Decision Making in Manufacturing and Services
Vol. 5 • 2011 • No. 1–2 • pp. 5–17

Neighbourhood Properties in Some Single Processor Scheduling Problem with Variable Efficiency and Additional Resources

Mateusz Gorczyca*, Adam Janiak*, Władysław Janiak*

Abstract. In the paper, we consider a problem of scheduling a set of tasks on a single processor. Each task must be preprocessed before it can be started on a processor. The efficiency of preprocessing is variable, i.e., the rate of the task preprocessing depends on the amount of continuously divisible resource allotted to this task. This dependency is given by concave, continuous, non-negative and strictly increasing function of the resource amount. The total consumption of resource at each moment is upper bounded. The objective is to minimize the maximum task completion time. The considered problem is NP-hard. Such a problem appears, e.g., in steel mill systems, where ingots (before hot rolling on the blooming mill) have to achieve the required temperature in the preheating process in soaking pits. Some new properties of the problem are proved. These properties are used to construct the procedure for evaluation of the neighbourhood. The procedure is proposed to improve the efficiency of algorithms based on the neighbourhood concept, such as metaheuristics. The computational experiment is conducted to examine the efficiency of the proposed procedure. The described approach can be easily used in the other discrete-continuous scheduling problems.

Keywords: scheduling, optimization, resource allocation, neighbourhood

Mathematics Subject Classification: 90B35

Revised: 25 August 2011

1. INTRODUCTION

Modern manufacturing and computer systems are so complicated that they cannot be properly described by the classical scheduling theory. Thus, many models have been introduced to scheduling area to reflect real world situations with increasing precision. One of such models assumes that task processing or preprocessing can be described by differential equation. This equation represents the dependance of the task processing or preprocessing rate on the additional, continuously divisible resource available in

* Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Z. Janiszewskiego 11/17, 50-372 Wrocław, Poland. E-mail: {mateusz.gorczyca, adam.janiak, wladyslaw.janiak}@pwr.wroc.pl

a system besides the processors. The problems based on this task model are called discrete-continuous scheduling problems (Józefowska and Węglarz, 1998).

The discrete-continuous scheduling problems received a lot of attention in the literature since 1960's. The mentioned model of task was introduced by Burkov (1966). At first, problems with preemptive tasks were under consideration, mainly with identical and unrelated parallel processors, and an upper bound on the level of renewable resource. The analyzed objective functions contained: makespan (Węglarz, 1989, 1979, 1991), mean flow-time (Węglarz, 1979), and maximum lateness (Węglarz, 1989). Some research was also devoted to problems with a doubly constrained resource (with an additional upper bound on the total amount of the consumed resource) with the makespan criterion (Węglarz, 1991) and the maximum lateness criterion (Węglarz, 1989).

In the past two decades the research was devoted mainly to non-preemptive task models. Among the others, the most frequently analyzed was the problem with parallel processors and the makespan criterion, since methodology for this problem states a framework for the other discrete-continuous scheduling problems (Józefowska and Węglarz, 1998). Properties of this problem were analyzed for cases with ready times (Janiak and Przysada, 1996a; Nowicki and Zdrzałka, 1984), and without them (Józefowska and Węglarz, 1998; Węglarz, 1989, 1979, 1991; Waligóra, 2009). Many methods were designed to solve the problem, from simple heuristics to sophisticated metaheuristics (Józefowska *et al.*, 1998, 2002). Also other problems with parallel processors were analyzed, for objective functions such as: mean flow-time (Józefowska *et al.*, 1997b; Józefowska and Węglarz, 1996), weighted flow-time (Nowicki and Zdrzałka, 1984) and maximum lateness (Janiak and Przysada, 1996b; Józefowska *et al.*, 1997a).

The reason for such a deep interest in the area of discrete-continuous scheduling problems lies in their applicability. These problems are encountered in computer and manufacturing systems. One of the situations is scheduling of tasks in multiprocessor computer systems with a virtual memory. Such a memory is usually divided into frames and their number is large enough to effectively model memory as a continuous resource. The speed of the task processing (program execution) depends on the number of frames allotted to it. The dependence is usually given by the program's lifetime curve (Denning, 1980). This situation was modelled as a discrete-continuous scheduling problem (Węglarz, 1980) and the algorithms were proposed for a memory management in the supercomputer CRAY X-MP (Janiak and Przysada, 1997). The other application is management of the scalable (SPP) and massively parallel (MPP) computer systems, in which the number of processors amounts to hundreds or thousands. Processors of the system can be treated as a continuous resource, whereas other resources like disks or drives are represented as processors in the scheduling problem (Józefowska and Węglarz, 1998).

