Abstract

Railway networks are faced to an increase demand of new services. This situation leads to train schedules close to the maximum capacity of the infrastructure. As the extension of the infrastructure is too expensive, an alternative solution is to improve traffic management in congested areas. In heavy traffic areas of rail networks, conflicts and subsequent train delays can cause considerable chain reactions during operations. A disturbance can quickly lead to many other delays called secondary delays or knock-on delays. To limit the secondary delays, new methods and models are necessary to optimize the use of scarce resources, such as platforms and track sections. The dispatcher controlling traffic can change train orders and routes to reduce delays caused by conflicts. This task of the dispatcher can be formulated as an optimization problem, where the decision variables are the selection of alternative routes and sequences for trains.

We have developed a constraint-based scheduling formulation of the train routing and scheduling optimization problem. This kind of model uses two complementary approaches: constraint programming and scheduling theory. The model is capable of considering a large number of commercial and technical characteristics drawn from real situations, e.g. passenger and rolling stock connections between trains, signals watching time, number of aspects signalling, sectional route release of interlocking systems, …

Introduction

In heavy traffic areas of rail networks, conflicts and subsequent train delays can cause considerable chain reactions during operations. A disturbance can quickly lead to many other delays called secondary delays or knock-on delays. To limit the secondary delays, new methods and models are necessary to optimize the use of scarce resources, such as platforms and track sections. The dispatcher controlling traffic can change train orders and routes to reduce delays caused by conflicts. This task of the dispatcher can be formulated as an optimization problem, where the decision variables are the selection of alternative routes and sequences for trains.

This optimization problem is a joint scheduling and allocation problem. It is a difficult and very tightly constrained combinatorial problem which makes it hard to solve using exact methods for a reasonable problem size.

We have developed a constraint-based scheduling (CBS) formulation of the train routing and scheduling optimization problem. A CBS model uses two complementary approaches: constraint programming and scheduling theory. The model was mainly applied to two kinds of railway areas: a complex junction [1], a large station [2]. In this paper we consider a line area.

The last updated and detailed description of the model is in [1]. An extension of the model to allow a better management of conflicts between trains running in opposite directions has been presented in
The published experiments with CBS model used a two-phase approach for solving the problem, i.e., the first step is to solve the allocation problem, and then, we will search for a solution to the scheduling problem. In this paper, we present a new heuristic resolution method based on an incremental approach. The algorithm combines the local decisions of allocation track sections and scheduling train runs on these sections. To heuristically choose between making resource allocation decisions or scheduling decisions, the algorithm uses aggregate demand curves. The curves are updated dynamically after each decision. These kinds of measurements are named «texture measurements» in constraint-based scheduling literature [3].

The paper is organized as follows: constraint programming and scheduling theory are briefly introduced. Our CBS formulation of the railway traffic management is recalled in the next section. Then, we present the two-phase and the incremental heuristic resolution methods. The following section shows the results obtained with problem instances from traffic of the Utrecht Den Bosch area of the Dutch railway network. The final section gives some preliminary conclusions.

**Constraint programming**

Constraint programming is a technique used to represent constraint satisfaction problems. In very general terms, a constraint satisfaction problem (CSP) is defined by the following ordered triple $P = (X, D, C)$ such that:

- $X = \{x_1, ..., x_n\}$ is a set of variables,
- $D = \{d_1, ..., d_n\}$ is a set of finite domains where a domain $d_i$ is a set of possible values for the variable $x_i$,
- $C = \{c_1, ..., c_n\}$ is a set of constraints where each constraint limits the combinations of possible values for a subset of variables $X$.

For a given CSP instance, it is possible to look for one solution, all the solutions, or an optimum solution with respect to an objective function. In the last case, a CSP associated with an objective function is a combinatorial optimization problem. Constraint programming (CP) provides the languages and tools which allow us to define a CSP and program resolution algorithms. These tools use a constraint-based process to reduce the search space which reduces the amount of computer processing required to resolve the problem. This process, which is referred to as «constraint propagation», not only validates the values that have been selected for some variables, but, above all, provides a way of removing values from the domains. The removed values can no longer feature in a solution in view of the decisions that have been made. It is this removal of values from the domains which makes it possible to converge towards a solution to the problem or, on the contrary, to demonstrate that there is no solution.

