



## Discrete Optimization

# Activity consolidation to improve responsiveness

Jeffrey L. Rummel <sup>a</sup>, Zhiping Walter <sup>b,\*</sup>, Rajiv Dewan <sup>c</sup>, Abraham Seidmann <sup>c</sup>

<sup>a</sup> School of Business, University of Connecticut, Storrs, CT 06269, USA

<sup>b</sup> College of Business and Administration, University of Colorado at Denver, P.O. Box 173364, Campus Box 165, Denver, CO 80217-3364, USA

<sup>c</sup> William E. Simon School of Business Administration, University of Rochester, Rochester, NY 14627, USA

Received 11 March 2002; accepted 30 July 2003

Available online 6 December 2003

### Abstract

There is a long history of modeling projects to meet time and cost objectives. Most of these models look at adjusting the level of resources available to the project in order to crash the time required to complete certain activities. These models usually take the activities and the graph structure of the project as given and fixed, but in practice there is often significant discretion in how activities are defined. This is especially important when there are information flows and time delays associated with the hand-off between an activity and its successor. This paper models the choice of how to meet the time and cost objectives through combining multiple activities into one while maintaining the original activity precedence relationships. A mixed-integer linear programming model is developed for the problem, and an implicit enumeration and a tabu search heuristic are tested with a suite of problem examples.

© 2003 Elsevier B.V. All rights reserved.

**Keywords:** Project management and scheduling; Tabu search; Business process reengineering; Combinatorial optimization; Workflow design

“Speed is everything in this business. We’re setting the pace for the industry.” – Michael Dell, Chairman and CEO of Dell Computer Corp. [Business Week, 4/7/1997 Pg. 132]

### 1. Introduction

Made to order manufacturing, a strategy successfully implemented by Dell, depends crucially on

a short cycle time from customer order to delivery. This kind of time-based competition is now pervasive in manufacturing as well as service industries where the responsiveness and cycle time, i.e., the time from placement of customer’s order to its fulfillment, is a key determinant of success. Planning customer projects in this kind of an environment requires a new set of tools such as Total Quality Management (TQM), Business Process Reengineering (BPR), or Six Sigma (Bowen et al., 1994; Hammer and Champy, 1993; Pande et al., 2000).

Conventionally, project planning is often done by modeling the project as a graph in the tradition of CPM/PERT literature (Kerzner, 1979; Meredith

\* Corresponding author. Tel.: +1-303-556-6620; fax: +1-303-556-5899.

E-mail address: [zhiping.walter@cudenver.edu](mailto:zhiping.walter@cudenver.edu) (Z. Walter).

and Mantel, 2000): Activities are represented as the nodes and precedence relationships are represented with directed arcs. Once a project graph has been created, a common next step is to analyze ways to shorten the duration of the project (its cycle time). Common methods include leveling the load on particular resources to avoid delaying critical activities (Demeulemeester and Herroelen, 1992; Demeulemeester et al., 1993), overlapping neighboring activities (Krishnan et al., 1997), and removing the coupling between sequential activities thereby transforming sequential processing into parallel processing. This kind of conventional approach suffers from two drawbacks: First, it is hard to decouple activities. Second, as this approach is activity and task focused, it ignores interactivity delay which may contribute significantly to overall cycle time.

Decoupling may be difficult to achieve since two activities may naturally require information exchange and hence parallel processing is not possible. Overlapping activities requires frequent exchange of preliminary information, which may result in high coordination costs. In projects where coordination can be a problem itself, this may not be desirable. The model proposed in this paper focuses on shortening the cycle time by consolidating activities—assigning multiple activities to one resource thereby eliminating the coordination and handoff delay between different activities if they are assigned to different resources. This approach is advisable for projects where coordination presents a great challenge or the handoff delays are significant so that the savings from consolidating activities out-weights the disadvantage of a more sequential project structure. From modeling perspective, we modify the CPM/PERT style project graph by allowing arcs to represent delays between activities, in addition to representing precedence relationships. This representation is suitable for the types of projects that our model focuses on: projects where delays due to inter-activity coordination and scheduling delays constitute a large portion of the overall cycle time.

Inter-activity or handoff delay considered in this paper comes from two sources: the need for coordination between neighboring activities that are assigned to different resources and from the differ-

ence in time when one activity is done and the next one started. For example, a courier service may be needed to deliver the results from the upstream activities to the downstream activities, or a resource assigned to an activity may not be a dedicated resource for this activity and hence delay is anticipated between this resource and its upstream resource. The delay can also come from necessary repetition of certain activity aspects when the resource processing the downstream activity needs to comprehend the results from the upstream activity. These repetitions might not have been incurred were the two activities assigned to the same resource.

Let us consider the process of underwriting and claims processing in the insurance industry. For many commodity insurance products, the process is easily accomplished with telephone operators or automated web sites. But in other insurance settings, the process is much more complex, due to the special nature of the risks being insured (or the claim being paid). There is the potential to use different networks of activities to complete a process, and this requires a method for evaluating the tradeoffs between them. For example, it may be possible to have an experienced employee do all the research and make judgments about a claim. It may also be possible to divide the work among lower level employees who each has expertise in a particular aspect of the claim process. Notice that if one experienced employee works on the entire claim, subsequent activities must wait until all the work is done, whereas if the work is divided, it may be possible for those activities to start earlier, as each of the lower level employees finish their part of the claim. On the other hand, in the former case, there isn't any delay due to coordination among different employees, while in the latter case, some coordination will be needed and hence delays can arise. If coordination is as efficient as assembly lines, delays will be small. However, most times, delays will be substantial—longer than the actual processing time. The IBM Credit case described by Hammer and Champy (Hammer and Champy, 1993), is a “lighthouse” example of significant inter-activity delays. In this case, IBM credit was competing with other corporate credit agencies to finance purchases of IBM goods and equipment.

Quick response was important and yet the average cycle time for the credit approval process was 6 days. Analysis revealed that the total time spent in performing the activities was only 90 minutes. IBM Credit responded by consolidating all the activities into a single activity performed by a generalist. This kind of situation is typical of transaction processing environments such as accounts payable, claims processing, etc., where most of the cycle time is taken up by inter-activity delays. It is these inter-activity delays that this paper focuses on in order to improve project cycle time. More specifically, this paper proposes to use “activity consolidation”—combining two or more activities into one, which is equivalent to assigning multiple activities to the same resource—to plan a project. We assume that consolidation will not affect activity times and that delays are either totally eliminated if the two activities are combined or kept as the same if they are not.

Eliminating inter-activity delays is not attained without incurring a cost. Consolidation may lead to increased agency costs (e.g., from reducing checks and control). It may also increase the cost by reducing productivity gains from specialization. Activity consolidations also require fixed costs, such as training and IT investment. Hence the proposed model considers the tradeoff between the costs of the total project duration and consolidation costs that might reduce that duration.

Since the modeling of activity consolidation is equivalent to assigning a common resource to the consolidated activities, these activities now must be completed sequentially, while previously, some of them may be processed in parallel. As a result, the downstream activities can start only after the combined activity as a whole has been completed. This interpretation of consolidation effectively results in changes in the project graph and is substantially different from existing project-scheduling models.

This approach of activity consolidation considers the tradeoffs between parallel and sequential flows of work and changes in the project structure. The idea is synthesized from the general rules or heuristics found in the Business Process Reengineering literature. These heuristics include “organize around outcomes,” “link parallel activities,”

“reducing checks and controls” (consolidation of the processing and the controlling activities so that the same person is responsible for both), “employee empowerment” (give the employee who gathers information the authority to make decisions), and “replace specialists with generalists” (Hammer, 1990). All of these heuristics implicitly involve activity consolidation in one form or another. Even though the effects of these consolidations have been evaluated individually (Buzacott, 1996), the combined effect of consolidation in different parts of a project graph on the cycle time of the entire project graph has not been modeled or analyzed. This paper is an attempt at that.

It is worth noting that although we focus more on the inter-activity delays rather than resources for task performance, we do not ignore resource issues. Any assignment of activities to a resource different from the default one will incur costs. This cost is explicitly modeled in the objective function and constraints the types of consolidation that can be utilized to improve the project cycle time. Further, consolidating activities results in decreased parallelism and hence resource scheduling for the new process remains feasible if the original plan is feasible. In this sense, the activity consolidation approach modeled in this paper is complementary to the resource-constraint based approaches. The two could be used in conjunction to decide on activity consolidation and allocation.

Another point of comparison between our model and conventional CPM/PERT approaches to cycle time improvement is that while the activities and precedence relationships are taken as given in CPM/PERT analysis, our model focuses heavily on redefining activities (by assignment of activities) and precedence relationships through consolidation, treating the consolidated activity as an atomic activity whose intermediate outputs are not available until the whole activity is completed.

The structure of the paper is as follows: Section 2 introduces our mixed-integer linear programming model using a stylized order fulfillment process. Section 3 discusses an implicit enumeration algorithm for solving the model. Section 4 discusses a tabu heuristic. We present our numerical results in

Section 5 and a detailed example in Section 6. The paper is concluded in Section 7.

## 2. Model description

The activity-consolidation model introduced here is most useful when the coordination between activities introduces time delays that might be saved if the two activities were consolidated. Fig. 1 summarizes the notation. As with the other parameters, our model assumes that the delays between two activities are known and relatively fixed.