In the paper we analyze the problem that appears in steel mills during the process of hot rolling of ingots (Janiak, 1989). Before the ingots are hot rolled on the blooming mill, they have to achieve the required temperature in the preheating process in soaking pits. The preheating process is dynamic one and is described by some differential equations, in which the rate of the change of the ingot temperature state at each moment depends on the amount of gas flow intensity allotted to the soak-

ing pit in which ingot is preheated. The ingot preheating time can be shortened by increasing gas flow intensity (i.e. the more gas is consumed the shorter lasts the preheating process). However, the efficiency of the resource utilization is decreasing with gas flow intensity, i.e., longer, slower preheating consumes less resource. Therefore, utilization of resource and the makespan is the trade-off in the described problem. The preheating time can be treated as a ready time of ingot (task) for the main (hot rolling) process on the blooming mill. This situation is described in terms of discrete-continuous scheduling problem in Section 2.

The considered problem was already analyzed in the literature (Janiak and Janiak, 2011). It was proved to be NP-hard and an optimal resource algorithm has been constructed. Usually, the next step is to propose some metaheuristic algorithms, similarly as it was done for other discrete-continuous scheduling problems (Józefowska *et al.*, 1998, 2002; Waligóra, 2009). However, there is an important issue that has not been analyzed for existing metaheuristic algorithms for this class of problems.

As it is well known from the methodology for these problems, the solution consists in two parts: the discrete one (schedule of tasks on processor or processors) and the continuous one (the resource allocation to tasks). The optimal resource allocation can be found for a given discrete part of a solution by solving nonlinear programming problem – see Józefowska and Węglarz (1998). However, it is computationally very demanding. Therefore, some heuristic methods of resource allocation were proposed to make algorithms faster (Waligóra, 2009), but in such a case there is no guarantee of the optimality.

The motivation of this paper is to propose a method that can provide an optimal resource allocation, but strongly decreases the computational effort. The target application of this method is construction and evaluation of neighbourhood in metaheuristic algorithms.

The neighbourhood of a given solution is defined as a set of all solutions that can be obtained by particular (and often slight) change of the given solution. Solution from the neighbourhood in the discrete-continuous scheduling problems is obtained in two steps. First, the change in the discrete part of a given solution is made, and then the optimal resource allocation for the changed discrete part is computed. The second step is computationally very expensive. Thus, our idea is to use a given solution to assess if the selected solution from the neighbourhood can be better than the given one. This can avoid taking the computational effort for solutions that do not guarantee improvement. To sum up, in the paper we present the concept of the neighbourhood evaluation for the discrete-continuous scheduling problems.

In order to apply the idea described above, we prove some properties of the neighbourhood of the considered problem – see Section 4. Before it can be done, some already known properties of the problem needed to be recalled in Section 3. The extensive example is provided in Section 4 to show how the proved properties can be utilized for the evaluation purposes. The procedure of the neighbourhood evaluation is presented in the same section. To examine the efficiency of the proposed procedure, the computational experiment has been conducted. The description of the experiment and the analysis of its results are presented in Section 5. Conclusion and directions for further research are given in Section 6.

2. PROBLEM FORMULATION

The set $T = \{1, \ldots, i, \ldots, n\}$ of non-preemptive tasks is given to be processed on a single processor. For each task $i \in T$ time of processing p_j is given in advance.

Each task $i \in T$ needs some preprocessing before it can be processed on a processor. The preprocessing is described with the following model:

$$\frac{dx_i(t)}{dt} = f_i(u_i(t)),$$

where:

- $x_i(t)$ is the state of preprocessing of task i at time t,
- $u_i(t)$ is the amount of continuous resource allotted to preprocessing of task i at time t,
- $f_i(\cdot)$ is a continuous, concave and increasing function that satisfies $f_i(0) = 0$.

The initial state $x_i(0) = 0$ and the final state $\hat{x}_i > 0$ of preprocessing are given for each task $i \in T$. In order to ready task for processing on a processor, its final state of preprocessing must be achieved, i.e., the ready time r_i of task i is the first moment in which its preprocessing is in the final state:

$$r_i := \min\{t|\ x_i(t) = \hat{x}_i\}.$$

The continuously divisible resource that is available for preprocessing of all tasks is renewable and its level $\hat{U} > 0$ is constant and known in advance. The *resource allocation* is defined as a piece-wise continuous vector function $\mathbf{u}(t) := [u_1(t), \ldots, u_i(t), \ldots, u_n(t)]$.

