Control and Cybernetics

vol. **43** (2014) No. 1

Cost-efficient project management based on critical chain method with partial availability of resources[*][†]

by

Grzegorz Pawiński and Krzysztof Sapiecha

Department of Computer Science
Kielce University of Technology
al.1000-lecia P.P. 7, 25-314 Kielce
Poland
g.pawinski@tu.kielce.pl, krzysztof.sapiecha@gmail.com

Abstract: Cost-efficient project management based on Critical Chain Method (CCPM) is investigated in this paper. This is a variant of the resource-constrained project scheduling problem (RCPSP) when resources are only partially available and a deadline is given, but the cost of the project should be minimized. RCPSP is a well-known NP-hard problem but originally it does not take into consideration the initial resource workload. A metaheuristic algorithm driven by a metric of a gain was adapted to solve the problem when applied to CCPM. Refinement methods enhancing the quality of the results are developed. The improvement expands the search space by inserting the task in place of an already allocated task, if a better allocation can be found for it. The increase of computation time is reduced by distributed calculations. The computational experiments showed significant efficiency of the approach, in comparison with the greedy methods and with genetic algorithm, as well as high reduction of time needed to obtain the results.

Keywords: project management and scheduling, resource allocation, resource constraints, metaheuristic algorithms, parallel processing

1. Introduction

Researchers' attention has been focused on making the best use of the scarce resources available since PERT and CPM were developed in the late 1950's. Resource allocation, in the framework of the Resource-Constrained Project Scheduling Problem (RCPSP), attempts to reschedule the project tasks efficiently using limited renewable resources minimizing the maximal completion

[*]Submitted: January 2013; Accepted: December 2013

[†]This is an extended and modified version of the paper presented at the Congress of Young IT Scientists, SMI 2012.

time of all activities. It assumes that resources are available from the beginning. Thus, allocation conflicts can only occur between the project tasks.

Critical Chain Project Management (CCPM) method is the successor of the project scheduling techniques. CCPM is based on the theory of constraint (TOC) - a philosophy used to develop specific management techniques, focusing on constraints that prevent project from reaching its goals. Rand (2000) described the relationships between the Goldratt's ideas (Goldratt, 1997) and the CPM/PERT approach.

In this paper a cost-efficient project management based on the CCPM is investigated when resources are partially available. Such constraint better fits the real world project management problems. Dealing with more than one project is common in IT business, for example, and so managers have to use the resource-sharing approach.

The RCPSP is an NP-hard problem. Allowing for the use of resources only in specified time periods, makes RCPSP computationally very complex. To our best knowledge this is the first approach taking into consideration the resources having initial workload for the solved RCPSP. A metaheuristic algorithm from Deniziak (2004) was adopted to work with human resources and the CCPM method. System refinement methods were enhanced by an extra step in order to expand the search space. Performance of the algorithm was increased by the distributed calculations. The efficiency of the approach is also investigated.

The next section of the paper contains a brief overview of related work. The problem statement and the research purpose are given in Section 3. Section 4 describes the algorithm and presents a concept of enhancement of scheduling methods. Results of evaluation of both centralized and distributed computing models for the algorithm, as well as a comparison with two other methods are given in Section 5. The paper ends with conclusions.

2. Related work

Related work considers RCPSP with dedicated resources but without initial schedules of already assigned tasks. A lot of researchers studied RCPSP and suggested two different approaches to solve this problem: exact procedures and heuristics. The branch and bound methods (see, e.g., Brucker et al., 1998, Demeulemeester and Herroelen, 1997, 2002) are the only exact methods which allow for generation of optimal solutions with an acceptable computational effort. Mingozzi et al. (1998) presented another method - a tree search algorithm. It is based on a new mathematical formulation that uses lower bounds and dominance criteria.

However, exact methods may require a significant amount of computation time, not acceptable in most cases. The results of investigations of Hartmann and Kolisch (2000) showed that the best performing approaches were the Genetic Algorithms (GAs) of Hartmann (1998) and the Simulated annealing (SA) procedure of Bouleimen and Lecocq (2003). The in-depth study of the performance of the latest RCPSP heuristics can be found in Kolisch and Hartmann

(2006). Heuristics described by these authors, include X-pass methods, also known as priority rule based heuristics, classical metaheuristics, such as GAs, Tabu search (TS), SA, and Ant Systems. Non-standard metaheuristics and other methods were presented, as well. The former consist of a local search and population-based approaches which have been proposed to solve the RCPSP. The authors, referred to, investigated a heuristic which applied a forward-backward and backward-forward improvement passes. They have found out that the technique gives excellent results. For detailed description of the heuristic schedule generation schemes, priority rules, and representations, we refer the Reader to Hartmann and Kolisch (2000).