The definition of an activity differs slightly in this model from the usual definition in project management models, especially resource constrained models. A usual project activity consists of a few steps that have historically been done as one package, but in our model, those steps will be broken into multiple activities, each of which is a unit of work where the skill level of doing that work is constant and which can be accomplished by a single worker sequentially. Each simple activity has a corresponding level of worker who could do that activity, and we assume that the most cost effective worker is the default for each activity. Now, if we consider combining two simple activities  $i$  and  $j$ , by allowing a single worker to accomplish both activities, how the combinations

should be done depends on the required skill level of each activity and the cost of the two possible combinations,  $x_{ij} = 1$  or  $x_{ji} = 1$ . When a person who would normally handle activities like  $i$  also must be able to handle  $j$  (or vice versa), there is a cost associated with the skill mismatch. If we set  $x_{ij} = 1$ , we are indicating that a particular person who initially has skill levels for only activity  $i$ , now has extra work to do and is now more resource constrained. This is a planning model, and so this model will determine how activities can be combined, which then would be used to staff the process.

An example graph is depicted in Fig. 2 to illustrate our model parameters and decision variables. The project consists of nine activities ( $n = 9$ ). Each activity  $i$  takes an amount of time ( $\tau_i$ ) to finish. In Fig. 2, we set  $\tau_i = i$ . A directed arc from  $i$  to  $j$  indicates that there is an information or material flow from  $i$  to  $j$ , which incurs delays ( $\gamma_{ij} = 4$  for all arcs). Note that there can be “redundant” arcs because the arcs not only represent precedence relationships, they also represent delays between arcs. For example, there may be arcs from  $i \rightarrow j$ ,  $j \rightarrow k$ , and  $i \rightarrow k$  directly. This can be true because the results of activity  $i$  need to be transferred to both activities  $j$  and  $k$ . The inter-activity delay from  $i$  to  $j$  can be different from the inter-activity delay from  $i$  to  $k$ . Without loss of

### Given variables and parameters:

$n$ :	number of activities in the process
$i, j, k$ :	indices for activities
$i \rightarrow j$ :	there is an information or material flow from activity $i$ to activity $j$
$\tau_i$ :	processing time of activity $i$
$\gamma_{ij}$ :	time delay between $i$ and $j$
$\delta$ :	delay cost per unit of time; assumed positive
$\Delta_{ij}$ :	cost of consolidation ( $j$ is combined into $i$ ); assumed positive, nonsymmetrical, and $\Delta_{ii} = 0, \forall i$

### Decision variables:

$f_i$ :	finish time of $i$
$s_i$ :	start time of $i$
$x_{ij}$ :	1 if activity $j$ is consolidated into activity $i$ , 0 otherwise
$y_{kij}$ :	1 if activity $i$ and $j$ are both consolidated into activity $k$ , 0 otherwise

Fig. 1. Summary of notations.

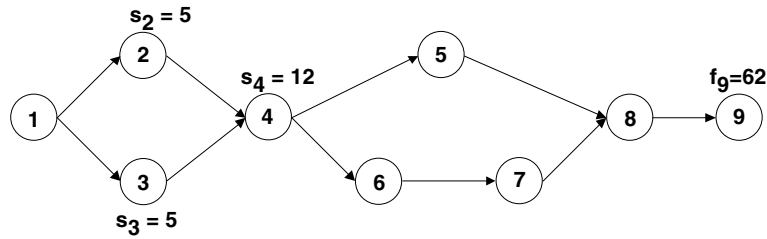


Fig. 2. Example project graph.

generality, all nodes can be labeled such that the project starts and ends at activities 1 and  $n$ , respectively, and if there is a directed arc from  $i$  to  $j$ , then  $i < j$ .

Real-valued decision variables  $s_i$  and  $f_i$  are the start and finish times of activity  $i$ . By assumption,  $s_1 = 0$  and  $f_n$  is the project duration (in this example, 62). If no activity is consolidated with any other activity, then the decision variable  $x_{ii} = 1$ ,  $x_{ji} = 0$ , and  $x_{ij} = 0$  for all  $j \neq i$ . If activity  $j$  is consolidated into activity  $i$ , then  $x_{ij} = 1$ ,  $x_{ii} = 1$  and  $x_{jk} = 0$  for all  $k$ , and the model will force  $s_i = s_j$  and  $f_i = f_j$ . The two scenarios, activity  $j$  consolidated into  $i$  ( $x_{ij} = 1$ ) or activity  $i$  consolidated into  $j$  ( $x_{ji} = 1$ ) are treated differently because the consolidation costs incurred in these two scenarios ( $\Delta_{ij}$  or  $\Delta_{ji}$ ) can be different. The model allows  $\Delta_{ij} \neq \Delta_{ji}$  because the consequence of activity  $j$  consolidated into  $i$  and activity  $i$  consolidated into  $j$  can be different. For example, the current workers on activities  $i$  and  $j$  may have different skills and hence would need different training to be able to do the consolidated activity. When two or more activities are consolidated into one, the choice of which activity is consolidated into which other activity is made to minimize related costs (for two activities, choose  $x_{ij} = 1$  if  $\Delta_{ij} < \Delta_{ji}$ ). Given the input cost parameters  $\Delta_{ij}$ 's for all activity pairs  $i$  and  $j$ , the cost of a particular consolidation pattern can be expressed as  $\sum_{i,j} x_{ij} \cdot \Delta_{ij}$ .

The problem of selecting the optimal consolidation pattern requires some benefit to compensate for the expenditure of the  $\Delta_{ij}$ . Let  $\delta$  be the cost per unit time of the project not being completed. Since  $f_n$  is the time required to complete the project, the problem of minimum cost project (MCP) can be expressed as the following:

MCP :

$$\min_{x_{ij}, f_n} \delta \cdot f_n + \sum_{i,j} x_{ij} \cdot \Delta_{ij}, \tag{1}$$

subject to:

$$\sum_i x_{ij} = 1 \quad \forall j, \tag{2}$$

$$\sum_k x_{jk} \leq n(1 - x_{ij}) \quad \forall i, j, \tag{3}$$

$$s_i + \sum_j x_{ij} \tau_j \leq f_i \quad \forall i, \tag{4}$$

$$f_i - M(1 - x_{ij}) \leq f_j \quad \forall i, j, \tag{5}$$

$$s_i + M(1 - x_{ij}) \geq s_j \quad \forall i, j, \tag{6}$$

$$x_{ki} + x_{kj} \geq 2y_{kij} \quad \forall k, \text{ arc } i \rightarrow j, \tag{7}$$

$$f_i + \left(1 - \sum_k y_{kij}\right) \gamma_{ij} - M \left(\sum_k y_{kij}\right) \leq s_j \quad \forall \text{ arc } i \rightarrow j, \tag{8}$$

$$s_1 = 0, \tag{9}$$

$$x_{ij}, y_{kij} \in \{0, 1\} \quad \forall i, j, k, \tag{10}$$

$$s_i, f_i \geq 0 \quad \forall i. \tag{11}$$

Constraint (2) states that activity  $j$  is consolidated with exactly one activity (could be itself). Constraint (3) states that if activity  $j$  is consolidated into some other activity  $i$ , then no other activity can be consolidated into  $j$ . Constraint (4) states that the finish time of activity  $i$  is the start time of activity  $i$  plus the sum of all activity times of activities that are in the same consolidated activity as activity  $i$ . Constraints (5) and (6) set the start and finish times of activity  $i$  to be the same as the start and finish times of activity  $j$  if  $x_{ij} = 1$  or  $x_{ji} = 1$ . When they are not consolidated, a large number  $M$  makes these constraints inactive.

$M$  needs to be a number greater than the project duration, and so could be set to the sum of the activity times and the arc delays, for example.

$y_{kij}$ 's are introduced to an otherwise non-linear constraint to make it linear. Constraint (7) ensures the integrity of definitions of  $x_{ki}$ ,  $x_{kj}$ , and  $y_{kij}$  by stating that if both  $i$  and  $j$  are consolidated into  $k$  ( $y_{kij} = 1$ ), then it must be the case that  $i$  is consolidated into  $k$  ( $x_{ki} = 1$ ) and  $j$  is consolidated into  $k$  ( $x_{kj} = 1$ ). Constraint (8) states that if the arc  $i \rightarrow j$  is not collapsed ( $\sum_k y_{kij} = 0$ ), then there is an arc delay  $\gamma_{ij}$  between finish time of activity  $i$  and start time of activity  $j$ , otherwise the large number  $M$  makes the constraint inactive. Note that while there are potentially  $n^3 y_{kij}$  variables, the number really only equals to the number of arcs in the project graph times the number of nodes. Constraints (7) and (8) are where our model differs from other project scheduling models: if  $i$  and  $j$  are consolidated into  $k$ , any activity that succeeds  $i$ ,  $j$ , or  $k$  can only start after the processing of all the

activities,  $i$ ,  $j$ , and  $k$ , are completed. In this way, the new composite activity  $k$  is atomic and does not release successor activities during the completion of the activity.

Constraint (10) defines the binary variables. Constraint (11) forces all start and finish times to be non-negative.

