Job Sequencing Bounds from Decision Diagrams

J. N. Hooker

Carnegie Mellon University
June 2017

Abstract. In recent research, decision diagrams have proved useful for the solution of discrete optimization problems. Their success relies on the use of relaxed decision diagrams to obtain bounds on the optimal value, either through a node merger or a node splitting mechanism. We investigate the potential of node merger to provide bounds for dynamic programming models that do not otherwise have a practical relaxation, in particular the job sequencing problem with time windows and state-dependent processing times. We prove general conditions under which a node merger operation yields a valid relaxation and apply them to job sequencing. Computational experiments show that, surprisingly, relaxed diagrams prove the optimal value when their size is only a small fraction of the size of an exact diagram. On the other hand, a relaxed diagram of fixed size ceases to provide a useful bound as the instances scale up.

1 Introduction

Decision diagrams have historically been used for circuit design and verification [1, 12, 22, 23] and a variety of other purposes [24, 27]. Recent research indicates that decision diagrams provide an alternative approach to discrete optimization and constraint solving [2, 9, 14]. Of special interest to optimization is the fact that decision diagrams are well suited to dynamic programming (DP) formulations, because the diagrams are essentially state transition graphs. Viewing DP from the perspective of decision diagrams opens the door to new techniques for solving DP problems, such as branch-and-bound methods [9].

A key element of this development is the use of *relaxed* decision diagrams to derive a bound on the optimal value. Optimization bounds are useful not only in branch-and-bound methods, but for assessing the quality of solutions obtained by heuristics, and perhaps for proving their optimality. Because the exact (non-relaxed) diagram tends to grow exponentially with the number of variables, it is vital to find relaxed diagrams of limited size.

Two methods for constructing relaxed decision diagrams of limited size were introduced in [2]: *node merger* and *node splitting*. Node merger reduces the size of the diagram by introducing some infeasible solutions. Node splitting works in the opposite direction: by beginning with a diagram that represents all possible solutions and creating new nodes to exclude some infeasible solutions.

We investigate here the potential for *node merger* to relax DP formulations, particularly those for which good relaxations are not currently available. While

a great advantage of DP is that it does not presuppose convexity, linearity, or even a closed-form description of the problem, this very generality often makes it difficult to find a good relaxation. The problem may also be difficult to solve due to the exponential growth of the state space, known as the "curse of dimensionality."

We focus on a job sequencing problem in which processing times are state-dependent. Due to this state dependency, the problem does not have a practical mixed-integer or other formulation for which a relaxation is readily available. We wish to determine whether relaxed decision diagrams of limited size can provide a useful bound and therefore assist in finding an optimal solution.

Node merger has not previously been applied to job sequencing problems, or apparently to any problems for which no effective relaxation is known. While various types of sequencing problems are solved by decision diagrams in [14], the relaxed diagrams are created by node splitting rather than merger. It is therefore important to investigate the potential of node merger as a relaxation technique for DP. Because the job sequencing problem studied here is a simple representative of a broad class of sequencing problems, the results could have wider implications.

After a brief review of previous work, we begin with a description of exact and relaxed decision diagrams and how they relate to DP, using the job sequencing problem as an example. We then prove general sufficient conditions under which node merger yields a valid relaxed diagram, since conditions of this kind have never appeared in the literature. They allow us to show that a proposed merger rule for the job sequencing problem produces a valid relaxation. We then describe alternative heuristics for selecting nodes to merge, because an effective merger heuristic is essential to obtaining tight bounds.

We report computational experiments that show the somewhat surprising result that bounds obtained from relaxed diagrams reach the optimal value rather quickly, when the diagrams are only a small fraction of the size of an exact diagram. This allows relaxed diagrams to prove the optimality of a heuristically obtained solution in much less time than would otherwise be necessary. On the other hand, relaxed diagrams of any given fixed size cease to prove a useful bound as the instances scale up. In the conclusion, we suggest a research direction that addresses this issue.

2 Previous Work

Decision diagrams were first proposed as an optimization method by [16, 20]. Other early applications to optimization include cut generation for integer programming [4], post-optimality analysis [15, 16], and vertex and facet enumeration [5]. The idea of a relaxed decision diagram was introduced by [2] for constraint programming. Subsequent applications are described by [6, 17–19]. Relaxed diagrams were used to obtain optimization bounds in [7, 11]. Branching within a relaxed diagram was introduced by [9]. Connections between decision diagrams and deterministic dynamic programming, including nonserial dynamic

programming, are discussed in [21]. A comprehensive survey of decision diagrams as an optimization technique appears in [8].