3. Problem statement

A single project consists of m tasks, which are precedence related by finish–start relationships with zero time lags. The relationship means that all predecessors have to be finished before a given task can be started. With each task an aggressive and safe time estimates are associated, corresponding to 50% ($T_{0.5}$) and 90% ($T_{0.9}$) confidence that the task will be completed on time, respectively.

To be processed, each task requires a human resource R. Resources are limited to one unit and therefore have to be applied to different tasks sequentially. To use a resource in a project, we have to cover the cost of its deployment, called unit cost (C_u). Since resources are unique, they may have individual costs of task execution per time unit (C_e). Whether the resource has allocated tasks or not its maintenance costs apply to the entire duration of the project. The cost is proportional to the resource cost C_e and project duration (T_p). Consequently, the total cost (C) of the project may be specified using the following equation:

$$C = \sum_{j=1}^{r} C_e(j) \cdot T_p + \sum_{j=1}^{n} \left(C_u(j) + \sum_{i=1}^{m} T_{0.9}(i,j) \cdot C_e(j) \right) \tag{1}$$

where $C_e(j)$ - cost of task execution per time unit by the resource j, T_p - project duration, $C_u(j)$ - resource j unit cost, $T_{0.9}(i,j)$ - task i safe time estimate when executed by the resource j, n - the number of resources used in the project schedule, m - the number of tasks, r - the number of resources in the resource library.

Furthermore, we assume that resources may have already allocated dummy tasks from a different project schedule. Such tasks cannot be moved. Hence, the resources are available only in particular time periods. Our goal is to allocate resources to the project tasks, taking into consideration availability of resources, in order to minimize the total cost of the project and to complete it before a deadline.

4. An algorithm of task scheduling

4.1. The algorithm

The metaheuristic algorithm from Deniziak (2004) was adapted to take into account specific features of human resources participating in a project schedule. It starts with an initial point and searches for the cheapest solution satisfying given time constraints. The initial schedule is a suboptimal solution generated by the greedy procedures. The procedures identify currently eligible activities, i.e. tasks without predecessors, trying to find a resource for each task. The resource is sought according to the smallest increase of project duration or project total cost.

In each pass of the iterative process, the current project schedule is being modified in order to get closer to the minimum of cost C. Direction of the search is determined by the metric of a gain. The gain defines the quality of improvement of the schedule. In order to avoid being trapped in a local minimum, total impact of modifications is measured for every new solution, as an increase of slack time $\Delta\Omega$:

$$\Delta\Omega = \sum_{i=1}^{m}(L_i - E_i) \tag{2}$$

where L_i - the latest task i start time, E_i - the earliest task i start time, m - the number of tasks (Tukel et al., 2006). Usually, the bigger the slack time, the broader the possibilities of resource allocation. If for any of the tasks L_i is less than E_i, the current solution does not satisfy time constraints and is rejected.

Finally, the gain obtained from the modifications of the current solution, taking into account the changes of the project total cost and the slack time, is evaluated using the following formula (Deniziak, 2004):

$$\Delta E = \left\{ \begin{array}{lll} \frac{-\Delta C}{\Delta\Omega} & for & \Delta\Omega < 0 \\ -\Delta C & for & \Delta\Omega = 0 \\ -\Delta C \cdot \Delta\Omega & for & \Delta\Omega > 0 \end{array} \right\}. \tag{3}$$

The input to the algorithm includes: project plan, resource library and maximal allowed project duration (a deadline).

4.2. Schedule refinement methods

Each pass in the iterative improvement process is an attempt to enhance the current solution in two stages. In the first stage a new resource is inserted (if it was not in the schedule). All tasks, giving a positive gain, are moved from other resources to the new resource and inserted in the position with the highest slack time. A task can be inserted if the resource has enough free time to execute it. Afterwards, resources without allocated tasks are removed from the current schedule. Finally, the best schedule goes to the next stage. If the project contains N tasks, D dummy tasks from the initial schedule and R resources, there are $(N + D) \cdot R$ possible modifications, in the worst case.

The second stage is performed only if at least two resources are left. Contrary to the first stage, resources are removed in order to get rid of the more expensive ones. Only a resource without allocated tasks from the current project can be removed. Hence, the algorithm tries to move all tasks from a resource onto other resources that still remain in the project schedule. A new resource with enough free time and giving the best gain is chosen for the task. If a new project schedule has a positive gain, it becomes the best one. In the worst case there are $(N + D) \cdot (R - 1)$ modifications. The iterative process is repeated for every resource from the resource library until no improvement can be found.

