Routing Trains Through Railway Stations: Model Formulation and Algorithms[1]

PETER J. ZWANEVELD, LEO G. KROON, H. EDWIN ROMEIJN, and MARC SALOMON

Rotterdam School of Management, Erasmus University Rotterdam, Rotterdam, The Netherlands

STÉPHANE DAUZÈRE-PÉRÈS

Department of Automatic Control and Production Engineering, Ecole des Mines de Nantes, Nantes, France

STAN P. M. VAN HOESEL

Department of Quantitative Economics, University of Limburg, Maastricht, The Netherlands

HARRIE W. AMBERGEN

Railned, Utrecht, The Netherlands

In this paper we consider the problem of routing trains through railway stations. This problem occurs as a subproblem in a project which the authors are carrying out in cooperation with the Dutch railways. The project involves the analysis of future infrastructural capacity requirements in the Dutch railway network. Part of this project is the automatic generation and evaluation of timetables. To generate a timetable a hierarchical approach is followed: at the upper level in the hierarchy a tentative timetable is generated, taking into account the specific scheduling problems of the trains at the railway stations at an aggregate level. At the lower level in the hierarchy it is checked whether the tentative timetable is feasible with respect to the safety rules and the connection requirements at the stations. To carry out this consistency check, detailed schedules for the trains at the railway yards have to be generated. In this paper we present a mathematical model formulation for this detailed scheduling problem, based on the Node Packing Problem (NPP). Furthermore, we describe a solution procedure for the problem, based on a branch-and-cut approach. The approach is tested in an empirical study with data from the station of Zwolle in The Netherlands.

In this paper we consider the problem of routing trains through railway stations. This problem usually occurs at three levels in the planning hierarchy of a railway company. At the *strategic* planning level, the problem occurs in the analysis of future infrastructural capacity requirements, such as the number of platforms in the station and the number of tracks at the station yard. At the *tactical* planning level, the problem occurs in the actual generation of timetables. Finally, at the *operational* level the problem occurs when timetables have to be adjusted for day-to-day disturbances, such as delays of trains.

Here, we consider the problem at the strategic level in the planning hierarchy. The authors are carrying out the research on this problem as members of a project-team for Railned and the Dutch railway company 'Nederlandse Spoorwegen' (NS). The former organization is responsible for determining the required future capacity of the Dutch railway infrastructure. Final objective of the project is to develop a decision support system, called DONS,

[1]This research is sponsored by Railned and Nederlandse Spoorwegen (Netherlands Railways).

Transportation Science
Vol. 30, No. 3, August 1996

0041-1655/96/3003-0181 $01.25

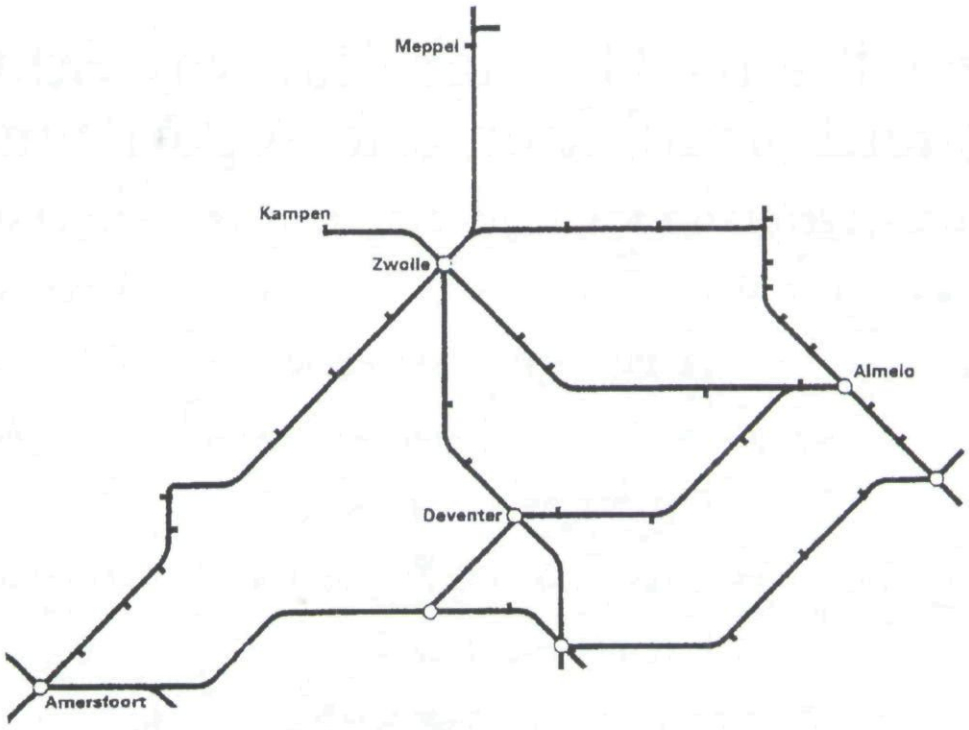

Fig. 1. Dutch railway network of Zwolle and surroundings.

that enables the strategic planners at the railway company to evaluate the infrastructural capacity requirements related to different scenarios with respect to the expected future demand for railway transportation. In order to carry out this evaluation at an appropriate level of detail, complete timetables have to be generated within DONS. However, the analysis by CAREY (1994) indicates that the generation of complete timetables is computationally feasible for small instances only. Therefore it was decided to follow a two-level hierarchical approach for the Dutch railway network. At the upper level in the hierarchy, tentative timetables are generated by a sub-system called CADANS (see SCHRIJVER and STEENBEEK, 1994; SERAFINI and UKOVICH, 1989).

The input of CADANS is the railway network (part of which is shown in Fig. 1), with estimated travel times between the stations, the lines and corresponding frequencies, and the connections between lines which have to be offered at the stations. Output of CADANS is a timetable (for one hour), which includes the arrival and departure times of the trains at the stations. However, since the tentative timetable does not take into account the detailed layout of the railway stations, but simply treats stations as nodes in the network, it is important to check whether a detailed schedule for the trains through the stations with the arrival and departure times generated by CADANS actually exists. In order to carry out this consistency check, a second subsystem called STATIONS (see KROON and ZWANEVELD, 1995) is currently under development. Output of STATIONS is a detailed schedule for the trains at the station yards, including an assignment of trains to routes and platforms at the stations. In generating the detailed schedule, safety rules, as well as connection requirements between trains and customer service considerations are taken into account. Furthermore, if not all trains can be scheduled according to the arrival and departure times suggested by CADANS, then STATIONS should try to find a schedule which is feasible within small deviations from the given arrival and departure times.

