CONSTRAINT PROGRAMMING APPROACH TO TRAIN SCHEDULING ON RAILWAY NETWORK SUPPORTED BY HEURISTICS

Snezana Mladenovic[a], Mirjana Cangalovic[b], Dragana Becejski – Vujaklija[b], Milan Markovic[a]
[a]Faculty of Transport and Traffic Engineering., University. of Belgrade,
Vojvode Stepe 305, Belgrade, Serbia and Montenegro
[b]Faculty of Organizational Science, University. of Belgrade,
Jove Ilica 156, Belgrade, Serbia and Montenegro
snezanam@sf.sf.bg.ac.yu, cangalovic.mirjana@fon.fon.bg.ac.yu,
draganab@fon.fon.bg.ac.yu, m.markovic@sf.sf.bg.ac.yu

Abstract

The paper deals with the problem known in literature as a problem of train or trip scheduling on railway network. Starting from the defined network topology and the time" assigned beforehand, the paper considers a dynamic train rescheduling in respond to disturbances that have occurred. Assuming that the train trips are jobs, which require the elements of infrastructure – resources, it was done by the mapping of the initial problem into a special case of job shop scheduling problem. In order to solve the given problem, a Constraint Programming approach has been used. To improve the time performance of available constraint programming tool and to meet the rescheduling requirements, three classes of heuristics working together in seeking the solution have been proposed, denoted as separation heuristics, bound heuristic and search heuristic. For the purpose of an experimental verification of the proposed heuristics algorithms, the first prototype of train rescheduling software tool has been designed. Experiments with the realized prototype of software tool indicate that this approach is capable of providing the support to operational railway control.

Keywords: Train scheduling; Job shop scheduling; Constraint programming; Heuristics
Topic Area: C6 Network Design, Optimal Routing and Scheduling

1. Introduction

The operational control in railway traffic should amend the current timetable which implementation is underway, taking into consideration complex dynamic interactions among the events, objective function and evaluation of the future situation in the system, and all together in real time. In the practice of traffic systems the dynamic scheduling is accomplished as reactive scheduling mostly. Decentralized methods dispatch jobs when it is necessary, using simple information available at the time of dispatching. This approach is not satisfactory, since it aims to a local solving of the problem. On the other hand, only small size problems can be solved by the methods of the mathematical programming.

The train scheduling problem belongs to a category of NP-hard problems of combinatorial optimization (Bater, 1998), and hence is complex for both modeling and solving. The train scheduling problem considered for a larger fragment of railway network, a longer planning period, and hence the higher number of trains is a part of designing the timetable carried out at the level of tactical planning. The assignment of train rescheduling is that on a smaller fragment of railway network, over a shorter planning period an operational reconstruction of timetable is made, in respond to disturbances those have arisen. The rescheduling may be considered to be a more difficult problem than an initial

scheduling because additional requirements are imposed to it: to find a solution in a given real time; to have a recovered schedule which will deviate from the initial one as little as possible; the solution if not optimal, to be at least "good enough" with respect to the assigned objective function, but also to other performances, etc.

However, only a few published papers deal with train rescheduling in real time. As a matter of fact, the current rescheduling systems test mostly if the solution proposed by the user is feasible one, and they are not doing full schedule regeneration (Chiu et al., 1996). It also can be noted that authors simplify the scheduling problem in two ways: by simplification of the network structure and omitting and/or approximating of constraints which govern the train movement (Kreuger et al., 1997, Oliveira, Smith, 2001). The basic aim of the research is to formulate the most realistic model and develop original heuristic algorithms, which in conjunction with constraint programming mechanisms in a suitable way are able to find a "good enough" solution of the dynamic train scheduling problem within the limited time.

At first, it was undertaken to the exact definition of the network topology, aiming to the minimum abstraction of the real situation. As a case study it was chosen, for its complexity, single-track line scheduling problem. Namely, the train scheduling on a double-track line can be assumed to be a relaxed problem of scheduling on a single-track line, there being no train crossing. Assuming that the train trips are jobs, which require the elements of infrastructure – resources, it was done the mapping of the initial problem into a special case of job shop scheduling problem. Beside the makespan, the most frequent criterion in job shop scheduling problems, the adequate criteria during the train scheduling are certainly those that take into consideration tardiness and different priorities various train categories.

An easy coding of the complex constraints which are concerned for the traffic regulation on railway network made us decide to use constraint programming method in solving the problem. In order to improve the time performance of available constraint programming tool, it is proposed three classes of heuristics, which are interacting among themselves in searching for the solution, and they are titled as bound heuristics, separation heuristics and search heuristics.

For the purpose of an experimental test of the proposed heuristic algorithms, the first prototype of software tool for train rescheduling has been constructed. Implemented software tool has already demonstrated at the level of the first prototype that the approach is usable to the problem of operational control in railway traffic, both in time performance and in solution quality.

The rest of the paper is arranged as follows: Section 2 defines in a concise and exact way the railway network topology. Section 3 – the problem of train scheduling on a single-track network is modeled as a job shop scheduling problem. The fourth, key section, deals with solving of the problem by Constraint Programming approach. After the train scheduling problem definition as a Constraint Satisfaction Optimization problem, a set of constraints, optimization criteria and heuristics for accelerating the arriving at a solution are discussed. In the fifth section the proposed method is evaluated on the selected real examples. The final considerations and possible lines of further research are presented in the last, sixth section.

2. Railway network topology

The railway network characteristics to be described herewith are at the same time general, but also specific with respect to terms used in practice.

The railway network elements are integral parts of lines and stations – facilities, resources; we shall denote them as set R. According to the properties concerning the

possible numbers of simultaneously present trains on a facility, numbers of entry and exist points of the facility and possibility of connection, we can distinguish three disjunctive subsets: block sections – set P, entry-exit facilities – set U and station tracks – set S. At one moment one train only may occupy the block section or entry-exit facility ($\mathcal{Capacity}(r)=1$, if $r \in P \cup U$), while the number of trains on station tracks is determined by station capacity – by number of its tracks ($\mathcal{Capacity}(r)>1$, if $r \in S$). Such a classification is close to the real one, and it is suitable for an exact formulation of constraints concerning the occupying and making available each of the resource classes.

On railway lines with two-way traffic the train movement directions are traditionally designated as **odd** and **even**. In further presentation, we shall assume that the train moves in odd direction if it runs from the entry to the exit points of the facility, or in even direction from the exit to the entry point of the facility. The railway network is built by connecting the exit points of one facility to entry points of other facility. Facilities from the set P have exactly one entry and exit point, respectively; facilities from the set S have mutually equal, and still higher than one, numbers of entry and exit points. The entry and exit points of the facility are provided with signals controlling its occupation. Let function $\mathcal{I}$ add to each facility the number of its entry points, and function $\mathcal{O}$ add to each facility the number of its exit points.

Let $L=\{po, kol, ui\}$ the set of possible facility types, and function $\mathcal{Type}$ adds type to each facility. The ternary relation μ on set L where $(x, y, z) \in \mu$ defines with which facility of x type the facility of y type may be connected through their entry point and with which facility of z type the facility of y type may be connected through the exit point. The network relation μ shall be represented by the set of triples, as:
$\mu = \{(po, po, po), (po, po, ui), (ui, po, po), (po, ui, po), (po, ui, kol), (kol, ui, po), (po, ui, ui),$
$(ui, ui, po), (kol, ui, ui), (ui, ui, kol), (ui, ui, ui), (ui, po, ui), (ui, kol, ui)\}$.

