On-demand Bound Computation for Best-First Constraint Optimization

Martin Sachenbacher and Brian C. Williams

MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139, USA {sachenba,williams}@mit.edu

Abstract. An important class of algorithms for constraint optimization searches for solutions guided by a heuristic evaluation function (bound). When only a few best solutions are required, significant effort can be wasted pre-computing bounds that are not used during the search. We introduce a method that generates—based on lazy, best-first variants of constraint projection and combination operators—only those bounds that are specifically required in order to generate a next best solution.

1 Introduction

Many problems in Artificial Intelligence can be framed as constraint optimization problems where only a few best ("leading") solutions are needed. For instance, in fault diagnosis it might be sufficient to compute the most likely diagnoses that cover most of the probability density space [6]. In planning, it might be sufficient to compute the best plan, and a few backup plans in case the best plan fails. When only a few leading solutions need to be generated, algorithms that search through the space of possible assignments guided by a pre-computed heuristic evaluation function (bound) [4] become inefficient, because only some of the bounds will typically be needed to compute the best solutions. We present a method called *best-first search with on-demand bound computation* (BFOB) that efficiently computes leading solutions for semiring-based CSPs [1]. BFOB optimally interleaves bound computation and best-first search, such that bounds are computed and assignments are expanded only as required to generate each next best solution. The algorithm can still generate all solutions and its complexity is never worse than performing bound computation as a separate pre-processing step, yet it can derive the best solutions much faster. The approach involves lazy, best-first variants of constraint combination and projection operators, and a streamed computation scheme that coordinates these operators by exploiting a tree decomposition [3, 5] of the optimization problem.

2 Semiring-based Constraint Optimization Problems

A constraint optimization problem (COP) over a c-semiring $S = (A, +, \times, \mathbf{0}, \mathbf{1})$ [1] is a triple (X, D, F) where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D = \{D_1, \ldots, D_n\}$ is a set of finite domains, and $F = \{f_1, \ldots, f_m\}$ is a set of constraints.

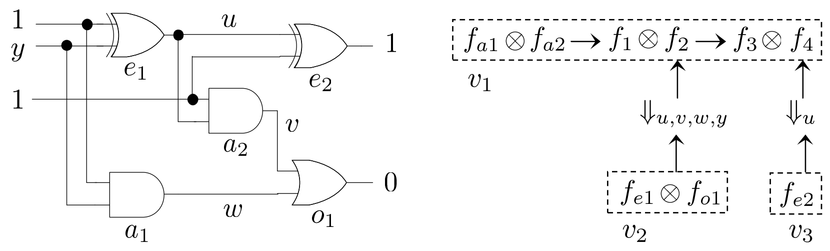

Fig. 1. Full adder example (*left*), consisting of two AND gates, one OR gate and two XOR gates, and computational scheme (*right*) for a tree decomposition of the example.

The constraints $f_j \in F$ are functions defined over $\mathrm{var}(f_j)$ assigning to each tuple a value in A. The $+$ operation of the c-semiring induces an order $\leq_S$ over A as follows: $a \leq_S b$ iff $a + b = b$. In this paper, we assume that $\leq_S$ is a total order.

Example. Diagnosis of the full adder circuit in Fig. 1 can be framed as a COP over the probabilistic c-semiring $S_p = ([0,1], \max, \cdot, 0, 1)$ with variables $X = \{u, v, w, y, a_1, a_2, e_1, e_2, o_1\}$. Variables u to y have domain $\{0,1\}$. Variables a_1 to o_1 describe the mode of a component and have domain {G,B}. If a component is good (G) then it correctly performs its boolean function; if it is broken (B) then no assumption is made about its behavior. Assume AND gates have a 1% probability of failure, and OR and XOR gates have a 5% probability of failure. Table 1 shows the resulting constraints, where each tuple is assigned the probability of its corresponding mode.

3 Optimization using Bound-Guided Search

Solutions to a COP can be found by searching through the space of possible assignments in best first order, guided by a heuristic evaluation function [4]. In A* search, the evaluation function is $f = g \times h$, composed of the value of the partial assignment made so far, g, and a heuristic h that provides an optimistic estimate (bound) on the value that can be achieved when completing the assignment. Kask and Dechter [4] show how h can be derived from a decomposition

Table 1. Constraints for the example (tuples with value **0** are not shown).

