TOPSU – RDM
A Simulation Platform for Online Railway Delay Management

Andre Berger [*]
University of Maastricht
6200 MD, Maastricht, The Netherlands
a.berger@ke.unimaasl.nl

Ralf Hoffmann
Technical University Berlin
10623 Berlin, Germany
rhoffman@math.tu-berlin.de

Ulf Lorenz [†]
Technical University Darmstadt
64289 Darmstadt, Germany
lorenz@mathematik.tu-darmstadt.de

Sebastian Stiller [‡]
Technical University Berlin
10623 Berlin, Germany
stiller@math.tu-berlin.de

ABSTRACT

Delays in a railway network is a common problem that railway companies face in their daily operations. When a train gets delayed, it may either be beneficial to let a connecting train wait so that passengers in the delayed train do not miss their connection, or it may be beneficial to let the connecting train depart on time to avoid further delays. These decisions naturally depend on the global structure of the network and on the schedule. The railway delay management (RDM) problem (in a broad sense) is to decide which trains have to wait for connecting trains and which trains have to depart on time.

The offline version (i.e. when all delays are known in advance) is already NP-hard for very special networks. In this paper we show that the online railway delay management (ORDM) problem is PSPACE-hard, and we present TOPSU – RDM, a simulation platform for evaluating and comparing different heuristics for the ORDM problem with stochastic delays. Our novel approach is to separate the actual simulation and the program that implements the decision making policy, thus enabling implementations of different heuristics to "compete" on the same instances and delay distributions. For RDM and other logistic planning processes, it is our goal to bridge the gap between theoretical models, which are accessible to theoretical analysis, but often too far away from practice, and the methods which are used in practice today, whose performance is almost impossible to measure.

[*]The first three authors have been partially supported by the European Regional Development Fund (ERDF).

[†]Part of this work was done while this author was on leave at Technical University Berlin.

[‡]This authors work was partially supported by the ARRIVAL project, within the 6th Framework Programme of the European Comission under contract no. FP6-021235-2.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Complexity;
G.2.3 [**Mathematics of Computation**]: Discrete Mathematics—*Applications*;
I.6 [**Computing Methodologies**]: Simulation and Modeling

General Terms

Transportation, Simulation, Stochastic Scheduling, Online Optimization, Experimental Algorithms, Heuristics

Keywords

Online Railway Delay Management, Web-based Simulation, PSPACE

1. INTRODUCTION

Delays in a railway network are one of the biggest problems for the daily operations of a railway company. Delayed trains and missed connections lead to dissatisfied customers and possibly refunds that have to be paid to delayed passengers.

When a train does get delayed, it may be beneficial to let another train wait so that passengers in the delayed train do not miss their connection. However, passengers in the waiting train will then get delayed and may in turn miss their connections. Due to the complexity of both, the network and the schedule, one decision may have a large impact on the propagated delays later during the day.

Even today, the decision whether a train should wait or not, is still made by a human dispatcher, mainly based on a lot of training and experience. However, to decrease the delays incurred by making bad decisions, it may be favorable to have these decisions made by an algorithm or at least support a dispatcher by making proposals. Algorithmically, this becomes the problem of finding a good wait policy, a mechanism that decides at any point in time which trains should depart and which trains should wait. Different objective functions may be defined for this problem. In this paper we will consider the problem of minimizing the total delay of all passengers.

It has been shown that even the offline version, i.e. the case when all delays are known in advance, is NP-hard even for very special networks [3, 4]. However, a branch-and-bound algorithm has been developed to solve the problem optimally by using an integer programming formulation for the problem [6].

This IP formulation is based on a model of railway delay management that uses an event-activity-network, where the nodes represent arrival or departure events, and the edges represent driving, transfer, or waiting activities. We will use this description in Section 3 to show that the online railway delay management (ORDM) problem is PSPACE-hard. The only network for which an optimal (polynomial time) algorithm is known is the line [3].

Thus, for practical applications, heuristics have to be developed. It lies in the nature of such heuristics, that their performance is really hard to measure. In particular, for a PSPACE-hard problem, it is infeasible to get good bounds on the optimal solution. Moreover, it is difficult to compare different heuristics due to differences in the model, the objective and the implementation. For the case of railway delay management, it is also important that the source delays are in some sense comparable when different heuristics are evaluated.

In order to overcome these problems, we have developed a simulation platform, on which different heuristics can be applied to different instances and evaluated. In Section 4 we will also discuss how our approach can be used for other logistics and production planning problems as well.

