Conflict-directed A* Search for Soft Constraints

Martin Sachenbacher[1] and Brian C. Williams[2]

[1] LMU München, Oettingenstraße 67, 80538 München, Germany
sachenba@pms.ifi.lmu.de
[2] MIT CSAIL, 32 Vassar Street, Cambridge, MA 02139, USA
williams@mit.edu

Abstract. As many real-world problems involve user preferences, costs, or probabilities, constraint satisfaction has been extended to optimization by generalizing hard constraints to soft constraints. However, as techniques such as local consistency or conflict learning do not easily generalize to optimization, solving soft constraints appears more difficult than solving hard constraints. In this paper, we present an approach to solving soft constraints that exploits this disparity by re-formulating soft constraints into an optimization part (with unary objective functions), and a satisfiability part. This re-formulation is exploited by a search algorithm that enumerates subspaces with equal valuation, that is, plateaus in the search space, rather than individual elements of the space. Within the plateaus, familiar techniques for satisfiability can be exploited. Experimental results indicate that this hybrid approach is in some cases more efficient than other known methods for solving soft constraints.

1 Introduction

Many real-world problems are naturally framed as optimization problems where the task is to find assignments to variables that optimize user preference, cost, or probability. Therefore, constraint satisfaction problems (CSPs) have been extended from satisfaction to optimization by the notion of soft constraints. One general framework for soft constraints are valued constraint satisfaction problems (VCSPs) [22, 1], which augment CSPs with a valuation structure and generalize many earlier notions such as fuzzy CSPs, probabilistic CSPs, or partial constraint satisfaction.

For the case of solving CSPs, techniques such as local consistency filtering [16] and conflict (nogood) learning [5] have proven to be very effective. Substantial progress has been made in extending these techniques to the more general case of soft constraints [2, 7]; however, the optimization case still appears far more difficult than the satisfaction case.

In practical applications, the constraints often exhibit structure or regularities that can be exploited in order to make optimization feasible. For instance, approaches based on tree decomposition [8, 12] exploit favorable properties of the constraint graph (limited width) to break down the problem into lower-dimensional subproblems.

In this paper, we present an approach to exploit a form of structure that is specific to optimization problems. It has been underlying algorithmic approaches in the area of model-based reasoning and diagnosis [26, 9] for quite some time. Model-based reasoning aims at describing the behavior of physical systems in terms of formal models, where some variables capture preferences (such as the failure probability of a component, or the cost of repairing it), and constraints capture consistency (such as the physically possible behavior of a component). As the number of variables defining preferences is typically small compared to the overall number of variables, there often exist large sets of assignments that have equal valuation, that is, large "plateaus" in the search space. Williams [27] presents an algorithmic approach called *conflict-directed A** that exploits this fact by coupling together optimization and satisfaction techniques. The algorithm enumerates plateaus (parts of the search space with the same valuation) in best-first order, and subsequently checks if there exists a consistent solution within the plateau; information about infeasible assignments is re-used between the plateaus in the form of conflicts. This approach can be more efficient than enumerating individual elements of the search space, because depending on the problem, there can be fewer plateaus than total elements in the search space.

In this paper, we generalize upon these ideas, and extend their applicability from model-based reasoning applications to the general case of soft constraints. Our approach consists of factoring VCSPs into a set of (unary) soft constraints that carry all the information about valuations of assignments, and a set of hard constraints that do not carry valuations but just need to be satisfied. Like in [18, 19], this re-formulation is based on introducing additional variables capturing the cost of violating constraints; a special case of this re-formulation is taking the dual of the problem [14]. Following the terminology of [27], we call the resulting hybrid representation *optimal CSP*; it makes explicit the optimization and satisfiability aspects of a problem. The idea is then to algorithmically exploit the separation into hard and soft problem parts by applying optimization techniques to the optimization part, and satisfiability techniques to the satisfiability part. Specifically, for a reasonably small soft constraint part, we can use A* search [10], which is optimal in the number of search nodes visited, but would typically be infeasible to apply on the original VCSP problem due to its memory requirements. For the hard constraint part, we can draw on well-established, efficient techniques for satisfiability problems, such as local consistency and conflict learning; in this paper, we build on an existing SAT solver for this purpose.

The paper is organized as follows: We review the definitions of valued CSPs [22] and optimal CSPs [27] and present a method for transforming between them. The transformation yields a separation into hard constraints and unary soft constraints. We then present a variant of conflict-directed A* that exploits this re-formulation by searching over sets of assignment with equal valuation, rather than searching over individual assignments of the variables in the problem. We give experimental results demonstrating that this algorithm can outperform other methods for solving valued CSPs, and finally we indicate some directions for future work.