**Scheduling theory**

Scheduling theory defines a formal framework and algorithms whose purpose is to position tasks, which may also be referred to as activities, over time. The performance of the activities requires resources which are subject to utilization or capacity constraints. The temporal variables which describe the performance of activities are connected by constraints of two types: temporal constraints and resource constraints.

The temporal constraints directly connect the temporal variables of activities according to principles which are specific to each application.

The resource constraints are related to the use and sharing of the resources by the activities. The resources are divided into consumable or renewable resources with the latter being either of limited capacity or with limited states. By sharing resources, indirect links between the temporal activity variables are generated by the satisfaction of capacity or state resource constraints.

**Constraint-based scheduling model**

Like Spigel [4], the basic idea of the CBS model of the railway traffic control is that a train passing through a control area is a job. According to scheduling theory, the concept of job is a set of activities linked by a set of precedence constraints. The movement of a train is a sequence of activities. Each activity is an elementary movement of the train through a track circuit. This is illustrated in Figure 1.
We recall that a track circuit is a track clear detection device made by an electrical circuit of which the rails of section form a part [5].

As the train remains on a track circuit until the next one becomes available for running, this limitation is called a «blocking constraint» in scheduling theory. Therefore, our model is similar to that of a blocking job shop scheduling problem [6, 7].

The main constraints of our model will be outlined briefly (there is a more detailed formulation in appendix A and in [1]).

As each track circuit is a resource, the choice of a route for a train is turned into resource assignments for a sequence of activities. A constraint enumerates the combination of tuples of values allowed for the route and track circuit variables.

The track circuits are modeled as unary resources, this leads to the constraint that two activities requiring the resource cannot overlap over time.

Within the duration of an activity, we distinguish the detection phase. For each train, a constraint links the route values with the earliest start and finish time of the detection phase of each activity.

For each activity, a waiting time variable models the time spent when the next resource is not available. This time is added in the expression of the duration of the activity.

The headway constraint between successive trains due to the block signaling system is formulated with a «synchronization constraint». Let us consider a block signaling system with two aspects. In this case, a train enters a block if no train is detected inside. Therefore, to enter a block, all resources of the track circuits inside the block must be available «at the same time». The start of each activity related to one block is synchronized with the start of the detection on the first track circuit of the block.

For the general case of a block system with n aspects, the synchronization is established with the entrance in the first track circuit of the (n-2)th previous block. The figure 2 shows a synchronization constraint within a Gantt chart. Each line denotes the use of a track circuit, a rectangle on a line means that an activity uses the track circuit. As it is a three-aspect signaling system, the use of the track circuits of a block are synchronized with the detection of the train on the first track circuit of the previous block.

To improve the management of an opposite direction conflict, a two-state resource is specified [2]. This state resource will make it possible to model the direction of train movements on this track circuit sequence. If a train has to run through an opposite direction conflict, the activities require the resource to be in a state that corresponds to the direction of run. The state constraint of the resource makes it possible to propagate scheduling decisions on activities of other trains and, consequently, improve the resolution methods.

For train scheduling, the criterion frequently used is the sum of train delays caused by conflicts. This criterion is formulated with the sum of the waiting-time variables.
Heuristic methods

According to the classification of the conventional scheduling problems, the CBS model of the railway traffic control is both a scheduling and allocation problem i.e. there are degrees of freedom for deciding both which activities to perform and when they are to be performed, and which resources to allocate to these activities.

To deal with the combined resource allocation and scheduling problem, we consider two approaches: the first one is a two-phase approach which performs independently resource allocation and scheduling, and the second one is an incremental approach where partial resource allocation and partial scheduling is repeated until a complete solution is generated.

This section gives a presentation of a heuristic method for each approach. These heuristics are compared in the experiments presented in the next section.

Two-phase approach (TPH)

According to the classification of the conventional scheduling problem, the rail traffic operational management problem of section is a joint scheduling and allocation problems i.e. there are degrees of freedom for deciding which activities to perform and when, and which resources to allocate to these activities.

The most widespread approach to solve the problem is a two-phase algorithm. The first step is to solve the allocation problem. After that, we search for a solution to the scheduling problem. The rationale behind this sequence of problem solving is that the feasibility of the scheduling decisions of activities depends on the allocated resources in order to check the resource capacity constraints.

