# Solving Soft Constraints by Separating Optimization and Satisfiability

Martin Sachenbacher and Brian C. Williams

MIT CSAIL, 32 Vassar Street, Cambridge, MA 02139, USA
{sachenba,williams}@mit.edu

**Abstract.** As many real-world problems involve user preferences, costs, or probabilities, the constraint framework has been extended from satisfaction to optimization by extending hard constraints to soft constraints. However, techniques for constraint satisfaction, such as local consistency or conflict learning, do not easily generalize to optimization. Thus, 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. We describe a search algorithm that exploits this re-formulation by enumerating subspaces with equal valuation, that is, plateaus in the search space, rather than individual elements of the space. Experimental results indicate that this hybrid approach can in some cases be more efficient than other 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) [20, 1], which augment CSPs with a valuation structure and subsume 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 can occur only in VCSPs, but not in CSPs: namely that the valuations are not distributed evenly across the space of assignments, but there rather exist large sets of assignments that have equal valuation (corresponding to "plateaus" in the search space).

Our approach exploits this by factoring optimization problems into a set of 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. A special instance of such a re-formulation is taking the dual of the problem [14], which yields a factorization into hard constraints and unary soft constraints.

The benefit of this re-formulation is that it allows to apply optimization techniques to the optimization part, and to apply satisfiability techniques to the satisfiability part. In particular, if the soft constraint part is small enough, it becomes feasible to use optimization techniques such as A* search [10], which is optimal in the number of search nodes visited, but would be infeasible to apply on the complete, original problem due to its memory requirements. For the hard constraint part, it becomes possible to use state-of-the-art the techniques for CSPs that exploit local consistency and conflicts.

This principled idea has been underlying algorithmic approaches in the area of model-based reasoning and diagnosis [24, 9] for quite some time. Model-based reasoning captures the behavior of physical systems in terms of constraint-based models, where a (typically small) subset of variables capture preferences (such as the failure probability of components, or the cost of repairing them), and constraints capture consistency. [25] formally defines these problems as so-called *optimal CSPs* and presents an algorithm called *conflict-directed A** that solves them using a mixture of optimization and satisfaction techniques. We generalize upon these methods, and by coupling them with a method for transforming valued CSPs into optimal CSPs, we extend their applicability to the general case of soft constraints. Our resulting hybrid 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. This can be more efficient than enumerating individual elements of the search space, because depending on the problem, there can be much fewer plateaus than total elements of the search space.

The remaining parts of the paper are organized as follows: We review the definitions of valued CSPs [20] and optimal CSPs [25] and present a method for transforming between them. The method is similar to dualization [14] in that it 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 sometimes outperforms other methods for solving valued CSPs, and we indicate several 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 $D = \{d_1, \ldots, d_n\}$, and constraints $C = \{c_1, \ldots, c_m\}$. Each constraint $c_j \in C$ is a relation $c_j \subseteq \Pi_{x_i \in \mathrm{var}(c_j)} d_i$ 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 [20]).** *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 [20]).** *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* $\phi$ *from* $C$ *to* $E$ *which associates a valuation with each constraint.*

For example, the problem of diagnosing the polycell circuit in Fig. 1 [25] can be framed as a VCSP with variables $X = \{a, b, c, d, e, f, g, x, y, z\}$. Each variable corresponds to 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 the gates in the circuit, and four unary constraints $f_c$, $f_d$, $f_f$, $f_g$ corresponding to the observations. The ternary constraints express that the gates are performing their boolean functions. The unary constraints express that the variables $c$, $d$, and $g$ are observed to be 1, whereas 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 $f_{o1}$, $f_{o2}$, $f_{o3}$, 2 for the constraints $f_{a1}$ and $f_{a2}$, and $\infty$ 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 [25], this type of optimization problem is called *optimal CSP*:

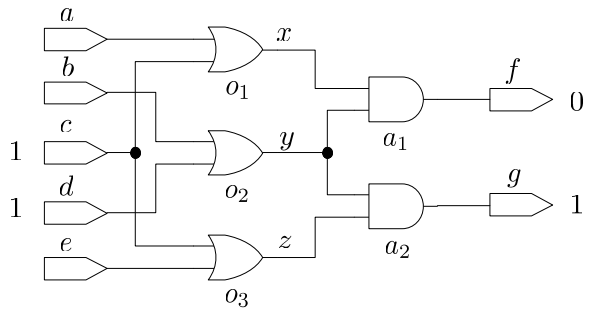


**Fig. 1.** The boolean polycell example consists of three OR gates and two AND gates. Variables $c$, $d$, $f$, and $g$ are observed as indicated.

**Definition 3 (Optimal CSP).** *An* optimal CSP *(OCSP) consists of a classical CSP* $(X, D, C)$*, with valuation structure* $(E, \leq, \oplus, \bot, \top)$*, and a set* $U$ *of unary functions* $u_j : y_j \to E$ *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 valuation such that there exists an extension to all variables $X$ that satisfies all constraints in the CSP. Hence, whereas a solution to a VCSP is a single assignments to $X$, a solution to an OCSP is an assignments 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 [19].