Relaxation of a decision diagram is superficially related to state space relaxation in dynamic programming [3, 13, 25, 26], but there are several fundamental differences. A relaxed decision diagram is created by splitting or merging nodes rather than mapping the state space into a smaller space. It can be tightened by filtering techniques. It is constructed dynamically as the diagram is built, rather than by defining a mapping a priori. It uses the same state variables as the exact formulation, which allows the relaxed diagram to be used as a branching framework for an exact branch-and-bound method. The relaxation can be calibrated to provide a bound of any desired quality, up to optimality, simply by adjusting the maximum width to be observed by the diagram while it is created.

3 Decision Diagrams

For our purposes, a decision diagram can be defined as a directed, acyclic multigraph in which the nodes are partitioned into *layers*. Each arc of the graph is directed from a node in layer j to a node in layer $j+1$ for some $j \in \{1, \dots, n\}$. Layers 1 and $n+1$ contain a single node, namely the root r and the terminus t, respectively. Each layer j is associated with a finite-domain variable $x_j \in D_j$. The arcs leaving any node in layer j have distinct *labels* in D_j, representing possible values of x_j at that node. A path from r to t defines an assignment to the tuple $x = (x_1, \dots, x_n)$ as indicated by the arc labels on the path. The decision diagram is *weighted* if there is a length (cost) associated with each arc.

Any discrete optimization problem with variables $x_1, \dots x_n$ and a separable objective function $\sum_j f_j(x_j)$ can be represented by a weighted decision diagram.[1] The diagram is constructed so that its r–t paths correspond to the feasible solutions of the problem, and the cost of an arc with label v_j leaving layer j is $f_j(v_j)$. The length (cost) of any r–t path is the objective function value of the corresponding solution. If the objective is to minimize, the optimal value is the length of a shortest r–t path.

Many different diagrams can represent the same problem, but for a given variable ordering, there is a unique *reduced* diagram that represents it [12, 21]. A diagram is reduced when for any pair of nodes u, u' in a given layer j, the set of u–t paths and their costs is different from the set of u'–t paths and their costs. That is, the two sets correspond to different sets of assignments to $x_j, \dots, x_n$ or different costs.

As an example, consider a small instance of the job sequencing problem with time windows (Table 1). The jobs must be sequenced so that each job j begins processing no earlier than the release time r_j and requires processing time p_j. We assume that for a given sequencing, the start time of job j is $s_j = \max\{r_j, s_i + p_i\}$, where i is the immediately preceding job in the sequence, and the first job

[1] Problems with nonseparable objective functions can also be represented, as described in [21], but to simplify exposition we omit this possibility.

Table 1. A small instance of a job scheduling problem.

j	r_j	p_j	d_j
1	0	3*	4
2	1	2	3
3	1	2	5

*2 when job 2 has previously been processed.

starts at its release time. The objective is to minimize total tardiness, where the tardiness of job j is $\max\{0, s_j + p_j - d_j\}$, and d_j is the job's due date.

We can make the processing time state-dependent by supposing that it depends on which jobs have already been processed. This frequently occurs in practice, as processing one job may involve the fabrication of parts that can be used when processing another job. In the example, we suppose that the processing time p_1 for job 1 is 2 (rather than 3) when job 2 has been processed.

Figure 1 shows a reduced decision diagram that represents the problem. Variable x_j represents the jth job in the sequence. Each arc indicates its label and immediate cost (the latter in parentheses). Each r–t path encodes a feasible schedule, and any shortest (minimum-cost) r–t path indicates an optimal solution of the problem.

When a problem is formulated recursively, a simple top-down compilation procedure yields a decision diagram that represents the problem. A general recursive formulation can be written

$$h_j(S_j) = \min_{x_j \in X_j(S_j)} \Big\{ c_j(S_j, x_j) + h_{j+1}\big(\phi_j(S_j, x_j)\big) \Big\} \tag{1}$$

Here, S_j is the *state* in stage j of the recursion, $X_j(S_j)$ is the set of possible *controls* (values of x_j) in state S_j, ϕ_j is the *transition function* in stage j, and $c_j(S_j, x_j)$ is the *immediate cost* of control x_j in state S_j. We assume there is single initial state S_1 and a single final state S_{n+1}, so that $h_{n+1}(S_{n+1}) = 0$ and

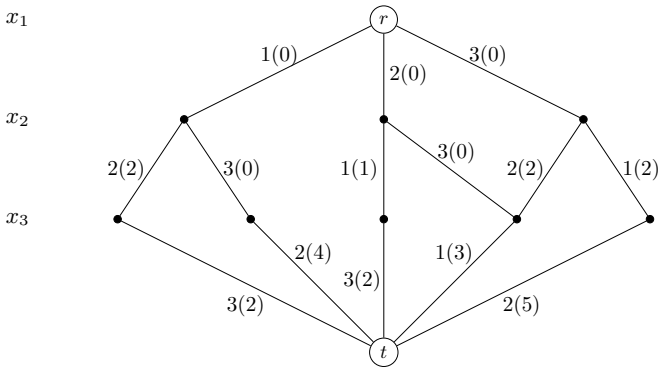

Fig. 1. Decision diagram for the job sequencing instance of Table 1.

