HUNGARIAN JOURNAL
OF INDUSTRY AND CHEMISTRY
VESZPRÉM
Vol. 41(1) pp. 27–34 (2013)

IMMEDIATE EVENT-AWARE MODEL AND ALGORITHM OF A GENERAL SCHEDULER

TIBOR DULAI[1]✉, ÁGNES WERNER-STARK[1], AND KATALIN M. HANGOS[1,2]

[1] Department of Electrical Engineering and Information Systems, University of Pannonia, Egyetem str. 10, Veszprém, H-8200, HUNGARY
[2] Process Control Research Group, Computer and Automation Research Institute, Kende u. 13-17, Budapest, H-1111 HUNGARY
✉E-mail: dulai.tibor@virt.uni-pannon.hu

A stochastic scheduling problem is investigated in this work that considers workpieces to be manufactured according to individual recipes containing manufacturing steps performed by workstations as resources. Unexpected stochastic breakdown of a workstation or the faulty termination of a recipe, when a manufacturing failure renders the workpiece out of specifications, forms the set of immediate events. A model and an algorithm are proposed as the basis of a scheduler, which takes into account the possible immediate events, estimates their probability and suggests resource allocations which provide the best overall work-flow even when an immediate event happens. This model includes the possibility of handling alternative resources that can substitute each other in case of an immediate event, like sudden technical failure of a resource. Immediate events are not exactly predictable; however, based on previous experiences, their probabilities can be estimated. Our model uses the properties of the resources (including how they can substitute other types of resources) and the required sequence of them during the workflow (i.e. the recipes). The proposed scheduling algorithm constructs a solution workflow that reacts in the best way (in average) even for an unexpected event. The proposed model and scheduling algorithm is illustrated on two industrial case studies.

Keywords: scheduling, resource allocation, alternative resources, immediate event-awareness

Introduction

Scheduling is an important and widely used topic of operations research. Besides of its theoretical importance, industries can also benefit from optimal schedules and resource allocation. Several different algorithms were established for organizing process elements on the time scale [1, 2], related to for example computer networks [3], business processes [4, 5] or industrial processes [6]. The results are usually represented on a Gantt chart.

There are also differences between these algorithms related to their application times. Some of them are applied offline before the scheduled processes starts, others are applied real-time. Real time scheduling techniques, and a special adaptive real-time scheduler is introduced in Ref. [10]. For improving their results, there are cases when the methods use historical data during the creation of the schedule [11].

The common scheduling methods usually handle the resources individually and do not take into account the relationships, e.g. the similar functionality between them. Only few publications investigate the cooperation possibilities that are enabled by the similar functionality of resources in scheduling [7].

In this paper a scheduling method is proposed, where resources may substitute each other and immediate failures of resources may happen. After we introduce the problem and its main building blocks, we present our model and the algorithm developed. Its operation and properties are demonstrated on a simple and a more complex problem as case studies.

Problem specification

A general scheduler intends to determine the placement of activities of resources on the time scale. In an advanced case one might consider additional features given in the problem's model – like substitution possibility of the resources – which help to redefine the classic scheduling problem to be usable in different real-life applications. This substitution may help in achieving a certain fault tolerance property in such a way that a technical failure causes the least possible negative effect. The basic aspects and sub-tasks related to this extended scheduling problem are collected in this section.

Cooperation

In some cases more than one resource is able to carry out a particular activity (usually with different productivity). It means that when the appropriate resource is busy, there can be another resource that is able to take over a particular task if required. We shall term this case a cooperative situation when substitution is possible, In other words, the resources can cooperate with each other. Taking the cooperation aspect into account during scheduling the performance may increase [8]. We introduced the so called substitution vector as an element of our model to handle substitution related sub-problems.

Any scheduler [1, 2] can be applied as a basis of the cooperative extension, which inputs the sequence of resources as input processes and has to arrange them on the time scale taking into account the availability and temporal constraints. All the improvements are done on this basic schedule.

Improving fault tolerance

When a schedule is ready and the workflow starts, an immediate event (e.g. a technical failure of a resource) in a process may influence other processes, too, in a negative way [9]. If the effects of that event are calculated in the schedule, then the solution with the best answer to immediate events can be selected from the set of solutions with the same performance, and the faults' negative impact on the schedule decreases. Our model deals also with this aspect.

The model

For the above mentioned problem set, we created an universal model and an algorithm to carry out the scheduling. The work intends to be the basis of several different tasks with the goal that the operation time should be minimal. In this section we introduce our model and the scheduler that works on this model.