## 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 viewing the constraints of the VCSP as decision variables of the OCSP, similar to the *hidden variable representation* described [14]. 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 the desired separation into a hard constraint part and a soft constraint part.

**Definition 4 (Translation of VCSP to OCSP).** *The translation of a VCSP* $(X, D, C)$ *with valuation structure* $(E, \leq, \oplus, \bot, \top)$ *and mapping* $\phi$ *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* $c_j \in C$*;*
- $D'$ *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* $\bot$ *and the value* false *to* $\phi(c_j)$*;*
- $C'$ *consists of one constraint* $c'_j$ *for each* $c_j \in C$*. Each* $c'_j$ *is a relation over variables* $var(c'_j) = \mathrm{var}(c_j) \cup y_j$*. An assignment* $t$ *to* $var(c'_j)$ *satisfies* $c'_j$ *if and only if* $t[\mathrm{var}(c_j)] \in c_j$ *and* $y_j =$ true *or* $t[\mathrm{var}(c_j)] \notin c_j$ *and* $y_j =$ false*.*

For example, the translation of the VCSP for the boolean polycell circuit yields an OCSP with variables $\{a, b, c, d, e, f, g, x, y, z, y_1, y_2, \ldots, y_9\}$. Variables $y_1$ to $y_9$ are decision variables, and variables $\{a, b, c, d, e, f, g, x, y, z\}$ are non-decision variables. There are nine unary functions $u_1$, $u_2$, …, $u_9 \in U$, and nine constraints $f_{o1}$, $f_{o2}$, $f_{o3}$, $f_{a1}$, $f_{a2}$, $f_c$, $f_d$, $f_f$, $f_g$ obtained by extending each constraint of the original VCSP with a decision variable.

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

The transformation as described in Def. 4 turns a VCSP with $n$ variables and $m$ constraints into an OCSP with $n + m$ variables and $2 \cdot m$ constraints. 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 5 (Reduced translation of VCSP to OCSP).** *A reduced translation of a VCSP* $(X, D, C)$ *with valuation structure* $(E, \leq, \oplus, \bot, \top)$ *and mapping* $\phi$ *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* $\phi(c_j) < \top$*;*
- $D'$ *and* $U$ *are as in Def. 4;*
- $C'$ *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 defined as in Def. 4.*

The equivalence of optimal solutions (Theorem 1) will also be preserved by the translation in Def. 5. 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.

For the boolean polycell example, the translation using Def. 5 no longer introduces a decision variable for the hard constraints $f_c$, $f_d$, $f_f$, $f_g$ corresponding to observations, and thus the resulting OCSP has only five decision variables $y_1$, $y_2$, …, $y_5$ corresponding to the constraints $f_{o1}$, $f_{o2}$, $f_{o3}$, $f_{a1}$, $f_{a2}$.

## 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 [25] describe such a hybrid approach for solving a subclass of OCSPs. The approach, called *conflict-directed A**, uses backtracking search with arc consistency and conflict-directed backjumping [5] on the hard constraints, and A* search [10] on the unary soft constraints. Conflict-directed backjumping is an instance of learning new constraints from inconsistencies that can be very effective for real-world constraint satisfaction problems. A* search is 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). Unfortunately, due to its memory requirements, A* search is hardly feasible as a solution method for general VCSPs. As observed in [25], however, the memory requirements of A* search on OCSPs are often much more modest, because only assignments to variables that have an associated cost (decision variables) need to be stored in the search queue, and conflicts from the CSP part can be exploited to 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. 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 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 $y_i \leftarrow \text{true}$ and $y_i \leftarrow \text{false}$. 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 $X \setminus Y$. 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.

Conflict-directed A* is thus a hybrid algorithm for OCSPs that 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 the assignments to the decision variables (corresponding to plateaus) in best-first order, and then checks the consistency of these assignment (corresponding to the plateau being empty or not). 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.

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

For instance, for the boolean polycell example and the OCSP encoding in Def. 5, 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 $\langle y_1 \leftarrow \text{true}, \text{y}_1 \leftarrow \text{false}\rangle$ with valuation 1 are created and enqueued. When establishing the best assignment $\langle y_1 \leftarrow \text{true}, y_2 \leftarrow \text{true}\rangle$ us-

**Algorithm 1** Conflict-directed A* for OCSPs

```
 1: if not (propagate() = conflict) then
 2:    queue ← ⟨∅, ⊥⟩
 3:    while queue ≠ ∅ do
 4:       ⟨a, value⟩ ← top(queue)
 5:       queue ← pop(queue)
 6:       if switchAssignment(a) then
 7:          updateAssignment(⟨a, value⟩)
 8:          if assignment with better value exists in queue then
 9:             queue ← push(queue, ⟨a, value⟩)
10:          else
11:             if exists y_i ∈ Y, y_i = unknown then
12:                queue ← push(queue, ⟨a ∪ (y_i ← true), v⟩)
13:                queue ← push(queue, ⟨a ∪ (y_i ← false), v ⊕ φ(c_i)⟩)
14:             else
15:                if consistentAssignment() then
16:                   output value as best solution
17:                   exit
18:                end if
19:             end if
20:          end if
21:       end if
22:    end while
23: end if
24: output no solution
```

ing switchAssignment(), propagation forces $y_3$ to be false, and thus updateAssignment() refines the assignment to $\langle y_1 \leftarrow \text{true}, y_2 \leftarrow \text{true}, y_3 \leftarrow \text{false}\rangle$ 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 $\langle y_1 \leftarrow \text{false}, y_3 \leftarrow \text{true}\rangle$ with valuation 1. The algorithm proceeds by assigning $y_2 \leftarrow \text{true}$ and $y_4 \leftarrow \text{true}$, at which point $y_5 \leftarrow \text{true}$ 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.

