A Job Allocation Algorithm for Parallel Processors

Nodari Vakhania
Facultad de Ciencias
Universidad Autonoma del Estado de Morelos
Av. Universidad 1001
Cuernavaca 62210, Mor., Mexico

Abstract

We are given a finite set of jobs of equal processing times with readiness times and tails and a set of identical processors. The aim is to schedule the given set of jobs on the given set of processors to minimize the total processing time (or makespan). An algorithm for that problem with the time complexity $O(n \log n)$ was proposed earlier in [10]. This algorithm improves the running time of the previously known best algorithm [9] under the assumption that the tails of all jobs are bounded by some constant. In this paper we show that an algorithm based on the ideas of the algorithm from [10] can be constructed in which the above restriction is removed.

Key words: scheduling, identical machines, readiness time, tail, computational complexity.

1. Introduction. We consider the following machine sequencing problem $P1$: there are given a set $I = \{1, 2, ..., n\}$ of *jobs* and a set $M = \{1, 2, ..., m\}$ of *machines* (or *processors*). Each job has to be performed on any of the given m machines; the *processing time* of any job (on any machine) is a given integer number p. Job $i\,(i = 1, 2, ..., n)$ is available at its integer *readiness time* a_i (this job cannot be started before the time a_i) and has an integer *tail* q_i (interpreted as an additional amount of time needed for the termination of job i once it is processed on a machine). A *schedule* is a function which assigns to each job a machine and starting time (on that machine). An (integer) *starting time* t_i^S of job i (in the schedule S) is the time at which this job is scheduled to be performed on a machine. The *completion time* of job i on a *machine* $c_i^S = t_i^S + p$. The *full completion time* of job i in the schedule S is $c_i^S + q_i$ (notice that q_i doesn't require any machine time). Each machine can handle at most one job at a time, that is, if jobs i and j are scheduled on the same machine then either $c_i^S \leq t_j^S$ or $c_j^S \leq t_i^S$. The *preemption* of jobs is not allowed, that is, each job is performed during the time interval $[t_i^S, t_i^S + p]$ on a machine. A *feasible schedule* is a schedule which satisfies the above restrictions. The objective is to find

an *optimal schedule*, that is, a feasible schedule which minimizes the *makespan* (the maximum full completion time of all jobs).

An alternate formulation of the above problem is the one with the *due dates* (abbreviated as $P2$): instead of the tail q_i for each job i an integer due date d_i is given (d_i is the desirable completion time of job i). The lateness L_i^S of job i in a schedule S is defined as:

$$L_i^S = \begin{cases} 0, \text{ if } c_i^S \leq d_i; \\ c_i^S - d_i, \text{ otherwise.} \end{cases}$$

The objective is to find an optimal schedule, that is, a feasible schedule which minimizes the maximum lateness $L_{max}^S = \max\{L_i^S | i = 1, 2, ...n\}$.

The equivalence between problems *P1* and *P2* is established by a simple transformation (see [2]).

If we allow in $P1$ and $P2$ different processing times we get strongly NP-complete problem even in the single-machine case (see [1,2,4,5,7]).

If we replace in $P2$ due dates with deadlines and we look for a feasible schedule, we get the corresponding feasibility problem PF (by $PF1$ the one-machine version of PF is abbreviated). In a feasible schedule S of PF no job can be delayed, that is, $c_i^S \leq d_i$, for $i = 1, 2, ..., n$ (in a feasible schedule of $P2$ we allow the existence of such jobs and we look for a schedule which minimizes the maximum delay).

It has been proved that PF is solvable in polynomial time.

An $O(n^2 \log n)$ algorithm for $PF1$ is described in [3]. In this algorithm the so-called active schedules are generated; U is an active (partial) schedule if $t_i^U + p \leq d_i$, for any job $i \in U$ and $d_j > y_U$ for any unscheduled job J, where $y_U = \max\{t_i^U + p | i \in U\}$; x–active schedule U_x is an active schedule with $y_{U_x} < x$. Beginning at the time $x = 0$, in the algorithm x–active schedules are generated, selecting at the moment x the job with the minimal due date among the jobs, which were not scheduled in U_{x-p} and available in the time interval $[x - p, x]$. This way new urgent job which becomes available at the moment x, can replace the earlier scheduled less urgent one when the latter job delays it. The resulting x–active schedule is feasible if it contains all the jobs, otherwise there is no feasible schedule.

An algorithm with the better performance for the same problem was proposed in [6]. This $O(n \log n)$ algorithm uses the so-called forbidden regions for the construction of a feasible schedule. A forbidden region is a time interval in a schedule in which it is forbidden to start any job. The algorithm consists of two parts. In part I the forbidden regions are defined and a failure is declared if there exists no feasible schedule. In part II schedule is generated using the "earliest deadline scheduling rule" and forbidden regions declared in part I.

The first polynomial algorithms for PF were proposed in [8] and [9] with the time complexity $O(n^3 \log \log n)$ and $O(n^2 m)$ respectively. Both algorithms apply the concept of the forbidden regions introduced in [6]. The algorithm in [8] uses the earliest deadline scheduling rule for the construction of feasible schedules, while

each time the next scheduled job failures to meet its deadline backtracking is accomplished and a new forbidden region (or barrier) is declared. In the algorithm from [9] a more sophisticated method is used for the generation of the forbidden regions before the jobs are actually scheduled what yields the better computation complexity results.

The minimization problem $P2$ can be solved by the repeated application of an algorithm for the corresponding feasibility problem PF: We iteratively increase the due dates of all jobs by some constant until we find a feasible schedule of the feasibility problem with the modified data. Since the maximum lateness will depend on the number of jobs, we may need to apply such an algorithm $O(n)$ times. Thus, algorithms [8,9] if applied to problem $P1$ have the time complexity $O(n^4 \log\log n)$ and $O(n^3 m)$, respectively.

Recently in [10] was proposed an algorithm for problem $P1$ which gives the better running time but under the assumption that the maximal job tail is bounded by some constant. Its time complexity is $O(n \log n)$, though without the above assumption it is $O({q_{max}}^3 n \log n)$ (here q_{max} is the maximal job tail), which makes it pseudo-polynomial. This algorithm accomplishes a "limited" enumeration of the set of the special kind of feasible schedules, the so called complementary schedules. Special behaviour alternatives are introduced for these schedules and they are repeatedly analyzed during the search.

The algorithm which we propose here uses the concepts of the one from [10] though it removes the restriction about the maximal job tail. It has the time complexity $O(\gamma m n \log n)$, where γ can take any of the values q_{max} or n. So, the proposed here algorithm improves the running time of the algorithm from [10] for the general case, as well as it improves the running time of the algorithms from [8] and [9]. Moreover, while the bounds of the latter algorithms are tight (the preprocessing stage in these algorithms takes the fixed amount of time), our algorithm applies an enumeration tree and, speaking informally, it is very unlike that the above bound will be ever reached. With the nodes of our enumeration tree, as in the algorithm from [10], the complementary schedules are associated. The complementary schedules are complete schedules obtained by the application of the greatest tail scheduling heuristic to a specially modified problem instance. With each complementary schedule we associate conjunctive graph. Our search for an optimal solution is accomplished on the bases of analysis of a critical path behavior in that graphs.

