13 Scheduling under Resource Constraints

The scheduling model we consider now is more complicated than the previous ones, because any task, besides processors, may require for its processing some additional scarce resources. Resources, depending on their nature, may be classified into types and categories. The classification into *types* takes into account only the functions resources fulfill: resources of the same type are assumed to fulfill the same functions. The classification into *categories* will concern two points of view. First, we differentiate three categories of resources from the viewpoint of resource constraints. We will call a resource *renewable*, if only its total usage, i.e. temporary availability at every moment, is constrained. A resource is called *non-renewable*, if only its total consumption, i.e. integral availability up to any given moment, is constrained (in other words this resource once used by some task cannot be assigned to any other task). A resource is called *doubly constrained*, if both total usage and total consumption are constrained. Secondly, we distinguish two resource categories from the viewpoint of resource divisibility: *discrete* (i.e. discretely-divisible) and *continuous* (i.e. continuously-divisible) resources. In other words, by a discrete resource we will understand a resource which can be allocated to tasks in discrete amounts from a given finite set of possible allocations, which in particular may consist of one element only. Continuous resources, on the other hand, can be allocated in arbitrary, a priori unknown, amounts from given intervals.

In the next three sections we will consider several basic sub-cases of the resource constrained scheduling problem. In Sections 13.1 and 13.2 problems with renewable, discrete resources will be considered. In Section 13.1 it will in particular be assumed that any task requires one arbitrary processor and some units of additional resources, while in Section 13.2 tasks may require more than one processor at a time (cf. also Chapter 6). Section 13.3 is devoted to an analysis of scheduling with continuous resources.

13.1 Classical Model

The resources to be considered in this section are assumed to be discrete and renewable. Thus, we may assume that s types of additional resources $R_1, R_2, \cdots, R_s$ are available in $m_1, m_2, \cdots, m_s$ units, respectively. Each task T_j requires for its processing one processor and certain fixed amounts of additional resources speci-

J. Blazewicz et al., *Handbook on Scheduling*, International Handbooks on Information Systems, https://doi.org/10.1007/978-3-319-99849-7_13

fied by the resource requirement vector $\boldsymbol{R}(T_j) = [R_1(T_j), R_2(T_j), \cdots, R_s(T_j)]$, where $R_l(T_j)$ $(0 \le R_l(T_j) \le m_l)$, $l = 1, 2, \cdots, s$, denotes the number of units of resource R_l required for the processing of T_j. We will assume here that all required resources are granted to a task before its processing begins or resumes (in the case of preemptive scheduling), and they are returned by the task after its completion or in the case of its preemption. These assumptions define a very simple rule to prevent system deadlocks (see e.g. [CD73]) which is often used in practice, despite the fact that it may lead to a not very efficient use of the resources.

We see that such a model is of special value in manufacturing systems where tasks, besides processors, may require additional limited resources for their processing, such as manpower, tools, space etc. One should also not forget about computer applications, where additional resources can stand for primary memory, mass storage, channels and I/O devices. Before discussing basic results in that area we would like to introduce a missing part of the notation scheme introduced in Section 3.4 that describes additional resources. In fact, they are denoted by parameter $\beta_2 \in \{\varnothing, res\ \lambda\delta\rho\}$, where

$\beta_2 = \varnothing$: no resource constraints,

$\beta_2 = res\ \lambda\delta\rho$: there are specified resource constraints;

$\lambda, \delta, \rho \in \{\cdot, k\}$ denote respectively the number of resource types, resource limits and resource requirements. If

$\lambda, \delta, \rho = \cdot$ then the number of resource types, resource limits and resource requirements are respectively arbitrary, and if

$\lambda, \delta, \rho = k$, then, respectively, the number of resource types is equal to k, each resource is available in the system in the amount of k units and the resource requirements of each task are equal to at most k units.

At this point we would also like to present possible transformations among scheduling problems that differ only by their resource requirements (see Figure 13.1.1). In this figure six basic resource requirements are presented. All but two of these transformations are quite obvious. Transformation $\Pi(res\cdots) \propto \Pi(res1\cdot\cdot)$ has been proved for the case of saturation of machines and additional resources [GJ75] and will not be presented here. The second, $\Pi(res1\cdot\cdot) \propto \Pi(res\cdot 11)$, has been proved in [BBKR86]; to sketch its proof, for a given instance of the first problem we construct a corresponding instance of the second problem by assuming the parameters all the same, except resource constraints. Then for each pair T_i, T_j such that $R_1(T_i) + R_1(T_j) > m_1$ (in the first problem), resource R_{ij} available in the amount of one unit is defined in the second problem. Tasks T_i, T_j require a unit of R_{ij}, while other tasks do not require this resource. It follows that $R_1(T_i) + R_1(T_j) \le m_1$ in the first problem if and only if $R_k(T_i) + R_k(T_j) \le 1$ for each resource R_k in the second problem.

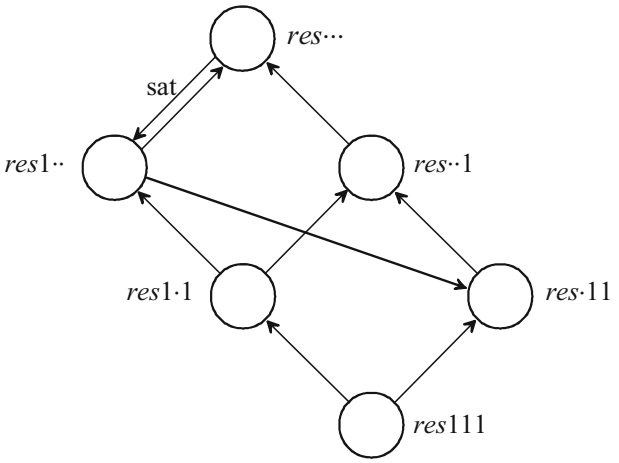

Figure 13.1.1 *Polynomial transformations among resource constrained scheduling problems.*

We will now pass to the presentation of some important results obtained for the above model of resource constrained scheduling. Space limitations prohibit us even from only quoting all these results, however, an extensive survey may be found in [BCSW86, BDM+99, Weg99]. As an example we chose the problem of scheduling tasks on parallel identical processors to minimize schedule length. Basic algorithms in this area will be presented.

Let us first consider the case of independent tasks and non-preemptive scheduling.

Problem $P2 \mid res\cdots, p_j = 1 \mid C_{max}$

The problem of scheduling unit-length tasks on two processors with arbitrary resource constraints and requirements can be solved optimally by the following algorithm.

Algorithm 13.1.1 *Algorithm by Garey and Johnson for* $P2 \mid res\cdots, p_j = 1 \mid C_{max}$ [GJ75].

begin

Construct an n-node (undirected) graph G with each node labeled as a distinct task and with an edge joining T_i to T_j if and only if $R_l(T_i) + R_l(T_j) \le m_l$, $l = 1, 2, \cdots, s$;

Find a maximum matching $\mathcal{F}$ of graph G;

Put the minimal value of schedule length $C^*_{max} = n - |\mathcal{F}|$;

Process in parallel the pairs of tasks joined by the edges comprising set $\mathcal{F}$;
Process other tasks individually;
end;

Notice that the key idea here is the correspondence between maximum matching in a graph displaying resource constraints and the minimum-length schedule. The complexity of the above algorithm clearly depends on the complexity of the algorithm determining the maximum matching. There are several algorithms for finding it, the complexity of the most efficient by Kariv and Even [KE75] being $O(n^{2.5})$. An example of the application of this algorithm is given in Figure 13.1.2 where it is assumed that $n = 6$, $m = 2$, $s = 2$, $m_1 = 3$, $m_2 = 2$, $\boldsymbol{R}(T_1) = [1, 2]$, $\boldsymbol{R}(T_2) = [0, 2]$, $\boldsymbol{R}(T_3) = [2, 0]$, $\boldsymbol{R}(T_4) = [1, 1]$, $\boldsymbol{R}(T_5) = [2, 1]$, and $\boldsymbol{R}(T_6) = [1, 0]$.

An even faster algorithm can be found if we restrict ourselves to the one-resource case. It is not hard to see that in this case an optimal schedule will be produced by ordering tasks in non-increasing order of their resource requirements and assigning tasks in that order to the first free processor on which a given task can be processed because of resource constraints. Thus, problem $P2\,|\,res1\cdot\cdot, p_j = 1\,|\,C_{max}$ can be solved in $O(n\log n)$ time.

If in the last problem tasks are allowed only for 0-1 resource requirements, the problem can be solved in $O(n)$ time even for arbitrary ready times and an arbitrary number of machines, by first assigning tasks with unit resource requirements up to m_1 in each slot, and then filling these slots with tasks having zero resource requirements [Bla78].

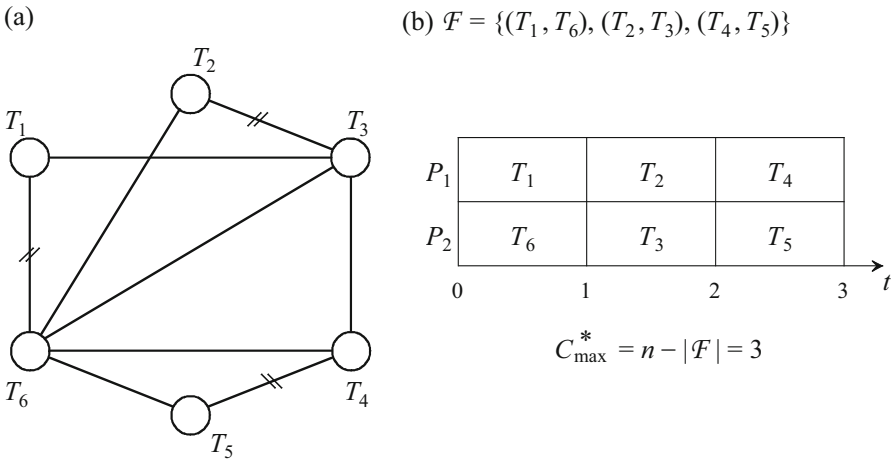

Figure 13.1.2 *An application of Algorithm* 13.1.1:
(a) *graph G corresponding to the scheduling problem,*
(b) *an optimal schedule.*

Problem $P|res\,sor, p_j=1|C_{max}$

When the number of resource types, resource limits and resource requirements are fixed (i.e. constrained by positive integers s, o, r, respectively), problem $P|res\,sor, p_j=1|C_{max}$ is still solvable in linear time, even for an arbitrary number of processors [BE83]. We describe this approach below, since it has a more general application. Depending on the resource requirement vector $[R_1(T_j), R_2(T_j), \cdots, R_s(T_j)] \in \{0,1,\cdots,r\}^s$, the tasks can be distributed among a sufficiently large (and fixed) number of classes. For each possible resource requirement vector we define one such class. The correspondence between the resource requirement vectors and the classes will be described by a 1-1 function f: $\{0,1,\cdots,r\}^s \rightarrow \{1,2,\cdots,k\}$, where k is the number of different possible resource requirement vectors, i.e. $k=(r+1)^s$. For a given instance, let n_i denote the number of tasks belonging to the i^{th} class, $i=1,2,\cdots,k$. Thus all the tasks of class i have the same resource requirement $f^{-1}(i)$. Observe that most of the input information describing an instance of problem $P|res\,sor, p_j=1|C_{max}$ is given by the resource requirements of n given tasks (we bypass for the moment the number m of processors, the number s of additional resources and resource limits o). This input may now be replaced by the vector $\boldsymbol{v}=(v_1,v_2,\cdots,v_k) \in \mathbb{N}_0^k$, where v_i is the number of tasks having resource requirements equal to $f^{-1}(i)$, $i=1,2,\cdots,k$. Of course, the sum of the components of this vector is equal to the number of tasks, i.e. $\sum_{i=1}^{k} v_i = n$.

We now introduce some definitions useful in the following discussion. An *elementary instance* of $P|res\,sor, p_j=1|C_{max}$ is defined as a sequence $\boldsymbol{R}(T_1), \boldsymbol{R}(T_2), \cdots, \boldsymbol{R}(T_u)$, where each $\boldsymbol{R}(T_i) \in \{1,2,\cdots,r\}^s - \{(0,0,\cdots,0)\}$, with properties $u \le m$ and $\sum_{i=1}^{u} \boldsymbol{R}(T_i) \le (o, o, \cdots, o)$. Note that the minimal schedule length of an elementary instance is always equal to 1. An *elementary vector* is a vector $\boldsymbol{v} \in \mathbb{N}_0^k$ which corresponds to an elementary instance. If we calculate the number L of different elementary instances, we see that L cannot be greater than $(o+1)^{(r+1)^s-1}$, however, in practice L will be much smaller than this upper bound. Denote the elementary vectors (in any order) by $b_1, b_2, \cdots, b_L$.

We observe two facts. First, any input $\boldsymbol{R}(T_1), \boldsymbol{R}(T_2), \cdots, \boldsymbol{R}(T_n)$ can be considered as a union of elementary instances. This is because any input consisting of one task is elementary. Second, each schedule is also constructed from elementary instances, since all the tasks which are executed at the same time form an elementary instance.

Now, taking into account the fact that the minimal length of a schedule for any elementary instance is equal to one, we may formulate the original problem

as that of finding a decomposition of a given instance into the minimal number of elementary instances. One may easily see that this is equivalent to finding a decomposition of the vector $\boldsymbol{v} = (v_1, v_2, \cdots, v_k) \in \mathbb{N}_0^k$ into a linear combination of elementary vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_L$, for which the sum of coefficients is minimal: Find $e_1, e_2, \cdots, e_L \in \mathbb{N}_0^k$ such that $\sum_{i=1}^{L} e_i \boldsymbol{b}_i = \boldsymbol{v}$ and $\sum_{i=1}^{L} e_i$ is minimal.

Thus, we have obtained a linear integer programming problem, which in the general case, would be NP-hard. Fortunately, in our case the number of variables L is fixed. It follows that we can apply a result due to Lenstra [Len83] which states that the linear programming problem with fixed number of variables can be solved in polynomial time depending on both, the number of constraints of the integer linear programming problem and log$a$, but not on the number of variables, where a is the maximum of all the coefficients in the linear integer programming problem. Thus, the complexity of the problem is $O(2^{L^2}(k\log a)^{c^L})$, for some constant c. In our case the complexity of that algorithm is $O(2^{L^2}(k\log n)^{c^L}) < O(n)$. Since the time needed to construct the data for this integer programming problem is $O(2^s(L+\log n)) = O(\log n)$, we conclude that the problem $P \mid res\ sor, p_j = 1 \mid C_{max}$ can be solved in linear time.

Problem Pm | res sor | C_{max}

Now we generalize the above considerations for the case of non-unit processing times and tasks belonging to a fixed number k of classes only. That is, the set of tasks may be divided into k classes and all the tasks belonging to the same class have the same processing and resource requirements. If the number of processors m is fixed, then the following algorithm, based on dynamic programming, has been proposed by Błażewicz et al. [BKS89]. A schedule will be built step by step. In every step one task is assigned to a processor at a time. All these assignments obey the following rule: if task T_i is assigned after task T_j, then the starting time of T_i is not earlier than the starting time of T_j. At every moment an assignment of processors and resources to tasks is described by a *state of the assignment process*. For any state a *set of decisions* is given each of which transforms this state into another state. A *value of each decision* will reflect the length of a partial schedule defined by a given state to which this decision led. Below, this method will be described in a more detail.

The state of the assignment process is described by an $m \times k$ matrix $\boldsymbol{X}$, and vectors $\boldsymbol{Y}$ and $\boldsymbol{Z}$. Matrix $\boldsymbol{X}$ reflects numbers of tasks from particular classes already assigned to particular processors. Thus, the maximum number of each entry may be equal to n. Vector $\boldsymbol{Y}$ has k entries, each of which represents the number of tasks from a given class not yet assigned. Finally, vector $\boldsymbol{Z}$ has m entries

and they represent classes which recently assigned tasks (to particular processors) belong to.

The initial state is that for which matrices $\boldsymbol{X}$ and $\boldsymbol{Z}$ have all entries equal to 0 and $\boldsymbol{Y}$ has entries equal to the numbers of tasks in the particular classes in a given instance.

Let S be a state defined by $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$. Then, there is a decision leading to state S' consisting of $\boldsymbol{X}'$, $\boldsymbol{Y}'$ and $\boldsymbol{Z}'$ if and only if

$$\exists\, t \in \{1,\cdots,k\} \text{ such that } Y_t > 0\,, \tag{13.1.1}$$

$$|\mathcal{M}| = 1\,, \tag{13.1.2}$$

where $\mathcal{M}$ is any subset of

$$\mathcal{F} = \{i \mid \sum_{1\le j\le k} X_{ij}p_j = \min_{1\le g\le m}\{\sum_{1\le j\le k} X_{gj}p_j\}\}\,,$$

and finally

$$R_l(T_t) \le m_l - \sum_{1\le j\le k} R_l(T_j)\,|\{g \mid Z_g = j\}|\,, \qquad l = 1,2,\cdots,s\,, \tag{13.1.3}$$

where this new state is defined by the following matrices

$$\begin{aligned}
X'_{ij} &= \begin{cases} X_{ij}+1 & \text{if } i \in \mathcal{M} \text{ and } j = t\,,\\ X_{ij} & \text{otherwise,}\end{cases}\\
Y'_j &= \begin{cases} Y_j - 1 & \text{if } j = t\,,\\ Y_j & \text{otherwise,}\end{cases}\\
Z'_i &= \begin{cases} t & \text{if } i \in \mathcal{M}\,,\\ Z_i & \text{otherwise.}\end{cases}
\end{aligned} \tag{13.1.4}$$

In other words, a task from class t may be assigned to processor P_i, if this class is non-empty (inequality (13.1.1) is fulfilled), there is at least one free processor (equation (13.1.2)), and resource requirements of this task are satisfied (equation (13.1.3)).