TOPSU – RDM has been developed within the TOPSU framework (**T**ournaments for **O**ptimal **P**lanning and **C**ontrol under **U**ncertainty[1]). TOPSU is an interactive framework for optimal planning and control of production or other control processes under uncertainty. It divides an optimization task in three parts: model building, an algorithm for solving the problem, which is induced from the model, and the experimental evaluation of the algorithm inside the model (cf. Fig. 1).

One most crucial point of TOPSU is that the framework does not only demand this partition, but also allows the distribution of these three tasks to different people. Thus it can be considered as a platform for the competition of algorithms. The second crucial point is the fact that TOPSU supports the influence of uncertainty within its implicit optimization model. We decided to incorporate this feature for two reasons. Firstly, practitioners often claim that production processes have massively to deal with several kinds of uncertainty. Secondly, production systems are typically so large that optimization must focus on a certain part of

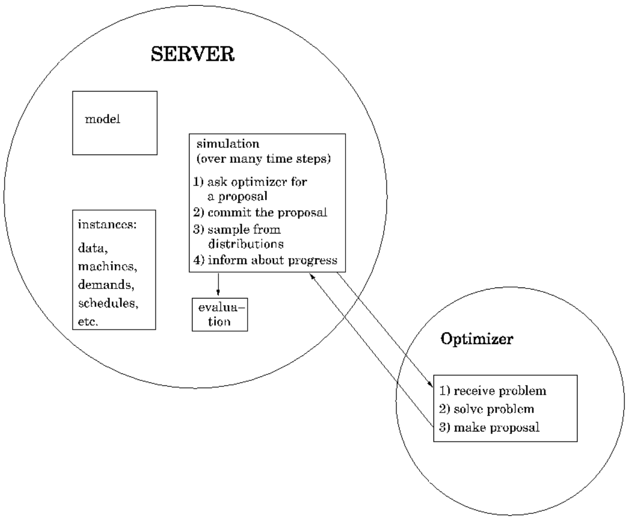


Figure 1: The TOPSU idea: splitting the tasks.

this system. Or in other words, we have to optimize parts of a supply chain. We think, it will be advantageous for the optimization of a supply chain, if its components are aware of uncertain boundaries.

Technically, TOPSU is realized with the help of the Internet. On one site, we have a so called server, where the ultimate simulation proceeds. At this server, all necessary data are available for download. The control of the simulation, however, is remotely executed at the so called clients.

The field of optimization under uncertainty, especially with probability based uncertainties, is a fast growing and more and more important area. Just think of planning tasks for railways or aircrafts, and remember your own experiences with disruptions. Disruptions reflect the fact that, at planning time, not all information is available. Optimization problems, considering these uncertainties, often become PSPACE-hard. In some cases it will be possible to extract easy subproblems which can be solved in polynomial time, and that simultaneously serve as good solutions for the real-world application. However, if this is not possible, simulation experiments promise a lot of gain in insights. Experimental work for the evaluation of optimization processes often has the disadvantage that results of different authors cannot be compared to each other. One reason is that each author examines slightly modified problems, and another reason is that measuring is not standardized. In TOPSU, the methodical tasks “Finding a problem for examination”, “Algorithms and Heuristics”, and “Measuring” are split to different persons. Moreover “Finding a problem for examination” and “Measuring” are centralized. That increases comparability and, simultaneously, the credibility of the experiments.

We will now give a brief overview of the contents of the paper. In Section 2 we present the model and the details of the RDM simulation platform. We show that ORDM is PSPACE-hard in Section 3, and give some conclusions in Section 4.

2. THE SIMULATION PLATFORM

In this section we will describe in more detail the model that is used in our simulation and the specifics of the sim-

[1]TOPSU is the abbreviation of the German translation.

ulation. The simulation platform consists of three parts – a *server* program that implements the model, a program that implements the wait policy (called an *engine*), and a *graphical user interface* (GUI) which enables the communication between the server and the engine and which provides a visualization of the simulation.

2.1 The Server

We start with a description of the model that is implemented on the server. In our model stations are the nodes and tracks are the edges of a directed graph on which the trains can move. Physical tracks that can be used in both directions are modeled as two distinct directed edges, and the server makes sure that only one of these two edges is used at any time. Each station and each track has a capacity, the maximum number of trains that can be in that station or on that track, respectively, at any time. Moreover, each edge has a timeslack, i.e. the minimum time that has to pass between two trains entering or leaving that track. The edges have the FIFO property, i.e. the trains leave an edge in the same order that they entered that edge. There is also a minimum halt time in the stations, the minimum time that trains have to stay in a station before they continue their scheduled route.