In this paper we discuss the procedures that are currently used within STATIONS to automatically generate the detailed schedules for the trains at the station yards. In Section 2 we describe the scheduling problem in more detail: we introduce the terminology, the notation, and the computational complexity of the problem. In Section 3 we show that the scheduling problem can be formulated as a Node Packing Problem (NPP), and we discuss the procedure to generate valid inequalities that are used within a branch-and-cut procedure. The branch-and-cut procedure itself is further outlined in Section 4. To show the effectiveness of the branch-and-cut procedure, the paper is concluded in Section 5

with an empirical study based on the station of Zwolle, one of the largest stations in The Netherlands.

1. PROBLEM DESCRIPTION, NOTATION AND COMPLEXITY

1.1. Problem Description

THE PROBLEM OF ROUTING trains through railway stations can be stated as follows: given the layout of a railway station and the global timetable for a set of trains, is it possible to route these trains through the station such that no pair of trains is conflicting, and such that a number of service considerations is satisfied? We will call this the *feasibility problem*. The reason that we are primarily interested in the feasibility issue is that, as stated before, we consider the problem at the strategic, as opposed to the tactical or the operational, level.

The characteristics of the problem that we describe in this paper pertain to the railway system in The Netherlands, which is very similar to most European systems.

A railway station can be entered by a train from a number of *entering points*, and it can be left through a number of *leaving points*. In general, each entering point can also serve as a leaving point, and vice versa. Furthermore, each of these points corresponds to a *direction of travel*. For example, the directions of travel of the Dutch railway station of Zwolle are Almelo, Amersfoort, Deventer, Kampen, and Meppel (see Fig. 1). The railway network outside the entering and leaving points is not taken into account in the problem addressed in this paper.

A railway station consists of *platforms* and of a large number of track *sections*. An *inbound route* is a sequence of sections linking an entering point to a platform. Similarly, an *outbound route* is a sequence of sections linking a platform to a leaving point. A *complete route* is either a combination of an inbound and an outbound route using the same platform, or a sequence of sections connecting an entering point to a leaving point, bypassing the platforms. There will often be many different routes between a given pair of entering and leaving points, and even several different routes that use the same platform.

The *arrival time* of a train is the time at which the train stops at a platform, after traveling along an inbound route. Similarly, the *departure time* of a train is the time at which the train leaves the platform along an outbound route. The arrival and departure times of the trains are generated by CADANS, and can thus be assumed given.

Clearly, the routing of each train will depend on the routings of other trains. Most importantly, safety rules of the Dutch railways dictate the following procedure. As soon as a train arrives at its entering point of the station, it claims an inbound route to a platform. Since any track section can only be claimed by one train at a time, an inbound route is not feasible for a particular train if any section of the route is already claimed by another train. As a train traverses its chosen route, it sequentially releases each of the track sections comprising the route. In particular, each section will be released after a certain minimum time interval (called the *buffer time*) has elapsed since a train left that section. This buffer time is included to improve the robustness of the timetable and corresponding routings with respect to disruptions in the daily operation of the railway system. A similar procedure is followed for the outbound route, and for the complete route if a train does not stop at a platform. Important for the modeling of the safety system is the exact calculation of the time at which a route is reserved and the time at which a section is released. Therefore the (inbound, outbound or complete) route has to be known for the calculation of all relevant time instants. These time instants are calculated using well-known formulas from the theory of dynamics, taking into account (i) the lengths of the routes, and (ii) the assumption that trains have either a constant velocity, or a constant acceleration or deceleration.

Although these are the current safety rules of the Dutch railway company, the model and solution method we will propose do not depend on this particular safety system. In fact, as we will see later, a very broad class of safety systems can be handled by our algorithm. Other constraints that have to be taken into account concern the coupling or uncoupling of trains at a platform, the requirement that certain trains use the same platform, and service considerations, like the possibility for passengers to transfer between certain pairs of trains.

1.2. Notation

To formulate our model for the feasibility problem, we denote the set of trains by T, the set of routes (complete, inbound, and outbound) by R, and the set of all platforms by P. The routes are made up of track sections, the set of which will be denoted by S. Furthermore, let $R_t \subseteq R$ denote the set of routes that can be used by train $t \in T$. The arrival and departure times of train t are denoted by a_t and d_t, usually in minutes. The latter is required by the Dutch railway company for reasons of convenience towards passengers, train drivers, and planners. We can deal with both sequential and cyclical timetables. A sequential timetable is simply a timetable

over some fixed planning horizon. A cyclical timetable is a timetable for, say, 1 hour, which is then repeated. The only adjustment that needs to be made for cyclical timetables is that we need to use arithmetic modulo 1 hour for all time computations.

If it is impossible to schedule all trains with the original arrival and departure times, we can allow for small deviations from these times. A *deviation* is a combination of a deviation from the arrival time and a deviation from the departure time. A deviation is denoted by $\delta = (\delta^a, \delta^d)$, where δ^a is the deviation from the arrival time, and δ^d is the deviation from the departure time. To reflect these deviations, let $a_{t\delta} = a_t + \delta^a$ be the arrival time of the train adjusted for the deviation. Similarly, let $d_{t\delta} = d_t + \delta^d$ denote the adjusted departure time. Since the arrival and departure times are required to be integers, the deviations should also be integers. Therefore, the set of deviations, which we will denote by Δ, is finite for cyclical timetables. For sequential timetables we will make the additional assumption that only a finite number of deviations will be allowed. Note that the original arrival and departure times correspond to a zero deviation vector $\delta = (0, 0)$ for all trains. In principle, the set of allowable deviations can even be train-dependent (due to circumstances *outside* the station).

For every train t we will introduce a set F_t of allowable route-deviation combinations. In this way we can, for instance, exclude certain routes or platforms for a particular train, and ensure that the time at the platform (i.e., $d_{t\delta} - a_{t\delta}$) will exceed some prespecified (and possibly train-dependent) value.

