Improved Job Sequencing Bounds from Decision Diagrams

J. N. Hooker

Carnegie Mellon University
jh38@andrew.cmu.edu

Abstract. We introduce a general method for relaxing decision diagrams that allows one to bound job sequencing problems by solving a Lagrangian dual problem on a relaxed diagram. We also provide guidelines for identifying problems for which this approach can result in useful bounds. These same guidelines can be applied to bounding deterministic dynamic programming problems in general, since decision diagrams rely on DP formulations. Computational tests show that Lagrangian relaxation on a decision diagram can yield very tight bounds for certain classes of hard job sequencing problems. For example, it proves for the first time that the best known solutions for Biskup-Feldman instances are within a small fraction of 1% of the optimal value, and sometimes optimal.

Keywords: Job sequencing · Decision diagrams · Lagrangian relaxation.

1 Introduction

In recent years, binary and multivalued decision diagrams (DDs) have emerged as a useful tool for solving discrete optimization problems [5, 6, 24]. A key factor in their success has been the development of *relaxed* DDs, which represent a superset of the feasible solutions of a problem and provide a bound on its optimal value. While an exact DD representation of a problem tends to grow exponentially with the size of the problem instance, a relaxed DD can be much more compact when properly constructed. The tightness of the relaxation can be controlled by adjusting the maximum allowed width of the DD.

Relaxed DDs are normally used in conjunction with a branching procedure [5, 12], much as is the linear programming (LP) relaxation in an integer programming solver. As branching proceeds, the relaxed diagram provides a progressively tighter bound. However, combinatorial problems are often solved with heuristic methods that do not involve branching. This is true, in particular, of job sequencing problems. In such cases it is very useful to have an independently derived lower bound that can provide an indication of the quality of the solution.

Recent research [20] has found that a relaxed DD can yield good bounds for hard job sequencing problems without branching. In fact, a surprisingly small relaxed DD, generally less than 10% the width of an exact DD, can yield a bound equal to the optimal value. On the other hand, since exact DDs grow rapidly with the instance size, relaxed DDs that are 10% of their width likewise grow rapidly.

As a result, relaxed DDs of reasonable width tend to provide progressively weaker bounds as the instances scale up.

It is suggested in [20] that Lagrangian relaxation could help strengthen the bounds obtained from smaller relaxed DDs. In this paper, we propose a general technique for relaxing a DD while preserving the ability to obtain Lagrangian bounds from the DD. The relaxed DD is constructed by merging nodes only when they agree on certain state variables that are crucial to forming the Lagrangian relaxation.

We find that for certain types of job sequencing problems, Lagrangian relaxation in relaxed DDs of reasonable width can provide very tight bounds on the optimal value. For example, we prove for the first time that the best known solution values of Biskup-Feldman single-machine scheduling instances are within a small fraction of one percent of the optimum, and sometimes optimal.

Furthermore, we identify general conditions under which Lagrangian relaxation can be implemented in a relaxed DD for purposes of obtaining bounds. The conditions are expressed in terms of structural characteristics of the dynamic programming model that defines the DD. They lead to a new tool for bounding not only job sequencing problems with suitable structure, but general deterministic dynamic programming models that satisfy the conditions.

2 Previous Work

Decision diagrams were introduced as an optimization method by [15, 18]. The idea of a relaxed diagram first appears in [1] as a means of enhancing propagation in constraint programming. Relaxed DDs were first used to obtain optimization bounds in [4, 7]. Connections between DDs and deterministic dynamic programming are discussed in [19].

Bergman, Ciré and van Hoeve first applied Lagrangian relaxation to decision diagrams in [3], where they use it successfully to strengthen bounds for the traveling salesman problem with time windows. They also use Lagrangian relaxation and DDs in [2] to improve constraint propagation.

We advance beyond Bergman et al. [3] in two ways. First, we show how to obtain bounds on tardiness and a variety of other objective functions from a stand-alone relaxed DD. The DD in [3] represents only an all-different constraint and can provide bounds only on total travel time (without taking time windows into account). The DD is embedded in a constraint programming (CP) model that contains the time window constraints. While constraints could be added to the CP model to obtain tardiness and other kinds of bounds from the CP solver, the DD itself cannot provide them. One or more additional state variables are necessary, which results in a more complicated DD than the one used in [3]. Our contribution is to define a new node merger scheme that relaxes such a DD while allowing Lagrangian relaxation to be applied.

Our second contribution is to analyze, in general, when and how Lagrangian relaxation can be combined with DDs. We introduce the concepts of an exact

state and an immediate penalty function and use these concepts to formulate sufficient conditions for implementing Lagrangian relaxation in a relaxed DD. This leads to a general method for bounding dynamic programming models that satisfy the conditions. We find that while the method generates impracticably large relaxed DDs for the job sequencing problems in [3] and [20], it is quite practical for several important types of job sequencing problems.

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 i to a node in layer $i+1$ for some $i \in \{1,\dots,n\}$. Layers 1 and $n+1$ contain a single node, namely the root r and the terminus t, respectively. Each layer i is associated with a finite-domain variable $x_i \in D_i$. The arcs leaving any node in layer i have distinct *labels* in D_i, representing possible values of x_i 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 finite-domain variables can be represented by a weighted decision diagram. The diagram is constructed so that its r–t paths correspond to the feasible solutions of the problem, and 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 [10, 19].

As an example, consider a job sequencing problem with time windows. Each job j begins processing no earlier than the release time r_j and requires processing time p_j. 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. Figure 1 shows a reduced decision diagram for a problem instance with $(r_1, r_2, r_3) = (0, 1, 1)$, $(p_1, p_2, p_3) = (3, 2, 2)$, and $(d_1, d_2, d_3) = (5, 3, 5)$. Variable x_i represents the ith job in the sequence, and arc costs appear in parentheses.