Our model (MCP) is NP-hard because the decision problem corresponding to MCP can be shown to be NP-Complete by a polynomial transformation of the Bin Packing Problem which is well known to be NP-Complete (see proof in Appendix A). We tried to solve samples of the problem with standard IP software, but found that the branch and bound techniques could not find solutions in a reasonable amount of time. The difficulty of this problem can be demonstrated with a couple of examples that show two types of difficulties that arise in finding good solutions: (1) consolidating activities may result in changes in the parallel structure of the graph and hence may

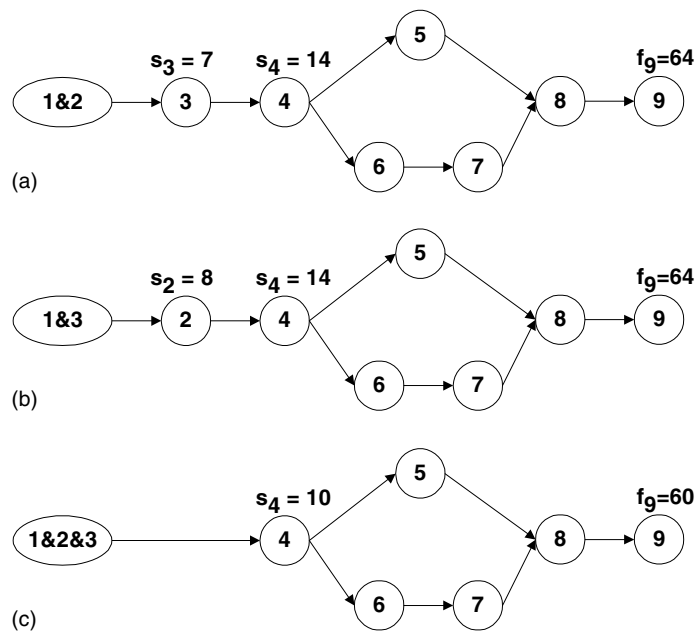


Fig. 3. Illustration of project duration changes when activities are consolidated: (a) new process structure after consolidating activities 1 and 2 in the process depicted in Fig. 2, resulting in longer cycle time; (b) new process structure after consolidating activities 1 and 3 in the process depicted in Fig. 2, resulting in longer cycle time; (c) new process structure after consolidating activities 1, 2 and 3 in the process depicted in Fig. 2, resulting in shorter cycle time.

or may not decrease the duration of the project, and (2) the optimal consolidation pattern may involve nodes that are not linked.

Fig. 3 denotes three consolidation patterns for the graph depicted in Fig. 2, showing that consolidation may or may not decrease the duration of the project. In Fig. 3a, consolidating activities 1 and 2 saves the delay between those two activities, but causes the entire project to take longer (64), due to the fact that  $f_1 = f_2 = 3$  and the start time of activity 4 is pushed back from  $s_4 = 12$  to  $s_4 = 14$ . If only activities 1 and 3 are consolidated (Fig. 3b), the start time of activity 4 is similarly pushed to 14. But when activities 1, 2, and 3 are all consolidated (Fig. 3c), the project duration is reduced to 60. The time savings from consolidating activities 1, 2, and 3 comes from the elimination of two delays ( $\gamma_{12}$  and  $\gamma_{13}$ ) so that now  $f_1 = f_2 = f_3 = 6$  and  $s_4 = 10$ .

Fig. 4 illustrates a case where consolidation can occur with nodes that are not connected. Let  $\delta = 1$ ,  $\tau_i = 1$ ,  $\gamma_{ij} = 1$  and  $\Delta_{ij} = 20$  except for

$\gamma_{23} = \gamma_{45} = 10$ , and  $\Delta_{42} = \Delta_{43} = \Delta_{45} = 1$ . From the original graph, collapsing the long delay arcs decreases the project duration, but at a large cost. The optimal solution is to consolidate activities not connected by arcs and to set  $x_{42} = x_{43} = x_{45} = 1$ . The cycle time of the consolidated graph is  $f_6 = 8$  and the total cost is  $f_6 * \delta + \Delta_{42} + \Delta_{43} + \Delta_{45} = 11$ . As such, heuristics based on the adjacency of nodes in the graph will not always find the optimal solution.

### 3. An implicit enumeration algorithm

A complete enumeration for this problem is impractical because of the factorial increase in the number of possible solutions (the size of the search space is  $\Omega = S(n, 1) + S(n, 2) + \dots + S(n, n)$ , where  $S(n, m)$  is the Stirling number of the second kind). Branching on the  $x_{ij}$ 's did not yield fruitful algorithm because that ignores the graph structure of the project and leads to the exploration of infeasible or cyclic graphs. Since consolidations can be made where there is no arc (as illustrated in Fig. 4, previously), branching just on the arcs will miss certain feasible solutions. We therefore modified the list of arcs and search on “enumeration pairs,” defined as a pair of nodes  $(i, j)$ ,  $i < j$ , where either there is an arc between the nodes or there is no directed path from  $i$  to  $j$ , but excludes pairs where there is directed path of length 2 or more from  $i$  to  $j$ . Those pairs are excluded because adding them to a node in the enumeration tree would result in either an infeasible solution or a consolidation pattern that can be derived by using other enumeration pairs. The Appendix provides the proof details that the enumeration described here will consider all feasible solutions.

For ease of exposition, call the two activities in  $(i, j)$  the  $i$ -activity and  $j$ -activity, respectively. The list of all enumeration pairs is then ordered so that if pair  $(i_1, j_1)$  comes before  $(i_2, j_2)$ , then either  $i_1 < i_2$  or  $i_1 = i_2$  and  $j_1 < j_2$ . For example, according to definitions above, the project graph in Fig. 5 produces this ordered list of enumeration pairs: (1, 2), (1, 3), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6). Notice that (1, 4) is omitted from the list because of the path from 1 to 2 to 4

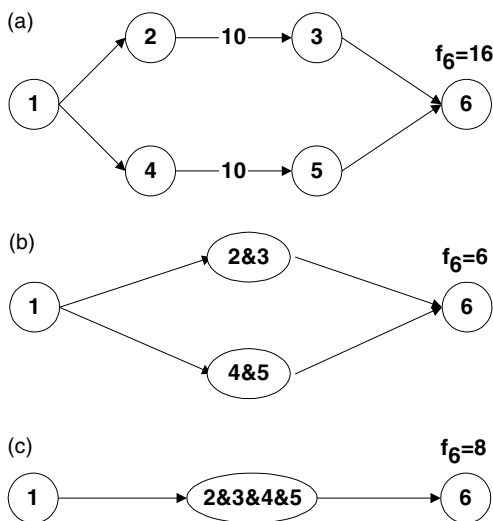


Fig. 4. Illustration of project duration changes when activities are consolidated: (a) original graph with project duration of 16 and an objective function value of 16; (b) new process structure after consolidating activities 2 with 3 and 4 with 5 to eliminate the long delays, with a project duration of 6 but an objective function value of 27; (c) new process structure after consolidating where there is no arc and where the project duration increases to 8, but with an objective function value of 11.

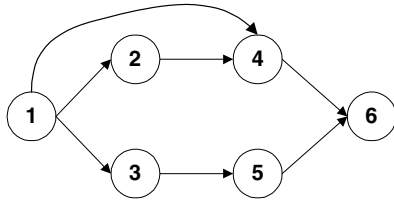


Fig. 5. A process example and ordered pairs.

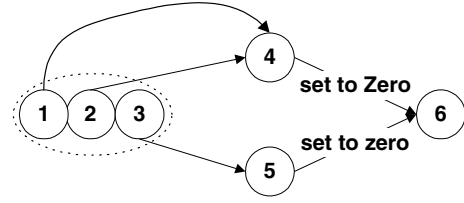


Fig. 6. Illustration of lower bound calculation.

(length of 2), but (2, 3), (2, 5), (3, 4) and (4, 5) are added because there is no directed path between the two nodes.

A node in the enumeration tree is defined by a set,  $S$ , of enumeration pairs that define the solution. The search starts with an empty set, which corresponds to the original graph. At any node in the search tree, each descendent is created by adding one enumeration pair not in  $S$ , but only a portion of the enumeration pair list needs to be considered to create immediate descendents. Let  $(i_{last}, j_{last})$  be the last pair added to  $S$ . The new pair  $(i, j)$  can be added to  $S$  to form the next node if and only if all following three conditions are met:

1.  $i > i_{last}$ ;
2.  $j$  does not coincide with any  $i$ -activity or  $j$ -activity in  $S$ ;
3. either  $i$  coincides with the largest  $j$ -activity in  $S$ , or  $i$  does not coincide with any  $j$ -activity in  $S$ .

For ease of discussion, if  $i$  coincides with the largest  $j$ -activity in  $S$ , call the descendent an *enlarging descendent*, otherwise, call it a *non-enlarging descendent*. The above three conditions together prevent either an infeasible node or one that exists in some other part of the enumeration tree. For example, in Fig. 5, if the current node is  $(1, 2) + (2, 4)$ , then  $(2, 5)$  is skipped because  $2 = i_{last}$  and adding  $(2, 5)$  would result in the same consolidation pattern as adding pair  $(4, 5)$ . Pair  $(3, 4)$  is skipped because 4 coincides with an existing  $j$ -activity and adding it would result in the same consolidation pattern as  $(1, 2) + (2, 3) + (3, 4)$ . The next pairs  $(3, 5)$ ,  $(4, 5)$ , and  $(4, 6)$  are all feasible enumeration pairs to add.

A lower bound when a non-enlarging descendent is created is computed by maintaining the  $\gamma_{ij}$ 's

for all the incoming and outgoing arcs for any groups of activities that are consolidated together and setting the rest of  $\gamma_{ij}$ 's to zero. For example, given the consolidation pattern,  $(1, 2) + (2, 3)$ , the lower bound for this node is the value of the objective function when arc delays on  $4 \rightarrow 6$  and  $5 \rightarrow 6$  are set to zero, as shown in Fig. 6. This lower bound cannot be used for enlarging descendents, because the cost of the graph at this level of the tree may increase when this pair is added, but deeper in the tree when other pairs are added, the cost may be reduced, as was demonstrated in Fig. 3. The lower bound for an enlarging descendent can be calculated by setting the consolidation cost to zero for the whole consolidated activity that the new node was added to. But this lower bound is too low to prune out nodes and is hence not used in the algorithm.

