DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9521

New Benchmark Results for the Resource-Constrained Project Scheduling Problem

by

Erik L. DEMEULEMEESTER

Willy S. HERROELEN

Katholieke Universiteit Leuven

Naamsestraat 69, B - 3000 Leuven

ONDERZOEKSRAPPORT NR 9521

New Benchmark Results for the Resource-Constrained Project Scheduling Problem

by

Erik L. DEMEULEMEESTER

Willy S. HERROELEN

D/1995/2376/22

NEW BENCHMARK RESULTS FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM

Erik L. Demeulemeester • Willy S. Herroelen

Department of Applied Economics
Katholieke Universiteit Leuven
Naamsestraat 69
B-3000 Leuven (Belgium)
Tel 32-16-32 69 70 - 32-16-32 69 72
Fax 32-16-32 67 32
e-mail: willy.herroelen@econ.kuleuven.ac.be.
erik.demeulemeester@econ.kuleuven.ac.be

June 1995

This research was sponsored by the Belgian National Science Foundation under Research Contract FKFO N° 2.0051.94

New Benchmark Results for the Resource-Constrained Project Scheduling Problem

Erik L. Demeulemeester • Willy S. Herroelen
Department of Applied Economics, Katholieke Universiteit Leuven,
Naamsestraat 69, B-3000 Leuven, Belgium

ABSTRACT

This paper reports on computational results obtained with an updated version of the branch-and-bound procedure previously developed by Demeulemeester and Herroelen (1992) for solving the resource-constrained project scheduling problem (RCPSP). The new code fully exploits the advantages of 32-bit programming provided by recent compilers running on platforms such as Windows NT® and OS/2®: flat memory, increased addressable memory and fast program execution. We study the impact of three important variables on the computation time for the RCPSP: addressable computer memory, the search strategy (depth-first, best-first or hybrid) and the introduction of an improved lower bound. We compare the results obtained by a truncated branch-and-bound procedure with the results generated by the minimum slack time heuristic and report on the dependency of its solution quality on the allotted CPU time.

(Project Scheduling - Resource Constraints; Branch-and-bound; 32-bit Programming; Computational Results)

1. Introduction

The resource-constrained project scheduling problem (RCPSP) involves the scheduling of project activities satisfying precedence and resource constraints in order to minimize the total project duration. The finish-start precedence constraints impose that an activity can only start upon completion of all its predecessor activities. Resources are renewable and assumed to be available in constant amounts. Resources are also demanded by an activity in constant amounts throughout the duration of the activity. Demeulemeester and Herroelen (1992) presented an efficient depth-first branch-and-bound procedure (subsequently referred to as the *DH-procedure*), which became the benchmark on the commonly used 110 test problems assembled by Patterson (1984). Computational experience confirmed the DH-procedure to be, on the average, almost twelve times faster than the best-first procedure developed by Stinson et al. (1978), previously reported (Patterson 1984) to be the most effective and efficient on this problem set.

Subsequent research by Kolisch, Sprecher and Drexl (1995) questioned the use of the 110 problem set and led to the development of *Progen*, a network generator which allows for the generation of RCPSP problem instances which satisfy preset problem parameters. Computational experience on a total of 480 problem instances, generated on the basis of a full factorial design, revealed that the DH-procedure could optimally solve 428 instances within one hour of computation time on an IBM PS/2 Model 55sx (80386sx processor, 15 MHz clockpulse). A close look at the 52 problems that could not be solved to optimality within the imposed time limit of one hour, however, indicated that the dominant factor which kept the procedure from finding the optimal solution, was not so much the computation time spent (as could be assumed from the results), but mainly the size of the computer memory that could be addressed. The DOS®-version of the personal computer code allowed for an addressable memory of (less than) 640 Kb (kilobytes) in total, while, mainly for efficiency reasons, matrices used by the algorithm could not exceed a size of 64 Kb.

Recent advances in 32 bit-compiler technology inspired us to implement a new code for the DH-procedure (subsequently referred to as the *new DH-procedure*) using a Microsoft Visual C++ 2.0® compiler under Windows NT 3.50®. As will be shown below, this resulted in a speed boost by a factor of almost ten as compared to the code used for our 1992 *Management Science*

paper, but, more importantly, initial experimentation revealed that many more of the 480 Kolisch, Sprecher and Drexl (KSD) instances could now be solved optimally, which confirmed our observation that addressable computer memory was the real bottleneck.

The objective of this paper is to report on the new benchmark results. The remainder of the paper is organized as follows. Section 2 describes the results obtained by the new DH-procedure on two problem sets: the commonly used Patterson set (1984) and the Progen set of Kolisch, Sprecher and Drexl (KSD) (1995). Section 3 discusses the impact of addressable computer memory on the solution quality obtained. In Section 4, we discuss the impact of three different search strategies (depth-first, best-first and hybrid search) on the performance of the DH-procedure, as well as the main reasons for the depth-first search strategy to remain a clear favorite. In Section 5, we report on the beneficiary impact of the use of an improved lower bound, adapted from recent research by Mingozzi et al. (1994). Section 6 studies the impact of allotted CPU time on the performance of the new DH-procedure. The truncated DH-procedure is shown to yield promising results. Section 7 is reserved for overall conclusions. All computational results that are presented in this paper were obtained on an IBM PS/2 Model P75 with a 486 processor running at 25 MHz and with 32 Mb of internal memory.