4 Dynamic Programming Models

Decicision diagrams most naturally represent problems with a dynamic programming formulation, because in this case a simple top-down compilation procedure yields a DD that represents the problem. A general dynamic programming formulation can be written

$$h_i(\boldsymbol{S}_i) = \min_{x_i \in X_i(\boldsymbol{S}_i)} \Big\{ c_i(\boldsymbol{S}_i, x_i) + h_{i+1}\big(\phi_i(\boldsymbol{S}_i, x_i)\big) \Big\} \tag{1}$$

Here, $\boldsymbol{S}_i$ is the *state* in stage i of the recursion. Typically the state is a tuple $\boldsymbol{S}_i = (S_{i1},\dots,S_{ik})$ of *state variables*. Also $X_i(\boldsymbol{S}_i)$ is the set of possible *controls* (values of x_i) in state $\boldsymbol{S}_i$, ϕ_i is the *transition function* in stage i, and $c_i(\boldsymbol{S}_i, x_i)$ is

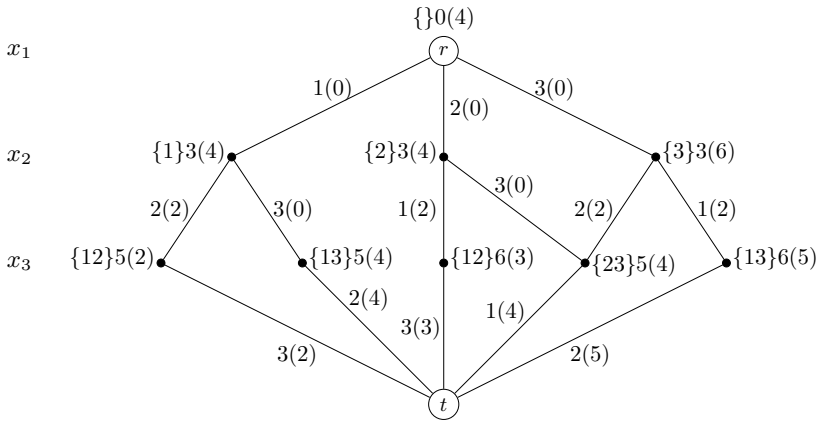

Fig. 1. Decision diagram for a small job sequencing instance, with arc labels and costs shown. States and minimum costs-to-go are indicated at nodes.

the *immediate cost* of control x_i in state $\boldsymbol{S}_i$. We assume there is single initial state $\boldsymbol{S}_1$ and a single final state $\boldsymbol{S}_{n+1}$, so that $h_{n+1}(\boldsymbol{S}_{n+1}) = 0$ and $\phi_n(\boldsymbol{S}_n, x_n) = \boldsymbol{S}_{n+1}$ for all states $\boldsymbol{S}_n$ and controls $x_n \in X_n(S_n)$. The quantity $h_i(\boldsymbol{S}_i)$ is the *cost-to-go* for state $\boldsymbol{S}_i$ in stage i, and an optimal solution has value $h_1(\boldsymbol{S}_1)$.

In the job sequencing problem, the state $\boldsymbol{S}_i$ is the tuple (V_i, t_i), where state variable V_i is the set of jobs scheduled so far, and state variable t_i is the finish time of the last job scheduled. Thus the initial state is $\boldsymbol{S}_1 = (\emptyset, 0)$, and $X_i(S_i)$ is $\{1, \dots, n\} \setminus V_i$. The transition function $\phi_i(\boldsymbol{S}_i, x_i)$ is given by

$$\phi_i\big((V_i, t_i), x_i\big) = \big(V_i \cup \{x_i\},\ \max\{r_{x_i}, t_i\} + p_{x_i}\big)$$

The immediate cost is the tardiness that results from scheduling job x_i in state (V_i, t_i). Thus if $\alpha^+ = \max\{0, \alpha\}$, we have

$$c_i\big((V_i, t_i), x_i\big) = \Big(\max\{r_{x_i}, t_i\} + p_{x_i} - d_{x_i}\Big)^+ \tag{2}$$

We recursively construct a decision diagram D for the problem by associating a state with each node of D. The initial state $\boldsymbol{S}_1$ is associated with the root node t and the final state $\boldsymbol{S}_{n+1}$ with the terminal node t. If state $\boldsymbol{S}_i$ is associated with node u in layer i, then for each $v_i \in X_j(\boldsymbol{S}_i)$ we generate an arc with label v_i leaving u. The arc terminates at a node associated with state $\phi_i(\boldsymbol{S}_i, v_i)$. 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. 1. Each node is labeled by its state (V_i, t_i), followed (in parentheses) by the minimum cost-to-go at the node. The cost-to-go at the terminus t is zero.

5 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 $x_1, \ldots, x_n$ with domains $X_1, \ldots, X_n$. 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 in top-down compilation by *merging* some nodes that are associated with different states. The object is to limit the width of the diagram (the maximum number of nodes in a layer). When we merge nodes with states $\boldsymbol{S}$ and $\boldsymbol{T}$, we associate a state $\boldsymbol{S} \oplus \boldsymbol{T}$ with the resulting node. The operator $\oplus$ is chosen so as to yield a valid relaxation of the given recursion.

The job sequencing problem discussed above uses a relaxation operator

$$(V_i, t_i) \oplus (V_i', t_i') = (V_i \cap V_i', \min\{t_i, t_i'\})$$

V_i is now the set of jobs scheduled along all paths to the current node, and t_i is the earliest finish time of the last scheduled jobs along these paths. The operator is illustrated in Fig. 2, which is the result of merging states $(\{1,2\},6)$ and $(\{2,3\},5)$ in layer 3 of Fig. 1. The relaxed states (V, 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 4 in Fig. 1.

Sufficient conditions under which node merger results in a relaxed decision diagram are developed in [20]. A state $\boldsymbol{S}_i'$ *relaxes* a state $\boldsymbol{S}_i$ when (a) all feasible controls in state $\boldsymbol{S}_i$ are feasible in state $\boldsymbol{S}_i'$, and (b) the immediate cost of any given feasible control in $\boldsymbol{S}_i$ is no less than its immediate cost in $\boldsymbol{S}_i'$. That is, $X_i(\boldsymbol{S}_i) \subseteq X_i(\boldsymbol{S}_i')$, and $c_i(\boldsymbol{S}_i, x_i) \geq c_i(\boldsymbol{S}_i', x_i)$ for all $x_i \in X_i(\boldsymbol{S}_i)$.