The railway network N can be defined as a directed acyclic graph $N=(R, A)$, where nodes are resources from R, and the arc $(r_p, r_q) \in A$ between resources r_p and r_q means that the exit point of the r_p facility is connected to the entry point of the r_q facility. The network is properly built if for each triple of connected facilities r_p, r_q, r_k where $(r_p, r_q) \in A$ and $(r_q, r_k) \in A$, goes that $(\mathcal{Type}(r_p), \mathcal{Type}(r_q), \mathcal{Type}(r_k)) \in \mu$ and if in each node r_q most of $\mathcal{I}(r_q)$ the arcs flow in, and flow out most of $\mathcal{O}(r_q)$ the arcs.

The stations are modeled as resources of type *kol*, which are, in accordance with relation μ, in connection with the resources of type *ui* through entry and exit points. Open-line sections between stations consist of one or more block sections (resources of type *po*) between which bifurcation points may be found (resources of type *ui*). This makes possible for a number of trains moving in the same direction may be present simultaneously on the open-line section between stations, where the distance between them is real, spatial. The increase of the number of facilities certainly makes the train scheduling problem more complex. The example of a railway network built according to the described rules is presented in Figure 1.

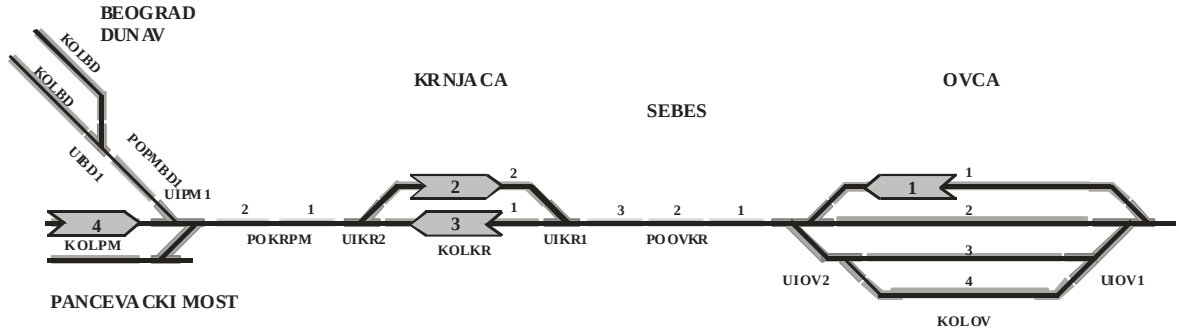

Figure 1. Real example of a railway network

3. Single-track train scheduling problem as a job shop scheduling problem

3.1. Job shop scheduling problem definition

In real service and production systems a need often arises to make decisions concerning the allocation of a limited number of types and limited quantities of different **resources by jobs** over time, along with optimization of some of the objective function. Each job consists of one or more stages requiring time and resources and they are referred to as **activities, operations** or **tasks**. One of the numerous definitions of scheduling is that the scheduling is a subclass of the planning problem focusing its attention on allocating resources over time (Cvetkovic et al., 1996, Pinedo, 1995).

Numerous researches over the past four decades have differentiated different scheduling models taking into consideration different criteria (Jones, Rabelo, 2002, Pinedo, 1995). The scheduling model of interest for presentation in this paper is a dynamic job shop problem. The model is a complex processing system with several resources and several activities, where each job has its inherent sequence of operations and inherent arrival time.

Formally, the job shop scheduling problem of size $n \times m$ in a general case may be formulated in the following way:

Let $J = \{ J_1,...,J_n \}$ be a set of jobs. For each job $J_i \in J$, $i = \overline{1,n}$ we specify a sequence of k_i its operations $J_i = (o_{i1},...,o_{ik_i})$. The operations must be processed in a fixed order, with no interruptions, on one of the resources $r_j \in R = \{ r_1, r_2,...,r_m \}$, $j = \overline{1,m}$. The processing time for each operation o_{ij} is fixed and amounts to p_{ij} and represents the input for the problem as well as a moment of job generation d_i. The earliest possible start time for the operation o_{ij} of job J_i, denoted as d_{ij}, equals the earliest time of completion of the preceding operation on that job and $d_{i1} = d_i$.

The constraints regarding the fixed order of operations for each of the jobs are referred to as **conjunctive constraints**. The constraints related to jobs processing on the same machine are generally called the **disjunctive constraints**. It is possible to seek the solution to a job shop problem under **special constraints.**

Feasible schedule is any one that satisfies all imposed constraints and finds an actual start time $\overline{d}_{ij}$, $\overline{d}_{ij} \geq d_{ij}$ for each operation on each of the jobs.

The most common criterion in the job shop scheduling is the minimization of maximum complete time of all jobs, so-called makespan.

There is a voluminous literature in the field of classical formulation of the job shop scheduling problem (Bierwirth et al., 1995, Pinedo, 1995). Nevertheless, only few articles require the solution of the job shop problem under conditions of special constraints, which is of a particular interest for the contents of this paper.

3.2. Mapping the train scheduling problem into a job shop scheduling problem

The timetable is an entry into the operational railway control. The timetable specifies starting and final points of journey as well as the scheduled arrival and departure times for each intermediate station en route. The timetable is said to schedule trains on a given railway infrastructure.

A real route is a series of all stations through which a train must pass from the origin to destination. This paper interprets the term route in somewhat modified way. The route is **a sequence of facilities** the train must cross on its journey from the origin to destination. A valid route in a network N is any route in shape of r_{l_1}, r_{l_2}, ..., r_{l_e} so that $\mathcal{Type}(r_{l_1})=\mathcal{Type}(r_{l_e})=kol$, (for trains moving in odd direction), i.e. its inversion (for trains moving on even direction).

Instead of the arrival and departure times for each train and each facility on its route, we shall assume that we know the ideal duration of occupation of each facility by train on its route. This occupation includes both the movement and any planned stopping. Since we know the planned time of train arrival to the first facility on the route – **planned train generation**, based on an ideal duration of occupation, we can assume that **the ideal train timetable** is known.

The train movement is a series of particular trips, operations of facilities occupation en route. Hence, each train is accompanied by a trip in a unique way. The train movements also can be considered as jobs to be scheduled to the infrastructure elements – resources. Thus established correspondence entitles us not to make a strict distinction among "train", "trip" and "job" in this paper.

The conflicts among trains arise when a number of requests for resources exceed its capacity, or when some of the imposed constraints have been disturbed regarding the train movement control. In a general case the solving of conflicts requires an introduction of delay into at least one of the conflict trips.

The mapping of the train scheduling problem into a special case of the job shop scheduling problem has been made as follows:

Let $R = P \cup U \cup S = \{r_1, r_2, ..., r_m\}$ be a set of railway infrastructure facilities available, $J = \{J_1, ..., J_n\}$ set of jobs, and N is the railway network. Each train trip $J_i \in J$ is a series of k_i operations $J_i = (o_{i1}, ..., o_{ik_i})$. To each train trip J_i corresponded the "1-1" mapping $\phi_i : \{1, 2, ..., k_i\} \rightarrow R$, which allocates to each operation o_{ij} of this trip a facility $\phi_i(j)$ from R on which this operation is planned to be processed. $\phi_i(1)$, $\phi_i(2)$, ..., $\phi_i(k_i)$ must represent one path on network N (when a *direction*(J_i) is odd) or its inversion, (*direction*(J_i) is even). Each operation o_{ij} has a fixed processing time p_{ij} corresponding to the time of facility occupation $\phi_i(j)$ by the job J_i. The function w joins to each job J_i its priority w_i. Each train is joined its category cat_i, for which we shall assume to specify all train attributes.