For the experiments presented in this paper, the allocation problem is solved with a complete labeling procedure of the route variables of the trains. This means that all route combinations will be tested. The order of the route assignment decisions is based on the domain size, i.e. the train with few routes will be assigned first.

The scheduling problem is solved by an exact method taken from the Ilog Scheduler library [8, 9]. The algorithm uses the following steps:

1. Select a resource among the resources required by unordered activities.
2. Select the activity to execute first among the unordered activities that require the chosen resource. Post the corresponding precedence constraints. Keep the other activities as alternatives to be tried upon backtracking.
3. Repeat step 2 until all the activities that require the chosen resource have been put in order.
4. Repeat steps 1 to 3 until all the activities that require a common resource have been put in order.

In step 1, the resource selected is the one whose «slack» is minimal. The slack is the difference between the availability (or «supply») of a resource and the demand for the resource over a specific period of time [8]. The slack is equivalent to a measure of the criticality of a resource. In step 2, the activity to schedule first is the activity with the minimal earliest start time and, in case of ties, the activity with the minimal latest start time.

In the first experiments to use this algorithm, we obtained weak time performances. The algorithm
could not find a feasible solution of many instances within the time limit. As in [10], to improve the algorithm, we start with a *greedy algorithm* to find a feasible solution in a short time. The greedy algorithm used applies a well-known dispatching rule: First Come First Served (FCFS). The upper bound of the criterion is set with the solution of the greedy algorithm and we proceed with the scheduling algorithm.

During the search with a *constraints programming model*, constraint propagation (also named *consistency procedure*) consists in reducing the set of possible values for variables. Each branching decision triggers constraint propagation and domain reduction of variables. When the domain of a variable is empty, there is no solution in the sub-tree. In the latter case, a new decision branching from an upper node is tried. A feasible solution is found when a value is found for all decision variables.

With regard to consistency procedures, in addition to the arc-consistency [11], three mechanisms are used to propagate the resource utilization constraints and adjust the time limits of activities [12]:

- The first mechanism relies on an explicit timetable of the variation of resource utilization and resource availability over time.
- The second mechanism is the «disjunctive constraint propagation»; it consists in maintaining arc-B-consistency [13] on temporal constraints.
- The third mechanism known as «edge finding» [14], considers an arbitrary set of activities \( \Omega \). It requires the same resource and determines whether \( A_i \) must take place before or after all activities in \( \Omega \).

These three propagation mechanisms make it possible to prune many non-feasible decisions by adjusting the time limits of activities and thus improve the efficiency of the search. These mechanisms are also called «implication rules» [7].

Backtracking signifies the choice of a new node to explore. This choice is made after finding a feasible solution or proving that there is no solution in the sub-tree of the current node. For all results reported here, we have only used the chronological backtracking: the new node to explore is the last variable labeled; if all domain values of this variable have been tested, backtracking moves to the previous variable.

**Incremental approach (INC)**

The incremental algorithm is inspired from solving techniques for job shop scheduling [3, 8, 15]. The idea is to use dynamic information from the structure of the problem to guide the search. More precisely, to improve the resolution of large scheduling problems, the activities are considered as independent decision points [15]. The search procedure repeatedly make a resource allocation to an activity or an activity schedule. At each step, there is no obligation to schedule the upstream or downstream activities of a job, nor the other activities competing for the same resource. During the search, the scheduling effort must be constantly directed towards the bottleneck activities that appear the most critical. The algorithm must also defines a criticality measurement procedure. This procedure revises the criticality of the unscheduled and unassigned activities whenever the search state changes, i.e., after an activity scheduling or resource allocation decision. Algorithms that implement dynamic analyses of the search space are called «texture measurement» based algorithms [3].

From the point of view of railway traffic management, the incremental algorithm with texture measurements allows us to shift global decisions of route allocation and train schedule respectively to the local decisions of track section allocation and track section movement schedule. By reducing the domains of variables, the constraint propagation procedure will allow us to reduce the chance of inconsistent allocation or inconsistent scheduling decisions.