Conflict-directed A* search can be further refined in a number of ways. [25, 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

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

**for all** $y_i \in Y$, $y_i \notin a$, $y_i \neq$ unknown **do**
  **if** $y_i =$ **true then**
    $\langle a, \text{value} \rangle \leftarrow \langle a \cup (y_i \leftarrow \textbf{true}), \text{value} \rangle$
  **else**
    $\langle a, \text{value} \rangle \leftarrow \langle a \cup (y_i \leftarrow \textbf{false}), \text{value} \oplus \phi(c_i) \rangle$
  **end if**
**end for**

**Procedure 3** switchAssignment($a$)

level $\leftarrow$ deepest level up to which $a$ and current assignment are equal
backtrack(level)
**for** $(y_i \leftarrow \text{val}) \in a$ **do**
  **if** $y_i \neq$ val **then**
    **return false**
  **else if** $y_i =$ unknown **then**
    $y_i \leftarrow$ val
    level $\leftarrow$ level $+ 1$
    **if** propagate() = conflict **then**
      CSP $\leftarrow$ CSP $\cup$ conflict
      **return false**
    **end if**
  **end if**
**end for**
**return true**

**Procedure 4** consistentAssignment()

decisionLevel $\leftarrow$ level
**while** exists $x_i \in X \setminus Y$, $x_i =$ unknown **do**
  **choose** val $\in d_i$
  $x_i \leftarrow$ val
  level $\leftarrow$ level $+ 1$
  $d_i \leftarrow d_i -$ val
  **if** propagate() = conflict **then**
    backtrackLevel $\leftarrow$ analyze(conflict)
    **if** backtrackLevel $\leq$ decisionLevel **then**
      **return false**
    **else**
      CSP $\leftarrow$ CSP $\cup$ conflict
      backtrack(backtrackLevel)
      level $\leftarrow$ backtrackLevel
    **end if**
  **end if**
**end while**
**return true**

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.

## 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 [23, 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 [22]. 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 [25, 15]) will have a strong impact for these examples.

## 8 Discussion and Related Work

In [14], Larrosa and Dechter already observed that transforming soft constraints into sets of hard and unary soft constraints may provide a useful starting point for algorithmic development. Conflict-directed A* is an instance of such an approach; it ties together two algorithms specialized to optimization and satisfaction (A* search and conflict-directed backjumping). The approach is inspired by techniques from model-based reasoning and diagnosis [24, 9], where problems can be naturally framed as a mixture of large hard constraints and unary objective functions (i.e., OCSPs).

The transformation of a VCSP into an OCSP makes this hybrid approach applicable to soft constraints. It can be viewed as a process of "pre-compiling" the objective function, which makes the preferences more explicit and can thus make the problem easier to solve. From this perspective, 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 drawback of our re-formulation technique is that it can 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 ratio up to which the 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 CSPs) could be reduced by incremental techniques that allow for computing only the difference between two CSP instances. In model-based reasoning and diagnosis, truth maintenance systems (TMS) [13], which keep track of the dependencies in the implication graph, are frequently used 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 [18], whereas [21] 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 [26]. 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 an algorithm that exploits this re-formulation by solving the optimization and satisfiability part separately using a combination of two 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, to appear.
[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, to appear.
[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] Sachenbacher, M., and Williams, B.C.: On-demand Bound Computation for Best-First Constraint Optimization, Proc. CP-2004 (2004)
[19] Sandholm, T.: An algorithm for optimal winner determination in combinatorial auctions. Proceedings IJCAI-1999 (1999)
[20] Schiex, T., Fargier, H., Verfaillie, G.: Valued Constraint Satisfaction Problems: hard and easy problems. Proc. IJCAI-95 (1995) 631–637
[21] Terrioux, C., Jégou, P.: Bounded Backtracking for the Valued Constraint Satisfaction Problems. Proc. CP-2003 (2003)
[22] TOOLBAR `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro`
[23] Walsh, T.: SAT vs. CSP. Proc. CP-2000 (2000) 441-456
[24] Weld, D.S., and de Kleer, J. (eds.): Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann (1989)
[25] Williams, B., Ragno, R.: Conflict-directed A* and its Role in Model-based Embedded Systems. Journal of Discrete Applied Mathematics, to appear.
[26] Xing, Z., and Zhang, W.: MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability. Artificial Intelligence **164** (1–2) (2005) 47-80