2. The new DH-procedure

The new DH-procedure is conceptually almost identical to the DH-procedure described in Demeulemeester and Herroelen (1992). For a good understanding of the remainder of this paper, we have to refer the reader to Demeulemeester and Herroelen (1992) for the details of the DH-procedure. The search strategy of the new DH-procedure is identical to the depth-first strategy used by the DH-procedure. It generates a depth-first search tree, the nodes of which correspond to partial schedules in which finish times temporarily have been assigned to a subset of activities of the project. The partial schedules are feasible, satisfying both the precedence and resource constraints. Partial schedules are only considered at time instants (decision points) m which correspond to the completion time of one or more project activities. The partial schedules are constructed by semi-active timetabling. Partial schedules are built up starting at time zero and proceed systematically throughout the search process by adding at each decision point subsets of activities, including the empty set, until a complete feasible schedule is obtained. Eligible

activities can start at time m if the resource constraints are not violated. A resource conflict at time m produces a new branching in the branch-and-bound tree. The branches describe ways to resolve the resource conflict by deciding on which combinations of activities are to be delayed (delaying alternatives). Only minimal delaying alternatives which do not contain others as a subset have to be considered. The delay of a delaying alternative is accomplished by adding temporal constraints causing the corresponding activities to be properly delayed. The search process continues until the dummy end activity has been scheduled. Every time such a complete schedule has been found, backtracking occurs: a new delaying alternative is chosen from the set of delaying alternatives at the highest level of the search tree that still has some unexplored delaying alternatives left, and branching continues from that node. When level zero is reached in the search tree, the search process is completed. Two dominance rules are used to prune the search tree: a left-shift dominance rule and a so-called cutset dominance rule (at time m the cutset consists of the set of unscheduled activities for which all predecessor activities belong to the partial schedule at time instant m). The DH-procedure was equipped with three lower bounding rules: the well-known critical path length bound *LB0*, a critical sequence lower bound *LB1* (Stinson et al. 1978) and an extended critical sequence lower bound *LB2* (Demeulemeester 1992). Subsequent research revealed that often *LB0* outperformed the critical sequence lower bounds *LB1* and *LB2*, when used in combination with the cutset dominance pruning rule (for more information on this trade-off between the lower bound calculation and the use of dominance rules, see Demeulemeester (1992)). Recently, Mingozzi et al. (1994) introduced a new lower bound, *LB3*, based on a new mathematical formulation for the RCPSP and implemented by using a heuristic for solving a set packing problem. We incorporated our version of *LB3* in the new DH-procedure using the heuristic procedure described in Section 5.

In addition to the removal of *LB1* and *LB2* and the possibility to use both *LB0* and *LB3*, the new DH-procedure is the result of three additional changes, which have been made in order to gain on speed and to exploit the power of modern 32-bit compiler architecture. The major change has to do with a new coding scheme for the cutset dominance rule. Being limited to matrices of at most 64 Kb, the original DOS-version of the DH-procedure used four matrices for coding the dominance rule. Two matrices of 64 Kb were used to store cutsets with the necessary information to apply the dominance rule and two matrices of 16 Kb contained the pointers to the

cutsets listed in the two 64 Kb matrices. The entry into one of these matrices was determined using a hashing function of a number that indicated the current cutset, *cs*, in a binary way: e.g. cs = 84 (binary: 00..001010100) indicates that activities 3, 5 and 7 are present in the partial schedule (their bit is set to 1). The flat memory model of 32-bit programming allows us to implement the cutset dominance rule using only two matrices: one very large cutset matrix (the impact of its size is discussed in Section 3) contains the cutsets with the additional information, while a second matrix of 256 Kb was used to store the pointers to the cutsets in the cutset matrix. This implementation has two important advantages: more cutsets can be saved (increasing the impact of the cutset dominance rule) and the code becomes simpler (improving the speed of its application).