The planned start time d_{ij} for each operation o_{ij} equals the earliest possible time, i.e. the earliest completion time of the preceding operation: $d_{ij} = d_{i(j-1)} + p_{i(j-1)}$, $2 \le j \le k_i$ and $d_{i1} = d_i$, where d_i is the planned job generation time of J_i. If c_i denotes the planned, and C_i the actual job completion time, then the tardiness T_i of job J_i is defined as

$T_i = \max(C_i - c_i, 0)$. We shall assume that the planned job completion time is the earliest possible, i.e. $c_i = d_i + \sum_{j=1}^{k_i} p_{ij}$.

The problem of determining the timetable, i.e. train scheduling over time consist of finding the actual start time $\overline{d}_{ij}$, $\overline{d}_{ij} \geq d_{ij}$, for each operation on each of the jobs avoiding conflicts, meeting additional constraints and optimizing the selected objective function. If the ideal timetable is feasible, $\overline{d}_{ij} = d_{ij}$ will apply to all operations on each of the jobs.

If trains in Figure 1. started the movement simultaneously and moved at the same speeds, the trains 1 and 2 would have a conflict on an open line between stations Krnjaca – Ovca, and trains 3 and 4 on an open line between stations Pancevacki most – Krnjaca. Hence, the ideal timetable is not feasible in this particular case. The visualization of this conflict is given in Figure 2 in the form that is known in railway traffic as the train diagram. Actually, the modified Gantt's diagram is in question, where the modification consists of touching resources on y-axis, and bars – rectangles are replaced by their diagonals, which symbolizes the train movement on the resource.

The problem of train scheduling is known in literature as NP-hard (Bater, 1998), i.e. an algorithm for solving this problem is not known, which will be finalized in polynomial time (Cai, Goh,1994).

4. Solving train scheduling problem by constraint programming approach

4.1. Definition of the train scheduling problem as CSO problem

Constraint Programming (CP) is a more recent approach in the field of programming languages, which attempts to reduce the gap between the problem description at a high level and algorithm implemented for its solving. One of possible definitions is for CP to deal with proposing the software architectures for simplifying the implementation of combinatory optimization algorithms. CP attracts the experts' attention in various fields, it being found that many problems in real world can be represented by **C**onstraints, where the **S**atisfaction of these constraints gives a solution for the **P**roblem in question (CSP).

The CP paradigm the focuses on manipulation of variable domains and relations between these variables expressed through constraints. These variables are actually the **decision variables**, but they will be referred to hereinafter as variables, in short.

Formally, CSP is defined (Marriott, Stuckey, 1998) as a triple (V, D, C), where: $V = \{v_1, ..., v_n\}$ is the finite set of variables presenting the problem, D is a function which joins to each variable in V its domain, i.e. $D(v_i) = D_i$, C is the finite set of constrains. Constraint $C_i \in C$ between the variables v_{j_1}, v_{j_2}, ..., $v_{j_k} \in V$, $k \leq n$ is any subset of Descartes' product of these variable domains, i.e. $C_i(v_{j_1}, v_{j_2}, ..., v_{j_k}) \subseteq D_{j_1} \times D_{j_2} ... D_{j_k}$.

The solution of CS problem is such an assignment of values from domain D_i to each of the variables v_i, $i = \overline{1,n}$, which would satisfy all constraints at the same time.

In real application there is an interest of determining the quality of the solution found. It is also sometimes an aim to find the best, optimal solution. The CS problem is therefore expanded by function *f* that joins to each solution a numerical measure of its efficiency – by an objective function. Function *f* is defined as an arithmetical expression over variables in *V*.

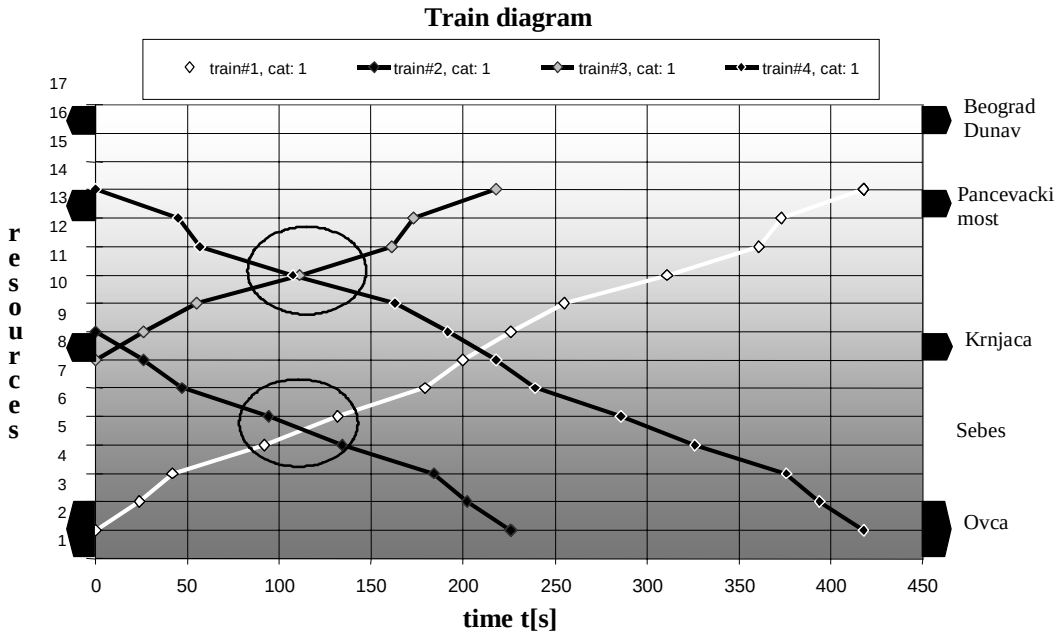

Figure 2. An infeasible timetable with conflicts on open line between stations Krnjaca – Ovca and Pancevacki most – Krnjaca

Formally, **C**onstraint **S**atisfaction **O**ptimization **P**roblem (CSOP) is defined as a quadruple: *(V, D, C, f)*, where V , D and C have been defined earlier, and function f over set S of CS problem solution is: $f : S \rightarrow$ *numeric measure of solution efficiency*. If any upper limit U_0 of function $f(S)$ is known (if the aim of optimization is minimization), then the constraint $f(S) < U_0$ may be added to the set of constraints of CSP problem, i.e. the problem $(V, D, C \cup \{f(S) < U_0\}, f)$ is under consideration. The problem solution $(V, D, C \cup \{f(S) < U_0\}, f)$ gives a new limit U_1, $U_1 < U_o$ for the minimization problem, and now the problem $(V, D, C \cup \{f(S) < U_1\}, f)$ may be solved. Hence, CSOP can be solved incrementally until an optimal solution is reached.