The safety rules described in the previous section can be represented by defining, for each pair of trains $t, t' \in T$, a set $F_{tt'}$. These sets contain *pairs* of allowable route-deviation combinations $(r, \delta; r', \delta')$. That is, $(r, \delta; r', \delta') \in F_{tt'}$ implies that the routing of train t on route r with deviation δ is *compatible* with the routing of train t' on route r' with deviation δ'. It is clear that this is a very flexible way of modeling the safety rules. Therefore, as was noted above, this modeling can accommodate a large variety of safety rules (including the current Dutch ones). In fact, the necessity of being able to handle such safety rules implies that routes should be taken into account explicitly, as opposed to, for instance, dealing directly with the sections of the railway station. It has already been noted that dealing with complete routes is necessary for the exact calculation of the reservation times of routes and the release times of sections.

Actually, many other constraints can be modelled in the same way. For instance, consider the situation where two trains have to be *coupled* at a railway station. One of the trains is called the *leading* train, while the other train, called the *following* train, has to be coupled onto the leading train. In this case, the leading train has to be assigned to a complete route, while the following train has to be assigned to an inbound route only (that is compatible with the route of the leading train). A similar situation occurs if a train has to be *uncoupled* into two parts. It is clear that these (un-)coupling constraints can be modelled using the sets $F_{tt'}$ introduced above. In this case, $(r, \delta; r', \delta')$ is an element of $F_{tt'}$ only if a necessary coupling or uncoupling procedure involving trains t and t' can be performed when using these route-deviation combinations.

Finally, consider the following service considerations. Firstly, convenience considerations towards the passengers may dictate that certain groups of trains all leave from the same platform. For instance, such a group of trains may consist of all trains leaving into the same direction. Secondly, one may wish to incorporate certain transfer possibilities between trains into the schedule. That is, pairs of trains need to use platforms that are close to each other. Moreover, there needs to be a certain minimum overlap in the time intervals spent at their respective platforms, in order to ensure that passengers can indeed transfer. Once again, these constraints can be modelled by appropriately adjusting the sets $F_{tt'}$ introduced above.

In the remainder of this paper we will only consider constraints that can be modelled using the sets $F_{tt'}$.

1.3. Complexity

The size of the feasibility problem can be measured in terms of the number of trains $|T|$, the layout of the railway station as measured by the number of track sections $|S|$, and the number of possible route-deviation combinations available for each train. The layout of the railway station will be characterized by the set of track sections S that make up the routes.

The feasibility problem

Given a railway station, with a corresponding set of track sections S, a set of trains T with corresponding default arrival and departure times a_t and d_t (for $t \in T$), a set of route-deviation combinations $F_t \subseteq R \times \Delta$, and sets $F_{tt'} \subseteq F_t \times F_{t'}$ containing all pairs of compatible route-deviation combinations for all pairs of trains $t, t' \in T$, determine whether all trains can be routed through the station.

In this section we will show that this problem is NP-complete.

THEOREM 1. *The feasibility problem is NP-complete.*

Proof. It is easy to see that the problem is in the class NP (see GAREY and JOHNSON (1979) for the definition). Now consider the problem of scheduling jobs with fixed start and end times on non-identical parallel machines, where each job can only be processed on a specified subset of the machines. ARKIN and SILVERBERG (1987) prove that this scheduling problem is NP-complete by a reduction from 3SAT. We will reduce this problem (which is described in more detail below) to the feasibility problem, thus proving that the latter is NP-complete as well.

The scheduling problem of Arkin and Silverberg can be described as follows:

> Given a set $J = \{J_1, \ldots, J_n\}$ of n jobs, the start and end times (a_i, d_i) of each job J_i, and a job-machine mapping between J and a set of k non-identical machines, determine whether all jobs can be processed.

For each job J_i, introduce a train i with arrival time a_i and departure time d_i. Let every machine M_j correspond to a route. If $\mathcal{M}_i$ denotes the set of machines on which job J_i can be processed, then let $F_i = \mathcal{M}_i$, where F_i is the set of allowable routes for train i. Now a route combination $(M_j, M_{j'})$ is in the set $F_{ii'}$ $(i \neq i')$ if (a) $j \neq j'$; (b) $d_{i'} \leq a_i$; or (c) $d_i \leq a_{i'}$. This instance of the feasibility problem (without deviations!) can be constructed in polynomial time. It is easy to see that a feasible schedule for the scheduling problem is equivalent to a feasible routing of all trains. □

The complexity of the feasibility problem is studied in more detail in KROON, ROMEIJN and ZWANEVELD (1995). In particular, it turns out that the feasibility problem is solvable in polynomial time if each train has at most two available route-deviation combinations, and it is NP-complete if each train can have three available route-deviation combinations. On the other hand, if the layout of the station is considered fixed and only minor deviations are allowed, then the problem is solvable in an amount of time that is polynomial in the number of trains to be routed. The involved algorithm is based on dynamic programming.

2. MODEL FORMULATION

2.1. Node Packing Formulation

IN THE PREVIOUS SECTION, the feasibility problem was stated in its *decision form*. That is, the question was: "is it possible to route all trains through the railway station?". In this section, we will switch to the optimization form of the feasibility problem: "what is the maximum number of trains that can be scheduled?". Obviously, solving the optimization form of the problem also solves the decision form of the problem, by simply checking whether the number of trains that are scheduled in the optimal solution is equal to the total number of trains. In fact, in the remainder of this paper we will address both the decision problem and the optimization problem as the *feasibility problem*, where the correct interpretation should be clear from the context.

We will show that the feasibility problem can be formulated as a *Node Packing Problem* (NPP). Formally, the Node Packing Problem (NPP) reads as follows:

> Let $G = (V, E)$ be an undirected graph, where V is the set of nodes, and E is the set of edges. Then, a *node packing* (or *stable set*) is a set $S \subseteq V$ such that no edge joins two members of S. Then the NPP is the problem of finding a node packing of maximum cardinality.

The general NPP is thus characterized by an undirected graph. For the feasibility problem, we define a vertex of the graph for each allowable train-route-deviation combination (t, r, δ). The identification of each individual combination is necessary because of the calculation of the traveling times of the trains. The exact calculation of the traveling times is of crucial importance in practice. The following edges are added to the graph:

(*i*) Connect a vertex (t, r, δ) to all other vertices associated with the same train t.
(*ii*) Connect a pair of vertices (t, r, δ) and (t', r', δ') if $(r, \delta; r', \delta') \notin F_{tt'}$.

