Propagating Separable Equalities in an MDD Store

T. Hadzic[1], J.N. Hooker[2], and P. Tiedemann[3]

[1] University College Cork
t.hadzic@4c.ucc.ie
[2] Carnegie Mellon University
john@hooker.tepper.cmu.edu
[3] IT University of Copenhagen
petert@itu.dk

Abstract. We present a propagator that achieves MDD consistency for a separable equality over an MDD (multivalued decision diagram) store in pseudo-polynomial time. We integrate the propagator into a constraint solver based on an MDD store introduced in [1]. Our experiments show that the new propagator provides substantial computational advantage over propagation of two inequality constraints, and that the advantage increases when the maximum width of the MDD store increases.

In [1] we proposed a width-limited *multivalued decision diagram* (MDD) as a general constraint store for constraint programming. We demonstrated the potential of MDD-based constraint solving by developing MDD-propagators for *alldiff* and *inequality* constraints. In this paper, we describe an MDD-propagator for the *separable equality* constraint that uses a pseudo-polynomial algorithm to achieve *MDD consistency*. We show the computational advantage of the new propagator over the existing approach of modeling equalities with two inequality propagators.

Preliminaries. A *constraint satisfaction problem* is specified with a set of constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$ on the variables $X = \{x_1, \ldots, x_n\}$ with respective finite domains $D_1, \ldots, D_n$. An MDD M is a tuple (V, r, E, var, D), where V is a set of vertices containing the special terminal vertex 1 and a root $r \in V$, $E \subseteq V \times V$ is a set of edges such that (V, E) forms a directed acyclic graph with r as the source and 1 as the sink for all maximal paths in the graph. Further, $var : V \rightarrow \{1, \ldots, n+1\}$ is a labeling of all nodes with a variable index, with $var(1) = n + 1$. D is a set containing an *edge domain* D_{uv} for each edge (u, v). We require that $\emptyset \neq D_{uv} \subseteq D_{var(u)}$ for all edges in E, and for convenience we take $D_{uv} = \emptyset$ if $(u, v) \notin E$. We work only with *ordered* MDDs. A total ordering $<$ of the variables is assumed, and all edges (u, v) respect the ordering; that is, $var(u) < var(v)$. For convenience, we assume that the variables in X are ordered according to their indices. Ordered MDDs can be viewed as arranged in n *layers* of vertices, with the vertices on each layer labeled with the same variable index. The *width* k of the MDD is the size of the largest layer. While MDDs

L. Perron and M. Trick (Eds.): CPAIOR 2008, LNCS 5015, pp. 318–322, 2008.

in general allow edges to skip layers, for the simplicity of representation in this paper we consider only MDDs without long edges; that is, for each $(u, v) \in E$, $var(v) = var(u) + 1$. Thus, if an $r \to 1$ *path* is defined to be a path $u_1, \ldots, u_{n+1}$ in which $u_1 = r$ and $u_{n+1} = 1$, then each $r \to 1$ path *represents* the subset of solutions $\prod_{i=1}^{n}(D_{u_i u_{i+1}})$. Let C be a constraint on the variables $\{x_1, \ldots, x_n\}$. For a given MDD M we have a notion of consistency that generalizes the well known *generalized arc consistency* (GAC) [2].

Definition 1 (MDD consistency). *A constraint C is* MDD consistent *with respect to M if, for every edge $(u, v) \in E$ with $i = var(u)$ and every value $\alpha_i \in D_{uv}$, there exists a tuple $(\alpha_1, \ldots, \alpha_n)$ satisfying C that is represented by an $r \to 1$ path passing through (u, v).*

1 A Propagator for the *Separable Equality* constraint

Unlike a standard domain-store propagator, which is specified only by the way it *prunes* infeasible values from a domain-store, an MDD-store propagator also *refines* the MDD representing the store by adding new vertices and edges. We develop such a propagator for the *separable equality* constraint, which for a set of unary functions $f_1, \ldots, f_n$ and a constant c is defined as

$$f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n) = c. \tag{1}$$

1.1 Pruning

One simple way to perform pruning on the constraint (1) is to do so for the two inequality constraints $\sum_{i=1}^{n} f_i(x_i) \leq c$ and $\sum_{i=1}^{n} f_i(x_i) \geq c$. We can achieve MDD consistency in linear time in the size of the MDD for each of these separately, using the inequality propagator described in [1]. Yet this ensures only that each remaining edge is on a shortest path with cost at most c and on a longest path with cost at least c. It therefore does not achieve MDD consistency for the equality constraint.