If we consider the start times of activities $\overline{d}_{ij}$ in formulation of job shop problem as decision variables, to which the finite domains are added, and the conjunctive, disjunctive and special constraints as a set of constraints, it is clear that the job shop scheduling problem is one CS problem. Supplemented by an objective function, it grows into a CSO problem. Hence, our initial train scheduling problem may be formulated as a CS problem, i.e. CSO problem.

The support in finding the solution in CP paradigm is offered by consistency methods and search strategies. The consistency methods make the constraint propagation through variable domains. In this way the variable domains are bounded and the search space reduced. The complete search is applied if the aim is to find an optimal solution; if the aim is a solution “good enough” in the limited time, a local search combined with constraint propagation is applied.

It was found long time ago that the problems of planning and scheduling can be presented elegantly in the CS manner, but the easiness of formulating the problem did not necessarily mean an easy solution. The CP approach has become an appealing technology for their solution only after the appearance of commercial CP tools. Within the available

CP tools, the consistency methods and search strategies have been implemented as their inference mechanisms.

The motivation for choosing the CP approach in solving of our problem is as follows:

- declarative nature of constraints in CP approach offers a comfort in formulating the numerous and complex constraints occurring in real train scheduling problem on a single-track railway network;
- the presence of commercial CP tools may significantly shorten the development time and length of the programming code of scheduling applications;
- separation of the constraint component from the search component offers a possibility to keep the constraints once formulated (these actually being regulations for train movements on the railway line which are relatively seldom changed), and to build the search component by considering the concrete objective function;
- possibility of dynamic modification of constraint set by adding new constraints in order to satisfy the current requirements;
- researchers' challenge to test a new approach to solving the train rescheduling problem! Namely, the train rescheduling on a single-track network has been a subject of research for some authors in this field for a number of years. Thus, in (Cicak et al., 2002, Mladenovic et al., 2001) the rules of dispatching the weighted priority and rules of heuristic scheduling have been used, aiming at minimizing the weighted number of late trains. As opposed to dispatching rules permitting quick decision-making, but not taking into account the global information, the CP approach carries out a systematic search and as such, it is time-consuming, but also capable of finding better solutions, in agreement with the assigned objective function.

4.2. Constraint component

In order to solve the train rescheduling problem by CP approach, it is necessary to define the constraint component. The constraint component in our research consists of four classes of constraints.

1. As the train scheduling problem is formulated as job shop scheduling problem, it is clear that the fixed route corresponds to **conjunctive constraints**, and the constraints related to job processing on the same resource, taking into consideration its capacity, are normal **disjunctive constraints**.

2. The following set of constraints is related to preventing trains collisions and it exists on each railway system. In the research that has been carried out, a minimum set of 8 **safety constraints** has been defined, covering all known regulations applicable in real-time train movement on a single-track line on the Serbia and Montenegro railway network. These constraints are designated as: Rule on speeds, Rule on stopping, Rule on occupying and making available of unary resources, Rule on occupying and making available of station tracks, Rule of sequencing, Crossing Rule, Rule of stop-over in a station and Rule of non-simultaneous arrivals to station.

These constraints are the same both for initial scheduling and for rescheduling. All constraints are formulated in meta-notation using previously introduced notation and making possible a simple mapping into an optimization programming language in the implementation phase. For example, we shall consider here in more detail:

Crossing rule

The rule is related to J_p, $J_q \in J$ so that $direction(J_p) \neq direction(J_q)$. The safety regulations specify that at least t_c time must elapse since the moment of making the line available till the moment of exit of the train in opposite direction on the same line. Let

$\phi_p(j_p) = \phi_q(j_q) = r_l$, $\phi_p(j_p+1) = \phi_q(j_q-1) = r_v$ for $2 \le j_p \le k_p - 1$, $2 \le j_q \le k_q - 1$ and $\mathcal{Type}(r_l)$=*kol*. From network relation μ it is clear that $\mathcal{Type}(r_v)$=*ui*. Hence,

$$(\overline{d}_{pj_p} + t_c \le \overline{d}_{q(j_q-1)}) \vee (\overline{d}_{qj_q} + t_c \le \overline{d}_{p(j_p+1)}).$$

3. The third group of rules is **constraints of the rescheduling model**. These constraints differentiate the rescheduling problem from the initial scheduling problem. Our model formulates one of such constraints:

The rule of entering the system without waiting

The modeling assumption that each job J_i must be taken for processing at the moment of generation, can be simply expressed by:

$$\overline{d}_{i1} = d_i.$$

This actually means that the jobs cannot be "piled up" before entering the system. This assumption is extremely reasonable for the case of rescheduling, where the railway network under consideration is only a small fragment of real railway network, where originating and destination stations in the model are in most cases only the intermediate stations in the real system.

4. Finally, a number of **special constraints** have been considered which can be of a practical importance in operational control and which can be included selectively in the constraint component, in relation to requirements put to the rescheduling system. These constraints offer a possibility of planning very specific traffic situations. The possibilities of defining such constraints are inexhaustible, and our model deals with: Rule of simultaneous stop-over in the station, Rule of time distance between the completion of one and generation of another job, Rule of unavailability of resources, Rule of special separation. For example, we shall consider here in detail:

Rule of simultaneous stop-over in the station

The request for every two trains to meet in the previously planned station is unsustainable under conditions of the timetable disturbances, and therefore the rescheduling, overtaking and crossing stations are determined dynamically. However, there is sometimes an interest for two trains to necessarily "meet" in one of the stations of the system. The rule of a simultaneous stop-over in the station should provide for trains J_p, $J_q \in J$ to stop simultaneously in a station at least for t_m time (e.g. due to the planned overtaking, crossing or changing trains by passengers). The station is specified by the resource $\mathcal{Type}(r_l)$=*kol*, $\phi_p(j_p) = \phi_q(j_q) = r_l$, for $1 \le j_p \le k_p$-1, $1 \le j_q \le k_q$-1. Let $t_{stop}(cat_i)$ be be additional time for train stopping J_i. Then,

$$\min(\overline{d}_{p(j_p+1)}, \overline{d}_{q(j_q+1)}) - \max((\overline{d}_{pj_p} + p_{pj_p} + t_{stop}(cat_p)), (\overline{d}_{qj_q} + p_{qj_q} + t_{stop}(cat_q))) \ge t_m.$$

4.3. Optimization criteria

The makespan is the most frequent criterion with the job shop scheduling problems. However, in the train scheduling, of interest are the criteria taking into consideration the delays and different priorities of different train categories. Therefore the following seven relevant optimization criteria, i.e. objective functions have been selected, for which optimization models have been developed:

- minimization of the maximum tardiness $T_{\max} = \max\{T_1, T_2, ..., T_n\}$. Although the criterion minimizes the maximum delay, several trips may suffer disturbances. The criterion is acceptable in situations when passenger trains prevail for scheduling;

➢ minimization of the maximum weighted tardiness $WT_{\max} = \max\{w_1T_1, w_2T_2, ..., w_nT_n\}$ may be an interesting criterion under the mixed traffic conditions. The weights w_i are usually the same for all trains of the same category;

➢ minimization of the total tardiness $D = \sum_{i=1}^{n} T_i$;

➢ minimization of the total weighted tardiness $WD = \sum_{i=1}^{n} w_i T_i$;

➢ minimization of the maximum slack of trains in stations, i.e. the minimization of the function $S_{\max} = \max\{\bar{d}_{i(j+1)} - (\bar{d}_{ij} + p_{ij}) \mid 1 \le i \le n,\ 1 \le j \le k_i ,\ \mathcal{Type}(\phi_i(j)) = kol\}$. Considering that the absolute compliance with the original timetable corresponds to the situation that $S_{\max} = 0$, this criterion offers a support to the idea the rescheduled timetable to be as close as possible to the original;

➢ minimization of maximum complete time of all jobs (makespan) $C_{max} = max\{C_1, ..., C_n\}$ expresses our wish for the trains to leave as soon as possible the fragment of railway network under consideration. In case of rescheduling, this objective function gives support to the localization of disturbances;

➢ minimization of the number of late jobs $|WJ|$, where $WJ = \{J_i \in J \mid C_i - c_i > 0\}$.