The resource allocation $\mathbf{u}(t)$ is *feasible* if it satisfies the following condition:

$$\sum_{i=1}^{n} u_i(t) \leqslant \hat{U}.$$

By S_i and $C_i = S_i + p_i$ we denote, respectively, the start and completion time of task i. Moreover, by z and $z(i)$ we denote, respectively, the sequence in which tasks are processed (where z is a permutation of elements of the set T) and the task on i-th position in the sequence z. Tasks are started without any delay, i.e.,

$$S_{z(i)} = \max\{r_{z(i)}, C_{z(i-1)}\},\ j = 1, \ldots, n,$$

where $C_{z(0)} = 0$.

The problem is to find such a sequence z^* and such a resource allocation $\mathbf{u}^*(t)$, for which makespan $C_{\max}(z, \mathbf{u}(t)) = \max\{C_1, \ldots, C_n\}$ is minimized. The considered problem is strongly NP-hard (Janiak and Janiak, 2011).

3. PROPERTIES OF THE OPTIMAL RESOURCE ALLOCATION

In this section we recall the form of optimal solution of the considered problem, which was described by Janiak and Janiak (2011). The properties of solution of this form are the starting point of our considerations and are further analyzed in Section 4. Mostly, these properties concern an optimal resource allocation for a given sequence of tasks. To simplify the notation, we assume in the sequel, without loss of generality, that $z = 1, 2, \ldots, n$, unless we state otherwise.

It has been proved by Janiak and Janiak (2011) that there exists the optimal solution for the considered problem, in which tasks are processed one after another without a break – see Figure 1. Thus, $C_{max} = S_1 + \sum_{i=1}^{n} p_i$ and to find an optimal resource allocation for a given sequence of tasks it is enough to minimize the start time of the first task in a sequence.

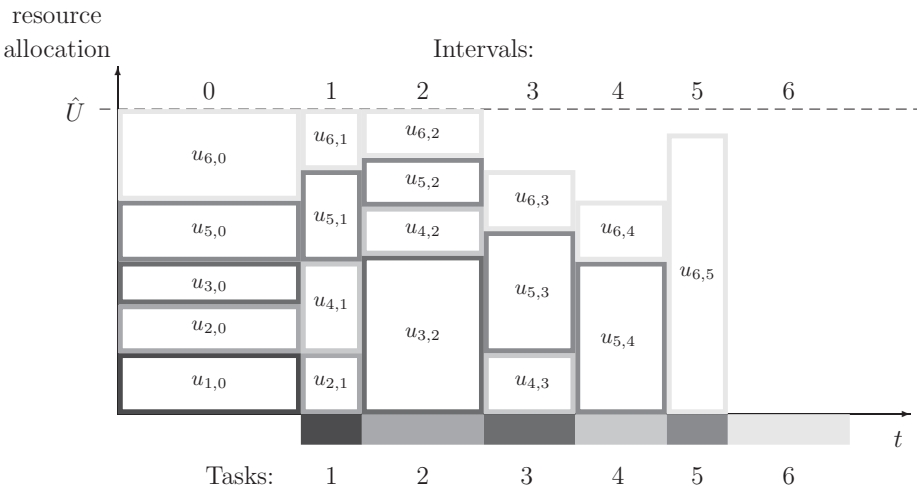


Fig. 1. *The exemplary solution of the form described in Section 3 for a sequence of 6 tasks.*

It has been also proved that in such an optimal solution the resource allocation is constant in time intervals defined by 0 and starting times of consecutive tasks in the sequence z – see Figure 1. Since tasks are processed without a break, the intervals of constant resource allocation are $[0, p_0]$, where $p_0 := S_1$, and

$$\left(\sum_{i=0}^{k-1} p_i, \sum_{i=0}^{k} p_i \right], \ k = 1. \ldots, n-1. \tag{1}$$

In the sequel we consider only such resource allocations.