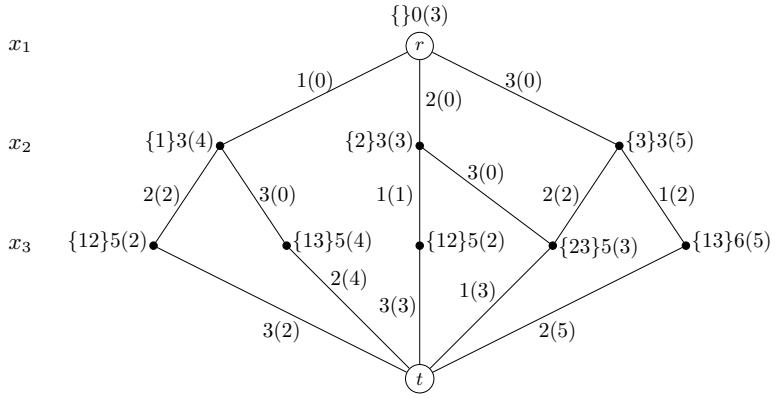

Fig. 2. Decision diagram, with states and minimum costs-to-go, for the job sequencing instance of Table 1.

$\phi_n(S_n, x_n) = S_{n+1}$ for all states S_n and controls $x_n \in X_n(S_n)$. The quantity $h_j(S_j)$ is the *cost-to-go* for state S_j in stage j, and an optimal solution has value $h_1(r)$.

In the job sequencing problem, the state S_j is the tuple (V_j, f_j), where V_j is the set of jobs scheduled so far, and f_j is the finish time of the last job scheduled. Thus the initial state is $r = (\emptyset, 0)$, and $X_j(S_j)$ is $\{1, \ldots, n\} \setminus V_j$. The transition function $\phi_j(S_j, x_j)$ is given by

$$\phi_j\big((V_j, f_j), x_j\big) = \big(V_j \cup \{x_j\},\ \max\{r_{x_j}, f_j\} + p_{x_j}(V_j)\big)$$

Note that the processing time $p_{x_j}(V_j)$ depends on the current state V_j as well as the control x_j. The immediate cost $c_j((V_j, f_j), x_j)$ is the tardiness that results from scheduling job x_j in state (V_j, f_j), namely $(\max\{r_{x_j}, f_j\} + p_{x_j}(V_j) - d_{x_j})^+$, where $\alpha^+ = \max\{0, \alpha\}$.

We recursively construct a decision diagram D for the problem by associating a state with each node of D. The initial state S_1 is associated with the root node t and the final state S_{n+1} with the terminal node t. If state S_j is associated with node u in layer j, then for each $v_j \in X_j(S)$ we generate an arc with label v_j leaving u. The arc terminates at a node associated with state $\phi_j(S, v_i)$. Nodes on a given layer are identified when they are associated with the same state.

The process is illustrated for the job sequencing example in Fig. 2. Each node is labeled by its state (V_j, f_i), followed (in parentheses) by the minimum cost-to-go at the node. The cost-to-go at the terminus t is zero.

4 Relaxed Decision Diagrams

A weighted decision diagram D' is a *relaxation* of diagram D when D' represents every solution in D with equal or smaller cost, and perhaps other solutions as well. To make this more precise, suppose layers $1, \ldots, n$ of both D and

D' correspond to variables $x_1, \dots, x_n$ with domains $X_1, \dots, X_n$. Then D' is a relaxation of D if every assignment to x represented by an r–t path P in D is represented by an r–t path in D' with length no greater than that of P. The shortest path length in D' is a lower bound on the optimal value of the problem represented by D. We will refer to a diagram that has not been relaxed as *exact*.

We can construct a relaxed decision diagram by top-down compilation, again based on the recursive model (1). The procedure is as before, except that rather than simply identify nodes in each layer that are associated with the same states, we may also *merge* some nodes. That is, we may identify some nodes that are associated with different states. The object is to keep the width of the diagram (the maximum number of nodes in a layer) within a predetermined bound W. When we merge nodes with states S and T, we associate a state $S \oplus T$ with the resulting node. The operator $\oplus$ is chosen so as to yield a valid relaxation of the given recursion.