In the next Section 2 we introduce the basic definitions and notations. In Section 3 we give some properties of the greatest tail schedules. Section 4 we study the complementary schedules, we describe the algorithm and indicate its computational complexity. In the final Section 5 we give the concluding remarks.

2. Basic Concepts. A schedule S can be represented by a directed weighted graph $G_S = (X, E \cup E_S)$, where X is the set of nodes in G_S, E is the set of *initial arcs* and E_S is the set of *complementary arcs*. The set X consists of $n+2$ nodes

$0, 1, ..., n, *$, where node $j \in X,\ j \notin \{0, *\}$ represents the unique job $j \in I$ (for the simplicity, from now on we may refer to node $i \in G_S$ as to job i); 0 and $*$ are fictitious source and sink nodes. For each $j \in I$ we have $(0, j) \in E$, $(j, *) \in E$. We associate the weight a_j (respectively, $p+q_j$) with each arc $(0, j)$ (respectively, $(j, *)$) from E. The set E_S we form in the following way. We add an arc (i, j) $(i, j \notin \{0, *\})$ to E_S with the associated weight p, when we schedule job j directly after job i, both on the same machine. The makespan of S is then the length of a critical path in G_S. Notice that a critical path in G_S can be found in time $O(n)$.

G_S contains m unconnected 'chains', each of them consisting of jobs scheduled successively on one particular machine. First such a chain consists of jobs scheduled on machine 1, the second such a chain consists of jobs scheduled on machine 2 and so on, the last chain consists of jobs scheduled on the last machine m (we number our set of machines arbitrarily so that we can distinguish them by indexes). Each of these chains may contain one or more critical paths. Among all critical paths in G_S we distinguish the ones associated with the machine (or equivalently, with the chain) with the greatest index and call them the *rightmost* critical paths. Among critical paths of one particular chain we distinguish critical paths with the maximal number of jobs. The first such a path in a chain we call the *maximal* path. We will be further interested mainly in the rightmost maximal paths. Notice that such a path is defined uniquely in any schedule. The rightmost maximal path in S we denote by $\mu(S)$.

Let $j_1, j_2, ..., j_n$ be a permutation of n jobs from I in a schedule S. We say that job $j_i,\ i = 1, 2, ..., n$ has the *ordinal number* i in S. The ordinal number of job j in S we denote by $\mathrm{ord}(j, S)$.

In a schedule we distinguish n *positions* on the given m machines. These positions are successively filled out by the jobs scheduled in turn on *adjacent* machines (adjacent to machine k is machine $k+1$ for $k = 1, 2, ..., m-1$, and adjacent to machine m is machine 1). The *starting time* of kth position in S, $t(k, S)$, is the starting time of the job scheduled in that position.

The complete schedules with which we deal in our algorithm are represented as a nodes in the *solution tree* T. We apply the *greatest tail heuristic* (abbreviated GTH) to generate these schedules. As we describe later, we iteratively modify our current problem instance in a special way and apply the GTH to the modified instances, generating in this way different schedules with our heuristic. The greatest tail heuristic is an adaptation of the smallest due date heuristic for a single machine: We repeatedly determine a ready job with the greatest tail and schedule it on the next adjacent machine. Below is the description.

PROCEDURE GREATEST TAIL.
{ returns the GTS }
BEGIN{greatest tail}
(0) $t := \min\{a_i | i \in I\};\ A := I;$
$R(k) := 0,\ k = 1, 2, ..., m;$
{ R_k is the release time of machine k }

(1) Among the unscheduled jobs $l \in A$ with $a_l \leq t$ schedule next job j with the greatest tail on machine k, $k = \mathrm{ord}(j) \bmod m$ (break ties arbitrarily);

$t_j := \max\{t, R_k\}$; $R_k := t_j + p_j$; $A := A \setminus \{j\}$;

$k' := (\mathrm{ord}(j) + 1) \bmod m$; { k' is the next available machine}

IF $A \neq \emptyset$ THEN $t := \max\{R_{k'}, \min\{a_i | i \in A\}\}$; go to (1)

ELSE GREATEST TAIL:= $\{t_j\}$ $(j = 1, 2, ..., n)$;

RETURN;

END.{greatest tail}

Schedules generated by the application of the above algorithm we call the *greatest tail schedules* (abbreviated GTS). All schedules we will be dealing with in this paper are the greatest tail schedules.

Proposition 1. If $\mathrm{ord}(j, S) > \mathrm{ord}(i, S)$ $(i, j \in S)$ *then* $t_j^S \geq t_i^S$, *for any greatest tail schedule* S.

Proof. Easily follows from the GTH. Indeed, for the first m positions the claim is obvious: Either all the jobs will start at time 0 or otherwise, if some job starts later, then that job has the minimal readiness time among all unscheduled jobs. Now take next m jobs with the ordinal numbers $m+1, m+2, \cdots, 2m$ and assume that we have two successively scheduled jobs i and j $(\mathrm{ord}(j) = \mathrm{ord}(i) + 1)$ such that $t_j < t_i$. Since the release time of the position in which i is scheduled cannot be more than that of the position in which j is scheduled (this we already showed) this implies that $r_j < r_i$. But then by the GTH j would be scheduled before i. Similarly we show the claim for all the remained jobs.//

A *gap* in a schedule is a time interval which is not occupied by any job. The greatest tail schedules consist of the sequence of one or more *blocks*. Intuitively, block is an 'isolated' part of a schedule. Any block, different from the last block (or equivalently, the first one, if they are the same) in any schedule contains the number of jobs which is multiple of m. Formally, block is the maximal sequence of successively scheduled jobs on adjacent machines such that the first m jobs in it are preceded by gaps or are the earliest scheduled jobs on their respective machines, and the last m scheduled jobs ($k < m$ jobs, correspondingly) are succeeded by gaps (are the latest scheduled jobs on their respective machines, correspondingly). For any two blocks B_1, $B_2 \in S$ either $B_1 > B_2$ or $B_1 < B_2$ holds, that is, either B_1 precedes B_2 in S or vice versa. In a given schedule, the *critical block* is the block containing the rightmost maximal path.

We give some other definitions.

The last scheduled job of the rightmost maximal path in S we call its *overflow job* and denote it by r or $r(S)$. Let B be the critical block in S. A job $l \in B$ such that $\mathrm{ord}(l) < \mathrm{ord}(r)$ we call an *emerge job* in S if $q_l < q_r$. We denote by $K'(S)$ the set of all emerge jobs in S.