The INC algorithm is structured as follows:

- Identify a resource \( R \) with the critical time point (i.e. the maximum contention time point).
- Identify two activities, \( A \) and \( B \), which require the resource \( R \) during the critical time point and which are unsequenced. Analyze the consequences of each sequence possibility to determine the best one based on calculations of the temporal associated slack [16].
- Identify an activity \( C \) that has the highest contribution to the critical time point of the resource \( R \) and that still have alternative resources.
- Choose between one of the two following choice points:
  - Constrain the best sequence activity \((A < B \ or \ B > A)\) with the alternative being the opposite sequence.
  - Constrain the activity \( C \) not to execute on the resource \( R \) with the alternative being that it must execute on this resource.
A key element of the algorithm is the calculation of the criticality time point as it implies the choice of the resource R and the activities A, B and C. The criticality measurement is made with texture measurement techniques [17]. These techniques are formed from metrics on the constraint graph representation that underlies a search state. In our case, we use the contention for the resource as a texture measurement technique.

For an activity A requiring a resource R, an individual demand curve is calculated based on the amount of R required by A. To estimate contention, the individual demands of each activity are aggregated for each resource. Aggregation is done by adding up the individual activity curves for that resource. This aggregate demand curve is used as a measure of the contention for the resource over time [3, 8]. Figure 3 shows an example borrowed from [3] to illustrate this calculation of the criticality. In this example, we consider three activities A, B and C with durations 40, 20 and 10. The activities require a unary resource R. The time intervals of the activity demand of A, B and C are respectively [10, 70], [15, 90] and [0, 20]. In this example, we assume a uniform probability distribution over the possible start times of the activities, i.e. each start time value has a probability of $1/\text{STD}$ ($\text{STD} =$ domain of possible values for the start time of an activity). The cumulated curve allows us to define the critical time point for resource R.

**Computational experiments**

This section presents the experiments carried out to evaluate the two heuristics outlined in the previous sections. We used a real case study from the Dutch railway network. The two heuristics were implemented in C++ language with the Ilog Solver 5.3 and Ilog Scheduler 5.2 libraries. The results were obtained on a PC equipped with a 1.66 GHz Intel® Duo CoreTM T2300 processor, 2GB ram and Linux operating system.

**Case study**

The real case study considered is the Utrecht Den Bosch area of the Dutch railway network. This railway dispatching area is around 50 km long and is shown in Fig. 4. The running time is between 3 and 30 minutes depending on the route length. There are 7 passenger stations and one stop for freight trains. The traffic is composed of passenger trains and freight trains.
Challenge F: Even more trains even more on time

Figure 3: Critical time point calculation

Figure 4: The layout of the Utrecht Den Bosch area
Numerical instances

For this experimental study, a set of 18 instance problems with increasing difficulty was considered. The difficulty of the instance problems is related to the number of trains (6 to 39 trains) and conflicts between trains (1 to 31). The instances were generated by selecting train services from the services of a peak hour. We did not consider rolling stock or passenger connection constraints.

Each instance includes a subset of the services of the timetable with some disturbances. As in [7], two kinds of perturbation have been considered: an entrance delay for a train or a blocked track section. The second perturbation leads to change the default route and the alternative routes of the trains whose route run through the blocked track section.

The maximum entrance delay for all instances varies from 1200s to 2280s and the average is around 600s. The blocked track section disturbances result in 8% to 14% of the trains having to use other alternative routes.

The instances considered correspond to instances with 3000 to 14000 variables and 3500 to 13000 constraints.

Evaluation of the heuristic resolution methods

For all the experiments, the limit of the computational effort is set at 180 seconds of CPU time (including set-up and preprocessing of data).

Table 1 compares the two heuristic methods. Columns 5-7 (TPH) report the results of the two-phase resolution method applied to the model. Columns 8-10 (INC) reports the results of the incremental resolution method.

The column 1-10 headings in the table 1 have the following meanings:
- inst.: the instance name
- # trains: the number of trains
- GS: the greedy solution values found,
- # conf.: the number of conflicts of the greedy solution,
- BS: the best solution value found by the method within the time limit,
- GAP %: the percentage improvement over the greedy solution,
- CPU: the CPU time needed to find the best solution.