Main parameters, notations and functions

The model and its parameters are designed in such a way that it can be applied for different kinds of scheduling problems (e.g. scheduling of industrial production processes, test processes, scheduling and managing the resources of electrical networks, etc.). For this reason we collected the necessary parameters that make possible the development of a general framework, which uses different kinds of resources in different processes. It is able to handle different needs. In this sub-section we introduce the main parameters and functions we use in our work. These are as follows:

$$\mathbf{Process} = \{proc_1, \ldots, proc_j\}, \tag{1}$$

stands for the set of processes;

$$\mathbf{P} = \{p_1, \ldots, p_l\}, \tag{2}$$

stands for the set of product types;

$$\mathbf{R} = \{r_1, \ldots, r_m\}, \tag{3}$$

stands for the set of resources;

$$\mathbf{O}(r_i) = \{o_{i1}, \ldots, o_{ik}\}, \tag{4}$$

stands for the set of the operation modes of resource i;

$$\mathbf{A} = \{a_1, \ldots, a_n, pause(\mathrm{t})\}, \tag{5}$$

stands for the set of basic activities, where $pause(t)$ is an empty activity with length of t hour;

$$ra : \mathbf{R} \times \mathbf{A} \rightarrow \{0, 1\}, \tag{6}$$

defines a function for determining whether a resource is able to perform an activity;

$$rap: \mathbf{R} \times \mathbf{A} \times \mathbf{P} \rightarrow \{0, 1\}, \tag{7}$$

defines a function for determining whether a special activity of a resource can be applied for a product type;

$$t: \mathbf{R} \times \mathbf{A} \times \mathbf{P} \times \mathbf{O} \rightarrow \mathbf{N}, \tag{8}$$

provides the suggested operation time of a resource in a given operation mode performing a given activity on a given product type;

$$q: \mathbf{R} \times \mathbf{A} \times \mathbf{P} \times \mathbf{O} \rightarrow [0,100], \tag{9}$$

provides capacity information: how many percent of a resource capacity is occupied by one piece of a given product type in a given operation mode of a resource while performing a given activity;

$$e: \mathbf{R} \times \mathbf{A} \times \mathbf{P} \times \mathbf{O} \rightarrow [0,100], \tag{10}$$

provides the probability of resource failure during performing a given activity on a given product type in a given operation mode;

$$s: \mathbf{R} \times \mathbf{A} \times \mathbf{P} \times \mathbf{O} \rightarrow \{0, 1\}, \tag{11}$$

results in a binary decision: whether a resource activity in a given operation mode on a given product type can be suspended without restarting it from its beginning;

$$rreq: \mathbf{R} \times \mathbf{A} \times \mathbf{P} \times \mathbf{O} \times \mathbf{DateTime} \rightarrow \mathbf{P}(\mathbf{R}, \mathbf{N}) \tag{12}$$

(power set on pairs of a resource and a natural number), provides the additional resource need of a resource's given activity on a given product type in a given operation mode in a given hour

$$subst(r_i, a_j, p_k, o_{il}) = [\ h_{i1} \ \ \ldots \ \ h_{im}\ ], \tag{13}$$

is the substitution vector, where m is the number of resources, h_{in} is a natural numbers for all $1 \leq n \leq m$, r_i is the i^{th} resource, a_j is a basic activity, p_k is a product type, o_{il} is an operation mode of resource i, h_{in} denotes how many percent of the productivity of resource i is needed in a given activity in a given operation mode on a given product type to substitute totally resource n supposing unchained operation time. The substitution vector is calculated based on the function *ra* and other functions (e.g. function t in case of time-based optimization).

$$map: \mathbf{R} \text{ x } \mathbf{DateTime} \rightarrow \mathbf{N}, \tag{14}$$

provides the information on accessibility of resources: how many resource of a given type is accessible in a given hour;

$$\mathbf{F}: \mathbf{R} \rightarrow \mathbf{N}, \tag{15}$$

provides the expected number of hours how long a given resource is unavailable in case of its failure;

$$t_{\text{start}}: \mathbf{Process} \rightarrow \mathbf{DateTime}, \tag{16}$$

shows the start time of a process;

$$t_{\text{maxend}}: \mathbf{Process} \rightarrow \mathbf{DateTime}, \tag{17}$$

shows the maximum finish time of a process;

$$prev: \mathbf{Process} \times \mathbf{A} \times \mathbf{N}^{+} \rightarrow \{\mathbf{A}, \emptyset\}, \tag{18}$$

provides the prior basic activity of the n^{th} occurrence of a given basic activity in a given process;

$$next: \mathbf{Process} \times \mathbf{A} \times \mathbf{N}^{+} \rightarrow \{\mathbf{A}, \emptyset\}, \tag{19}$$

provides the following basic activity of the n^{th} occurrence of a given basic activity in a given process;

$$maxdelay: \mathbf{Process} \times \mathbf{A} \times \mathbf{N}^{+} \rightarrow \mathbf{N}, \tag{20}$$

provides the maximal duration of time out, which is tolerated by the n^{th} occurrence of a given basic activity in a given process;

$$dur: \mathbf{Process} \times \mathbf{A} \times \mathbf{N}^{+} \rightarrow \mathbf{Q}, \tag{21}$$

provides the time scale, which number the default operation time of the of the n^{th} occurrence of a given basic activity in a given process has to be multiplied with, for getting the real operation time of the activity.