Figure 1 The application of the cutset dominance rule

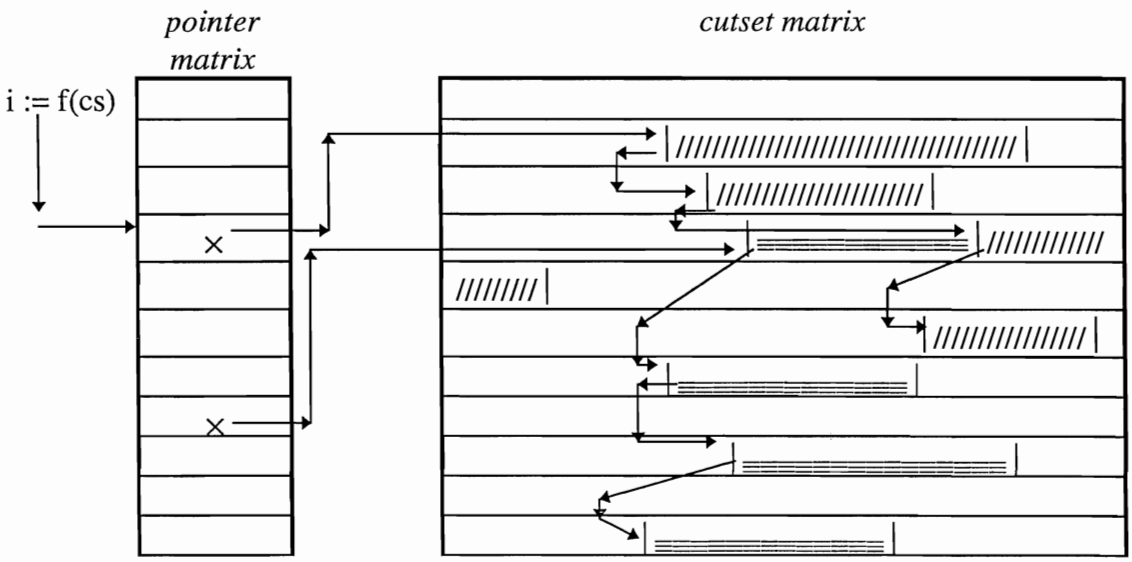


Figure 1 gives some additional information on how the cutset dominance rule is implemented in the new DH-procedure. For each cutset *cs*, the hashing function *i:=f(cs)* yields an entry in the pointer matrix which points to the start of a list of previously saved cutsets with corresponding cutset information. Each of these cutsets should be tested to see if it dominates *cs*.

A second change involved merging different resource types into one global resource type. This change became possible because integers automatically consist of 32 bits when 32-bit programming is used. Using, for instance, 8 bits for every resource type allows us to combine

four resource types into one 32 bit integer representing one global resource type. The cumulative demand for each individual resource type is then restricted to at most 127 resource units. An example may indicate how this works. Assume that for a certain RCPSP instance the availability for three resource types amounts to 8, 12 and 10 units, respectively. The resource availability for this project is then represented by the number *av* = 2760 (hexadecimal: 000A0C08 or binary: 00000000 00001010 00001100 00001000, where the last 8 bits represent the first resource type, the second but last group of 8 bits the second resource type, and so on ...). If at a certain decision point the cumulative resource requirements for the (temporarily) scheduled activities amounts to 7, 11 and 13, respectively, then this will be indicated by the number *req* = 3511 (hexadecimal: 000D0B07 or binary: 00000000 00001101 00001011 00000111). Subtracting *req* from *av*, we obtain a result of -751 (hexadecimal: FFFD0101 or binary: **1**1111111 **1**1111101 **0**0000001 **0**0000001), indicating that the remaining resource capacity equals 1, 1 and -3, respectively, revealing a resource conflict at this decison point. The occurrence of a resource conflict can be easily tested by checking whether bits 8, 16, 24 or 32 (shown in bold), are set to one. In this example, bits 24 and 32 are both set to 1, revealing a resource conflict. Combining several resource types into one (or a few) global resource type(s) leads to a definite speed-up of the code.

A last minor change involved reversing all loops in the program, if possible, from running up to running down. For instance the program code: for (i = 1; i <= n; i++) was changed into: for (i = n; i >= 1; i--). This cosmetic change led to a decrease in computation time by some 10 percent.

The computational results of the *new DH-procedure*, using a cutset matrix of 24 Mb (megabytes), a pointer matrix of 256 Kb, and a CPU time limit of 3600 seconds, can be found in Table 1.

From Table 1 we observe that the average computation time needed to solve all the 110 Patterson problems decreased from 0.215 seconds for the DH-procedure to 0.026 seconds for the new DH-procedure using *LB0* and 0.025 seconds using *LB3*. This is not a totally fair comparison as about 0.01 seconds are needed in the new DH-procedure to initialize the 256 Kb matrix that contains the pointers to the saved cutset information. The magnitude of this matrix only comes into play when difficult problems are being solved, so that in fact, on the Patterson problem set,

Table 1 Comparison between the DH-procedure and the new DH-procedure

Test problem set	DH-procedure	New DH-procedure with *LB0*	New DH-procedure with *LB3*
110 Patterson problems	110/110 0.215 sec*	110/110 0.026 sec***	110/110 0.025 sec***
480 KSD problems	428 / 480** 79.907 sec	479 / 480 14.757 sec***	479/480 12.331 sec***

* IBM PS/2 Model 70 A21 with a 80386 processor and coprocessor running at 25 MHz
** IBM PS/2 Model 55sx with a 80386 processor (no coprocessor) running at 15 MHz
*** IBM PS/2 Model P75 with a 80486 processor running at 25 Mhz

most of this initialization time is lost. It can be observed that the new DH-procedure now solves 479 out of the 480 KSD instances to optimality in an average time (for the optimally solved problems) of 14.757 seconds using *LB0* and 12.331 seconds using *LB3*. The DH-procedure needed an average time of 79.907 seconds to solve only 428 problems. The new DH-procedure optimally solved the remaining problem (KSD291) within 9,515.12 seconds using *LB0* and 10,261.80 seconds using *LB3* (the use of *LB0* (*LB3*) yielded the optimal solution after 497 (714) seconds, but the remaining time was spent on confirming optimality). To the best of our knowledge, this is the first time that an optimal solution could be found for all 480 KSD instances (the results reported by Mingozzi et al. (1994) must be interpreted with care, due to the fact that their code produced erroneous optimal solutions for some of the instances (Mingozzi et al. 1995)).