<table>
<thead>
<tr>
<th>inst.</th>
<th># trains</th>
<th>GS</th>
<th># conf.</th>
<th>BS</th>
<th>GAP %</th>
<th>CPU</th>
<th>BS</th>
<th>GAP %</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>6</td>
<td>122</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>3.06</td>
<td>0</td>
<td>100</td>
<td>6.96</td>
</tr>
<tr>
<td>U2</td>
<td>8</td>
<td>122</td>
<td>2</td>
<td>0</td>
<td>100</td>
<td>3.56</td>
<td>0</td>
<td>100</td>
<td>11.08</td>
</tr>
<tr>
<td>U3</td>
<td>10</td>
<td>556</td>
<td>2</td>
<td>53</td>
<td>90</td>
<td>6.82</td>
<td>53</td>
<td>90</td>
<td>11.82</td>
</tr>
<tr>
<td>U4</td>
<td>12</td>
<td>772</td>
<td>3</td>
<td>53</td>
<td>93</td>
<td>7.91</td>
<td>53</td>
<td>93</td>
<td>12.29</td>
</tr>
<tr>
<td>U5</td>
<td>14</td>
<td>1106</td>
<td>7</td>
<td>562</td>
<td>49</td>
<td>11.02</td>
<td>155</td>
<td>86</td>
<td>17.71</td>
</tr>
<tr>
<td>U6</td>
<td>16</td>
<td>1172</td>
<td>9</td>
<td>601</td>
<td>48</td>
<td>18.63</td>
<td>166</td>
<td>86</td>
<td>18.34</td>
</tr>
<tr>
<td>U7</td>
<td>18</td>
<td>1208</td>
<td>10</td>
<td>601</td>
<td>50</td>
<td>21.41</td>
<td>173</td>
<td>86</td>
<td>19.65</td>
</tr>
<tr>
<td>U8</td>
<td>20</td>
<td>1258</td>
<td>11</td>
<td>737</td>
<td>41</td>
<td>145.71</td>
<td>173</td>
<td>86</td>
<td>28.81</td>
</tr>
<tr>
<td>U9</td>
<td>22</td>
<td>1607</td>
<td>16</td>
<td>1607</td>
<td>0</td>
<td>3.29</td>
<td>322</td>
<td>80</td>
<td>40.30</td>
</tr>
<tr>
<td>U10</td>
<td>24</td>
<td>3080</td>
<td>21</td>
<td>3080</td>
<td>0</td>
<td>3.73</td>
<td>912</td>
<td>70</td>
<td>73.35</td>
</tr>
<tr>
<td>U11</td>
<td>26</td>
<td>3246</td>
<td>23</td>
<td>3246</td>
<td>0</td>
<td>3.71</td>
<td>1754</td>
<td>46</td>
<td>65.57</td>
</tr>
<tr>
<td>U12</td>
<td>28</td>
<td>3281</td>
<td>25</td>
<td>3281</td>
<td>0</td>
<td>3.77</td>
<td>1697</td>
<td>48</td>
<td>74.98</td>
</tr>
<tr>
<td>U13</td>
<td>30</td>
<td>3290</td>
<td>26</td>
<td>3290</td>
<td>0</td>
<td>4.13</td>
<td>1288</td>
<td>61</td>
<td>82.29</td>
</tr>
<tr>
<td>U14</td>
<td>32</td>
<td>3292</td>
<td>27</td>
<td>3292</td>
<td>0</td>
<td>3.93</td>
<td>1289</td>
<td>61</td>
<td>85.85</td>
</tr>
<tr>
<td>U15</td>
<td>34</td>
<td>3335</td>
<td>28</td>
<td>3335</td>
<td>0</td>
<td>4.02</td>
<td>1289</td>
<td>61</td>
<td>94.54</td>
</tr>
<tr>
<td>U16</td>
<td>36</td>
<td>3344</td>
<td>29</td>
<td>3344</td>
<td>0</td>
<td>4.13</td>
<td>1304</td>
<td>63</td>
<td>99.36</td>
</tr>
<tr>
<td>U17</td>
<td>38</td>
<td>3514</td>
<td>31</td>
<td>3514</td>
<td>0</td>
<td>4.22</td>
<td>1304</td>
<td>63</td>
<td>102.27</td>
</tr>
<tr>
<td>U18</td>
<td>39</td>
<td>3546</td>
<td>32</td>
<td>3546</td>
<td>0</td>
<td>4.19</td>
<td>1368</td>
<td>61</td>
<td>104.55</td>
</tr>
</tbody>
</table>

*In seconds

The results indicate that TPH method improves the greedy solution within the time limit only for the
instances U1 to U8.