2 Valued CSPs

A classical *constraint satisfaction problem* (CSP) is a triple (X, D, C) with variables $X = \{x_1, \ldots, x_n\}$, finite domains $D = \{\mathrm{dom}(x_1), \ldots, \mathrm{dom}(x_n)\}$, and constraints $C = \{c_1, \ldots, c_m\}$. Each constraint $c_j \in C$ is a relation $c_j \subseteq \Pi_{x_i \in \mathrm{var}(c_j)} \mathrm{dom}(x_i)$ over variables $\mathrm{var}(c_j) \subseteq X$. An assignment t to variables $\mathrm{var}(c_j)$ *satisfies* the constraint if $t \in c_j$, and *violates* it otherwise.

Definition 1 (Valuation Structure [22]). *A* valuation structure *is a tuple $(E, \leq, \oplus, \bot, \top)$ where E is a set of valuations, totally ordered by $\leq$ with a minimum element $\bot \in E$ and a maximum element $\top \in E$, and $\oplus$ is an associative, commutative, and monotonic binary operation with identity element $\bot$ and absorbing element $\top$.*

The set of valuations E expresses different levels of constraint violation, such that $\bot$ means satisfaction and $\top$ means unacceptable violation. The operation $\oplus$ is used to combine (aggregate) several valuations. A constraint is *hard*, if all its valuations are either $\bot$ or $\top$.

Definition 2 (Valued Constraint Satisfaction Problem [22]). *A* valued constraint satisfaction problem *(VCSP) consists of a classical CSP (X, D, C) with valuation structure $(E, \leq, \oplus, \bot, \top)$, and a mapping ϕ from C to E which associates a valuation with each constraint.*

For example, the problem of diagnosing the polycell circuit in Fig. 1 [27] can be framed as a VCSP with variables $X = \{a, b, c, d, e, f, g, x, y, z\}$. Each variable models a boolean signal and has domain $\{0, 1\}$. The VCSP has five ternary constraints f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2} corresponding to gates in the circuit, and four unary constraints f_c, f_d, f_f, f_g corresponding to observations. The constraints f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2} express that the respective gates are performing their boolean functions. The constraints f_c, f_d, f_f, f_g express that variables c, d, and g are observed to be 1, and variable f is observed to be 0. The valuation structure $(\mathbb{N}_0^+ \cup \infty, +, \leq, 0, \infty)$ captures the cost of violating a constraint, which we assume to be 1 for the constraints f_{o1}, f_{o2}, f_{o3}, to be 2 for the constraints f_{a1} and f_{a2}, and to be ∞ for the constraints modeling the observations.

Given a VCSP, the problem is to find an assignment t to X which mimimizes the combined valuation of all violated constraints, $\bigoplus_{\{c_j \in C | t[\mathrm{var}(c_j)] \notin c_j\}} \phi(c)$. For the boolean polycell example, the minimum valuation of an assignment is 1, corresponding to a fault of a single OR gate.

3 Optimal CSPs

Since solving VCSPs is more complex than solving classical CSPs, an algorithmic approach that is based on spliting the VCSP into a set of classical (hard) constraints and a set of valued (soft) constraints can be useful.

In the following, we consider a specialization of this approach where the constraints are divided into hard constraints and unary soft constraints. In [27], this type of optimization problem is called *optimal CSP*:

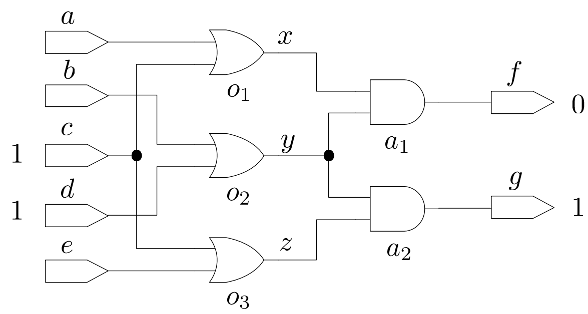

Fig. 1. The boolean polycell example consists of three OR gates and two AND gates. Variables c, d, f, and g are observed as indicated.

Definition 3 (Optimal CSP). *An* optimal CSP *(OCSP) consists of a classical CSP* (X, D, C)*, with valuation structure* $(E, \leq, \oplus, \perp, \top)$*, and a set* U *of unary functions* $u_j : \mathrm{dom}(y_j) \rightarrow E$ *defined over a subset* $Y \subseteq X$ *of the variables. The variables in* Y *are called decision variables, and the variables in* $X \setminus Y$ *are called non-decision variables.*

An OCSP can be viewed as a special case of a VCSP where soft constraints (constraints with valuation $\phi(c_j) < \top$) must be unary. A solution to an OCSP is an assignment to Y with minimal total valuation (combined valuations of functions u_j), such that there exists an extension to all variables X that satisfies all the constraints in the CSP. Hence, whereas a solution to a VCSP is a single assignments to X, a solution to an OCSP is an assignment to the decision variables Y that can stand for a whole collection of assignments to X that have all the same valuation (plateau) and differ only with respect to the non-decision variables $X \setminus Y$. It is observed in [14] that a number of optimization problems can be directly expressed with hard and unary soft constraints, that is, as OCSPs; an example are combinatorial auctions [21].