3. The impact of addressable computer memory

As already indicated in Section 1, the amount of computer memory that can be addressed by a branch-and-bound procedure seems to be a bottleneck for finding optimal solutions to difficult problem instances. In both the original DH-procedure and the new DH-procedure, almost all memory is used for applying the cutset dominance rule. Therefore we have tested the new code with different memory sizes allocated for storing the cutset matrix. We have used memory sizes of 256 Kb, 1 Mb, 4 Mb, 16 Mb and 24 Mb (the maximum that could be assigned on our computer equipped with 32 Mb of internal memory). Table 2 lists the number of KSD instances

solved to optimality by the new DH-procedure (using *LB0*) within one hour of computation time together with the average computation time required.

Table 2 The impact of addressable computer memory

Amount of memory allocated for storing the cutset matrix	Number of optimally solved problems within a CPU time limit of 1 hour	Average computation time required by the optimally solved problems*
256 Kb	456 / 480	49.338 seconds
1 Mb	468 / 480	38.277 seconds
4 Mb	472 / 480	12.247 seconds
16 Mb	477 / 480	11.522 seconds
24 Mb	479 / 480	14.757 seconds

* IBM PS/2 Model P75 with a 80486 processor running at 25 MHz

From Table 2 we can observe that the use of a cutset matrix of 256 Kb, which amounts to twice the size that we used in the DOS-based procedure, allows us to solve 456 out of the 480 KSD instances. As the size of this matrix is increased to 24 Mb, we clearly see that more and more problems are being solved optimally, while the average computation time consumed by the optimally solved problems decreases steadily. Only 2 out of the 480 KSD instances need the full 24 Mb for storing their cutset information, indicating that the amount of computation time that still can be gained by increasing the memory for the cutset dominance rule becomes minor.

4. The impact of the search strategy

Branch-and-bound procedures may use different philosophies of searching the tree. In the *depth-first search* performed by the DH-procedure, new branchings are produced by a resource conflict. The branches describe ways to resolve the resource conflict; i.e., decisions about which combinations of activities are to be delayed (delaying alternatives). For each such delaying alternative a lower bound is computed and the depth-first branching strategy branches from the sprouted node (delaying alternative) having the smallest lower bound. Backtracking occurs when a schedule is completed or a branch is to be fathomed by the lower bound calculation and/or dominance rules. When backtracking, delaying alternatives are explored at the same level of the search tree. If there is no delaying alternative left unexplored at this level, the procedure

backtracks to the previous level. When level zero is reached in the search tree, the search process is completed and the optimal solution is secured.

A *best-first search* strategy, used for example in the Stinson procedure (Stinson et al. 1978), continues branching from the node which has the lowest bound among all pending nodes at *all* levels of the search tree, while backtracking would always lead the procedure to continue from the pending node with the smallest lower bound.

A definite advantage of depth-first procedures is that they may lead to feasible solutions of relative good quality, relatively fast. In general, best-first procedures have to visit fewer nodes in the search tree. An alternative branching strategy which we label *hybrid search*, branches in exactly the same way as done by a depth-first search: branching continues from the best node among the ones just created. Upon backtracking, however, the procedure continues branching from the best node at all levels in the tree.

In recoding, we have equipped the new DH-procedure with the possibility to perform a depth-first, best-first or hybrid search strategy. The resulting average computation times to solve all 110 Patterson instances are indicated in Table 3.

Table 3 The impact of the search strategy used on the speed of the new DH-procedure (using *LB0*)

Search strategy	Average computation time*
Depth-first search	0.026 seconds
Best-first search	0.031 seconds
Hybrid search	0.031 seconds

* IBM PS/2 Model P75 with a 80486 processor running at 25 MHz

As can be observed, the depth-first search strategy yielded the smallest computation times. The best-first and hybrid search strategies, however, performed equally well and, on average, needed an additional CPU time of less than 0.005 seconds. The clear advantage of the depth-first strategy showed up when we tried to tackle the KSD instances. As shown in Table 1, the depth-first strategy performed very well. However, our attempt to obtain significant results using the best-first and hybrid search strategies on the KSD problem set failed. Many KSD instances could not be solved optimally, because the branching information (necessary for

backtracking) caused a memory overflow. The main difference in storing branching information between a depth-first search and both the best-first and hybrid processes lies in the fact that a stack can be used for the former. The branching information needs to be stored only once at each level of the search tree, while for the other search strategies this information needs to be stored for all nodes in the search tree. Moreover, the backtracking process causes the best-first and hybrid search to store the finish times of all scheduled activities, while depth-first search only needs to store the finish times of all activities in progress. The additional advantage of providing fast heuristic solutions of good quality (see Section 6), adds to the conviction that a depth-first search strategy remains a clear favorite for solving RCPSP instances.