The best solutions of the two heuristics are reported on the figure 5. This figure shows that the incremental method INC gives better results. It improves the greedy solution for all the instances. The sum of delays found by INC method is on average 50% lower than the one obtained by TPH method and the computational time to find these good routes and schedules for trains is lower than 105 seconds. We can notice that INC has some weaker performances for two instances: U10 and U11. These weak performances can be explained by the train conflicts running in the same direction along a large number of track sections. This kind of conflict introduces many rank and resource allocation decisions which are equivalent and which don’t improve the criteria.

Figure 5: The two-phase method TPH vs the incremental method INC

Conclusion

Constraint-based programming is a problem modeling and solving tool which has proved its value for resolving a large number of combinatorial problems. We have shown how this modeling approach can be used to deal with the train routing and scheduling problems. The model that has been presented is capable of taking into account a large number of technical and commercial characteristics drawn from real situations. Trials with problems of increasing size have shown that good quality solutions can be obtained with processing times which are compatible with the operational constraints.

Acknowledgment

The author wishes to thank ProRail, the infrastructure manager of the Dutch railway tracks, for supplying data used in the experiments.

This work has been partially supported by « le Conseil Régional Nord-Pas de Calais» and the European Union research funds (FEDER).

References

Appendix A: Constraint based formulation of the train dispatching problem

Symbols:

- $\text{dom}(r)$: Alternative routes of a train
- $R$: Route of a train.
- $|R|$: Number of track circuits of $R$.
- $T$: Set of trains considered.
- $n_t$: Number of activities train $t$.
- entranceTime$_t$: Entrance time of train $t$.
- $A_1, \ldots, A_n$: Sequence of $n$ activities of a train run.

Variables:

- $r$: Train route.
- $tc_1, \ldots, tc_n$: Resources required by each activity $A_1, \ldots, A_n$ of a train run.
- $\text{start}(A_i)$: Start time of the activity of index $i$.
- $\text{end}(A_i)$: Completion time of the activity of index $i$.
- $w_i$: Waiting time (i.e. delay) associated with an activity $A_i$.
- $\text{esd}_i^0$: Earliest start time of detection of elementary run $i$ if the entrance time of the train equals zero.
- $\text{efd}_i^0$: Earliest finish time of detection of elementary run $i$ if the entrance time of the train equals zero.

Model formulation:

\[ \min \sum_{i \in T} \text{end}(A_n_i) - (\text{entranceTime}_t + \text{efd}_i^0) \quad (1) \]

The objective function of equation (1) minimizes the sum of train delays caused by conflicts.

subject to:

\[ \text{assignment}(r, tc_1, \ldots, tc_n) \quad (2) \]

Constraint (2) links the alternative routes variable $r$ and the resource variables $tc_i$ of each activity.
Challenge F: Even more trains even more on time

\[(\text{end}(A_i) \leq \text{start}(A_j)) \vee (\text{end}(A_j) \leq \text{start}(A_i))\]  \hspace{1cm} (3)

Equation (3) is the disjunctive constraint of two activities \(A_i, A_j\) requiring the same unary resource that cannot overlap in time, in non-preemptive scheduling.

\[
\text{startDetection}(r, \text{estd}^0_1, \ldots, \text{estd}^0_n)
\]

\[
\text{endDetection}(r, \text{eftd}^0_1, \ldots, \text{eftd}^0_n)
\]

The two constraints (4) and (5) link the route variable \(r\) of a train with respectively the earliest start time and the earliest finish time of detection of elementary runs.

\[
\text{end}(A_i) = \text{entranceTime} + \text{eftd}^0_i + \sum_{j=1}^{i} w_j
\]

Equation (6) links the end variable of an activity \(A_i\) with its earliest finish time of detection and the waiting and acceleration time variables of the previous activities.

\[
\text{syncIndex}(r, s_1, \ldots, s_n)
\]

The constraint (7) links the value of the index variables \(s_i\) to the value of the route variable \(r\) depending of the number of aspects of the block signalling system.

\[
\text{start}(A_i) = \text{end}(A_{i-1}) - (\text{eftd}^0_{i-1} - \text{estd}^0_{i-1})
\]

The constraint (8) establishes the synchronisation of the start variables of activities of one block.