The sequence of jobs scheduled in S between the emerge job l with the maximal ordinal number and the overflow job r (including this job) we call the *emerge sequence* and denote it by $C(S)$. An emerge job in S is said to be emerge for $C(S)$.

Notice that jobs of $C(S)$ are scheduled successively on adjacent machines and hence may belong to different paths in G_S.

We denote by $L(S)$ the length of the (rightmost) maximal path in G_S and by $L(S,j)$ the length of a longest path to the node j in G_S.

We deal with the special kind of the greatest tail schedules which we call the *complementary schedules*. Our first GTS we obtain by the application of the GTH to the initial problem instance (we call it the *initial complementary schedule*). Then iteratively, we modify our current problem instance and we build new GTS applying the GTH to the modified instance. Let S be a GTS and let $l \in K'(S)$. A complementary schedule of S, S_l is a GTS constructed for the problem instance specially modified from the current problem instance in such a way that job l is rescheduled after $C(S)$ and no job scheduled in S after $C(S)$ occurs before $C(S)$. Below we describe how the complementary schedules are built.

Let I^* contains all jobs scheduled before $C(S)$ in S except job $l \in K'(S)$ and all jobs of $C(S)$. Let, further S^* be the (partial) schedule obtained by the application of the GTH to the jobs of I^*. We redefine the readiness times of all jobs $i \in I \setminus I^*$ (including job l) as follows: $a_i := \max\{t_r^{S^*}, a_i\}$ (the value $t_r^{S^*}$ is called the *threshold value* for l). If we now apply the GTH to the remained jobs from $I \setminus I^*$, extending the partial schedule S^*, we obtain the final complementary schedule S_l.

In a modified problem instance the tails of all jobs are the same as in the initial problem instance, although the readiness times of some jobs (namely, all the emerge jobs) are increased artificially by the repeated application of the above described procedure. The threshold value of l $t_r^{S^*}$ is defined in such a way that these jobs are 'forced' and scheduled after more urgent jobs leaving for them some additional space.

From the definition of the complementary schedule it follows that job j with the minimal starting time in $C(S)$ will occupy the position $\mathrm{ord}(S,j)-1$ and the overflow job $r = r(S)$ will be shifted one position left in S_l. Clearly, the aim of generating complementary schedule S_l is to decrease the lenght of the rightmost maximal path in S.

The rightmost maximal path and the respective overflow job might alternate in different ways in a newly generated complementary schedule, that is, the consequences of rescheduling of an emerge job after an emerge sequence might be different. We distinguish five *behavior alternatives* of the rightmost maximal path (and the emerge job) of S in S_l.

The critical path in S_l is said to be:

(a) unmoved if $r(S_l) = r(S)$;

(b) rested on l if $r(S_l) = l$;

(c) shifted forward (respectively, *(d) shifted backward*) if $r(S_l)$ and $r(S)$ are in the same block ($r(S_l) \neq r(S)$, $r(S_l) \neq l$) and $\mathrm{ord}(r(S_l), S_l) > \mathrm{ord}(r(S), S_l)$ (respectively, $\mathrm{ord}(r(S_l), S_l) < \mathrm{ord}(r(S), S_l)$);

otherwise, the critical path is said to be *(e) relocated*, that is, $r(S_l)$ and $r(S)$ belong to different blocks.

Thus, for the instances of alternative (a) the overflow job of S_l is the same as that of S; for the instances of alternative (b) the overflow job in S_l becomes the rescheduled emerge job of S l. In the case of the instances of alternative (c) (respectively, alternative (d)) the overflow jobs of S_l and S still belong to the same block and the overflow job of S_l is scheduled after (respectively, before) the overflow job of S in S_l. Finally, for the instances of alternative (e) the overflow job in S_l "moves" to a block, different from the block to which the overflow job of S belongs. So, all the alternatives except the latter one are "local": For the instances of the first four alternatives we 'stay' in the old critical block (making further the necessary rearrangement) while with an instance of alternative five we "leave" the old critical block and make the necessary rearrangement in the new critical block. We repeatedly analyze the behaviour of the critical path in the newly generated complementary schedules (as it will be described later, different alternatives cause different computational efforts).

It is easy to check that all the five alternatives are attainable. The five alternatives are also exhaustive (we can refer to one of them in any S_l): The overflow job in S_l may remain the same as in S (the alternative (a)) or change to l (the alternative (b)). Otherwise, either it can move to another block (the alternative (e)) or stay in the current block. For the latter case we have two possibilities: Either $r(S_l)$ is scheduled after $r(S)$ in S_l (the alternative (c)) or it is scheduled before $r(S)$ (the alternative (d)). Thus, we proved the following

Proposition 2. The alternatives (a)–(e) are attainable and exhaustive.

3. Properties of the Complementary Schedules. Let S be a GTS and $l \in K'(S)$. We have the following

Lemma 1. There arises at least one gap in S_l between the $(\mathrm{ord}(l,S)-1)$*st scheduled job and the overflow job r.*

Proof. Let j be a job with the minimal readiness time among all jobs of $C(S)$ and jobs scheduled between job l and $C(S)$ in S. Assume first that j is not an emerge job. Then $q_j > q_l$. This yields $a_j > t_l^S$, since otherwise job j would be scheduled at the moment t_l^S in S by the GTH. From the definition of the complementary schedule we have that no job from those which were scheduled after $C(S)$ in S can occupy any interval before $C(S)$ in S_l. Then we obviously there is a gap $[t_l^S, a_j)$ in S_l.

Now suppose j is an emerge job with $q_j \le q_l$ and $a_j \le t_l^S$ (if $a_j > t_l^S$ then we have a gap $[t_l^S, a_j)$). Again, by the GTH and the definition of the complementary schedule, j will be $\mathrm{ord}(l,S)$st scheduled job in S_l and there will be a gap in S_l strictly before $(\mathrm{ord}(j,S)+1)$st scheduled job if j is the only remained emerge job. Suppose not, and suppose the next position is also occupied by another emerge job. Then we look for the position next to next and continue in this manner until we find the first non-emerge job (obviously, such a job will appear) and then we will a gap strictly before that job.//

The other useful properties of the complementary and greatest tail schedules we give in the following three lemmas and in Theorem 1.

Lemma 2. The makespan of a GTS *cannot be improved by rescheduling any non-emerge job.*

Proof. Obviously follows from the definition of a non-emerge job and Proposition 1.//

Lemma 3. The makespan of a GTS S *cannot be improved by reordering jobs of the emerge sequence* $C(S)$.

Proof. Suppose that in the emerge sequence $C(S)$ job m precedes job l and that we have interchanged the order of these two jobs in the schedule S'. Consider the two following possibilities: $a_l \leq t_m^S$ and $a_l > t_m^S$.

If $a_l \leq t_m^S$ then $q_m \geq q_l$ (by the GTH). Job m can be scheduled before or after the overflow job r in S'. The first alternative is obvious (see Proposition 1). For the second one we easily obtain $L(S', m) > L(S, r)$ since $q_m \geq q_r$.