Here (*i*) ensures that each train is assigned to at most one route-deviation combination, and (*ii*) excludes conflicting train-route-deviation combinations for pairs of trains.

Obviously, a node packing represents a feasible routing of a number of trains through the railway station. A node packing of maximum cardinality represents a feasible routing of as many trains as possible through the railway station. Note that a reasonable upper-bound on the maximum cardinality node packing is known, namely the total number of trains.

2.2. Valid Inequalities

The feasibility problem, when regarded as a Node Packing Problem, can be formulated as an integer

linear program as follows.

$$\max \sum_{t \in T} \sum_{(r,\delta) \in F_t} X_{tr\delta}$$

subject to

$$X_{tr\delta} + X_{tr'\delta'} \leq 1 \quad \text{for all } t \in T;\ (r,\ \delta),\ (r',\ \delta') \in F_t \tag{1}$$

$$X_{tr\delta} + X_{t'r'\delta'} \leq 1 \quad \text{for all } t \neq t' \in T;\ (r,\ \delta) \in F_t;\ (r',\ \delta') \in F_{t'};\ (r,\ \delta;\ r',\ \delta') \notin F_{tt'} \tag{2}$$

$$X_{tr\delta} \in \{0,\ 1\} \quad \text{for all } t \in T;\ (r,\ \delta) \in F_t.$$

This formulation is, in general, not very tight. That is, the optimal solution to the LP-relaxation is not close to the optimal IP-solution (see NEMHAUSER and WOLSEY, 1988). However, it is possible to tighten the formulation as follows.

First, note that the subgraph in the Node Packing graph of all nodes corresponding to a single train is a complete graph, i.e. a *clique*. Since every pair of nodes corresponding to a single train is connected, and only one of each pair of variables corresponding to a pair of connected nodes can be set to one, we know that only one out of all variables corresponding to one train may be set to one—which yields the following valid inequalities:

$$\sum_{(r,\delta) \in F_t} X_{tr\delta} \leq 1 \quad \text{for all } t \in T. \tag{3}$$

These inequalities can then replace equations (1) in the above integer linear programming problem, thereby both tightening the formulation, and strongly reducing the number of constraints.

It is clear that this line of reasoning can be extended to arbitrary cliques in the graph (see e.g. PADBERG, 1973). These valid inequalities are called *clique inequalities*. However, the number of clique inequalities is exponential in the problem size. An efficient choice must therefore be made, based upon the characteristics of the problem. In this section we present the general idea behind our selection procedure. The implementation is discussed in detail in section 3.2.

The valid inequalities (3) represent cliques containing nodes corresponding to a single train. A straightforward extension of this is to consider all nodes corresponding to *two trains*, and to find a clique in that subgraph.

Indeed, consider two trains $t \neq t'$, and route-deviation combinations satisfying $(r,\ \delta;\ r',\ \delta')$, $(r,\ \delta;\ r'',\ \delta'') \notin F_{tt'}$ (where $(r,\ \delta) \in F_t$ and $(r',\ \delta')$, $(r'',\ \delta'') \in F_{t'}$). Then the model will contain the following constraints:

$$X_{tr\delta} + X_{t'r'\delta'} \leq 1$$

$$X_{tr\delta} + X_{t'r''\delta''} \leq 1.$$

However, since only one route-deviation combination can be chosen for train t', we also have $X_{t'r'\delta'} + X_{t'r''\delta''} \leq 1$. We can now replace the first two constraints by the single constraint

$$X_{tr\delta} + X_{t'r'\delta'} + X_{t'r''\delta''} \leq 1.$$

Continuing this line of reasoning to all feasible route-deviation combinations for train t' that are incompatible with $(r,\ \delta)$ for train t, we obtain

$$X_{tr\delta} + \sum_{(r',\delta') \notin F_{t'}^{tr\delta}} X_{t'r'\delta'} \leq 1$$

where $F_{t'}^{tr\delta}$ denotes the set of route-deviation combinations for train t' that are compatible with route-deviation combination $(r,\ \delta)$ for train t, i.e.

$$F_{t'}^{tr\delta} = \{(r',\ \delta') \in F_{t'} : (r,\ \delta;\ r',\ \delta') \in F_{tt'}\}.$$

Repeating this process for each pair of trains, the set of constraints (2) can be replaced by

$$X_{tr\delta} + \sum_{(r',\delta') \notin F_{t'}^{tr\delta}} X_{t'r'\delta'} \leq 1 \quad \text{for all } t, t' \in T;\ (r,\ \delta) \in F_t.$$

Adding these new constraints does not only improve the upper bound obtained by LP-relaxation, but also strongly reduces the number of constraints in the problem since the corresponding constraints (2) are removed.

Taking this process one step further, let $(r_1,\ \delta_1)$ and $(r_2,\ \delta_2)$ be two feasible route-deviation combinations for train t, that is, $(r_1,\ \delta_1)$, $(r_2,\ \delta_2) \in F_t$. Suppose that the set of route-deviation combinations of train t' that are not compatible with train t and $(r_1,\ \delta_1)$ (given by the set $F_{t'} \setminus F_{t'}^{tr_1\delta_1}$) intersects the set of route-deviation combinations of train t' that are not compatible with train t and $(r_2,\ \delta_2)$ (given by the set $F_{t'} \setminus F_{t'}^{tr_2\delta_2}$), i.e., $(F_{t'} \setminus F_{t'}^{tr_1\delta_1}) \cap (F_{t'} \setminus F_{t'}^{tr_2\delta_2}) = F_{t'} \setminus (F_{t'}^{tr_1\delta_1} \cup F_{t'}^{tr_2\delta_2}) \neq \emptyset$. Then it is clear that we can add the following valid inequality

$$X_{tr_1\delta_1} + X_{tr_2\delta_2} + \sum_{(r',\delta') \notin (F_{t'}^{tr_1\delta_1} \cup F_{t'}^{tr_2\delta_2})} X_{t'r'\delta'} \leq 1.$$

Generalizing this to a set $A \subseteq F_t$ of route-deviation combinations for train t, we can add:

$$\sum_{(r,\delta) \in A} X_{tr\delta} + \sum_{(r',\delta') \notin \cup_{(r,\delta) \in A} F_{t'}^{tr\delta}} X_{t'r'\delta'} \leq 1. \tag{4}$$

The same idea could obviously be extended to combinations of three or more trains.

