the call to DIRECTEDPROP in algorithm 4 shall be replaced by Algorithm 7 DIRECTEDCLUSTERPROP, that is able to solve tree-structured subproblems which form a hyper tree where hyper-arcs are allowed to share more than one variable.

Algorithm 7 DIRECTEDCLUSTERPROP is based on Freuder's idea to solve k-structured trees [Freuder, 1990]. The idea is to group the variables of a constraint problem into clusters in such a way that the arcs between the clusters form a tree. These clusters form subproblems that can be treated as single constraint variables where the set of all solutions to a cluster can be considered as the domain. Now, algorithms on solving tree-structured problems are able to combine solutions to the clusters in a consistent way. The complexity of this algorithm depends on the size k of the largest cluster.

Algorithm 7 applies this idea in a two phase procedure. The first phase spanning over the lines 1 to 8 determines the cluster containing variable x' where x' is the smallest variable due to $>$. Line 1 collects

constraints with x' as smallest local variable. As in Algorithm 5 DIRECTEDPROP, these constraints have to be propagated in order to collect the conflicts within the subtree below x'. The following loop starting with line 3 adds all constraints (and their local variables) to the cluster that overlap with more than one variable. The procedure exits immediately with result $conflicts$ if x' is either not the smallest local variable of a constraint or x' is not the smallest variable of the cluster. In these cases, DIRECTEDCLUSTERPROP is a *no operation* because the cluster containing x' has to be propagated later on in order to follow the tree-structure of the connections between the clusters. At the end of the first phase, X' and C' are the components of a cluster where x' is the smallest variable — and, therefore, the interface to the clusters which are higher in the tree. This cluster contains all constraints with x' as local variable and additionally all constraints that overlap in more than one variable.

The goal of phase two is to collect for each assignment $x' \leftarrow d$ to variable x' the conflicts of the best assignment to the variables X' of the clusters that labels x' with d since x' is the only variable that may also be part of clusters which are higher in the tree. This phase starts with line 10, an enumeration of all admissible assignments to the variables in $X' \setminus \{x'\}$. This loop collects all conflicts that have been detected for the current assignment into $tupleConflicts$. Shallow propagation of all constraints in C' adds all conflicts with constraints of the current cluster. $bestConflicts$ is set according to the result of the best assignment to the variables in the cluster. As a final result, $conflicts$ is set to the least important conflicts that have been detected. The effort for this algorithm grows with $|C'| \cdot |D|^{|X'|-1}$ multiplied with the effort for constraint propagation when only one local variable is not labeled.

Refer again to Fig. 3: The size of the subproblem is 21. The largest cluster in this nurse rostering example is of size 6 (row for nurse 3). This means that rather large subproblems can be optimized within iterative repair at comparably low costs. The effort for the large repair step of Fig. 3 should be pretty much the same as the effort for a sequence of repair steps each treating a single cluster by the branch-and-bound — but the results will differ in most cases. The application of DIRECTEDCLUSTERPROP conducts a global optimization whereas each step in the sequel only optimizes each cluster with respect to the shift assignments that are currently valid in the other clusters.

5 Conclusion

This paper suggests to apply efficient algorithms for special constraint problems — and exhibit for instance a k-tree structure — to real world applications. An example from a nurse rostering domain as well as first empirical results on randomly generated constraint problems illustrated the potential of such algorithms to improve the performance of repair steps in search by iterative improvement.

The paper sketched necessary modifications of algorithms taken from the literature in order to integrate special purpose algorithms into state of the art optimization by branch-and-bound extended with constraint propagation.

The suggested use of efficient algorithms within iterative improvement raises new questions on opportunities to control the search. Iterative improvement as presented in this paper is based on the heuristic detection of subproblems in the application that are responsible for suboptimal solutions. The suggested extensions reduce the effort for many extensive repair steps. As a consequence, further strategies for search control have to consider both: Assumptions on reasons for deficiencies in the current solution *and* assumptions on the effort for finding an improvement whereas the latter depends strongly on the applicability of efficient algorithms.

References

[Dechter and Pearl, 1987] Rina Dechter and Judea Pearl. The cycle-cutset method for improving search performance in AI applications. In *Proceedings of the 3rd IEEE Conference on AI Applications*, Orlando, FL, 1987.

[Dechter *et al.*, 1990] Rina Dechter, Avi Dechter, and Judea Pearl. Optimization in constraint networks. In R. M. Oliver and J. Q. Smith, editors, *Influence Diagrams, Belief Nets and Decision Analysis*, pages 411–425. John Wiley & Sons, 1990.

[Dechter, 1990] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[Dubois *et al.*, 1993] Didier Dubois, Hélène Fargier, and Henri Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. of the 2nd IEEE International Conference on Fuzzy Systems*, pages 1131–1136, San Francisco, CA, 1993.

[Freuder and Wallace, 1992] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[Freuder, 1982] Eugene C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, January 1982.

[Freuder, 1990] Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the AAAI*, pages 4–9, 1990.

[Meyer auf'm Hofe, 1997] Harald Meyer auf'm Hofe. ConPlan/ SIEDAplan: Personnel assignment as a problem of hierarchical constraint satisfaction. In *PACT-97: Proceedings of the Third International Conference on the Practical Application of Constraint Technology*, pages 257–272, London, UK, April 1997. Practical Application Expo.

[Meyer auf'm Hofe, 1999] Harald Meyer auf'm Hofe. *Kombinatorische Optimierung mit Constraintverfahren — Problemlösung ohne anwendungsspezifische Suchstrategien*. Dissertation, Fachbereich Informatik der Universität Kaiserslautern, November 1999. (Submitted.)

[Meyer auf'm Hofe, 2000] Harald Meyer auf'm Hofe. Nurse rostering as constraint satisfaction with fuzzy constraints and inferred control strategies. In Eugene C. Freuder and Rick J. Wallace, editors, *Volume on Constraint Programming and Large Scale Optimisation Problems*, DIMACS Volume Series, 2000. (Accepted.)

[Snow and Freuder, 1990] Paul Snow and Eugene C. Freuder. Improved relaxation and search methods for approximate constraint satisfaction with a maximin criterion. In *Proc. of the 8^{th} biennal conf. of the canadian society for comput. studies of intelligence*, pages 227–230, May 1990.