It is frequently necessary to introduce additional state variables to define a suitable merger operation [9], and this is the case in the job sequencing example. The state at a node will consist of (V, U, f), where V and f are as before, and U contains the jobs that occur along some path from the root. The processing time $p_{x_j}(U)$ of a job x_j depends on the jobs in U. The transition function is

$$\phi_j\big((V, U, f), x_j\big) = \big(V \cup \{x_j\}, U \cup \{x_j\}, \max\{r_{x_j}, f\} + p_{x_j}(U)\big)$$

and the immediate cost is $c_j((V, U, f), x_j) = (\max\{r_{x_j}, f\} + p_{x_j}(U) - d_{x_j})^+$. Merging states (V, U, f) and (V', U', f') results in state $(V \cap V', U \cup U', \min\{f, f'\})$. We will see in the next section that this merger operation results in a valid relaxation.

The merger operation is illustrated in Fig. 3, which is the result of merging states $(\{1, 2\}, 5)$ and $(\{2, 3\}, 5)$ in layer 3 of Fig. 2. The expanded states (V, U, f) are shown at each node, followed by the minimum cost-to-go in parentheses. The shortest path now has cost 2, which is a lower bound on the optimal cost of 3 in Fig. 2.

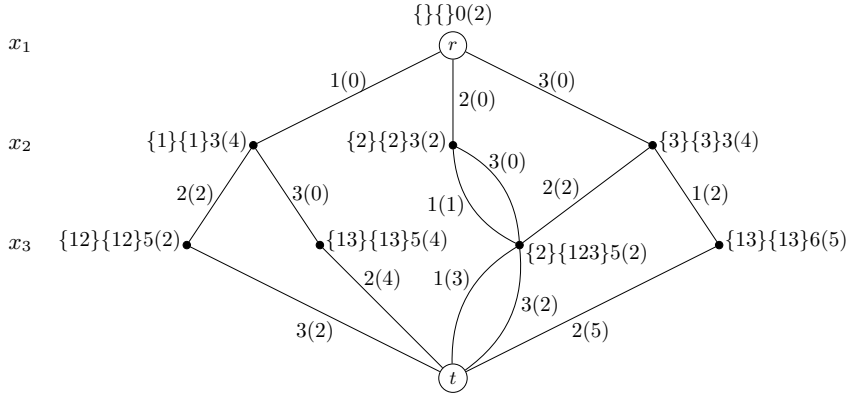

Fig. 3. A relaxation of the decision diagram in Fig. 2.

5 Conditions for Node Merger

We now develop general sufficient conditions under which node merger results in a relaxed decision diagram. Such conditions have apparently not be explicitly stated in the literature. It is shown in [19] that it suffices for the merged state to be a union of the states merged, but it is not useful to represent the merged state as a union of states. The merged state must be given in terms of the state variables in the states merged, so that the construction of the relaxed diagram can proceed with the same transition function and immediate cost function as at other nodes.

We will say that a state S' in layer j *relaxes* a state S in layer j when (a) all feasible controls in state S are feasible in state S', and (b) the immediate cost of any given feasible control in S is no less than its immediate cost in S'. That is, $X_j(S) \subseteq X_j(S')$, and $c_j(S, x_j) \geq c_j(S', x_j)$ for all $x_j \in X_j(S)$. Then node merger results in a valid relaxation when two conditions are satisfied. One is a condition on the transition function generally: when one state relaxes another, this must continue to hold when the same control is applied to both states. That is,

(C1) If state S' relaxes state S, then given any control v that is feasible in S, $\phi(S', v)$ relaxes $\phi(S, v)$.

The second condition places a requirement on the merger operation specifically. Namely, when two states are merged, the resulting state relaxes both of the states that are merged.

(C2) $S \oplus T$ relaxes both S and T.

We can now prove the relevant theorem. Let $c(P)$ be the cost (length) of path P in a decision diagram. It is also convenient to let D'_k be the first k layers of the relaxed diagram D' obtained during top-down compilation, but just before identifying and merging nodes in layer k. Thus $D'_{n+1} = D'$.

Theorem 1. *If conditions (C1) and (C2) are satisfied, the merger of nodes with states S and T within a diagram D results in a valid relaxation of D.*

Proof. Let D be the exact decision diagram that results from top-down compilation, and let D' be the diagram that results from top-down compilation with node merger. It suffices to show claim (H_k) inductively for $k = 1, \ldots, n+1$:

(H_k) Consider any path P from r to any u in layer k of D. Then D'_k contains a path P', from r to a node u' in layer k, that represents the same assignment to $(x_1, \ldots, x_{k-1})$. Furthermore $c(P) \geq c(P')$, and the state S' at u' relaxes the state S at u.