We denote (constant) amount of resource allotted to the i-th task in the k-th interval by u_{ik} (where $[0, p_0]$ is 0-th interval). From (1) we have:

$$u_{ik} = f_i^{-1}\left(\frac{x_{ik}}{p_k}\right), i = 1, \ldots, n,\ k = i+1, \ldots, n-1, \tag{2}$$

where x_{ik} is the part of the final state of task i processed in k-th interval using resource amount u_{ik}. Notice, that this part cannot be negative, i.e.,

$$x_{ik} \geqslant 0, \quad i = 1, \ldots, n, \quad k = 0, \ldots i-1. \tag{3}$$

In order to finish the task, its parts (of the preprocessing state) have to sum up to the final state:

$$\sum_{k=0}^{n-1} x_{jk} = \hat{x}_j,\ j = 1, \ldots, n. \tag{4}$$

Since the level of the resource available to process all the tasks is upper bounded, the sum of amounts allotted in each interval cannot exceed the value of the bound:

$$\sum_{j=1}^{n} f_j^{-1}\left(\frac{x_{jk}}{p_k}\right) \leqslant \hat{U},\ k = 1, \ldots, n-1, \tag{5}$$

The zeroth interval, i.e., the interval $[0, S_1]$, is obviously of length S_1. It is easy to prove the following property for the zeroth interval.

Property 1. The resource allocation is optimal for a given sequence of tasks if and only if the total amount of the resource allotted in the zeroth interval is equal to the resource level $\hat{U}$.

Proof. It is enough to prove that increasing the total amount of the resource allotted in the zeroth interval we decrease the makespan. Notice, that since $f_i(\cdot)$ is increasing function, $f_i^{-1}(\cdot)$ is increasing as well. Thus, it follows from (2) that increasing the resource amounts u_{ik} (until the total amount is equal to $\hat{U}$) we decrease $p_0 = S_1$ and therefore we decrease $C_{max} = S_1 + \sum_{i=1}^{n} p_i$. ∎

As it follows from (2) and the above property, the length of the zeroth interval can be found based on parts of states in this interval, i.e. $S_1 := G(x_{10}, \ldots, x_{n0})$, by solving the following equation with respect to S_1:

$$\sum_{j=1}^{n} f_j^{-1}\left(\frac{x_{j0}}{S_1}\right) = \hat{U}. \tag{6}$$

The above considerations can be summed up in the following corollary:

Corollary 1. *The optimal resource allocation for a given sequence of tasks can be found by solving the following optimization problem:*

minimize S_1 ***subject to*** *(3)–(5),*

which is denoted in the sequel by OP.

In the sequel, we refer to the solution that consist in sequence z and the optimal resource allocation for this sequence as to the *solution based on sequence* z.

The properties and form of the solution presented in this section are used in the next one to prove the properties of the neighbourhood of a solution based on sequence z.

4. PROPERTIES OF THE NEIGHBOURHOOD

In this section we prove another property of the optimal resource allocation. Using this property we analyze the neighbourhood of a solution based on sequence z. Then, the concept of evaluation of the neighbourhood is presented for a given example of a solution based on sequence z. The section ends with the scheme of procedure of neighbourhood evaluation.

First, we prove the following property.

Property 2. If for a given resource allocation $\mathbf{u}(t)$:

a) in some interval k, $0 \leqslant k < n-1$ not all of the resource is used, i.e.,

$$\sum_{i=k+1}^{n} u_{ik} < \hat{U}, \tag{7}$$

b) in all previous intervals all of the resource is used, i.e.,

$$\sum_{i=l+1}^{n} u_{il} = \hat{U}, \quad l = 0, \ldots, k-1 \tag{8}$$

c) in each previous interval l a positive part of one of the tasks $l+2, \ldots, n$ is processed,

then $\mathbf{u}(t)$ is not optimal.

Proof. Suppose that the resource allocation $\mathbf{u}(t)$ satisfies conditions a) - c) from the thesis. We prove that there exists better resource allocation.

We use induction. As it follows from Property 1, the thesis is true for $k = 0$. We prove that if it is true for $k = j$, then it is true for $k = j + 1$.

Suppose that condition a) is satisfied for the interval $j + 1$ and conditions b) and c) are satisfied for intervals $0, \ldots, j$. Thus, from c) we know that some resource amount is allotted in interval j to some task $i \in \{j+2, \ldots, n\}$. However, as it follows from a), not all of the resource is used in interval $j + 1$. Thus, we can allot some resource amount to task i in the interval $j + 1$ and in the same time decrease the resource amount allotted to this task in the interval j. Resource allocation obtained in this way satisfies conditions a) – c) for $k = j$. ∎