If one (or more) conditions (13.1.1) through (13.1.3) are not satisfied, then no task can be assigned at this moment. Thus, one must simulate an assignment of an idle-time task. This is done by assuming the following new state S'':

$$\begin{aligned}
X''_{ij} &= \begin{cases} X_{ij} & \text{if } i \notin \mathcal{F}\,,\\ X_{hj} & \text{otherwise,}\end{cases}\\
\boldsymbol{Y}'' &= \boldsymbol{Y}\,,\\
Z''_i &= \begin{cases} Z_i & \text{if } i \notin \mathcal{F}\,,\\ 0 & \text{otherwise,}\end{cases}
\end{aligned} \tag{13.1.5}$$

where h is one of these g, $1 \le g \le m$, for which

$$\sum_{1\le j\le k} X_{gj}p_j = \min_{\substack{1\le i\le m \\ i\notin \mathcal{F}}} \left\{ \sum_{1\le j\le k} X_{ij}p_j \right\}.$$

This means that the above decision leads to state S'' which repeats a pattern of assignment for processor P_h, i.e. one which will be free as the first from among those which are busy now.

A decision leading from state S to S' has its value equal to

$$\max_{1\le i\le m} \left\{ \sum_{1\le j\le k} X_{ij}p_j \right\}. \tag{13.1.6}$$

This value, of course, is equal to a temporary schedule length.

The final state is that for which the matrices $\boldsymbol{Y}$ and $\boldsymbol{Z}$ have all entries equal to 0. An optimal schedule is then constructed by starting from the final state and moving back, state by state, to the initial state. If there is a number of decisions leading to a given state, then we choose the one having the least value to move back along it. More clearly, if state S follows immediately S', and S (S' respectively) consists of matrices $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$ ($\boldsymbol{X}'$, $\boldsymbol{Y}'$, $\boldsymbol{Z}'$ respectively), then this decision corresponds to assigning a task from $\boldsymbol{Y}-\boldsymbol{Y}'$ at the time $\min_{1\le i\le m} \left\{ \sum_{1\le j\le k} X_{ij}p_j \right\}$.

The time complexity of this algorithm clearly depends on the product of the number of states and the maximum number of decisions which can be taken at the states of the algorithm. A careful analysis shows that this complexity can be bounded by $O(n^{k(m+1)})$, thus, for fixed numbers of task classes k and of processors m, it is polynomial in the number of tasks.

Let us note that another dynamic programming approach has been described in [BKS89] in which the number of processors is not restricted, but a fixed upper bound on task processing times p is specified. In this case the time complexity of the algorithm is $O(n^{k(p+1)})$.

Problem $P|res\cdots,p_j=1|C_{max}$

It follows that when we consider the non-preemptive case of scheduling of unit length tasks we have five polynomial time algorithms and this is probably as much as we can get in this area, since other problems of non-preemptive scheduling under resource constraints have been proved to be NP-hard. Let us mention the parameters that have an influence on the hardness of the problem. First, different ready times cause the strong NP-hardness of the problem even for two processors and very simple resource requirements, i.e. problem $P2|res1\cdot,r_j,p_j=1|C_{max}$ is already strongly NP-hard [BBKR86] (From Figure 13.1.1 we see that problem $P2|res\cdot 11,r_j,p_j=1|C_{max}$ is strongly NP-hard as well). Second, an increase in the number of processors from 2 to 3 results in the strong NP-hardness

of the problem. That is, problem $P3\,|\,res1\cdot\cdot, r_j, p_j = 1\,|\,C_{max}$ is strongly NP-hard as proved by Garey and Johnson [GJ75]. (Note that this is the famous 3-PARTITION problem, the first strongly NP-hard problem.) Again from Figure 13.1.1 we conclude that problem $P3\,|\,res\cdot 11, r_j, p_j = 1\,|\,C_{max}$ is NP-hard in the strong sense. Finally, even the simplest precedence constraints result in the NP-hardness of the scheduling problem, that is, the $P2\,|\,res111, chains, p_j = 1\,|\,C_{max}$ is NP-hard in the strong sense [BLRK83]. Because all these problems are NP-hard, there is a need to work out approximation algorithms. We quote some of the results. Most of the algorithms considered here are list scheduling algorithms which differ from each other by the ordering of tasks on the list. We mention three approximation algorithms analyzed for the problem [1].

1. *First fit* (*FF*). Each task is assigned to the earliest time slot in such a way that no resource and processor limits are violated.

2. *First fit decreasing* (*FFD*). A variant of the first algorithm applied to a list ordered in non-increasing order of $R_{max}(T_j)$, where $R_{max}(T_j) = \max\{R_l(T_j)/m_l \mid 1 \le l \le s\}$.

3. *Iterated lowest fit decreasing* (*ILFD* - applies for $s = 1$ and $p_j = 1$ only). Order tasks as in the *FFD* algorithm. Put C as a lower bound on C^*_{max}. Place T_1 in the first time slot and proceed through the list of tasks, placing T_j in a time slot for which the total resource requirement of tasks already assigned is minimum. If we ever reach a point where T_j cannot be assigned to any of C slots, we halt the iteration, increase C by 1, and start over.

Below we will present the main known bounds for the case $m < n$. In [KSS75] several bounds have been established. Let us start with the problem $P\,|\,res1\cdot\cdot, p_j = 1\,|\,C_{max}$ for which the three above mentioned algorithms have the following bounds:

$$\frac{27}{10} - \left\lceil \frac{37}{10m} \right\rceil < R^{\infty}_{FF} < \frac{27}{10} - \frac{24}{10m},$$

$$R^{\infty}_{FFD} = 2 - \frac{2}{m},$$

$$R_{ILFD} \le 2.$$

We see that the use of an ordered list improves the bound by about 30%. Let us also mention here that problem $P\,|\,res\cdots, p_j = 1\,|\,C_{max}$ can be solved by the approximation algorithm based on the two machine aggregation approach by Röck and

[1] Let us note that the resource constrained scheduling for unit task processing times is equivalent to a variant of the bin packing problem in which the number of items per bin is restricted to m. On the other hand, several other approximation algorithms have been analyzed for the general bin packing problem and the interested reader is referred to [CGJ84] for an excellent survey of the results obtained in this area.

Schmidt [RS83], as described in Section 7.3.2 in the context of flow shop scheduling. The worst case behavior of this algorithm is $R = \lceil \frac{m}{2} \rceil$.

Problem $P|res\cdots|C_{max}$

For arbitrary processing times some other bounds have been established. For problem $P|res\cdots|C_{max}$ the first fit algorithm has been analyzed by Garey and Graham [GG75]:

$$R_{FF}^{\infty} = \min\{\frac{m+1}{2}, s+2-\frac{2s+1}{m}\}.$$

Finally, when dependent tasks are considered, the first fit algorithm has been evaluated for problem $P|res\cdots,prec|C_{max}$ by the same authors:

$$R_{FF}^{\infty} = m.$$

Unfortunately, no results are reported on the probabilistic analysis of approximation algorithms for resource constrained scheduling.

Problem $P|pmtn,res1\cdot1|C_{max}$

Now let us pass to preemptive scheduling. Problem $P|pmtn,res1\cdot1|C_{max}$ can be solved via a modification of McNaughton's rule (Algorithm 5.1.8) by taking

$$C_{max}^{*} = \max\{\max_{j}\{p_j\}, \sum_{j=1}^{n} p_j/m, \sum_{T_j \in \mathcal{Z}_R} p_j/m_1\}$$

as the minimum schedule length, where $\mathcal{Z}_R$ is the set of tasks for which $R_1(T_j) = 1$. The tasks are scheduled as in Algorithm 5.1.8, the tasks from $\mathcal{Z}_R$ being scheduled first. The complexity of the algorithm is obviously $O(n)$.

Problem $P2|pmtn,res\cdots|C_{max}$

Let us consider now the problem $P2|pmtn,res\cdots|C_{max}$. This can be solved via a transformation into the transportation problem [BLRK83].

Without loss of generality we may assume that task $T_j, j = 1,2,\cdots,n$, spends exactly $p_j/2$ time units on each of the two processors. Let $(T_j, T_i), j \neq i$, denote a resource feasible task pair, i.e. a pair for which $R_l(T_j) + R_l(T_i) \leq m_l$, $l = 1,2,\cdots,s$. Let $\mathcal{Z}$ be the set of all resource feasible pairs of tasks. $\mathcal{Z}$ also includes all pairs of the type (T_j, T_{n+1}), $j = 1,2,\cdots,n$, where T_{n+1} is an idle time (dummy) task. Now we may construct a transportation network. Let $n+1$ sender nodes correspond to the $n+1$ tasks (including the idle time task) which are processed on

processor P_1 and let $n+1$ receiver nodes correspond to the $n+1$ tasks processed on processor P_2. Stocks and requirements of nodes corresponding to T_j, $j = 1,2,\cdots,n$, are equal to $p_j/2$, since the amount of time each task spends on each processor is equal to $p_j/2$. The stock and the requirement of two nodes corresponding to T_{n+1} are equal to $\sum_{j=1}^{n} p_j/2$, since these are the maximum amounts of time each processor may be idle. Then, we draw directed arcs (T_j, T_i) and (T_i, T_j) if and only if $(T_j, T_i) \in \mathcal{Z}$, to express the possibility of processing tasks T_j and T_i in parallel on processors P_1 and P_2. In addition we draw an arc (T_{n+1}, T_{n+1}). Then, we assign for each pair $(T_j, T_i) \in \mathcal{Z}$ a cost associated with arcs (T_j, T_i) and (T_i, T_j) equal to 1, and a cost associated with the arc (T_{n+1}, T_{n+1}) equal to 0. (This is because an interval with idle times on both processors does not lengthen the schedule). Now, it is quite clear that the solution of the corresponding transportation problem, i.e. the set of arc flows $\{x^*_{ji}\}$, is simply the set of the numbers of time units during which corresponding pairs of tasks are processed (T_j being processed on P_1 and T_i on P_2).

The complexity of the above algorithm is $O(n^4 \log \sum p_j)$ since this is the complexity of finding a minimum cost flow in a network, with the number of vertices equal to $O(n)$.

Problem $Pm|pmtn, res\cdots|C_{max}$

Now let us pass to the problem $Pm|pmtn, res\cdots|C_{max}$. This problem can still be solved in polynomial time via the linear programming approach (5.1.14) - (5.1.15) but now, instead of the processor feasible set, the notion of a *resource feasible set* is used. By the latter we mean the set of tasks which can be simultaneously processed because of resource limits (including processor limit). At this point let us also mention that problem $P|pmtn, res\cdot\cdot 1|C_{max}$ can be solved by the generalization of the other linear programming approach presented in (5.1.24) - (5.1.27). Let us also add that the latter approach can handle different ready times and the L_{max} criterion. On the other hand, both approaches can be adapted to cover the case of the uniconnected activity network in the same way as that described in Section 5.1.1.

Finally, we mention that for the problem $P|pmtn, res1\cdot\cdot|C_{max}$, the approximation algorithms *FF* and *FFD* had been analyzed by Krause et al. [KSS75]:

$$R^{\infty}_{FF} = 3 - \frac{3}{m},$$

$$R^{\infty}_{FFD} = 3 - \frac{3}{m}.$$

Surprisingly, the use of an ordered list does not improve the bound.

13.2 Scheduling Multiprocessor Tasks

In this section we combine the model presented in Chapter 6 with the resource constrained scheduling. That is, each task is assumed to require one or more processors at a time, and possibly a number of additional resources during its execution. The tasks are scheduled preemptively on m identical processors so that schedule length is minimized.

We are given a set $\mathcal{T}$ of tasks of arbitrary processing times which are to be processed on a set $\mathcal{P} = \{P_1, \cdots, P_m\}$ of m identical processors. There are also s additional types of resources, $R_1, \cdots, R_s$, in the system, available in the amounts of $m_1, \cdots, m_s \in \mathbb{N}$ units. The task set $\mathcal{T}$ is partitioned into subsets,

$$\mathcal{T}^j = \{T_1^j, \cdots, T_{n_j}^j\}, \; j = 1, 2, \cdots, k,$$

k being a fixed integer $\leq m$, denoting a set of tasks each requiring j processors and no additional resources, and

$$\mathcal{T}^{jr} = \{T_1^{jr}, \cdots, T_{n_j^r}^{jr}\}, \; j = 1, 2, \cdots, k,$$

k being a fixed integer $\leq m$, denoting a set of tasks each requiring j processors simultaneously and at most m_l units of resource type R_l, $l = 1, \cdots, s$ (for simplicity we write superscript r to denote "resource tasks", i.e. tasks or sets of tasks requiring resources). The resource requirements of any task T_i^{jr}, $i = 1, 2, \cdots, n_j^r$, $j = 1, 2, \cdots, k$, are given by the vector $\boldsymbol{R}(T_i^{jr}) \leq (m_1, m_2, \cdots, m_s)$.

We will be concerned with preemptive scheduling, i.e. each task may be preempted at any time in a schedule, and restarted later at no cost (in that case, of course, resources are also preempted). All tasks are assumed to be independent, i.e. there are no precedence constraints or mutual exclusion constraints among them. A schedule will be called feasible if, besides the usual conditions each task from $\mathcal{T}^j \cup \mathcal{T}^{jr}$ for $j = 1, 2, \cdots, k$ is processed by j processors at a time, and at each moment the number of processed $\mathcal{T}^{jr}$-tasks is such that the numbers of resources used do not exceed the resource limits. Our objective is to find a feasible schedule of minimum length. Such a schedule will be called *optimal*.

First we present a detailed discussion of the case of one resource type ($s = 1$) available in r units, unit resource requirements, i.e. resource requirement of each task is 0 or 1, and $j \in \{1, k\}$ processors per task for some $k \leq m$. So the task set is assumed to be $\mathcal{T} = \mathcal{T}^1 \cup \mathcal{T}^{1r} \cup \mathcal{T}^k \cup \mathcal{T}^{kr}$. A scheduling algorithm of complexity $O(nm)$ where n is the number of tasks in set $\mathcal{T}$, and a proof of its correctness

are presented for $k = 2$. Finally, a linear programming formulation of the scheduling problem is presented for arbitrary values of s, k, and resource requirements. The complexity of the approach is bounded from above by a polynomial in the input length as long as the number of processors is fixed.

Process of Normalization

First we prove that among minimum length schedules there exists always a schedule in a special normalized form: A feasible schedule of length C for the set $\mathcal{T}^1 \cup \mathcal{T}^{1r} \cup \mathcal{T}^k \cup \mathcal{T}^{kr}$ is called *normalized* if and only if $\exists\, w \in I\!N_0$, $\exists\, L \in [0, C)$ such that the number of $\mathcal{T}^k$-, $\mathcal{T}^{kr}$-tasks executed at time $t \in [0, L)$ is $w + 1$, and the number of $\mathcal{T}^k$-,$\mathcal{T}^{kr}$-tasks executed at time $t \in [\mathrm{L}, C)$ is w (see Figure 13.2.1). We have the following theorem [BE94].

Theorem 13.2.1 *Every feasible schedule for the set of tasks* $\mathcal{T}^1 \cup \mathcal{T}^{1r} \cup \mathcal{T}^k \cup \mathcal{T}^{kr}$ *can be transformed into a normalized schedule.*

Proof. Divide a given schedule into columns such that within each column there is no change in task assignment. Note that since the set of tasks and the number of processors are finite, we may assume that the schedule consists only of a finite number of different columns. Given two columns $\boldsymbol{A}$ and $\boldsymbol{B}$ of the schedule, suppose for the moment that they are of the same length. Let $n_{\boldsymbol{A}}^j$, $n_{\boldsymbol{A}}^{jr}$, $n_{\boldsymbol{B}}^j$, $n_{\boldsymbol{B}}^{jr}$ denote the number of $\mathcal{T}^j$-, $\mathcal{T}^{jr}$-tasks in columns $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, $j \in \{1, k\}$. Let $n_{\boldsymbol{A}}^0$ and $n_{\boldsymbol{B}}^0$ be the numbers of unused processors in $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. The proof is based on the following claim.

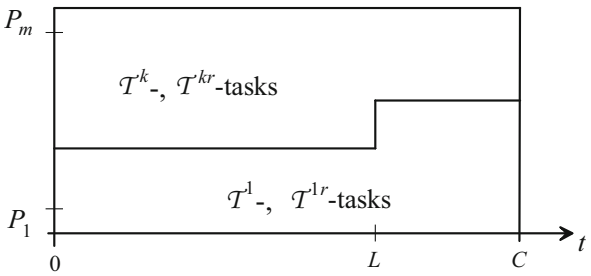

Figure 13.2.1 *A normalized form of a schedule.*

Claim 13.2.2 *Let* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *be columns as above of the same length, and* $n_{\boldsymbol{B}}^k + n_{\boldsymbol{B}}^{kr} \geq n_{\boldsymbol{A}}^k + n_{\boldsymbol{A}}^{kr} + 2$. *Then it is always possible to shift tasks between* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *in such a way that afterwards* $\boldsymbol{B}$ *contains one task of type* $\mathcal{T}^k$ *or* $\mathcal{T}^{kr}$ *less than before.* (The claim is valid for any $k \geq 2$.)

Proof. We consider two different types of task shifts, Σ_1 and Σ_2. They are presented below in an algorithmic way. Algorithm 13.2.3 tries to perform a shift of one $\mathcal{T}^k$-task from $\boldsymbol{B}$ to $\boldsymbol{A}$, and, conversely, of some $\mathcal{T}^1$-and $\mathcal{T}^{1r}$-tasks from $\boldsymbol{A}$ to $\boldsymbol{B}$. Algorithm 13.2.4 tries to perform a shift of some, say $j+1$ $\mathcal{T}^{kr}$-tasks from $\boldsymbol{B}$ to $\boldsymbol{A}$, and, conversely, of j $\mathcal{T}^k$-tasks and some $\mathcal{T}^1$-, $\mathcal{T}^{1r}$-tasks from $\boldsymbol{A}$ to $\boldsymbol{B}$.

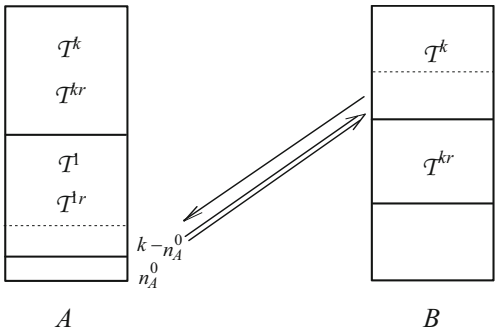

Figure 13.2.2 *Shift of tasks in Algorithm* 13.2.3.