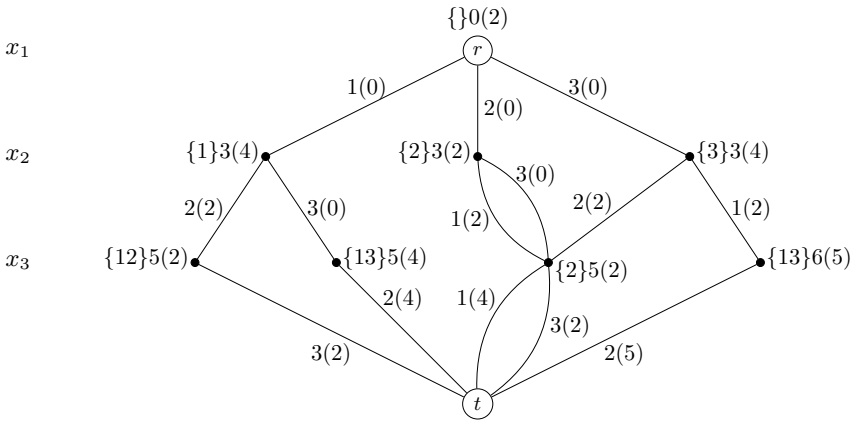

Fig. 2. A relaxation of the decision diagram in Fig. 1.

Theorem 1 ([20]). *If the following conditions are satisfied, the merger of nodes with states $\boldsymbol{S}_i$ and $\boldsymbol{T}_i$ within a decision diagram results in a valid relaxation of the diagram.*

- *$\boldsymbol{S}_i \oplus \boldsymbol{T}_i$ relaxes both $\boldsymbol{S}_i$ and $\boldsymbol{T}_i$.*
- *If state $\boldsymbol{S}'_i$ relaxes state $\boldsymbol{S}_i$, then given any control v that is feasible in $\boldsymbol{S}_i$, $\phi(\boldsymbol{S}'_i, v)$ relaxes $\phi(\boldsymbol{S}_i, v)$.*

All relaxed diagrams we consider are associated with a dynamic programming model that satisfies the conditions of the theorem. The shortest path problem in the relaxed DD requires a modification of the original dynamic programming model that accounts for the merger of states. Also, it is sometimes necessary to use additional state variables to obtain a valid relaxation [5], and so we replace the state vector $\boldsymbol{S}_i$ with a possibly enlarged vector $\bar{\boldsymbol{S}}_i$. The recursive model becomes

$$\bar{h}_i(\bar{\boldsymbol{S}}_i) = \min_{x_i \in X_i(\bar{\boldsymbol{S}}_i)} \Big\{ c_i(\bar{\boldsymbol{S}}_i, x_i) + \bar{h}_{i+1}\Big(\rho_{i+1}\big(\phi_i(\bar{\boldsymbol{S}}_i, x_i)\big)\Big) \Big\} \tag{3}$$

where $\rho_{i+1}(\bar{\boldsymbol{S}}_{i+1})$ is a relaxation of state $\bar{\boldsymbol{S}}_{i+1}$ that reflects the merger of states in stage $i+1$ of the recursion. In the example, $\bar{\boldsymbol{S}}_i = \boldsymbol{S}_i$, since no additional states are necessary to formulate the relaxation.

It will be convenient to distinguish *exact* from *relaxed* state variables in model (3). A state variable S_{ij} in (3) is exact if any sequence of controls $x_1, \dots, x_{i-1}$ that leads to a given value of S_{ij} in the original recursion (1) leads to that same value in the relaxed recursion (3). Otherwise S_{ij} is relaxed. In the example, neither V_i nor t_i is exact if any pair of states can be merged. However, if we permit the merger of (V_i, t_i) and (V'_i, t'_i) only when $t_i = t'_i$, then state variable t_i is exact. In general we have the following, which is easy to show.

Lemma 1. *A state variable S_{ij} is exact if states $\boldsymbol{S}_i$, $\boldsymbol{S}'_i$ are merged only when $S_{ij} = S'_{ij}$.*

6 Lagrangian Duality and Decision Diagrams

Consider an optimization problem

$$z^* = \min_{\boldsymbol{x} \in \mathcal{X}} \big\{ f(\boldsymbol{x}) \;\big|\; \boldsymbol{g}(x) = \boldsymbol{0} \big\} \tag{4}$$

where $\boldsymbol{x} = (x_1, \dots, x_n)$, $\boldsymbol{g}(\boldsymbol{x}) = (g_1(\boldsymbol{x}), \dots, g_m(\boldsymbol{x}))$ and $\boldsymbol{0} = (0, \dots, 0)$. The condition that $\boldsymbol{x} \in \mathcal{X}$ is typically represented by a constraint set. A *Lagrangian relaxation* of (4) has the form

$$\theta(\boldsymbol{\lambda}) = \min_{\boldsymbol{x} \in \mathcal{X}} \big\{ f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x}) \big\} \tag{5}$$

where $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_m)$. The relaxation *dualizes* the constraints $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$. It is easy to show that $\theta(\boldsymbol{\lambda})$ is a lower bound on z^* for any $\boldsymbol{\lambda} \in \mathbb{R}^m$. The *Lagrangian dual* of (4) seeks the tightest bound $\theta(\boldsymbol{\lambda})$:

$$\max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \big\{ \theta(\boldsymbol{\lambda}) \big\} \tag{6}$$

The motivation for using a Lagrangian dual is to obtain tight bonds while dualizing troublesome constraints. If $\boldsymbol{g}(\boldsymbol{x})$ depends on a very small number of state variables, then dualizing the constraint $\boldsymbol{g}(\boldsymbol{x}) = 0$ may allow one to solve the problem within time and space constraints.