In addition to stations and tracks, the server has information about the trains, the schedule, and the passengers. Each entry in the schedule consists of a train, an edge, a departure time, an arrival time, and a pointer to the delay distribution for this entry. Passenger flows are called origin destination pairs, each having a weight (corresponding to the number of passengers), a start time, and a list of edges that these passengers are going to traverse during their travel. Note that rerouting of trains and passengers is not allowed in our model. Additionally, there is a (global) constant, the minimum change time, which is the time needed for a passenger to transfer between one train and another. It is assumed, that each passenger always uses the first train going towards his/her next intermediate destination. This may be the train he/she is currently in, or another train that is heading in the same direction and not leaving before the minimum change time has passed.

The simulation running on the server is discrete-time and event based. During initialization, for each train, an event for its first entry in the schedule is created and inserted into a (time-sorted) priority event queue. As the trains move along the edges, events will be taken out from the event queue and new events will be generated. An event consists of a time, a train, an edge, and an indicator whether this event means the train wants to enter or leave that edge at the specified time. The server will then run the following loop until the end of the schedule has been reached:

1. Collect queries at current time from the event queue.
2. Send a query message to the engine.
3. Receive a result message from the engine.
4. Commit the answers from the result message, if feasible.
5. Sample delays for trains that in fact have left a station.
6. Send message about committed decisions and sampled delays to the engine.

We will now describe some details of the points above.

Collect queries: In this step all events from the top of the event queue, whose event time is equal to the current simulation time, are checked for feasibility and inserted into a query collection that will be sent to the engine. This means that no query is generated for an event which cannot be implemented at that time, e.g. a train wants to enter an edge that is full (i.e. the number of trains on that edge equals the edge's capacity). In this case, a new event for that train is created at the earliest possible time at which the "infeasibility reason" may disappear, e.g. the time of the next event of the first train on that edge.
Sending queries and receiving results: All queries that have been collected in the previous step will be sent to the engine in a single message. The simulation on the server stops until a corresponding result message is received from the engine.
Committing decisions: Similar to the method used while collecting queries, only those queries will be committed which are feasible and were answered positively by the engine. Whenever a query is committed, e.g. a train is entering a track, a new event for that train to leave the edge is created at the expected arrival time – the scheduled travel time plus the sampled delay. If all queries were denied by the engine, the simulation jumps to the next point in time in the event queue, and the events corresponding to the current queries are postponed to that time.
Sampling delays: Delays are sampled whenever a train actually leaves a station. A delay is sampled from the distribution linked to the corresponding entry in the schedule and added to the travel time for that train on that edge.
Sending information to the engine: After committing the decisions and sampling the delays, a message is sent to the engine to inform about the decisions and the delays.

The actual arrival and departure times are stored during the simulation. After the end of the simulation, the objective value of the simulation is computed. This is actually the only time when the passenger data is used. For each origin destination pair the actual travel time is computed and the scheduled travel time is subtracted. The sum of these (weighted) delays is the objective value. It may happen, that a passenger does not reach his/her final destination due to large delays or a bad wait policy. In this case, the actual travel time is replaced by a large constant (e.g. the costs to pay for an accommodation for that passenger).

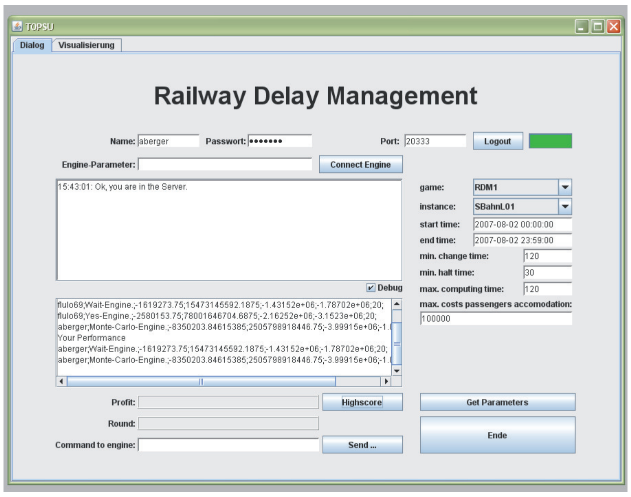
For each instance and each user that runs an engine on that instance, an entry is written to the highscore list of that instance. For several runs by the same user on the same instance, average scores can be seen in the graphical user interface (see below).