Claim (H_1) is trivially true. We therefore suppose (H_k) is true and show (H_{k+1}). Consider a path $\bar{P}$ from r to $\bar{u}$ in layer $k+1$ of D. We must show that D'_{k+1} contains a path $\bar{P}'$, from r to $\bar{u}'$, that represents the same assignment

to $(x_1, \ldots, x_k)$. Furthermore, we must show that $c(\bar{P}) \geq c(\bar{P}')$, and that the state $\bar{S}'$ at $\bar{u}'$ relaxes the state $\bar{S}$ at $\bar{u}$.

Let P be the portion of path $\bar{P}$ that extends from r to a node u in layer k of D. By the induction hypothesis (H_k), D'_k contains a path P', terminating at some node u' in layer k, that represents the same assignment to $(x_1, \ldots, x_{k-1})$. Now D'_{k+1} is formed by identifying and merging nodes in layer k of D'_k and generating arcs from the resulting nodes. Thus if S' is the state at u' in D'_k, S' is merged with zero or more other states to form a state T. If v is a control that extends P to $\bar{P}$, then v is likewise a feasible control in state T. This is because T relaxes S' by condition (C2), and S' relaxes S by (H_k), which implies that T relaxes S. Now we can let $\bar{P}'$ be the path in D'_{k+1} that results from extending P' at state T with control v. To see that $c(\bar{P}) \geq c(\bar{P}')$, note that

$$c(\bar{P}) = c(P) + c_k(S, v) \geq c(P') + c_k(S', v) \geq c(P') + c_k(T, v) = c(\bar{P}')$$

where the first inequality is due to (H_k), and the second inequality is due to condition (C2). To show that $\bar{S}'$ relaxes $\bar{S}$, we note again that T relaxes S. This and condition (C1) imply that $\bar{S}'$ relaxes $\bar{S}$, as desired. □

The merger operation defined earlier for the job sequencing problem satisfies conditions (C1) and (C2). To see that (C1) is satisfied, suppose (V', U', f') relaxes (V, U, F), which means that $V' \subseteq V$, $U' \supseteq U$, and $f' \leq f$. Then if control v is applied to either state, we have $V' \cup \{v\} \subseteq V \cup \{v\}$ and $U' \cup \{v\} \supseteq U \cup \{v\}$. Also

$$\min\{r_v, f'\} + p_v(U') \leq \min\{r_v, f\} + p_v(U)$$

because $f' \leq f$ and $p_v(U') \leq p_v(U)$, the latter due to the fact that $U' \supseteq U$. So $\phi_j((V', U', f'), v)$ relaxes $\phi_j((V, U, f), v)$, and (C1) follows. To show (C2), recall that (V, U, f) and (V', U', f') are merged to form $(V \cap V', U \cup U', \min\{f, f'\})$. The merger relaxes the two states that are merged because $V \cap V' \subseteq V, V'$, $U \cup U' \supseteq U, U'$, and $\min\{f, f'\} \leq f, f'$.

6 Merging Heuristics

We now address the question of which nodes to merge in a given layer so as to reduce the width to W. The merger strategy should be designed so that the diagram remains exact, to the extent possible, along paths that are likely to be optimal. Merging nodes in the remainder of the diagram will not affect the optimal solution and therefore the bound. The merger strategy is particularly important when processing times are state-dependent, because the additional state variable U results in smaller values for the state variable f and therefore shorter paths in the diagram, yielding a weaker bound.

We therefore need some indication of whether a given node is likely to lie on a shortest path. The most readily available indication is the state variable f, since it represents the finish time of the most recent job processed. If f is large, then the cost of a path through the node is more likely to be large. We

can merge two nodes with the largest values of f and repeat the process until the width is reduced to W. We will refer to this as the *finish time heuristic*.

Another possibility is to compute the shortest path to a given node from the root. If the shortest path is already long, it is likely to be still longer by the time it reaches the terminus. We therefore merge nodes to which the shortest path from the root is longest. We will refer to this as the *shortest path heuristic*.

In the next section, we compare the effectiveness of these heuristic against a control heuristic that consists of randomly selecting nodes to merge.

7 Computational Experiments

The aim of the computational experiments is to determine how the quality of the bound depends on the width of the relaxed decision diagram. To our knowledge, there are no benchmark sets for sequence-dependent processing times, and so we ran tests on randomly generated problem instances. A meaningful assessment of the bound quality requires that we be able to solve the instances exactly, because the optimal values vary widely from zero to a rather large number. We must therefore generate instances that are small enough to be solved exactly in reasonable time.

Dynamic programming is the only viable method for exact solution of problems with general state dependence, including problems with sequence-dependent processing times. Since the state space becomes impracticably large for instances with more than 14 jobs, we generated and solved instances with 12 and 14 jobs. We found that the results are quite consistent over these instances, which suggests that the pattern may continue for larger instances.