If $a_l > t_m^S$ then we have a gap inside $C(S)$ in S'. Again, job m can be scheduled before or after the overflow job r. In the first case we obviously have $L(S', r) \geq L(S, r)$. In the second case, $L(S', m) \geq L(S, r)$ (since $q_m \geq q_r$).//

Let $\delta_S = c_l^S - a_j$, where l is the latest scheduled emerge job in the GTS S and $a_j = \min\{a_i | i \in C(S)\}$.

Lemma 4. The lower bound on the value of an optimal schedule is $L(S) - \delta_S$.

Proof. $L = t_r^S + p + q_r$ is the makespan of S. We cannot improve this value by reordering of jobs of the emerge sequence (Lemma 3). Then we can improve it only if we reschedule some other jobs in such a way that jobs from $C(S)$ would start their processing earlier. But by the definition of δ_S and the GTH, none of the jobs $i \in C(S)$ can start their processing earlier than at time $t_i^S - \delta_S$. Then the value $L(S) = L(S, r)$ can be decreased at most by δ_S and hence $L(S) - \delta_S$ is the resulting lower bound.//

We switch now to the complementary schedules.

Theorem 1. An optimal schedule belongs to the set of complementary schedules.

Proof. Consider any GTS S. We claim that if this schedule is not optimal then we can improve it only by generating complementary schedules. Coming from the definition of a complementary schedule, then we have to show that S cannot be improved by:

1. Rescheduling of any non-emerge job. This we have from Lemma 2;

2. Reordering the jobs of $C(S)$. We have this from Lemma 3;

3. Rescheduling an emerge job inside the emerge sequence. If we reschedule an emerge job inside the emerge sequence $C(S)$ then we can decrease $L(S)$ at most by δ_S (Lemma 4) while we increase it by p ($\delta_S < p$);

4. Reordering the jobs of a block different from the critical block. This case is obvious.//

Let S be a complementary schedule with the rightmost maximal path μ. Consider the set of the complementary schedules S_l, $l \in K'(S)$ and the magnitude, by which the length of μ is reduced in each of these schedules. As the following lemma shows

this magnitude may only decrease while we apply an emerge job which has an ordinal number less than that of an already applied emerge job.

Lemma 6. $L(S_l, r) \leq L(S_k, r)$ *if* $\mathrm{ord}(l, S) > \mathrm{ord}(k, S)$ $(l, k \in K'(S))$.

Proof. Consider the complementary schedule S_k with job j scheduled in $\mathrm{ord}(k, S)$st position in it. From the GTH we have that $t_j^{S_l} \geq t_l^S$ and we have the similar condition for all the jobs scheduled between job k and the overflow job r in S (in the other words, the starting time of a job, scheduled in ith, $\mathrm{ord}(k, S) \leq i \leq \mathrm{ord}(r, S)$, position in S_k is equal to or more than that of a job scheduled in the same position in S). Analogously, the first late position in S_l is $\mathrm{ord}(l, S)$th position. Now the condition of the lemma obviously implies inequalities of the form $t(s, S_l) \leq t(s, S_k)$, $s = \mathrm{ord}(l, S), \mathrm{ord}(l, S) + 1, ..., r$ (that is, all positions between the positions $\mathrm{ord}(l, S)$ and r in S_l will start no later than in S_k). Then our claim is proved since in both S_l and S_k job r is scheduled in $(\mathrm{ord}(r, S) - 1)$th position.//

In our solution tree T we destinguish two kinds of schedules, an open and closed ones. A *closed schedule* is a schedule without successors which cannot have successors, while an *open schedule* is a schedule which is not closed and has no successors.

Intuitively, it should be clear that if the critical path in S_l is rested on l then this schedule can be closed:

Lemma 7. Suppose in the complementary schedule $S_l, l \in K'(S)$, *the critical path is rested on* l. *Then:*

1. S_l *can be closed;*

2. Any complementary schedule S_k, *such that* $\mathrm{ord}(k, S) < \mathrm{ord}(l, S)$ *and* $q_k \geq q_l$ *can be neglected.*

Proof. Part 1. Suppose l' is an emerge job in S_l (if there is no such a job then S_l can be closed, see Lemma 2). This job is also emerge in S since $q_{l'} < q_l$. If $\mathrm{ord}(l', S) > \mathrm{ord}(l, S)$ then S_l can be neglected (this lemma, part 2). Let now $\mathrm{ord}(l', S) < \mathrm{ord}(l, S)$. Consider the complementary schedule $(S_l)_{l'}$. If $L((S_l)_{l'}, l') \geq L(S_l, l)$ then obviously $(S_l)_{l'}$ can be neglected. Assume $L((S_l)_{l'}, l') < L(S_l, l)$. Then also $L(S_{l'}, l') < L(S_l, l)$ since job l' in $S_{l'}$ will be scheduled in an earlier position than in $(S_l)_{l'}$ (see Proposition 1), hence $L(S_{l'}) < L(S_l)$ and again S_l can be closed.

Part 2. Obviously follows from Lemma 6.//

The following lemma, like Lemma 5, enables us to reduce the number of complementary schedules we generate. In a sense, it extends Lemma 6.

Lemma 8. If $\mathrm{ord}(l, S) > \mathrm{ord}(k, S)$ *and* $q_l \leq q_k$ $l, k \in K'(S)$, $S \in T$ *then the complementary schedule* S_k *can be neglected if the complementary schedule* S_l *is generated.*

Proof. Suppose that the critical path in S_l is rested on l. Then it is rested on k in S_k and $L(S_l, l) \leq L(S_k, k)$ since $q_l \leq q_k$. If the critical path in S_l is unmoved then from Lemma 6 we have

$$L(S_l, r) \leq L(S_k, r) \qquad (*)$$

and obviously the schedule S_k can be neglected.

Let the critical path in S_l be shifted forward. If the critical path in S_k is rested on k then this schedule cannot be further improved (Lemma 7); also it cannot be better than S_l since $q_l \leq q_k$ and $\mathrm{ord}(l,S) > \mathrm{ord}(k,S)$ (Lemma 6).

Suppose the critical path in S_l is unmoved. Again from Lemma 6 we have $L(S_l) < L(S_l,r) \leq L(S_k,r) = L(S_k)$ and obviously to decrease $L(S_k,r)$ we have to generate a complementary schedule of the form $(S_k)_{k'}$, $k' \in K'(S)$, $k' > k$ (as we already showed in Theorem 1, an optimal schedule is among the complementary schedules). We have $(S_k)_{k'} = (S_{k'})_k$. Then if there exists no $k'' \in K'(S)$, $\mathrm{ord}(k,S) > \mathrm{ord}(k'',S) > \mathrm{ord}(k',S)$ such that $q_{k''} \leq q_{k'}$, according to this thorem $(S_{k'})_k$ will be generated and $(S_k)_{k'}$ can be neglected. Otherwise, we continue to apply recursively this reasoning first to job k'' and then to the remained emerge jobs with the similar property until we find a complementary schedule which is generated by the conditions of the theorem.