4 Translation from Valued CSPs to Optimal CSPs

In general, a VCSP may have non-unary soft constraints and thus it does not necessarily have the form of an OCSP. However, it is possible to *transform* a VCSP into an OCSP with an equivalent optimal solution. This transformation is based on introducing additional variables (decision variables) that capture the cost of violating a constraint of the VCSP, analogous to the *hidden variable representation* described in [14] and the construction of *disjunctive constraints* for over-constrained problems described in [18, 19]. The translation demonstrates that OCSPs, though syntactically more restricted than VCSPs, actually have the same expressive power as VCSPs. OCSPs could therefore be viewed as a "normalization" of VCSPs that achieves our desired separation into a hard constraint part and a soft constraint part.

Introducing a decision variable for every constraint turns a VCSP with n variables and m constraints into an OCSP with $n+m$ variables. We can further reduce the size of the OCSP by observing that for any hard constraint c_j in the VCSP ($\phi(c_j) = \top$), choosing the value *false* for its corresponding decision variable y_j can never give rise to a solution of the OCSP because it will immediately lead to the valuation $\top$. Therefore, we do not need to introduce decision variables for hard constraints in the VCSP.

Definition 4 (Translation of VCSP to OCSP). *The translation of a VCSP (X, D, C) with valuation structure $(E, \leq, \oplus, \perp, \top)$ and mapping ϕ into an OCSP (X', D', C') with unary functions U over decision variables $Y \subseteq X'$ is defined as follows:*

- *X' consists of X and one decision variable y_j for each constraint $c_j \in C$ for which $\phi(c_j) < \top$;*
- *D' consists of D and the domain* {true, false} *for each decision variable y_j;*
- *U consists of one unary function u_j per decision variable y_j. The function maps the value* true *to $\perp$ and the value* false *to $\phi(c_j)$;*
- *C' consists of one constraint c'_j for each $c_j \in C$. If $\phi(c_j) = \top$ then $c'_j = c_j$, else c'_j is a relation over variables $var(c'_j) = \mathrm{var}(c_j) \cup y_j$. An assignment t to $var(c'_j) = \mathrm{var}(c_j) \cup y_j$ satisfies c'_j iff $t[\mathrm{var}(c_j)] \in c_j$ and $y_j = \mathrm{true}$ or $t[\mathrm{var}(c_j)] \notin c_j$ and $y_j = \mathrm{false}$.*

For example, the translation of the VCSP for the boolean polycell circuit yields an OCSP with variables $\{a, b, c, d, e, f, g, x, y, z, y_1, y_2, \ldots, y_5\}$. Variables $\{a, b, c, d, e, f, g, x, y, z\}$ are non-decision variables, and variables y_1 to y_5 are decision variables, obtained by extending the constraints f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2} with an additional variable. There are five unary functions $u_1, u_2, \ldots, u_5 \in U$, capturing the cost of violating the constraints f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2}. No decision variables need to be introduced for the hard constraints f_c, f_d, f_f, f_g corresponding to observations.

Theorem 1. *A VCSP and its translation to an OCSP have the same optimal solution.*

Note that for the special case of a VCSP that is actually a CSP (a VCSP where $\phi(c_j) = \top$ for all $c_j \in C$), the reduced translation is the CSP itself. Therefore, solving a CSP as an OCSP does not incur any overhead.

5 Solving OCSPs

The separation of valued CSPs into unary soft constraints and hard constraints can be algorithmically exploited by coupling together specialized algorithms for each part. In particular, for the hard constraint part, we can employ techniques that are highly optimized for satisfaction problems, and for the soft constraint part, we can employ techniques that work best for a relatively small optimization problem but would be infeasible for the original, bigger problem. This hybrid algorithmic approach can be more efficient than general solvers for soft constraints that do not make assumptions about how the valuations are distributed over the space of assignments.

5.1 Conflict-directed A* Search

Williams and Ragno [27] describe such a hybrid approach for solving a subclass of OCSPs. The approach, called *conflict-directed A**, exploits the distinction between decision variables (which determine the valuation of an assignment) and non-decision variables (which determine only the consistency of an assignment) by treating them separately: it enumerates assignments to the decision variables (corresponding to plateaus) in best-first order. Once a complete assignment to the decision variables has been found, it is checked whether the CSP part can be satisfied by assigning the remaining, non-decision variables. If the CSP is satisfiable (corresponding to the plateau being non-empty), an optimal solution has been found. If the CSP is unsatisfiable (corresponding to the plateau being empty), one or more conflicts (inconsistent instantiations of variables, see [5]) are extracted and used to speed up and focus the further search. Depending on the problem structure, there can be fewer plateaus than individual elements of the search space, and therefore this two-step approach can be more efficient than enumerating the individual elements of the search space.