Algorithm 13.2.3 *Shift* Σ_1.

```
begin
if n_B^k > 0        -- i.e. B has at least one task of type T^k
then
  begin
  Shift one task of type T^k from column B to column A;
    -- i.e. remove one of the T^k-tasks from B and assign it to A
  if n_A^0 < k
  then
    begin
    if n_A^1 + n_A^0 ≥ k
    then Shift k - n_A^0 T^1-tasks from A to B
    else
      if There are at least k - n_A^0 - n_A^1 unused resources in B
      then
        begin
        Shift n_A^1 T^1-tasks from A to B;
        Shift k - n_A^0 - n_A^1 T^1r-tasks from A to B;
        end
      else write('Σ1 cannot be applied: resource conflict');
    end;
```

```
  end
else write('Σ1 cannot be applied: B has no T^k-task');
end;
```

Algorithm 13.2.4 *Shift* Σ_2.

```
begin
if n_B^{kr} > 0        -- i.e. B has at least one task of type T^{kr}
then
  begin
  if n_A^{1r} = 0
  then
    begin
    Shift one T^{kr}-task from B to A;
    if n_A^0 < k
      then
      if n_A^{kr} < r          -- i.e. no resource conflicts in A
        then Shift k - n_A^0 T^1-tasks from A to B
        else Write('Σ2 cannot be applied: resource conflict');
    end
  else  -- i.e. in the case of n_A^{1r} > 0
    begin
    if there are numbers j, λ1, and λ2 such that
          λ1 + λ2 = k - n_A^0 if n_A^0 < k, and 1 otherwise,
          0 ≤ j < λ2,
          j ≤ n_A^k, j < n_B^{kr},
          λ1 ≤ n_A^1, λ2 ≤ n_A^{1r},
          n_B^{kr} + λ2 - j - 1 ≤ r,
          n_A^{kr} + n_A^{1r} + j + 1 - λ2 ≤ r
    then  -- perform the following shifts simultaneously
      begin
      Shift j + 1  T^{kr}-tasks from B to A;
      Shift j  T^k-tasks from A to B;
      Shift λ1  T^1-tasks from A to B;
      Shift λ2  T^{1r}-tasks from A to B;
      end
    else write('Σ2 cannot be applied');
    end;
  end
```

else write('Σ_2 cannot be applied: $\boldsymbol{B}$ has no $\mathcal{T}^{kr}$-task');
end;

Before we prove that it is always possible to change columns $\boldsymbol{A}$ and $\boldsymbol{B}$ in the proposed way by means of shifts Σ_1 and Σ_2 we formulate some assumptions and simplifications on the columns $\boldsymbol{A}$ and $\boldsymbol{B}$ (detailed proofs are left to the reader).

(a1) Without loss of generality we assume that all the tasks in $\boldsymbol{A}$ and $\boldsymbol{B}$ are pairwise independent, i.e. they are not parts of the same task.

(a2) $n_{\boldsymbol{A}}^{k} + n_{\boldsymbol{A}}^{kr} \le n_{\boldsymbol{B}}^{k} + n_{\boldsymbol{B}}^{kr} - 2$ (condition of Claim 13.2.2). From that we get

$$n_{\boldsymbol{A}}^{1r} + n_{\boldsymbol{A}}^{1} + n_{\boldsymbol{A}}^{0} \ge n_{\boldsymbol{B}}^{1r} + n_{\boldsymbol{B}}^{1} + n_{\boldsymbol{B}}^{0} + 2k.$$

(a3) We restrict our considerations to the case $n_{\boldsymbol{A}}^{1r} \ge n_{\boldsymbol{B}}^{1r} + k$ because otherwise shift Σ_1 or Σ_2 can be applied without causing resource problems.

(a4) Next we can simplify the considerations to the case $n_{\boldsymbol{B}}^{1r} = 0$. Following (a3) and the fact that, whatever shift we apply, at most k tasks of type $\mathcal{T}^{1r}$ are shifted from $\boldsymbol{A}$ to $\boldsymbol{B}$ (and none from $\boldsymbol{B}$ to $\boldsymbol{A}$) we conclude that we can continue our proof without considering $n_{\boldsymbol{B}}^{1r}$ tasks of type $\mathcal{T}^{1r}$ in both columns.

(a5) Now we assume $n_{\boldsymbol{A}}^{0} = 0$ or $n_{\boldsymbol{B}}^{0} = 0$ as we can remove all the processors not used in both columns.

(a6) Again we can simplify our considerations by assuming $n_{\boldsymbol{B}}^{0} = 0$ and $n_{\boldsymbol{B}}^{1} = 0$. For suppose $n_{\boldsymbol{B}}^{0} > 0$ or $n_{\boldsymbol{B}}^{1} > 0$, we can remove all the idle processors and $\mathcal{T}^{1}$-tasks from column $\boldsymbol{B}$ and $n_{\boldsymbol{B}}^{0} + n_{\boldsymbol{B}}^{1}$ idle processors or tasks of type $\mathcal{T}^{1}$ or $\mathcal{T}^{1r}$ from column $\boldsymbol{A}$. This can be done because there are enough tasks $\mathcal{T}^{1}$ and $\mathcal{T}^{1r}$ (or idle processors) left in column $\boldsymbol{A}$.

The two columns are now of the form shown in Figure 13.2.3.

Now we consider four cases (which exhaust all possible situations) and prove that in each of them either shift Σ_1 or Σ_2 can be applied. Let $\gamma = \min\{n_{\boldsymbol{A}}^{1r}, \max\{k - n_{\boldsymbol{A}}^{0}, 1\}\}$.

Case I: $n_{\boldsymbol{B}}^{kr} + \gamma \le r$, $n_{\boldsymbol{B}}^{k} > 0$. Here Σ_1 can be applied.

Case II: $n_{\boldsymbol{B}}^{kr} + \gamma \le r$, $n_{\boldsymbol{B}}^{k} = 0$. In this case Σ_2 can be applied.

Case III: $n_{\boldsymbol{B}}^{kr} + \gamma > r$, $n_{\boldsymbol{B}}^{k} > 0$.

If $0 \le n_{\boldsymbol{A}}^{1r} \le k - n_{\boldsymbol{A}}^{0}$,
or $n_{\boldsymbol{A}}^{1r} > k - n_{\boldsymbol{A}}^{0}$, $k - n_{\boldsymbol{A}}^{0} \le 0$,
or $n_{\boldsymbol{A}}^{1r} > k - n_{\boldsymbol{A}}^{0} > 0$, $n_{\boldsymbol{A}}^{0} + n_{\boldsymbol{A}}^{1} \ge k$,

or $n_A^{1r} > k - n_A^0 > 0$, $n_A^0 + n_A^1 < k$, $n_B^{kr} + k - n_A^0 - n_A^1 \le r$,

we can always apply Σ_1. In the remaining sub-case,

$$n_A^{1r} > k - n_A^0 > 0,\ n_A^0 + n_A^1 < k,\ n_B^{kr} + k - n_A^0 - n_A^1 > r,$$

Σ_1 cannot be applied and, because of resource limits in column $\boldsymbol{B}$, a Σ_2-shift is possible only under the additional assumption

$$n_B^{kr} + k - n_A^0 - n_A^1 - n_A^k - 1 \le r.$$

What happens in the sub-case $n_B^{kr} + k - n_A^0 - n_A^1 - n_A^k - 1 > r$ will be discussed in a moment.

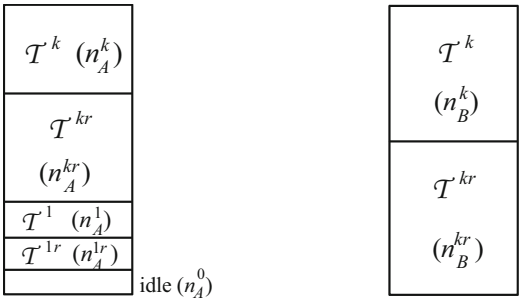

Figure 13.2.3 *Restructuring columns in Claim* 13.2.2.

Case IV: $n_B^{kr} + \gamma > r$, $n_B^k = 0$.

Now, Σ_2 can be applied, except when the following conditions hold simultaneously:

$n_A^{1r} > k - n_A^0 > 0$, $n_A^0 + n_A^1 < k - 1$, and
$n_B^{kr} + k - n_A^0 - n_A^1 - n_A^k - 1 > r$.

We recognize that in cases III and IV under certain conditions neither of the shifts Σ_1, Σ_2 can be applied. These conditions can be put together as follows:

$$n_B^k \ge 0, \text{ and } n_B^{kr} + k - n_A^0 - n_A^1 - n_A^k - 1 > r.$$

We prove that this situation can never occur: From resource limits in column $\boldsymbol{A}$ we get

$$n_B^{kr} + k - n_A^0 - n_A^1 - n_A^k - 1 > r \ge n_A^{1r} + n_A^{kr}.$$

Together with $kn_B^{kr} \le m$ we obtain

$$(k-1)(n_A^1 + n_A^{1r} + n_A^0) - k(k-1) < 0,$$

but from (a2) we know $n_A^1 + n_A^{1r} + n_A^0 \geq 2k$, which contradicts $k > 1$. □

Having proved Claim 13.2.2, it is not hard to prove Theorem 13.2.1. First, we observe that the number of different columns in each feasible schedule is finite. Then, applying shifts Σ_1 or Σ_2 a finite number of times we will get a normalized schedule (for pairs of columns of different lengths only a part of the longer column remains unchanged but for one such column this happens only a finite number of times). □

Before we describe an algorithm which determines a preemptive schedule of minimum length we prove some simple properties of optimal schedules [BE94].

Lemma 13.2.5 *In a normalized schedule it is always possible to process the tasks in such a way that the boundary between $\mathcal{T}^k$-tasks and $\mathcal{T}^{kr}$-tasks contains at most k steps.*

Proof. Suppose there are more than k steps, say $k+i$, $i \geq 1$, and the schedule is of the form given in Figure 13.2.4. Suppose the step at point L lies between the first and the last step of the $\mathcal{T}^k$-, $\mathcal{T}^{kr}$-boundary.

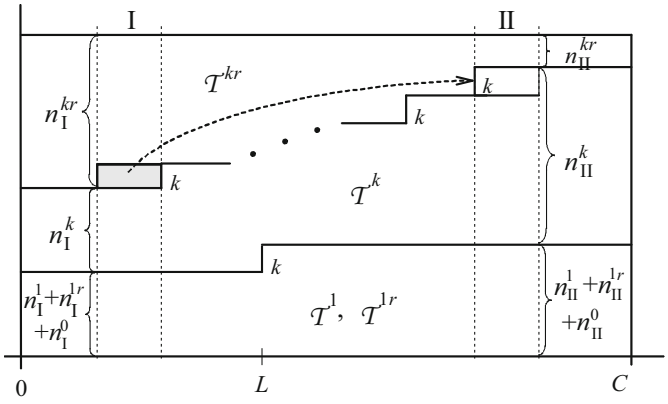

Figure 13.2.4 *k-step boundary between $\mathcal{T}^k$- and $\mathcal{T}^{kr}$-tasks.*

We try to reduce the location of the first step (or even remove this step) by exchanging parts of $\mathcal{T}^{kr}$-tasks from interval I with parts of $\mathcal{T}^k$-tasks from interval II. From resource limits we know:

$$n_{\mathrm{II}}^{1r} + n_{\mathrm{II}}^{kr} \leq r,\ n_{\mathrm{I}}^{1r} + n_{\mathrm{I}}^{kr} \leq r\,.$$

As there are $k+i$ steps, we have $n_{\mathrm{I}}^{kr} = n_{\mathrm{II}}^{kr} + k + i$. Consider possible sub-cases:

(*i*) If $n_{\mathrm{II}}^{1r} + n_{\mathrm{II}}^{kr} < r$, then exchange the $\mathcal{T}^k$- and $\mathcal{T}^{kr}$-tasks in question. This exchange is possible because in I at least one $\mathcal{T}^{kr}$-task can be found that is independent of all the tasks in II, and in II at least one $\mathcal{T}^k$-task can be found that is independent of all the tasks in I.

(*ii*) If $n_{\mathrm{II}}^{1r} + n_{\mathrm{II}}^{kr} = r$, then the shift described in (*i*) cannot be performed directly. However, this shift can be performed simultaneously with replacement of a $\mathcal{T}^{1r}$-task from II by a $\mathcal{T}^1$-task (or idle time) from I, as can be easily seen.

If the step at point L in Figure 13.2.4 is the leftmost or rightmost step among all steps considered so far, then the step removal works in a similar way. □

Corollary 13.2.6 *In case* $k = 2$ *we may assume that the schedule has one of the forms shown in Figure* 13.2.5. □

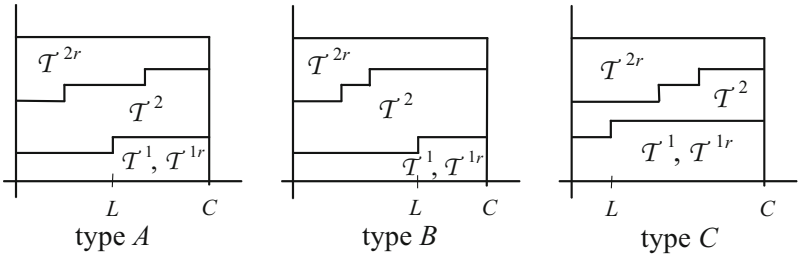

Figure 13.2.5 *Possible schedule types in Corollary* 13.2.6.

Lemma 13.2.7 *Let* $k = 2$. *In cases* (B) *and* (C) *of Figure* 13.2.5 *the schedule can be changed in such a way that one of the steps in the boundary between* $\mathcal{T}^k$ *and* $\mathcal{T}^{kr}$ *is located at point* L, *or it disappears*.

Proof. The same arguments as in the proof of Lemma 13.2.5 are used. □

Corollary 13.2.8 *In case* $k = 2$, *every schedule can be transformed into one of the types given in Figure* 13.2.6. □

Let us note that if in type ***B*1** (Figure 13.2.6) not all resources are used during interval $[L, C)$, then the schedule can be transformed into type ***B*2** or ***C*2**. If in type ***C*1** not all resources are used during interval $[L, C)$, then the schedule can be transformed into type ***B*2** or ***C*2**. A similar argument holds for schedules of type ***A***.

The Algorithm

In this section an algorithm of scheduling preemptable tasks will be presented and its optimality will then be proved for the case $k = 2$. Now, a lower bound for the schedule length can be given. Let

$$X^j = \sum_{T_i \in \mathcal{T}^j} p_i^j, \qquad X^{jr} = \sum_{T_i^{jr} \in \mathcal{T}^{jr}} p_i^{jr}, \quad j = 1, k.$$

It is easy to see that the following formula gives a lower bound on the schedule length,

$$C = \max\{C_{max}^r, C'\} \tag{13.2.1}$$

where

$$C_{max}^r = (X^{1r} + X^{kr})/r,$$

and C' is the optimum schedule length for all $\mathcal{T}^1$-, $\mathcal{T}^{1r}$-, $\mathcal{T}^k$-, $\mathcal{T}^{kr}$-tasks without considering resource limits (cf. Section 6.1).

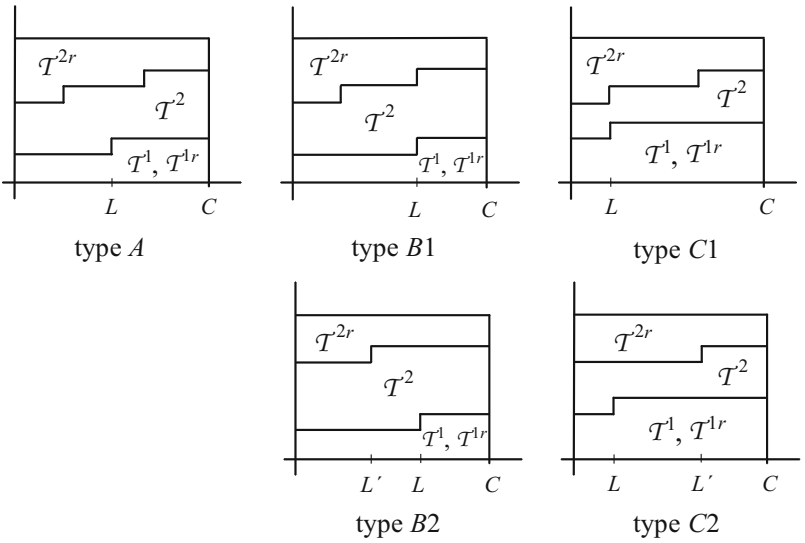

Figure 13.2.6 *Possible schedule types in Corollary* 13.2.8.

In the algorithm presented below we are trying to construct a schedule of type ***B*2** or ***C*2**. However, this may not always be possible because of resource constraints causing "resource overlapping" in certain periods. In this situation we first try to correct the schedule by exchanging some critical tasks so that resource limits are not violated, thus obtaining a schedule of type ***A***, ***B*1** or ***C*1**. If this is not possible, i.e. if no feasible schedule exists, we will have to re-compute bound C in order to remove all resource overlappings.

Let L and L' be the locations of steps as shown in the schedules of type ***B2*** or ***C2*** in Figure 13.2.6. Then

$$L \equiv (X^2 + X^{2r}) \bmod C, \text{ and } L' \equiv X^{2r} \bmod C\,. \tag{13.2.2}$$

In order to compute resource overlapping we proceed as follows. Assign $\mathcal{T}^2$- and $\mathcal{T}^{2r}$-tasks in such a way that only one step in the boundary between these two types of tasks occurs; this is always possible because bound C was chosen properly. The schedule thus obtained is of type ***B2*** or ***C2***. Before the $\mathcal{T}^1$- and $\mathcal{T}^{1r}$-tasks are assigned we partition the free part of the schedule into two areas, *LA* (left area) and *RA* (right area) (cf. Figure 13.2.7). Note that a task from $\mathcal{T}^1 \cup \mathcal{T}^{1r}$ fits into *LA* or *RA* only if its length does not exceed L or $C - L$, respectively. Therefore, all "long" tasks have to be cut into two pieces, and one piece is assigned to *LA*, and the other one to *RA*. We do this by assigning a piece of length $C - L$ of each long task to *RA*, and the remaining piece to *LA* (see Section 5.4 for detailed reasoning). The *excess* $e(T_i)$ of each such task is defined as $e(T_i) = p_i - C + L$, if $p_i > C - L$, and 0 otherwise.