2.2 The Graphical User Interface

The graphical user interface (GUI, cf. Fig. 2) enables the communication between the server and the engine. It is also used to connect engine and server, display highscores, and for visualizing the network and the simulation.

The steps in using the GUI to run a simulation are as follows:

- Login (username and password can be obtained from the authors).
- Choosing RDM as the "game".



Name: aberger Passwort: ••••••• Port: 20333

Engine-Parameter:

```
15:43:01: Ok, you are in the Server.
```

game: RDM1
instance: SBahnL01
start time: 2007-08-02 00:00:00
end time: 2007-08-02 23:59:00
min. change time: 120
min. halt time: 30
max. computing time: 120
max. costs passengers accomodation: 100000

[✓] Debug

```
flulo69;Wait-Engine.;-1619273.75;15473145592.1875;-1.43152e+06;-1.78702e+06;20;
flulo69;Yes-Engine.;-2580153.75;78001646704.6875;-2.16252e+06;-3.1523e+06;20;
aberger;Monte-Carlo-Engine.;-8350203.84615385;2505798918446.75;-3.99915e+06;-1.0
Your Performance
aberger;Wait-Engine.;-1619273.75;15473145592.1875;-1.43152e+06;-1.78702e+06;20;
aberger;Monte-Carlo-Engine.;-8350203.84615385;2505798918446.75;-3.99915e+06;-1.0
```

Profit:
Round:
Command to engine:

Figure 2: The graphical user interface of TOPSU – RDM.

- Choosing an instance.
- Getting the pre-specified parameters.
- Connecting an engine.
- Starting a simulation.

If necessary and recognized by the engine, parameters can be passed to the engine via the GUI. The user can also specify a subinterval of the pre-defined timeframe on which the simulation should run, or change parameters such as the cost that is added to the objective for passengers who do not reach their final destination. However, highscores will be only written when the full timeframe with the pre-specified parameters is simulated.

In the visualization panel (cf. Fig. 3) of the GUI, the user can stop and continue the simulation, and proceed stepwise. This may be helpful for the analysis, at least for smaller instances.

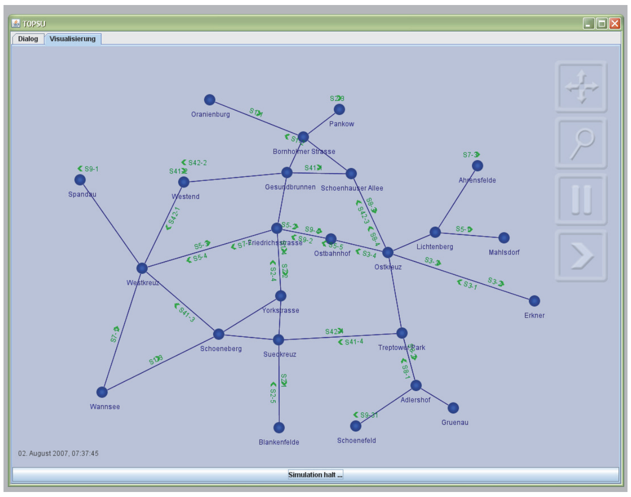


Figure 3: A visualization of the simulation for a (simplified) Berlin S-Bahn schedule.

2.3 The Engine

An engine for railway delay management, i.e. a program implementing a certain wait policy, basically just has to say yes or no to the queries sent by the server. It may do so, of course, without keeping any information about the network, the schedule, or the passengers. "Intelligent" engines, however, will need such information.

This may be information such as previous arrivals and departures, the current location of a train, or the next event of a train in the event queue on the server. For algorithm/wait policy developers, a set of Java classes is available that take care of keeping up to date all the necessary data during a simulation. An engine can use these classes and just has to implement the method that determines the answers to the queries posed by the server. Instructions to implement an engine are available at the TOPSU – RDM webpage [1]. Sample engines that implement the "Always Yes", "Wait for all connections" and "Wait randomly" are available for testing purposes.

3. ONLINE RAILWAY DELAY MANAGEMENT IS PSPACE-HARD

Definition 1 *The* complexity class *PSPACE is the set of all decision problems that can be decided on a deterministic Turing machine using space limited by a polynomial in the input size. A problem P is said to be PSPACE*-hard, *if there is a Karp-reduction from every problem in PSPACE to P. A problem in PSPACE that is PSPACE-hard is called PSPACE*-complete.