5. The introduction of an improved lower bound

Recently, Mingozzi et al. (1994) introduced a new lower bound, *LB3*, which is based on a new mathematical formulation for the RCPSP. *LB3* proves to be stronger than Stinson's critical sequence lower bound, *LB1*, on both the Patterson and the KSD problems sets. Similar to *LB1*, the use of *LB3* is restricted to cases where the precedence constraints allow for pairs of activities to be scheduled in parallel, while the resource constraints do not.

Mingozzi et al. (1994) compute *LB3* using a heuristic for solving a set packing problem. We incorporated the following version of *LB3* in the new DH-procedure. For each activity $i \in A$ we determine its possible *companions,* i.e., the activities with which it can be scheduled in parallel, respecting both the precedence and resource constraints. All unscheduled activities i with a non-zero duration are then entered in a list L in non-decreasing order of the number of companions (non-increasing duration as a tie-breaker). The following procedure then yields a lower bound, *LB3*, for the partial schedule under consideration:

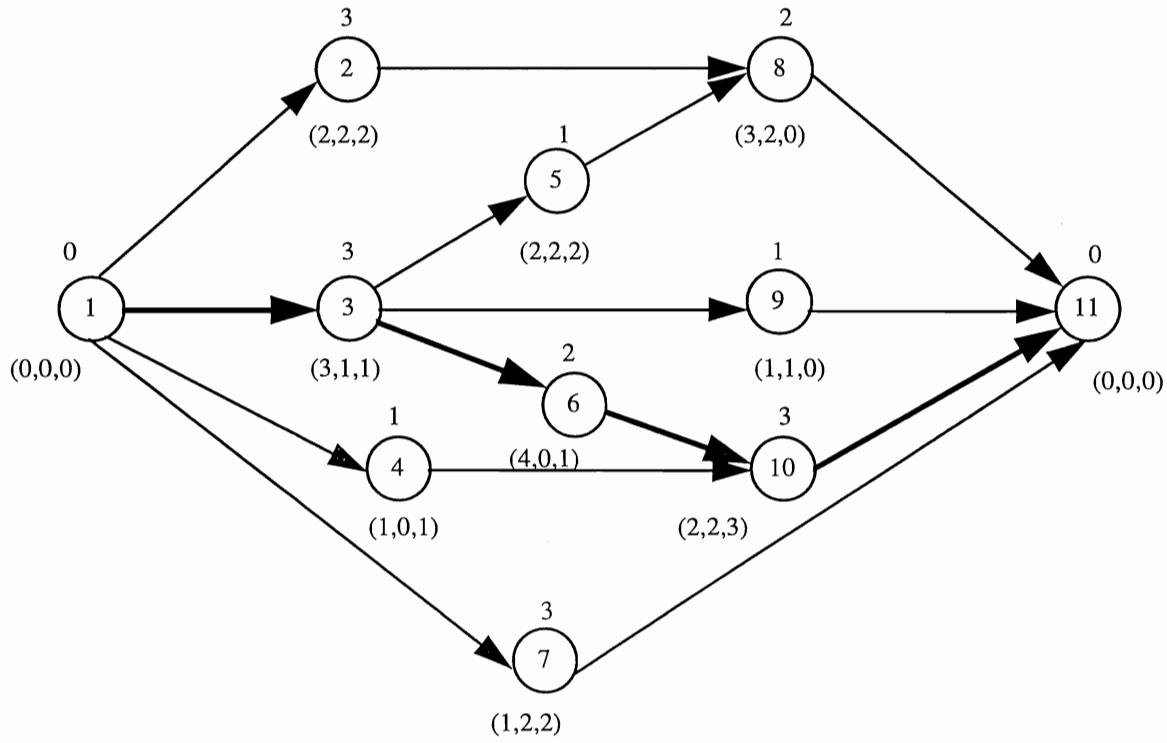
LB3:= the earliest completion time of the activities in progress
while list L not empty **do**
    Take activity j on top of list L and determine its duration d_j;
    LB3:=LB3 + d_j;
    Remove activity j and its companions from list L;
enddo.

The calculation of *LB3* can be illustrated on the problem example of Figure 2, which is borrowed from Mingozzi et al. (1994). The numbers above each node represent the activity

durations. The three numbers below each node denote the daily requirements for three resource types, each with a constant availability of 4 units. The companions for each activity can be listed as follows:

- Activity 1: none
- Activity 2: 4 / 5 / 7 / 9
- Activity 3: 4 / 7
- Activity 4: 2 / 3 / 5 / 7 / 8 / 9
- Activity 5: 2 / 4 / 7 / 9
- Activity 6: none
- Activity 7: 2 / 3 / 4 / 5 / 8 / 9
- Activity 8: 4 / 7 / 9
- Activity 9: 2 / 4 / 5 / 7 / 8 / 10
- Activity 10: 9
- Activity 11: none

Figure 2 A simple example of an RCPSP instance



Assuming that we want to compute *LB3* for the initial node in the search tree (no activities scheduled yet), the list *L* of unscheduled activities with a non-zero duration can be written as L=(6,10,3,8,2,5,7,4,9). Scanning the list, leads to the following:

LB3:=0;	L=(6,10,3,8,2,5,7,4,9) not empty;
LB3=0+2=2;	L=(10,3,8,2,5,7,4,9) not empty;
LB3=2+3=5;	L=(3,8,2,5,7,4) not empty;
LB3=5+3=8;	L=(8,2,5) not empty;
LB3=8+2=10;	L=(2,5) not empty;
LB3=10+3=13;	L=ϕ.

