A failed attempt to optimize variable ordering with tools for Constraints Solving

Edmund M. Clarke and Ofer Strichman *

Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA. Email: {emc,ofers}@cs.cmu.edu

Abstract. The 'minimal cutwidth' and 'minimal lifespan' are known hypergraph problems with strong relevance to variable ordering for both SAT and BDDs. We describe a formalization of these problems as CSP. Our experiments indicate that realistic instances of these problems can not be solved with CSP solvers.

1 Introduction

This note summarizes our attempt to solve the NP complete *min cut-width* (MCW) and *min life-span* (MLS) problems with a general-purpose constraint solver. Both of these problems are relevant for finding good variable ordering for SAT and BDDs. While modeling these problem as Constraint Solving Problems (CSP) is easy, our experiments show that even state-of-the-art CSP tools can not solve non-trivial instances, let alone realistic examples. We conclude that the currently available tools for finding approximate solutions are the only option for solving realistic instances.

2 The minimal cut-width problem

2.1 Problem definition

We begin with defining several notations. Let $G(V, E)$, $|V| = n$, be a hypergraph, and let $\mathcal{O} : v_1, \ldots, v_n$ be a strict order on V. We denote by $v_i < v_j$ the fact that v_i precedes v_j in the given order (we omit the order to which this relation refers to for conciseness of notation. This will be clear from the context). The notation $v_i \leq v_j$ is interpreted as $i = j$ or $v_i < v_j$.

* This research was sponsored by the Semiconductor Research Corporation (SRC) under contract no. 99-TJ-684, the National Science Foundation (NSF) under grant no. CCR-9803774, the Army Research Office (ARO) under grant DAAD19-01-1-0485, the Office of Naval Research (ONR), and the Naval Research Laboratory (NRL) under contract no. N00014-01-1-0796. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of SRC, ARO, NSF, ONR, NRL, the U.S. government or any other entity.

Definition 1. *An edge* $e \in E$ crosses *variable* v_i *in a given order* $\mathcal{O}$ *iff* $\exists v_j, v_k, j \neq k.\ v_j, v_k \in e \wedge v_j \leq v_i \leq v_k$.

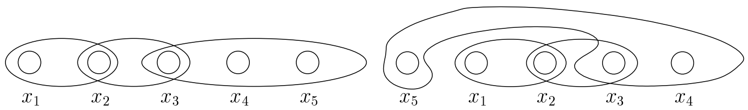
In other words, the edge e crosses v_i if e contains variables both to the left and to the right of v_i in the given order. We use this definition in order to define the cutwidth of G at point (variable) v_i:

Definition 2. *The cut-width of* $G(V, E)$ *at point* v_i *with respect to the variable order* $\mathcal{O}$, $1 \leq i \leq n$, *is the number of edges in* E *that cross* v_i.

The cutwidth of G is the maximal cutwidth among all points $v_1 \ldots v_n$. The minimal cutwidth problem is to find a variable order such that the cutwidth of G is minimal.

The following example demonstrates how an ordering affects the cutwidth of a graph.

Example 1. Let $V = x_1, \ldots, x_5$ and let $E = \{(x_1, x_2), (x_2, x_3), (x_3, x_4, x_5)\}$. The figure below corresponds to the two variable orderings $\mathcal{O}_1 : x_1, x_2, x_3, x_4, x_5$ (left) and $\mathcal{O}_2 : x_5, x_1, x_2, x_3, x_4$ (right). $\mathcal{O}_1$ imposes a cutwidth of 2, while $\mathcal{O}_2$ imposes a cutwidth of 3 (due to x_2).



(a) (b)

□

We now define a construction of a hypergraph $G(V, E)$ from a CNF formula φ. The set of vertices V correspond to the set of variables in φ. Each clause in φ corresponds to one hyperedge in E (note that the phase of each literal is ignored in this construction).

2.2 The relevance of cutwidth to SAT and BDD

The relevance of MCW to both BDD variable ordering and SAT solving with static variable ordering was described, for example, in [3] and in [6]. Two theoretical results establish the connection between the cutwidth and the complexity of solving a given formula: Berman[1] established that under certain conditions on the variable order, the BDD size of a circuit is bounded by $n * 2^c$ where n is the number of inputs and c is the cutwidth. In [7] a cutwidth-based SAT algorithm is given which guarantees a time complexity bounded by $n * 2^c$, where in this case n is the number of clauses in the CNF representation of the formula.