If the critical path in S_k is shifted backward then again from Lemma 6 we have that $L(S_k) \geq L(S_l)$ and due to the equal processing times, obviously, none of the complementary schedules, the successors of S_k, can have makespan better than that of S_l (speaking informally, if we succeed in S_k we will be brought to a schedule which cannot be better than S_l).

Let now, in both S_l and S_k, a critical path be shifted forward and consider the sequence of the successively generated complementary schedules of the form $(..(S_l)_l \ldots)_l$, $(..(S_k)_k..)_k$ (these schedules are obtained from S_l and S_k by rescheduling repeatedly jobs l and k, respectively, as an emerge jobs). Observe that, if k is emerge in $(..(S_k)_k..)_k$, than l is also emerge in $(..(S_l)_l..)_l$ since $q_l \leq q_k$. Besides, the ordinal number of l in $(..(S_l)_l \ldots)_l$ is greater than or equal to the ordinal number of k in $(..(S_k)_k..)_k$ (again, because $q_l \leq q_k$). This again implies inequality of the form (*). The lengths of a critical paths in the considered schedules are decreasing step–by–step and the number of such schedules is bounded by the maximal tail (for the details we refer to our proof of Theorem 2). As a result, we are brought either to the situation when the job k, or both l and k become non–emerge (these jobs cannot be further used for a schedule improvement), or to one of the situations considered above while for all intermediate complementary schedules inequalities of the form (*) are satisfied.

Suppose now that the critical path in S_l is shifted backward. Let r' be the overflow job in S_l. Then clearly, $L(S_k)$ cannot be less than $L(S_l) = L(S_l,r')$ (again Lemma 6) and $L(S_k) = L(S_l)$ only if $L(S_k) = L(S_k,r')$. To conclude this case, to any successor of S_k we apply a reasoning applied above for the different behaviour alternatives.

Now the alternative (e) obviously reduces to one of the alternatives (a)-(c) and the lemma is proved.//

Relying on Lemma 8 we can reduce the set of emerge jobs we consider: A subset $K(S)$ of $K'(S)$ we call the *reduced set of emerge jobs* if for any pair of jobs k,l in $K(S)$, such that $\mathrm{ord}(l,S) < \mathrm{ord}(k,S)$ we have $q_l < q_k$.

4. The algorithm.

4.1. The r-restricted complementary schedules.

In this section we study one of the major types of the complementary schedules, the so called *r-restricted complementary schedules* and we give an upper bound on their maximal number in T.

Let $r \in I$. The complementary schedule S is called a *simple* complementary schedule of r if it is the first generated complementary schedule in which r is the overflow job.

A successor of a simple complementary schedule of r S is an r-restricted complementary schedule if job $r = r(S)$ is the overflow job in it (the second r in this definition is not a variable, its a letter).

From Lemma 7 of the previous section we immediately get that we can have at most one instance of alternative (b) for any r. Hence, we have the following

Proposition 3. In an r-restricted complementary schedule we may have only an instance of one of the alternatives (a), (c), (d), (e).

Before we give an upper bound on the number of r-restricted complementary schedules we need to introduce an auxiliary notions.

Let S be the son of S'. We say that job r is *perturbed* in S if $t_r^S > t_r^{S'}$. Assume that the overflow job r is perturbed in S. Then clearly, only an instance of alternative (c) is possible in S and that will occur as a result of rearrangement of some jobs scheduled before r in S, namely, as a result of rescheduling an emerge job after an emerge sequence. So, the emerge sequence of S' precedes r in S.

It is easy to observe that the number of jobs scheduled before r in S should be the same as that in its parent S' if r is perturbed in S: clearly, by the GTH, it cannot be more; if it is less than that in S' then r cannot be perturbed since a new arisen gap in S (see Lemma 1) has a length less than the job processing time.

Now we concentrate our attention on the overflow jobs which are perturbed. Assume S is an r-restricted complementary schedule with the overflow job r perturbed in it, let C be the emerge sequence in the parent of S and let C' be the first former emerge sequence succeeding C in S. Let, further μ be the total number of jobs scheduled between C and C' in S. We have the following lemma:

Proposition 4. In any successor of S the number of jobs scheduled between C and C' can be bounded by $\mu - 1$, that is, only successors of S in which no more than $\mu - 1$ jobs are scheduled between C and C' can be considered.

Proof. Let S^* be any successor of S such that the number of jobs scheduled in it between C and C' is equal or more than μ. The jobs of C might be shifted one (or more) position(s) left in S^* (in comparison with S), or not. Only in the first case the jobs of C' might start earlier in S^* than in S. Consider the two cases separately.

For the second case (i.e., for the case when jobs of C do not start earlier), because of the equality of the processing times, obviously, we will have $L(S^*, r) \geq L(S, r)$ what shows our claim.

Consider the first case. The positions which occupy the jobs of C in S^* cannot start in S^* earlier than in S since the jobs which occupy these positions in S^*

have the readiness times more than or equal to that of jobs which occupied these positions in S (by the GTH). For that reason, the released positions in S^* occupied in S by the (last) jobs of C cannot start in S^* earlier than in S. Hence, there are no positions before C' in S^* which start earlier than in S and so, similarly to the case above, we have $L(S^*, r) \geq L(S, r)$ and the lemma is proved.//

On the basis of the above proposition we restrict the set of the complementary schedules we create. Assume we generate an r-restricted complementary schedule S such that its overflow job is perturbed (in the other words, S is the complementary schedule which overflow job is the overflow job of at least one its predecessor and that job is perturbed in S). We bound the number of jobs which might be scheduled before r in any successor of S declaring the *interlock number* for C, $\iota(C)$. We set $\iota(C) = \mu - 1$ (we keep all the notations from above). Let S^* be a successor of S. We say that the interlock number $\iota(C)$ is *respected* in S^* if exactly $\iota(C)$ jobs are scheduled between C and the next to C emerge sequence C' in that schedule. From now on we consider only the complementary schedules which respect all the interlock numbers. We describe later in this section how we build them.

Let O be the set of the overflow jobs in the algorithm and let $\nu = |O|$. Assume $C_1, C_2, ..., C_\nu$ is the succession of the respective emerge sequences (we may have repetitions in it). Let, further $\pi = \min\{\max_{k=1,2,...,\nu} |C_k|, m\}$.

Lemma 9. The total number of r-restricted complementary schedules in T is bounded by $O(\nu\pi)$.

Proof. We devide the proof into two parts. In part 1 we show our claim under the assumption that no job from O is perturbed in the algorithm. In part 2 we remove this restriction and we show that the bound of part 1 still holds.

Part 1. We have no interaction between the different emerge sequences in the algorithm, i.e., no rearrangement carried out in our complementary schedules will 'disturb' the already 'arranged' parts.