Lagrangian duality can be illustrated in the minimum tardiness job sequencing problem discussed earlier. The variables $x_1, \dots, x_n$ should have different values, or equivalently, that each job j should occur in a given solution exactly once. Since a relaxed DD does not enforce this condition, we dualize the constraint $\boldsymbol{g}(\boldsymbol{x}) = 0$, where $\boldsymbol{g}(\boldsymbol{x}) = (g_1(\boldsymbol{x}), \dots, g_n(\boldsymbol{x}))$ and

$$g_j(\boldsymbol{x}) = -1 + \sum_{i=1}^{n} (x_i = j), \quad \text{with} \quad (x_i = j) = \begin{cases} 1 \text{ if } x_i = j \\ 0 \text{ otherwise} \end{cases}$$

As observed in [3], the Lagrangian penalty $\lambda_j g_j(\boldsymbol{x})$ can be represented in a relaxed DD by adding λ_j to the cost of each arc corresponding to control j and subtracting $\sum_j \lambda_j$ from the cost of each arc leaving the root node. Then the length of each r–t path includes the Lagrangian penalty $\boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x})$ for the corresponding solution $\boldsymbol{x}$, which is zero if $\boldsymbol{x}$ satisfies the all-different constraint.

In general, $\boldsymbol{g}(\boldsymbol{x})$ can be computed recursively in a relaxed DD when there is a vector-valued *immediate penalty function* $\boldsymbol{\gamma}_i(\bar{\boldsymbol{S}}_i^L, x_i)$ for which

$$\boldsymbol{g}(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{\gamma}_i(\bar{\boldsymbol{S}}_i^L, x_i) \tag{7}$$

where each $\bar{\boldsymbol{S}}_i^L$ is a tuple consisting of *exact* state variables $S_{i\ell}$ for $\ell \in L$. In the example, the constraint function $\boldsymbol{g}(\boldsymbol{x})$ requires no state information, and $\bar{\boldsymbol{S}}_i^L$ is empty. The immediate penalty $\boldsymbol{\gamma}_i((V_i, t_i), x_i)$ can be written simply $\boldsymbol{\gamma}_i(x_i)$, where

$$\gamma_{ij}(x_i) = \begin{cases} (x_i = j) & \text{if } i \in \{2, \dots, n\} \\ (x_i = j) - 1 & \text{if } i = 1 \end{cases} \tag{8}$$

for $j = 1, \dots, n$.

Implementing a Lagrangian relaxation (4) in a relaxed DD also requires that the original objective function value $f(\boldsymbol{x})$ be computed as part of the path length. Normally, the path length is only a lower bound on $f(\boldsymbol{x})$. To compute $f(\boldsymbol{x})$ exactly in the relaxed recursion (3), we must have an immediate cost function that depends only on exact state variables. That is, we must have

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \bar{c}_i(\bar{\boldsymbol{S}}_i^K, x_i) \tag{9}$$

where $\bar{\boldsymbol{S}}_i^K$ is a tuple consisting of exact state variables S_{ik} for $k \in K$. In the example, the immediate cost function (2) depends on the state variable t_i, which must therefore be exact. We therefore have $\bar{\boldsymbol{S}}_i^K = (t_i)$. Due to Lemma 1, we can ensure that t_i is exact by merging nodes only when t_i has the same value in the corresponding states. Thus Lagrangian relaxation can be implemented in the minimum tardiness example if we merge nodes in this fashion.

The above observations can be summed up as follows.

Theorem 2. *Lagrangian relaxation (5) can be implemented in a relaxed DD if there are immediate cost functions $\bar{c}_i(\bar{\boldsymbol{S}}_i^K, x_i)$ for which (9) holds and immediate penalty functions $\boldsymbol{\gamma}_i(\bar{\boldsymbol{S}}_i^L, x_i)$ for which (7) holds, where $\bar{\boldsymbol{S}}_i^K$ and $\bar{\boldsymbol{S}}_i^L$ consist entirely of exact state variables in $\bar{\boldsymbol{S}}_i$. In this case the recursion for computing shortest paths in the relaxed DD becomes*

$$\bar{h}_i(\bar{\boldsymbol{S}}_i, \boldsymbol{\lambda}) = \min_{x_i \in X_i(\bar{\boldsymbol{S}}_i)} \left\{ \bar{c}_i(\bar{\boldsymbol{S}}_i^K, x_i) + \boldsymbol{\lambda}^T \boldsymbol{\gamma}_i(\bar{\boldsymbol{S}}_i^L, x_i) + \bar{h}_{i+1}\Big(\rho_{i+1}\big(\phi_i(\bar{\boldsymbol{S}}_i, x_i)\big), \boldsymbol{\lambda}\Big) \right\}$$

Corollary 1. *Lagrangian relaxation (5) can be implemented in a relaxed DD if nodes are merged only when their states agree on the values of the state variables on which the immediate cost functions and the immediate penalty functions depend. That is, nodes with states $\bar{\boldsymbol{S}}_i$ and $\bar{\boldsymbol{T}}_i$ are merged only when $\bar{\boldsymbol{S}}_i^K = \bar{\boldsymbol{T}}_i^K$ and $\bar{\boldsymbol{S}}_i^L = \bar{\boldsymbol{T}}_i^L$, where K and L are in as Theorem 2.*

7 Problem Classes

We now examine a few classes of job sequencing problems to determine whether they are suitable for Lagrangian relaxation on a relaxed DD. All of these problems have an all-different constraint that is dualized as before using the immediate penalty function (8). Since this function depends on no state variables, the state variables that must be exact are simply those on which the immediate cost function depends. That is, $\bar{\boldsymbol{S}}_i^L$ is empty, and suitability for relaxation depends on which variables are in $\bar{\boldsymbol{S}}_i^K$. We note that even when DD-based Lagrangian relaxation is not suitable for a given problem class, it may be useful when combined with branching, or when the relaxed DD is embedded in a larger model.