3. THE ALGORITHM

IN THIS SECTION we propose an algorithm for solving the feasibility problem, based on the integer programming formulation of the problem, and the possibility to add valid inequalities. The algorithm is designed to be effective, i.e., its computation time is small for problem instances occurring in practice. For more remarks on this point, see HOFFMAN and PADBERG (1993) and PADBERG and RINALDI (1990). Very generally, the algorithm reads as follows:

Routing algorithm

Step 0. Initialization: generate all routing alternatives (t, r, δ), and determine, for all pairs $t \neq t' \in T$, the sets $F_{tt'}$ of pairs of routing alternatives that can be chosen simultaneously.

Step 1. Preprocessing: simplify the problem by removing *dominated* routing alternatives.

Step 2. Formulate the problem as an integer programming problem, and tighten the problem by adding valid inequalities. The integer programming problem is the first subproblem to be investigated by the branch-and-cut procedure (Step 4).

Step 3. Use heuristics to obtain a good initial solution.

Step 4. Apply a branch-and-cut procedure to obtain the optimal solution to the problem.

In the remainder of this section we will make the steps in the algorithm more concrete.

3.1. Initialization and Preprocessing

In Steps 0 and 1 of the algorithm the (useful) routing possibilities for all trains have to be determined, and the admissible combinations of routing possibilities for combinations of trains. We will assume that, for every train t, the set of allowable routing possibilities F_t is given, together with the other train specific data. Using this information, plus the safety, (un-)coupling, and connection requirements and (other) service considerations, we can determine the sets $F_{tt'}$ in $\mathcal{O}(|T|^2|R|^2|\Delta|^2)$ time.

However, it may happen that certain routing possibilities are dominated by others. In that case we can eliminate such routing possibilities, thereby reducing the size of the problem. In particular, a routing possibility $(\bar{r}, \bar{\delta})$ for train $\bar{t}$ does not need to be considered if there is another routing possibility $(\bar{r}, \bar{\delta})$ which leaves at least the same options open for all other trains, i.e. if there exists some $(\bar{r}, \bar{\delta})$ such that

$$\{(r, \delta):(\bar{r}, \bar{\delta}; r, \delta) \in F_{\bar{t}t}\} \subseteq \{(r, \delta):(\bar{r}, \bar{\delta}; r, \delta) \in F_{\bar{t}t}\}$$

for all trains $t \neq \bar{t}$. The variable $X_{\bar{t}\bar{r}\bar{\delta}}$ can then be eliminated from the problem.

We check dominance only for variables corresponding to routes using the same platform. As we will see later, adding this preprocessing step has a significant effect on the size of the problem, and thus on the time necessary to solve it.

3.2. Adding Valid Inequalities

To tighten the formulation of the LP-relaxation of the problem, we use a procedure to generate a subset of the valid inequalities (4) which were discussed at the end of Section 2.2.

In general, the valid inequalities of the form (4) have the drawback that the time needed for constructing all of them grows exponentially in the size of the problem (as does their number). Therefore we propose to generate only the subclass of valid inequalities where the set A is such that $\cup_{(r,\delta)\in A} F_{t'}^{tr\delta}$ is equal to $F_{t'}^{t\bar{r}\bar{\delta}}$ for some $(\bar{r}, \bar{\delta}) \in A$. In other words, we only consider those cases where the set of feasible route-deviation combinations for train t' that are not compatible with train t and $(\bar{r}, \bar{\delta})$ is included in the set of route-deviation combinations for train t' that are not compatible with train t and (r, δ), for all $(r, \delta) \in A$. Thus $F_{t'} \setminus F_{t'}^{t\bar{r}\bar{\delta}} \subseteq F_{t'} \setminus F_{t'}^{tr\delta}$ (or, equivalently, $F_{t'}^{tr\delta} \subseteq F_{t'}^{t\bar{r}\bar{\delta}}$) for all $(r, \delta) \in A$.

In this way we will obtain an initial problem description that is rather compact, does not require much computing time to generate and, as we will see in Section 4, is quite tight. Below we will propose a procedure for generating these valid inequalities in a systematic (and efficient) way.

The algorithm described below should be executed for every pair of (different) trains $(t, t') \in T \times T$. In the algorithm, several sets are built up sequentially. In particular, in iteration k of the algorithm, the sets R^i $(i = 1, \ldots, k)$ are subsets of the possible routings for train t' (i.e., $R^i \subseteq F_{t'}$). Recall that the set $F_{t'}^{tr\delta}$ is the set of routings that are allowable for train t', and are consistent with routing possibility (r, δ) for train t. We will denote its complement in $F_{t'}$ by $\bar{F}_{t'}^{tr\delta}$, so $\bar{F}_{t'}^{tr\delta} = F_{t'} \setminus F_{t'}^{tr\delta}$. Furthermore, the sets $S_=^i$ $(i = 1, \ldots, k)$ are subsets of F_t, having the following property: for all $(r, \delta) \in S_=^i$: we have $(r, \delta) \in F_t$ and $\bar{F}_{t'}^{tr\delta} = R^i$. Similarly, the sets $S_\subset^i$ $(i = 1, \ldots, k)$ denote subsets of F_t such that $R^i \subset \bar{F}_{t'}^{tr\delta}$ for all $(r, \delta) \in S_\subset^i$. Now the valid inequality procedure can be described as follows:

Valid inequality procedure

Step 0. Set $k = 0$.

Step 1. Select a routing (r, δ) from F_t that has not been considered yet. If all routings have been considered, go to Step 4. Set $S = \emptyset$, and rank the sets R^i $(i = 1, \ldots, k)$ in order of non-decreasing cardinality.

Step 2. For $i = 1, \ldots, k$, compare $\bar{F}_{t'}^{tr\delta}$ to R^i:
- If $R^i \subset \bar{F}_{t'}^{tr\delta}$, then $S_{\subset}^i = S_{\subset}^i \cup \{(r, \delta)\}$.
- If $R^i = \bar{F}_{t'}^{tr\delta}$, then $S_{=}^i = S_{=}^i \cup \{(r, \delta)\}$, and return to Step 1.
- If $\bar{F}_{t'}^{tr\delta} \subset R^i$, then $S = S \cup S_{=}^i$.

Step 3. Increment k, let $R^k = \bar{F}_{t'}^{tr\delta}$, $S_{=}^k = \{(r, \delta)\}$, $S_{\subset}^k = S$, and return to Step 1.