We show that for any S and r the creation of at most π successors of S will be necessary to obtain a successor of S, S^π, with the 'desired' property. Then we indicate that we can close S^π if it is still r-restricted.

Indeed, consider the sequence of r-restricted complementary schedules of S, $(S^1, S^2, ..., S^\pi)$, obtained successively by rescheduling in turn the respective emerge jobs. Let j be a job from $C(S)$ with the minimal readiness time. Then in S^1 job j will occupy the position $\mathrm{ord}(j, S) - 1$ and will be preceded by a gap (by the definition of the complementary schedule and Lemma 1). Analogously, in S^2 job j will occupy the position $\mathrm{ord}(j, S) - 2$ while the next job from $C(S)$ will occupy the position $\mathrm{ord}(j, S) - 1$ and both jobs will be preceded by a gap. Now, similarly, in S^π the first π jobs of $C(S)$ will be preceded by gaps. Then these jobs start at their earliest possible starting times and hence $L(S^\pi) = L(S^\pi, r)$ cannot be further improved (if $\pi \leq m$ this claim is obvious, otherwise, we apply Lemma 3). Therefore S^π can be closed and the bound of the lemma is obvious.

Part 2. Assume S is an r-restricted complementary schedule with the perturbed overflow job r_1. Let in general, r_i be the overflow job corresponding to the emerge

sequence C_i and let C_2 be the emerge sequence in the parent of S (as we already noticed, C_2 precedes C_1 in S).

Assume S^* is the sun of S. No more than $\iota(C_2)$ jobs will be scheduled between C_2 and C_1 in S^*, that is, the number of jobs scheduled before C_1 in S^* is one less then that in S. Clearly, S^* is a complementary schedule. So we have a new gap between C_2 and C_1 in S^* (see Lemma 1). But a gap cannot have a length more than or equal to the job processing time. Consequently, r_1 cannot be perturbed in S^* and in any its successor as a result of rearrangement of jobs scheduled after C_2.

Thus, r_1 and clearly, r_2, can be only perturbed as a result of rearrangement of jobs scheduled before C_2. Assume that one of the jobs r_1 or r_2 is the overflow job in the r-restricted complementary schedule S_1, a successor of S^* and that this job is perturbed in S_1. Analogously, assume C_3 is the emerge sequence of S_1 directly preceding C_2. In S_1^*, the son of S_1 no more than $\iota(C_3)$ jobs are scheduled between C_3 and C_2. Again, in S_1^* and in any its successor neither of the jobs r_1 or r_2 can be perturbed as a result of rearrangement of jobs scheduled after C_3, and the total number of generated schedules with r_1 or r_2 perturbed is 2 (the total number of the overflow jobs which belong to C_1 or C_2).

Similarly, for $k > 2$ we have the succession of the respective emerge sequences $C_k, C_{k-1}, ..., C_1$. The total number of times the overflow jobs from $C_1, C_2, ..., C_k$ are perturbed is k and each new perturbation causes the creation of a single complementary schedule.

Now, relying on the result of Part 1 and applying that $k \leq \nu$ we get the following bound for the total number of r-restricted complementary schedules:

$$k + \overbrace{\pi + \cdots + \pi}^{\nu \text{ times}} \leq \nu(\pi + 1) = O(\nu\pi).$$

The lemma is proved.//

Although the succession $C = C_k, ..., C_1$ discussed in the above proof consists of the separate emerge sequences, they are tighted in the one connected sequence in the following sense: Whenever a job $r_i \in C_i$ is perturbed, we shall respect the interlock numbers of all the succeeding sequences $C_{i-1}, ..., C_2$. The number of jobs scheduled before r_i should be decreased by one. Hence, some job, l, scheduled before C_i should be rescheduled after C_1. Clearly, l should be an emerge job for C_1, that is, we should have $q_l < q_{r_1}$.

From the above discussion we can easily see that for the generation of the complementary schedules which respect the interlock numbers we can use the technique similar to that which we apply for the construction of the complementary schedules. There is a slight modification: Now we have to look for a job l which is emerge not only for C_i but also for C_1 and then we have to modify the readiness time of that job in such a way that it will be rescheduled after C_1 by the GTH. So, keeping the notations from the procedure of section 2, for the threshold value of l we take now the completion time of the last job of C_1 (not C_i) in the schedule S^*.

4.2. The l-restricted complementary schedules.

In the previous section we gave a bound on the total number of r-restricted complementary schedules. These schedules involve the instances of alternatives (a), (c), (d), (e). Now we consider the instances of alternative (b). Let r be the overflow job in $S \in T$ and let the critical path in the complementary schedule S_l ($l \in K(S)$) be rested on l (the tail of l is not 'small enough' to be applied further). Then we look for another emerge job l', with $q_{l'} < q_l$. If the tail of this job again turns out not to be 'small enough' we look for the next emerge job, and we continue similarly.

Let $B = B(S)$ be the critical block in $S \in T$ and suppose we generate a complementary schedule S_l rescheduling $l \in K(S)$ after $C(S)$. Then the block B in S_l may *split*, that is, from one block $B \in S$ we may get two or more new blocks in S_l (in the proof of Lemma 9, Part I we had an example of block splitting when we constructed the complementary schedule with the 'desired' property).

Any block raised as a result of block splitting we call a *secondary block*. Blocks raised as a result of splitting of one particular block we call *relative* blocks.

The special care is necessary to be taken whenever the critical block in a given complementary schedule is secondary. Speaking roughly, a block split "violates" the "natural way" of how we "normally" would treat a complementary schedule and it may cause the "loss" of a "potential" emerge job.

Consider the complementary schedule of stage h, $S(h) = S$ and assume that its critical block $B = B(S)$ is splited into relative secondary blocks B_1 and B_2 in the son of S $S(h+1)$ on stage $h+1$. Let, further B_2 be the critical block of stage h' ($h' > h$). Clearly, we cannot restart earlier any job from our critical block B_2 by rescheduling jobs of B_1 (we may only increase the gap between the two blocks). But a "potential" emerge job for $S(h')$ may belong to the block B_1, this will be the case when some unapplied job from $K(S)$ has the tail "small enough" and hence "theoretically" could be applied in S' (notice that the set $K(S)$ is splited into two parts in $S(h')$: Job l belongs to the block B_2 while all the rest of jobs from $K(S)$ are in B_1.)

So, we may have "potential" emerge jobs which are not "practically available" in $S(h')$. In order to make them available, we made the necessary correction in our current problem instance in such a way that blocks B_1 and B_2 again merge into block B. Let $MERGE(B_1, B_2)$ be the procedure which carries out the above merging.