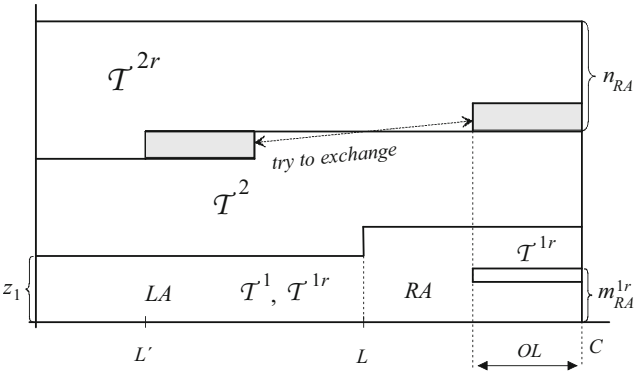

Figure 13.2.7 *Left and right areas in a normalized schedule.*

The task assignment is continued by assigning all excesses to *LA*, and, in addition, by putting as many as possible of the remaining $\mathcal{T}^{1r}$-tasks (so that no resource violations occur) and $\mathcal{T}^1$-tasks to *LA*. However, one should not forget that if there are more long tasks in $\mathcal{T}^1 \cup \mathcal{T}^{1r}$ than $z_1 + 2$ (cf. Figure 13.2.7), then each such task should be assigned according to the ratio of processing capacities of both sides *LA* and *RA*, respectively. All tasks not being assigned yet are assigned to *RA*. Hence only in *RA* resource limits may be violated. Take the sum *OL* of processing times of all $\mathcal{T}^{1r}$-tasks violating the resource limit. *OL* is calculated in the algorithm given below. Of course, *OL* is always less than or equal to $C - L$,

and the $\mathcal{T}^{1r}$-tasks in RA can be arranged in such a way that at any time in $[L, C)$ no more than $r+1$ resources are required.

Resource overlapping (OL) of $\mathcal{T}^{1r}$- and $\mathcal{T}^{2r}$-tasks cannot be removed by exchanging $\mathcal{T}^{1r}$-tasks in RA with $\mathcal{T}^{1}$-tasks in LA, because the latter are only the excesses of long tasks. So the only possibility to remove the resource overlapping is to exchange $\mathcal{T}^{2r}$-tasks in RA with $\mathcal{T}^{2}$-tasks in LA (cf. Figure 13.2.7). Suppose that τ ($\leq OL$) is the maximal amount of $\mathcal{T}^{2}$-, $\mathcal{T}^{2r}$-tasks that can be exchanged in that way. Thus resource overlapping in RA is reduced to the amount $OL - \tau$. If $OL - \tau = 0$, then all tasks are scheduled properly and we are done. If $OL - \tau > 0$, however, a schedule of length C does not exist. In order to remove the remaining resource overlapping (which is in fact $OL - \tau$) we have to increase the schedule length again.

Let n_{RA} be the number of $\mathcal{T}^{2}$- or $\mathcal{T}^{2r}$-tasks executed at the same time during $[L, C)$. Furthermore, let z_1 be the number of processors not used by $\mathcal{T}^{2}$- or $\mathcal{T}^{2r}$-tasks at time 0, let m_{RA}^{1r} be the number of processors executing $\mathcal{T}^{1r}$-tasks in RA (cf. Figure 13.2.7), and let l_{RA}^{1} be the number of $\mathcal{T}^{1}$-tasks executed in RA and having excess in LA. The schedule length is then increased by some amount ΔC, i.e.

$$C = C + \Delta C, \text{ where } \Delta C = \min \{\Delta C_a, \Delta C_b, \Delta C_c\}, \tag{13.2.3}$$

and ΔC_a, ΔC_b, and ΔC_c are determined as follows.

(a) $$\Delta C_a = \frac{OL - \tau}{m_{RA}^{1r} + (m - z_1 - 2)/2 + l_{RA}^{1}}.$$

This formula considers the fact that the parts of $\mathcal{T}^{1r}$-tasks violating resource limits have to be distributed among other processors. By lengthening the schedule the following processors will contribute processing capacity:

- m_{RA}^{1r} processors executing $\mathcal{T}^{1r}$-tasks on the right hand side of the schedule,
- $(m - z_1 - 2)/2$ pairs of processors executing $\mathcal{T}^{2}$- or $\mathcal{T}^{2r}$-tasks and contributing to a decrease of L (and thus lengthening part RA),
- l_{RA}^{1} processors executing $\mathcal{T}^{1}$-tasks whose excesses are processed in LA (and thus decreasing their excesses, and hence allowing part of $\mathcal{T}^{1r}$ to be processed in LA).

(b) If the schedule length is increased by some Δ then L will be decreased by $n_{RA}\Delta$, or, as the schedule type may switch from ***C*2** to ***B*2** (provided L was small enough, cf. Figure 13.2.6), L would be replaced by $C + \Delta + L - n_{RA}\Delta$. In order to avoid the latter case we choose Δ in such a way that the new value of L will be 0, i.e. $\Delta C_b = L / n_{RA}$.

Notice that with the new schedule length $C+\Delta C$, $\Delta C \in \{\Delta C_a, \Delta C_b\}$, the length of the right area RA, will be increased by $\Delta C(n_{RA}+1)$.

(c) Consider all tasks in $\mathcal{T}^1$ with non-zero excesses. All tasks in $\mathcal{T}^1$ whose excesses are less than $\Delta C(n_{RA}+1)$ will have no excess in the new schedule. However, if there are tasks with larger excess, then the structure of a schedule of length $C+\Delta C$ will be completely different and we are not able to conclude that the new schedule will be optimal. Therefore we take the shortest task T_s of $\mathcal{T}^1$ with non-zero excess and choose the new schedule length so that T_s will fit exactly into the new RA, i.e.

$$\Delta C_c = \frac{p_s - C + L}{1 + n_{RA}} .$$

The above reasoning leads to the following algorithm [BE94].

Algorithm 13.2.9

Input: Number m of processors, number r of resource units, sets of tasks $\mathcal{T}^1$, $\mathcal{T}^{1r}$, $\mathcal{T}^2$, $\mathcal{T}^{2r}$.

Output: Schedule for $\mathcal{T}^1 \cup \mathcal{T}^{1r} \cup \mathcal{T}^2 \cup \mathcal{T}^{2r}$ of minimum length.

begin
Compute bound C according to formula (13.2.1);
repeat
Compute L, L' according to (13.2.2), and the excesses for the tasks of $\mathcal{T}^1 \cup \mathcal{T}^{1r}$,
Using bound C, find a normalized schedule for $\mathcal{T}^2$- and $\mathcal{T}^{2r}$-tasks by assigning $\mathcal{T}^{2r}$-tasks from the top of the schedule (processors P_m, $P_{m-1}, \cdots,$) and from left to right, and by assigning $\mathcal{T}^2$-tasks starting at time L, to the processors P_{z_1+1} and P_{z_1+2} from right to left (cf. Figure 13.2.8);
if the number of long $\mathcal{T}^1$- and $\mathcal{T}^{1r}$-tasks is $\leq z_1+2$
then
Take the excesses $e(T)$ of long $\mathcal{T}^1$- and $\mathcal{T}^{1r}$-tasks, and assign them to the left area LA of the schedule in the way depicted in Figure 13.2.8
else
Assign these tasks according to the processing capacities of both sides LA and RA of the schedule, respectively;
if LA is not completely filled
then Assign $\mathcal{T}^{1r}$-tasks to LA as long as resource constraints are not violated;
if LA is not completely filled

then Assign $\mathcal{T}^1$-tasks to LA;
Fill the right area RA with the remaining tasks in the way shown in Figure 13.2.8;
if resource constraints are violated in interval $[L, C)$
then
Compute resource overlapping $OL - \tau$ and correct bound C according to (13.2.3);
until $OL - \tau = 0$;
end;

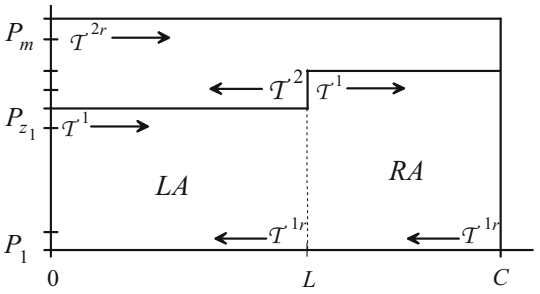

Figure 13.2.8 *Construction of an optimal schedule.*

The optimality of Algorithm 13.2.9 is proved by the following theorem [BE94].

Theorem 13.2.10 *Algorithm* 13.2.9 *determines a preemptive schedule of minimum length for* $\mathcal{T}^1 \cup \mathcal{T}^{1r} \cup \mathcal{T}^2 \cup \mathcal{T}^{2r}$ *in time* $O(nm)$. □

The following example demonstrates the use of Algorithm 13.2.9.

Example 13.2.11 Consider a processor system with $m = 8$ processors, and $r = 3$ units of resource. Let the task set contain 9 tasks, with processing requirements as given in the following table:

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
processing times	10	5	5	5	10	8	2	3	7
number of processors	2	2	2	2	1	1	1	1	1
number of resource units	1	0	0	0	1	1	1	0	0

Table 13.2.1.

Then,

$$X^1 = 10\,,\ X^{1r} = 20\,,\ X^2 = 15\,,\ X^{2r} = 10\,,$$

$$C^r_{max} = (X^{1r} + X^{2r})/r = 10\,,\ C' = (X^1 + X^{1r} + 2X^2 + 2X^{2r})/m = 10\,,$$

i.e. $C = 10$ and $L = 5$. The first loop of Algorithm 13.2.9 yields the schedule shown in Figure 13.2.9. In the schedule thus obtained a resource overlapping occurs in the interval [8,10). There is no way to exchange tasks, so $\tau = 0$, and an overlapping of amount 2 remains. From equation (13.2.3) we obtain $\Delta C_a = 1/3$, $\Delta C_b = 5/2$, and $\Delta C_c = 2/3$. Hence the new schedule length will be $C = 10 + \Delta C_a = 10.33$, and $L = 4.33$, $L' = 10.0$. In the second loop the algorithm determines the schedule shown in Figure 13.2.10, which is now optimal. □

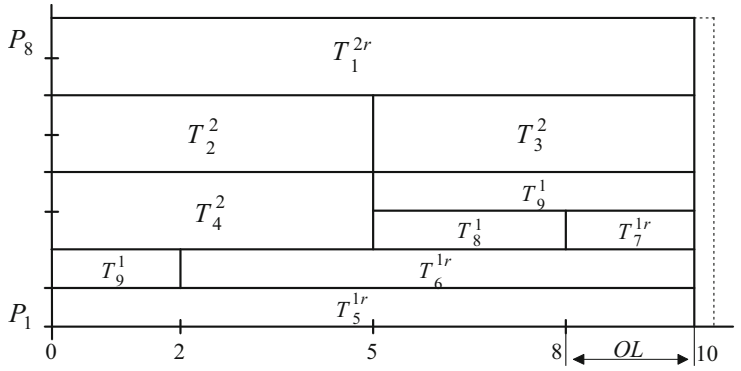

Figure 13.2.9 *Example schedule after the first loop of Algorithm* 13.2.9.

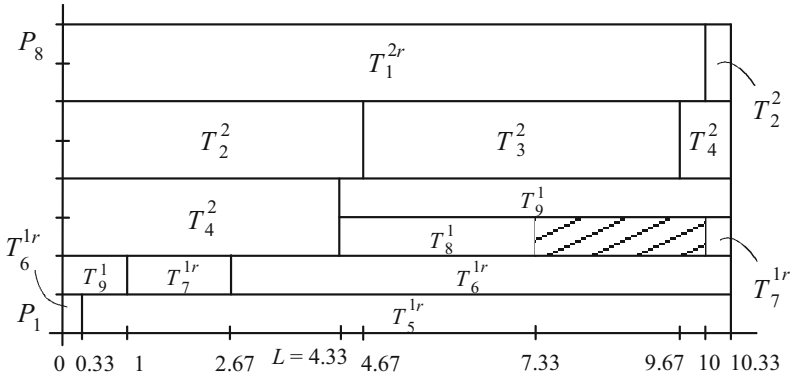

Figure 13.2.10 *Example schedule after the second loop of Algorithm* 13.2.9.

Linear Programming Approach to the General Case

In this section we will show that for a much larger class of scheduling problems one can find schedules of minimum length in polynomial time. We will consider tasks having arbitrary resource and processor requirements. That is, the task set $\mathcal{T}$ is now composed of the following subsets:

$\mathcal{T}^j, j = 1,\cdots,k$, tasks requiring j processors each and no resources, and

$\mathcal{T}^{jr}, j = 1,\cdots,k$, tasks requiring j processors each and some resources.

We present a linear programming formulation of the problem. Our approach is similar to the *LP* formulation of the project scheduling problem, cf. (5.1.15)-(5.1.16). We will need a few definitions. By a resource feasible set we mean here a subset of tasks that can be processed simultaneously because of their total resource and processor requirements. Let M be the number of different resource feasible sets. By variable x_i we denote the processing time of the i^{th} resource feasible set, and by Q_j we denote the set of indices of those resource feasible sets that contain task $T_j \in \mathcal{T}$. Thus the following linear programming problem can be formulated:

$$\begin{aligned} &\textit{Minimize} && \sum_{i=1}^{M} x_i \\ &\textit{subject to} && \sum_{i \in Q_j} x_i = p_j \qquad \text{for each } T_j \in \mathcal{T}, \\ & && x_i \geq 0, \qquad i = 1, 2, \cdots, M. \end{aligned}$$

As a solution of the above problem we get optimal values x_i^* of interval lengths in an optimal schedule. The tasks processed in the intervals are members of the corresponding resource feasible subsets. As before, the number of constraints of the linear programming problem is equal to n, and the number of variables is $O(n^m)$. Thus, for a fixed number of processors the complexity is bounded from above by a polynomial in the number of tasks. On the other hand, a linear programming problem may be solved (using e.g. Karmarkar's algorithm [Kar84]) in time bounded from above by a polynomial in the number of variables, the number of constraints, and the sum of logarithms of all the coefficients in the *LP* problem. Thus for a fixed number of processors, our scheduling problem is solvable in polynomial time.

13.3 Scheduling with Continuous Resources

In this section we consider scheduling problems in which, apart from processors, also continuously-divisible resources are required to process tasks. Basic results will be given for problems with parallel, identical processors (Section 13.3.2) or a single processor (Sections 13.3.3, 13.3.4) and one additional type of continuous, renewable resource. This order of presentation follows from the specificity of task models used in each case.

13.3.1 Introductory Remarks

Let us start with some comments concerning the concept of a continuous resource. As we remember, this is a resource which can be allotted to a task in an arbitrary, unknown in advance amount from a given interval. We will deal with renewable resources, i.e. such for which only usage, i.e. temporary availability is constrained at any time. This "temporary" character is important, since in practice it is often ignored for some doubly constrained resources which are then treated as non-renewable. For example, this is the case of money for which usually only the consumption is considered, whereas they have also a "temporary" nature. Namely, money treated as a renewable resource mean in fact a "flow" of money, called *rate of spending* or *rate of investment*, i.e. an amount available in a given period of a fixed length (week, month). The most typical example of a (renewable) continuous resource is power (electric, hydraulic, pneumatic) which, however, is in general doubly constrained since apart from the usage, also its consumption, i.e. energy, is constrained. Other examples we get when parallel "processors" are driven by a common power source. "Processors" mean here e.g. machines with proper drives, electrolytic tanks, or pumps for refueling navy boats.

We should also stress that sometimes it is purposeful to treat a discrete (i.e. discretely-divisible) resource as a continuous one, since this assumption can simplify scheduling algorithms. Such an approach is allowed when there are many alternative amounts of (discrete) resource available for processing each task. This is, for example, the case in multiprocessor systems where a common primary memory consists of hundreds of pages (see [Weg80]). Treating primary memory as a continuous resource we obtain a scheduling problem from the class we are interested in.

In the next two sections we will study scheduling problems with continuous resources for two models of task processing characteristic (time or speed) vs. (continuous) resource amount allotted. The first model is given in the form of a continuous function: task processing speed vs. resource amount allotted at a given time (Section 13.3.2), whereas the second one is given in the form of a continuous function: task processing time vs. resource amount allotted (Section 13.3.3). The first model is more natural in majority of practical situations, since it reflects directly the "temporary" nature of renewable resources. It is also more general and allows a deep a priori analysis of properties of optimal schedules due to the form of the function describing task processing speed in relation to the allotted amount of resource. This analysis leads even to analytical results in some cases, and in general to the simplest formulations of mathematical programming problems for finding optimal schedules. However, in situations when all tasks use constant resource amounts during their execution, both models are equivalent. Then rather the second model is used as the direct generalization of the traditional, discrete model.

In Section 13.3.4 we will consider another type of problems, where task processing times are constant, but their ready times are functions of a continuous

resource. This is another generalization of the traditional scheduling model which is important in some practical situations.

13.3.2 Processing Speed vs. Resource Amount Model

Assume that we have m identical, parallel processors $P_1, P_2, \cdots, P_m$, and one additional, (continuous, renewable) resource available in amount $\hat{U}$. For its processing task $T_j \in \mathcal{T}$ requires one of the processors and an amount of a continuous resource $u_j(t)$ which is arbitrary and unknown in advance within interval $(0, \hat{U}]$.