Step 4. For $i = 1, \ldots, k$, create the following constraint:

$$\sum_{(r,\delta)\in S_{=}^i} X_{tr\delta} + \sum_{(r,\delta)\in S_{\subset}^i} X_{tr\delta} + \sum_{(r',\delta')\in R^i} X_{t'r'\delta'} \leq 1 .$$

This constraint is *lifted* by sequentially extending the corresponding clique to a maximal clique. The sequence in which the connected variables are investigated is chosen randomly, based on a uniform distribution over the remaining variables.

In Step 1 the sets R^i are ranked in order to decrease the time required for the comparisons if $R^i = \bar{F}_{t'}^{tr\delta}$, since in that case no set R^j, with $j \geq i$, can satisfy $R^j \subset \bar{F}_{t'}^{tr\delta}$. In the actual implementation, this ranking step is avoided by adding the new set at the right position in the list, and renumbering the sets, in Step 3.

3.3. Heuristics

A heuristic yielding a good initial solution is essential for obtaining an efficient branch-and-cut algorithm. We propose two (randomized) heuristics, that can be used to generate several feasible solutions for the optimization problem.

The first heuristic is based on the idea that, for each train, a routing possibility should be chosen that leaves room for as many trains as possible. In other words, a routing is chosen such that the number of trains that will have at least one routing possibility left is maximized. Given this, a routing possibility is chosen that maximizes the total number of routing possibilities left. The next train to be routed will be selected randomly from the remaining trains.

Heuristic 1

Step 0. Select and remove a train, say t^*, from the set T.

Step 1. Choose routing possibility $(r, \delta) \in F_{t^*}$ that maximizes the number of trains for which at least one routing possibility is left, i.e. choose (r, δ) to maximize

$$|\{t' \in T: \exists (r, \delta; r', \delta') \in F_{t^*t'}\}|.$$

Out of all possibilities (r, δ) attaining this maximum, randomly choose a routing possibility (r^*, δ^*) that also maximizes the total number of remaining routing possibilities:

$$\sum_{t'\in T} |\{(r', \delta'): (r, \delta; r', \delta') \in F_{t^*t'}\}|.$$

Step 2. If $T = \emptyset$, stop. Otherwise, update the sets F_t and $F_{tt'}$ to reflect the route choice for t^*. Randomly select and remove a new train t^* from the set T, and return to Step 1.

The first heuristic does not take into account how the number of routes that will be available for the remaining trains after a certain train is routed is balanced over the remaining trains. The second heuristic tries to choose, for the train that is under consideration, a routing that balances, as much as possible, the number of routes available for all remaining trains over all remaining trains. The next train to be routed will be the train that has the least number of routing possibilities left. Through this selection this heuristic aims at scheduling the "most critical" train.

Heuristic 2

Step 0. Select and remove a train, say t^*, from the set T.

Step 1. For all routing possibilities $(r, \delta) \in F_{t^*}$, determine how many routing possibilities can still be selected for the remaining trains in T, i.e. determine, for all $t \in T$ and all $(r, \delta) \in F_{t^*}$,

$$N_t(r, \delta) = \max\{\epsilon, |\{(r', \delta'): (r, \delta; r', \delta') \in F_{t^*t}\}|\}.$$

Then choose a routing possibility (r^*, δ^*) for train t^* such that

$$\prod_{t\in T} N_t(r^*, \delta^*) \geq \prod_{t\in T} N_t(r, \delta)$$

for all $(r, \delta) \in F_{t^*}$.

Step 2. If $T = \emptyset$, stop. Otherwise, update the sets F_t and $F_{tt'}$ to reflect the route choice for t^*. Select and remove a new train t^* from T

such that $t^* = \arg\min_{t \in T}\{|F_t| : |F_t| > 0\}$, and return to Step 1.

The constant ϵ with $0 < \epsilon < 1$ appears in Step 1 of this heuristic to force $N_t(r, \delta)$ to be positive for all pairs $(r, \delta) \in F_{t^*}$. This results in an appropriate choice between routing possibilities (r_1, δ_1) and (r_2, δ_2) also if $\Pi_t N_t(r_1, \delta_1)$ and $\Pi_t N_t(r_2, \delta_2)$ would both have been zero otherwise. Note that, given a sufficiently small value of ϵ, the first optimization criterion of Heuristic 1 is automatically included in this heuristic. The motivation for using, in Step 1, the product (instead of the sum) of the number of routing possibilities for each of the trains, is that the former tends to choose a routing possibility (r^*, δ^*) for train t^* for which the values of $N_t(r^*, \delta^*)$ have a smaller variance over $t \in T$ than the latter.

Both heuristics are started $|T|$ times, choosing each train once as the initial train.

3.4. Branch-and-Cut

Within the branch-and-cut procedure a number of subproblems is solved sequentially. A stack containing the so-called *active* subproblems is maintained during the procedure.

Branch-and-cut procedure

Step 0. Initialize the stack by the overall problem.

Step 1. Choose the subproblem from the top of the stack. If the stack is empty, then stop—the best solution found so far is optimal.

Step 2. Solve the LP-relaxation of the subproblem in process. Use a rounding heuristic to obtain a feasible solution. Compare this solution with the best solution found so far. Determine whether the best solution found so far dominates the subproblem in process. If so, stop and return to Step 1.

Step 3. Decide whether it may be useful to tighten the LP-relaxation. If this is the case, then search (for a limited time) for new valid inequalities. If new valid inequalities are found, add them to the subproblem and return to Step 2.

Step 4. Split the subproblem in process into two disjoint subproblems according to some branching rule, and put them onto the stack. Return to Step 1.

We will now explain the steps of the branch-and-cut procedure in more detail.

3.4.1. Step 1

The subproblems will be taken from the stack in LIFO order, corresponding to a depth-first search of the branch-and-bound tree.

3.4.2. Step 2

To determine whether we need to further investigate the subproblem under consideration we compare the (truncated!) objective function value of the LP-relaxation with the objective value of the best feasible solution to the IP-problem found so far. If the former is not larger than the latter, we do not need to search the subtree corresponding to the current subproblem any further. (Note that we can use the truncated objective function value of the LP-solution since the optimal solution will have an integer objective function value.) If the former *is* larger, then we use a rounding heuristic to obtain a (new) feasible solution, which is a candidate for the best solution found so far. The rounding heuristic first orders the variables in non-increasing order of their LP-solution value. Then the variables are considered in this order, and are set to one if possible.