The original algorithm does not check the gain resulting from inserting a task in a gap between dummy tasks, when there are already assigned tasks. The task may be allocated there in another iteration, but only if tasks from the gap were moved. Moreover, different tasks from other resources could take their place earlier. We have improved the algorithm by expanding the search space. This is done by inserting the task in place of an already allocated task, if a better allocation can be found for it. However, in this step, tasks may be moved only to the resources having enough free time to execute them. The improvement requires at most $(N + D) \cdot (R - 1)$ more modifications in each stage. However, considering task relation dependencies, reduced availability of resources and their removal from the schedule, the possibilities are considerably limited.

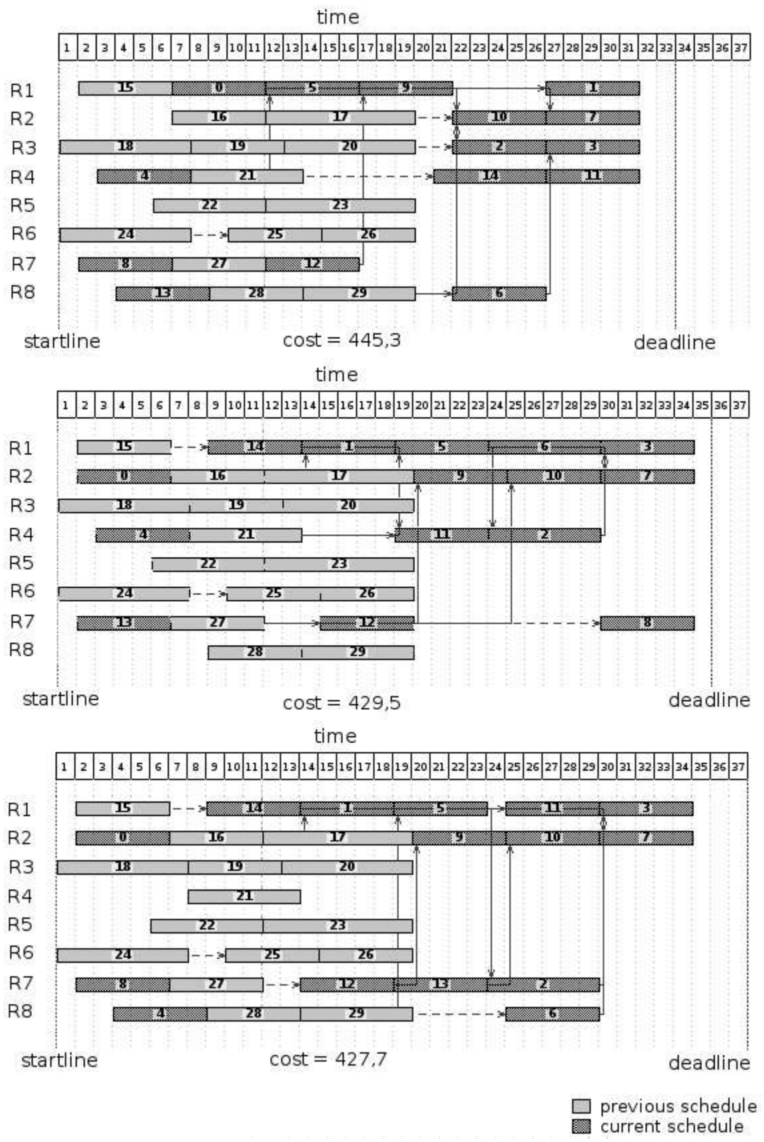
At the very end, project tasks may be shifted right using ALAP algorithm to the latest feasible position into their forward free slack. It should be noticed that all the tasks are scheduled without violating their logical relationships or resource constraints. Moreover, the obtained project schedule has to satisfy given time constraints. Project schedules not meeting a deadline are rejected. Influence of the deadline on the project duration and cost is shown in Fig. 1. It is also illustrated how the project tasks are allocated to resources.