When  $S$  is the current node in the enumeration tree, and the lower bound computed at this time is worse than the current best solution, immediate non-enlarging descendents of  $S$  can be pruned, but care needs to be taken to still evaluate the non-immediate enlarging descendents of  $S$ . To accomplish this, when a branch of the tree is cut and an immediate enlarging descendant,  $T$ , of  $S$  is generated by adding a new pair  $(i, j)$ , the  $(i_{last}, j_{last})$  of  $T$  (which is the basis for the next round of tree-expanding) is not modified to  $(i, j)$ , but is kept as  $(i_{last}, j_{last})$  of  $S$ .

The enumeration tree is expanded and traversed using depth-first search. If the current node on the tree is evaluated to be infeasible to the original problem, all descendents of the current node on the search tree are fathomed. If the lower bound of the current node is evaluated to be worse than the current best solution, then all non-enlarging descendents (immediate or otherwise) are fath-

omed. At the next iteration, an enlarging descendent is always investigated since the lower bound cannot fathom these nodes. When the end of the enumeration list is encountered, the search backtracks to the parent node (by deleting the last enumeration pair used) and creating a new node with the next enumeration pair on the list.

**Theorem.** *The above implicit enumeration procedure guarantees the optimal solution.*

**Proof.** The theorem is a direct result of Lemmas 4 and 5 in Appendix A.  $\square$

#### 4. Heuristic solution method: Simple greedy heuristic and a tabu search

Two heuristics are considered. The first is a simple greedy heuristic. At each step, the program identifies the critical path and chooses a pair-wise consolidation on the critical path that would reduce the total cost of the project at this step. If no reduction is possible, the heuristic terminates.

The second heuristic is a tabu search (Glover and Laguna, 1993) that is based on the same enumeration pairs defined for the implicit enumeration. As before, let  $S$  denote the current solution, which is a list of enumeration pairs. The cost function,  $C$ , is the objective function of problem MCP, but for the search, there is a related value function,  $V$ , which is  $C$  augmented with a penalty function based on which enumeration pairs are in  $S$ . The search works by making a *move* from one solution to another in the *neighborhood*. A valid move either adds to  $S$  one enumeration pair not presently in  $S$  or removes one enumeration pair from  $S$ . As with the enumeration algorithm, when adding a new pair to  $S$ , only feasible pairs discussed in Section 3 need to be considered. A neighborhood is the set of solutions that can be derived from the current solution by making each valid move. As the search progresses, certain moves will be made tabu (prohibited) to direct the search toward the global optimum.

The length of the tabu list needs to be carefully chosen: If it is too long, there are few feasible moves to make in the neighborhood search, and if

it is too short, local optima are revisited. Since there are two types of valid moves, moves that are tabu are implemented as two separate lists, one for additions to  $S$  and one for removing from  $S$ . Early in the search, most moves should be available, so the initial length of the tabu lists are  $\sqrt{n}$  for removal moves (where  $n$  is the number of nodes in the project graph) and  $\sqrt{2m}$  for addition moves (where  $m$  is the total number of enumerations pairs generated for the original graph). Later in the search, the lengths of the tabu lists are gradually increased to twice their original sizes in order to diversify the search.

To diversify the search further so that local optima will be escaped, the search also maintains a “memory”, defined as the number of times an enumeration pair is added to make a move. Pairs with higher frequency are those that are more effective in reducing the project duration. In order not to be trapped with only high frequency pairs, the value function is defined as  $C + \text{frequency} * 2\delta/n$ , where  $2\delta/n$  is the penalty cost chosen after experimenting with the tabu search. It is proportional to cycle time cost because objective function depends heavily on  $\delta$ . Dividing the penalty by  $n$  means that early in the search, “good” pairs will incur a low penalty, but later in the search as the frequency increases, those pairs will become more and more costly, forcing the tabu heuristic to try different pairs, potentially climbing hills to escape local optima. The multiplier 2 affects the speed and accuracy of the tabu search: If the penalty is too small, the search will cycle through the same solution many times, but if it is too big, the penalty might push the search away from an area of the solution space too quickly (and perhaps miss the global optimum).

There are four solution versions that are maintained at all times during the search:  $S(\text{best})$  is the current solution with the overall minimum cost,  $S(\text{now})$  is the basis for the neighborhood search,  $S(\text{save})$  is the best move discovered during the neighborhood search and  $S(\text{trial})$  is one of the possible neighborhood moves. For each solution, maintain the objective function value  $C(*)$  and the value function  $V(*)$ . The tabu search starts with the original graph structure, at which  $S(\text{best}) = S(\text{now}) = \emptyset$ .

```

Begin
  Set  $S(\text{best}) = S(\text{now}) = \emptyset$ 
   $C(\text{best}) = \infty$ 
  Set IterationCount = 0
  Repeat
    Set  $S(\text{save}) = S(\text{now})$ 
     $V(\text{save}) = \infty$ 
    For each  $(i, j)$  in enumeration pair list
      If  $(i, j)$  is not in  $S(\text{now})$ , then add  $(i, j)$ 
      to  $S(\text{now})$  to obtain  $S(\text{trial})$ 
      Else remove  $(i, j)$  from  $S(\text{now})$  to obtain
       $S(\text{trial})$ ;
      If  $C(\text{trial}) < C(\text{best})$ , then
        set  $S(\text{best}) = S(\text{trial})$ 
        set IterationCount = 0
      If the move that created  $S(\text{trial})$  is not
      tabu and  $V(\text{trial}) < V(\text{save})$  then
        set  $S(\text{save}) = S(\text{trial})$ 
    End for loop for enumeration pair list
    Set  $S(\text{now}) = S(\text{save})$ 
    Increment IterationCount
    If  $S(\text{save})$  was obtained by adding a pair,
    then
      add the move to Add Tabu list
      update its frequency
    Else
      add the move to Remove Tabu list
      update its frequency
    Until IterationCount exceeds a certain speci-
    fied limit
    Return  $S(\text{best})$  as solution
  End

```

Setting  $S(\text{best}) = S(\text{trial})$  when  $C(\text{trial}) < C(\text{best})$  and setting  $S(\text{save}) = S(\text{trial})$  when the move that created  $S(\text{trial})$  is not tabu and  $V(\text{trial}) < V(\text{save})$  allow us to look at tabu moves for potential global optimum, but ignore it in the local search. In addition, using the value function for comparing neighborhood moves allows the tabu heuristic to move to solutions that have a higher objective function value  $C(*)$  and climb out of local optima.

## 5. Experimental design

In this section, the performance of the tabu search heuristic and the greedy heuristic under

different parameter scenarios are examined. The tabu search can be allowed to complete any number of neighborhood searches prior to terminating. Two different versions were tested: the first terminates after 500 iterations without an improvement in the best solution (Tabu500), and the second terminates after only 100 iterations (Tabu100). When the problem size allows, the tabu search and the greedy heuristic can be compared to the optimal solution found by the implicit enumeration. For larger problems, the solutions obtained from the heuristics are compared with two “base case” solutions: the project with no consolidations (Base) and the project with all the activities consolidated as a single activity (1-node).

Graph examples were randomly generated to test the heuristic on problems with different characteristics. For each setting of the parameters, 50 different replications were solved. The factorial design consisted of the following five factors:

(1) The number of nodes in the graph was varied over six settings: 12, 15, 18, 21, 24, and 27 activities. For the first three, the enumeration could find optimal solutions, but the last three had state spaces too large to find optimal solutions within one hour. All the factor settings for problems of 18 or fewer activities were examined and then used to determine problems with “hard” parameter settings. Larger problems of 21, 24, and 27 nodes were solved only for a partial factorial design.

(2) The number of arcs in the graph was varied across three settings (high, medium, and low). Since the number of activities affects how dense a graph is, this factor is defined by the percentage of the possible  $(n-1)(n-2)/2$  arcs (those that would be present if every activity were connected to each downstream activity). Low density graphs had 12.5% of the possible arcs, medium density graphs had 27.5% of the possible arcs, and high density graphs had 42.5% of the possible arcs. Each arc was created by randomly selecting the two activities for the arc, so more than one arc could connect any two activities (the longest delay determines the critical path). It is possible that an activity could remain unconnected after the random arc generation, and in that case, arcs were added to connect the activity to the graph. The actual number of arcs in the graphs could therefore vary slightly around the target value.

(3) The cost of combining two activities ( $\Delta_{ij}$ ) was randomly drawn from a uniform distribution that had either a low mean and low variance (10–50) or a high mean and high variance (10–100). The cost of consolidating activity  $i$  into activity  $j$  is generated separately from the cost of consolidating activity  $j$  into activity  $i$ , so the cost matrix is not symmetric.

(4) The time required for each activity ( $\tau_i$ ) was randomly drawn from a uniform distribution that either had a low mean and low variance (10–50) or a high mean and high variance (10–100).

(5) The arc delays between activities ( $\gamma_{ij}$ ) were randomly drawn from a uniform distribution that either had a low mean and low variance (0–10) or a high mean and high variance (0–50). This results in some problems where the delay times are similar in magnitude to the activity times, but for most of the problems the delays are much smaller than the activity times.

(6) The cost per unit time of the project ( $\delta$ ) was randomly generated from a uniform distribution with a range from 10 to 100.