In this example, we observe that *LB3* exceeds both the critical path lower bound *LB0* = 8 and the critical sequence lower bound *LB1* = 11.

It is clear that other (more computationally intensive) heuristics can be used to calculate the lower bound *LB3*. The procedure described here is very fast and generally improves the critical path lower bound, *LB0*, if there are pairs of activities that can be scheduled in parallel taking into consideration the precedence constraints only, but cannot be scheduled in this manner if resource constraints are taken into consideration.

Computational results obtained by the new DH-procedure on the Patterson problems, using *LB3* in combination with a depth-first, best-first and hybrid search strategy, are given in Table 4.

Table 4 Results of *LB3* on the Patterson problem set

Search strategy	Average computation time
Depth-first search	0.025 seconds
Best-first search	0.026 seconds
Hybrid search	0.024 seconds

* IBM PS/2 Model P75 with a 80486 processor running at 25 MHz

A close look at the results of Table 3 and Table 4 reveals that for all three search strategies, the use of *LB3* allows the average computation time to be slightly reduced. The hybrid search strategy now performs best, but is closely followed by the depth-first and best-first strategies.

We have also tested the depth-first strategy on the KSD instances for various amounts of memory allocated for storing the cutset matrix. Table 5 gives the results.

Table 5 The performance of LB3 on the KSD instances

Amount of memory allocated for storing the cutset matrix	Number of optimally solved problems within a CPU time limit of 1 hour	Average computation time required by the optimally solved problems*
256 Kb	466 / 480	56.456 seconds
1 Mb	470 / 480	17.126 seconds
4 Mb	475 / 480	6.229 seconds
16 Mb	478 / 480	10.439 seconds
24 Mb	479 / 480	12.331 seconds

* IBM PS/2 Model P75 with a 80486 processor running at 25 Mhz

Table 2 and Table 5 reveal the beneficiary effect of the use of the stronger lower bound *LB3*. With 256 Kb of memory available for storing the cutset matrix, the use of *LB3* allows to solve ten more problems than could be solved using *LB0*, at the expense of a somewhat higher average computation time. The power of *LB3* is most striking when 1 or 4 Mb of memory is used: it then allows to solve more problems at approximately half the computational time. The improvement in the computational results due to the use of *LB3* is due to two separate factors: fewer nodes are branched from and better solutions are reached much earlier in the search process due to the stronger lower bound. The higher the amount of memory allocated for storing the cutset matrix, the smaller the impact of *LB3* and the larger the impact of the cutset dominance rule becomes. With 16 Mb of memory or more, the power of *LB3* is somewhat offset by the cutset dominance rule: almost the same number of instances can be solved within computer times which are somewhat smaller.

6. The impact of allotted computer time

A definite advantage of a depth-first search strategy over a best-first search is that during the branch-and-bound search gradually improving feasible solutions are generated very quickly. An important question relates to the quality of the feasible solutions obtained by a truncated branch-and-bound procedure which is made to stop once a feasible solution is found. Table 6 compares the average project length over all 480 KSD instances that is found by a parallel minimum slack (MINSLK) heuristic with the first feasible solution that is found by the new DH-procedure

(depth-first, *LB0 and LB3,* and 24 Mb for storing the cutset matrix). MINSLK was chosen for comparison because the many computational studies reported in the literature rank it (and the equivalent late start time (LST) heuristic) among the best performing priority based parallel heuristics (see Alvarez-Valdes and Tamarit 1989, Boctor 1990, Davis and Patterson 1975, Elsayed 1982, Kolisch 1995, Lawrence 1985, Pascoe 1966, Patterson 1973, 1976, Thesen 1976, Ulusoy and Özdamar 1989, Valls et al. 1992, Whitehouse & Brown 1979).

Table 6 Comparison of MINSLK and the truncated new DH-procedure

Average optimal project length	58.869
Average project length obtained by MINSLK	61.871
Average CPU time for MINSLK*	0.002 seconds
Average project length of the first feasible solution found by the new DH-procedure	
using *LB0**	62.069
using *LB3**	61.979
Average CPU time for the new DH-procedure to reach the first feasible solution	
using *LB0**	0.012 seconds
using *LB3**	0.012 seconds
Average CPU time for the new DH-procedure to reach a solution at least as good as the MINSLK solution	
using *LB0**	0.641 seconds
using *LB3**	0.038 seconds

* IBM PS/2 Model P75 with a 80486 processor running at 25 Mhz

From Table 6 it can be observed that, on the KSD problem set, MINSLK yields slightly better solutions than the truncated new DH-procedure both in terms of computational speed and solution quality (an average CPU time of only 0.002 seconds and an average deviation of 5.10 % from the optimal project duration as compared to an average deviation of 5.28 % (5.44 %) for the first feasible solution found by the new DH-procedure using *LB3* (*LB0*)). An average of 0.641 seconds is needed by the new DH-procedure using *LB0* to find a solution that is at least as good as the MINSLK solution (for many KSD instances the first solution found by the new DH-procedure is better than the one found by MINSLK). However, the major part of this computation time (290.31 seconds) is used to find the MINSLK solution for problem 290. Without this

problem, the average time is only 0.036 seconds. Using *LB3*, the average time CPU time needed to reach a solution at least as good as the MINSLK solution is 0.038 seconds.