5. Evaluation of the algorithm

5.1. Centralized processing

The efficiency of the algorithm was estimated on projects from PSPLIB, developed by Kolisch and Sprecher (1996). The library for RCPSP consists of 480 project instances in groups of ten, which have been systematically generated by varying three parameters: network complexity, resource factor, and resource strength. The parameters have a big impact on the hardness of the project instances (Kolisch and Sprecher, 1996). The set with 30 non-dummy activities is the hardest standard set of RCPSP-instances for which all optimal solutions are currently known (Demeulemeester and Herroelen, 1997). Thus, in our study, we used project instances with 30 non-dummy activities and from 4 to 8 resources with random data. Resource unit cost and cost of task execution may vary

Figure 1. Influence of the deadline on project cost and duration



by up to 10% from default values, which are 20 and 1 respectively. In each project, resource availability was reduced by randomly allocating 30 tasks from a project instance, which is located in the same group. We tested 96 projects that comprise two randomly selected instances from each of the groups. However, our results cannot be compared with optimal results from PSPLIB because of a different problem statement. So, we have compared them with the results obtained by Genetic Algorithm (GA) of Hartmann (1998). The GA population consisted of 30 individuals. In the evolution process, pairs of individuals were randomly drawn from the population and subjected to the operation of one-point crossover and mutation. If newly created genotypes did satisfy the time constraints, they were added to the current population. Thus, population size in each of the generations was at most 60. Finally, the 3-tournament reproduction operator was used 30 times to reduce the population to its former size. This iterative process was repeated until 100 generations were reached. The obtained results were averaged over 10 runs.

At first, we tested the improvement in the solutions achieved by the proposed refinement methods. Fig. 2 shows a task graph of project 3015_1, which contains 30 tasks. Scheduling results by the metaheuristic algorithm and its improvement are shown in Fig. 3. Dummy tasks, which could not be moved, were randomly assigned to 6 resources. The improvement allowed for decreasing the project cost by 8% owing to completion of the project 9 days earlier. Its advantage is the capability of backtracking from the earlier allocation decisions due to increased possibility of moving tasks. Tasks that are more critical are assigned earlier and therefore their successors may also be assigned earlier (e.g. task #13 in Fig. 3). As a result, tasks are better fitted into the gaps between the dummy tasks. Experimental results showed that the improvement of the refinement methods is suitable for projects with more restricted availability of resources (having lots of gaps).

Further tests were supposed to examine the project cost and the project time gain with respect to the resource number and the deadline. The results of comparison of methods, showing the arithmetic mean of 96 test instances, are given in Tables 1 and 2. The cost of projects with four resources was the lowest with the use of GA. However, the metaheuristic algorithm gives better results along with the increase of the number of resources. Other methods assign tasks to all resources, while in our approach the resources with deployment cost bigger than the gain resulting from executing tasks by them are unused. The longer the project, the more flexibility in the resource allocation is observed. So, tasks can be delayed and consequently moved from unprofitable resources. Afterwards, the resources can be removed from the schedule. On the other hand, a too long deadline in GAs decreases the quality of the newly created chromosomes and gives worse results. The best result was obtained for the deadline of 100 in projects with 4 resources, while in projects with 8 resources for the deadline of 40.

Experimental tests showed that the improvement makes it possible to reduce the project cost by 3.83% and the project duration by 3.31% in projects with

Figure 2. A task graph of project 3015_1 from PSPLIB

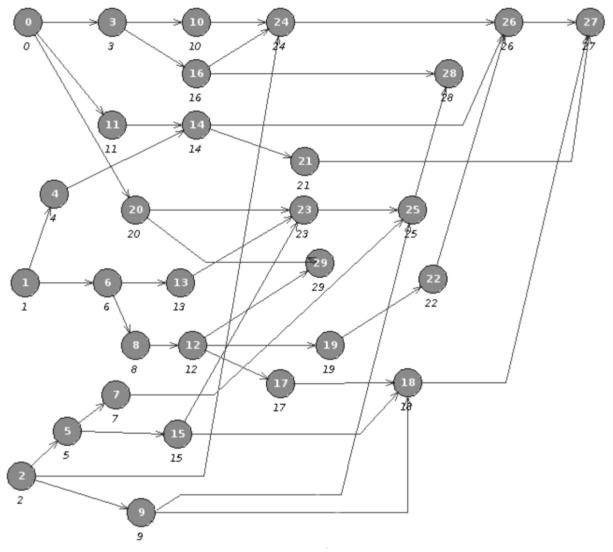


Figure 3. Comparison of the metaheuristic algorithm (b) and its improvement (a) while scheduling a project with 30 tasks and 6 resources