It is widely assumed that the complexity class PSPACE is not contained in NP. In other words, there are problems in PSPACE for which there is no polynomial time checkable certificate. If this holds, e.g., for the railway delay management problem, then one may not evaluate a delay management strategy in polynomial time. One could not decide for every value k in polynomial time whether a certain strategy scores in expectation better or worse than k. In general, this would also inhibit the comparison of different strategies. Still, we want to be in the position to prefer one approach to a problem over another on a sound basis. Therefore, we consider a PSPACE-hardness proof as a justification for a simulation based evaluation of strategies.

We will prove in this section that the following simple version of the online railway delay management problem is already PSPACE-hard, i.e., at least as hard (by polynomial time reduction) as any problem in PSPACE.

A known PSPACE-complete problem is the following:

Definition 2 *Deciding whether a logical expression of the following type is true*

$$\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n : \bigwedge_i \bigvee_j z_{ij},$$

where z_{ij} are literals in the variables $\{x_1, \ldots, x_n\}$ and their negations, is called the Quantified Boolean Formula (QBF) *problem.*

We will reduce QBF to a simple version of the online delay management problem.

3.1 The Basic Online Delay Management Problem.

We present the delay management problem for this reduction on an event (nodes) activity (arcs) network, whereas our simulation contains an infrastructure graph. It will become clear that the model used for the reduction is even slightly simpler than that of the simulation. This means that every instance of the reduction model can be described as an instance for our simulation tool. Yet, some further complicating aspects like single tracks, which are included in the simulation, are not need for the reduction, and thus not modeled in this section.

An instance of the *basic online delay management* (BODM) problem

$$(G, \pi, \tau, \mathcal{D})$$

is defined by an event activity network $G = (V, A)$, a nominal timetable π, a vector of random variables for minimum durations of the arcs τ, and a mathematical object $\mathcal{D}$ defining a cost model for the delay management. The task is to give a strategy that constructs a disposition timetable π' in every scenario.

The *event activity network* is a digraph $G = (V, A)$, where V is the disjoint union of nodes representing *arrival events*, V_A, and those for *departure events*, V_D. The arc set A is the disjoint union of the set of *driving arcs* A_D, and the set of *transfer arcs* A_T. A transfer arc always leads from an arrival to a departure node. Whereas driving arcs either lead from a departure to an arrival node, or vice versa. In the latter case driving arcs are also called more adequately *stopping arcs.* A stopping arc models a train in a station, a driving arc a train driving between stations, and a transfer arc represents some passengers changing trains.

Most real-world instances fulfill, that every node has one or zero incoming and independently one or zero outgoing driving arcs. Though there are rare real-world instances which do not fulfill this condition, for our reduction we can respect it.

The real-valued vector $\pi \in \mathbb{R}^V$ is called the *timetable*, and serves as a reference to measure the delay. The A-dimensional vector of random variables $\tau : \Omega \to \mathbb{R}_+^A$ over some probability space $(\Omega, \mathcal{F}, \mu)$ represents the *minimum duration* $\tau(\omega)$ of an arc in a scenario $\omega \in \Omega$.

Different ways to define the cost model $\mathcal{D}$ are possible. Basically, we are given a set of origin-destination pairs with a certain weight, i.e., we know how many passengers want to travel from a certain starting station to a certain final destination. In the simulation their paths through the infrastructure network are fixed. They may follow that path on different trains, but the sequence of stations they pass is fixed. In each scenario the total delay of the passengers in π' compared to π, plus a certain cost for those passengers who will not reach their destination at all, defines the cost.

An alternative way to define the costs, specifies a certain cost for each transfer that is broken and for each arrival which is delayed in π'. The two models are not tantamount, but can be translated into each other in many cases. For the reduction we will use the model used in the simulation. But we will sometimes refer to the costs as the costs of breaking a transfer or delaying a train, because these terms are more convenient in the context, and in our case can easily be translated into the original cost model.

For a scenario $\omega \in \Omega$ we seek a disposition timetable π'. This timetable must respect the realized minimum durations $t(\omega)$, and the old timetable π, i.e., $\pi \leq \pi'$.