7.1 Sequencing with Time Windows

Problems in which jobs with state-independent processing times are sequenced, possibly subject to time windows, are generally conducive to Lagrangian relaxation on DDs. The problem of minimizing total tardiness is discussed above, and computational results are presented in Section 8. Minimizing makespan or the number of late jobs is treated similarly.

A popular variation on the problem minimizes the sum of penalized earliness and tardiness with respect to a common due date [8, 9, 11, 16, 17, 22, 25]. Earliness of job j is weighted by α_j and lateness by β_j. The recursive model for an exact DD uses the same state variables (V_i, t_i) as the minimum tardiness problem. However, a valid relaxed DD requires an additional state variable s_i that represents latest start time, while t_i again represents earliest finish time. The transition and immediate cost functions are

$$\begin{aligned}
\bar{\phi}_i\big((V_i, s_i, t_i), x_i\big) &= \big(V_i \cup \{x_i\},\ s_i + p_{x_i},\ t_i + p_{x_i}\big) \\
\bar{c}_i\big((V_i, s_i, t_i), x_i\big) &= \alpha_{x_i}\big(s_i + p_{x_i} - d_{x_i}\big)^+ + \beta_{x_i}\big(t_i + p_{x_i} - d_{x_i}\big)^+
\end{aligned}$$

The merger operation is

$$(V_i, s_i, t_i) \oplus (V_i', s_i', t_i') = \big(V_i \cap V_i', \max\{s_i, s_i'\}, \min\{t_i, t_i'\}\big)$$

The immediate cost depends on both s_i and t_i, which means that both of these state variables must be exact. This may appear to result in a large relaxed DD, because nodes can be merged only when they agree on both state variables. However, since s_i and t_i are initially equal, and these state variables are exact, they remain equal throughout the relaxed DD construction. The resulting DD is therefore the same that would result if a single state variable were exact. Computational results are presented Section 8.

7.2 Time-Dependent Costs and/or Processing Times

Costs and processing times can be time-dependent in two senses: they may depend on the position of each job in the sequence, or on the clock time at which the job is processed. Both senses occur in the literature, and both can be treated with Lagrangian relaxation on DDs.

If the processing time p_{ij} of job j depends on the position i of the job in the sequence, then the immediate cost in the relaxation is

$$\bar{c}_i((V_i, t_i), x_i) = \big(\max\{t_i, r_{x_i}\} + p_{ix_i} - d_{x_i}\big)^+$$

which depends only on the state variable t_i. Since $\bar{c}_i$ is already indexed by the position i, any other element of cost that depends on i is easily incorporated into the function. Thus we need only ensure that states are merged only when they agree on t_i, a condition that is already satisfied in the relaxed model described above for minimum-tardiness sequencing problems.

If the processing time $p_j(s)$ of job j depends on the time s at which job j starts, the immediate cost is

$$\bar{c}_i((V_i, t_i), x_i) = \big(\max\{t_i, r_{x_i}\} + p_{x_i}(\max\{t_i, r_{x_i}\}) - d_{x_i}\big)^+$$

which again depends only on state variable t_i. Any other time-dependent element of cost likewise depends only on t_i, and so states can be merged whenever they agree on t_i.

7.3 Sequence-Dependent Processing Times

We refer to a job j's processing time as sequence dependent when its processing time $p_{j'j}$ depends on the immediately preceding job j' in the sequence. When there are no time windows and the objective is to minimize travel time, the problem is a traveling salesman problem. The state variables are V_i and the immediately preceding job y_i. The transition and immediate cost functions are

$$\begin{aligned} \bar{\phi}_i\big((V_i, y_i), x_i\big) &= \big(V_i \cup \{x_i\}, x_i\big) \\ \bar{c}_i\big((V_i, y_i), x_i\big) &= p_{y_i x_i} \end{aligned} \tag{10}$$

Since the immediate cost depends only on state y_i, nodes can be merged whenever they are reached using the same control. This permits a great deal of reduction in the relaxed DD and suggests that a Lagrangian approach to bounding can be effective. While pure traveling salesman problems are already well solved, a DD-based Lagrangian bounding technique may be useful when there are side constraints.

When there are time windows in the problem, an additional state variable t_i representing the finish time of the previous job is necessary for a stand-alone relaxed DD. The transition function is

$$\bar{\phi}_i\big((V_i, y_i, t_i), x_i\big) = \big(V_i \cup \{x_i\}, x_i, \max\{r_{x_i}, t_i\} + p_{y_i x_i}\big) \tag{11}$$

The immediate cost functions for minimizing travel time and total tardiness, respectively, are

$$\bar{c}_i\big((V_i, y_i, t_i), x_i\big) = (r_{x_i} - t_i)^+ + p_{y_i x_i} \tag{12}$$

$$\bar{c}_i\big((V_i, y_i, t_i), x_i\big) = \big(\max\{r_{x_i}, t_i\} + p_{y_i x_i} - d_{x_i}\big)^+ \tag{13}$$

In either case, the immediate cost depends on two state variables y_i and t_i, and nodes can be merged only when they agree on these variables. This is likely to result in an impracticably large relaxed DD. For example, if there are 50 jobs and a few hundred possible values of t_i, a layer of the relaxed DD could easily expand to tens of thousands of nodes. We confirmed this with preliminary experiments on the Dumas instances [14]. Lagrangian relaxation on a stand-alone relaxed DD therefore does not appear to be a promising approach to bounding TSP problems with time windows. One can, of course, use the simpler DD described by (10) to bound travel time (although not total tardiness), as is done in [3]. However, this relaxation ignores time windows altogether and would yield a weaker bound than (11)–(12).