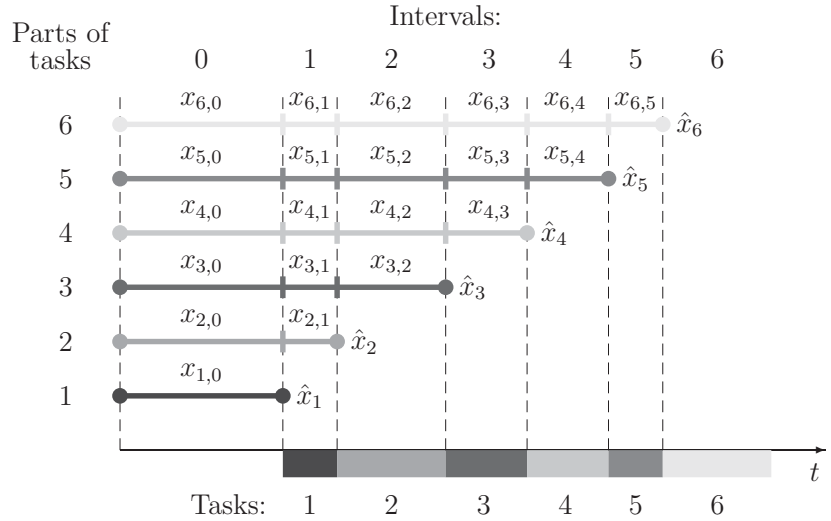


Fig. 2. *The parts of tasks (preprocessing states) corresponding to the resource allocation from Figure 1.*

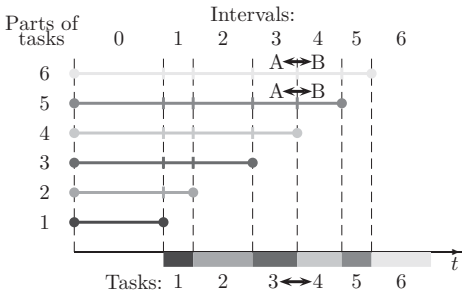
Now we show how to use the above property to evaluate the neighbourhood of a solution based on sequence $z = (1, 2, 3, 4, 5, 6)$. This solution is presented in Figure 1. As it follows from (2) and (6), the solution can be presented in the equivalent way – as a sequence of tasks and the parts of tasks processed in the intervals. In Figure 2 the solution from Figure 1 is presented in such a way.

Now consider the sequence $z' = (1, 2, 4, 3, 5, 6)$ from the neighbourhood of sequence z, obtained by swapping tasks 3 and 4 in sequence z. To find out if solution based on z' is better we can simply solve OP for a sequence z'. However, it would be computationally very demanding. Instead, we use the optimal resource allocation we already have for sequence z.

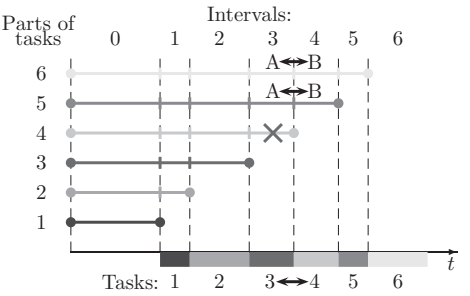
Notice, that since tasks 3 and 4 are swapped, then the lengths of the intervals 3 and 4 are swapped as well. If we swap also parts of the tasks 5 and 6, then all the constraints for tasks 5 and 6 remain satisfied – see Fig. 3.a). However, the part of task 4 cannot appear in interval 3, since task 4 starts earlier in z' – see Fig. 3.b). On the other hand, since task 3 starts later, its part can appear in interval 3 – see Fig. 3.c).

Thus, the part of this task processed in some other interval can be decreased, e.g., in the interval 2 – see Fig. 3.d). Notice, that since we lost the part of task 4 in interval 3 then at least one part of this task must be increased in one of the earlier intervals, e.g., in interval 2 – see Fig. 3.e).

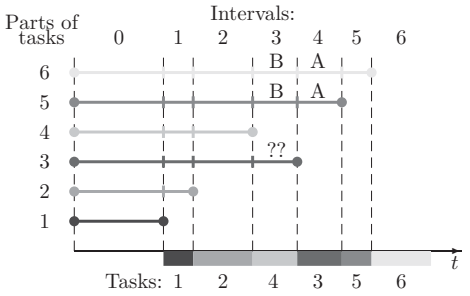
In this way we obtained the set of tasks parts that need to be recalculated to find the feasible resource allocation for sequence z' based on the optimal resource allocation for sequence z. The tasks parts from this set are denoted by the double question marks in Figure 3.e).