The proposed algorithm

Our algorithm was created to take into account the cooperation possibilities of the resources and to have failure-aware behaviour during scheduling. The algorithm can be separated into three main blocks, as it is shown in *Fig.1*.

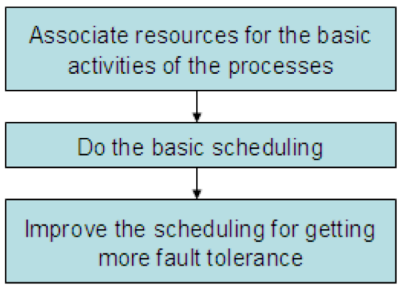
Associate resources for the basic activities of the processes
Do the basic scheduling
Improve the scheduling for getting more fault tolerance

Figure 1: The basic parts of our algorithm

In the first phase, we associate resources for each basic activity of the processes. At the start, processes are described only as a sequence of basic activities. We have to turn it into the "language" of resources. In this task, we use the function *ra* for determining, which resources are able to carry out the desired basic activity. Furthermore, we deal with the minimization of the operation time; we select the resource, which has the operation mode with the minimal operation time, i.e. r_i with

$$\min(t(\text{r}_\text{i}, \text{A}, \text{P}, \text{o}_\text{il})), \forall o_{il} \in O(r_i), ra(r_i, A) = 1 \tag{22}$$

is selected.

The first step of our algorithm results in a sequence of resources for each process. The second phase of the algorithm is performed by the main scheduler task, taking into account the alternative resources.

The first task in this phase is to select the "basic" process, which has the least robustness. We do it by selecting $proc_i$ with

$$\min(t_{\max end}(proc_i) - t_{start}(proc_i) - TDUR), \tag{23}$$

where *TDUR* is the sum of the durations of all of the basic activities of $proc_i$.

After the determination of the basic process, the algorithm enters a loop, in which it selects a process from the set of the remaining processes and attempts to place all the basic activities of the selected process element-by-element on the time scale. During this operation the algorithm handles the substitutability of the resources. The insertion of a basic activity starts at the initial activity of the process and each activity is inserted into the earliest possible time point. This assumption is a fundamental point in the method. The insertion of an activity at its earliest time may have two possible outcomes:

- successful insertion
- unsuccessful insertion, which means that there is not enough amount of the resource to serve the activity which starts at its earliest time point. We call this case a collision.

In case of a collision, the algorithm attempts to find an alternative resource using the substitution vector,

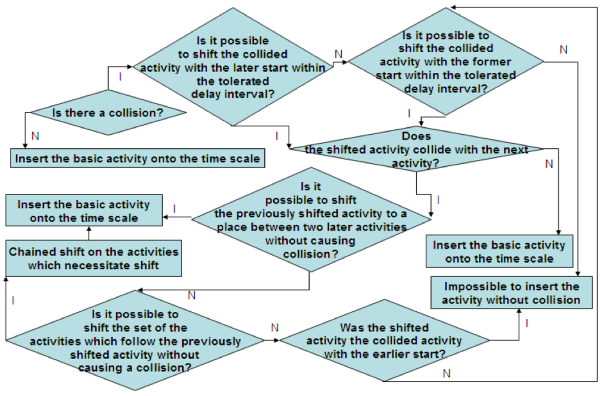
Is it possible to shift the collided activity with the later start within the tolerated delay interval?
Is it possible to shift the collided activity with the former start within the tolerated delay interval?
Is there a collision?
Insert the basic activity onto the time scale
Does the shifted activity collide with the next activity?
Insert the basic activity onto the time scale
Is it possible to shift the previously shifted activity to a place between two later activities without causing collision?
Chained shift on the activities which necessitate shift
Insert the basic activity onto the time scale
Impossible to insert the activity without collision
Is it possible to shift the set of the activities which follow the previously shifted activity without causing a collision?
Was the shifted activity the collided activity with the earlier start?
I
N

Figure 2: Activity insertion onto the time scale

which has enough leftover capacity to carry out the activity at the desired earliest time point. If this is not possible, because there is not enough resource at the desired time, the algorithm tries to shift one of the collided activities to solve the problem. The detailed mechanism of the activity-insertion task is illustrated in *Fig.2*. When the activity-shift resulted in no success, the algorithm attempts to repeat the same process with other alternative resources. If there is no success, the whole process will be left from the final solution; otherwise the initial scheduling is ready.