Instances are generated as follows. The normal processing time p_j is drawn uniformly from the interval $[p_{\min}, p_{\max}]$. To make the processing time state dependent, we reduce it to $p_j/2$ when j is even and job $j-1$ has already been processed. The release time is drawn uniformly from $[0, np_{min}/2]$, where n is the number of jobs. The due date is $d_j = r_j + p_j + \text{slack}_j$, where slack_j is drawn uniformly from $[kp_{min}, kp_{\max}]$. We used $k = 4$ for 12 jobs and $k = 5$ for 14 jobs to obtain minimum tardiness values that are generally positive but not unrealistically large. We also set $[p_{\min}, p_{\max}] = [10, 16]$. The first 5 random instances are used for each problem size, after discarding those with zero minimum tardiness, because they provide no information about the quality of the bound.

Figures 4 and 5 show the results, using the finish time heuristic. The bound is plotted against the width of the relaxed diagram, where the latter is on a logarithmic scale. The bounds are sampled at 10 points per factor of 10 for 12 jobs, and 20 points per factor of 10 for 14 jobs. The far right end of each curve represents an exact decision diagram for the instance, which is equivalent to the state transition graph for the dynamic programming formulation of the problem. Selected computation times are shown on the curves. The computation time for a given width is very similar across all instances of a given size. The computation

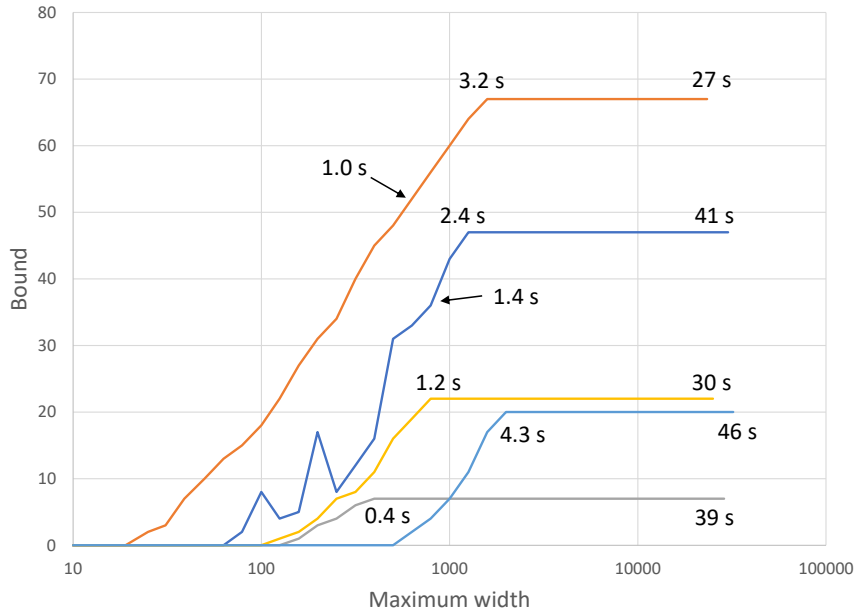

Fig. 4. 12-job instances: relaxation bound versus decision diagram width, up to the width of an exact diagram. Selected computation times are shown.

time for the exact diagram indicates the time necessary to solve the problem by dynamic programming.

The curves follow a different pattern than those reported for other types of problems, which tend to approach the optimal value asympotically [7]. Once the curves begin to rise above zero, they increase rapidly and level off at the optimal value when the width is less than one-tenth the width of an exact diagram. Specifically, when there are 12 jobs, the optimal value is achieved for diagrams that are between 1/32 and 1/15 the width of an exact diagram, and when there are 14 jobs, between 1/26 and 1/10 the width of an exact diagram. On the other hand, the bound does not rise above zero until the width of the relaxed diagram is roughly 1/1000 to 1/25 the width of an exact diagram. This indicates that diagrams of a fixed maximum width, such as 1000 or 10000, cease to provide useful bounds as the instances scale up. The curves are not always monotonic because the the bound depends on the merging heuristic, which may happen to perform better for a smaller width than a slightly larger one.