As the random variable τ unfolds over time, the disposition timetable π' must be the result of a non-anticipative strategy. The decision problem, which we show to be PSPACE-hard, is the following question:

Definition 3 *The following question is called the* BODM decision problem. *Given a BODM instance, and a point in time t_0. Is there a non-anticipative strategy for the subsequent decisions that achieves a cost value lower than a budget B in every realization of τ, that has probability greater than zero conditional to the given realization of τ until t_0.*

3.2 Reduction of QBF to the BODM Decision Problem

For a given Boolean formula in conjunctive normal form, $\bigwedge_i \bigvee_j z_{ij}$, with literals in the set of variables $\{x_1, \ldots, x_n\}$ and their negations, we construct an instance of the BODM decision problem. For this BODM instance exists a strategy that achieves a cost lower than the budget B, if and only if the quantified Boolean formula, $\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n : \bigwedge_i \bigvee_j z_{ij}$, is true. Thus, the BODM decision problem is PSPACE-hard.

In our construction we use fixed and non-fixed trains. A train is fixed in the sense that delaying this train would automatically exceed the budget by yielding a cost $M_0 > B$. Nevertheless, we use fixed trains, that are a priori fixed to be late. Such a late fixed train has an initial delay prior to the decisions of the strategy, but may neither be delayed any further, nor has a buffer time to compensate the delay. We introduce the late, fixed trains to explain transfers that are a priori broken, i.e., lead from an arrival (of a late, fixed train) to an earlier departure. The costs for the initial delays of those trains are constant for all further unfolding of the scenario and all disposition timetables. Therefore, we can neglect them.

The non-fixed trains fall into two different groups. Each train of the first group, the variable-trains, corresponds to a variable x_i of the Boolean formula. If such a train is delayed, we will interpret the corresponding variable x_i as being false, and true if the train is on time. The trains of the second group serve as modeling variables. They can also be delayed or run on time while the strategy is carried out, but the reduction will be constructed such that their delay is entirely dependent on the delay of the variable-trains.

For modeling reasons we want that for every non-fixed train the decision about running delayed or on time must be taken at the start of the train‘s ride and kept until the final destination. We enforce this by an incoming transfer from a late, fixed train at the beginning of the ride and an outgoing transfer to an on time, fixed train at the end. The non-fixed train cannot meet both transfers. Thus it always incurs the cost for breaking one of these transfers, M_1. Let m be the number of non-fixed trains, then the total budget $M_1 m + C < B < M_1(m + 1) + C$ is set such that none of these trains may break both transfers. (The constant C is the constant cost of all gadgets, as explained below.)

With these ingredients, on time fixed trains, late fixed trains, variable-trains, modeling-trains, and the rule that any of the latter two types‘ trains must be scheduled either always late or always on time, we will below device a gadget for a NON-operator and one for a multiple AND-operator.

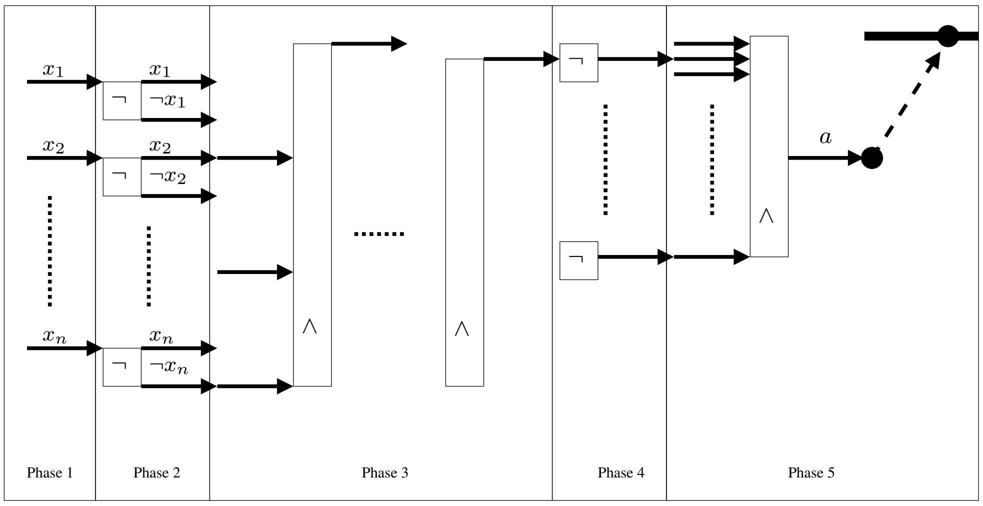


Figure 4: Using the Gadgets.

The NON-gadget will yield that a certain modeling train is delayed if and only if a certain other train is on time. The AND-gadget has an out-train that is on time, if and only if all trains of a certain set are on time. Before we describe the mechanism of these gadgets, we will first show how they are used to reduce the QBF, $\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n : \bigwedge_i \bigvee_j z_{ij}$. Actually, we use an alternative way to write the Boolean formula namely, $\bigwedge_i \bigvee_j z_{ij} = \bigwedge_i \neg(\bigwedge_j \neg z_{ij})$.