In the final, third phase of our algorithm the schedule is tried to be modified in such a way that it should have a higher fault tolerance than the original one. This phase assumes that fault happens at the processing of special activities of the processes and calculates its effect into the schedule. We do not want to hurt the time constraints of the original problem that's why one of the main parameters of this phase are the maximum finish time values *(t_{maxend})* of the processes. We process the processes starting from their end in the following way:

Let **WP** be the process list we work on. At the start **WP** = **Process**. For each element of **WP**, we create a pointer, which points at the penultimate activity of the process. The activities pointed by the pointers (let their set be **WA**) are candidates for assuming them to have fault. Starting from this initial state, this phase of the algorithm works as follows:

1. Select the a_j activity from **WA** with the highest error probability:

$$\max(e(r_i, a_j, p_k, o_{il})),\ a_j \in WA \tag{24}$$

2. If the start of the basic activity which follows a_i can be shifted by a_i's duration plus by $F(R(a_i))$, where $R(a_i)$ is the resource which carries out a_i without causing collision and without exceeding $t_{\mathrm{maxend}}(proc_k)$, where $proc_k$ is the process which belongs a_i to, then
(i) the shift will be done, and
(ii) the basic activity, which precedes a_i in the process will be added to **WA** and a_i will be taken out from there. It means that the pointer of the process will be set one step backward.

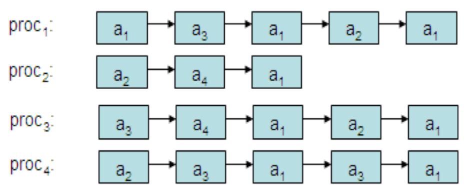

Figure 3: Activity sequence of the example processes

Else, a_i is taken out from **WA** and its process is taken out from **WP**, too. It means that the activities of the process are not modified any more.

This phase of the algorithm modifies the placement of the basic activities of the processes on the time scale in such a way that there is no need to change the scheduled activities in case of failures with high probability. It means that these failures will not cause delay and exceed of the maximum finish time.

Case studies

In this section, the previously presented algorithm is applied to two hypothetical manufacturing examples: we show how it takes into account the substitutability of the resources and how to make the schedule ready for possible technical failures.

A simple problem and its solution

Let there be four processes with the same start and maximum finish time ($t_{\mathrm{start}} = 0$ and $t_{\mathrm{maxend}} = 65$ are identical in each cases). The sequences of the process activities can be seen in *Fig.3*.

In this example, we work only with one product type (P), and we intend to minimize the maximum operation time. Let's suppose that in this example there are two pieces of resource r_1, while only 1 piece of the others, and

$$ra(r_i, a_j) = 1 \text{ only if } i=j, \tag{25}$$

excepting two cases:

$$ra(r_2, a_3) = 1 \tag{26}$$

and

$$ra(r_3, a_2) = 1, \tag{27}$$

as well. Moreover, the substitution vector of r_2 is:

$$subst(r_2, a_3, \mathbf{P}, \mathbf{O}) = [\ 0 \quad 100 \quad 125 \quad 0\ ], \tag{28a}$$

while the substitution vector of r_3 is:

$$subst(r_3, a_2, \mathbf{P}, \mathbf{O}) = [\ 0 \quad 200 \quad 100 \quad 0\ ], \tag{28b}$$

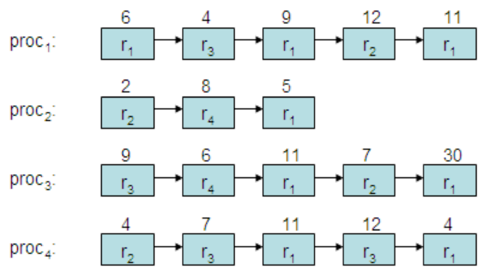

Figure 4: Sequence of the initial resource allocation

which means that r_2 can substitute r_3, and r_3 can substitute r_2. However, the substitution results in 125 percent or 200 percent more time, respectively.

Another assumption is when only r_1 has two operation modes: in one of its operation mode its operation time at a_1 activity on product **P** can be reduced with 10%. As our goal is to minimize the necessary time, we will use this operation mode of r_1 in each case. Moreover, assume that all the resources are unavailable for 2 time units in case of failure.

Based on this information, the first step of our algorithm creates the sequence of the resources for each process. This sequence completed with the durations of each activity after user-based modification of the default operation times is illustrated in *Fig. 4*. The second phase of the algorithm determines the basic process; in our case it is $proc_3$ with its 63 unit length. Its representation on the time scale is shown in *Fig.5*.

After this selection a loop is started, choosing the less robust process from the remaining set. The next process is $proc_1$. Its first activity can be placed onto the time scale without collision. The placement of the second activity collides with the basic process's first activity. The first reaction of the algorithm is to search for alternative resource. Resource r_2 can substitute r_3 without collision; however in this case the operation time will increase from 4 to 5 units. The third activity of $proc_1$ can be inserted, since there are two pieces of resource r_1. The fourth activity collides again; however substitution solves this problem, and the problem-free placement of the final activity of the process can be seen in *Fig.6*.