f_{a1}:

a1	w	y	
G	0	0	.99
G	1	1	.99
B	0	0	.01
B	0	1	.01
B	1	0	.01
B	1	1	.01

f_{a2}:

a2	u	v	
G	0	0	.99
G	1	1	.99
B	0	0	.01
B	0	1	.01
B	1	0	.01
B	1	1	.01

f_{e1}:

e1	u	y	
G	1	0	.95
G	0	1	.95
B	0	0	.05
B	0	1	.05
B	1	0	.05
B	1	1	.05

f_{e2}:

e2	u	
G	0	.95
B	0	.05
B	1	.05

f_{o1}:

o1	v	w	
G	0	0	.95
B	0	0	.05
B	0	1	.05
B	1	0	.05
B	1	1	.05

of the constraint network into an acyclic instance called a bucket tree [5]. The tree is evaluated bottom-up (that is, in post-order) using dynamic programming to compute a constraint h_{v_i} for each tree node v_i. Let function $g^{(i)}$ be defined as the combination of all functions of the nodes $v_1, \ldots, v_i$ of the bucket tree, and let function $h^{(i)}$ be defined as the combination of all functions of the nodes $c_1, \ldots, c_l$ that are children of $v_1, \ldots, v_i$:

$$g^{(i)} = \bigotimes_{j=1}^{i} (\bigotimes_{f_k \in v_j} f_k), \quad h^{(i)} = \bigotimes_{j=1}^{l} h_{c_j}.$$

Then the value $g^{(i)}(x_1^0, \ldots, x_i^0) \times (h^{(i)} \Downarrow_{x_{i+1}})(x_{i+1}^0)$ is an upper bound (with respect to $\leq_S$) on the value that can be achieved when extending the assignment $x_1 = x_1^0, \ldots, x_i = x_i^0$ by $x_{i+1} = x_{i+1}^0$. We generalize the derivation of bounds from bucket trees to tree decompositions [3, 5] by assigning variables in groups: Let $p = v_1, \ldots, v_n$ be a pre-order of the nodes V of a tree decomposition. Then p defines an ordering on groups of variables $G_1, \ldots, G_{|V|} \subseteq X$, by letting $G_1 = \text{var}(h_{v_1})$, $G_{i+1} = \text{var}(h_{v_{i+1}}) \setminus (G_1 \cup \ldots \cup G_i)$. Consider the tree decomposition in Fig. 1, consisting of three nodes v_1, v_2, and v_3. If the variables in the group $G_1 = \{u, v, w, y, a_1, a_2\}$ have been assigned (that is, node v_1 has been traversed), then $g^{(1)} \otimes h^{(1)}$ with $g^{(1)} = f_{a1} \otimes f_{a2}$ and $h^{(1)} = h_{v_2} \otimes h_{v_3}$ is a bounding function for the value that can be achieved when extending the assignment by assigning the variables in the group $G_2 = \{e_1, o_1\}$.

4 On-Demand Bound Computation

When only a few best solutions are required, computing bounds for all assignments is wasteful, since typically a large percentage of the bounds is not needed in order to compute the best solutions. The key to capturing this intuition formally is the following monotonicity property of c-semirings, which is an instance of preferential independence [2]:

Proposition 1. *If $h_0 \leq_S h_1$ for $h_0, h_1 \in A$, then for $g_0 \in A$, $g_0 \times h_0 \leq_S g_0 \times h_1$.*

It implies that in best-first search, it is sufficient to consider only the expansion with the best value (and keeping a reference to its next best sibling). This is sufficient because all other expansions cannot lead to solutions that have a better value with respect to the order $\leq_S$. The constraint-based A* scheme in [7] exploits this principle in order to significantly limit the successor nodes created at each expansion step. The idea pursued in this paper is to generalize on this: if it is unnecessary to create all possible expansions of a node, then it is also unnecessary to compute bounds on all possible expansions of a node. This allows us to interleave best-first search and the computation of a bounding function h, such that h is computed only to an extent that it is actually needed in order to generate a next best solution. We call this approach *best-first search with on-demand bound computation* (BFOB).

```
function nextBestComb($f1, f2$)
  while (queue $\neq \emptyset$) do
    $\langle i, j, v\rangle \leftarrow$ pop(queue)
    $\langle t1, v1\rangle \leftarrow$ at($f1, i$)
    if ($\langle t1, v1\rangle \neq$ nil) then
      $\langle t2, v2\rangle \leftarrow$ at($f2, j$)
      if ($\langle t2, v2\rangle \neq$ nil) then
        $t \leftarrow t1 \otimes t2$
        if (var($f1$) $\not\supseteq$ var($f2$)) then
          $\langle t1', v1'\rangle \leftarrow$ at($f1, i$+1)
          if ($\langle t1', v1'\rangle \neq$ nil) then
            push(queue,$\langle i$+1$, j, v1' \times v2\rangle$)
          end if
        end if
        if ($i = 1$) then
          $\langle t2', v2'\rangle \leftarrow$ at($f2, j$+1)
          if ($\langle t2', v2'\rangle \neq$ nil) then
            push(queue,$\langle i, j$+1$, v1 \times v2'\rangle$)
          end if
        end if
        if ($t \neq$ nil) then return $\langle t, v1 \times v2\rangle$
        end if
      end if
    end if
  end while
  return nil
```

```
function nextBestProj($f$)
  while (index $\neq 0$) do
    $\langle t, v\rangle \leftarrow$ at($f$, index)
    if ($\langle t, v\rangle \neq$ nil) then
      $t1 \leftarrow t \Downarrow_{\mathrm{var}(f_{\mathrm{result}})}$
      index $\leftarrow$ index + 1
      for each $\langle t2, v2\rangle$ in $f_{\mathrm{result}}$ do
        if ($t1 = t2$) then goto while
        end if
      end for
      return $\langle t1, v\rangle$
    else
      index $\leftarrow 0$
    end if
  end while
  return nil
```