Our results seem to be in line with the ones obtained by Kolisch (1995) on a subset of 308 KSD instances. He reports that a single pass LST (latest start time priority rule, which for a parallel scheduling scheme is equivalent to MINSLK) performs best and yields an average deviation from the optimal solution of 5.06 %. This corresponds quite well with the average deviation of 5.10 % that we find for all 480 KSD instances. Kolisch (1995) also studies a regret based biased random sampling approach in which each activity is assigned a certain probability for being scheduled next. This probability is dependent on the priority rule that is used and is calculated using a regret function (see Kolisch (1995) for details). Kolisch shows that the performance-ranking of priority rules does not differ for single-pass scheduling and sampling. However, sampling improves the performance of single-pass scheduling significantly at the expense of more computational effort. For the 308 KSD instances, sampling in combination with the late finishing time heuristic (LFT) performs best and yields an average deviation of 2.08 % in an average computation time of 1.12 seconds on an IBM PC with a 80386dx processor and 40 Mhz clockpulse.

Table 7 reveals how both the average project length on all the 480 KSD instances and the resulting average deviation from the optimum found by the new DH-procedure do vary with the allotted CPU time.

Table 7 The impact of allotted computer time on the performance of the new DH-procedure

	Average project length		**Average deviation from the optimum**	
Time limit (seconds)	**with *LB0***	**with *LB3***	**with *LB0***	**with *LB3***
0.1	60.021	59.967	1.957 %	1.865 %
0.2	59.781	59.646	1.550 %	1.320 %
0.5	59.533	59.363	1.129 %	0.839 %
1	59.356	59.235	0.828 %	0.623 %
5	59.125	59.060	0.435 %	0.326 %
30	58.946	58.940	0.131 %	0.120 %
60	58.923	58.902	0.092 %	0.057 %
300	58.892	58.881	0.039 %	0.021 %
1200	58.869	58.869	0 %	0 %
3600	58.869	58.869	0 %	0 %

From Table 7 we can observe that running the new DH-procedure (using *LB0* or *LB3*) for 0.1 seconds on all the 480 KSD instances yields project lengths that deviate, on average, less than 2.00 % from the optimal solution. This result, obtained on an IBM PS/2 Model P75 with a 486 processor running at 25 Mhz and 24 Mb for storing the cutset matrix, already beats the 2.08 % deviation obtained by LFT on an 80386dx processor and 40 Mhz clockpulse using regret based biased random sampling. Specifying a time limit of 1 second and using *LB0* allows the new DH-procedure to find project lengths that deviate on average only 0.83 % from the optimum; the use of *LB3* reduces the average deviation to 0.62 %. With a CPU time limit of one minute, the average deviation drops below 0.1 % when the new DH-procedure uses *LB0*, and drops below 0.06 % when *LB3* is used. From Table 7 we can also observe that all optimal solutions for the 480 KSD instances are found (not yet confirmed) within 20 minutes of CPU time. We think that these results constitute a strong case for introducing truncated branch-and-bound procedures in commercial project planning software.

7. Conclusions

This paper reports on computational results obtained with the new DH-procedure, a 32-bit version of the branch-and-bound procedure previously developed by Demeulemeester and Herroelen (1992) for solving the RCPSP. It is shown that this new implementation on an IBM PS/2 Model P75 with a 486 processor running at 25 Mhz and with 24Mb of addressable memory for storing the cutset matrix, optimally solves all Patterson problems using a simple critical path based lower bound *(LB0)* in an average computation time of 0.026 seconds, while 479 out of the 480 KSD instances are solved optimally in an average time of 14.757 seconds. The remaining problem (KSD291) was solved within 3 hours of computation time. As this is the first time that all 480 KSD instances are solved optimally, these results constitute a new benchmark for the RCPSP.

Subsequently, we have tested the impact of the addressable computer memory, the search strategy and the introduction of an improved lower bound on the computation time for the RCPSP. It is shown that the real bottleneck for finding optimal solutions to difficult problem

instances is the amount of computer memory that is available for storing the cutset matrix, and not that much the specified limit on the computation time.

With regard to the search strategy, our tests indicate that a depth-first, best-first or hybrid search strategy perform almost equally well on the Patterson problems. However, many KSD instances could not be solved using a best-first or hybrid strategy because too much memory was needed. As a depth-first search strategy has the additional advantage that heuristic solutions are found very fast, we consider it to be a clear favorite for solving the RCPSP.