The third process to handle is $proc_4$. Its first activity can be placed easily; however, the second activity has collision and there is no possibility to substitute it without collision either. This case requires the shift of the activity as illustrated by *Fig.2*. The activity that shifts the later start is reason for a pause was defined in $proc_4$. The third and fourth activities have to be also shifted. The difference between the two cases is the resource that is able to substitute gets free earlier. This is why the fourth activity will be carried out by r_2 instead of r_1, started after a short pause. After the placement of the process's final activity and all the activities of $proc_2$,, this provided the schedule with some activities of the previously placed processes shifted necessarily twice, as illustrated in *Fig.7*.

The final phase of the algorithm is to make the schedule to be fault-aware. As t_{maxend} = 65 and the length of $proc_3$ is 64, we will not modify this process. Similarly, we will not modify $proc_1$ and $proc_4$ either. Only $proc_2$ lets the algorithm to prepare it to be fault-aware, and the modification can be applied to all of its

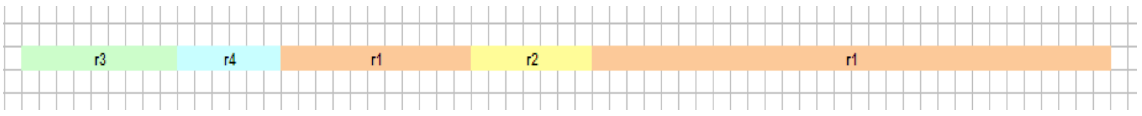

Figure 5: Placement of the basic process onto the time scale

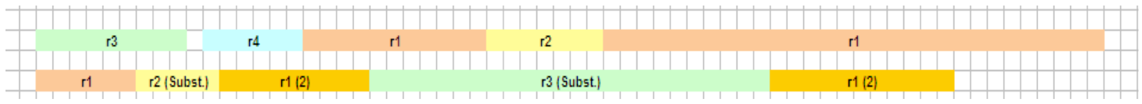

Figure 6: The schedule after handling *proc*1

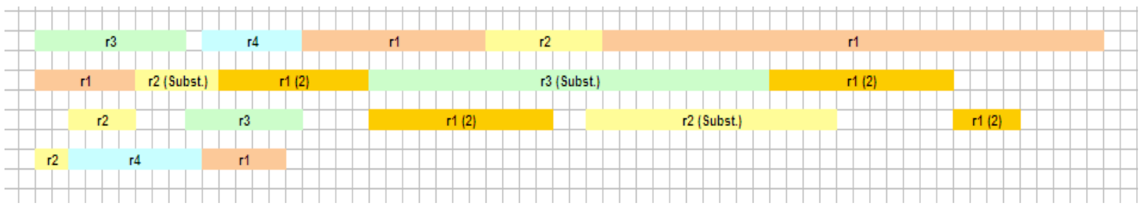

Figure 7: The schedule after the second phase of the algorithm

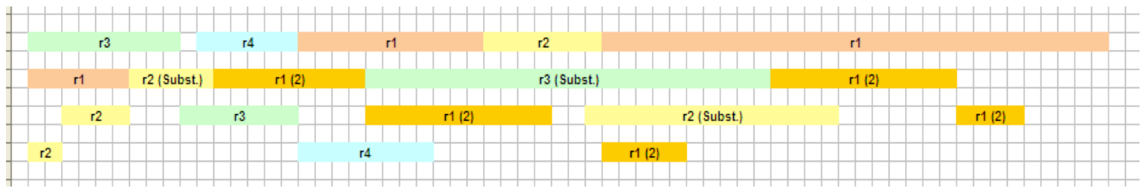

Figure 8: The final schedule

Table 1: Some properties of the processes of the second case study

process	start time, t_{start}	max. finish time, t_{maxend}
p_1	0	30
p_2	15	30
p_3	7	30
p_4	3	25
p_5	0	23
p_6	6	29

Table 2: The effect of failure on the second example's resources

Resource	*r*1	*r*2	*r*3	*r*4	*r*5	*r*6
Time of unavailability (F) [time unit]	2	1	3	2	2	1