A more intuitive explanation of why ordering that results in smaller cutwidth is expected to result in a smaller BDD and a faster SAT procedure, is the following. The relevance for BDD ordering stems for the fact that an order that minimizes the cutwidth tends to group together variables that mutually affect the result of the formula. The latter is a known criterion for good variable ordering. For example, in the formula $\varphi : x_1 = x_2 \wedge x_3 = x_4$ the variable order

$\mathcal{O}_1 : x_1, x_2, x_3, x_4$ results in a smaller BDD than $\mathcal{O}_2 : x_1, x_3, x_2, x_4$. The order $\mathcal{O}_1$ also results in a smaller cutwidth of G than the one obtained with $\mathcal{O}_2$.

The relevance of MCW to SAT is of the same nature: deciding on related variables close to each other minimizes the amount of required backtracking. For example, let (x_1, x_2, x_3) be a clause in φ, and assume that this clause can only be satisfied in φ if $x_1 = 1$. Also, assume that the SAT solver assigned 0 to x_1 and that the decision order is x_1, <other n variables>, $x_2, x_3, \ldots$. This implies that the wrong assignment to x_1 will potentially be revealed only when deciding x_2, which is only after n other variables are decided. In this case the procedure will backtrack to the beginning, loosing all the work done in between. On the other hand, if the variables were close to each other, the wrong assignment would be discovered immediately.

In the next section we formulate the MCW problem as a CSP.

3 Minimal cutwidth as a CSP

We use the variables listed in Fig. 1 to formulate the CSP corresponding to the MCW problem.

Variable	Domain	Range	Description
n			Number of variables.
cl			Number of clauses.
l_c^i		$1 \le c \le cl$, $1 \le i \le \|c\|$	The i-th literal in clause c.
$x_{i,j}$	$\{0, 1\}$	$1 \le i, j \le n$	Indicates whether variable j is located in location i.
$w_k^{c,i}$	$\{-1, 1\}$	$1 \le k \le n$ $1 \le c \le cl$ $1 \le i \le \|c\|$	Indicates whether literal i of clause c is positioned to the left (-1) or to the right (1) of position k.
r_k^c	$\{0, 1\}$	$1 \le k \le n$ $1 \le c \le cl$	Indicates whether clause c crosses location k.
$CutWidth$			The largest no. of clauses crossing any location.

Fig. 1. Variables used in the CSP formulation. The top 3 variables are inputs. The 'Domain' column indicates the possible values that the computed variables can have.

Our objective is to minimize the cut-width:

$$\text{Min} \;\; CutWidth \tag{1}$$

Subject to the following constraints:

Every variable is positioned in one location:

$$\forall_j \sum_{i=1}^{n} x_{i,j} = 1 \tag{2}$$

Every location is assigned one variable:

$$\forall_i \sum_{j=1}^{n} x_{i,j} = 1 \tag{3}$$

$w_k^{c,i} \in \{-1, 1\}$ indicates whether literal i of clause c is positioned in the left (-1) or right (1) to position k:

$$\forall_{k=1}^{n} \forall_{c=1}^{cl} \forall_{i=1}^{|c|} \; w_k^{c,i} = \sum_{j=k+1}^{n} x_{j,l_c^i} - \sum_{j=1}^{k-1} x_{j,l_c^i} \tag{4}$$

$r_k^c \in \{0, 1\}$ indicates whether clause c crosses location k:

$$\forall_{k=1}^{n} \forall_{c=1}^{cl} \; r_k^c = |\sum_{i=1}^{|c|} w_k^{c,i}| \neq |c| \tag{5}$$

$CutWidth$ is the largest no. of clauses crossing any location:

$$\forall_{k=1}^{n} \sum_{c=1}^{cl} r_k^c \leq CutWidth \tag{6}$$

All together this formulation results in $(n \cdot (l + cl + 3))$ constraints, where l is the number of literals in the formula.