The enumeration of assignments is based on A* search [10], an instance of best-first search that uses a lower bound g for the partial assignment made so far, and an optimistic estimate h of the value that can be achieved when completing the assignment; at each point in the search, A* expands the assignment with the best combined value of g and h. A* search is *run-time optimal* [3] in that it visits a minimum number of search nodes (among all search methods having access to the same heuristics). Due to its memory requirements, which are worst-case exponential in the number of variables, A* search would hardly be feasible as a solution method for general VCSPs. However, as observed in [27], the memory requirements of A* search can be much more modest in the case of OCSPs, because only assignments to variables that have an associated cost (decision variables) need to be stored in the search queue; in addition, the conflicts further reduce the size of the queue.

In the following, we present a simplified variant of conflict-directed A* that is adapted to OCSPs obtained from VCSPs. As a heuristic estimate h for the cost of completing an assignment to the decision variables, we simply sum up the best possible valuation for each remaining (unassigned) decision variable; since the best possible valuation of a decision variable is $\perp$, it means h is equal to $\perp$. The pseudo-code of the algorithm is shown in Alg. 1. First, local consistency is established in the CSP part of the OCSP. If an inconsistency arises during local propagation, then the OCSP has no consistent solution (no assignment with valuation better than $\top$). Otherwise, the algorithm performs a best-first (A*) search over assignments to the decision variables Y of the OCSP, using a priority queue of (partial) assignments to Y that is ordered by their valuation. The A* search is based on two sub-procedures updateAssignment() and switchAssignment(), shown in Proc. 2 and Proc. 3, respectively. Procedure switchAssignment() establishes a (partial) assignment a to the decision variables from the queue, trying to reuse as much as possible the current search tree; it backtracks to the deepest point in the search tree up to which the current assignment to Y and a are the same. If an inconsistency occurs while trying to establish the assignment, then a conflict is extracted and added to the set of constraints, and the assignment is discarded. Next, updateAssignment() is used to assign decision variables that have only one value remaining, and extend the assignment (and in particular, its valuation) accordingly. Since this update might increase the valuation of the current assignment, it is now possible that it is no longer the best assignment; in this case, the assignment is pushed back into the queue. Otherwise (if the current assignment is still the best one), it is checked whether the assignment to the decision variables is complete. If the assignment is incomplete, the algorithm chooses a next decision variable y_i to assign and enqueues the two possible branches $y_i \leftarrow$ true and $y_i \leftarrow$ false. If the assignment to the decision variables is complete, then the algorithm uses procedure consistentAssignment() to check if the assignment is consistent with the CSP. To this end, consistentAssignment() tries to extend the assignment to $Y \subseteq X$ to an assignment to X by assigning the remaining (non-decision) variables $X \backslash Y$. In Proc. 4, this is done using depth-first search with conflict-directed backjumping. The current level of the search tree (which so far involves only decision variables) is frozen in variable decisionLevel, and whenever a conflict occurs that would require to backup higher than this level (backtrackLevel smaller than or equal to decisionLevel), the current assignment to the decision variables must be inconsistent and is discarded. Otherwise, the assignment is output as the next best solution.

For instance, for the boolean polycell example and the OCSP encoding in Def. 4, the algorithm has to assign five decision variables $y_1, y_2, \ldots, y_5$ corresponding to the constraints $f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2}$. Conflict-directed A* starts with an empty assignment to the decision variables. Propagation does not prune any values for the decision variables, so the algorithm assigns a decision variable. Assume the decision variables are assigned in the order $y_1, y_2, \ldots, y_5$. The algorithm thus creates two new assignments, $\langle y_1 \leftarrow \text{true}\rangle$ with valuation 0 and $\langle y_1 \leftarrow \text{false}\rangle$ with valuation 1, and puts them on the queue. The algorithm pops the assignment $\langle y_1 \leftarrow \text{true}\rangle$ from the queue and establishes it using function switchAssignment(). Two new assignments, $\langle y_1 \leftarrow \text{true}, y_2 \leftarrow \text{true}\rangle$ with valuation 0 and $\langle y_1 \leftarrow \text{true}, \text{y}_2 \leftarrow \text{false}\rangle$ with valuation 1 are created and enqueued. When establishing the best assignment $\langle y_1 \leftarrow \text{true}, y_2 \leftarrow \text{true}\rangle$ using switchAssignment(), propagation forces y_4 to be false, and thus updateAssignment() refines the assignment to $\langle y_1 \leftarrow \text{true}, y_2 \leftarrow \text{true}, y_4 \leftarrow \text{false}\rangle$ with valuation 2. Since a better assignment exists in the queue, this assignment is pushed back into the queue, and the next best assignment, say $\langle y_1 \leftarrow \text{false}\rangle$ with valuation 1, is considered. Since this new assignment and the current assignment share no common prefix, switchAssignment() needs to backtrack up to y_1 in order to establish this assignment. After propagation, the updated assignment becomes $\langle y_1 \leftarrow \text{false}, y_4 \leftarrow \text{true}\rangle$ with valuation 1. The algorithm proceeds by assigning $y_2 \leftarrow$ true and $y_3 \leftarrow$ true, at which point $y_5 \leftarrow$ true can be derived by propagation, and therefore a complete decision variable assignment $\langle y_1 \leftarrow \text{false}, y_2 \leftarrow \text{true}, y_3 \leftarrow \text{true}, y_4 \leftarrow \text{true}, y_5 \leftarrow \text{true}\rangle$ with valuation 1 is obtained. Procedure consistentAssignment() determines that this assignment is consistent (a satisfying assignment to the non-decision variables is e.g. $\langle a \leftarrow 1, b \leftarrow 1, c \leftarrow 1, d \leftarrow 1, e \leftarrow 0, f \leftarrow 0, g \leftarrow 1, x \leftarrow 0, y \leftarrow 1, z \leftarrow 1\rangle$), and thus outputs value 1 as the optimal solution.