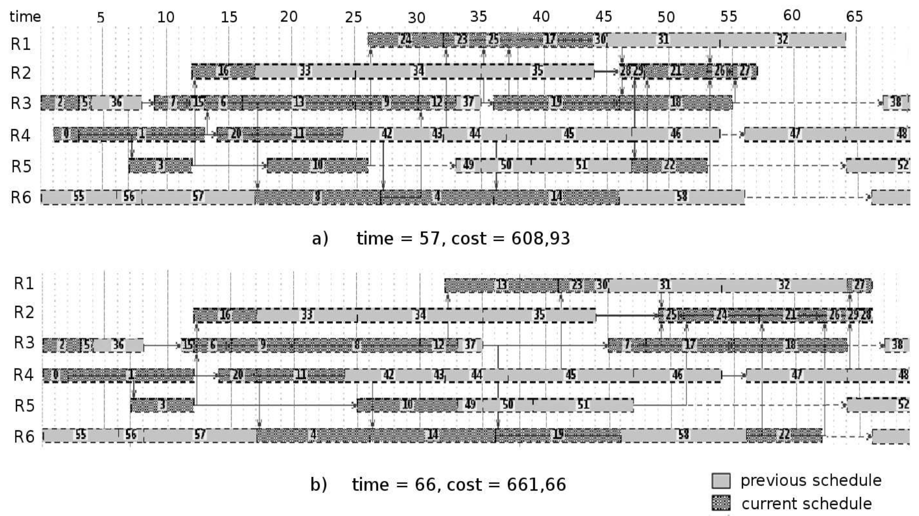


Table 1. Comparison of results of methods for the fastest initial schedule

		greedy		**opt**		**impr**		**GA**	
res_{num}	deadline	time	cost	time	cost	time	cost	time	cost
	40	80.78	572.61	78.38	560.12	78.24	559.21	74.63	544.15
	50	80.68	572.17	78.24	559.57	78.18	558.95	74.5	543.81
	60	80.59	571.79	78.16	559.22	78.14	558.56	74.55	543.85
4	70	80.55	571.69	77.96	558.68	77.89	557.8	74.65	544.29
	80	80.55	571.69	77.95	557	77.44	554.85	74.13	542.26
	90	80.55	571.69	77.52	552.75	76.54	549.82	74.11	541.17
	100	80.55	571.69	78.09	554.13	76.94	550.01	73.88	540.02
	40	67.04	686.59	65.48	663.95	65.03	660.21	66.65	658.94
	50	67.04	686.86	65.49	663.81	64.99	659.77	66.4	657.77
	60	67.03	686.77	65.41	663.09	64.81	658.33	66.55	658.9
6	70	67.03	686.77	65.26	659.81	64.72	655.4	66.44	656.9
	80	67.03	686.77	65.22	657	64.88	650.63	66.19	655.27
	90	67.03	686.77	65.33	658.23	65.2	653.01	66.44	656.7
	100	67.03	686.77	65.43	658.1	65.25	653.76	66.22	655.78
	40	59.21	797.75	58.54	763.99	57.96	758.94	62.01	766.96
	50	59.18	797.5	58.5	763.6	57.98	758.66	62.33	769.92
	60	59.18	797.5	58.53	761.49	57.96	756.68	62.32	769.06
8	70	59.18	797.5	58.7	760.9	57.96	752.66	62.48	769.21
	80	59.18	797.5	58.79	759.28	58.34	753.1	62.28	768.65
	90	59.18	797.5	58.79	759.43	58.41	754.46	61.85	768.16
	100	59.18	797.5	58.86	760.76	58.59	754.95	61.86	769.01

res_{num} - number of resources, greedy - greedy algorithm, opt - optimization algorithm, impr - the improved algorithm, GA - genetic algorithm.

four resources, by 5.24% and 4.31% in projects with six resources and by 5.65% and 2.11% in projects with eight resources. The approach is better than GA for projects with more resources than four. In projects with eight resources it allowed for reducing the project cost by 2.3% and the project duration by 6.6% compared to GA. However, the computation time dramatically grows due to the increased number of schedule modifications. Fortunately, it can be significantly reduced by distributed calculations.

5.2. Distributed processing

Schedule modifications can be done independently at every stage of the iterative improvement process. To start with, there are several attempts in the process to add a resource to the current schedule. Main computer (server) is responsible for distributing each attempt to remote computers doing calculations (workers). Workers reschedule the current solution according to the received project data and search parameters. Each worker calculates the schedule modifications for a different resource. Afterwards, the results of modifications are sent back to the server and passed to the next stage. After every attempt to add a resource, the second stage of the procedure starts, in which attempts to remove a resource are performed, similarly. Finally, the server chooses the best schedule in the second stage and then from all the second stages.