To achieve MDD consistency we use the following procedure. In the first phase, for each node u the algorithm computes the cost $L_{down}(u)$ of the cheapest path and the cost $H_{down}(u)$ of the most expensive path leading from u to the terminal. In the second phase it marks the edges in the MDD store that are on at least one $r \to 1$ path representing a solution of the constraint. In the final phase all unmarked edges are removed from the MDD.

The pseudo-code for the algorithm Mark-Support implements the second phase with a dynamic programming recursion and is shown in Figure 1. It is initially invoked on the root r and the right-hand side c. When invoked on a node u it searches for a path through the MDD from u with the given cost. For each edge (u, u') and $\alpha \in D_{u,u'}$, the algorithm recursively checks if there exists a path of cost $c - f_{var(u)}(\alpha)$ from u' to the terminal, and the result of this query is cached as $cache(u', c - f_{var(u)}(\alpha))$. If the result is positive the edge is marked to indicate that it must not be pruned. For each node u the previously computed

$L_{down}(u)$ and $H_{down}(u)$ values provide an early cutoff, because there can be no path of cost c from u if the cheapest path from u is too expensive or the most expensive path is too cheap. Note that if the width of the MDD store is 1 then the algorithm is essentially the domain store filter of [3].

```
MARK-SUPPORT(u, cost)
 1  if cache(u, cost) ≠ UNKNOWN
 2     then return cache(u, cost)
 3     else if L_down(u) > cost ∨ H_down(u) < cost
 4             then return false
 5     else cache(u, cost) ← false
 6          for (u, u') ∈ E and α ∈ D_{u,u'}
 7              do if MARK-SUPPORT(u', cost − f_{var(u)}(α))
 8                    then marked ← marked ∪ (u, u', α)
 9                         cache(u, cost) ← true
10          return cache(u, cost)
```

Fig. 1. The algorithm shown above, initially invoked as MARK-SUPPORT(r, c) for a constraint $\sum_{1 \leq i \leq n} f_i(x_i) = c$, ensures that an edge along with a value from its edge domain is in *marked* iff there is a path through that edge, using the corresponding value with cost exactly c

Complexity. The complexity of a propagation step is dominated by the execution of MARK-SUPPORT as the other phases can be done in linear time. Since each call to MARK-SUPPORT only does constant work in addition to the recursive calls, we can evaluate the time based on the number of recursive calls alone. A call to MARK-SUPPORT on a node u only results in recursive calls if the given cost has not been processed before for u. Let $L_{up}(u)$ and $H_{up}(u)$ be the cost of the cheapest and most expensive path from the root to u. Then an upper bounds on the number $q(u)$ of distinct costs that any node u will be queried for is $q(u) = c - L_{up}(u) - (c - H_{up}(u)) + 1 = H_{up}(u) - L_{up}(u) + 1$. Hence the total number of recursive calls over all nodes can be bounded by

$$\sum_{u \in V} q(u)|D_{var(u)}| = O\left(q(1) \max_{x_i \in X}\{|D_i|\}|V|\right) = O\left(q(1) \max_{x_i \in X}\{|D_i|\}nk\right).$$

Note that this bound increases linearly with the width k. This is very pessimistic, however, as a larger width makes the store a more precise approximation that allows fewer candidate solutions. This results in fewer paths to a given node, and therefore in most cases fewer distinct costs of these paths, which translates into fewer recursive calls per node. Thus, a larger width can *decrease* the time required to execute MARK-SUPPORT. Additionally, a more refined store will allow more edges to be pruned. Hence a larger width could be expected to reduce the overall solution time. We verify this behavior in our empirical results.

1.2 Refining

An MDD propagator refines the MDD through *node splitting* [1]. We first select a node $u \in V$ and create an isomorphic copy $u' \in V$ by copying every outgoing edge (u, t) into (u', t) along with all edge labels $D_{u',t} = D_{u,t}$. We then copy ingoing edges (s, u) into (s, u') along with a *subset* $D_{s,u'} \subseteq D_{s,u}$ of edge labels that are then removed from the original edges: $D_{s,u} \leftarrow D_{s,u} \setminus D_{s,u'}$. Fig. 2 shows an example of a node split and subsequent propagation for an *alldiff* constraint.