Theorem 2. *The conflict-directed A* algorithm in Alg. 1 computes the optimal solution of a given OCSP.*

Conflict-directed A* search can be further refined in a number of ways. [27, 15] describe extensions that reduce the size of the search queue by generating new entries only at a point where the current assignment to the decision variables becomes inconsistent, and an extension to the case of non-binary decision variables that generates only next best child assignments instead of all children at once. It is also easy to extend the algorithm such that it enumerates the solutions in best-first order, instead of computing only the optimal solution.

Algorithm 1 Conflict-directed A* for OCSPs

```
if not (propagate() = conflict) then
  queue ← ⟨∅, ⊥⟩
  while queue ≠ ∅ do
    ⟨a, value⟩ ← first(queue)
    queue ← removeFirst(queue)
    if switchAssignment(a) then
      updateAssignment(⟨a, value⟩)
      if assignment with better value exists in queue then
        queue ← push(queue, ⟨a, value⟩)
      else
        if exists y_i ∈ Y, y_i = unknown then
          queue ← push(queue, ⟨a ∪ (y_i ← true), v⟩)
          queue ← push(queue, ⟨a ∪ (y_i ← false), v ⊕ φ(c_i)⟩)
        else
          if consistentAssignment() then
            output value as best solution
            exit
          end if
        end if
      end if
    end if
  end while
end if
output no solution
```

Procedure 2 updateAssignment($\langle a, \text{value}\rangle$)

```
for all y_i ∈ Y, y_i ∉ a, y_i ≠ unknown do
    if y_i = true then
        ⟨a, value⟩ ← ⟨a ∪ (y_i ← true), value⟩
    else
        ⟨a, value⟩ ← ⟨a ∪ (y_i ← false), value ⊕ φ(c_i)⟩
    end if
end for
```

Procedure 3 switchAssignment(a)

```
level ← deepest level up to which a and current assignment are equal
backtrack(level)
for (y_i ← val) ∈ a do
    if y_i ≠ val then
        return false
    else if y_i = unknown then
        y_i ← val
        level ← level + 1
        if propagate() = conflict then
            CSP ← CSP ∪ conflict
            return false
        end if
    end if
end for
return true
```

Procedure 4 consistentAssignment()

```
decisionLevel ← level
while exists x_i ∈ X \ Y, x_i = unknown do
    choose val ∈ dom(x_i)
    x_i ← val
    level ← level + 1
    dom(x_i) ← dom(x_i) − val
    if propagate() = conflict then
        backtrackLevel ← analyze(conflict)
        if backtrackLevel ≤ decisionLevel then
            return false
        else
            CSP ← CSP ∪ conflict
            backtrack(backtrackLevel)
            level ← backtrackLevel
        end if
    end if
end while
return true
```

6 Implementation

We have implemented the transformation of VCSPs into OCSPs and the conflict-directed A* search algorithm in C++. Conflict-directed A* search was implemented on top of zChaff [17], one of the most efficient complete solvers for boolean satisfiability (SAT) problems. The main reasons why we choose zChaff is that it offers (1) a highly optimized data-structure for local consistency (unit propagation), called *two-literal watching scheme*; (2) a method for extracting small conflicts from inconsistencies, based on so-called *unique implications points* (UIPs), which correspond to dominators in the implication graph; and (3) an efficient variable and value ordering heuristic called *variable state independent decaying sum* (VSIDS), which biases the search towards variables that occur in recently learned clauses, i.e., conflicts. (In addition, zChaff uses other techniques such as random restarts, which we do not exploit in our prototype).