The task processing model is given in the form:

$$\dot{x}_j(t) = dx_j(t)/dt = f_j[u_j(t)], \; x_j(0) = 0, \; x_j(C_j) = \tilde{x}_j \tag{13.3.1}$$

where $x_j(t)$ is the state of T_j at time t, f_j is a (positive) continuous, non-decreasing function, $f_j(0) = 0$, C_j is the (unknown in advance) completion time of T_j, and $\tilde{x}_j > 0$ is the known final state, or processing demand, of T_j. Since a continuous resource is assumed to be renewable, we have

$$\sum_{j=1}^{n} u_j(t) \le \hat{U} \text{ for each } t. \tag{13.3.2}$$

As we see, the above model relates task processing speed to the (continuous) resource amount allotted to this task at time t. Let us interpret the concept of a *task state*. By the state of task T_j at time t, $x_j(t)$, we mean a measure of progress of the processing of T_j up to time t or a measure of work related to this processing. This can be, for example, the number of standard instructions of a computer program already processed, the volume of a fuel bunker already refueled, the amount of a product resulting from the performance of T_j up to time t, the number of man-hours or kilowatt-hours already spent in processing T_j, etc.

Let us point out that in practical situations it is often quite easy to construct this model, i.e. to define f_j, $j = 1, 2, \cdots, n$. For example, in computer systems analyzed in [Weg80], the f_j's are progress rate functions of programs, closely related to their lifetime curves, whereas in problems in which processors use electric motors, the f_j's are functions: rotational speed vs. current density.

Let us also notice that in the case of a continuous resource changes of the resource amount allotted to a task within interval $(0, \hat{U}]$ does not mean a task preemption.

To compare formally the model (13.3.1) with the model

$$p_j = \phi_j(u_j), \; u_j \in (0, \hat{U}] \tag{13.3.3}$$

where p_j is the processing time of T_j and ϕ_j is a (positive) continuous, non-increasing function, notice that the condition $x_j(C_j) = \tilde{x}_j$ is equivalent to

$$\int_0^{C_j} f_j[u_j(t)]dt = \tilde{x}_j . \tag{13.3.4}$$

Thus, if $u_j(t) = u_j$, i.e. is constant for $t \in (0,C_j]$, we have

$$C_j = p_j = \tilde{x}_j / f_j(u_j), \text{ i.e. } \phi_j = \tilde{x}_j / f_j(u_j) . \tag{13.3.5}$$

In consequence, if T_j is processed using a constant resource amount u_j, (13.3.5) defines the relation between both models. It is worth to underline that, as we will see, on the basis of the model (13.3.1) one can easily and naturally find the conditions under which tasks are processed using constant resource amounts in an optimal schedule.

Assume now that the number n of tasks is less than or equal to the number m of machines, and that tasks are independent. The first assumption implies that in fact we deal only with the allocation of a continuous resource, since the assignment of tasks to machines is trivial. This is a "pure" (continuous) resource allocation problem, as opposed to a "mixed" (discrete-continuous) problem, when we have to deal simultaneously with scheduling on machines (considered as a discrete resource) and the allocation of a continuous resource.

If $n \le m$ (then it is sufficient to assume $n = m$, since for $n < m$, $m - n$ machines are idle) our goal is to find a piece-wise continuous vector function $\boldsymbol{u}^*(t) = (u_1^*(t), u_2^*(t),\cdots, u_n^*(t))$, $u_j^*(t) \ge 0$, $j = 1, 2,\cdots, n$, such that (13.3.1) and (13.3.2) are satisfied, and $C_{max} = \max\{C_j\}$ reaches its minimum C_{max}^*. This problem was studied in a number of papers (see [Weg82] as a survey) under different assumptions concerning task and resource characteristics. Below we present few basic results useful in our future considerations. To this end we need some additional denotations.

Let us denote by $\mathcal{U}$ the set of *resource allocations*, i.e. all values of a vector function $\boldsymbol{u}(t)$, or all points $\boldsymbol{u} = (u_1, u_2,\cdots, u_n) \in \mathbb{R}^n$, $u_j \ge 0$ for $j = 1, 2,\cdots, n$, satisfying the relation

$$\sum_{j=1}^{n} u_j \le \hat{U} .$$

Further, we will denote by $\mathcal{V}$ the set defined as follows:

$$\begin{aligned} &\boldsymbol{v} = (v_1, v_2,\cdots, v_n) \in \mathcal{V} \text{ if and only if } \boldsymbol{u} \in \mathcal{U}, \\ &\text{and } v_j = f_j(u_j), j = 1, 2,\cdots, n . \end{aligned} \tag{13.3.6}$$

As the functions f_j are monotonic for $j = 1, 2,\cdots, n$, it is obvious that (13.3.5) defines a univalent mapping between $\mathcal{U}$ and $\mathcal{V}$, and thus we can call the points $\boldsymbol{v}$ *transformed resource allocations*. It is easy to prove (see, e.g. [Weg82]) that C_{max}^* as a function of final states of tasks $\tilde{\boldsymbol{x}} = (\tilde{x}_1, \tilde{x}_2,\cdots, \tilde{x}_n)$ can always be expressed as

$$C^*_{max}(\tilde{\boldsymbol{x}}) = \min\{C_{max} > 0 \mid \tilde{\boldsymbol{x}}/C_{max} \in \mathrm{co}\mathcal{V}\} \tag{13.3.7}$$

where $\mathrm{co}\mathcal{V}$ is the convex hull of $\mathcal{V}$, i.e. the set of all convex combinations of the elements of $\mathcal{V}$. Notice that (13.3.7) gives a simple geometrical interpretation of an optimal solution of our problem. Namely, it says that C^*_{max} is always reached at the intersection point of the straight line given by the parametric equations

$$v_j = \tilde{x}_j/C_{max},\ j = 1, 2, \cdots, n \tag{13.3.8}$$

and the boundary of set $\mathrm{co}\mathcal{V}$. Since, according to (13.3.6), the shape of $\mathcal{V}$, and thus $\mathrm{co}\mathcal{V}$, depends on functions $f_j, j = 1, 2, \cdots, n$, we can study the form of optimal solutions in relation to these functions. Let us consider two special, but very important cases:

(*i*) concave $f_j,\ j = 1, 2, \cdots, n$, and

(*ii*) $f_j \le c_j\, u_j,\ c_j = f_j(\hat{U})/\hat{U},\ j = 1, 2, \cdots, n$.

It is easy to check that in case (*i*) set $\mathcal{V}$ is already convex, i.e. $\mathrm{co}\mathcal{V} = \mathcal{V}$. Thus, the intersection point defined above is always a transformed resource allocation (see Figure 13.3.1 for $n = 2$).

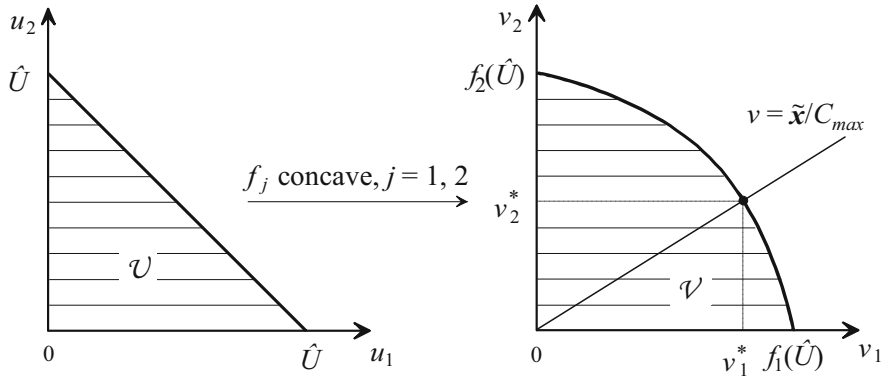

Figure 13.3.1 *The case of concave* $f_j, j = 1, 2$.

This means that in the optimal solution tasks are processed fully in parallel using constant resource amounts $u^*_j, j = 1, 2, \cdots, n$. To find these amounts let us notice that the equation of the boundary of $\mathcal{V}$ has the form $\sum_{j=1}^{n} f_j^{-1}(v_j) = \hat{U}$ (we substitute u_j from (13.3.6) for the equation of the boundary of $\mathcal{U}$, i.e. $\sum_{j=1}^{n} u_j = \hat{U}$), where f_j^{-1} is the function inverse to $f_j, j = 1, 2, \cdots, n$. Substituting v_j from (13.3.8), we get for the above equation

$$\sum_{j=1}^{n} f_j^{-1}(\tilde{x}_j / C_{max}) = \hat{U}. \tag{13.3.9}$$

For given $\tilde{x}_j$, $j = 1, 2, \cdots, n$, the (unique) positive root of this equation is equal to the minimum value C^*_{max} of C_{max}. Of course

$$u_j^* = f_j^{-1}(\tilde{x}_j / C^*_{max}), j = 1, 2, \cdots, n. \tag{13.3.10}$$

It is worth to note that equation (13.3.9) can be solved analytically for some important cases. In particular, this is the case of $f_j = c_j u_j^{1/\alpha_j}$, $c_j > 0$, $\alpha_j \in \{1, 2, 3, 4\}$, $j = 1, 2, \cdots, n$, when (13.3.9) reduces to an algebraic equation of an order ≤ 4. Furthermore, if $\alpha_j = \alpha \geq 1, j = 1, 2, \cdots, n$, we have

$$C^*_{max} = [\frac{1}{\hat{U}} \sum_{j=1}^{n} (\tilde{x}_j / c_j)^{\alpha}]^{1/\alpha}. \tag{13.3.11}$$

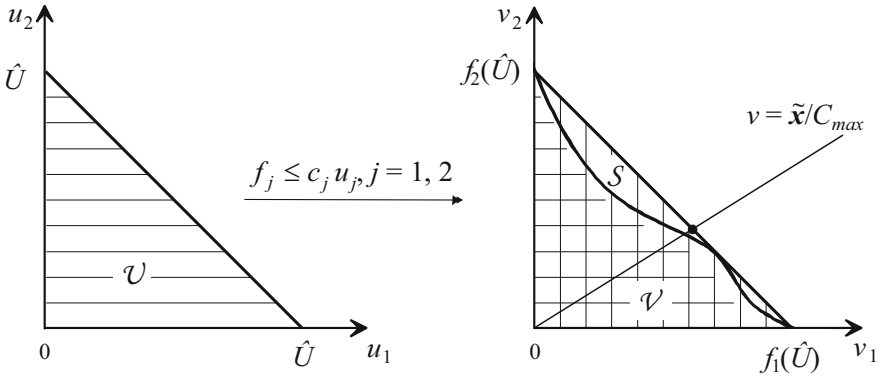

Figure 13.3.2 *The case of* $f_j \leq c_j u_j$, $c_j = f_j(\hat{U})/\hat{U}, j = 1, 2$.

Let us pass to the case (*ii*). It is easy to check that now set $\mathcal{V}$ lies entirely inside simplex $\mathcal{S}$ spanned on the points $(0, \cdots, 0, f_j(\hat{U}), 0, \cdots, 0)$, where $f_j(\hat{U})$ appears on the j^{th} position, $j = 1, 2, \cdots, n$ (see Figure 13.3.2 for $n = 2$). This clearly means that $\text{co}\mathcal{V} = \mathcal{S}$, and that the intersection point of the straight line defined by (13.3.7) and the boundary of $\mathcal{S}$ most probably is not a transformed resource allocation (except for the case of linear f_j, $j = 1, 2, \cdots, n$). However, one can easily verify that the same value C^*_{max} is obtained using transformed resource allocations whose convex combination yields the intersection point just discussed. These always are, of course, the extreme points on which simplex $\mathcal{S}$ is spanned. This fact implies directly that in case (*ii*) there always exists the solution of the length $C^*_{max} = \sum_{j=1}^{n} \tilde{x}_j / f_j(\hat{U})$ in which single tasks are processed consecutively (i.e.

on a single machine) using the maximum resource amount $\hat{U}$. Of course, this solution is not unique if we assume that there is no time loss concerned with a task preemption. However, there is no reason to preempt a task if preemption does not decrease C^*_{max}.

Thus, in both cases, (*i*) and (*ii*), there exist optimal solutions in which each task is processed using a constant resource amount. Consequently, in these cases the model (13.3.1) is mathematically equivalent to the model (13.3.3).

In the general case of arbitrary functions f_j, $j = 1, 2, \cdots, n$, one must search for transformed resource allocations whose convex combination fulfills (13.3.8) and gives the minimum value of C_{max}.

Assume now that tasks are dependent, i.e. that a non-empty relation $\prec$ is defined on $\mathcal{T}$. To represent $\prec$ we will use task-on-arc digraphs, also called activity networks (see Section 3.1). In this representation we can order nodes, i.e. events in such a way that the occurrence of node i is not later than the occurrence of node j if $i < j$. As is well known, such an ordering is always possible (although not always unique) and can be found in time $O(n^2)$ (see, e.g. [Law76]). Using this ordering one can utilize the results obtained for independent tasks to solve corresponding resource allocation problems for dependent tasks. To show how it works we will need some further denotations. Denote by $\mathcal{T}_k$ the subset of tasks which can be processed in the interval between the occurrence of nodes k and $k+1$, by $\tilde{x}_{jk} \geq 0$ a part of $T_j \in \mathcal{T}_k$ (i.e. a part of $\tilde{x}_j$) processed in the above interval, by $\Delta^*_k(\{\tilde{x}_{jk}\}_{T_j \in \mathcal{T}_k})$ the minimum length of this interval as a function of task parts $\{\tilde{x}_{jk}\}_{T_j \in \mathcal{T}_k}$, and by $\mathcal{K}_j$ the set of indices of $\mathcal{T}_k$'s such that $T_j \in \mathcal{T}_k$.

Of course, task parts $\{\tilde{x}_{jk}\}_{T_j \in \mathcal{T}_k}$ are independent for each $k = 1, 2, \cdots, K-1$; K being the total number of nodes in the network, and thus for calculating of Δ_k's as functions of these parts, we can utilize the results obtained for independent tasks. To illustrate this approach let us start with the case (*ii*) discussed previously. Considering the optimal solution in which task parts are processed consecutively in each interval k we see that this is equivalent to the consecutive processing of entire tasks in an order defined by relation $\prec$. Moreover, this result is independent on the ordering of nodes in the network. Unfortunately, the last statement is not true in general for other cases of f_j's.

Consider now the case (*i*) of concave f_j, $j = 1, 2, \cdots, n$, and assume that nodes are ordered in the way defined above. Thus, for calculating $\Delta^*_k(\{\tilde{x}_{jk}\}_{T_j \in \mathcal{T}_k})$, $k = 1, 2, \cdots, K-1$, one must solve for each $\mathcal{T}_k$ an equation of type (13.3.9)

$$\sum_{T_j \in \mathcal{T}_k} f_j^{-1}(\tilde{x}_{jk}/\Delta_k) = \hat{U}. \tag{13.3.12}$$

of which Δ^*_k is the (unique) positive root for given $\{\tilde{x}_{jk}\}_{T_j \in \mathcal{T}_k}$. As already mentioned before, this equation can be solved analytically for some important cases.

The step which remains is to find a division of $\tilde{x}_j$'s into parts $\tilde{x}^*_{jk}$, $j = 1, 2, \cdots, n$; $k \in \mathcal{K}_j$ ensuring the minimum value of C_{max}. This is equivalent to the solution of the following non-linear programming problem:

$$\textit{Minimize} \qquad C_{max} = \sum_{k=1}^{K-1} \Delta^*_k(\{\tilde{x}_{jk}\}_{T_j \in \mathcal{T}_k}) \tag{13.3.13}$$

$$\textit{subject to} \qquad \sum_{k \in \mathcal{K}_j} \tilde{x}_{jk} = \tilde{x}_j, \qquad j = 1, 2, \cdots, n, \tag{13.3.14}$$

$$\tilde{x}_{jk} \geq 0, \qquad j = 1, 2, \cdots, n, \ k \in \mathcal{K}_j. \tag{13.3.15}$$

It can be proved (see e.g. [Weg82]) that C_{max} given by (13.3.13) is a convex function of $\tilde{x}_{jk}$'s for arbitrary f_j's, thus we have a convex programming problem with linear constraints. Its solution is the optimal solution of our problem for the preemptive case and given ordering of nodes. Using the Lagrange theorem one can verify that for $f_j = c_j u_j^{1/\alpha}$, $\alpha > 1$, when C^*_{max} is given by (13.3.11), the solution does not depend on the ordering of nodes. Of course, this is always true when the ordering of nodes is unique, i.e. for a uan (cf. Section 3.1). In general, however, in order to find a solution which is optimal over all possible orderings of nodes one must solve the corresponding convex programming problem for each of these orderings and choose a solution with the smallest value of C_{max}.

To illustrate the way of formulating the optimization problem (13.3.13)-(13.3.15) let us consider a simple example.