Figure 6 compares the performance of three merging heuristics for one of the 12-job instances, namely the finish time, shortest path, and random selection heuristics. The choice of heuristic is clearly key, as the finish time heuristic is vastly superior to the others. Examination of the shortest paths from the root at various nodes reveals why the shortest path heuristic fails. The shortest path length tends to remain at zero in the upper layers of the diagram, which means there is no guidance for node merger, resulting in bad merger decisions that

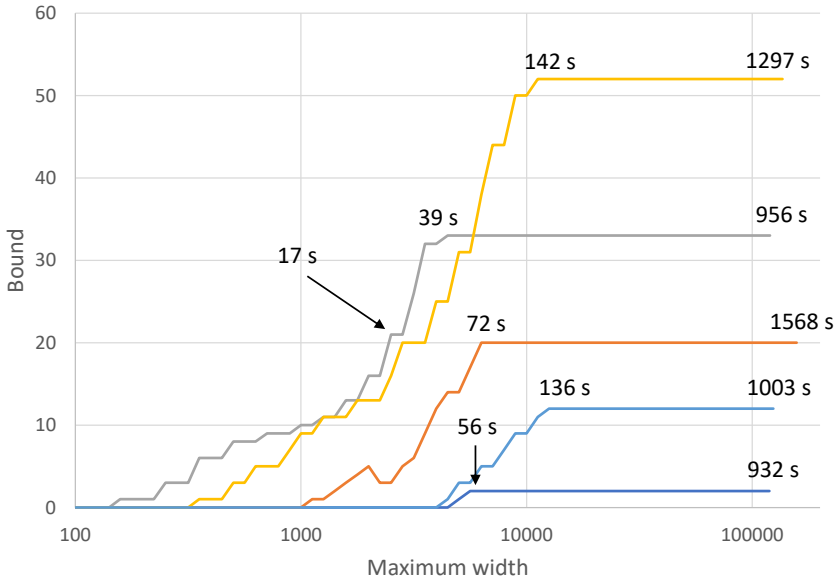

Fig. 5. 14-job instances: relaxation bound versus decision diagram width, up to the width of an exact diagram. Selected computation times are shown.

propagate through the remainder of the diagram. Random merger is even worse, resulting in no useful bounds until the diagram is nearly exact.

8 Conclusions

We undertook a preliminary investigation of the potential of node merger as a relaxation mechanism for dynamic programming problems, particularly those for which no practical relaxation method exists. We focused on the job sequencing problem with time windows and state-dependent processing times, because it has no known mixed integer programming or other model that yields a useful relaxation.

We first proved two conditions that are jointly sufficient for a node merger operation to yield a valid relaxation, one a condition on the transition function of the dynamic programming model, and one a condition on the merger operation itself. We then formulated a merger rule for the job sequencing problem and used the conditions to show that it results in a relaxed diagram.

Computational testing revealed that relaxed diagrams for this problem have different characteristics than have been reported for other types of problems. The relaxed diagram yields the optimal value when its width is a small fraction of the width of an exact diagram. This allows a relaxed diagram to prove the

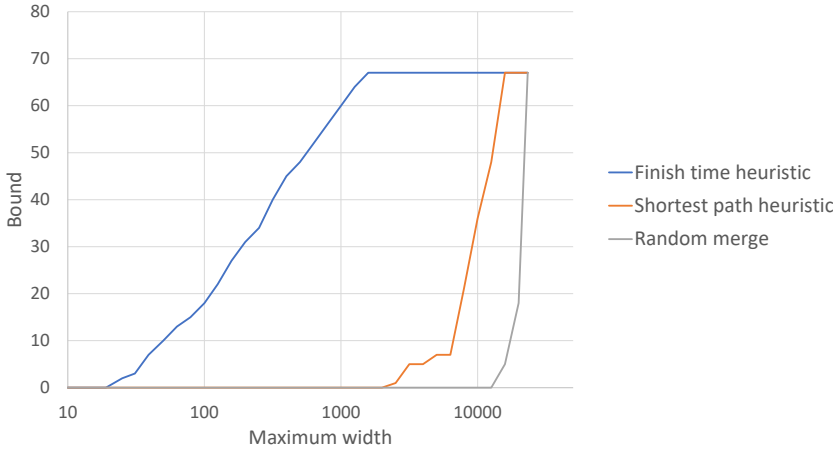

Fig. 6. Comparison of bound quality of three node merging heuristics on a 12-job instance.

optimality of a solution obtained heuristically, using much less computation than would otherwise be necessary.

On the other hand, diagrams of a fixed maximum width cease to provide useful bounds as the instances scale up. A intriguing line of research would be to use Lagrangian methods to strengthen the bounds provided by smaller diagrams. These methods adjust the arc costs in the relaxed diagram to exclude poor solutions while retaining a valid bound [10]. They have been used successfully in other contexts and could prove a valuable enhancement of node merger for the relaxation of dynamic programming models.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Transactions on Computers C-27, 509–516 (1978)
2. Andersen, H.R., Hadžić, T., Hooker, J.N., Tiedemann, P.: A constraint store based on multivalued decision diagrams. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming (CP 2007). LNCS, vol. 4741, pp. 118–132. Springer (2007)
3. Baldacci, R., Mingozzi, A., Roberti, R.: New state-space relaxations for solving the traveling salesman problem with time windows. INFORMS Journal on Computing 24(3), 356–371 (Jul 2012)
4. Becker, B., Behle, M., Eisenbrand, F., Wimmer, R.: BDDs in a branch and cut framework. In: Nikoletseas, S. (ed.) Experimental and Efficient Algorithms, Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA 05). LNCS, vol. 3503, pp. 452–463. Springer (2005)
5. Behle, M., Eisenbrand, F.: 0/1 vertex and facet enumeration with BDDs. In: Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX). pp. 158–165. SIAM (2007)