Our prototypic implementation of conflict-directed A* adopts zChaff's local propagation scheme, its conflict extraction method, and its variable/value ordering heuristic for the non-decision variables. The decision variables are currently assigned in no specific order. Using a SAT solver as the underlying satisfiability engine means that the CSP part of the OCSP has to be first encoded as a SAT problem, by mapping variables to boolean variables, and mapping constraints to clauses in conjunctive normal form (CNF). For this purpose, we choose a logarithmic SAT encoding of the CSP [11], although other encodings are equally possible (see [25, 6] for two alternative encodings).

7 Experimental Results

We evaluated our prototype on various examples of valued CSPs, and compared its performance against other algorithms for solving soft constraints.

The algorithms we compared against are branch-and-bound with maintaining existential directional arc consistency (BB-MEDAC) [7], and cluster tree elimination (CTE) [4]. BB-MEDAC is a recently proposed search algorithm that combines depth-first branch-and-bound with a form of arc consistency generalized to soft constraints. In our experiments we used the implementation that is part of the TOOLBAR package [24]. CTE is an inference algorithm for both hard constraints and soft constraints that is based on decomposing the constraint graph into a tree structure, and solving it using dynamic programming. In our experiments, the tree was computed using a greedy min-fill heuristic.

All the examples shown below, apart from the random problems, are taken from the TOOLBAR repository. All experiments were performed under Windows XP using a 2.8 GHz Pentium 4 PC with 1 GB of Ram.

7.1 Academic Problems

First, we tried conflict-directed A* on three academic puzzles. Since these examples involve only hard constraints, the corresponding OCSPs do not contain any decision variables, and thus conflict-directed A* can solve these problems as efficiently as the underlying satisfiability engine (in our implementation, zChaff with the given SAT encoding). For all three algorithms, we used a time bound of 1 minute. Table 1 summarizes the results. Although these examples are relatively small, note that CTE fails to solve all but one of them within the given time bound.

Table 1. Results for academic puzzles (containing only hard constraints).

	CDA*	BB-MEDAC	CTE
zebra (25 variables, 19 constraints)	0.188 sec	0.016 sec	0.047 sec
send (11 variables, 32 constraints)	0.312 sec	0.031 sec	> 1 min
donald (15 variables, 51 constraints)	2.828 sec	0.156 sec	> 1 min

7.2 Random Problems

Next, we compared the algorithms on random Max-CSP problems. Max-CSPs are instances of VCSPs where each constraint has cost 1; thus, the task is to minimize the number of violated constraints. To generate the examples, we used a random binary constraint model with four parameters N, K, C, and T, where N is the number of variables, K the domain size, C the number of constraints, and T the tightness of each constraint (number of tuples having cost 1). Again, we used a time bound of 1 minute. Table 2 summarizes the results for six classes of random Max-CSP, averaged over 10 instances each.

Table 2. Results for random Max-CSPs (10 instances each).

(N, K, C, T)	CDA*	BB-MEDAC	CTE
(40, 4, 60, 4)	0.0346 sec	0.0092 sec	1.461 sec
(40, 4, 60, 8)	2.184 sec	0.022 sec	4.136 sec
(40, 4, 60, 12)	> 1 min	0.0468 sec	7.325 sec
(25, 4, 100, 4)	0.818 sec	0.0156 sec	> 1 min
(25, 4, 100, 8)	> 1 min	0.169 sec	> 1 min
(25, 4, 100, 12)	> 1 min	0.131 sec	> 1 min

For all these examples, BB-MEDAC converges very fast towards the optimal solution. Unfortunately, conflict-directed A* does not perform well for the denser and tighter instances. Further analysis of these cases reveals that the algorithm actually quickly finds small conflicts that could potentially guide the A* search towards the optimal solution, but then tries many assignments to the decision variables that are useless as they are not relevant to (i.e., do not resolve) those conflicts. Thus, we expect that using a similar variable ordering heuristic for the decision variables as for the non-decision variables (focusing on variables involved in conflicts) could substantially improve the performance of conflict-directed A* for these cases.

7.3 Real-world Problems

Finally, we evaluated the performance of our algorithm on four real-world circuit examples. These are obtained by turning SAT instances from the DIMACS challenge into Max-CSPs by making each clause a constraint with cost 1. For these examples, we used a time bound of 10 minutes. Table 3 summarizes the results.

Table 3. Results for DIMACS circuit examples.