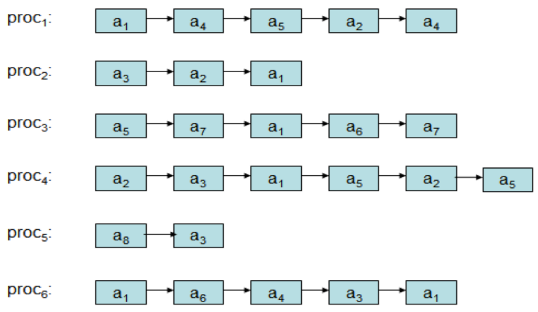

Figure 9: The activities of the second example's processes

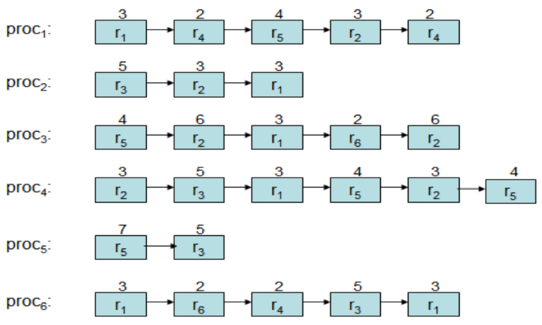
proc1: 3 r1, 2 r4, 4 r5, 3 r2, 2 r4
proc2: 5 r3, 3 r2, 3 r1
proc3: 4 r5, 6 r2, 3 r1, 2 r6, 6 r2
proc4: 3 r2, 5 r3, 3 r1, 4 r5, 3 r2, 4 r5
proc5: 7 r5, 5 r3
proc6: 3 r1, 2 r6, 2 r4, 5 r3, 3 r1

Figure 10: Resource-sequence of the example's processes

activities. The result can be seen in *Fig.8*.

A more complex problem and its solution

In this example we deal with six processes. In contrast with the previous example, they differ in their start and maximum finish time as shown in *Table 1*, where the values are represented in time units. The activity-sequences of the processes are illustrated in *Fig.9*.

In the example form *Fig.9*, eight different activities are applied. The next question is which resources are able to carry out these activities. In this example, there are six kinds of resources, six resource types (r_1 - r_6). Suppose that two pieces of resource type 2, 3 and 5 exist, while all the other resource types have only one representative. The substitution vectors of the resources are as follows:

$$subst(r_1,a_2,\mathbf{P},\mathbf{O}) = [\ 100\quad 150\quad 0\quad 0\quad 0\quad 0\ ], \tag{29a}$$

$$subst(r_2,a_1,\mathbf{P},\mathbf{O}) = \begin{bmatrix} 150 & 100 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{29b}$$

$$subst(r_3,a_5,\mathbf{P},\mathbf{O}) = \begin{bmatrix} 0 & 0 & 100 & 0 & 300 & 0 \end{bmatrix}, \tag{29c}$$

$$subst(r_5,a_3,\mathbf{P},\mathbf{O}) = \begin{bmatrix} 0 & 0 & 120 & 0 & 100 & 0 \end{bmatrix}, \tag{29d}$$

$$subst(r_4,a_1,\mathbf{P},\mathbf{O}) = \begin{bmatrix} 166 & 0 & 0 & 100 & 0 & 0 \end{bmatrix}. \tag{29e}$$

It means that r_1 and r_2 resources may substitute each other, r_3 and r_5 resources are able to substitute each other and r_4 resource is capable to substitute r_1 resource. Other substitutions are not possible. These vectors show that

$$ra(r_2, a_1) = 1, \tag{30}$$
$$ra(r_1, a_2) = 1, \tag{31}$$
$$ra(r_3, a_5) = 1, \tag{32}$$
$$ra(r_5, a_3) = 1 \text{ and} \tag{33}$$
$$ra(r_4, a_1) = 1. \tag{34}$$

Moreover, we suppose that

$$ra(r_i, a_j) = 1, \text{ if } i = j; \tag{35}$$
$$ra(r_2, a_7) = 1 \text{ and} \tag{36}$$
$$ra(r_5, a_8) = 1. \tag{37}$$

This means that a_7 activity can be carried out by r_2 resource, and a_8 activity can be carried out only by r_5 resource. In case of failure, the resources are unavailable as much as shown in *Table 2*.

We suppose that each resource works only in one operation mode and all of the resources have the same error probability. In the example we work only with one product type **(P)**, and we intend to minimize the maximum operation time. Applying the first step of our algorithm, we create the sequence of the resources for each process. This sequence completed with the durations of each activity after user-based modification of the default operation times as illustrated in *Fig.10*.

At this phase we apply the second step of our algorithm and determine the robustness of each processes. In the calculation, we use the following computation method, as we mentioned earlier:

$$t_{\max end}(proc_i) - t_{start}(proc_i) - TDUR \tag{38}$$

The values obtained are summarized in *Table 3*.

Table 3: The robustness values of the example's processes

process	$proc_1$	$proc_2$	$proc_3$	$proc_4$	$proc_5$	$proc_6$
robustness	16	4	2	0	11	8

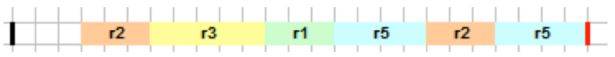

Figure 11: Placement of the second example's basic process onto the time scale ($proc_4$)

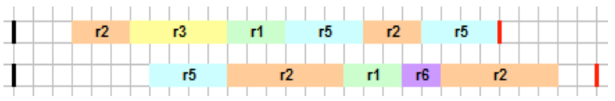

Figure 12: Placement of the second example's second process onto the time scale ($proc_3$)