7.4 State-Dependent Processing Times

We refer to a job's processing times as state dependent when they depend on one or more of the state variables in the recursion, such as the set V_i of jobs already processed. Such a problem is studied in [20], where the processing time is less if a certain job has already been processed. State variables are again V_i and t_i, but to build a relaxed DD we need an additional state variable U_i representing the sets of jobs that have been processed along some path to the current node. The transition function and immediate cost function are

$$\bar{\phi}_i\big((V_i, U_i, t_i), x_i\big) = \big(V_i \cup \{x_i\}, U_i \cup \{x_i\}, \max\{r_{x_i}, t_i\} + p_{x_i}(U_i)\big)$$
$$\bar{c}_i\big((V_i, U_i, t_i), x_i\big) = \big(\max\{r_{x_i}, t_i\} + p_{x_i}(U_i) - d_{x_i}\big)^+$$

where the processing time is $p_{x_i}(U_i)$. The merger operation is

$$(V_i, U_i, t_i) \oplus (V_i', U_i', t_i') = \big(V_i \cap V_i', U_i \cup U_i', \min\{t_i, t_i'\}\big)$$

Since the cost depends on both t_i and U_i, these state variables must be exact, and states can be merged only when they agree on the values of t_i and U_i. This predicts that the relaxed DD will grow rapidly with the number of jobs. DD-based Lagrangian relaxation is therefore not a promising approach to this type of problem.

8 Computational Experiments

8.1 Problem Instances

To assess the quality of bounds obtained from Lagrangian relaxation on relaxed DDs, it is necessary to obtain problem instances with known optimal values, or values that are likely to be close to the optimum. We carry out tests on two well-known sets of instances, corresponding to two sequencing problems identified earlier to be suitable for bounding. One is the set of minimum weighted tardiness instances of Crauwels, Potts and Wassenhove [13], which we refer to as the CPW instances. The other is the Biskup-Feldman collection of minimum weighted earliness-plus-tardiness instances with a common due date [8].

The CPW set consists of 125 instances of each of three sizes: 40 jobs, 50 jobs, and 100 jobs. We compute bounds for first 25 instances in the 40- and 50-job sets. These instances exhibit a wide range of gradually increasing tardiness values, thus providing a diverse selection for testing. Optimal solutions are given in [8] for all of these instances except instance 14 with 40 jobs, and instances 11, 12, 14 and 19 with 50 jobs. Solution values presented in [8] for the unsolved instances are apparently the best known.

The Biskup-Feldman collection includes 10 instances of each size, where the sizes are 10, 20, 50, 100 and 200 jobs. We study the instances with 20, 50 and 100 jobs. The instances specify only the processing times p_j and the earliness/tardiness weights α_j, β_j described earlier. The common due date for all jobs in an instance is not specified. Typical practice is to set the due date equal to $d(h) = \lfloor h \sum_j p_j \rfloor$, where h is a parameter.

We compare our lower bounds against the best known solution values reported in [25]. These authors compute the earliness penalty with respect to $d(h_1)$ and the tardiness penalty with respect to $d(h_2)$ for $h_1 < h_2$, so that the penalty for each job j is $\alpha_j(d(h_1) - t_j)^+ + \beta_j(t_j - d(h_2))^+$, where t_j is the finish time of job j. Heuristics are used in [25] to solve instances with $(h_1, h_2) = (0.1, 0.2)$, $(0.1, 0.3)$, $(0.2, 0.5)$, $(0.3, 0.4)$, and $(0.3, 0.5)$. We study instances with $(h_1, h_2) = (0.1, 0.2)$ and $(0.2, 0.5)$ to provide a look at contrasting cases. To our knowledge, none of these instances have been solved to proven optimality.

8.2 Solving the Lagrangian Dual

We use subgradient optimization to solve the Lagrangian dual problem (6). Each iterate $\boldsymbol{\lambda}^k$ is obtained from the previous by the update formula

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \sigma_k \boldsymbol{g}(\boldsymbol{x}^k)$$

where $\boldsymbol{x}^k$ is the value of $\boldsymbol{x}$ obtained when computing $\theta(\boldsymbol{\lambda}^k)$ in (5). The art of Lagrangian optimization is choosing the step size σ_k. This choice is avoided in [3] by using the Kelly-Cheney-Goldstein bundle method [21] of deriving $\boldsymbol{\lambda}^k$ from previous iterates. However, Polyak's method [23] seems better suited to our purposes, because it is much easier to implement, and it requires as a parameter only an upper bound θ^* on the optimal value of $\theta(\boldsymbol{\lambda})$. Such a bound is available in practice, because one seeks to estimate how far a known solution value lies from the optimal value, and that solution value is an upper bound θ^*. Polyak's method defines the step size to be

$$\sigma_k = \frac{\theta^* - \theta(\boldsymbol{\lambda}^k)}{\|\boldsymbol{g}(\boldsymbol{x}^k)\|_2^2}$$

8.3 Building the Relaxed Diagram

In previous work [1, 4, 6], heuristically-selected nodes are merged in each layer after all states obtainable from the previous layer are generated. Since we wish to merge all nodes that agree on t_i, and no others, we merge these nodes as we generate the states, rather than first generating all possible states and then merging nodes. This drastically reduces computation time and results in reasonable widths that gradually increase as layers are created.

8.4 Computational Results

The computational tests were run on a Dell XPS-13 laptop computer with Intel Core i7-6560U (4M cache, 3.2 GHz) and 16GB memory. Results for the CPW instances appear in Table 1. The table displays optimal (or best known) values and DD-based bounds, as well as the absolute and relative gap between the two. It also shows the maximum width of the relaxed DD (always obtained in layer n), the time required to build the DD, and the time consumed by the subgradient algorithm. Since a subgradient iteration requires only the solution of a shortest path problem in the relaxed DD, we allowed the algorithm to run for 50,000 iterations to obtain as much improvement in the bound as seemed reasonably possible. Due to slow convergence, which is typical for subgradient algorithms, a bound that is nearly as tight can be obtained by executing only, say, 20% as many iterations.