	CDA*	BB-MEDAC	CTE
ssa0432-003 (435 variables, 1027 constraints)	14.547 sec	> 10 min	1.219 sec
ssa7552-038 (1501 variables, 3575 constraints)	28.312 sec	> 10 min	142.969 sec
ssa2670-141 (986 variables, 2315 constraints)	101.765 sec	> 10 min	6.21 sec
ssa2670-130 (1359 variables, 3321 constraints)	233.89 sec	> 10 min	53.203 sec

CTE performs best for most of these examples; however, the run-times for CTE in Table 3 show only run-times of CTE itself and do not include the time for computing the tree decomposition, which takes longer than the run-time of CTE for some of the examples. Also, CTE requires significantly more memory than the other algorithms for most of the examples. BB-MEDAC, which performed best for the academic and random examples, cannot solve any of the DIMACS examples within the given time bound. In fact, even after 10 minutes of computation, its lower bound (best valuation found so far) is often far off the optimal solution. We suspect that this has to do with the fact that BB-MEDAC performs local propagation (existential directional arc consistency) for binary constraints only, and defers the propagation of non-binary constraints until they become binary. Thus, the propagation scheme is not effective for the DIMACS examples where almost all constraints are non-binary. In contrast, conflict-directed A* exploits efficient local propagation (zChaff's two literal scheme) for any hard constraints. In fact, for instance ssa7552-038, which has optimal cost 0, conflict-directed A* requires only one call to the SAT engine (zChaff) in order to solve it. The actual run-time of zChaff for this example is only a fraction of the run-time given in Table 3, indicating that the current implementation of conflict-directed A* wastes significant time constructing unnecessary search queue entries. We therefore expect that further improvements to the algorithm to reduce the size of the search queue by creating entries only as needed (as described in [27, 15]) will have a strong impact for these examples.

8 Discussion and Related Work

The idea of re-formulating problems into a part describing the cost of a solution and a part describing its feasibility is not new. Larrosa and Dechter [14] already found that dualization turns soft constraints into a set of hard constraints and unary soft constraints, without losing expressiveness. Petit et al. [18, 19] describe an approach to model optimization problems by specifying an additional variable (decision variable) for each constraint, capturing the cost of its violation; they show how this representation allows for additional expressiveness, for example, specifying "meta-constraints" between decision variables to control the distribution of violations. In contrast, our goal is to use re-formulation to make the structure in soft constraints more explicit; in particular, we believe separating optimization aspects from satisfaction aspects may provide a useful starting point for algorithmic development. Conflict-directed A* is an instance of such an approach, and it is inspired by research on model-based reasoning and diagnosis [26, 9], where problems can be naturally framed as a mixture of hard constraints and unary objective functions (that is, OCSPs).

From the perspective of viewing re-formulation as a process of "pre-compiling" preferences, the separation into unary soft constraints and hard constraints is only a special case; it is not actually required by the approach that the soft constraints are unary. Another useful view of the re-formulation into OCSPs is that of giving a "normal form" for soft constraints, which makes the degree to which the problem is an optimization problem vs. a satisfaction problem more explicit. It seems that research in soft constraints has so far focussed on expressive, unifying frameworks, but much less on such canonical representations. Optimal CSPs could provide a starting point in this direction.

A potential drawback of the re-formulation is that it may increase the size of the problem; since one decision variable is introduced for each soft constraint, the resulting OCSP may be much bigger than the original VCSP, especially if it has a high ratio of constraints to variables. However, even if the re-formulation incurs an increase in the problem size, the benefit of applying dedicated solvers to each part of the problem (as in conflict-directed A*) may still outweigh the increase in the search space. The identification of problem classes for which re-formulation is beneficial is a subject of further research.

As already indicated in Sec. 5.1, several improvements to conflict-directed A* are possible, in particular for switchAssignment(), the procedure that is most critical to the performance of the algorithm. The cost of switching between two A* search nodes (corresponding to two different assignments to the decision variables, i.e., two different CSPs) could be reduced by incremental techniques that allow for computing only the difference between two CSP instances. Truth maintenance systems (TMS) [13], which keep track of the dependencies in the implication graph, are frequently used in model-based reasoning and diagnosis for this purpose. However, the additional bookkeeping necessitated by the TMS creates a trade-off between between making the context switch more efficient and making the satisfiability check more efficient.

Another direction for future work is to combine conflict-directed A* search with structural (tree decomposition) methods. As can be seen from the experiments, the two approaches are fairly complementary to each other, and decomposing the problem into smaller subproblems can dramatically improve performance on examples with low tree width. The combination would involve an instance of conflict-directed A* running on every cluster in the tree, and a special set of decision variables that capture the cost of assignments to variables shared between clusters (separator variables). We are currently working on such a decomposed version of conflict-directed A*. Some earlier work on combining best-first search with tree decompositions can be found in [20], whereas [23] describes a method for (the simpler case of) combining depth-first search with tree decompositions.