Every combination of the arc/cost/time factors was included for graph sizes of 12, 15, and 18 nodes (1200 graphs for each problem size). For these problems, the combination of low time mean and variance and high delay mean and variance resulted in the most difficult problems (where the heuristic failed to find the optimal solution), so those two parameters were fixed for the graphs of size 21, 24, and 27 nodes while varying the other six factor combinations (three arc density settings, two cost settings), resulting in 300 graphs for each problem size. A total of 4500 problems were generated. Each problem is solved with Tabu500, Tabu100, base scenario, and 1-node scenario. The implicit enumeration was not used for the last 900 problems of size 21, 24, or 27 nodes.

For the implicit enumeration, the average CPU time (an Intel Celeron processor) shows explosive growth in Table 1. Both versions of tabu search demonstrate more linear increases in CPU time for the small problems, but as the problem size increases, the much larger neighborhood at each iteration of the tabu search causes the CPU time to increase more quickly for Tabu500.

Table 1  
CPU times for Tabu100, Tabu500, and enumeration algorithms

$n$	Average CPU seconds for Tabu500	Average CPU seconds for Tabu100	Average CPU seconds for implicit enumeration
12	1.52	0.35	0.57
15	5.32	1.21	14.45
18	17.38	4.12	382.37
21	42.03	12.00	–
24	143.96	41.34	–
27	497.01	160.11	–

In terms of solution accuracy, Table 2 documents the number of times each heuristic found the optimal solution out of 3600 problems solved with the implicit enumeration, average solution inaccuracy for all 3600 problems and for those problems that tabu search did not find the optimal solution, and worst case scenario for each problem size.

The simple greedy algorithm performed worse than the tabu searches, as expected. Although the average solution inaccuracy for the greedy algorithm is acceptable, the worst-case scenario missed the optimal solution by 23.9%, 19.0%, or 16.2% for problems of size 12, 15, and 18, respectively. Worst-case scenario occurs when the optimal solution involves consolidating activities across parallel workflows, as shown in Fig. 3, or consolidating activities that do not form a subgraph, as shown in Fig. 4. Note that even though Tabu500 performed better than Tabu100 in general, the worst case scenario for Tabu500 happened to be worst than that of Tabu100 for 18-node problems. This is because Tabu500 may not take the same path when visit the solution space.

Table 3 further illustrates the performance between the greedy algorithm and the tabu searches. Here, the greedy algorithm is compared with both Tabu500 and Tabu100 to obtain number of cases in which the greedy heuristic found the same solution as the tabu search, did better than tabu search, or did worse than tabu search. The objective function values (total costs) are compared to see how much worse the greedy heuristic is. Again, on average, the simple greedy algorithm’s performance is close to that of tabu search, but the

Table 2  
Solution accuracy for Tabu500, Tabu100, and Greedy heuristics

Solution method	Num of optimal solutions found	Average error rate (all 3600 problems)	Average error rate when optimal solutions not found	Worst case problems		
				12-Node	15-Node	18-Node
Tabu500	3432	0.05%	1.16%	6.96%	5.36%	8.47%
Tabu100	3213	0.11%	1.05%	6.26%	7.03%	4.58%
Greedy	2252	1.06%	2.83%	23.90%	19.00%	16.20%

Table 3  
Comparing the greedy heuristic to Tabu500 and Tabu100

Solution method	Greedy solution compared to tabu			Average error rate when greedy obtained worse solution		
	Same	Better	Worse			
Tabu500	2583	0	1917	2.93%		
Tabu100	2788	1	1711	3.01%		
	Worst case difference from tabu solution					
	12-Node	15-Node	18-Node	21-Node	24-Node	27-Node
Tabu500	23.91%	17.76%	16.20%	17.00%	12.74%	11.34%
Tabu100	23.90%	17.76%	16.20%	17.00%	10.70%	9.75%

worst-case scenario illustrates that the tabu search was able to guarantee better solutions.

For all problems, the tabu search and greedy heuristic can be compared to the cost of leaving the graph unchanged (Base), or at the other extreme, collapsing all the activities into a single sequential chain (1-node). Table 4 shows the average gains (1-heuristic\_obj/base\_obj) to be made by modifying the project graph.

The improvements of consolidation over the base case are most prominent for those cases that have larger delays. In our numerical examples, when inter-activity delays are drawn from uniform distribution of (0–50), the average savings over the base model is 20.6% if the activity times are drawn

from uniform (10–50) and 13.7% if the activity times are drawn from uniform (10–100). However, when inter-activity delays are drawn from uniform (0–10), these two numbers are 5.1% and 3.0%, respectively. This result is consistent with the type of the projects our model is designed to improve: namely those where inter-activity delays are significant compared to activity processing times. In the next section, we use an example to demonstrate the savings obtained from consolidation. In that example, activity times were drawn from uniform (10–100) and inter-activity delays were drawn from uniform (0–50). Tabu search results in a project graph that is 16.4% improvement in objective function value over the original project graph.

Table 4  
Percentage improvement

	Improvement versus base	Improvement versus 1-node
Tabu500	12.6%	25.3%
Tabu100	12.5%	25.2%
Greedy	11.6%	24.2%

## 6. An example

An 18-activity problem (Appendix B) was selected from the set of problems solved to illustrate how the different heuristics find solutions other than the optimal. The graph structure is one of the

high arc-density problems, as if the project were broken into small activities with much interdependence among the activities. Before activity consolidation, the project duration is 887, the objective function value is 15,520 and the graph looks like Fig. 7.

The greedy heuristic will consider consolidating most activities in this graph because of the high density of arcs, but will not directly consider pairs that are not connected, such as 5 and 7. This heuristic reduces the project duration to 746 with an objective function value of 13,207 and a graph that looks like Fig. 8.

Notice that there is now a precedence arc between activities 5 and 7, because the heuristic first

combines activities 7 and 8. At the end of the algorithm, pair-wise combinations of any of the composite nodes will make the objective function worse, but that does not imply that an optimal solution has been found. The Tabu100 heuristic will search possible combinations more completely and results in a graph with a duration of 738, an objective function value of 13,119, and the graph structure in Fig. 9.

The large initial activity can only be found when many consolidations are considered simultaneously. The Tabu500 heuristic has more iterations, and so the penalty function takes on different values and allows possible solutions not identified as “improving” by the Tabu100 search.

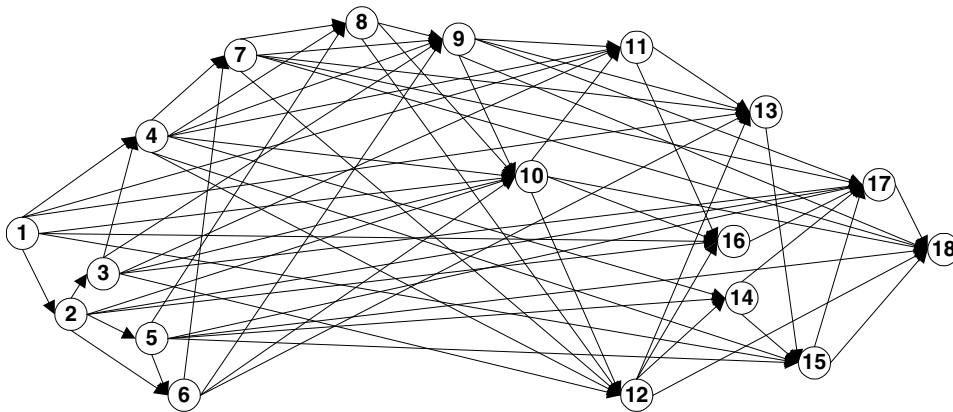


Fig. 7. The original project structure of the 18-activity example problem.

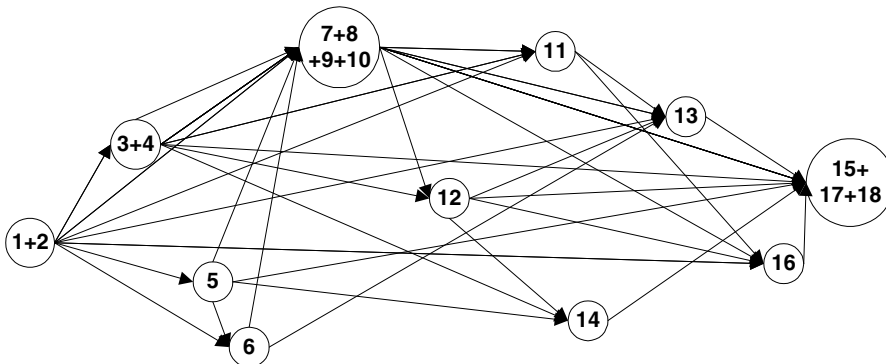


Fig. 8. Result obtained by applying the greedy heuristic.

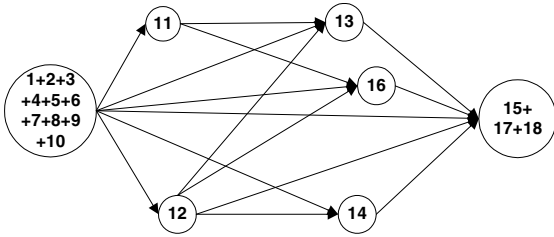


Fig. 9. Result obtained by applying Tabu 100 Heuristic.

The solution discovered reduces the project duration to 728 with an objective function value of 12,969 and the graph structure in Fig. 10.

Finally, the implicit enumeration was applied to the problem and found the optimal solution with a project duration of 728, an objective function value of 12,942 and a slightly different graph structure shown in Fig. 11.

The difference between the last two solutions consists of whether to combine activities 5 and 6 and whether to consolidate activity 18 with activities 15 and 17. Notice how the precedence relationships change subtly between the two solutions, so that tabu lists and the penalty function could prevent the Tabu500 search from finding the last few moves to the optimal solution.