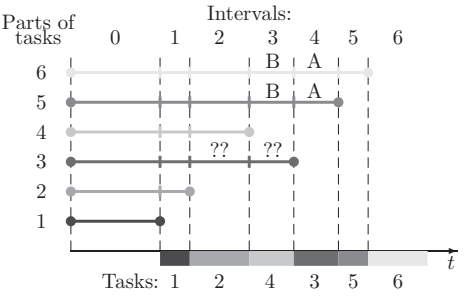
a) The swap of tasks 3 and 4 and parts of tasks 5 and 6 in intervals 3 and 4



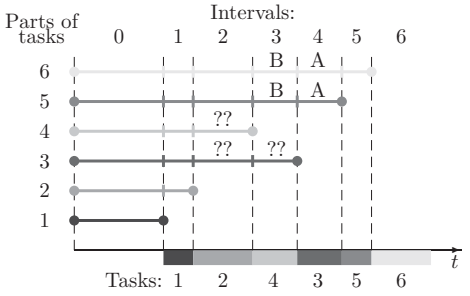
b) The part of task 4 cannot be processed in interval 3 after swap.



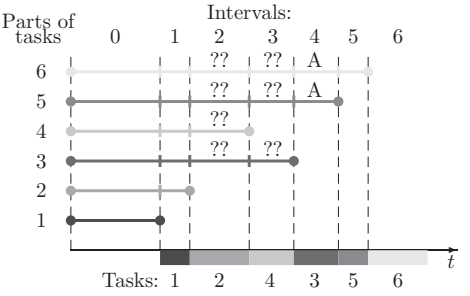
c) However, the part of task 3 can be processed in interval 3 after swap.



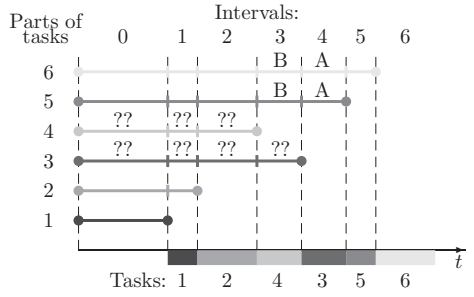
d) Thus, the part of task 3 can be decreased in the other interval.



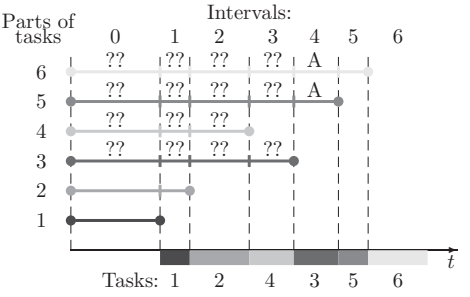
e) Since the part of task 4 from interval 3 is lost, some other part of this task must be increased.



f) All parts of tasks from intervals 3 and 4 can be recalculated to improve the evaluation.



g) Another idea is to recalculate all parts of tasks 3 and 4.



h) Any subset of parts of tasks which includes at least one part of task 4 can be recalculated.

Fig. 3. *The concept of a neighbourhood evaluation described in Section 4.*

Obviously, the feasible resource allocation in which only this three tasks parts need to be changed may not exist. Thus, we can recalculate some larger set of tasks parts. For example, we can recalculate all tasks parts in intervals 2 and 3 – see Fig. 3.f), or all tasks parts of tasks 3 and 4 – see Fig. 3.g). In fact, any set of tasks parts including at least one part of task 4, like for example set the set from Figure 3.h), can lead to the feasible resource allocation. The bigger is the set of recalculated tasks parts, the higher are the chances to find the feasible resource allocation for sequence z', but also higher is the computational effort needed for recalculation.

Suppose that conditions b) and c) from Property 2 are satisfied for intervals 0, 1, 2, 3 and 4 of the optimal resource allocation for sequence z. Thus, it follows from Property 2 that if through recalculation we obtain the feasible resource allocation and in one of these intervals not all of the resource is used, then there exists solution based on sequence z' better than the considered solution for sequence z. To find this better solution we need to solve OP. However, we can significantly decrease the computational complexity of this operation by using the obtained feasible resource allocation as a starting point in the nonlinear programming procedure.

To sum up, our concept of evaluation of the neighbourhood can be formalized as the following procedure:

Input: The solution based on a given sequence z.
Output: The evaluation value for all and a feasible resource allocation for some sequences from the neighbourhood of z.