6. Bergman, D., Ciré, A.A., van Hoeve, W.J., Hooker, J.N.: Variable ordering for the application of BDDs to the maximum independent set problem. In: Jussien, N., Petit, T. (eds.) CPAIOR Proeeedings. LNCS, vol. 7298, pp. 34–49. Springer (2012)
7. Bergman, D., Ciré, A.A., van Hoeve, W.J., Hooker, J.N.: Optimization bounds from binary decision diagrams. INFORMS Journal on Computing 26, 253–268 (2013)
8. Bergman, D., Ciré, A.A., van Hoeve, W.J., Hooker, J.N.: Decision Diagrams for Optimization. Springer (2016)
9. Bergman, D., Ciré, A.A., van Hoeve, W.J., Hooker, J.N.: Discrete optimization with binary decision diagrams. INFORMS Journal on Computing 28, 47–66 (2016)
10. Bergman, D., Ciré, A.A., van Hoeve, W.J.: Lagrangian bounds from decision diagrams. Constraints 20, 346–361 (2015)
11. Bergman, D., van Hoeve, W.J., Hooker, J.N.: Manipulating MDD relaxations for combinatorial optimization. In: Proceedings of CPAIOR. LNCS, vol. 6697, pp. 20–35 (2011)
12. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35, 677–691 (1986)
13. Christofides, N., Mingozzi, A., Toth, P.: State-space relaxation procedures for the computation of bounds to routing problems. Networks 11(2), 145–164 (1981)
14. Ciré, A.A., van Hoeve, W.J.: Multivalued decision diagrams for sequencing problems. Operations Research 61, 1411–1428 (2013)
15. Hadžić, T., Hooker, J.N.: Postoptimality analysis for integer programming using binary decision diagrams. Tech. rep., Carnegie Mellon University (2006)
16. Hadžić, T., Hooker, J.N.: Cost-bounded binary decision diagrams for 0-1 programming. In: Loute, E., Wolsey, L. (eds.) CPAIOR 2007 Proceedings. LNCS, vol. 4510, pp. 84–98. Springer (2007)
17. Hadžić, T., Hooker, J.N., O'Sullivan, B., Tiedemann, P.: Approximate compilation of constraints into multivalued decision diagrams. In: Stuckey, P.J. (ed.) Principles and Practice of Constraint Programming (CP 2008). LNCS, vol. 5202, pp. 448–462. Springer (2008)
18. Hadžić, T., Hooker, J.N., Tiedemann, P.: Propagating separable equalities in an MDD store. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008 Proceedings. Lecture Notes in Computer Science, vol. 5015, pp. 318–322. Springer (2008)
19. Hoda, S., van Hoeve, W.J., Hooker, J.N.: A systematic approach to MDD-based constraint programming. In: Cohen, D. (ed.) Principles and Practices of Constraint Programming (CP 2010). LNCS, vol. 6308, pp. 266–280. Springer (2010)
20. Hooker, J.N.: Discrete global optimization with binary decision diagrams. In: GICOLAG 2006. Vienna, Austria (December 2006)
21. Hooker, J.N.: Decision diagrams and dynamic programming. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013 Proceedings. LNCS, vol. 7874, pp. 94–110. Springer (2013)
22. Hu, A.J.: Techniques for efficient formal verification using binary decision diagrams. Thesis CS-TR-95-1561, Stanford University, Department of Computer Science (Dec 1995)
23. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell Systems Technical Journal 38, 985–999 (1959)
24. Loekito, E., Bailey, J., Pei, J.: A binary decision diagram based approach for mining frequent subsequences. Knowl. Inf. Syst 24(2), 235–268 (2010)
25. Mingozzi, A.: State space relaxation and search strategies in dynamic programming. In: Proceedings of Abstraction, Reformulation, and Approximation. Lecture Notes in Computer Science, vol. 2371, pp. 51–51. Springer (2002)

26. Righini, G., Salani, M.: New dynamic programming algorithms for the resource constrained shortest path problem. Networks 51, 155–170 (2008)
27. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. SIAM monographs on discrete mathematics and applications, Society for Industrial and Applied Mathematics (2000)