### 7. Conclusions

This paper introduces consolidation of activities as a vehicle for project planning. Consolidation eliminates inter-activity delays due to material and information flows, but reduces the amount of parallel processing possible in the graph (consolidated activities are performed sequentially). Further, consolidation redesigns the activities and

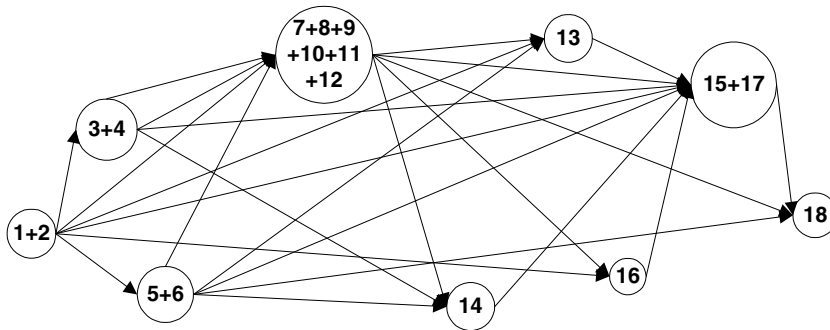


Fig. 10. Result obtained by applying the Tabu 500 Heuristic.

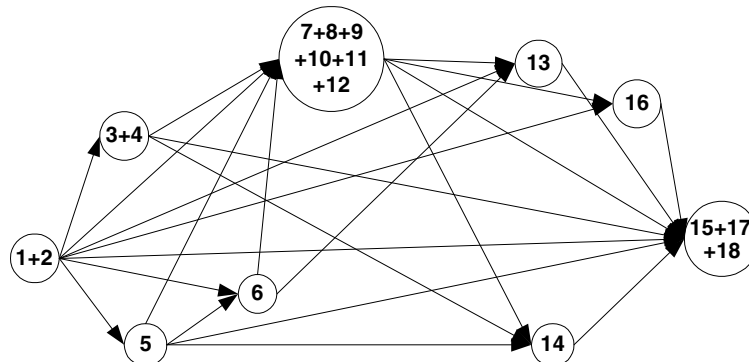


Fig. 11. Result obtained by applying the implicit enumeration algorithm.

precedence relationships—combing multiple activities into one atomic activity implies that precedence relationships apply to all of the combined activities.

The enumeration and heuristics provided and serve to explore the nature of the problem. The enumeration state space explodes quickly and greedy heuristics are likely to provide poor worst case behavior. Techniques like the tabu heuristic are required to solve problems where consolidation causes many changes in the graph, and where a consolidation may only yield benefits in conjunction with many others. Both the tabu and the greedy heuristics are shown to be effective for the problem, but it is possible that further work could improve processing efficiency and solution accuracy of these heuristics. In addition, large projects can often be decomposed into smaller ones, and so our algorithms could provide the basis for solving much larger problems than those explored here.

Our future research on consolidation will be to model its use during project execution, for example, to alleviate the workload at a bottleneck activity. To apply consolidation in this scenario, the basic principle formulated here can be combined with resource-constrained scheduling problems, which would add the aspect of activity queuing in front of busy resources.

**Appendix A. Proof that MCP is NP-hard**

*Minimum Cost Project (MCP):*

Instance: Description of the problem as in the text.

Question: Is there a project plan with total cost less than  $B$ ?

*Bin Packing Problem (BPP):*

Instance: Finite set  $U$  of items, a positive integer size  $s(u)$  for each item  $u \in U$ , bin capacity positive integer  $B$ , and number of bins  $K$ .

Question: Is there an assignment of items to bins such that the sum of weights of items in each bin is no more than  $B$ ?

**Proof.** By component design, we will convert all instances of BPP to an instance of MCP. It is known that BPP is NP Complete. Construct a graph with the following vertices and edges:

Vertices:

- Start vertex  $s$
- Finish vertex  $f$
- Bin node  $i_1, \dots, i_K$
- A node  $j$  for each activity with activity time equal to the size

Edges:

- An edge from  $s$  to each bin node with  $\gamma = 0$  and  $\Delta = M$ .
- An edge from each bin node to each activity node with  $\gamma = M$  and  $\Delta = 0$ .
- An edge from activity node to the finish node with  $\gamma = 0$  and  $\Delta = M$ . A constant  $M \geq B$  and  $\delta = 1$ .

The construction is polynomial. First we show that any solution to the created MCP is a solution to the BPP.

Each of the activity nodes will be consolidated into a bin node. If this is not the case then there is a hand off cost of  $M$  between a bin node and the activity node that is not consolidated with any bin node. This exceeds  $B$ . The sum of activity weights consolidated into each bin node has to be less than  $B$  or else the project cost will exceed  $B$ . Hence, we can create a solution for the BPP by considering the activities consolidated into each of the  $K$  bin nodes as the items in the bins.

Constructing a solution for the MCP from a BPP solution is easy: Consolidate the activities that go into each bin into the corresponding bin node. Since maximum weight of the activities is less than  $B$ , the cost of the consolidated process is also less than  $B$ .

We have exhibited that for each instance of the BPP, there is a one to one mapping between the solutions to the BPP and the constructed MCP. Hence the MCP is NP Complete.

Since the decision problem is NP complete, the optimization problem is NP Hard.  $\square$

**Lemma 1.** Let  $n$  be the number of activities in a graph. If activity pair  $(i, j)$ ,  $i < n$  and  $j \leq n$ , is not an enumeration pair (i.e., there is an indirect path from  $i$  to  $j$ ), then there exists an arc  $i \rightarrow k$ , such that  $(i, k)$  is an enumeration pair, and there is a path from  $k$  to  $j$ .

**Proof.** Since  $(i, j)$  is not an enumeration pair, there is an indirect path from  $i$  to  $j$ . Then  $\exists t_1$ , such that  $i \rightarrow t_1$  is an arc, and there is a path from  $t_1$  to  $j$ . If  $(i, t_1)$  is an enumeration pair, the proof is complete. Otherwise, there is an indirect path from  $i$  to  $t_1$ . Then  $\exists t_2$  such that,  $i \rightarrow t_2$  is an arc and there is path from  $t_2$  to  $t_1$ . Since there is a path from  $t_1$  to  $j$ , so there is a path from  $t_2$  to  $j$ . If  $(i, t_2)$  is an enumeration pair, the proof is complete. Otherwise, the process can be repeated. Since there are finite number of arcs starting with activity  $i$ , eventually there will be  $t_s$ , such that  $i \rightarrow t_s$  is an arc and an enumeration pair, and there is a path from  $t_s$  to  $j$ .  $\square$

**Definition 1 (Feasible group).** A group of nodes  $\{i_1, i_2, \dots, i_n\}$  is called a *feasible group* if there does not exist a node  $k$  of the project graph, such that  $k$  is not in the group, and at least one activity in the group precedes  $k$ , but  $k$  precedes another activity in the group.

**Lemma 2.** If  $\{i_1, i_2, \dots, i_n\}$  is a feasible group, where  $i_1 < i_2 < \dots < i_n$ , then both groups  $\{i_2, \dots, i_n\}$  and  $\{i_1, i_2, \dots, i_{n-1}\}$  are feasible groups.

**Proof.** If  $\{i_2, \dots, i_n\}$  is not a feasible group, then  $\exists k \notin \{i_2, \dots, i_n\}$ , such that  $i_p$  precedes  $k$  and  $k$  precedes  $i_q$  for some  $i_p, i_q \in \{i_2, \dots, i_n\}$ . Since  $i_1 < i_p$  then  $k \neq i_1$  (otherwise,  $i_p$  cannot precede  $k$ ). So we found  $k \notin \{i_1, i_2, \dots, i_n\}$ ,  $i_p, i_q \in \{i_1, i_2, \dots, i_n\}$ ,  $i_p$  precedes  $k$  and  $k$  precedes  $i_q$ , which means that  $\{i_1, i_2, \dots, i_n\}$  is also not feasible. This is a contradiction. The proof that  $\{i_1, i_2, \dots, i_{n-1}\}$  is also a feasible group is similar.  $\square$

**Lemma 3.** If any group consisting of three or more activities,  $\{i_1, i_2, \dots, i_n\}$ , where  $i_1 < i_2 < \dots < i_n$ , is a feasible group, then  $(i_1, i_2), \dots, (i_{n-1}, i_n)$  are enumeration pairs.

**Proof.** Proof by induction method.

(i) Prove that if any group consisting of three activities,  $\{i_1, i_2, i_3\}$ , where  $i_1 < i_2 < i_3$  is a feasible group, then both  $(i_1, i_2)$  and  $(i_2, i_3)$  are enumeration pairs.

Suppose  $(i_1, i_2)$  and  $(i_2, i_3)$  are not both enumeration pairs. Without the loss of generality, let us suppose  $(i_1, i_2)$  is not an enumeration pair. By Lemma 1,  $\exists$  arc  $i_1 \rightarrow k$ , such that  $(i_1, k)$  is an enumeration pair, and there is a path from  $k$  to  $i_2$ . Certainly,  $k < i_2 < i_3$ , hence  $k$  is not in the original group. Since  $i_1$  precedes  $k$ ,  $k$  precedes  $i_2$ , and  $k$  is not in the group, then the group is an infeasible group. Contradict the assumption.

(ii) Assume that if any group consisting of  $n > 3$  activities,  $\{i_1, i_2, i_3, \dots, i_n\}$ , where  $i_1 < i_2 < i_3 \dots < i_n$ , is a feasible group, then  $(i_1, i_2), (i_2, i_3), \dots, (i_{n-1}, i_n)$  are all enumeration pairs.