3.4.3. Step 3

Since we search for violated clique inequalities, we may restrict ourselves to a search over the fractional variables only. The reason for this is that variables having the value one can never form a clique of cardinality two (or more) with a variable having a nonzero value. A variable having the value zero is not interesting, since it does not contribute to the value of the clique. Therefore such variables can be added later to turn a clique into a maximal clique.

The fractional variables are ordered in order of non-increasing value. Starting with the first variable, we try to find a clique containing that variable, adding the variables in the given order. If a maximal clique containing only fractional variables is found in this way, and if the sum of the values of the variables in the clique is larger than one (i.e., we have found a *violated clique inequality*), we are done. If not, we start the procedure again with the next variable in the list.

If we have found a violated clique inequality, then it is *lifted* by extending the corresponding clique to a maximal clique. Note that in this step of the algorithm we search for *any* violated clique: not just the ones discussed in Section 2.2. The corresponding valid inequality is added to the problem formulation for all problems at the current node of the branch-and-bound tree and their offspring, even though the valid inequalities are valid for the original problem (and thus for all its subproblems). We repeat the above procedure, after removing the variables contained in the maximal clique, until no more cliques exist. Now the LP-relaxation can be solved again. If no violated cliques are found, we split the current subproblems into new subproblems in Step 4.

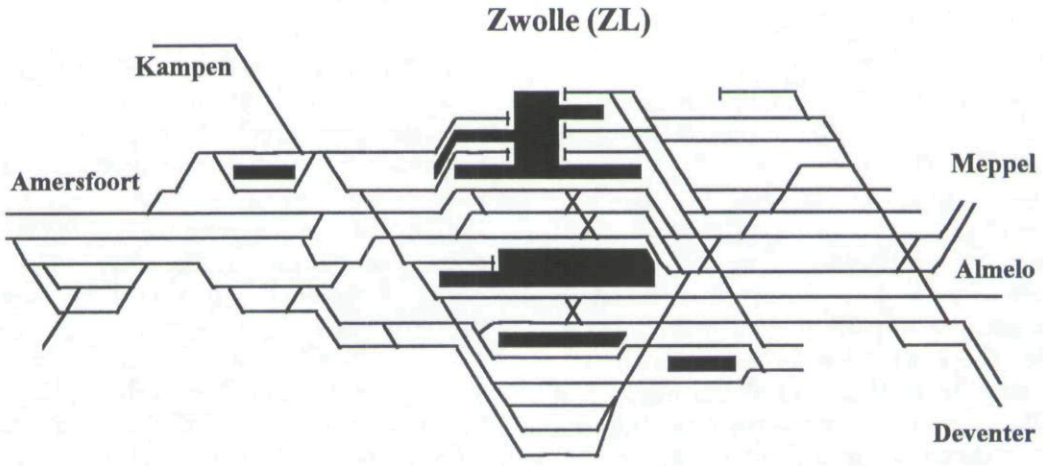

Fig. 2. Infrastructure of the railway station of Zwolle.

3.4.4. Step 4

We propose the following branching rule. Select one of the cliques found by the valid inequality procedure or in Step 3, or generate a new clique if no appropriate clique can be found among those. Create two subproblems: one by setting the total value of the variables in the clique to one, and the other by setting the total value of the variables in the clique to zero. (Note that in the latter subproblem all variables in the clique are set to zero, and can therefore be eliminated from the subproblem.) Both problems are put onto the stack, where we make sure that the first subproblem is put on top of the stack, so that it will be selected next in Step 1. A clique will be selected based on its cardinality and on its value (i.e. the value of the sum of the variables in it). Note that, if a clique is selected with a value of one, the optimal solution to the LP-relaxation of the first subproblem generated is equal to the optimal solution to the LP-relaxation of the parent problem. Therefore, we do not need to solve the LP-relaxation in the next occurrence of Step 2. Actually, we use this observation by choosing a branching clique with value one several times before choosing a clique having value less than one.

The motivation for this branching rule is the following. In many cases a clique represents the use of a particular part of the railway station at a particular time instant. Setting the value of the clique to zero will therefore have the effect of not using that part of the railway station at that time instant. Therefore this branching rule can be expected to be quite powerful.

4. COMPUTATIONAL RESULTS

IN THIS SECTION we present computational results obtained with the algorithm described in the previous section. Several variants are investigated, which were selected after ample discussion with the planners of Dutch railways. Results of 742 different problem instances are presented in this section. The infrastructure that we used as a base case is described in detail in Section 4.1. In Section 4.2 we describe the timetables, and in Section 4.3 we discuss the computational results themselves. This section is concluded with an investigation of the effects of the buffer time on the capacity of a railway station.

4.1. Infrastructure

We use the railway station of Zwolle as the basis for our computational experiments. The railway station Zwolle is a railway station in the north-eastern part of The Netherlands. The railway station has 15 platforms, and roughly 100 signals and 135 sections, about 60 of which contain a switch. The average number of allowable routes per train is 60, while the maximum number is 120. The layout of the railway station of Zwolle is shown in Fig. 2.

Since our data of the station layout lacks information on the exact locations of the sections, while we do have information on the locations of the signals, we make the assumption that each section is released after the buffer time has elapsed following the passing of the first signal following that section by a train. Therefore each section is released somewhat later than in practice, and thus our problem is slightly more restrictive than the actual problem. We compensated for this by choosing the model buffer time equal to zero in most experiments, and thus approximating the real buffer time by this difference in release times.

TABLE I

Characteristics of the Problems

Timetable	Deviations	n	m	n^*	m^*	m_v	m'_v	$\neq 0$
Current	no	1,054	110,889	324	10,842	834	893	13,870
Random18	no	1,036	123,050	304	9,905	714	485	10,563
Current	yes	3,162	1,493,532	1,331	283,456	3,407	581	281,088
Random18	yes	3,108	1,108,000	1,085	143,250	2,705	565	124,450

4.2. Timetables

We use both the current timetable and a number of randomly generated timetables, which are all cyclical with a period of one hour. The current timetable has been in effect (with only minor changes) for many years, and will remain in effect in the near future. This timetable is (almost) equal to past timetables since it is very hard to generate new timetables by hand. Therefore, only slight adjustments have been made from one year to the other.