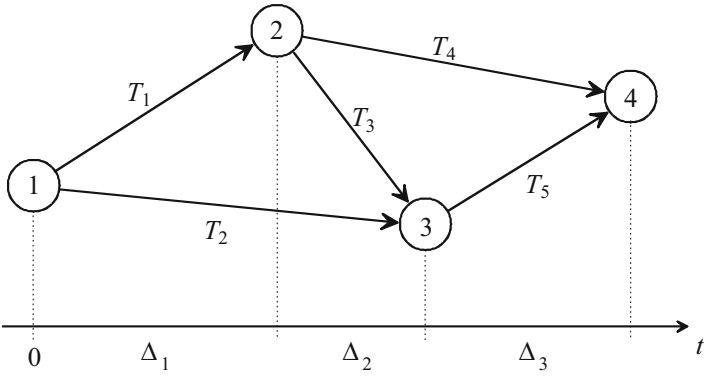

Figure 13.3.3 *Example of a uniconnected activity network.*

Example 13.3.1 Consider the uan given in Figure 13.3.3. Let $\hat{U} = 1$, $f_j = u_j$ for $j = 1, 3, 5$, and $f_j = 2u_j^{1/2}$ for $j = 2, 4$. Subsets of tasks which can be processed between the occurrence of consecutive nodes are:

$$\mathcal{T}_1 = \{T_1, T_2\}, \ \mathcal{T}_2 = \{T_2, T_3, T_4\}, \ \mathcal{T}_3 = \{T_4, T_5\}$$

Sets of indices of $\mathcal{T}_k$'s such that $T_j \in \mathcal{T}_k$ are:

$$\mathcal{K}_1 = \{1\},\ \mathcal{K}_2 = \{1, 2\},\ \mathcal{K}_3 = \{2\},\ \mathcal{K}_4 = \{2, 3\},\ \mathcal{K}_5 = \{3\}\ .$$

Since all the functions f_j are concave, we use equation (13.3.12) to calculate $\Delta_k^*(\{\tilde{x}_{jk}\}_{T_j \in \mathcal{T}_k})$ for $k = 1, 2, 3$. For Δ_1^* we have

$$\tilde{x}_{11}/\Delta_1^* + \tilde{x}_{21}^2/4\Delta_1^{*2} = 1\ ,$$

and thus $\Delta_1^*(\tilde{x}_{11}, \tilde{x}_{21}) = \left(\tilde{x}_{11} + \sqrt{\tilde{x}_{11}^2 + \tilde{x}_{21}^2}\right)/2$. Similarly,

$$\tilde{x}_{22}^2/4\Delta_2^{*2} + \tilde{x}_{32}/\Delta_2^* + \tilde{x}_{42}^2/4\Delta_2^{*2} = 1\ ,$$

$$\Delta_2^*(\tilde{x}_{22}, \tilde{x}_{32}, \tilde{x}_{42}) = \left(\tilde{x}_{32} + \sqrt{\tilde{x}_{22}^2 + \tilde{x}_{32}^2 + \tilde{x}_{42}^2}\right)/2$$

and

$$\tilde{x}_{43}^2/4\Delta_3^{*2} + \tilde{x}_{53}/\Delta_3^* = 1\ ,$$

$$\Delta_3^*(\tilde{x}_{43}, \tilde{x}_{53}) = \left(\tilde{x}_{53} + \sqrt{\tilde{x}_{43}^2 + \tilde{x}_{53}^2}\right)/2\ .$$

The problem is to minimize the sum of the above functions subject to the constraints $\tilde{x}_{11} = \tilde{x}_1$, $\tilde{x}_{21} + \tilde{x}_{22} = \tilde{x}_2$, $\tilde{x}_{32} = \tilde{x}_3$, $\tilde{x}_{42} + \tilde{x}_{43} = \tilde{x}_4$, $\tilde{x}_{53} = \tilde{x}_5$, $\tilde{x}_{jk} \geq 0$ for all j, k. Eliminating five of the variables from the above constraints, a problem with two variables remains. □

Notice that the reasoning performed above for dependent tasks remains valid if we replace the assumption $n \leq m$ by $|\mathcal{T}_k| \leq m$, $k = 1, 2, \cdots, K-1$.

Let us now consider the case that the number of machines is less than the number of tasks which can be processed simultaneously[2]. We start with independent tasks and $n > m$. To solve the problem optimally for the preemptive case we must, in general, consider all possible assignments of machines to tasks, i.e. all m-element combinations of tasks from $\mathcal{T}$. Keeping for them denotation $\mathcal{T}_k$, $k = 1, 2, \cdots, \binom{n}{m}$, we obtain a new optimization problem of type (13.3.13)-(13.3.15).

For the non-preemptive case we consider all maximal sequences of $\mathcal{T}_k$'s such that each task appears in at least one $\mathcal{T}_k$ and all $\mathcal{T}_k$'s containing the same task are consecutively indexed (non-preemptability!). Such sequences will be called *feasible*. It is easy to notice that a feasible sequence consists of $n - m + 1$ elements (i.e. sets $\mathcal{T}_k$). To find an optimal schedule in the general case we have to solve

[2] Recall that this assumption is not needed when in the optimal solution tasks are processed on a single machine, i.e. if $f_j \leq c_j u_j$, $c_j = f_j(\hat{U})/\hat{U}$, $j = 1, 2, ..., n$.

the problem of type (13.3.13)-(13.3.15) for each of the feasible sequences and to choose the best solution.

It is easy to see that finding an optimal schedule is computationally very difficult in general, and thus it is purposeful to construct heuristics. For the non-preemptive case the idea of a heuristic approach can be to choose one or several feasible sequences of m-tuples of tasks described above and solve a problem of type (13.3.13)-(13.3.15) for each of them. These sequences can be chosen in many different ways. A general advise is based on the following reasoning. Assume $n = 5$ and $m = 3$. Then, a feasible sequence consists of $5 - 3 + 1 = 3$ sets $\mathcal{T}_k$ of 3 elements each. Exemplary feasible sequences are: $\mathcal{S}_1 = (\{T_1, T_2, T_3\}, \{T_2, T_3, T_4\}, \{T_3, T_4, T_5\})$, $\mathcal{S}_2 = (\{T_1, T_2, T_3\}, \{T_1, T_2, T_4\}, \{T_1, T_2, T_5\})$.

Define now the *structure* of a sequence as the vector $(|\mathcal{K}_1|, |\mathcal{K}_2|, ..., |\mathcal{K}_n|)$ where $|\mathcal{K}_j|$ is the cardinality of the set of indices of those $\mathcal{T}_k$'s for which $T_j \in \mathcal{T}_k$. It is easy to see that the structure of $\mathcal{S}_1$ is (1, 2, 3, 2, 1), whereas that of $\mathcal{S}_2$ is (3, 3, 1, 1, 1). The basic idea is to study the correspondence between the structure of feasible sequences and the vector of processing demands $\tilde{\boldsymbol{x}}$ of tasks in order to achieve possibly uniform workload for particular machines. If all f_i are concave and identical then we can even identify optimal sequences. For example, under the above assumptions, and $n = 5$, $m = 3$, $\tilde{\boldsymbol{x}} = (10, 20, 30, 20, 10)$, sequence $\mathcal{S}_1$ is optimal, whereas $\mathcal{S}_2$ is optimal for $\tilde{\boldsymbol{x}} = (30, 30, 10, 10, 10)$. This follows from the fact that the division of processing demands of tasks defined as $\tilde{x}_j/|\mathcal{K}_j|$, $j = 1, 2, ..., 5$, corresponds exactly to the uniform workload. Particular algorithms, their worst case behavior and computational results are given in [JW98].

Another idea, for an arbitrary problem type, consists of two steps:

(a) Schedule task from $\mathcal{T}$ on machines from $\mathcal{P}$ for task processing times $p_j = \tilde{x}_j/f_j(\hat{u}_j), j = 1, 2, \cdots, n$, where the $\hat{u}_j$'s are fixed resource amounts.

(b) Allocate the continuous resource among parts of tasks in the schedule obtained in step (a).

Usually in both steps we take into account the same optimization criterion (C_{max} in our case), although heuristics with different criteria can also be considered. Of course, we can solve each step optimally or heuristically. In the majority of cases step (b) can easily be solved (numbers of task parts processed in parallel are less than or equal to m; see Figure 13.3.4 for $m = 2$, $n = 4$) when, as we remember, even analytic results can be obtained for the sets $\mathcal{T}_k$. However, the complexity of step (a) is radically different for preemptive and non-preemptive scheduling. In the first case, the problem under consideration can be solved exactly in $O(n)$ time using McNaughton's algorithm, whereas in the second one it is NP-hard for any fixed value of m ([Kar72]; see also Section 5.1). In the latter case approximation algorithms as described in Section 5.1, or dynamic pro-

gramming algorithms similar to that presented in Section 13.1 can be applied (here tasks are divided into classes with equal processing times).

The question remains how to define resource amounts $\hat{u}_j$, $j = 1, 2, \cdots, n$, in step (a). There are many ways to do this; some of them were described in [BCSW86] and checked experimentally in the preemptive case. Computational experiments show that solutions produced by this heuristic differ from the optimum by several percent on average. However, further investigations in this area are still needed. Notice also that we can change the amounts $\hat{u}$ when performing steps (a) and (b) iteratively.

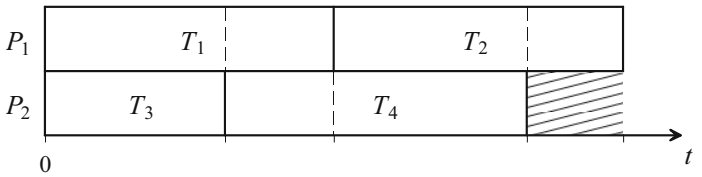

Figure 13.3.4 *Parts of tasks processed in parallel in an example schedule.*

Let us stress once again that the above two-step approach is pretty general, since it combines (discrete) scheduling problems (step (a)) with problems of continuous resource allocation among independent tasks (step (b)). Thus, in step (a) we can utilize all the algorithms presented so far in this book, as well as many others, e.g. from constrained resource project scheduling (see, e.g. [W99]). On the other hand, in step (b) we can utilize several generalizations of the results presented in this section. We will mention some of them below, but first we say few words about dependent tasks and $|\mathcal{T}_k| > m$ for at least one k. In this case one has to combine the reasoning presented for dependent tasks and $n \le m$, and that for independent tasks and $n > m$. This means, in particular, that in order to solve the preemptive case, each problem of type (13.3.13)-(13.3.15) must be solved for all m-elementary subsets of sets $\mathcal{T}_k$, $k = 1, 2, \cdots, K-1$.

We end this section with few remarks concerning generalizations of the results presented for continuous resource allocation problems. First of all we can deal with a doubly constrained resource, when, apart from (13.3.2), also the constraint $\sum_{j=1}^{n} \int_0^{C_j} f_j[u_j(t)]dt \le \hat{V}$ is imposed, $\hat{V}$ being the consumption constraint [Weg81]. Second, each task may require many continuous resource types. The processing speed of task T_j is then given by $\dot{x}_j(t) = f_j[u_{j1}(t), u_{j2}(t), \cdots, u_{js}(t)]$, where $u_{jl}(t)$ is the amount of resource R_l allotted to T_j at time t, and s is the number of different resource types. Thus in general we obtain multi-objective resource allocation problems of the type formulated in [Weg91]. Third, other optimality criteria can be considered, such as $\int_0^{C_{max}} g[\boldsymbol{u}(t)]dt$ [NZ81], $\Sigma w_j C_j$ [NZ84a, NZ84b] or

L_{max} [Weg89]. Finally, sequences of sets of dependent tasks can be studied [JS88].

Let us also mention about an application to Grid Scheduling [MWW04]. An extensive survey pf the results concerning scheduling under resource constraints can be found in [WJMW11].

13.3.3 Processing Time vs. Resource Amount Model

In this section we consider problems of scheduling non-preemptable tasks on a single machine, where task processing times are linear, decreasing and continuous functions of a continuous resource. The task processing model is given in the form

$$p_j = b_j - a_j u_j,\ \underset{\sim}{u}_j \le u_j \le \tilde{u}_j,\ j = 1, 2, \cdots, n \tag{13.3.16}$$

where $a_j > 0$, $b_j > 0$, and $\underset{\sim}{u}_j$ and $\tilde{u}_j \in [0, b_j/a_j]$ are known constants. The continuous resource is available in maximal amount $\hat{U}$, i.e. $\sum_{j=1}^{n} u_j \le \hat{U}$. Although now the resource is not necessarily renewable (this is not a temporary model), we will keep denotations as introduced in Section 13.3.2. Scheduling problems using the above model were broadly studied by Janiak in a number of papers we will refer to in the sequel. Without loss of generality we can restrict our considerations to the case that lower bounds $\underset{\sim}{u}_j$ of resource amounts allotted to the particular tasks are zero. This follows from the fact that in case of $\underset{\sim}{u}_j > 0$ the model can be replaced by an equivalent one in the following way: replace b_j by $b_j - a_j \underset{\sim}{u}_j$ and $\tilde{u}_j$ by $\tilde{u}_j - \underset{\sim}{u}_j$, $j = 1, 2, \cdots, n$, and $\hat{U}$ by $\hat{U} - \sum_{i=1}^{n} \underset{\sim}{u}_i$, finally, set all $\underset{\sim}{u}_j = 0$. Given a set of tasks $\mathcal{T} = \{T_1, \cdots, T_n\}$, let $\boldsymbol{z} = [z(1), \cdots, z(n)]$ denote a permutation of task indices that defines a feasible task order for the scheduling problem, and let $\mathcal{Z}$ be the set of all such permutations (partial or complete ones). A schedule for $\mathcal{T}$ can then be characterized by a pair $(\boldsymbol{z}, \boldsymbol{u}) \in \mathcal{Z} \times \mathcal{U}$. The value of a schedule $(\boldsymbol{z}, \boldsymbol{u})$ with respect to the optimality criterion γ will be denoted by $\gamma(\boldsymbol{z}, \boldsymbol{u})$. A schedule with an optimal value of γ will briefly be denoted by $(\boldsymbol{z}^*, \boldsymbol{u}^*)$.

Let us start with the problem of minimizing C_{max} for the case of equal ready times and arbitrary precedence constraints [Jan88a]. Using a slight modification of the notation introduced in Section 3.4, we denote this type of problems by $1 \mid prec, p_j = b_j - a_j u_j, \Sigma u_j \le \hat{U} \mid C_{max}$. It is easy to verify that an optimal solution $(\boldsymbol{z}^*, \boldsymbol{u}^*)$ of the problem is obtained if we chose an arbitrary permutation $\boldsymbol{z} \in \mathcal{Z}$ and allocate the continuous resource according to the following algorithm.

Algorithm 13.3.2 *for finding* $\boldsymbol{u}^*$ *for* $1\,|\,prec, p_j = b_j - a_j u_j, \Sigma u_j \le \hat{U}\,|\,C_{max}$ [Jan88a].

begin
for $j := 1$ **to** n **do** $u_j^* := 0$;
while $\mathcal{T} \neq \emptyset$ **and** $\hat{U} > 0$ **do**
  begin
  Find $T_k \in \mathcal{T}$ for which $a_k = \max_j \{a_j\}$;
  $u_k^* := \min\{\tilde{u}_k, \hat{U}\}$;
  $\hat{U} := \hat{U} - u_k^*$;
  $\mathcal{T} := \mathcal{T} - \{T_k\}$;
  end;
$\boldsymbol{u}^* := [u_1^*, \cdots, u_n^*]$; -- $\boldsymbol{u}^*$ is an optimal resource allocation
end;

Obviously, the time complexity of this algorithm is $O(n \log n)$.

Consider now the problem with arbitrary ready times, i.e. $1\,|\,prec, r_j, p_j = b_j - a_j u_j, \Sigma u_j \le \hat{U}\,|\,C_{max}$. One can easily prove that an optimal solution $(\boldsymbol{z}^*, \boldsymbol{u}^*)$ of the problem is always found if we first schedule tasks according to an obvious modification of Algorithm 4.5.2 by Lawler - thus having determined $\boldsymbol{z}^*$ - and then allocate the resources according to Algorithm 13.3.3.

Algorithm 13.3.3 *for finding* $\boldsymbol{u}^*$ *for* $1\,|\,prec, r_j, p_j = b_j - a_j u_j, \Sigma u_j \le \hat{U}\,|\,C_{max}$ [Jan88a].

begin
for $j := 1$ **to** n **do** $u_j^* := 0$;
$S_{z^*(1)} := r_{z^*(1)}$;
$l := 1$;
for $j := 2$ **to** n **do** $S_{z^*(j)} := \max\{r_{z^*(j)}, S_{z^*(j-1)} + b_{z^*(j-1)}\}$;
  -- starting times of tasks in permutation $\boldsymbol{z}^*$ for $\boldsymbol{u}^*$ have been calculated
$\mathcal{J} := \{\boldsymbol{z}^*\}$; -- construct set $\mathcal{J}$
while $\mathcal{J} \neq \emptyset$ **and** $\hat{U} \neq 0$ **do**
  begin
  Find the biggest index k, $l \le k \le n$, for which $r_{z^*(k)} = S_{z^*(k)}$;
  $\mathcal{J} := \{z^*(j) \mid k \le j \le n$, and $u_{z^*(j)}^* < \tilde{u}_{z^*(j)}\}$;
  Find index t for which $a_{z^*(t)} = \max\{a_{z^*(j)} \mid \boldsymbol{z}^*(j) \in \mathcal{J}\}$;
  $d := \min\{S_{z^*(i)} - r_{z^*(i)} \mid t < i \le n\}$;
  $y := \min\{\tilde{u}_{z^*(t)}, \hat{U}, d/a_{z^*(t)}\}$;
  $u_{z^*(t)}^* := u_{z^*(t)}^* + y$;
  $\hat{U} := \hat{U} - y$;
  for $i := t$ **to** n **do** $S_{z^*(i)} := S_{z^*(i)} - y a_{z^*(t)}$;

$l := k$;
-- new resource allocation and task starting times have been calculated
end;
$\boldsymbol{u}^* := [u_1^*, \cdots, u_n^*]$; -- $\boldsymbol{u}^*$ is an optimal resource allocation
end;

The complexity of this algorithm is $O(n^2)$, and this is the complexity of the whole approach for finding $(\boldsymbol{z}^*, \boldsymbol{u}^*)$, since Algorithm 4.5.2 is also of complexity $O(n^2)$.

Let us now pass to the problems of minimizing maximum lateness L_{max}. Since problem $1 \mid prec,\ p_j = b_j - a_j u_j,\ \Sigma u_j \le \hat{U} \mid L_{max}$ is equivalent to problem $1 \mid prec,\ r_j,\ p_j = b_j - a_j u_j,\ \Sigma u_j \le \hat{U} \mid C_{max}$ (as in the case without additional resources), its optimal solution can always be obtained by finding $\boldsymbol{z}^*$ according to the Algorithm 4.5.2 and $\boldsymbol{u}^*$ according to a simple modification of Algorithm 13.3.3.

It is also easy to see that problem $1 \mid r_j,\ p_j = b_j - a_j u_j,\ \Sigma u_j \le \hat{U} \mid L_{max}$ is strongly NP-hard, since the restricted version $1 \mid r_j \mid L_{max}$ is already strongly NP-hard (see Section 4.3). For the problem $1 \mid prec,\ r_j,\ p_j = b_j - a_j u_j,\ \Sigma u_j \le \hat{U} \mid L_{max}$ where in addition precedence constraints are given, an exact branch and bound algorithm was presented by Janiak [Jan86c].