4.4. The choice of strategy, policy and method of rescheduling

Following the ideas presented in (Vieira et al., 2003), the essential steps in implementation of rescheduling are the choice of **factors, strategy, policy and method of rescheduling.**

The rescheduling is activated after recognition of the rescheduling factor. The rescheduling factor – the disturbance, in our case study is an unplanned forwarding of train to the station that is equipped with the rescheduling system.

The strategy is necessarily predictive-reactive since there is an original timetable.

The choice of policy depends on the assessment of the minimum time spacing between the consecutive rescheduling factors and the expected run time for the rescheduling procedure. In a general case, the policy may be periodic, event-driven and hybrid. This paper presents a hybrid policy.

So far as the choice of method is concerned, it is clear that due to time limit for rescheduling implementation, one should focus on partial rescheduling methods.

The global rescheduling procedure is represented by pseudocode as follows. In effect, an infinite loop is in question: A disturbance is identified in the first part of the loop body, and its processing is made in the second part.

```
procedure Reschedule(DB, MB)
inputs   DB,    database
         MB,    base of scheduling models
forever
  disturbance←false
  repeat
    Trigger (DB, disturbance, ActiveJobs, J, t_g)
  until disturbance ∧ t = k·period, k ∈ Z
  SelectModel (ActiveJobs, MB, DB, Model, MaxB)
  SelectPreparationModel (MB, Model,PreparationModel)
  Prepare (J,PreparationModel, Delays)
```

SeparateAndScheduleRelatedJobs (*J*,ActiveJobs, Model, Delays, Schedule)
SaveSchedule (DB, Schedule)
PresentSchedule (Schedule)
end forever
end procedure

The imperative clauses designate the procedures and functions calls, where for some of them, due to the level of abstraction, the code may be omitted. All symbols in this one, but also in the following algorithms are of a mnemonic character, and therefore the comment is missing in a number of places.

The assumption is that the database DB comprises the initial schedule and network topology, as well as the updated dynamic data concerning the schedule implementation. The occurrence of an unexpected dynamic data in database DB triggers a rescheduling procedure. The optimization models available are incorporated in the modelbase MB. This offers a support to the idea, on the basis of the user's wish, the period of the day when the rescheduling is made, statistical analysis of the system history, etc., to choose a model which optimizes one or the other optimization function. The procedure ***SelectModel***, in accordance with a certain criterion, selects a scheduling model from the modelbase MB. The procedure ***SelectPreparationModel*** selects a preparation model corresponding to the chosen scheduling model.

Procedure ***Trigger*** verifies the contents of database DB, waiting for information on disturbance. If a piece of information on unplanned dispatching of one or more trains, the procedure returns a set of all jobs the processing of which is underway at moment t - set ActiveJobs, as well as the set of all jobs J, including the jobs from the set ActiveJobs, but also all not commenced jobs the generation of which is planned up to the limit of the planning period t_g, as well as the bound of the panning period itself t_g. It is clear that the planning period $[t, t_g]$ multiple exceed the maximum flow time of jobs. Let $J_i \in$ ActiveJobs be job dispatched to the station represented by resource r_l, $\mathcal{T}ype(r_l)=kol$ and let *j*-th operation of J_i job be performed on that resource, i.e. $\phi_i(j) = r_l$. The rescheduling of the remaining part of the job J_i will be carried out starting from operation o_{ij}, and hence we can assume that the pos_i position from which the job J_i scheduling starts is $pos_i = j$. The expected train arrival to that station is a moment of actual generation $\bar{d}_i$ of the rest of the job J_i. For jobs $J_i \in J \setminus$ ActiveJobs $pos_i = 1$, and d_i is the planned job generation subject to the initial schedule.

The procedure ***Prepare*** will be described in the part of the paper related to bound heuristics, and procedure ***SeparateAndScheduleRelatedJobs*** in the part describing separation heuristics.

The procedure ***SaveSchedule*** accommodates the recovered schedule in database, and procedure ***PresentSchedule*** visualizes the schedule in an adequate way.

4.5. Heuristics

Manufacturers of commercial CP tools claim that it is precise enough to formulate what the problem is (the constraint component and objective function), and CP tools are capable of finding an optimal solution or a series of feasible solutions thanks to inference incorporated algorithms. Experiments made with ILOG Solver CP tool and its extension for scheduling purposes ILOG Scheduler, prove unreliability of exclusive reliance on CP

tools and their search algorithms! The process of arriving even up to the first solution is sometimes very time-consuming and as such cannot meet the rescheduling requirements. Wishing to take advantage of good properties of CP approach (declarative nature of the constraint, separation between constraint and search components, possibility of constraint dynamic propagation, presence of CP tools) which have already been discussed, an idea naturally arose to "support" the CP tools by heuristics based on knowledge of the real problem. For this purposes three classes of heuristics have been formulated, marked as bound heuristics, separation heuristics and search heuristics.

Bound heuristics

The aim of the bound heuristics is to limit the domains of decision variables and objective function in order to increase the search efficiency. Three types of bounds are proposed:

1. Initial bound – all variables take value from interval `[origin, horizon]`, where the origin and horizon are assessed on the basis of the knowledge of the real problem:

`horizon=origin` $+ \sum_{J_i \in J} (\sum_{o_{ij} \in J_i} p_{ij} + t_{stop}(cat_i) + t_{start}(cat_i))$,

where `origin` $= \min\{d_i \mid J_i \in J\}$, i. e.

`horizon=origin` $+ \sum_{J_i \in J} ((c_i - d_i) + t_{stop}(cat_i) + t_{start}(cat_i))$,

i.e. `horizon` is determined by sum of duration of all operations, increased by additional times for stopping $t_{stop}(cat_i)$ and starting $t_{start}(cat_i)$ for every train $J_i \in J$.