Assume that we executed the procedure $MERGE(B_1, B_2)$ and assume that the set of emerge jobs in the resulting schedule is empty, that is, there is no "potential" emerge job for $S(h')$ in B_1. We may have two possibilities. First, B is non-secondary; second, B is secondary. In the first case there can exist no "potential" emerge job and we can close the current schedule. Consider the second case and suppose that B', $B' < B$ is relative to B block, directly preceding B. Then we may have a hope that a "potential" emerge job belongs to B'. Hence, we again apply procedure $MERGE(B', B)$. In the case if there are no emerge jobs in the resulting schedule and the resulting block in still secondary we repeat the process.

We terminate it when the resulting block is non-secondary.

An application of a "potential" emerge job might be necessary in two cases which we specify below:

First, assume we have an instance of alternative (b) in a complementary schedule of S S_l, $l \in K(S)$. Where we have to look for the next appropriate emerge job? Clearly, first, in $K(S)$. But, shall we stop trying if there are no more jobs in $K(S)$ which have not being already tried? As we already noticed, not if the critical block of S is secondary. Second, assume we are brought to a schedule with an empty set of emerge jobs . Similarly, we apply the "potential" emerge jobs if the critical block of that schedule is secondary.

Procedure $MERGE(B_1, B_2)$ is accomplished by modifying the current problem instance of stage h'. In order to merge the blocks B_1 and B_2 we restore the readiness time of the last applied emerge job l of $K(S)$ (this job is not "good enough" to be applied further). Let β be the current readiness time of l. We reassign first to l its readiness time, prior to the current one. Then we check if there exists an emerge job in the resulting schedule. If so, we take the first such a job l' (the unapplied emerge job with the greatest ordinal number) and we set $a_{l'} := \beta$. In this way we "activate" l' rescheduling it after all jobs of $C(S)$.

Whenever we have an instance of alternative (b) in a complementary schedule S_l we close it (Lemma 7) and backtrack to the parent S of that schedule and try the next emerge job from $K(S)$. Assume that there are no "untried" emerge jobs left in $K(S)$ (the first case above), or the set of emerge jobs in S is originally empty (the second case above). Let B be the critical block in S. If B is not secondary we know that there can exist no "potential" emerge job for S and we close it. Otherwise, we call the above described procedure. We repeat the process for any of the two above cases until we are brought to the non-secondary critical block. Then we close the current schedule. How many times we have to keep trying? As below Lemma 10 shows no more than p times. Before stating this lemma we give the following definition.

A complementary schedule S_l in which the critical path is rested on l, or such that $K(S_l) = \emptyset$ we call the l-*restricted* complementary schedule if the critical block in it is secondary.

Lemma 10. For each $S \in T$, the total number of l-restricted complementary schedules of S in T is no more than p.

Proof. Let S_l be the l-restricted complementary schedule of S and let j be the job with the ordinal number $\mathrm{ord}(l, S_l) - 1$ in S_l (i.e., the job scheduled strictly before l in S_l). From the GTH we have that $q_j \geq q_l$, but we also have that $q_j \leq q_l - p$ since otherwise the critical path in G_{S_l} would pass through the node j (by the definition of S_l). Now in each of the newly generated l-restricted schedule we have to apply an emerge job with the tail, stricly less than that of the previous one. Then, in at most $(p+1)$st such a generated schedule j will become the overflow job and this proves the lemma.//

4.3. The Formal Description and the Computational Complexity of the Algorithm.

In this section we give the description of our algorithm. The algorithm enumerates the generated complementary schedules in the solution tree T. A complementary schedule is associated with each node of T while the root of T represents the initial complementary schedule.

We number the nodes of T in the order as they are created and with each node we associate a *stage* in the algorithm. A stage characterizes certain state in the algorithm with the already generated set of complementary schedules. At any stage h we construct one complementary schedule $S(h)$ obtained by rescheduling one emerge job l in its parent-schedule. For l we take the emerge job of the parent-schedule with the maximal ordinal number. Although we construct one successor for each $S \in T$ at ones, as we described in section 4.2., for instances of alternative (b) we may backtrack to S and generate its another successor applying the next emerge job of $K(S)$ with the greatest ordinal number.

When we generate the initial complementary schedule S^I, we determine the right-most maximal path, the overflow job $r(S^I)$ and the set of emerge jobs in it. If the latter set is empty we stop (S^I is an optimal solution, see Lemma 2). Otherwise, we mark the overflow job $r(S^I)$ and we generate one successor $(S^I)_l$ of S^I, where l is the emerge job with the maximal ordinal number in S^I. Iteratively, let $S \in T$ be the complementary schedule of stage h and let S_l be the son of S respecting all the interlock numbers (again, l is the emerge job with the maximal ordinal number in S). If the critical block $B(S)$ of S is splited in S_l, we mark the new arisen blocks and we keep the current stage number (section 4.2). We analyze the behaviour alternative in S_l after we determine the overflow job and the set of emerge jobs in it. If the overflow job $r(S_l)$ is marked (i.e., S_l is the r-restricted complementary schedule) and that job is perturbed, we declare the new interlock number and we generate a new complementary schedule respecting new interlock number (section 4.1.).

We close S_l if the critical path in it is rested on l (Lemma 7). If $B(S_l)$ is not marked, we go back to S and generate the complementary schedule $S_{l'}$, applying the next emerge job $l' \in K'(S)$, if such a job exists, if not, we stop. Otherwise (if $B(S_l)$ is marked), we go back to the stage specified for that block and we build the new complementary schedule as it was described in section 4.2.

Regardless of the behaviour alternative in S_l, we stop, if the set of emerge jobs in S_l is empty and the critical block in this schedule is not marked.

In the description below we use some notations. Being at stage h, $LAST(K(S))$ is the last applied emerge job from the set $K(S)$ by that stage, or it is the last job of $K(S)$ (the one with the greatest ordinal number) if no job from $K(S)$ is yet applied by stage h. $NEXT(K(S))$ is the job, next to the job $LAST(K(S))$ in $K(S)$; if there exists no such a job then $NEXT(K(S)) = \emptyset$. Further, $OLD(a_l)$ is the readiness time of job l, prior to its current readiness time. $B(S)$ is the critical block of S. $INTERLOCK(S)$ creates the interlock number for $C(S)$ as decribed

in section 4.1.