0. Construct the set of all sequences from the neighbourhood of sequence z. For each sequence from this set perform the following steps:

1. Select the set of recalculated tasks parts.
2. Recalculate the tasks parts from the set to obtain the feasible resource allocation.
3. Based on the recalculation results, calculate the evaluation value for the sequence.

Step 2 of the above procedure is very general and can be performed in many ways. One of them is to solve the optimization problem similar to OP. The decision variables of this problem are tasks parts selected in Step 1. The constraints are calculated based on the values of tasks parts that are not recalculated and equations (3)–(5). The objective is the difference between resource level and the resource needed to process (recalculated and not recalculated) tasks parts in the first interval that contains recalculated variable.

Step 3 is also general. For the way of performing Step 2 described above, the value of the evaluation can be value of the objective of the mentioned optimization problem. Notice, that this value is negative if there is no feasible resource allocation. Therefore, the sequence for which the value is the highest can be considered as the most promising.

To examine the efficiency of the proposed procedure we have conducted computational experiment described in the next section.

5. COMPUTATIONAL EXPERIMENT

The procedure has been examined for 100 instances of the problem with 8 tasks. As function (1) we have used $f_i(u_i) = c_i \cdot u_i^{1/\alpha_i}$, $\alpha_i \geqslant 1$, considered to be a good benchmark function for discrete-continuous scheduling problems (Janiak and Janiak, 2011; Józefowska and Węglarz, 1998; Józefowska *et al.*, 1998). Parameter α_i has been chosen randomly with uniform distribution from the set {1,2}, coefficient c_i from the interval [1, 50], and the final state $\hat{x}_i$ and the task processing time from the interval [1, 10]. Procedure has been also examined for the instances where the task processing time has been chosen randomly from the intervals [1, 5] and [1, 50], but the results have been similar as for the interval [1, 10].

Experiment has been performed for the neighbourhood based on a swap operation, i.e., the neighbourhood for a given sequence z consists in all sequences that can be obtained by swapping two consecutive tasks in z.

We have examined improvement defined as a relative difference between objective value for a given solution and the objective value computed for the most promising sequence from the neighbourhood. As the objective value we have taken the start time of the first task in the sequence, which, as it is shown in Section 3, can be considered as equivalent to a criterion value. This start time has been given for the optimal resource allocation obtained by solving OP. To solve OP we have used the Sequential Quadratic Programming (SQP) method, which is reported as one of the best in constrained nonlinear programming (Schittowski, 1985). We used the implementation of the SQP method available in Optimization Toolbox 4.3 of the Matlab R2009b environment. The procedure has been run with default settings. The starting values of task parts has been generated by equal distribution of the final state of each task between all intervals in which this task is preprocessed, i.e., $x_{ik} = \hat{x}_i/i, \;\; i = 1, \ldots, n, \;\; k = 0, \ldots, i-1$.

The most promising sequence from the neighbourhood has been chosen by using our evaluation procedure described in Section 3. Step 2 and 3 of the procedure have been performed in a way described at the end of Section 4. To solve the optimization problem in Step 2 of the procedure we have also used the SQP method with default settings. The starting values of recalculated tasks parts have been set to 0. If the feasible solution could not have been found by the procedure, the evaluation value for the considered sequence from the neighbourhood has been set to $-\infty$.

The improvement for the set of recalculated variables defined in a way similar to the presented in Figure 3.e), 3.f), 3.g) and 3.h), has been equal to, respectively, 35.8%, 40.4% 40.9% and 49.5%. The time of performance of the evaluation procedure was, respectively, 38.4, 16.6, 31.3 and 7.4 times shorter than the average time needed to find the optimal resource allocation for the original sequence.

As it follows from the above results, the proposed procedure can quickly and efficiently evaluate the neighbourhood for a given solution. Notice, that even for the small set of recalculated tasks parts the better solution is frequently detected. However, the time of performance increases with the number of recalculated tasks parts faster than the quality of improvement. The trade-off between these two values should be subject of further investigation, since the presented experiment is undoubtedly only preliminary research.

6. CONCLUSION

In the paper we analyzed the single processor scheduling problem with variable efficiency of preprocessors. The rate of the task preprocessing depends on the amount of continuously divisible resource allotted to this task. The minimized criterion is the makespan. The problem is known to be NP-hard.

The considered problem is one of many difficult discrete-continuous scheduling problems. Our objective was to find out if the neighbourhood of the discrete-continuous scheduling problem can be analyzed in terms of continuous part of the solution. Until now, the neighbourhood was considered only for the discrete part of the solution. To be more precise, we were interested if based on a continuous part of a given solution we can evaluate solutions from the neighbourhood without computing the optimal resource allocation for them.