Table 2. Comparison of the results of methods for the cheapest initial schedule

		greedy		**opt**		**impr**		**GA**	
res_{num}	deadline	time	cost	time	cost	time	cost	time	cost
4	40	82.24	577.94	79.88	565.53	79.55	564.33	74.63	544.15
	50	82.09	577.33	79.74	565.17	79.41	563.72	74.5	543.81
	60	82.05	577.2	79.7	565.06	79.32	563.63	74.55	543.85
	70	82.05	577.2	79.63	564.77	79.29	563.47	74.65	544.29
	80	82.05	577.2	79.28	562.44	78.7	559.92	74.13	542.26
	90	82.05	577.2	79.54	560.71	78.78	557.08	74.11	541.17
	100	82.05	577.2	79.31	558.93	78.65	555.72	73.88	540.02
6	40	68	692.52	66.54	672.01	66.08	666.25	66.65	658.94
	50	68	692.46	66.52	671.23	66.09	666.3	66.4	657.77
	60	68	692.46	66.48	670.52	65.96	664.7	66.55	658.9
	70	68	692.46	66.55	667.72	65.79	659.85	66.44	656.9
	80	68	692.46	66.27	663.74	65.75	656.68	66.19	655.27
	90	68	692.46	66.43	663.61	65.89	657.04	66.44	656.7
	100	68	692.46	66.6	663.36	66.67	660.19	66.22	655.78
8	40	59.6	799.25	58.57	766.92	58.23	763.44	62.01	766.96
	50	59.6	799.25	58.58	766.54	58.26	763.68	62.33	769.92
	60	59.6	799.25	58.6	764.41	58.22	761.16	62.32	769.06
	70	59.6	799.25	58.85	762.19	58.34	759.2	62.48	769.21
	80	59.6	799.25	58.82	760.45	58.38	756.31	62.28	768.65
	90	59.6	799.25	58.83	761.07	58.69	758.41	61.85	768.16
	100	59.6	799.25	59.04	762.06	58.9	759.01	61.86	769.01

res_{num} - number of resources, greedy - greedy algorithm, opt - optimization algorithm, impr - the algorithm improvement, GA - genetic algorithm.

Tests were performed on computers with Intel® Core$^{\text{TM}}$ i5-760 Processor (8M Cache, 2,80 GHz) and 2 GB of RAM memory. The server was using multithreading for spreading and gathering data, in parallel. It contained a pool of workers that were running on remote computers as independent processes. Calculations could be done by any available worker. In order to use the entire computing power, each computer was running as many processes as the number of processor cores. As the result, computer resources were better utilized and fewer computers had to be used (four times less). Moreover, multiprocessing has no costs of context switching and synchronization. On the other hand, the cost of transmitting data has to be considered.

Results of the tests are shown in Figs. 4 and 5. The two figures present how changing the number of workers affects the time of computations in respect of the number of resources (Fig. 4) and the number of tasks (Fig. 5). The number of tasks added from the initial schedule is the same as in the current schedule. The computation time significantly falls as the number of workers grows. This decline is particularly visible for the small number of resources. It is worth noticing that the computation time was reduced down to even 6% of the sequential computation time for the project with 60 tasks and 12 resources. Generally, if the project contained more resources, the result was obtained faster, because the number of schedule modifications depends on the number of resources. As for the number of tasks, the bigger it was, the bigger the amount of data that had

Figure 4. Time of computations compared with the number of workers for the constant number of tasks