Finally, consider problems with the optimality criteria ΣC_j and $\Sigma w_j C_j$. Problem $1 \mid prec,\ p_j = b_j - a_j u_j,\ \Sigma u_j \le \hat{U} \mid \Sigma C_j$ is NP-hard, and problem $1 \mid r_j,\ p_j = b_j - a_j u_j,\ \Sigma u_j \le \hat{U} \mid \Sigma C_j$ is strongly NP-hard, since the corresponding restricted versions $1 \mid prec \mid \Sigma C_j$ and $1 \mid r_j \mid \Sigma C_j$ are NP-hard and strongly NP-hard, respectively (see Section 4.2). The complexity status of problem $1 \mid p_j = b_j - a_j u_j,\ \Sigma u_j \le \hat{U} \mid \Sigma w_j C_j$ is still an open question. It is easy to verify for any given $\boldsymbol{z} \in \mathcal{Z}$ the minimum value of $\Sigma w_j C_j$ in this problem is always obtained by allocating the resource according to the following algorithm of complexity $O(n \log n)$.

Algorithm 13.3.4 *for finding* $\boldsymbol{u}^*$ for $1 \mid p_j = b_j - a_j u_j,\ \Sigma u_j \le \hat{U} \mid \Sigma w_j C_j$ [Jan88a].
begin
$\mathcal{J} := \{ z \}$; -- construct set $\mathcal{J}$
while $\mathcal{J} \neq \emptyset$ **do**
begin
Find $z(k) \in \mathcal{J}$ for which $a_{z(k)} \sum_{j=k}^{n} w_{z(j)} = \max_{z(i) \in \mathcal{J}} \{ a_{z(i)} \sum_{j=i}^{n} w_{z(j)} \}$;
$u_{z(k)}^* := \min \{ \tilde{u}_{z(k)}, \max \{0, \hat{U}\} \}$;
$\hat{U} := \hat{U} - u_{z(k)}^*$;
$\mathcal{J} := \mathcal{J} - \{ z(k) \}$;
end;

$\boldsymbol{u}^* := [u_1^*, \cdots, u_n^*]$; -- $\boldsymbol{u}^*$ is an optimal resource allocation
end;

An exact algorithm of the same complexity can also be given for this problem if for any two tasks T_i, T_j either $T_i \prec T_j$ or $T_j \prec T_i$, where $T_i \prec T_j$ means that $b_i \le b_j$, $a_i \ge a_j$, $\tilde{u}_i \ge \tilde{u}_j$, and $w_i \ge w_j$. In this case the optimal permutation $\boldsymbol{z}^*$ is obtained by ordering the jobs according to $\prec$, and the algorithm of the optimal resource allocation is as follows: $u_{z^*(j)}^* = \min\{\tilde{u}_{z^*(j)}, \max\{0, \hat{U}_j\}\}$ for $j = 1, 2, \cdots, n$, where $\hat{U}_1 = \hat{U}$, $\hat{U}_{j+1} = \hat{U}_j - u_{z^*(j)}^*, j = 1, 2, \cdots, n-1$.

Now let us pass to the criterion which is specific to scheduling problems with additional continuous resources, namely to the criterion denoting the total resource utilization, i.e. $U = \sum_{j=1}^{n} u_j$. This criterion should be minimized subject to the constraint $\gamma < \hat{\gamma}$ where γ is a classical schedule performance measure and $\hat{\gamma}$ is a given value of γ. Of course, scheduling problems of minimizing Σu_j are closely related to corresponding problems with criterion γ. Additionally, we use the fact that for the considered problems it is easy to calculate the maximum value $\tilde{\gamma}$ of γ.

We illustrate this idea for the criterion $\gamma = C_{max}$, i.e. for problem $1 \mid prec, p_j = b_j - a_j u_j, C_{max} < \hat{C} \mid \Sigma u_j$. It is obvious that the upper bound for C_{max}, $\tilde{C}_{max} = \min_{z \in Z} \{C_{max}(z, 0)\} = C_{max}(\boldsymbol{z}^*, 0)$. Thus, we have the following modification of Algorithm 13.3.2.

Algorithm 13.3.5 *for finding* $\boldsymbol{u}^*$ *for* $1 \mid prec, p_j = b_j - a_j u_j, C_{max} \le \hat{C} \mid \Sigma u_j$ [Jan91a].
begin
for $j := 1$ **to** n **do** $u_j^* := 0$;
$U := 0$;
$C_{max} := \tilde{C}_{max}$;
while $\mathcal{T} \neq \emptyset$ **and** $C_{max} > \hat{C}$ **do**
  begin
  Find $T_k \in \mathcal{T}$ for which $a_k = \max_j \{a_j\}$;
  $u_k^* := \min\{\tilde{u}_k, \max\{0, (C_{max} - \hat{C})/a_k\}\}$;
  $U := U + u_k^*$;
  $C_{max} := C_{max} - a_k u_k^*$;
  $\mathcal{T} := \mathcal{T} - \{T_k\}$;
  end;
if $\mathcal{T} = \emptyset$ **and** $C_{max} > \hat{C}$
then no solution exists
else $\boldsymbol{u}^* := [u_1^*, \cdots, u_n^*]$; -- u^* is an optimal resource allocation
end;

Knowing how to solve a problem for criteria γ and Σu_j, one can also find the set of all Pareto-optimal (i.e. efficient or non-dominated) solutions $(\boldsymbol{z}^{\mathbf{P}}, \boldsymbol{u}^{\mathbf{P}})$ for *bi-criterion problems* ([Jan91a]). As an example, consider the problem $1\,|\,prec, p_j = b_j - a_j u_j\,|\,C_{max} \wedge \Sigma u_j$. Of course, $\underset{\sim}{C}_{max} = \min\limits_{\boldsymbol{z} \in \mathcal{Z}} \{C_{max}(\boldsymbol{z}, \tilde{\boldsymbol{u}})\} = \sum\limits_{j=1}^{n} (b_j - a_j \tilde{u}_j)$ is a lower bound for C_{max}. In our problem, for each value $C_{max} \in [\underset{\sim}{C}_{max}, \tilde{C}_{max}]$, any feasible permutation $\boldsymbol{z} \in \mathcal{Z}$ can be taken as Pareto-optimal permutation $\boldsymbol{z}^{\mathbf{P}}$. In order to find the set $\mathcal{U}^{\mathbf{P}}$ of all Pareto-optimal resource allocations $\boldsymbol{u}^{\mathbf{P}}$, we determine the Pareto curve (which is a convex, decreasing and piece-wise linear function) from the following algorithm of time complexity $O(n\log n)$.

Algorithm 13.3.6 *for finding the Pareto curve in* $1\,|\,prec, p_j = b_j - a_j u_j\,|\,C_{max} \wedge \Sigma u_j$ [Jan91a].

begin
for $j := 1$ **to** n **do** $u^*_{z(j)} := 0$;
$i := 0$;
$C^0_{max} := \tilde{C}_{max}$;
$U^0 := 0$;
while $\mathcal{T} \neq \emptyset$ **do**
  begin
  $i := i + 1$;
  Find $T_k \in \mathcal{T}$ for which $a_k = \max\limits_j \{a_j\}$;
  $u^*_k := \tilde{u}_k$;
  $C^i_{max} := C^{i-1}_{max} - a_k \tilde{u}_k$;
  $U^i := U^{i-1} + \tilde{u}_k$;
  $a^i := 1/a_k$;
  $\mathcal{T} := \mathcal{T} - \{T_k\}$;
  for $l := 1$ **to** n **do** $u^i_l := u^*_l$;
  end;
end;

Obtained pairs (C^0_{max}, U^0), $(C^1_{max}, U^1), \ldots, (C^n_{max}, U^n)$ are consecutive break-points of the Pareto curve; a^i is the slope of the i^{th} segment of this curve, $i = 1, 2, \ldots, n$. The set $\mathcal{U}^{\mathbf{P}}$ is the sum of n segments joining the points $\boldsymbol{u}^i$, $\boldsymbol{u}^{i+1}$, $i = 0, 1, 2, \ldots, n-1$, where $\boldsymbol{u}^0 = 0$.

In [JK96] the problem was considered with given deadlines $\tilde{d}_j$ and minimization of the total weighted resource consumption, i.e. the problem $1\,|\,p_j = b_j - a_j$

u_j, $C_j \le \tilde{d}_j \mid \sum w_j u_j$. This problem is solvable in $O(n \log n)$ time for a continuously-divisible resource and is NP-hard for a discrete resource. A fully polynomial approximation scheme is presented for the last case.

The paper [CJK98] is devoted to the following machine scheduling problems with linear models of task processing times and with a discrete resource: $1 \mid p_j = b_j - a_j u_j, F_1 \le K \mid F_2$, $1 \mid p_j = b_j - a_j u_j, F_2 \le K \mid F_1$ and $1 \mid p_j = b_j - a_j u_j \mid F_1 \wedge F_2$, where F_1 and F_2 is a criterion of resource and completion time type, respectively. More precisely, $F_1 \in \{g_{max}, \sum u_j, \sum w_j u_j\}$ and $F_2 \in \{C_{max}, c_{max}, \sum U_j, \sum w_j U_j, \sum C_j, \sum w_j C_j\}$, where $g_{max} = \max\{g_j(u_j)\}$ ($g_j(u_j)$ is a nondecreasing resource cost function), $c_{max} = \max\{c_j(C_j)\}$ ($c_j(C_j)$ is a nondecreasing penalty cost function), and $\sum w_j U_j$ is the weighted number of tardy tasks (see Section 3.1). Computational complexities of the problems and the general scheme for the construction of Pareto sets and Pareto set ε-approximations were also presented.

In [Jan99] the model (13.3.16) was extended to one with $p_j = b_j + a'_j S_j - a_j u_j$, where S_j is a task starting time and a'_j is a task model parameter. The problems of minimization of the makespan, the total completion time and the lateness with the extended model including the constraint on the maximal resource amount $\hat{U}$, and also their inverse versions, were investigated e.g. in [IJR00].

Further generalizations concern the application of the model (13.3.16) for machine setup times [Jan99]. Single machine batch scheduling with resource dependent setup and processing time was examined in [CJK01], where polynomial time algorithms were presented to find an optimal batch sequence and resource allocations such that either the total weighted consumption $\sum w_j u_j$ is minimized subject to meeting task deadlines d_j, or the maximum task lateness is minimized subject to an upper bound on the total weighted resource consumption. Next, single machine group scheduling with resource dependent setup and processing times with continuous or discrete resource were considered in [NCJK 05, JKP05] for various criteria.

To end this section let us mention some results obtained for the processing time vs. resource amount model in case of dedicated processors. Two-machine flow shop problems with linear task models were studied by Janiak [Jan88b, Jan89a], where it was proved that the problem is NP-hard for the single criteria $\gamma = C_{max}$ and $\gamma = \Sigma u_j$, even for identical values of a_j on one of the machines and fixed processing times on the second machine. Approximation algorithms and an exact branch and bound algorithm were also presented in these papers. Flow shop and job shop problems with convex task models were considered in [GJ87, Jan86b, Jan88c, Jan88d, JS94, JP98, CJ00].

13.3.4 Ready Time vs. Resource Amount Model

In this section we assume that task processing times are given constants but ready times are continuously dependent on the amount of allocated continuous resource, i.e.

$$r_j = f_j(u_j),\ \underset{\sim}{u}_j \le u_j \le \tilde{u}_j,\ j = 1, 2, \cdots, n, \tag{13.3.17}$$

where all the lower and upper bounds of resource allocations, $\underset{\sim}{u}_j$ and $\tilde{u}_j$, are known constants.

As in Section 13.3.3 tasks are assumed to be non-preemptable, and we consider single machine problems only. Problems of this type appear e.g. in the ingot preheating process in steel mills [Jan91b].

Problem $1 \mid r_j = f_j(u_j), \Sigma u_j \le \hat{U} \mid C_{max}$

This problem is strongly NP-hard even in the special case of linear functions f_j (see (13.3.16)) and $\underset{\sim}{u}_j = 0$, $j = 1, 2, \cdots, n$, and is NP-hard in the case of $a_j = a$, $j = 1, 2, \cdots, n$ [Jan91b]. However, for identical models of r_j, i.e. for $f_j = f$, $\underset{\sim}{u}_j = \underset{\sim}{u}$ and $\tilde{u}_j = \tilde{u}$ for all j, the problem can be solved in polynomial time. In this case we know from [Jan86c] that an optimal solution $(\boldsymbol{z}^*, \boldsymbol{u}^*)$ is obtained by scheduling tasks according to non-increasing processing times p_j (thus defining permutation $\boldsymbol{z}^*$) and by allocating the continuous resource for $\boldsymbol{z}^*$ according to the following formulas: if

$$f(\tilde{u}_{z^*(1)}) + \sum_{j=1}^{n} p_{z^*(j)} \ge f(\underset{\sim}{u}) + \sum_{j=2}^{n} p_{z^*(j)}$$

where

$$\tilde{u}_{z^*(1)} = \min\{(\hat{U} - (n-1)\underset{\sim}{u}),\ \tilde{u}\},$$

then

$$u^*_{z^*(1)} = \tilde{u}_{z^*(1)},\ u^*_{z(j)} = \underset{\sim}{u},\ j = 2, 3, \cdots, n.$$

Otherwise,

$$u^*_{z^*(j)} = f^{-1}\Big(r - \big(\sum_{i=j}^{k-1} p_{z^*(i)} + d\big)\Big),\ j = 1, 2, \cdots, k-1,$$

$$u^*_{z^*(k)} = f^{-1}(r - d),\ u^*_{z^*(j)} = \underset{\sim}{u},\ j = k+1, k+2, \cdots, n,$$

where $r = f(\underset{\sim}{u})$, and $k-1$ is the maximal natural number such that

$$\Big(\sum_{j=1}^{k-1} f^{-1}\big(r - \sum_{i=j}^{k-1} p_{z^*(i)}\big) + (n - (k-1))\underset{\sim}{u} \le \hat{U}\Big) \text{ and } \Big(f^{-1}\big(r - \sum_{j=1}^{k-1} p_{z^*(j)}\big) \le \tilde{u}\Big),$$

$$d = \min\{(r - \sum_{j=1}^{k-1} p_{z^*(j)} - f(\tilde{u})), d'\},$$

with d' following from the equation

$$\sum_{j=1}^{k-1} f^{-1}(r - \sum_{i=j}^{k-1} p_{z^*(i)} - d') + f^{-1}(r - d') + (n-k)\underset{\sim}{u} = \hat{U}.$$

Thus, if we are able to calculate f, f^{-1} and d' in time $O(g(n))$, then $(\boldsymbol{z}^*, \boldsymbol{u}^*)$ is calculated in $O(\max\{g(n), n\log n\})$ time, i.e. this time is polynomial if $g(n)$ is polynomial. For example, this is the case if f is linear. In special situations where f_j is linear and $b_j = b$ for $j = 1,2,\cdots,n$, algorithms of time complexity $O(n\log n)$ exist. These situations are as follows:

(*i*) $\tilde{u}_j = \tilde{u},\ p_j = p,\ j = 1, 2, \cdots, n,$

(*ii*) $a_j = a,\ p_j = p,\ j = 1, 2, \cdots, n.$

An optimal solution $(\boldsymbol{z}^*, \boldsymbol{u}^*)$ is obtained by scheduling the tasks according to non-increasing values of a_j in case (*i*), non-increasing $\tilde{u}_j$ in case (*ii*), and by allocating the continuous resource using corresponding modifications of the above formulae [Jan89b].

For arbitrary linear functions f_j, Janiak [Jan89b] was able to prove that for given $\boldsymbol{z} \in \mathcal{Z}$, an optimal resource allocation $\boldsymbol{u}_z^*$ can be calculated in $O(n^2)$ time using the following algorithm.

Algorithm 13.3.7 *for finding* $\boldsymbol{u}^*$ *for* $1 \mid r_j = b_j - a_j u_j, \Sigma u_j \le \hat{U} \mid C_{max}$ [Jan89b].

begin
for $j := 1$ **to** n **do**
  begin
  $u^*_{z(j)} := 0;$
  $C_{z(j)} := b_{z(j)} + \sum_{i=j}^{n} p_{z(i)};$
  end;
$\mathcal{J} := \{z(j) \mid j = 1, 2, \cdots, n\};$
$l := 0;$
$C_0 := 0;$
$\mathcal{J}_0 := 0;$
while $\mathcal{J} \neq \emptyset$ **do**
  begin
  $l := l + 1;$
  Find set $\mathcal{J}_l = \{z(j) \mid z(j) \in \mathcal{J}$ and $C_{z(j)} = \min_{z(i) \in \mathcal{J}} \{C_{z(i)}\}\};$
  $\mathcal{J} = \mathcal{J} - \mathcal{J}_l;$

end;
$Q := \mathcal{J}_l$**;**
while $(\hat{U} \neq 0$ **and** $l \neq 0$ **and** $\min_{j \in Q} \{\tilde{u}_j - u_j^*\} \neq 0)$ **do**
begin
$x := \min \{C_q - C_p, \hat{U} / \sum_{j \in Q} (1/a_j), \min_{j \in Q} \{a_j(\tilde{u}_j - u_j^*)\}\}$**;**
-- p and q are indices of tasks belonging to sets Q and $\mathcal{J}_{l-1}$, respectively
for $j \in Q$ **do** $u_j^* := u_j^* + x/a_j$**;**
$\hat{U} := \hat{U} - \sum_{j \in Q} x/a_j$**;**
$l := l - 1$**;**
$Q := Q \cup \mathcal{J}_l$**;**
end;
$\boldsymbol{u}_z^* := [u_1^*, \cdots, u_n^*]$**;** -- $\boldsymbol{u}_z^*$ is an optimal resource allocation for permutation $\boldsymbol{z}$
end;

In the same paper it has been shown that in the case of $a_j = a$, $\tilde{u}_j = \tilde{u}$, $p_j = p$ for $j = 1, 2, \cdots, n$, an optimal solution $(\boldsymbol{z}^*, \boldsymbol{u}^*)$ is obtained when tasks are scheduled in order of non-decreasing b_j and the resource is allocated according to Algorithm 13.3.7. The same is also true for problems in which the above permutation is in accordance with the non-increasing orders of a_j, $\tilde{u}_j$ and p_j. Of course, Algorithm 13.3.7 can also be used for finding resource allocations for permutations $\boldsymbol{z} \in \mathcal{Z}$ defined heuristically. In [Jan89b] 25 such heuristics with the (best possible) worst case bound 2 were compared experimentally. The best results for "low" resource level ($\hat{U} = 0.2 \cdot \sum_{j=1}^{n} \tilde{u}_j$) were produced by ordering tasks according to non-decreasing b_j, whereas for "high" resource level ($\hat{U} = 0.9 \cdot \sum_{j=1}^{n} \tilde{u}_j$) sorting tasks according to non-decreasing values of $b_j - a_j\tilde{u}_j$ turned out to be most efficient.