In our implementation, we used a SAT solver (zChaff) to check consistency of the candidates (plateaus) enumerated by A* search, mainly for the reason that it provides an efficient implementation of local propagation and conflict extraction. Recently, the problem of extending SAT solvers to optimization counterparts where either the number of satisfied clauses must be maximized (max-SAT) or the clauses carry a weight to be maximized (weighted max-SAT) has received considerable attention [28]. Much of this work still focuses on extending the basic DPLL search algorithm that underlies most complete SAT solvers (especially the unit propagation and variable ordering heuristic) to this case, and does not yet exploit more advanced concepts like conflicts. Still, it would be interesting to compare such approaches to our method.

9 Conclusion

We presented an approach for transforming VCSPs into hard constraints and unary soft constraints (OCSPs), and exploiting this re-formulation by solving the optimization and satisfiability part separately using a combination of specialized algorithms. Because it can exploit structure in the search space by enumerating whole sets of assignments with equal valuations (plateaus) rather than just individual assignments, this hybrid approach can be more efficient than algorithms that work directly on the VCSP. We presented an instance of this approach, called conflict-directed A*, and its prototypic implementation on top of a SAT solver. The prototype can outperform other solvers for VCSPs on some problems of practical importance. Promising directions for future research include more sophisticated, incremental methods for the critical step of switching between plateaus, and incorporating structural decomposition methods.

References

[1] Bistarelli, S., et al.: Semiring-based CSPs and Valued CSPs: Frameworks, Properties, and Comparison. Constraints **4** (3) (1999) 199–240

[2] Cooper, M., and Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence **154** (2004) 199-227

[3] Dechter, R., Pearl, J.: Generalized Best-First Search Strategies and the Optimality of A*. Journal of the ACM **32** (3) (1985) 505–536
[4] Dechter, R., Pearl, J.: Tree clustering for constraint networks. Artificial Intelligence **38** (1989) 353–366
[5] Dechter, R.: Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. Artificial Intelligence **41** (1990) 273-312.
[6] Gent, I.P.: Arc consistency in SAT. Proc. ECAI-2002 (2002)
[7] de Givry, S., Zytnicki, M., Heras, F., and Larrosa, J.: Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. Proc. of IJCAI-2005 (2005)
[8] Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence **124** (2) (2000) 243–282
[9] W. Hamscher, W., Console, L., and de Kleer, J. (eds.): Readings in Model-Based Diagnosis, Morgan Kaufmann (1992)
[10] Hart, P. E., Nilsson, N. J., and Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Sys. Sci. Cybern. **SSC–4** (2) (1968) 100-107.
[11] Iwama, K, and Miyazaki, S.: SAT-variable complexity of hard combinatorial problems. IFIP World Computer Congress (1994) 253-258
[12] Kask, K., et al.: Unifying Tree-Decomposition Schemes for Automated Reasoning. Technical Report, University of California, Irvine (2001)
[13] de Kleer, J.: An Assumption based TMS, Artificial Intelligence **28** (1) (1986) 127–162
[14] Larrosa, J., and Dechter, R.: On the Dual Representation of non-binary Semiring-based CSPs. Proceedings SOFT-2000 (2000)
[15] Li, H., and Williams, B.C.: Generalized Conflict Learning for Hybrid Discrete/Linear Optimization, Proc. CP-2005 (2005)
[16] Mackworth, A.: Constraint satisfaction. Encyclopedia of AI (second edition) **1** (1992) 285–293
[17] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., and Malik, S.: Chaff: Engineering an efficient SAT solver. In Proc. of the Design Automation Conference (DAC) (2001)
[18] Petit, T., Régin, J.-C., and Bessière, C.: Meta-Constraints on Violations for Over-Constrained Problems, Proc. ICTAI-2000 (2000) 358–365.
[19] Petit, T., Régin, J.-C., and Bessière, C.: Specific Filtering Algorithms for Over-Constrained Problems, Proc. CP-2001 (2001) 451–465.
[20] Sachenbacher, M., and Williams, B.C.: On-demand Bound Computation for Best-First Constraint Optimization, Proc. CP-2004 (2004)
[21] Sandholm, T.: An algorithm for optimal winner determination in combinatorial auctions. Proceedings IJCAI-1999 (1999)
[22] Schiex, T., Fargier, H., Verfaillie, G.: Valued Constraint Satisfaction Problems: hard and easy problems. Proc. IJCAI-95 (1995) 631–637
[23] Terrioux, C., Jégou, P.: Bounded Backtracking for the Valued Constraint Satisfaction Problems. Proc. CP-2003 (2003)
[24] Toolbar `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro`
[25] Walsh, T.: SAT vs. CSP. Proc. CP-2000 (2000) 441-456
[26] Weld, D.S., and de Kleer, J. (eds.): Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann (1989)
[27] Williams, B., Ragno, R.: Conflict-directed A* and its Role in Model-based Embedded Systems. Journal of Discrete Applied Mathematics, to appear.
[28] Xing, Z., and Zhang, W.: MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability. Artificial Intelligence **164** (1–2) (2005) 47-80