The time horizon of the constructed BODM instance is split into five phases (cf. Figure 4).

1. In the first phase the variable-trains are delayed or not. The incoming transfer from the fixed train determines the order by which these decisions must be taken. Thus we reflect the consecutive mechanism in the QBF.

   For those variable-trains that correspond to an all quantified x_i, we choose the random variable τ such that it may do either of the following: Delay that train before the incoming transfer from a late, fixed train at the beginning, or delay the train immediately before the outgoing transfer to a punctual, fixed train at the end of that train. The time difference between these transfers is chosen such that further delaying that train by the delay management is prohibited by the budget constraint. In this way we model the all-quantified variables by variable-trains.

2. Then each variable-train runs through a NON-gadget, producing its negated train, which is late if and only if the variable-train was on time.

3. The variable-trains and their negations pass through the AND-gadgets for each of the re-written clauses. A re-written clause i, $\bigwedge_j \neg z_{ij}$, is modeled by an AND-gadget with in-trains corresponding to the variable-trains or negated variable-trains z_{ij}.

4. Each out-train of those AND-gadgets is negated.

5. Finally all of those negations enter the central AND-gadget. The out-train a of this gadget has a tight transfer to a fixed train. If that out-train is late, it yields a cost of M_2.

We will make sure that every AND- and NON-gadget yields a fixed cost in every scenario. Thus, we can choose B and M_2 such that the total cost is below B, if the train a is on time, and the cost exceeds B, if a is late. This completes the reductions.

3.3 The NON-Gadget

The initial state of a NON-gadget is depicted in Figure 5. The lower train, the in-train is always late for a transfer. We draw a rhombus to symbolize some fixed delay that should explain this fact. The upper train can wait for the lower train and thus keep the connection (Figure 7). But, if the lower train is additionally delayed before the rhombus, the upper train would have to wait so long, that is has to break a transfer to a fixed train. (Fixed trains are always drawn as fat lines.) The costs for breaking this transfer are M_0, i.e., would immediately exceed the budget. Thus, the strategy will break the transfer from the in-train to the out-train, and the latter will leave the gadget on time (Figure 6). A delayed in-train yields an on time out-train, and vice versa.

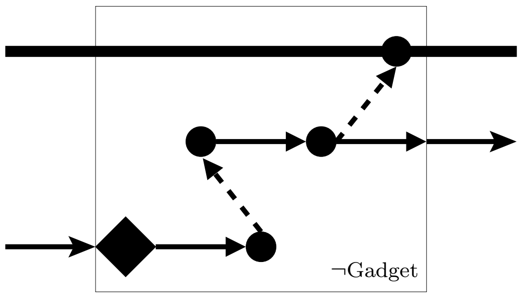


Figure 5: The Initial Situation.

Still the gadget would not work, because we cannot guarantee that the transfer is not broken, if the in-train is on time, or the out-train is delayed although it breaks the trans-

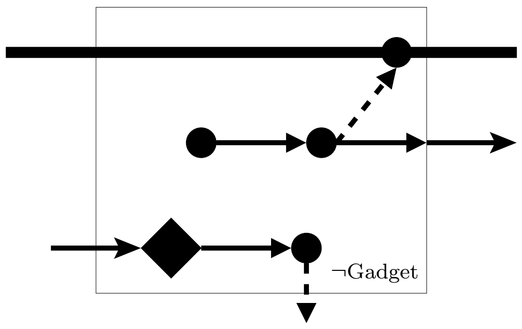


Figure 6: Delayed yields On Time.

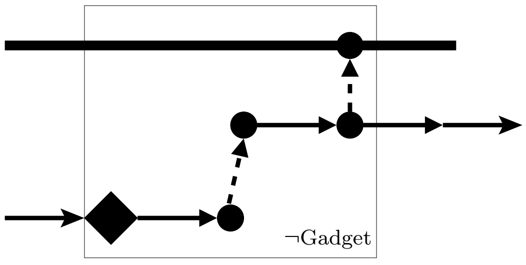


Figure 7: On Time yields Delayed.

fer from a late in-train. To exclude these cases, we have to make sure that a NON-gadget yields a fixed cost in both dispositions we desire (in-train on time & out-train late and vice versa), and exceeds this cost in any other disposition. To this end, let c_w be the cost of delaying the out-train, c_b the cost of breaking the transfer from the in-train, and $c_b - c_w = c_g$ a positive number. We introduce an a priori broken transfer from a late, fixed train to the in-train, which to break costs c_g. This transfer is broken, if and only if the in-train is on time. In other words, the desired dispositions are the only two dispositions by which the gadget has cost less or equal to c_b—and those yield cost equal to c_b.