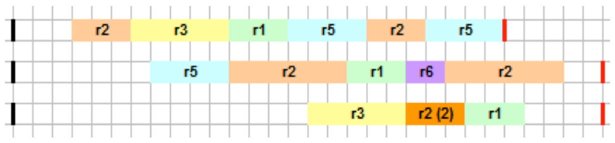

Figure 13: Placement of the 2nd example's third process onto the time scale ($proc_2$)

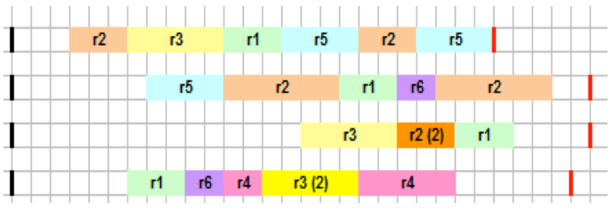

Figure 14: Placement of the second example's fourth process

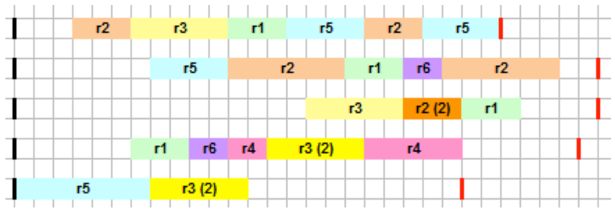

Figure 15: Placement of the second example's fifth process onto the time scale ($proc_5$)

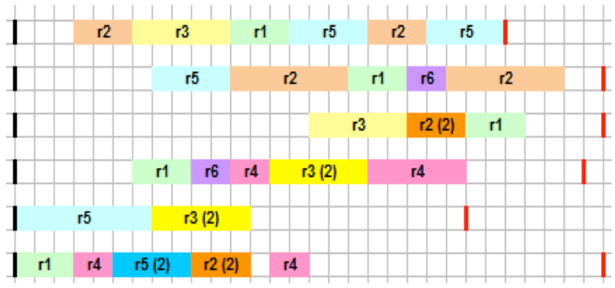

Figure 16: Placement of all processes of the second example onto the time scale ($proc_1$)

Table 3 indicates that the basic process is $proc_4$, which is the least robust process. We place its activities onto the time scale as a chart illustrates this in *Fig.11*. The black line represents the 0 time point and the red one signs the maximal finish time of the process. The second least robust process is $proc_3$. The placement of its activities onto the time scale can be done easily, without any collision as shown in *Fig.12*. The third least robust process is $proc_2$. During its placement there is one collision: its second activity collides with $proc_4$'s fifth activity. However, it doesn't cause any problem, because there are two pieces of resource r_2. The second resource of r_2 resource type can carry out the activity in the originally planned time. The result is shown in *Fig.13*.

Looking for the next least robust process, $proc_6$ is the next. Its fourth activity collides, thus the second piece of r_3 resource type has to be used for carrying it out. Moreover, its final activity also collides. Unfortunately, there is only one piece of r_1 resource type. However, r_2 resource type can substitute an r_1 resource. The problem is that both r_2 resources are occupied at the desired time. There is another resource type, r_4, which is able to substitute an r_1 resource with a bit more necessary time than r_2. As r_4 is free at the desired time, it will take over the task. The time it requires for its task is 1.66 times more than an original r_1-type resource would need for that. *Fig.14* illustrates the results.

The second most robust process is $proc_5$. During the placement of its activities onto the time scale, we find that the second activity collides with $proc_4$'s second activity. Fortunately, there is a second piece of r_3 resource type which can be used freely for the desired time interval. It solves the problem and results a chart as seen in *Fig.15*.

At last, three problematic cases happen during process 1: the collision of the third and fourth activities can be solved by the second piece of the desired resource types; however, in case of the final activity only time shift can solve the collision, because there is only one piece of r_4 resource type and it can not be substituted by any other resource. The result is shown in *Fig.16*.

After placing all activities onto the time scale and solving all collisions, only the fault-tolerance-related improvement need to be done, as the final step of our algorithm. This phase intends to shift the activities of the processes in time, starting from their end by the duration of their preceding activity plus its F value. Because of the maximal finish time constraint, $proc_4$ and $proc_3$ (the first two processes on *Fig.16*) cannot be modified. If we look at *Fig.16*'s third process, its last activity requires $F(r_1)$ = 2 time units plus its normal operation time in case of failure. If we take into account the maximal finish time of the process, its place cannot be modified. This is the case for the fourth process, as well. The fifth process in in *Fig.16* ends with r_3 resource. In case of a failure, it requires 8 time units until it reaches the maximal finish time of the process. Taking this into account, only 3 time units remain,