The bounds in Table 1 are reasonably tight. The gap is well below one percent in most cases, and below 0.1% in about a quarter of the cases, although a few of the bounds are rather weak. The optimal value was obtained for one instance. The gap for instance 14 in the 40-job table suggests that the best known value is probably not optimal, while no such inference can be drawn for the 4 unsolved 50-job instances.

Results for the Biskup-Feldman instances appear in Table 2, where the bounds are compared with the best known solutions. The relaxed DDs are the same for the two sets of due dates $(h_1, h_2) = (0.1, 0.2), (0.2, 0.5)$; only the costs differ. The bounds are very tight, resulting in gaps that are mostly under 0.1%. This

indicates that the known solutions are, at worst, very close to optimality. In fact, optimality is proved for 8 instances, which represents the first time that any of these instances have been solved. The bounds may be equal to the optimal value for other instances, since the known values displayed may not be optimal.

9 Conclusion

We have shown how Lagrangian relaxation in a stand-alone relaxed decision diagram can yield tight optimization bounds for certain job sequencing problems. We also characterized problems on which this approach is likely to be effective; namely, problems in which a relaxed DD of reasonable width results from a restricted form of state merger. The restriction is that states may be merged only when they agree on the values of state variables on which the cost function and dualized constraints depend in a recursive formulation of the problem.

Based on this analysis, we observed that job sequencing problems with state-independent processing times and time windows are suitable for this type of bounding, whether one minimizes tardiness, makespan or the number of late jobs. The same is true when processing times are dependent on when the job is processed or its position in the sequence. The traveling salesman problem can also bounded in this fashion. However, the TSP with time windows, as well as problems with state-dependent processing times in general, are normally unsuitable for DD-based Lagrangian relaxation, unless the relaxed diagram is combined with branching or embedded in a larger model.

We ran computational experiments on 110 instances from the well-known Crauwels-Potts-Wassenhove and Biskup-Feldman problem sets, with sizes ranging from 20 to 100 jobs. We found that DD-based Lagrangian relaxation can provide tight bounds for nearly all of these instances. This is especially true of the Biskup-Feldman instances tested, all of which were unsolved prior to this work. We showed that the best known solutions are almost always within a small fraction of one percent of the optimum, and we proved optimality for 8 of the solutions. To our knowledge, these are the first useful bounds that have been obtained for these instances.

More generally, our analysis can be used to identify dynamic programming models that may have a useful relaxation based on relaxed decision diagrams and Lagrangian duality.

Table 1. Comparison of bounds with optimal values (target) of CPW instances. Computation times are in seconds.

	40 jobs							50 jobs						
Instance	Target	Bound	Gap	Percent gap	Max width	Build time	Subgr time	Instance	Target	Bound	Gap	Percent gap	Max width	Build time	Subgr time
1	913	883	30	3.29%	3163	16	1287	1	2134	2100	34	1.59%	4525	48	2633
2	1225	1179	46	3.76%	3652	20	1420	2	1996	1864	132	6.61%	4453	53	2856
3	537	483	54	10.06%	3556	20	1443	3	2583	2552	31	1.20%	4703	52	2697
4	2094	2047	47	2.24%	3568	20	1427	4	2691	2673	18	0.67%	4585	55	2703
5	990	980	10	1.01%	3305	18	1312	5	1518	1342	176	11.59%	4590	52	2658
6	6955	6939	16	0.23%	3588	20	1406	6	26276	26054	222	0.84%	4490	48	2562
7	6324	6299	25	0.40%	3509	20	1437	7	11403	11128	275	2.41%	4357	45	2499
8	6865	6743	122	1.78%	3508	20	1393	8	8499	8490	9	0.11%	4396	46	2501
9	16225	16049	176	1.08%	3699	22	1468	9	9884	9507	377	3.81%	4696	52	2660
10	9737	9591	146	1.50%	3426	19	1346	10	10655	10594	61	0.57%	4740	53	2738
11	17465	17417	48	0.27%	3770	23	1493	11	*43504	43472	32	0.07%	4597	50	2606
12	19312	19245	67	0.35%	3644	22	1435	12	*36378	36303	75	0.21%	4500	48	2655
13	29256	29003	253	0.86%	3736	22	1506	13	45383	45310	73	0.16%	4352	47	2521
14	*14377	14100	277	1.93%	3609	21	1406	14	*51785	51702	83	0.16%	4699	52	2656
15	26914	26755	159	0.59%	3849	23	1554	15	38934	38910	47	0.12%	4650	52	2630
16	72317	72120	197	0.27%	3418	19	1382	16	87902	87512	390	0.44%	4589	49	2623
17	78623	78501	122	0.16%	3531	20	1384	17	84260	84066	194	0.23%	4359	45	2526
18	74310	74131	179	0.24%	3524	20	1431	18	104795	104633	162	0.15%	4448	47	2505
19	77122	77083	39	0.05%	3407	19	1320	19	*89299	89163	136	0.15%	4609	50	2660
20	63229	63217	12	0.02%	3506	20	1344	20	72316	72222	94	0.13%	4678	51	2659
21	77774	77754	20	0.03%	3766	22	1433	21	214546	214476	70	0.03%	4406	47	2580
22	100484	100456	28	0.03%	3489	20	1382	22	150800	150800	0	0%	4098	39	418
23	135618	135617	1	0.001%	3581	21	1375	23	224025	223922	103	0.05%	4288	44	2441
24	119947	119914	33	0.03%	3477	19	1295	24	116015	115990	25	0.02%	4547	49	2620
25	128747	128705	42	0.03%	3597	22	1339	25	240179	240172	7	0.003%	4639	51	2686

*Best known solution

*Best known solution

Table 2. Comparison of bounds with best known values (target) of Biskup-Feldman instances.