ALGORITHM EQD;
BEGIN
PROCEDURE BACKTRACK(h);
BEGIN {backtrack}
$S := S(h)$; $l := NEXT(K(S))$;
IF $l = \emptyset$ THEN STOP
$k := LAST(K(S))$; $a_l := a_k$; $a_k := OLD(a_k)$;
$h := h + 1$;
$S(h) := S_l$=GREATEST TAIL;
RETURN
END; {backtrack}
(0) initial settings
For each block $B \in S$ and each $i \in I$: $TAG(B) := \emptyset$; $TAG(i) := false$;
IF $K(S) = emptyset$ THEN STOP; { S is an optimal solution}
ELSE
BEGIN
$TAG(r(S))$:=true;
$l := LAST(K(S))$ { l is such that ord$(l, S) = \max\{\text{ord}(i, S) \mid i \in K(S)\}$}
MODIFY(l);
$h := 0$;
$S(h) := S_l$ =GREATEST TAIL;
END
(1)
IF ($K(S_l) = \emptyset$ and $B(S_l)$ is not marked) THEN STOP
IF the critical path in S_l is rested on l
THEN BEGIN Close S_l;
IF $B(S_l)$ is marked THEN
BEGIN BACKTRACK($Tag(B(S_l))$); GO TO (1) END
ELSE BEGIN $l := NEXT(K(S))$;
IF $l \neq \emptyset$ THEN
BEGIN
MODIFY(l);
$h := h + 1$;
$S(h) := S_l$=GREATEST TAIL; GO TO *(1)*
END
ELSE STOP
END;
END {rested};
IF $B(S) \in S_l$ is splited into $B_1, ..., B_k$
THEN $TAG(B_i) := h,\ i = 2, ..., k$;
IF $TAG(r(S_l)) = true$ and $r(S_l)$ is perturbed THEN

BEGIN
INTERLOCK(S_l);
MODIFY(l);
$h := h + 1$;
$S(h) := S$
END
END. {eqd}

Theorem 2. The time complexity of the algorithm is $O(\gamma mn \log n)$, where γ can take any of the values q_{max} or n.

Proof. First we show that the number of schedules created in T is no more than $m\gamma$.

This claim is easy for $\gamma = n$. Indeed, the total number of overflow jobs cannot exceed n. We can have up to n simple complementary schedules. For the number of r-restricted complementary schedules we have the bound $O(mn)$, with $\nu = n$ (Lemma 9). Hence, the total number of schedules cannot exceed $O(mn + n) = O(mn)$.

Now assume $\gamma = q_{max}$. In each $S \in T$ an instance of one of the behaviour alternative should occur (Proposition 2). We consider each of them separately.

Assume first that we have consequent instances of alternative (c) in the constructed complementary schedules in T. We generate one complementary schedule at each level in T. We show that the number of levels in T will not exceed q_{max}. Indeed, let in the complementary schedule of the first level S_l the critical path is shifted forward to job j ($j = r(S_l)$) and let $r = r(S)$, where S is the parent-schedule of S_l, i.e., the initial GTS. We claim that $q_j \leq q_r - 1$. Indeed, there can be scheduled no more than $m - 1$ (m is the number of machines) jobs in S_l (different from job r) started at time $t_r^{S_l}$ and having the tail equal to q_r. There can exist no job started in S_l at time $t_r^{S_l}$ or later and having the tail greater than q_r since otherwise a critical path in S would pass through this job. All of the jobs with the tail equal to q_r are scheduled before job r in S_l since r belongs to the rightmost critical path. Thus a critical path cannot be shifted forward to any of these jobs and we get that $q_j \leq q_r - 1$.

Assume in the complementary schedule $S' = (S_l)_{l'}$ of level 2 ($l' \in K(S_l)$) the critical path is shifted forward to job j' ($j' > j$). We use the reasoning similar to the above and get that $q_{j'} \leq q_r - 2$; for the complementary schedule of level 3 we get $q_{j''} \leq q_r - 3$, and so on. Thus the number of created schedules, or, equivalently, the number of emerge jobs will not exceed q_r.

The instances of alternative (d) are treated similarly as the instances of alternative (c): using the analogous reasoning, we show that the total number of levels in T will not be more than $\bar{q} = q_{\max} - q_{\min}$, $q_{\min} = \min\{q_i | i = 1, 2, ..., n\}$ (if a critical path is shifted from job j to job j', $j' < j$ we should have $q_{j'} \geq q_j + 1$; from this inequality we can easily get the above bound).

The instances of alternative (e) we devide into two parts. Suppose the critical path is relocated from block B' to B''; then either $B' < B''$ or $B' > B''$. Clearly, the instances in the first case can be treated similarly as the instances of alternative (c) and the instances in the second case can be treated similarly as the instances of alternative (d).

The instances of alternative (b) cause an additional factor of p (Lemma 10) and the instances of alternative (a) are covered by the bound $O(m)$ from Lemma 9 (see Part I of the proof). Now applying this lemma with $\nu = q_{max}p$ we obtain the overall bound $O(mq_{max}p) = O(mq_{max})$ (the constant factor p can be excluded by deviding all the data in our initial problem instance by p).

For each schedule $S \in T$ we apply the GTH with the time complexity $O(n \log n)$ and spend time $O(n)$ to find an emerge sequence and overflow job in G_S. We also spend time $O(n)$ to find the set of emerge jobs. Then we add in constant time new boundary interval to the current set of boundary intervals.

Altogether, we have the time complexity $O(mq_{max})(O(n \log n) + O(n) + O(n)) = O(mn \log nq_{max})$.

The Theorem is proved.//

5. Concluding Remarks. The algorithm proposed improves the running time of the previously known best algorithms. Algorithms from [8,9], as well as the one from [6], are based on the concept of the forbidden regions. In fact, we showed that we can avoid the construction of the forbidden regions what takes time $O(n^2)$ (without special preprocessing). In the solution tree generated by our algorithm, the number of nodes, or the constructed complementary schedules, is bounded by a polynomial on the maximum tail and the number of machines. In the constructed complementary schedules we obtain the gaps which, in fact, serve the same purpose as the forbidden regions in [6,8,9].

References:

[1] K.R. Baker and Zaw–Sing Su. Sequencing with due dates and early start times to minimize maximum tardiness. *Naval Res. logist. Quart.* 21, 171–177 (1974).

[2] P. Bratley, M. Florian and P. Robillard. On sequencing with earliest start times and due–dates with application to computing bounds for $(n/m/G/F_{max})$ problem. *Naval Res. logist. Quart.* 20, 57–67 (1973).

[3] J. Carlier. Problèmes d'ordonnancement à durées égales. Technical report (1981) Institut de Programmàtion, Université Paris, IV–75012 Paris, France.

[4] J. Carlier. The one–machine sequencing problem. *European J. of Operational Research.* 11, 42–47 (1982).

[5] M.R. Garey, D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP–completeness (Freeman, San Francisco, 1979).

[6] M.R. Garey, D.S. Johnson, B.B. Simons and R.E. Tarjan. Scheduling unit–time tasks with arbitrary release times and deadlines. *SIAM J. Comput.* 10, 256–269 (1981).

[7] G. McMahon and M. Florian. On scheduling with ready times and due dates to minimize maximum lateness. *Operations Research.* 23, 475–482 (1975).

[8] B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM J. Comput.* 12, 294-299 (1983).

[9] B. Simons, M. Warmuth. A fast algorithm for multiprocGTSor scheduling of unit-length jobs. *SIAM J. Comput.* 18, 690-710 (1989).

[10] N. Vakhania. Sequencing with readiness times and tails on parallel machines. *Proc. of the Twelfth ACM symposium on Applied Computing*, 438-446 (1997).