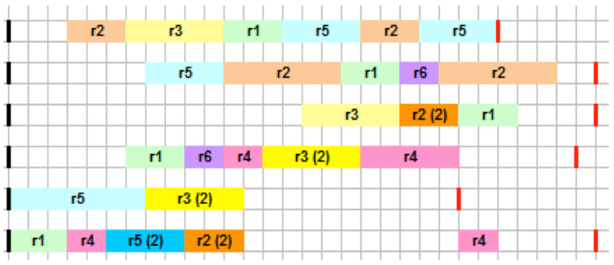

Figure 17: Building fault tolerance into the last process of the second example

which is not enough for the previous activity. That's why we do not modify this process, either. The final activity of the last process, $proc_1$ is carried out by an r_4 resource. In case of a failure, it requires 4 extra time units. In order to avoid collision we shift this activity a bit more forward as shown in *Fig.17*. As $F(r_2) = 1$, and a_2 activity requires 3 time units to be carried out by r_2 resource, $proc_1$'s fourth activity can also be shifted in time. Its previous activity would require 6 time units after itself. Its shift would result 1 extra time unit for previous activity, which is not enough for making it ready for tolerating fault. This hinders building in more fault tolerance. The final result of the algorithm is shown in *Fig.18*, in which we have a schedule, which took into account the possible substitutions, the capabilities of the resources; moreover in some places, it tolerates faults of resources without the need of rescheduling.

Conclusion

We presented a model and algorithm for creating schedules, which tolerate some resource-failures and utilize the substitutability possibilities of the resources. Our goal was to establish a model, which can be the basis of applications in different segments of scheduling problems and makes it possible to generate schedules with the optimization criteria related to the operation time.

We introduced an algorithm, which creates an initial schedule on the basis of the input processes and the substitutability of the resources. This can be improved with preparation for likely failures of resources. The operation of the algorithm was presented on two hypothetical case studies.

Acknowledgements

This research was supported by the European Union and Hungary and co-financed by the European Social Fund through the project TÁMOP-4.2.2.C-11/1/KONV-2012-0004 and TAMOP-4.2.2.A-11/1/KONV-2012-0072, and in part by the Hungarian Research Fund (OTKA) grant number K83440. Support from the National Research Centre for Development and Market Introduction of Advanced Information and Communication Technologies is also acknowledged.

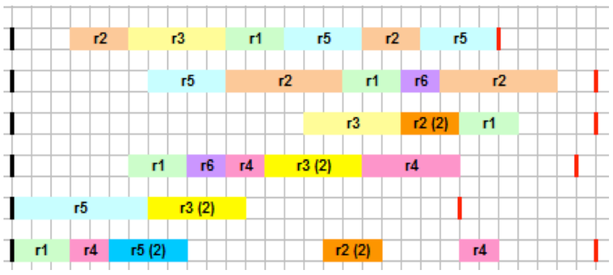

Figure 18: Building fault tolerance into the last process of the second example

REFERENCES

[1] PINEDO M.L.: Scheduling: Theory, Algorithms, and Systems, 4th ed., Springer, New York, USA, *2012*

[2] BRUCKER P.: Scheduling algorithms, 5th ed., Springer, Osnabrück, Germany, *2007*

[3] BRAHIMI B., AUBRUN C., RONDEAU E.: Modelling and Simulation of Scheduling Policies Implemented in Ethernet Switch by Using Coloured Petri Nets, Proc. 11th IEEE Int. Conf. Emerging Technologies and Factory Automation, Prague, Czech Republic, *2006*, 667–674

[4] BARBA I., VALLE C. DEL: A Job-Shop Scheduling Model of Software Development Planning for Constraint-based Local Search, Int. Journal of Software Engineering and Its Applications, *2010*, 4(4), 1–16

[5] XU J., LIU C., ZHAO X., YONGCAREON S.: Business process scheduling with resource availability constraints, Proc. OTM'10, Hersonissos, Crete, Greece, *2010*, 1, 419–427

[6] SULE D.R.: Production Planning and Industrial Scheduling: Examples, Case Studies and Applications, 2nd Ed., CRC Press, Ruston, LA, USA, *2007*

[7] MURTHY S., AKKIRAJU R., RACHLIN J., WU F.: Agent-based cooperative scheduling, Proc. AAAI Workshop on Constraints and Agents, Providence, RI, USA, *1997*, 112–117

[8] DULAI T., WERNER-STARK Á.: Immediate event-aware routing based on cooperative agents, Proc. Factory Automation, Veszprém, Hungary, *2012*, 144–148

[9] PALOMBARINI J., MARTINEZ E.: SmartGantt – An intelligent system for real time rescheduling based on relational reinforcement learning, Expert systems with Applications: An International Journal, *2012*, 38(11), 10251–10268

[10] NANDANWAR J., SHRAWANKAR U.: An adaptive real time task scheduler, IJCSI International Journal of Computer Science Issues, *2012*, 9(6/1), 335–340

[11] GREGG C., BOYER M., HAZELWOOD K., SKADRON K.: Dynamic heterogeneous scheduling decisions using historical runtime data, Proc. 2nd Workshop on Applications for Multi- and Many-Core Processors, San Jose, CA, *2011*