2. Lower bound of objective function - is estimated by a special procedure. The estimate is based on solving the preparation model that solves the conflict between two jobs in isolation, disregarding the consequences it might have on other jobs. The aim of procedure ***Prepare*** is to find the minimum delay to incorporate in a pair of jobs, if such pair of jobs is observed in isolation. The element `Delays[i,k]` of matrix `Delays` is an optimal value of the objective function in solving the conflict between jobs J_i and J_k, using a preparation scheduling model. Algorithm of procedure ***Prepare*** has the following form:

procedure ***Prepare***(J, `PreparationModel, Delays`)
inputs J, set of all jobs, includes jobs the processing of which is underway at moment t and non-commenced jobs up to upper bound of planning period t_g
`PreparationModel` scheduling model
returns `Delay,` matrix containing minimum delay if a conflict between two jobs is solved in isolation

forall $J_i \in J$
  forall ($J_k \in J : i<k$)
    if not($(c_i < d_k) \vee (c_k < d_i)$ **) then**
      `origin` $\leftarrow$ **min(** d_i, d_k **)**
      `horizon` $\leftarrow$ `origin`**+(** $c_i - d_i$ **)+** $t_{stop}(cat_i) + t_{start}(cat_i)$ **+(** $c_k - d_k$ **)+**
        $t_{stop}(cat_k) + t_{start}(cat_k)$
      `Jobs` $\leftarrow \{J_i, J_k\}$
      `LowerBound` $\leftarrow 0$
      // initial upper bound for objective function
      `UpperBound` $\leftarrow$ `MaxB`

// running of the model solving the conflict between two jobs
SolveModel (`Model, Jobs, origin, horizon, LoverBound, UpperBound, Schedule, ObjectiveFunction, Makespan`)
// optimal value of objective function is stored in `Delays` matrix
`Delays[i,k]`←`ObjectiveFunction`
end if
end forall
end forall
end procedure

A set of inter-related jobs will be referred to as `RelatedJobs`. Function ***Estimate***, called on every time before solving the scheduling model over the set `RelatedJobs`, on the basis of `Delays` matrix sets the lower bounds for the objective function. For objective functions which are not of a summary type, except for the number of late trains the following shall apply:

$$\texttt{LowerBound} = \max_{J_i, J_j \in RelatedJobs} \{\texttt{Delays[i,j]} \mid \; j > i\},$$

and for summary type objective functions:

$$\texttt{LowerBound} = \mathit{sum}_{J_i, J_j \in RelatedJobs} \{\texttt{Delays[i,j]} \mid \; j > i\}.$$

The lower bound of the number of late trains is a number of non-zero rows in `Delays` matrix.

Since the `PreparationModel` solves the conflict between a pair of jobs in isolation, disregarding the consequences this might have on other jobs, it is very probable that such solution of a conflict situation include a conflict between jobs that have not had it initially. It is also less probable that the solution of one conflict will necessarily resolve some other initial conflicts. Therefore, such heuristic for estimation of the lower bound of objective function in most cases will help in avoiding the unfruitful ways of search, and only in a negligibly small number of cases we will give up very good solutions.

3. Upper bound of objective function – – is dynamically bound during the search. For example, if the objective is to minimize total tardiness, then, after finding a feasible solution with total delay D, we add to the model a constraint:

$$\sum_{J_i \in J} (\bar{d}_{ik_i} + p_{ik_i} - c_i) \le D\,.$$

The propagation of this constraint reduces the domains of decision variables. In other words, the propagation discards the branches on the search tree which would not lead to better solutions than those already found.

Separation heuristics

The aim is to separate and schedule at the same time only those activities that can influence one another, in order that the system may respond faster to the recognized rescheduling factor.

The set of jobs that must be scheduled jointly is already denoted as `RelatedJobs`. The procedure ***SeparateAndScheduleRelatedJobs*** initially allocates to the set of related jobs a set `ActiveJobs` (the jobs the processing of which is underway at the moment of disturbance). The procedure ***SolveModel*** solves the scheduling model. The set `AdditionalRelatedJobs` includes the jobs which due to cascade effects among operations must be added to the set `RelatedJobs`. The procedure ***ExcludeActivities*** has an assignment to recognize in the set `RelatedJobs` those jobs, i.e. their activities that may be considered definitely scheduled and exclude them from further scheduling. The algorithm stops when the set `RelatedJobs` remains empty. This happens in two

situations: when all jobs up to the upper bound of the planning period are scheduled, or if a significant time division between jobs that suffer disturbances and the remaining jobs occurred.

procedure ***SeparateAndScheduleRelatedJobs*** **(** J, ActiveJobs, Model, Delays, Schedule)

inputs J, set of all jobs includes jobs the processing of which is underway at the moment of respond to disturbance t and expected non-commenced jobs up to the planning period upper bound t_g

ActiveJobs, the jobs the processing of which is underway at the moment of respond to disturbance

Model, selected scheduling model

Delays, matrix of minimum tardiness for solving the conflicts

returns Schedule, the schedule of all activities directly or indirectly affected by activities pertaining to jobs from the set ActiveJobs

RelatedJobs ← ActiveJobs

repeat

origin ← $\min\{d_i \mid J_i \in$ RelatedJobs $\}$

horizon ← origin + $\sum_{J_i \in RelatedJobs}((c_i - d_i) + t_{stop}(cat_i) + t_{start}(cat_i))$

Estimate (Delays, Model, LowerBound)

UpperBound=MaxB

// solving the model

SolveModel (Model, RelatedJobs, origin, horizon, LowerBound, UpperBound, Schedule, ObjectiveFunction, Makespan)

MinGen ← t_g

// identification of additional jobs to be generated before completion maximum of
// related jobs and determine minimum moments of such jobs generation

AdditionalRelatedJobs ← {}

forall ($J_i \in J : J_i \notin$ RelatedJobs)

if $d_i <$ Makespan **then**

AdditionalRelatedJobs ← $\{J_i\} \cup$ AdditionalRelatedJobs

if $d_i <$ MinGen **then**

MinGen ← d_i

end if

end if

end forall

// identification of the last station which the train entered not later than the minimum of
// moments generation of additional jobs MinGen

forall ($J_i \in J : J_i \in$ RelatedJobs)

// the moment of train entering the last station MinGen

// is a candidate for new generation time of the rest of the J_i ...

$d_i^n \leftarrow \max_j\{\bar{d}_{ij} \mid \mathcal{Type}(\phi_i(j)) = kol \wedge \bar{d}_{ij} \le$ MinGen$\}$

//... the number of operation performed in the last station before MinGen

// is a candidate for a new position of job generation J_i

$poz_i^n \leftarrow \max\{ j \mid \mathcal{T}ype(\phi_i(j)) = kol \wedge \bar{d}_{ij} \le$ MinGen$\}$

end forall

// some activities pertaining to jobs from the RelatedJobs are
// definitely scheduled, since they cannot be affected by AdditionalRelatedJobs

StorePartialSchedule (Schedule, RelatedJobs)

ExcludeActivities (RelatedJobs, AdditionalRelatedJobs, MinGen)

RelatedJobs $\leftarrow$ RelatedJobs $\cup$ AdditionalRelatedJobs

// the algorithm stops when the set RelatedJobs becomes empty

until RelatedJobs ={}

end procedure

Search heuristics

The aim is to allocate as early as possible start times to activities so that all imposed constraints are satisfied, i.e. find a feasible solution.