We proved properties of the problem that allowed to construct the procedure for evaluation of the neighbourhood. The procedure was examined in the computational experiment. The results of the experiment shown that the presented idea leads to efficient and fast method which can significantly improve the metaheuristic algorithms for discrete-continuous scheduling problems.

Conducted research is only a preliminary step. More advanced experiments for different types of neighbourhood are needed. Also problems with the discrete part of the solution different than a sequence of tasks are potential application of the proposed method. Another research is suggested to apply the presented idea in different metaheuristic algorithms and possibly other algorithms based on the concept of neighbourhood.

REFERENCES

Burkov, V., 1966. Resource allocation as a time-optimal control problem. (in Russian). *Avtomat. i Telemeh.*, **27**(2), 114–129.

Denning, P. J., 1980. Working sets past and present. *IEEE Transactions on Software Engineering*, **6**(1), 64–84.

Janiak, A., 1989. Minimization of the blooming mill standstills mathematical model – suboptimal algorithms. *Zeszyty Naukowe Akademii Górniczo-Hutniczej s. Mechanika*, **8**(2), 37–49.

Janiak, A., Janiak, W., 2011. Single-processor scheduling problem with dynamic models of task release dates. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, **41**(2), 264–271.

Janiak, A., Przysada, J., 1996a. Czasowo-optymalne szeregowanie zadań z różnymi terminami gotowości na równoległych maszynach. *Zeszyty Naukowe Politechniki Śląskiej s. Automatyka*, **117**, 99–109.

Janiak, A., Przysada, J., 1996b. Zastosowanie systemów wieloprocesorowych w zrobotyzowanych systemach sterowania. In *Materiały V Krajowej Konferencji Robotyki*, vol. 2, 149–158.

Janiak, A., Przysada, J., 1997. Algorytmy szeregowania zadań i rozdziału zasobów na procesorach równoległych - przykłady zastosowań. In *Materiały III Konferencji Komputerowe Systemy Wielodostępne*, 101–107.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., and Węglarz, J., 1997a. Discrete-continuous scheduling to minimize maximum lateness. In *Proceedings of the Fourth International Symposium on Methods and Models in Automation and Robotics MMAR'97*, 947–952.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., and Węglarz, J., 1997b. Discrete-continuous scheduling to minimize the mean flow time – computational experiments. *Computational Methods in Science and Technology*, **3**, 25–37.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., and Węglarz, J., 1998. Local search metaheuristics for discrete-continuous scheduling problems. *European Journal of Operational Research*, **107**(2), 354–370.

Józefowska, J., Waligóra, G., and Węglarz, J., 2002. Tabu list management methods for a discrete-continuous scheduling problem. *European Journal of Operational Research*, **137**(2), 288–302.

Józefowska, J., Węglarz, J., 1996. Discrete-continuous scheduling problems - mean completion time results. *European Journal of Operational Research*, **94**(2), 302–309.

Józefowska, J., Węglarz, J., 1998. On a methodology for discrete-continuous scheduling. *European Journal of Operational Research*, **107**(2), 338–353.

Nowicki, E., Zdrzałka, S., 1984. Optimal control policies for resource allocation in an activity network. *European Journal of Operational Research*, **16**(2), 198–214.

Schittowski, K., 1985. Nlqpl: A fortran-subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, **5**, 485–500.

Waligóra, G., 2009. Tabu search for discrete-continuous scheduling problems with heuristic continuous resource allocation. *European Journal of Operational Research*, **193**(3), 849–856.

Węglarz, J., 1979. Project scheduling with discrete and continuous resources. *IEEE Transactions on Systems, Man and Cybernetics*, **9**(10), 644–650.

Węglarz, J., 1980. Multiprocessor scheduling with memory allocation – a deterministic approach. *IEEE Transactions on Computers*, **29**(8), 703–709.

Węglarz, J., 1989. Scheduling under continuous performing speed vs. resource amount activity models. In *Advances in Project Scheduling*, Słowiński, R. and Węglarz, J., editors. Elsevier Science Publishers, Amsterdam, 273–295.

Węglarz, J., 1991. Synthesis problems in allocating continuous, doubly constrained resources among activities. In *Proceedings of the XI Triennal International Conference on Operational Research IFORS'90*, Pergamon Press, Oxford, 715–724.