Using an adapted version of the new lower bound, *LB3*, that was introduced by Mingozzi et al. (1994), the average computation time for solving the 110 Patterson problems reduces (slightly) to 0.025 seconds, while the same 479 KSD instances can be solved optimally in an average computation time of 12.33 seconds as compared to 14.76 seconds with a critical path lower bound. The lower bound *LB3* outperforms *LB0*: for different amounts of memory allocated for storing the cutset matrix, it solves more instances at a smaller computational cost. The beneficiary impact of *LB3* decreases as the amount of memory allotted for storing the cutset matrix increases. These results indicate that the search for better, easy to compute lower bounds in combination with strong dominance rules, remains an important research issue.

Finally, we have considered how a truncated new DH-procedure compares to the heuristics that have been presented in the literature. We found that for many KSD instances the first solution found by the new DH-procedure is better than the one found by MINSLK. In addition, the truncated new DH-procedure performed slightly better than the best regret based biased random sampling approach presented by Kolish (1995) when a CPU time limit of 0.1 seconds was specified. The average deviation from the optimal solution was less than 2.00 %, while Kolisch reports an average deviation of 2.08 % for the latest finish time heuristic in the sampling approach in an average computation time of 1.12 seconds. Extending the time limit to 1 second, the average deviation decreases to 0.83 % using *LB0* and 0.62 % using *LB3*, whereas a CPU time limit of 1 minute and the use of *LB3* suffice to make the average deviation drop below 0.06 %. We feel that these results represent a strong case for introducing truncated branch-and-bound procedures in commercial project planning software.

References

Alvarez-Valdes, R. and J.M. Tamarit, "Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis", in: Slowinski, R. and J. Weglarz (Eds.): *Advances in Project Scheduling* (1989), Elsevier, Amsterdam, 113-134.

Boctor, F.F., "Some Efficient Multi-Heuristic Procedures for Resource-Constrained Project Scheduling", *European Journal of Operational Research*, Vol. 49 (1990), 3-13.

Cooper, D.F., "Heuristics for Scheduling Resource-Constrained Projects: An Experimental Investigation", *Management Sci.*, 22(1976), 1186-1194.

Davis, E.W. and J.H. Patterson, "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling", *Management Sci.*, 21(1975), 944-955.

Demeulemeester, E., "Optimal Algorithms for Various Classes of Multiple Resource-Constrained Project Scheduling Problems", unpublished Ph.D. Dissertation, Katholieke Universiteit Leuven, Belgium (1992).

Demeulemeester, E. and W. Herroelen, "A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem", *Management Sci.*, 38(1992), 1803-1818.

Elsayed, E.A., "Algorithms for Project Scheduling with Resource Constraints", *International Journal of Production Research*, 20(1982), 95-103.

Kolisch, R., "Serial and Parallel Resource-Constrained Project Scheduling Methods Revisited: Theory and Computation", Research Paper, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Germany, May 1995.

Kolisch, R., A. Sprecher and A. Drexl, "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems: Easy and Hard Instances", *Management Sci.*, (1995), to appear.

Lawrence, S.R., "Resource-Constrained Project Scheduling - A Computational Comparison of Heuristic Scheduling Techniques", Working Paper, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, USA, 1985.

Mingozzi, A., V. Maniezzo, S. Ricciardelli and L. Bianco, "An Exact Algorithm for Project Scheduling with Resource Constraints Based on a New Mathematical Formulation", Technical Report N° 32, University of Bologna, September 1994.

Mingozzi, A., V. Maniezzo, S. Ricciardelli and L. Bianco, private communication (1995).

Pascoe, T.L., "Allocation of Resources CPM", *Revue Française de Recherche Opérationelle*, 38(1966), 31-38.

Patterson, J.H., "Alternate Methods of Project Scheduling with Limited Resources", *Naval Research Logistics Quarterly*, 20(1973), 767-784.

Patterson, J.H., "Project Scheduling: The Effect of Problem Structure on Heuristic Performance", *Naval Research Logistics Quarterly*, 23(1976), 95-124.

Patterson, J.H., "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem", *Management Sci.*, 30(1984), 854-867.

Stinson, J.P., E.W. Davis and B. Khumawala, "Multiple Resource-Constrained Scheduling Using Branch-and-Bound", *AIIE Trans.*, 10(1978), 252-259.

Thesen, A., "Heuristic Scheduling of Activities Under Resource and Precedence Restrictions", *Management Sci.,* 23(1976), 412-422.

Ulusoy, G. and L. Özdamar, "Heuristic Performance and Network/Resource Characteristics in Resource-Constrained Project Scheduling", *Journal of the Operational Research Society*, 40(1989), 1145-1152.

Valls, V., M.A. Perez and M.S. Quintanilla, "Heuristic Performance in Large Resource-Constrained Projects", Working Paper, Departament d'Estadistica i Investigacio Operativa, Universitat de Valencia, Spain, 1992.

Whitehouse, G.E. and J.R. Brown, "Genres: An Extension of Brooks Algorithm for Project Scheduling with Resource Constraints", *Computers and Industrial Engineering*, 3(1979), 261-268.