Fig. 2. Best-first variants of constraint combination and constraint projection.

The approach requires to compute the functions h_{v_i} incrementally and in best-first order. We achieve this—akin to streamed, on-demand computation in distributed databases—by applying constraint projection and combination operations only partially, that is, to subsets of the tuples of the constraints. Consider the scheme of functions and operations shown in Fig. 1. The best tuple of function f_{e2} is $\langle e_2{=}\mathrm{G}, u{=}0\rangle$ with value .95 (first tuple of f_{e2} in Table 1). The projection of this tuple on u, which is $\langle u{=}0\rangle$ with value .95, is necessarily a best tuple of f_4. Similarly, a best tuple of f_{a1} can be combined with a best tuple of f_{a2}, for instance the first tuples of f_{a1} and f_{a2} in Table 1. The resulting tuple $\langle u{=}0, v{=}0, w{=}0, y{=}0, a_1{=}\mathrm{G}, a_2{=}\mathrm{G}\rangle$ with value .98 is necessarily a best tuple of constraint f_1. Eventually, a best tuple for h_{v_1} can be computed from the scheme without visiting large parts of the constraints f_{a1}, f_{a2}, f_{e1}, f_{e2}, and f_{o1}.

The functions nextBestProj() and nextBestComb() shown in Fig. 2 implement such best-first variants of the constraint operators $\Downarrow$ and $\otimes$, respectively. The helper function at(f, i) returns the i-th best tuple of a constraint f, or generates it, if necessary, by calling the constraint operator producing f. For each projection operator, index is initially 1, and for each combination operator,

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

T	C	N	K	BFPB (% time)	BFOB (% time)
4 (25%)	20	15	4	100%	1.4%
8 (50%)	20	15	4	100%	3.2%
4 (25%)	15	10	4	100%	4.5%
8 (50%)	15	10	4	100%	14.3%
4 (25%)	20	10	4	100%	9.7%
8 (50%)	20	10	4	100%	38.8%

queue is initially $\{\langle 1, 1, \mathbf{1} \rangle\}$. Initially, the tuples of the constraints are sorted and inserted at the inputs (leafs) of the scheme. BFOB then assigns variables in groups following a pre-order traversal of the tree as described in Sec. 3, and when expanding a search node, it computes a bound on-demand for the best expansion using function at(). The best-first variants of the constraint operators have the same worst-case complexity as their counterparts $\Downarrow$ and $\otimes$. However, the average complexity of on-demand function computation can be much lower if only some best tuples of the resulting function are required.

5 Experimental Results

We compared the performance of BFOB *relative* to the alternative approach of pre-computing all functions h_{v_i}. We call this alternative algorithm BFPB (it is analogous to the algorithm BFMB described in [4]). Table 2 shows the results of experiments with three classes of Max-CSP problems for the task of generating a single best solution. The results indicate that BFOB leads to significant savings especially when computing best solutions to problems with low constraint tightness and sparse to medium network density.

References

[1] Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based Constraint Solving and Optimization. Journal of ACM, **44** (2) (1997) 201–236

[2] Debreu, C.: Topological methods in cardinal utility theory. In: Mathematical Methods in the Social Sciences, Stanford University Press (1959)

[3] Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence **124** (2) (2000) 243–282

[4] Kask, K., Dechter, R.: A General Scheme for Automatic Generation of Search Heuristics from Specification Dependencies. Artificial Intelligence **129** (2001) 91–131

[5] Kask, K., et al.: Unifying Tree-Decomposition Schemes for Automated Reasoning. Technical Report, University of California, Irvine (2001)

[6] de Kleer, J.: Focusing on Probable Diagnoses. Proc. AAAI-91 (1991) 842–848

[7] Williams, B., Ragno, R.: Conflict-directed A* and its Role in Model-based Embedded Systems. Journal of Discrete Applied Mathematics, to appear.