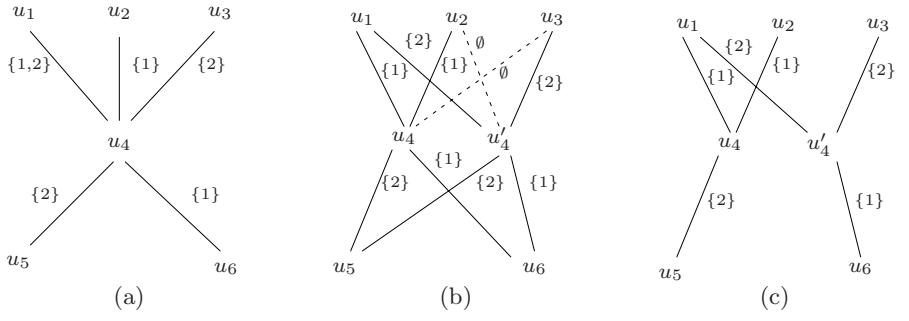

Fig. 2. (a) Part of an MDD store representing a relaxation of a global alldiff, just before splitting on the node u_4. Note that while there are obvious inconsistencies between the edge domains (such as label 1 in domains of (u_1, u_4) and (u_4, u_6)), we cannot remove any value. (b) A new node u_4' has been created and some of the edge domain values to u_4 have been transferred to u_4'. There are no labels on (u_2, u_4') and (u_3, u_4), so the edges need not be created. (c) After the split we can prune inconsistent values and as a result remove edges (u_4, u_6) and (u_4', u_5).

Our splitting strategy selects a splitting node and a subset of incoming edges to be redirected by heuristically estimating the *quality* of the resulting split. For an *equality* constraint we try to increase the potential for subsequent pruning by maximizing the shortest path $L(u')$ and minimizing the longest path $H(u')$ passing through u'. In particular, we try to minimize the expected difference $H(u') - L(u')$. This splitting strategy is used with both the pseudo-polynomial pruning and pruning based on two inequalities.

2 Empirical Results

We randomly generated a number of problem instances involving 3 separable equalities over 15 variables with a domain of size 3. We measured the time necessary to find a solution using two inequality propagators and our equality propagator. For each x_i and $v \in D_i$, we randomly selected $f_i(v) \in [-10000, 10000]$. The results are shown in Figure 3. We can observe that enforcing the stronger MDD consistency through an equality constraint consistently outperforms the weaker consistency enforced by two inequalities. The computation time for two inequalities increases with larger width, while reducing for the new equality propagator.

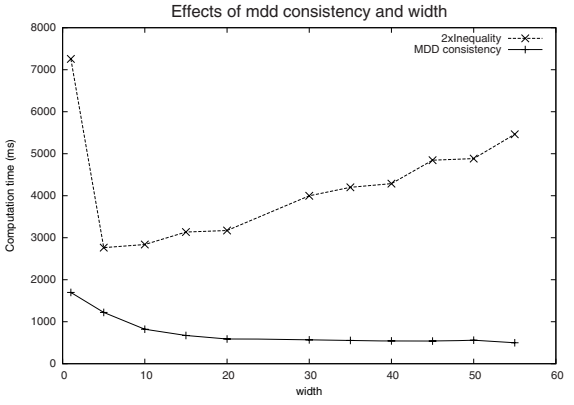

Fig. 3. The effect of MDD width (horizontal axis) on computation time (in ms, vertical axis) when using (a) two inequality propagators to propagate the equality constraint and (b) the MDD consistent equality propagator introduced in this paper

3 Conclusions and Future Work

We presented a propagator for the MDD store that achieves MDD consistency for a separable equality constraint in pseudo-polynomial time. From our empirical results we observed that the extra overhead is worthwhile in practice. In particular, the benefit increases as the width of the MDD store increases. In future work we intend to develop an approximate propagation scheme based on caching for small ranges of cost instead of a single precise cost.

References

1. Andersen, H.R., Hadzic, T., Hooker, J.N., Tiedemann, P.: A constraint store based on multivalued decision diagrams. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 118–132. Springer, Heidelberg (2007)
2. van Hoeve, W.J., Katriel, I.: Global constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming. Foundations of Artificial Intelligence, pp. 169–208. Elsevier Science Publishers, Amsterdam (2006)
3. Trick, M.: A dynamic programming approach for consistency and propagation for knapsack constraints. In: Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-01), pp. 113–124 (2001)