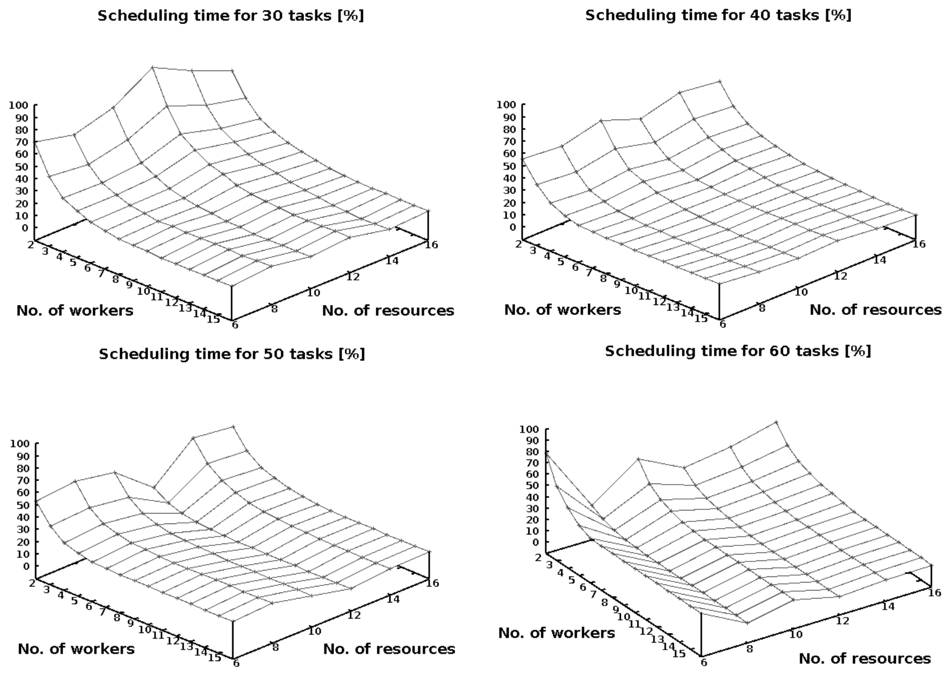


Figure 5. Time of computations compared with the number of workers for the constant number of resources

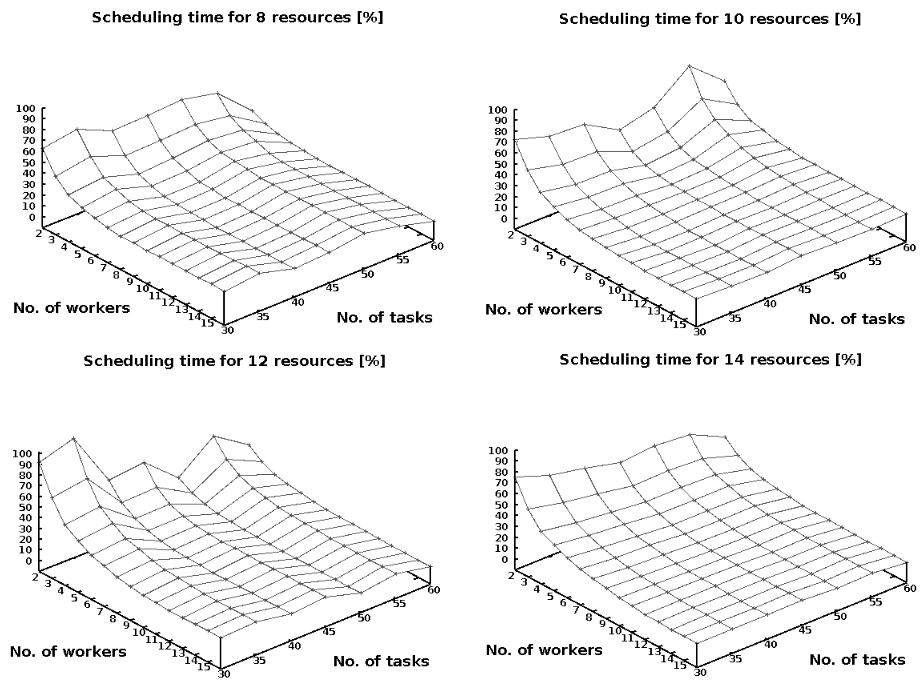
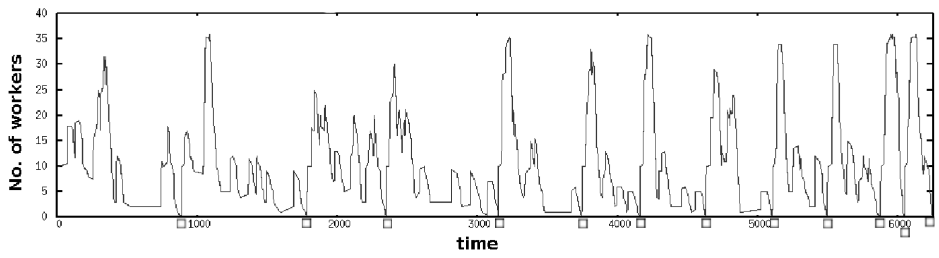


Figure 6. Number of workers used in the schedule refinement method



to be transmitted between the server and the workers. Yet, the computation time fluctuated with a growing trend.

Further tests were meant to examine how many workers should be used to schedule a project in parallel. In the process there are $(N + D)^2 \cdot (R - 1) \cdot R$ schedule modifications, in the worst case. Workers calculate modifications for a resource, so no more than $(R-1)\cdot R$ will be needed at the same time. Test results showed that for a project with R resources a maximum number of $R \cdot 3$ workers is sufficient. Thus, Fig. 6 shows the usage of 36 workers during scheduling of a project with 35 tasks and 10 resources. Squares below the horizontal axis show the end of each of the iterations. The first stage of the iteration requires R resources. If some adding attempt ends, other $R - 1$ workers are used. All of them are busy when several second stages begin. Afterwards, worker usage falls due to waiting for the completion of all second stages. Workers are better utilized in the last iterations because these iterations are shorter. The average number of workers used for project scheduling is shown in Table 3. It was calculated as a weighted mean where weight is proportional to the period of time during which workers were used. Although the average worker usage grows as the number of resources increases, it does not reach $2 \cdot R$. Moreover, it falls as the number of tasks grows because calculations last longer for a greater number of tasks and the need for workers is spread in time. Workers also spend more time waiting when the number of tasks is increased.