Note that we use NON-gadgets, which also output the in-train, and NON-gadgets that only output the out-train.

3.4 The AND-Gadget

The AND-gadget is fairly simple: All in-trains have to be on time for the out-train to be on time. Therefore, all in-trains have a tight connection of breaking cost M_0 to the out-train. Again, we have to make sure, that the out-train is not scheduled late although all in-trains are on time. To this end, all in-trains run along the same track for a short distance. There are some passengers that want to travel this distance, but come from a late, fixed train. Only if at least one of the in-trains is late, these passengers will reach their destination. The cost c_h of not serving these passengers equals the cost of delaying the out-train. Thus, the gadget at least costs c_h, and will exceed this cost, in case the out-train is late though all in-trains are punctual.

4. FUTURE WORK / CONCLUSIONS

4.1 Refinement of the Model

The following refinements can be made to the model to improve the applicability of the simulation tool and to get the model closer to practice. First, different objectives should be implemented and should be made available to the users in the GUI.

Moreover, the minimum change times of passengers do depend on the station where a passenger changes and may also depend on the passenger himself. Similarly, the halt times in a station may differ during peak periods and may also depend on the train and on the station.

4.2 More Intelligent Engines

We have mentioned that some trivial engines (“Always Yes”, “Wait for all connections” and “Wait randomly”) have been implemented. They can be used to compare other, more *intelligent* engines.

We are currently working on designing and implementing two engines that will hopefully perform better than the engines mentioned above. The first algorithm explores ideas from Monte Carlo tree search, as have been used in before in Computer Go programs [5] and in production management problems [2]. This approach tries to simulate the outcome of the possible decisions that are available, and may thus also be used just as a decision support tool that evaluates and/or estimates the outcome of certain decisions.

The second wait policy is based on the optimal algorithm that solves the offline RDM problem [6]. Both wait policies are currently being implemented and will be tested and evaluated using the TOPSU – RDM simulation tool.

4.3 A Scheduling and Planning Simulation Generator

Currently, we are working on a generalization of the RDM simulator. We aim at building a kind of simulation generator. We restrict our efforts to optimization problems, where some passive objects pass some other active objects. This includes a lot of transportation problems as well as production problems. Sets of passive objects may be transformed while they are processed within the active objects. Active objects might be machines and passive ones might be items which are processed inside the machines. Active objects also might be stations and tracks, and the passive would then be trains, which pass through the tracks and stations. We think, a proper entity-relationship description, plus some extra information, including message-layout between clients and server suffices to automatically construct a generic simulation block, where the messaging, as well as the basic event-handling of the simulator are included.

5. ACKNOWLEDGEMENTS

We would like to thank Robert Pankrath for developing the visualization for TOPSU – RDM.

6. REFERENCES

[1] A. Berger, R. Hoffmann, U. Lorenz, and S. Stiller. TOPSU–RDM – A Web-Based Simulation for Railway Delay Management. http://wwwcs.uni-paderborn.de/cs/ag-monien/PERSONAL/FLULO/PP/TOPSU_RDM1.html.

[2] G. Chaslot, S. de Jong, J.-T. Saito, and J. Uiterwijk. Monte-Carlo Tree Search in Production Management Problems. In *Proceedings of BNAIC 2006*, (2006).

[3] M. Gatto, B. Glaus, R. Jacob, L. Peeters, and P. Widmayer. Railway delay management: Exploring its algorithmic complexity. In *Algorithm Theory - Proceedings SWAT 2004*, volume 3111 of *LNCS*, pages 199–211. Springer, 2004.

[4] M. Gatto, R. Jacob, L. Peeters, and A. Schöbel. The computational complexity of delay management. In D. Kratsch, editor, *Graph-Theoretic Concepts in Computer Science: 31st International Workshop (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*, 2005.

[5] C. R. Efficient selectivity and backup operators in monte-carlo tree search. In *5th International Conference on Computer and Games*, 2006.

[6] A. Schöbel. Integer programming approaches for solving the delay management problem. *Lecture Notes in Computer Science*, 2006. to appear.