The procedure ***SetStartTimesOfActivities*** setting start times of activities find feasible solution. The order of variables is in accordance with increasing lower bound of their domains, as well as the order of values within the domain. As soon as an activity is assigned a start time, the domains (intervals) are updated corresponding to start times of unscheduled activities. At the moment one of the domains remains empty, by backtracking the algorithm tries to find a node in search where a wrong decision has been taken. The algorithm stops when all activities have obtained the start times (the found feasible solution), or if there is no alternative after an error (no solution).

procedure ***SetStartTimesOfActivities*** (Jobs)

inputs Jobs, set of jobs to be scheduled starting from position pos_i

returns $\{\bar{d}_{ij} \mid J_i \in$ Jobs, $pos_i \le j \le k_i\}$,
set of start times for all activities pertaining to jobs starting from their actual position pos_i

begin

Activities $\leftarrow \{o_{ij} \mid J_i \in$ Jobs, $pos_i \le j \le k_i\}$

PostponedActivities $\leftarrow$ {}

ScheduledActivities $\leftarrow$ {}

$d_0 \leftarrow \min\{d_{ij} \mid o_{ij} \in$ Activities $\}$

repeat

SimultaneousActivities $\leftarrow \{o_{ij} \mid o_{ij} \in$ Activities $\wedge d_{ij} = d_0\}$

k $\leftarrow$ ***Cardinality***(SimultaneousActivities)

// creating array of all subsets of SimultaneousActivities without empty
// set, arranged by non-decreasing cardinality

CreateArrayOfSubsets (SimultaneousActivities, ArrayOfSubsets)

n $\leftarrow 2^k - 1$

ok $\leftarrow$ false

repeat

PostponedActivities $\leftarrow$ {}

// test if the set of operations incorporated in ArrayOfSubsets[n] can
// start at moment d_0

if ***CanStart*** (ArrayOfSubsets[n], d_0) **then**

PostponedActivities $\leftarrow$ SimultaneousActivities \

```
                                        ArrayOfSubsets[n]
        UpdateStartTimes (PostponedActivities)
        // a new minimum start time for remaining activities
        d_a ← min{d_ij | o_ij ∈ Activities\SimultaneousActivities}
        d_p ← max{d_ij | o_ij ∈ PostponedActivities }
        if not (d_p < d_a) then
          ScheduledActivities ← ScheduledActivities
                                        ∪ArrayOfSubsets[n]
          ok←true
        else
          n←n-1
          // backtracking to select another set of activities to start at moment d_0
          Backtrack
        end if
      end if
    until ok
    d_0 ← d_a
  until ScheduledActivities = Activities
end procedure
```

The procedure ***SolveModel*** calls on procedure ***SetStartTimesOfActivities***, and thereafter, on the basis of a found feasible solution a new upper bound for objective function is set, and propagation of this constraint reduces the domains of other decision variables. The procedure ***SolveModel*** stops further search if the upper bound of the objective function matches up with the lower bound or if rescheduling of start times of activities does not lead to a better value of objective function.

Such iterative process forms an optimum partial schedule, and if the preceding heuristics were successful, such partial schedules form a "good enough schedule" within the limited time.

5. Method evaluation

Validation of heuristic algorithms is in a general case very complicated. One of the ways is their experimental verification. For this purpose, the first prototype of software tool for train rescheduling has been designed and implemented.

CP tool ILOG Solver and its upgrade for scheduling purpose ILOG Scheduler, manufactured by French company ILOG (http://www.ilog.com), have been used in implementation. ILOG Scheduler permits to create models in the terms of resources, activities and time constraints. The optimization models in our case are formed in OPL modeling language, while the combination and control with optimization models has been achieved by using a procedural language OPL Script. The integrated development environment OPL Studio enabled us to create and modify the models using OPL, to combine and manage the models using the language OPL Script, and to run the models by ILOG Solver and ILOG Scheduler. The trial version of OPL Studio is available on Internet (http://www.ilog.com/download/opl) and this has been actually used for implementation of the first prototype of train scheduling system.

Figure 3. presents a realized window of user interface of the first prototype of train scheduling system – a graph presentation of the recovered train schedule for infeasible schedule shown in Figure 2.

Experiments have been carried out on a fragment of real railway network (a part of Belgrade Railway Junction), with actual train categories operating there, but with traffic frequency immensely exceeding the real one. The jobs (trains) are “piled up” on purpose to test the endurance of the method. The train categories were joined by priorities assessed by expertise. All seven relevant objective functions participated in the experiment, as discussed in 4.3. Table 1. presents a yield of heuristic algorithms on selected examples that differ by numbers of jobs for rescheduling and initial numbers of conflicts. Each set of jobs suffering disturbances includes trains of different categories and different movement directions. All experiments have been implemented on personal computer Intel (R) Pentium(R) 4 CPU, 2GHz. From the analysis of experiment results the following conclusion may be drawn:

- CPU time of schedule recovery depends on the number of activities and number of conflicts;
- solving of initial conflicts may bring up additional conflicts;
- in most cases the time performance is satisfactory;
- a heuristic nature of the approach has been demonstrated (in an insignificant number of cases the best known solution for the given objective function has not been found).

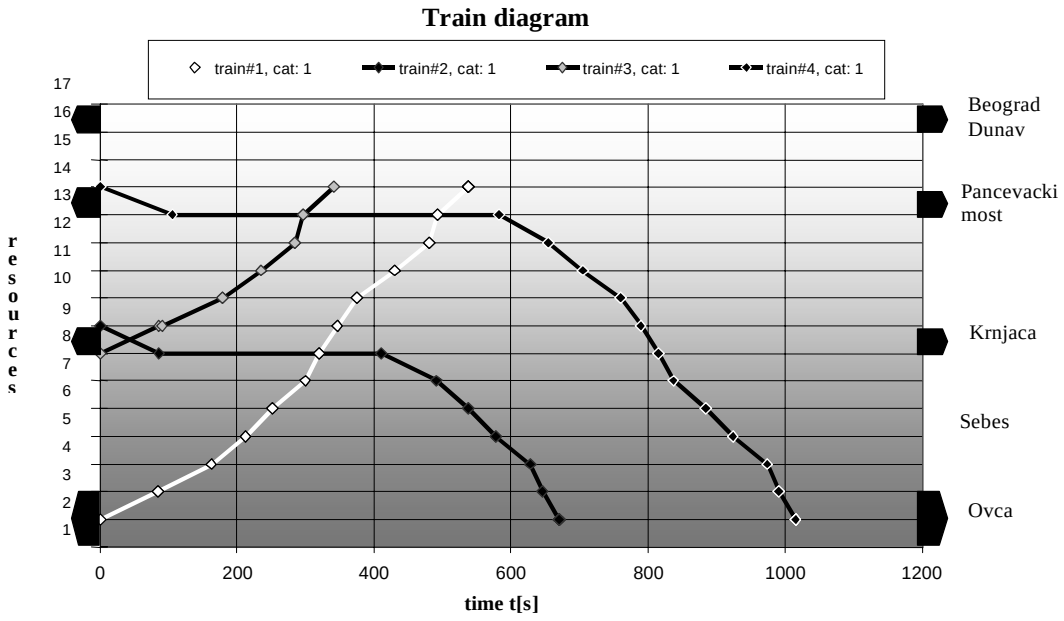

Figure 3. Recovered timetable for infeasible timetable in Figure 2. if the objective is the minimization of total tardiness D

Table 1. A yield of heuristic algorithms in the schedule recovery – selected real examples