6. Conclusions

In this paper, a new constraint for the RCPSP was added. The algorithm can allocate tasks to resources only in particular time periods. It minimizes the project total cost taking into consideration time requirements. We have proposed an improvement of the refinement methods in order to increase the possibility of finding the best resource allocation. Various tests were performed on projects containing 30 new tasks and 30 tasks from the previous schedule. Statistics show that the gain resulting from the application of the algorithm

Table 3. Average number of workers used for project scheduling

	No. tasks			
No. resources	30	35	40	45
6	12.4	4.39	5.67	7.92
8	6.63	6.09	7.04	6.37
10	15.26	10.1	10.06	5.06
12	13.4	11.13	9.98	6.07
14	23.64	13.98	13.77	9.18
16	12.96	16.15	15.21	7.39

varies depending on the number of resources used and the time constraints. It appears that it is more beneficial to use fewer resources and finish the project later. As resource deployment costs are high, using a resource in the project schedule has to be beneficial. Unused resources can then be dedicated to other projects. The later the deadline, the more schedule results are allowed and the flexibility of the algorithm grows. Usually this allows the algorithm to obtain better results. Yet, too low deadline requirements decrease algorithm efficiency. The number of precedence relationships in the project plan is also important. A high number of task dependencies reduces the search space and deteriorates the results.

Experimental results showed a significant reduction of the project cost as well as the project duration, compared to the greedy procedures and to the genetic algorithm. The improvement of refinement methods is suitable for projects with more restricted availability of resources. In such projects, it gave a reduction of the project cost by 8% and of the project duration by 14% when compared with the original algorithm. However, only parallelization of the algorithm may allow for solving efficiently, faster and better, the more complex real life problem. The performance of the algorithm was increased by the distributed processing, in which "workers" on many computers were calculating different schedule modifications in the same time. Hence, the computation time required by the sequential scheduling can be significantly decreased. Even few workers may lower the computation time several times over. A maximum of $R \cdot 3$ workers suffices to reduce the computation time even by the factor of 10. Furthermore, computer resources were well utilized due to multithreading and multiprocessing. The number of computers is inversely proportional to the number of cores. Apart from the project total cost, the algorithm takes into account the changes of the slack time and therefore has the capacity of getting out of the local minimum. Future work will concentrate on both the impact of other factors on reduction of cost and better worker utilization.

References

BOULEIMEN, K. AND LECOCQ, H. (2003) A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple modes version. *European Journal of Operational Research* **149**, 268–281.

BRUCKER, P., KNUST, S., SCHOO, A. AND THIELE, O. (1998) A branch-and-bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research* **107**, 272–288.

DEMEULEMEESTER, E. L. AND HERROELEN, W. S. (1997) New benchmark results for the resource-constrained project scheduling problem. *Management Science* **43**, 1485–1492.

DEMEULEMEESTER, E. L. AND HERROELEN, W. S. (2002) *Project Scheduling. A Research Handbook.* Springer.

DENIZIAK, S. (2004) Cost-efficient synthesis of multiprocessor heterogeneous systems. *Control and Cybernetics* **33**, 341-355.

GOLDRATT, E.M. (1997) *Critical Chain.* The North River Press Publishing Corporation, Great Barrington.

HARTMANN, S. (1998) A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling. *Naval Research Logistics* **45**, 733-750.

HARTMANN, S. AND KOLISH, R. (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research* **127**, 394-407.

KOLISH, R. AND SPRECHER, A. (1996) PSPLIB A project scheduling library. *European Journal of Operational Research* **96**, 205-216.

KOLISH, R. AND HARTMANN, S. (2006) Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research* **174**, 23–37.

MINGOZZI, A., MANIEZZO, V., RICCIARDELLI, S. AND BIANCO, L. (1998) An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science* **44**, 714–729.

RAND, G.K. (2000) Critical chain: the Theory of Constraints applied to project management. *International Journal of Project Management* **18** (3), 173–177.

TUKEL, O.I., ROM, W.O. AND EKSIOGLU, S.D. (2006) An investigation of buffer sizing techniques in critical chain scheduling. *European Journal of Operational Research* **172**, 401–416.