(iii) Prove that if any group consisting of  $n + 1$  activities,  $\{i_1, i_2, i_3, \dots, i_n, i_{n+1}\}$ , where  $i_1 < i_2 < i_3 \dots < i_n < i_{n+1}$  is a feasible group, then  $(i_1, i_2), (i_2, i_3), \dots, (i_{n-1}, i_n)$ , and  $(i_n, i_{n+1})$  are all enumeration pairs.

Since  $\{i_1, i_2, i_3, \dots, i_n, i_{n+1}\}$  is a feasible group, by Lemma 2,  $\{i_1, i_2, \dots, i_n\}$  is a feasible group. Then by the previous assumption,  $(i_1, i_2), (i_2, i_3), \dots, (i_{n-1}, i_n)$  are all enumeration pairs. Similarly,  $\{i_2, i_3, \dots, i_{n+1}\}$  is a feasible group and hence  $(i_2, i_3), \dots, (i_n, i_{n+1})$  are all enumeration pairs. (iii) is proved.  $\square$

**Lemma 4.** If the lower bound is not applied, then the procedure described in Section 3 would visit all possible feasible solutions.

**Proof.** Let  $S$  be a feasible solution.

Case 1:  $S$  contains no group.

If the feasible solution,  $S$ , does not contain any groups, then  $S$  represents the initial graph with no consolidations. This feasible solution is visited at the root of the enumeration tree.

Case 2:  $S$  contains only one feasible group.

Proof by induction method:

(i) First prove that if  $S$  has only one feasible group that consists of two activities  $\{i, j\}$ , then  $S$  is visited.

Without loss of generality, assume  $i < j$ .  $(i, j)$  must be an enumeration pair, otherwise, there is an indirect path from  $i$  to  $j$ . Then by Lemma 1  $\exists k$ ,  $i \rightarrow k$  is an arc, and there is a path from  $k$  to  $j$ . So,  $i$  precedes  $k$  and  $k$  precedes  $j$ . Since  $k \notin \{i, j\}$ ,  $\{i, j\}$  is not a feasible group. This is a contradiction. Hence  $(i, j)$  is an enumeration pair. Solution  $S$  will be visited at enumeration tree with depth = 1.

(ii) Assume that if  $S$  has only one feasible group,  $\{i_1, i_2, \dots, i_m\}$ , that consists of  $m$  activities,  $3 \leq m \leq n - 1$ , then  $S$  is visited.

(iii) Prove that if  $S$  has only one feasible group,  $\{i_1, i_2, \dots, i_m, i_{m+1}\}$ , that consists of  $m + 1$  activities,  $3 \leq m \leq n - 1$ , then  $S$  is visited. By Lemma 3,  $\{i_1, i_2\}, \dots, \{i_m, i_{m+1}\}$  are all enumeration pairs. At the enumeration tree of depth =  $m$  where  $\{i_1, i_2, \dots, i_m\}$  is visited, the enumeration pair  $\{i_m, i_{m+1}\}$  is a candidate pair to add since  $i_m > i_{\text{last}} = i_{m-1}$  and  $i_m$  corresponds to the largest  $j$ -activity in the group, and  $i_{m+1}$  does not coincide with any  $i$ -activity or  $j$ -activity in the current solution. So  $\{i_m, i_{m+1}\}$  can be added in enumeration tree at depth =  $m + 1$ . So  $\{i_1, i_2, \dots, i_m, i_{m+1}\}$  will be visited.

Case 3: The feasible solution has  $K$  feasible groups,  $K \geq 2$ .

Let  $\{i_1^k, i_2^k, \dots, i_{m_k}^k\}$ ,  $i_1^k < i_2^k < \dots < i_{m_k}^k$ ,  $k \in \{1, \dots, K\}$  be a feasible group in  $S$ . According to Lemma 3,  $(i_1^k, i_2^k), (i_2^k, i_3^k), \dots, (i_{m_k-1}^k, i_{m_k}^k)$  are all feasible pairs. Order all feasible pairs from all groups in the same order as used in the original enumeration pairs list and denote the resulting list  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_m}, i_{q_m})$ , where  $m = \sum_{k=1}^K m_k$ . We prove, by induction method, that  $S$ , denoted by  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_m}, i_{q_m})$ , will be visited.

(i) Proved that if  $(i_{p_1}, i_{q_1})$  denotes the current solution, then it will be visited.

The prove is the same as in (i) of Case 2.

(ii) Assume that the solution denoted by  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_{m-1}}, i_{q_{m-1}})$  will be visited.

(iii) Prove that the solution denoted by  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_m}, i_{q_m})$  will be visited.

At the enumeration tree of depth =  $m - 1$  where  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_{m-1}}, i_{q_{m-1}})$ , is visited, if adding the enumeration pair  $(i_{p_m}, i_{q_m})$  does not enlarge any existing group,  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_m}, i_{q_m})$

will be visited when the lower bound is not applied. If adding  $(i_{p_m}, i_{q_m})$  enlarges an existing group  $k$ , then  $(i_{p_m}, i_{q_m})$  is the last pair,  $(i_{m_k-1}^k, i_{m_k}^k)$ , of group  $k$ . According to the proof of Case 2, a solution denoted with all pairs in group  $k$  consists of only one group and will be visited. We derive that  $i_{m_k-1}^k$  coincides with the largest node in group  $k$  at that time, and  $i_{m_k}^k$  does not coincide with any node in group  $k$ . Since any node and any feasible pair can appear in at most one group and  $(i_{p_m}, i_{q_m})$  is  $(i_{m_k-1}^k, i_{m_k}^k)$ , so  $i_{p_m}$  corresponds to the largest  $j$ -activity in a group, and  $i_{q_m}$  does not coincide with any  $i$ -activity or  $j$ -activity in the current solution. So  $(i_{p_m}, i_{q_m})$  can be added in enumeration tree at depth =  $m$ . So  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_m}, i_{q_m})$  will be visited.  $\square$

**Lemma 5.** *The lower bound does not eliminate the optimal solution.*

**Proof.** Suppose the current solution,  $S$ , consists of  $k$  groups,  $k \geq 0$ . Similar to the proof of Lemma 4, Let  $\{i_1^k, i_2^k, \dots, i_{m_k}^k\}$ ,  $i_1^k < i_2^k < \dots < i_{m_k}^k$ ,  $k \in \{1, \dots, K\}$  be a feasible group in  $S$ . According to Lemma 3,  $(i_1^k, i_2^k), (i_2^k, i_3^k), \dots, (i_{m_k-1}^k, i_{m_k}^k)$  are all feasible pairs. Order these pairs in the same order as used in the original enumeration pairs list and denote the resulting list  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_m}, i_{q_m})$ , where  $m = \sum_{k=1}^K m_k$ . Further denote the current best solution as  $f_{\text{best}}\delta + c_{\text{best}}$ . Let  $f_b$  be the finish time by keeping all arc delays going to or coming out of any groups in  $S$  and setting other arcs to zero arc delays. The lower bound thus obtained is then  $f_b\delta + c_s$ . Currently, the lower bound is active, so  $f_b\delta + c_s > f_{\text{best}}\delta + c_{\text{best}}$ .

Let  $T$  be a descendant of  $S$  (immediate or non-immediate) and a feasible solution. Denote the finish time of  $S$  and  $T$  as  $f_s$  and  $f_t$ , respectively, and the total combination cost as  $c_s$  and  $c_t$ , respectively.

If  $T$  consists of the same  $k$  groups and one or more other groups (i.e., a non-enlarging descendent), then  $f_t \geq f_b$ , and  $c_t \geq c_s$ . When the lower bound is active (i.e.,  $f_b\delta + c_s > f_{\text{best}}\delta + c_{\text{best}}$ ) we have  $f_t\delta + c_t \geq f_b\delta + c_s > f_{\text{best}}\delta + c_{\text{best}}$ . So  $T$  cannot be the optimal solution and hence eliminating  $T$  does not eliminate the optimal solution.

If  $T$  contains a group that is an enlargement of one or more existing groups, since  $T$  is a descendent of  $S$ , all pairs used to construct  $S$  are also used to construct  $T$ . Denote the additional enumeration pairs used to construct  $T$  as the ordered pair list  $(i_1, j_1), \dots, (i_s, j_s), s \geq 1$ .

Case 1: Only one of  $(i_1, j_1), \dots, (i_s, j_s), s \geq 1$  enlarges an existing group. Denote this pair  $(i_t, j_t)$ .

Clearly,  $(i_{p_1}, i_{q_1}), (i_{p_2}, i_{q_2}), \dots, (i_{p_m}, i_{q_m})$ , plus pair  $(i_t, j_t)$  denote an immediate descent,  $G$ , of  $S$ . According to the procedure described in Section 3,

$G$  is visited and  $(i_{\text{last}}, j_{\text{last}})$  of  $G$  is the same as that of  $S$ . Therefore, since  $(i_1, j_1), \dots, (i_s, j_s), s \geq 1$  are candidate pairs at  $S$ , they are candidate pairs at  $G$ . So  $T$  is a non-enlarging descendent of  $G$ . If  $T$  is the optimal solution, then the lower bound obtained at  $G$  is not active, hence  $T$  will be visited.

Case 2: Two or more pairs of  $(i_1, j_1), \dots, (i_s, j_s), s \geq 2$ , enlarge an existing group.