3.1 A tighter upper bound

The time complexity bound $n * 2^C$ on solving a SAT instance, as described in [7], is not tight. It considers the cutwidth of the graph, which is the worst cutwidth at all points. A more refined upper-bound is one that considers the cutwidth in all points rather than the worst one. This results in the bound $\sum_{i=1}^{n} 2^{c_i}$. In order to compute the new bound we define the variable cw_k, $1 \leq k \leq n$ as the cut-width at point k:

$$cw_k = \sum_{c=1}^{cl} r_k^c \tag{7}$$

We now alter the objective function:

$$Min \sum_{k=1}^{n} 2^{cw_k} \tag{8}$$

4 Minimizing total life-span

The life span of a clause c in a given order $\mathcal{O} : x_1 ... x_n$ is the largest distance between two literals in c in this order. The total life span of a CNF formula with respect to a given variable order is the sum of all life-spans of all clauses. It was observed in [3] that the total life span of G is equal to the total cutwidth. This is the reason that the two measures are defined in very similar ways.

To define the total life-span we add the equality

$$LifeSpan = \sum_{k=1}^{n} \sum_{c=1}^{cl} r_k^c \tag{9}$$

and change the objective function accordingly:

$$\text{Min} \quad LifeSpan \tag{10}$$

The criterion of minimum life-span was lately used for finding a good variable quantification order when performing image computations with BDD's [5].

5 Experiments

As indicated in the introduction, our experiments shows that CSP tools cannot solve realistic instances of either MCW or MLS. We used OPL-studio to model these problems, and tried to solve them with the built-in CSP constraint solver 'Solver'[1]. As far as we know Solver is considered to be one of the strongest of its kind. We ran it with several examples, but unfortunately it could not solve anything beyond toy examples. For example, it could not solve after 24 hours a 2-CNF instance with 50 clauses and 51 variables. The table below summarizes some experiments we made.

CNF variables	CNF clauses	Solver constraints	Solver variables	Memory (MB)	Time (sec.)
7	5	126	155	0.35	0.3
14	12	546	701	0.72	100.1
26	21	1716	2315	2.5	*
51	50	7803	10252	16.4	*

In a second attempt, the authors of B-Prolog, (a modeling and solving environment for finite-domain constraints) tried to solve some of these instances with their tool and failed as well.

For comparison, realistic CNF instances used, for example, in bounded model checking, can have hundreds of thousands of clauses and the number of variables

[1] We ran Solver with its default configuration.

is in the same order of magnitude. The tool used in [3], called CAPPO[2], and the tool used in [7], called hMetis [4] uses various heuristics for computing a good, yet not optimal ordering. From these references we learn that these tools are capable of handling instances with tens of thousands of variables in a short amount of time.

We hope that the importance of solving the minimal cut-width problem will lead the CSP community to consider it as a standard benchmark, and that special optimizations, tailored for this problem, will eventually be found.

References

1. C.L. Berman. Circuit width, register allocation, and ordered binary decision diagrams. *IEEE Transactions on Computer-Aided Design*, 1991.
2. A. Caldwell, A. Kahng, and I. Markov. Can recursive bisection produce routable placements? In *Proc. Design Automation Conference(DAC)*, 2000.
3. Karem A. Sakallah Fadi A. Aloul, Igor L. Markov. Faster sat and smaller bdds via common function structure. In *Proc. of the international conference on Computer Aided Design (ICCAD)*, 2001.
4. G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: applications in vlsi design. In *Proc. Design Automation Conference(DAC)*, 1997.
5. In-Ho Moon and Fabio Somenzi. Border-block triangular form and conjunction schedule in image computation. In *Proc. Formal Methods in Computer Aided Design (FMCAD)*, 2000.
6. M. Prasad, P. Chong, and K. Keutzer. Why is atpg easy? In *Proceedings of the 36 th Design Automation Conference DAC99*, pages 22–28, 1999.
7. Dong Wang, Edmund Clarke, Yunshan Zhu, and Jim Kukula. Using cutwidth to improve symbolic simulation and boolean satisfiability. In *HLDVT*, 2001.