	$(h_1, h_2) = (0.1, 0.2)$					$(h_1, h_2) = (0.2, 0.5)$						
Instance	Target	Bound	Gap	Percent gap	Subgr time	Target	Bound	Gap	Percent gap	Subgr time	Max width	Build time
20 jobs												
1	4089	4089	0	0%	1	1162	1162	0	0%	1	323	0.12
2	8251	8244	7	0.08%	28	2770	2766	4	0.14%	27	287	0.08
3	5881	5877	4	0.07%	27	1675	1669	6	0.36%	28	287	0.08
4	8977	8971	6	0.07%	27	3113	3108	5	0.16%	27	287	0.08
5	4028	4024	4	0.10%	32	1192	1187	5	0.42%	32	341	0.10
6	6306	6288	18	0.29%	26	1557	1557	0	0%	1	271	0.09
7	10204	10204	0	0%	1	3573	3569	4	0.11%	29	305	0.09
8	3742	3739	3	0.08%	25	990	979	11	1.11%	25	267	0.08
9	3317	3310	7	0.21%	21	1056	1055	1	0.09%	22	230	0.07
10	4673	4669	4	0.09%	29	1355	1349	6	0.44%	30	320	0.09
50 jobs												
1	39250	39250	0	0%	16	12754	12752	2	0.02%	501	931	2.8
2	29043	29043	0	0%	191	8468	8463	5	0.06%	524	931	2.9
3	33180	33180	0	0%	300	9935	9935	0	0%	66	836	2.4
4	25856	25847	9	0.03%	549	7373	7335	38	0.52%	521	932	2.8
5	31456	31439	17	0.05%	540	8947	8938	9	0.10%	529	932	3.0
6	33452	33444	8	0.02%	544	10221	10213	8	0.08%	532	932	2.9
7	42234	42228	6	0.01%	491	12002	11981	21	0.17%	465	835	2.4
8	42218	42203	15	0.04%	491	11154	11141	13	0.12%	478	833	2.4
9	33222	33218	4	0.01%	503	10968	10965	3	0.03%	508	884	2.7
10	31492	31481	11	0.03%	529	9652	9650	3	0.03%	522	932	2.9
100 jobs												
1	139573	139556	17	0.01%	4075	39495	39467	28	0.07%	3968	1882	42
2	120484	120465	19	0.02%	4065	35293	35266	27	0.08%	4068	1882	44
3	124325	124289	36	0.03%	3957	38174	38150	24	0.06%	4059	1882	42
4	122901	122876	25	0.02%	3903	35498	35467	31	0.09%	3964	1882	42
5	119115	119101	14	0.01%	3925	34860	34826	34	0.10%	4016	1882	42
6	133545	133536	9	0.007%	3987	35146	35123	23	0.07%	3961	1882	43
7	129849	129830	19	0.01%	4027	39336	39303	33	0.08%	3974	1882	43
8	153965	153958	7	0.005%	3722	44963	44927	36	0.08%	3865	1784	39
9	111474	111466	8	0.007%	3930	31270	31231	39	0.12%	4008	1882	42
10	112799	112792	7	0.006%	3936	34068	34048	20	0.06%	4003	1882	42

References

1. 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)
2. Bergman, D., Ciré, A.A., van Hoeve, W.J.: Improved constraint propagation via Lagrangian decomposition. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming (CP 2015). LNCS, vol. 9255, pp. 30–38. Springer (2015)
3. Bergman, D., Ciré, A.A., van Hoeve, W.J.: Lagrangian bounds from decision diagrams. Constraints 20, 346–361 (2015)
4. 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)
5. 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 (2014)
6. Bergman, D., Ciré, A.A., van Hoeve, W.J., Hooker, J.N.: Decision Diagrams for Optimization. Springer (2016)
7. 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)
8. Biskup, D., Feldman, M.: Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. Computers and Operations Research 28, 787–801 (2001)
9. Biskup, D., Feldman, M.: On scheduling around large restrictive common due windows. European Journal of Operational Research 162, 740–761 (2005)
10. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35, 677–691 (1986)
11. Chen, Z.L.: Scheduling and common due date assignment with earliness-tardiness penalties and batch delivery costs. European Journal of Operational Research 93, 49–60 (1996)
12. Ciré, A.A., van Hoeve, W.J.: Multivalued decision diagrams for sequencing problems. Operations Research 61, 1411–1428 (2013)
13. Crauwels, H., Potts, C., Wassenhove, L.V.: Local search heuristics for the single machine total weighted tardiness scheduling problem. INFORMS Journal on Computing 10, 341–350 (1998)
14. Dumas, Y., Desrosiers, J., Gelinas, E., Solomon, M.M.: An optimal algorithm for the traveling salesman problem with time windows. Operations Research 43, 367–371 (1995)
15. 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)
16. Hall, N.G., Posner, M.E.: Earliness-tardiness scheduling problems, I: Weighted deviation of completion times about a common due date. Operations Research 39, 836–846 (1991)
17. Hall, N.G., Posner, M.E., Sethi, S.P.: Earliness-tardiness scheduling problems, II: Weighted deviation of completion times about a restrictive common due date. Operations Research 39, 847–856 (1991)
18. Hooker, J.N.: Discrete global optimization with binary decision diagrams. In: GICOLAG 2006. Vienna, Austria (December 2006)

19. 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)
20. Hooker, J.N.: Job sequencing bounds from decision diagrams. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming (CP 2017). LNCS, vol. 10416, pp. 565–578. Springer (2017)
21. Lemaréchal, C.: Lagrangian relaxation. In: Jünger, M., Naddef, D. (eds.) Computational Combinatorial Optimization. Lecture Notes in Computer Science, vol. 2241, pp. 112–156. Springer (2001)
22. Ow, P.S., Morton, T.E.: The single machine early/tardy problem. Management Science 35, 177–191 (1989)
23. Polyak, B.T.: Introduction to Optimization (translated from Russian). Optimization Software, New York (1987)
24. Serra, T., Hooker, J.N.: Compact representation of near-optimal integer programming solutions. Mathematical Programming (to appear)
25. Ying, K.C., Lin, S.W., Lu, C.C.: Effective dynamic dispatching rule and constructive heuristic for solving single-machine scheduling problems with a common due window. International Journal of Production Research 55, 1707–1719 (2017)