No. of example	Initial No. of conflicts	No. of jobs involved in the disturbance	Schedule performances								
			No. of conflicts solved	CPU time	T_{max}	WT_{max}	D	WD	S_{max}	make-span	$\|WJ\|$
1.	1	2	1	3.84	250	500	370	980	250	2634	2
			1	3.38	250	500	370	980	250	2634	2
			1	4.09	250	500	370	980	250	2634	2
			1	4.24	250	500	370	980	250	2634	2
			1	20.44	250	500	370	980	223	2634	2
			1	3.72	250	500	370	980	250	2634	2
			1	3.30	440	1760	440	1760	320	2766	1
2.	2	3	2	9.54	450	900	809	1858	450	6681	3
			2	9.15	450	900	809	1858	450	6681	3
			2	9.84	576	2304	884	2920	456	6499	3
			2	9.32	484	968	913	2066	309	6751	3
			2	8.97	484	968	913	2066	309	6751	3
			2	9.17	576	2304	884	2920	456	6499	3
			2	7.59	992	1984	1112	2464	992	7156	2
3.	3	3	3	10.74	641	2564	1064	3015	521	2067	3
			3	10.18	794	1632	1230	2482	428	2193	3
			3	9.69	641	2564	1064	3015	521	2067	3
			3	10.26	794	1632	1230	2482	428	2193	3
			3	10.90	794	1632	1230	2482	428	2193	3
			3	9.88	641	2564	1064	3015	521	2067	3
			5	8.62	641	2564	1064	3015	521	2067	3
4.	3	5	3	12.59	532	1064	1237	2296	532	7330	5
			3	13.29	532	1064	1237	2296	532	7330	5
			3	16.83	532	1064	1237	2296	532	7330	5
			3	17.83	532	1064	1237	2296	532	7330	5
			3	28.64	532	1064	1478	2537	363	7330	5
			3	14.67	854	1708	1331	2484	854	7102	4
			5	21.59	1904	3808	3556	7112	1652	7816	2
5.	5	5	6	25.77	918	3672	2463	7578	392	2854	4
			6	23.48	920	3024	2580	7536	578	2878	4
			6	27.78	918	3672	2457	6990	392	2754	4
			6	28.24	918	3672	2457	6990	392	2754	4
			6	24.33	918	3672	2463	7578	392	2854	4
			6	30.73	918	3672	2457	6990	392	2754	4
			6	24.12	918	3672	2463	7578	392	2854	4
6.	6	7	8	24.54	622	1244	2727	5351	619	6867	7
			8	18.82	622	1244	2727	5351	619	6867	7
			8	32.36	673	1346	2484	4677	673	7055	6
			8	32.13	673	1346	2484	4677	673	7055	6
			8	53.99	622	1244	2727	5351	619	6867	7
			8	58.88	622	1244	2727	5351	619	6867	7
			8	46.55	673	1346	2484	4677	673	7055	6

6. Conclusions

The paper presents a very realistic railway transport model. Namely, actual line-side signals that limit resources have been taken into account. These signals control if a train may proceed its trip on a particular resource. In the literature available (Kreuger et al., 1997) the authors manipulate with approximate time spacing and not with real spatial spacing of trains, which may be notably different if there is a significant difference in length of resources and in movement speeds of trains. Also, as opposed to (Oliveira, Smith, 2001) the traffic mixture has been taken into account, so that different train categories have been allocated different priorities.

The train rescheduling problem has been formulated and solved as Constraint Satisfaction Optimization Problem. As adequate optimization criteria those taking into consideration tardiness and established priorities between different train categories (e.g. the maximum tardiness, total tardiness, maximum weighted tardiness, etc.). In order to improve the time performance of available constraint programming tool ILOG Solver and to meet the rescheduling requirements, three classes of heuristics working together on seeking the solution have been proposed. They are referred to as separation heuristic, bound heuristic and search heuristic.

For the purpose of an experimental verification of proposed heuristic algorithms, the first prototype of software tool for train rescheduling has been designed and implemented. The experiments carried out on a fragment of real railway network (a part of the Belgrade Railway Junction), with real train categories in operation in that junction, with real possible disturbances, have proven a validity of the approach to problems of operational railway traffic control.

Although the main objective of the research is an operational reconstruction of the timetable under the conditions of disturbances, the described approach has a high degree of universality within the given problem category. By a relaxation of strict time limits and an increased size of the problem, the method is capable of solving the problems of initial train scheduling on a real network up to optimization within reasonable time.

Also, by solving a difficult problem of train scheduling, we have paved the way for solving a whole series of problems, the core of which is the train scheduling, e.g. timetable preparation, determining of economically acceptable capacity utilization interval, the estimate of stopping and waiting of trains for traffic reasons, projection of investment activities results, identification of bottlenecks in infrastructure, choice of a possible solution of a conflict point, research on allocation of block sections and line-side signals etc.

The research presented in this paper should not be considered as a closed system. It might be of interest to further upgrade the network model, a constraint component, and special constraints, in particular, as well as the described heuristic algorithms. One could also make experiments with a distribution of search procedure on several processors, provided a parallel CP tool is available.

References

Bater, W. M., 1998. Computer aided railway engineering, In Computers in Railways VI, Mellit, B., Hill R. J., Allan J., Sciutto, G., Brebbia, C. A. (Eds.), WIT press - Computational Mechanics Publications, Comreco Rail Ltd York, England, pp. 199-211.

Bierwirth, C., Kopfer, H., Mattfeld, D. C., Rixen, I., 1995. Genetic Algorithm based scheduling in dynamic manufacturing systems, In Proceedings of Second IEEE International Conference on Evolutionary Computation, IEEE Press, pp. 439-443.

Cai, X., Goh, C. J., 1994. A fast heuristic for the train scheduling problem, Computers and Operations Research 21(5) 499-510.

Chiu, C. K., Chou, C. M., Lee, J. H. M., Leung, H. F., Leung, Y. W., 1996. A constraint-based interactive train rescheduling tool, In Proceedings of Second International Conference on Principles and Practice of Constraint Programming, pp. 104-118.

Cvetkovic, D., Cangalovic, M., Dugosija, Dj., Kovacevic-Vujcic, V., Simic, S., Vuleta J., 1996. Kombinatorna optimizacija – matematicka teorija i algoritmi, Drustvo operacionih istrazivaca Jugoslavije, Beograd.

Cicak, M., Veskovic, S., Mladenovic, S., 2002. Modeli za utvrdjivanje kapaciteta zeleznice, Saobracajni fakultet i Zelnid, Beograd.

Jones, A., Rabelo, L. C., 2002. Survey of job shop scheduling techniques, Technical Paper, (Downloadable from website http://www.mel.nist.gov/msidlibrary/doc/luis.pdf).

Kreuger, P., Carlsson, M., Olsson, J., Sjoland, T., Astrom, E., 1997. Trip scheduling on single track networks – the TUFF train scheduler, Workshop on Industrial Constraint Directed Scheduling, pp.1-12, (Downloadable from website `http://citeseer.nj`. nec.com/kreuger97trip.html).

Marriott, K., Stuckey, P. J., 1998. Programming with Constraints: An Introduction, The Massachusetts Institute of technology Press, Cambridge.

Mladenovic, S., Veskovic, S., Cicak, M., 2001. SIZES programski sistem za utvrdjivanje kapaciteta jednokolosecne pruge, XLIV Konferncija za ETRAN, Bukovicka Banja, Zbornik radova, Sveska III, str. 63-66.

Oliveira, E., Smith, B. M., 2001. A hybrid constraint-based method for single-track railway scheduling problem, Report 2001.04, School of Computing, University of Leeds, (Downloadable from website http://www.comp.leeds.ac.uk/scs/doc/reports/2001/2001_04. pdf).

Pinedo, M., 1995. Scheduling: Theory, Algorithms and Systems, Prentice Hall,

Vieira, E. G, Herrmann, J. W., Lin, E., 2003. Rescheduling manufacturing systems: a framework of strategies, policies and methods, Journal of Scheduling 6, 39-62.