By induction method, using a proof similar to that of Case 1, we can conclude that if  $T$  is an optimal solution, then  $T$  will be visited.  $\square$

## Appendix B. Data to the example

[Number of nodes = 18, $\delta = 17.49$ ]											
[Node	$i$	Activity time $\tau_i$		Consolidation costs $\Delta_{ij} (j = 1, \dots, 18)$							
[Node	1	41.94		0	35.97	36.58	44.42	39.32	16.89	24.91	16.13
44.12	12.69	16.91	31.98	31.06	38.49	29.67	13.2	38.87	10.57		
[Node	2	25.46		31.26	0	15.71	36.96	36.92	43.43	12.28	38.08
31.08	38.4	24.94	17.34	11.6	31.98	48.84	21.79	19.81	32.63		
[Node	3	34.84		32.75	14.28	0	17.16	13.63	26.41	48.93	36.3
36.9	44.48	21.15	10.39	12.39	34.78	29.95	14.92	36.07	21.91		
[Node	4	41.58		16.23	11.45	47.85	0	19.99	17.09	12.71	42.57
10.36	12.35	17.79	24.7	41.8	34.55	11.59	32.1	23.2	49.55		
[Node	5	30.29		22.53	23.78	22.17	28.14	0	24.27	21.24	31.77
43.34	16.69	19.91	37.92	28.39	20.97	18.81	16.04	19.61	40.63		
[Node	6	54.63		22.94	48.99	12.23	25.84	32.54	0	37.3	45.93
18.8	21.45	17.24	36.18	27.9	24.58	46.52	42.3	17.69	40.81		
[Node	7	12.33		39.1	46.18	28.58	37.97	37.56	23.01	0	31.52
38.92	15.39	37.5	34.09	23.02	17.68	16.88	27.19	20.05	27.56		
[Node	8	64.69		43.79	13.62	46.74	27.06	44.56	44.22	26.25	0
44.65	13.48	42.39	46.95	26.93	23.49	31.83	30.24	32.23	32.09		
[Node	9	38.21		18.64	35.49	46.97	33.92	26.42	16.36	21.65	47.91
0	20.25	17.37	41.11	12.14	29.81	11.99	37.36	12.99	28.39		
[Node	10	38.87		35.15	18.79	23.55	49.69	47.51	32.88	37.75	47.74
21.45	0	16.77	18.92	35.63	33.53	23.28	41.16	13.61	15.51		
[Node	11	31.66		35.31	11.64	49.97	40.56	49.59	27.56	35.93	31.5
40.24	15.16	0	44.49	21.67	32.11	42.36	44.45	27.2	49.35		
[Node	12	30.85		37.92	44.23	15.94	36.37	45.95	12.12	26.41	32.63
24.16	26.46	26.4	0	33.61	28.39	42.29	23.32	39.58	30.28		
[Node	13	48.69		26.95	43.7	22.91	44.39	40.09	20.06	16.44	25.91
19.17	41.89	43.73	35.69	0	44.87	49.63	15.95	22.71	22.77		
[Node	14	52.92		28.22	24.11	33.94	39.49	35.24	18.32	15.59	11.74
16.88	28.18	11.68	36.44	33.85	0	34.07	33.57	22.03	13.3		
[Node	15	25.88		10.62	42.94	24.45	11.96	30.81	17.69	44.23	10.53
31.39	37.04	12.86	34.56	19.5	22.1	0	19.54	19	22.77		

**Appendix B** (*continued*)

[Number of nodes = 18,  $\delta = 17.49$ ]

[Node	$i$	Activity time $\tau_i$		Consolidation costs $\Delta_{ij}$ ( $j = 1, \dots, 18$ )							
[Node	16	60.51		18	25.11	49.98	24.26	32.2	23.52	21.66	18.47
	11.91	47.45	30.71	38.81	18.34	35.74	13.06	0	20.04	19.78	
[Node	17	91.13		28.27	46	17.2	37.95	30.14	37.74	19.9	26.02
	47.37	11.7	31.13	43.05	14.15	27.76	43.7	33.08	0	10.83	
[Node	18	49.2		45.63	34.58	34.49	41.91	35.73	13.57	19.62	10.29
	49.3	26.71	28.69	47.63	33.67	34.46	12.82	43.06	13.89	0	
[Arc	$i$	$\rightarrow$	$j$	$\gamma_{ij}$							
[Arc	8	$\rightarrow$	9	16.7							
[Arc	8	$\rightarrow$	12	30.32							
[Arc	6	$\rightarrow$	10	39.6							
[Arc	6	$\rightarrow$	9	11.73							
[Arc	4	$\rightarrow$	9	22.01							
[Arc	10	$\rightarrow$	16	8.19							
[Arc	13	$\rightarrow$	15	3.4							
[Arc	12	$\rightarrow$	13	35.92							
[Arc	15	$\rightarrow$	17	31.95							
[Arc	14	$\rightarrow$	17	37.71							
[Arc	15	$\rightarrow$	18	0.56							
[Arc	9	$\rightarrow$	13	33.62							
[Arc	9	$\rightarrow$	11	48.55							
[Arc	11	$\rightarrow$	13	19.54							
[Arc	4	$\rightarrow$	11	40.12							
[Arc	2	$\rightarrow$	6	0.41							
[Arc	1	$\rightarrow$	10	6.76							
[Arc	1	$\rightarrow$	4	46.38							
[Arc	5	$\rightarrow$	8	32.17							
[Arc	14	$\rightarrow$	15	17.22							
[Arc	1	$\rightarrow$	13	47.72							
[Arc	4	$\rightarrow$	14	47.4							
[Arc	3	$\rightarrow$	17	42.41							
[Arc	6	$\rightarrow$	7	26.49							
[Arc	3	$\rightarrow$	9	11.6							
[Arc	10	$\rightarrow$	18	25.07							
[Arc	7	$\rightarrow$	17	23.87							
[Arc	7	$\rightarrow$	8	47.7							
[Arc	10	$\rightarrow$	12	49.81							
[Arc	7	$\rightarrow$	9	21.12							
[Arc	3	$\rightarrow$	10	26.33							
[Arc	5	$\rightarrow$	14	32.72							
[Arc	5	$\rightarrow$	15	40.36							
[Arc	17	$\rightarrow$	18	0.24							
[Arc	12	$\rightarrow$	18	12.85							

(continued on next page)

**Appendix B** (continued)

---

[Number of nodes = 18,  $\delta = 17.49$ ]

---

[Arc	$i$	$\rightarrow$	$j$	$\gamma_{ij}$
[Arc	3	$\rightarrow$	12	35.96
[Arc	8	$\rightarrow$	10	12.25
[Arc	4	$\rightarrow$	15	11.8
[Arc	10	$\rightarrow$	11	33.7
[Arc	3	$\rightarrow$	11	2.4
[Arc	4	$\rightarrow$	12	36.12
[Arc	1	$\rightarrow$	11	25.96
[Arc	11	$\rightarrow$	16	31.03
[Arc	7	$\rightarrow$	18	37.51
[Arc	7	$\rightarrow$	12	35.9
[Arc	9	$\rightarrow$	10	16.7
[Arc	5	$\rightarrow$	6	20.42
[Arc	2	$\rightarrow$	16	26.69
[Arc	3	$\rightarrow$	4	28.06
[Arc	7	$\rightarrow$	13	19
[Arc	5	$\rightarrow$	17	25.95
[Arc	4	$\rightarrow$	7	16.42
[Arc	1	$\rightarrow$	2	48.04
[Arc	4	$\rightarrow$	8	42.13
[Arc	1	$\rightarrow$	16	13.48
[Arc	12	$\rightarrow$	16	21.9
[Arc	12	$\rightarrow$	14	17.33
[Arc	9	$\rightarrow$	18	19.88
[Arc	2	$\rightarrow$	17	37.9
[Arc	5	$\rightarrow$	18	28.96
[Arc	6	$\rightarrow$	13	0.11
[Arc	2	$\rightarrow$	5	20.47
[Arc	1	$\rightarrow$	15	5.93
[Arc	4	$\rightarrow$	10	44.48
[Arc	9	$\rightarrow$	17	8.23
[Arc	16	$\rightarrow$	17	16.48
[Arc	2	$\rightarrow$	10	15.39
[Arc	2	$\rightarrow$	3	48.79

---

**References**

- Bowen, H.K., Clark, K.B., Holloway, C.A., Wheelwright, S.C., 1994. *The Perpetual Enterprise Machine*. Oxford University Press, New York.
- Buzacott, J., 1996. Commonalities in reengineering processes: Models and issues. *Management Science* 42 (5), 768–782.
- Demeulemeester, E., Dodin, B., Herroelen, W., 1993. A random activity network generator. *Operations Research* 41 (5), 972–980.
- Demeulemeester, E., Herroelen, W., 1992. A branch-and-bound procedure for the multiple resource-constrained project

- scheduling problem. *Management Science* 38 (12), 1803–1818.
- Glover, F., Laguna, M., 1993. Tabu search. In: Reeves, C.R. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*. Halsted Press, an Imprint of John Wiley & Sons, New York, pp. 70–150.
- Hammer, M., 1990. Reengineering work: Don't automate, obliterate. *Harvard Business Review* 68 (4), 104–112.
- Hammer, M., Champy, J., 1993. *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Business, New York.
- Kerzner, H., 1979. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. Van Nostrand Reinhold, New York.
- Krishnan, V., Eppinger, S., Whitney, D.E., 1997. A model-based framework to overlap product development activities. *Management Science* 43 (4), 437–451.
- Meredith, J.R., Mantel, S.J., 2000. *Project Management: A Managerial Approach*, fourth ed. John Wiley & Sons, New York.
- Pande, P.S., Neuman, R.P., Cavanagh, R.R., 2000. *The Six Sigma Way: How GE, Motorola, and Other Top Companies are Honing Their Performance*. McGraw-Hill Professional Publishing, New York.