Problem $1 \mid r_j = f_j(u_j), C_{max} \leq \hat{C} \mid \Sigma u_j$

Similarly as for $1 \mid r_j = f_j(u_j), \Sigma u_j \leq \hat{U} \mid C_{max}$ it can be proved that the considered problem is already strongly NP-hard for $f_j = b_j - a_j u_j$, $j = 1, 2, \cdots, n$, and NP-hard for $a_j = a$, $j = 1, 2, \cdots, n$ (see [Jan91b]). Also similarly to the solution of the first problem, if $f_j = f$, $\underset{\sim}{u}_j = \underset{\sim}{u}$ for all j, the problem is solved optimally by scheduling tasks according to non-increasing p_j (thus defining permutation $\boldsymbol{z}^*$) and by allocating the resource according to the following condition. If

$$r + \sum_{j=1}^{n} p_j - \hat{C} \leq p_{z^*(1)}, \text{ where } r = f(\underset{\sim}{u}),$$

then

$$u^*_{z^*(1)} = f^{-1}\big(\hat{C} - \sum_{j=1}^{n} p_j\big),\ u^*_{z^*(j)} = \underset{\sim}{u},\ j = 2,3,\cdots,n,$$

and otherwise

$$u^*_{z^*(j)} = f^{-1}\big(\hat{C} - \sum_{i=j}^{n} p_{z^*(i)}\big) = f^{-1}\big(r - \sum_{i=j}^{k-1} p_{z^*(i)} - d\big) \quad \text{for } j = 1,2,\cdots,k-1,$$

$$u^*_{z^*(k)} = f^{-1}\big(\hat{C} - \sum_{i=k}^{n} p_{z^*(i)}\big) = f^{-1}(r-d),$$

$$u^*_{z^*(j)} = f^{-1}(r) = \underset{\sim}{u},\ j = k+1, k+2,\cdots,n,$$

where k is the maximal natural number such that

$$\sum_{i=1}^{k-1} p_{z^*(i)} \le r + \sum_{j=1}^{n} p_j - \hat{C},$$

$$d = r + \sum_{j=1}^{n} p_j - \hat{C} - \sum_{i=1}^{k-1} p_{z^*(i)} = r + \sum_{i=k}^{n} p_{z^*(i)} - \hat{C}.$$

Thus, if we are able to calculate f and f^{-1} in $O(g(n))$ time, then finding $(\boldsymbol{z}^*, \boldsymbol{u}^*)$ needs $O(\max\{g(n), n\log n\})$ time.

Notice that it is generally sufficient to consider $\hat{C}$ for which $\underset{\sim}{C}_{max} \le \hat{C} \le \widetilde{C}_{max}$, where $\underset{\sim}{C}_{max} = \min_{z\in Z} C_{max}(\boldsymbol{z}, \widetilde{\boldsymbol{u}})$ and $\widetilde{C}_{max} = \min_{z\in Z} C_{max}(\boldsymbol{z}, \underset{\sim}{\boldsymbol{u}})$. In particular, for identical $f_j, \underset{\sim}{u}_j, \widetilde{u}_j, j = 1,2,\cdots,n$, we have

$$C_{max}(\boldsymbol{z}, \widetilde{\boldsymbol{u}}) = \underset{\sim}{C}_{max} = f(\widetilde{u}) + \sum_{j=1}^{n} p_j$$

and

$$C_{max}(\boldsymbol{z}, \underset{\sim}{\boldsymbol{u}}) = \widetilde{C}_{max} = f(\underset{\sim}{u}) + \sum_{j=1}^{n} p_j \quad \text{for each } \boldsymbol{z} \in Z.$$

If functions f_j are not identical and linear, then for given $\boldsymbol{z} \in Z$ an optimal $\boldsymbol{u}^*_z$ is obtained in $O(n)$ time using the formula [Jan91b]

$$u^*_{z(j)} = \max\{0, (b_{z(j)} + \sum_{i=j}^{n} p_{z(i)} - \hat{C})/a_{z(j)}\},\ j = 1,2,\cdots,n. \tag{13.3.18}$$

This follows simply from the linear programming formulation of the problem. On the same basis it is easy to see that the cases:

(*i*) $b_j = b,\ \widetilde{u}_j = \widetilde{u},\ p_j = p,\ j = 1,2,\cdots,n,$

(*ii*) $b_j = b,\ a_j = a,\ p_j = p,\ j = 1,2,\cdots,n,$

(*iii*) $a_j = a,\ \widetilde{u}_j = \widetilde{u},\ p_j = p,\ j = 1,2,\cdots,n$

are solvable in $O(n\log n)$ time by scheduling tasks according to non-increasing a_j in case (*i*), non-increasing $\tilde{u}_j$ in case (*ii*), and non-increasing b_j in case (*iii*), and by allocating the resource according to (13.3.18). For each of these cases $\boldsymbol{z}^*$ does not depend on $\hat{C}$, and $\underset{\sim}{C}_{max} = C_{max}(\boldsymbol{z}^*, \tilde{\boldsymbol{u}})$, $\tilde{C}_{max} = C_{max}(\boldsymbol{z}^*, 0)$.

Heuristics in which $\boldsymbol{z}$ is defined heuristically and $\boldsymbol{u}_z^*$ is calculated according to (13.3.18) were studied in [Jan91b]. The best results were obtained by scheduling tasks according to non-decreasing b_j. Unfortunately, the worst-case performance of these heuristics is not known.

On the basis of the presented results, the set of all Pareto-optimal solutions can be constructed for some bi-criterion problems of type $1 \mid r_j = f_j(u_j) \mid C_{max} \wedge \Sigma u_j$ using the ideas described in [JC94]. For linear models this set was constructed in [Jan 91b].

The problems considered in this section were generalized in [Jan 97] for the case with arbitrary precedence constraints, where it was proved that they are NP-hard even for identical linear models of r_j. When additionally all processing times are identical, the optimal solution (z^*, u^*) can be constructed in $O(n^2)$ time.

In [JL94, Jan99] the single and parallel machine scheduling problems with nonlinear function: release time vs. resource consumption, common for all tasks, with different task resource consumption rates were considered. The following criteria were minimized: the total weighted task completion time subject to a constrained maximal resource amount [JL94], the total resource utilization subject to a constrained total weighted completion time, and the bi-criteria approach [Jan99]. The borders between NP-hard and polynomially solvable cases were found.

Further generalization of the release time model was made in [Jan99], where the single machine scheduling problem with the model (13.3.1) applied to release times was considered. Due to some problem properties, the difficult dynamic resource allocation problem was reduced to a simple convex programming one. Some approximation algorithms with the worst case analysis were also presented.

References

BBKR86 J. Błażewicz, J. Barcelo, W. Kubiak, H. Röck, Scheduling tasks on two processors with deadlines and additional resources, *Eur. J. Oper. Res.* 26, 1986, 364-370.

BCSW86 J. Błażewicz, W. Cellary, R. Słowiński, J. Węglarz, *Scheduling under Resource Constraints: Deterministic Models*, J. C. Baltzer, Basel, 1986.

BDM+99 P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch, Resource-constrained project scheduling: notation, classification, models, and methods, *Eur. J. Oper. Res.* 112, 1999, 3-41.

BE83 J. Błażewicz, K. Ecker, A linear time algorithm for restricted bin packing and scheduling problems, *Oper. Res. Lett.* 2, 1983, 80-83.

BE94 J. Błażewicz, K. Ecker, Multiprocessor task scheduling with resource requirements, *Real-Time Syst.* 6, 1994, 37-54.

BKS89 J. Błażewicz, W. Kubiak, J. Szwarcfiter, Scheduling independent fixed-type tasks, in: R. Słowiński, J. Węglarz (eds.), *Advances in Project Scheduling*, Elsevier, Amsterdam, 1989, 225-236.

Bla78 J. Błażewicz, Complexity of computer scheduling algorithms under resource constraints, *Proceedings of the 1st Meeting AFCET - SMF on Applied Mathematics*, Palaiseau, 1978, 169-178.

BLRK83 J. Błażewicz, J. K. Lenstra, A. H. G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, *Discret Appl. Math.* 5, 1983, 11-24.

CD73 E. G. Coffman Jr., P. J. Denning, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, N. J., 1973.

CGJ84 E. G. Coffman Jr., M. R. Garey, D. S. Johnson, Approximation algorithms for bin-packing - an updated survey, in: G. Ausiello, M. Lucertini, P. Serafini (eds.), *Algorithms Design for Computer System Design*, Springer, Vienna, 1984, 49-106.

CGJP83 E. G. Coffman Jr., M. R. Garey, D. S. Johnson, A. S. La Paugh, Scheduling file transfers in a distributed network, *Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Montreal, 1983.

CJ00 T.-C. E. Cheng, A. Janiak, A permutation flow-shop scheduling problem with convex models of operation processing times, *Ann. Oper. Res*. 96, 2000, 39-60.

CJK98 T.-C. E. Cheng, A. Janiak , M.Y. Kovalyov, Bicriterion single machine scheduling with resource dependent processing times, *SIAM J. Optim*. 8, 1998, 617-630.

CJK01 T.-C. E. Cheng, A. Janiak , M.Y. Kovalyov, Single machine batch scheduling with resource dependent setup and processing times, *Eur. J. Oper. Res*. 135, 2001, 177-183.

GG75 M. R. Garey, R. L. Graham, Bounds for multiprocessor scheduling with resource constraints, *SIAM J. Comput.* 4, 1975, 187-200.

GJ75 M. R. Garey, D. S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, *SIAM J. Comput.* 4, 1975, 397-411.

GJ87 J. Grabowski, A. Janiak, Job-shop scheduling with resource-time models of operations, *Eur. J. Oper. Res.* 28, 1987, 58-73.

IJR00 D. Iwanowski, A. Janiak, A. Rogala, Scheduling jobs with start time and resource dependent processing times, in: K. Inderfurth, G. Schwodianer, W. Domschke, F. Juhnke, P. Kleinschmidt, G. Wascher (eds), *Operations Research Proceedings 1999*, Springer, Berlin, 2000, 389-396.

Jan86a A. Janiak, One-machine scheduling problems with resource constraints, in: A. Prékopa, J. Szelezán, B. Strazicky (eds.), System Modelling and Optimization, *Lect. Notes Contr. Inf.* 84, 1986, 358-364.

Jan86b A. Janiak, Flow-shop scheduling with controllable operation processing times, in: H. P. Geering, M. Mansour (eds.), *Large Scale Systems: Theory and Applications*, Pergamon Press, 1986, 602-605.

Jan86c A. Janiak, Time-optimal control in a single machine problem with resource constraints, *Automatica* 22, 1986, 745-747.

Jan88a A. Janiak, Single machine sequencing with linear models of jobs subject to precedence constraints, *Archiwum Automatyki i Telemechaniki* 33, 1988, 203-210.

Jan88b A. Janiak, Permutation flow shop problem with linear models of operations, *Zeszyty Naukowe Politechniki Śląskiej, Automatyka* 94, 1988, 125-138 (in Polish).

Jan88c A. Janiak, Minimization of the total resource consumption in permutation flow-shop sequencing subject to a given makespan, *Journal of Modelling, Simulation and Control* 13, 1988, 1-11.

Jan88d A. Janiak, General flow-shop scheduling with resource constraints, *Int. J. Prod. Res.* 26, 1988, 1089-1103.

Jan89a A. Janiak, Minimization of resource consumption under a given deadline in two-processor flow-shop scheduling problem, *Inf. Process. Lett.* 32, 1989, 101-112.

Jan89b A. Janiak, Minimization of the blooming mill standstills - mathematical model. Suboptimal algorithms, *Zeszyty Naukowe AGH, Mechanika* 8, 1989, 37-49.

Jan91a A. Janiak, *Exact and Approximation Algorithms of Job Scheduling and Resource Allocation in Discrete Industrial Processes*, Prace Naukowe Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej 87, Monografie 20, Wrocław, 1991 (in Polish).

Jan91b A. Janiak, Single machine scheduling problem with a common deadline and resource dependent release dates, *Eur. J. Oper. Res.* 53, 1991, 317-325.

Jan97 A. Janiak, Computational complexity analysis of single machine scheduling problems with job release dates dependent on resources, in: U. Zimmermann, U. Derigs, W. Gaul, R.H. Möhring, K.P. Schuster (eds.), *Operations Research Proceedings 1996,* Springer, Berlin, 1997, 203-207.

Jan98a A. Janiak, Single machine sequencing with linear models of release dates, *Nav. Res. Logist.* 45, 1998, 99-113.

Jan98b A. Janiak, Minimization of the makespan in a two-machine problem under given resource constraints, *Eur. J. Oper. Res*. 107, 1988, 325-337.

Jan99 A. Janiak, *Selected Problems and Algorithms of Scheduling and Resource Allocation*, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1999 (in Polish).

JC94 A. Janiak, T.-C. E. Cheng, Resource optimal control in some simple-machine scheduling problems, *IEEE Trans. Aut. Contr.* 39, 1994, 1243-1246.

JK96 A. Janiak, M. Y. Kovalyov, Single machine scheduling subject to deadlines and resource dependent processing times, *Eur. J. Oper. Res*. 94, 1996, 284-291.

JKP05 A. Janiak , M. Y. Kovalyov, M.-C. Portmann, Single machine group scheduling with resource dependent setups and processing times, *Eur. J. Oper. Res*. 162, 2005, 112-121.

JL94 A. Janiak, C.-L. Li, Scheduling to minimize the total weighted completion time with a constraint on the release time resource consumption, *Math. Comput. Model.* 20, 1994, 53-58.

JP98 A. Janiak, M.-C. Portmann, Genetic algorithm for the permutation flow-shop scheduling problem with linear models of operations, *Ann. Oper. Res*. 83, 1998, 95-114.

JS88 A. Janiak, A. Stankiewicz, On time-optimal control of a sequence of projects of activities under time-variable resource, *IEEE Trans. Aut. Contr.* 33, 1988, 313-316.

JS94 A. Janiak, T. Szkodny, Job-shop scheduling with convex models of operations, *Math. Comput. Model.* 20, 1994, 59-68.

JW98 J. Józefowska, J. Węglarz, On a methodology for discrete-continuous scheduling, *Euro. J. Oper. Res.* 107, 1998, 338-353.

Kar72 R. M. Karp, Reducibility among combinatorial problems, in: R. E. Miller, J. W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 1972, 85-103.

Kar84 N. Karmarkar, A new polynomial-time algorithm for linear programming, *Combinatorica* 4, 1984, 373-395.

KE75 O. Kariv, S. Even, An $O(n^2)$ algorithm for maximum matching in general graphs, *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, 1975, 100-112.

KSS75 K. L. Krause, V. Y. Shen, H. D. Schwetman, Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems, *J. ACM* 22, 1975, 522-550 (Erratum: *J. ACM* 24, 1977, 527).

Law76 E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York 1976.

Len83 H. W. Lenstra, Jr., Integer programming with a fixed number of variables, *Math. Oper. Res.* 8, 1983, 538-548.

McN59 R. McNaughton, Scheduling with deadlines and loss functions, *Manage. Sci.* 12, 1959, 1-12.

MWW04 M. Mika, G. Waligóra, J. Węglarz, A metaheuristic approach to scheduling workflow jobs on a grid, in: J. Nabrzyski, J. M. Schopf, J. Węglarz (eds.), *Grid Resource Management*, Kluwer, Boston 2004, 295-318.

NCJK05 C.T. Ng, T.-C. E. Cheng, A. Janiak, M. Y. Kovalyov, Group scheduling with controllable setup and processing times, Minimizing total weighted completion time, *Ann. Oper. Res*. 133, 2005, 163-174.

NZ81 E. Nowicki, S. Zdrzalka, Optimal control of a complex of independent operations, *Int. J. Syst. Sci.* 12, 1981, 77-93.

NZ84a E. Nowicki, S. Zdrzalka, Optimal control policies for resource allocation in an activity network, *Eur. J. Oper. Res.* 16, 1984, 198-214.

NZ84b E. Nowicki, S. Zdrzalka, Scheduling jobs with controllable processing times as an optimal control problem, *Int. J. Contr.* 39, 1984, 839-848.

SW89 R. Słowiński, J. Węglarz (eds.), *Advances in Project Scheduling*, Elsevier, Amsterdam, 1989.

WBCS77 J. Węglarz, J. Błażewicz, W. Cellary, R. Słowiński, An automatic revised simplex method for constrained resource network scheduling, *ACM Trans. Math. Softw.* 3, 295-300, 1977.

Weg80 J. Węglarz, Multiprocessor scheduling with memory allocation - a deterministic approach, *IEEE Trans. Comput.* C-29, 1980, 703-709.

Weg81 J. Węglarz, Project scheduling with continuously-divisible, doubly constrained resources, *Manage. Sci.* 27, 1981, 1040-1052.

Weg82 J. Węglarz, Modelling and control of dynamic resource allocation project scheduling systems, in: S. G. Tzafestas (ed.), *Optimization and Control of Dynamic Operational Research Models*, North-Holland, Amsterdam, 1982.

Weg89 J. Węglarz, Project scheduling under continuous processing speed vs. resource amount functions, 1989. in: R. Słowiński, J. Węglarz (eds.), *Advances in Project Scheduling*, Elsevier, 1989, 273-277.

Weg91 J. Węglarz, Synthesis problems in allocating continuous, doubly constrained resources, in: H. E. Bradley (ed.), *Operational Research '90 - Selected Papers from the 12th IFORS International Conference*, Pergamon Press, Oxford, 1991, 715-725.

Weg99 J. Węglarz (ed.), *Project Scheduling - Recent Models, Algorithms and Applications*, Kluwer Academic Publ., 1999.

WJMW11 J. Węglarz, J. Józefowska, M. Mika, G. Waligóra, Project scheduling with finite or infinite number of activity processing modes, *Eur. J. Oper. Res.* 208, 2011, 177-205.