We use random timetables to simulate the future (and at this point unknown) situation where the (initial) timetable is provided by CADANS. The characteristics of these randomly generated timetables are as follows (all time references are in minutes):

- Nine different combinations of entering and leaving directions for trains that are currently used, are identified.
- No, one, two or three trains are generated for a combination of an entering and a leaving direction.
- The arrival time of one train for a combination of an entering and a leaving direction is drawn uniformly from the set $\{0, \ldots, 59\}$, if at least one train is generated for that combination. The difference between the arrival time and the departure time of the train is drawn uniformly from the set $\{5, \ldots, 10\}$. If two trains are generated for a combination of an entering and a leaving direction, the arrival time and departure time of the second train is 30 minutes later. If three trains are generated for a combination, the arrival and departure times are respectively 20 and 40 minutes later (modulo 60).

We will use as a basis two trains per combination of an entering and a leaving direction. Thus as base case a total of 18 trains are generated, which is the expected usage of railway station Zwolle in the future. As an additional constraint, we require that trains using the same combination of entering and leaving directions use the same platform. This will complicate the routing of trains since the routing possibilities of trains will become more dependent on each other (see also KROON, ROMEIJN and ZWANEVELD, 1995).

We also consider the following two scenarios with respect to the allowable deviations from the arrival and departure times.

1. No deviations are allowed, i.e. $\Delta = \{(0, 0)\}$.
2. Three deviations are allowed, i.e. $\Delta = \{(-1, -1), (0, 0), (1, 1)\}$, i.e. the arrival and departure times can be shifted by one minute either backward or forward in time.

If, in the remainder of this section, we refer to the problem *with* deviations, we will mean the problem with the 3 deviations from scenario 2.

4.3. Performance of the Algorithm

In this section we focus on the performance of our algorithm. In Section 4.3.1 the detailed performance is reported for problem instances from base case scenarios. The results for other problem instances are reported in Section 4.3.2.

4.3.1. Base Case Scenarios

We report the detailed results of our computational experiments for the base case instances. These experiments are conducted both on the current timetable and on 20 randomly generated timetables. This yields 21 timetables for each of the 2 scenarios with respect to the deviations, or 42 base case problems to be solved. All results concerning random timetables are averages over the 20 instances. In Tables I–III the results concerning the randomly generated base case timetables are indicated by 'random18'. In Table I we can see that the problems we obtain for Zwolle have about 1,100 variables (n) and 120,000 node packing constraints (m; as in equations (1) and (2)) if we do not allow for deviations from the original arrival and departure times. If we *do* allow for deviations, we obtain a problem with roughly 3,200 variables and 1,300,000 node packing constraints. The density of the node packing graph is thus about 10%. The dominance rule reduces the number of variables by about 65%, and the number of constraints by 70–90%. In Table I the number of variables and constraints after ap-

TABLE II

Results of the Heuristics

Timetable	Buffer	Deviations	H1	H2	Max(H1, H2)	LP	⌊LP⌋	Opt
Current	0	no	17.4	17.8	18.0	18.0	18.0	18.0
Random18	0	no	16.0	16.1	16.3	16.3	16.3	16.3
Random18	0	yes	17.5	17.7	17.8	17.8	17.8	17.8

plying the dominance rule is denoted by n^* and m^* respectively.

The performance of the valid inequality procedure is satisfactory. It performs well with respect to the reduction of the number of constraints. Without deviations the number of constraints is reduced from about 10,000 (in the form of equation (1) and (2)) to less than 1,000 (see the column labeled m_v in Table I). With deviations, the reduction was even more dramatic, namely from about 150,000 to about 2,700 in the case of randomly generated timetables and from about 280,000 to about 3,500 for the current timetable. Because of the lifting procedure and the fact that the valid inequality procedure is started for each combination of two trains twice, it may happen that some clique inequalities are found more than once. The number of times that this occurs is denoted in Table I by m'_v. Without deviations this occurred in about 40% of cases, while the corresponding number with deviations is about 15%. This difference can be explained by the fact that in the latter case the number of maximal cliques in the graph is much larger than in the first case, and therefore the randomized lifting procedure will find more distinct maximal cliques. The number of nonzero coefficients (see the column labeled '$\neq$ 0' in Table I) in the constraint matrix of the IP formulation remains roughly the same.

Both heuristics are started with each train chosen as the initial train once. No clear conclusion can be drawn on which heuristic performs best. The average solution of both heuristics is about 3% worse than the optimal solution. The results are summarized in Table II. In all cases, the (truncated) optimal solution value of the LP-relaxation of the initial problem (after applying the valid inequality procedure) is equal to the value of the best found heuristic solution, so that the optimal solution is reached without having to resort to the branch-and-cut procedure (illustrating the strength of the preprocessing step, the valid inequality procedure, and the heuristics). Note that for the current timetable all trains can be routed without allowing deviations. Therefore the corresponding problem *with* deviations does not have to be solved.

The computing times (in CPU seconds on a SUN LX workstation, using ANSI C for programming, and CPLEX 2.1 for solving the LP-relaxations) of the various steps of the algorithm are summarized in Table III. All problem instances without deviations were solved within 85 CPU seconds, whereas all problem instances with deviations were solved within 400 CPU seconds. The most time consuming part of the solution procedure is the initialization step (the determination of the sets $F_{tt'}$), as can be observed from Table III.

4.3.2. Analysis of Other Scenarios

Apart from the computational experiments with the base case scenarios, we also investigated a number of other problem instances, all based on randomly generated timetables:

1. Different layouts of the railway station are considered. We represent the different layouts by the number of platforms of the railway station. The number of platforms is varied between 1 and 17. We used the randomly generated timetables with 18 trains and without deviations.
2. Different numbers of trains are considered. The number of trains is varied between 3 and 27. The latter represents a frequency of three for each of the directions of the timetables of Section 4.2. Railway station Zwolle was used for this analysis.

The total number of problem instances generated was 360. The aggregated results of these variants

TABLE III

Computation Times (in CPU Seconds) of the Steps of the Algorithm

Timetable	Buffer	Deviations	Step 0	Step 1	Step 2	Step 3	Step 4	Total
Current	0	no	76.6	2.4	3.3	1.9	0.5	84.6
Random18	0	no	33.0	2.4	2.2	1.4	0.6	39.5
Random18	0	yes	150.6	56.0	36.9	14.8	7.7	266.0

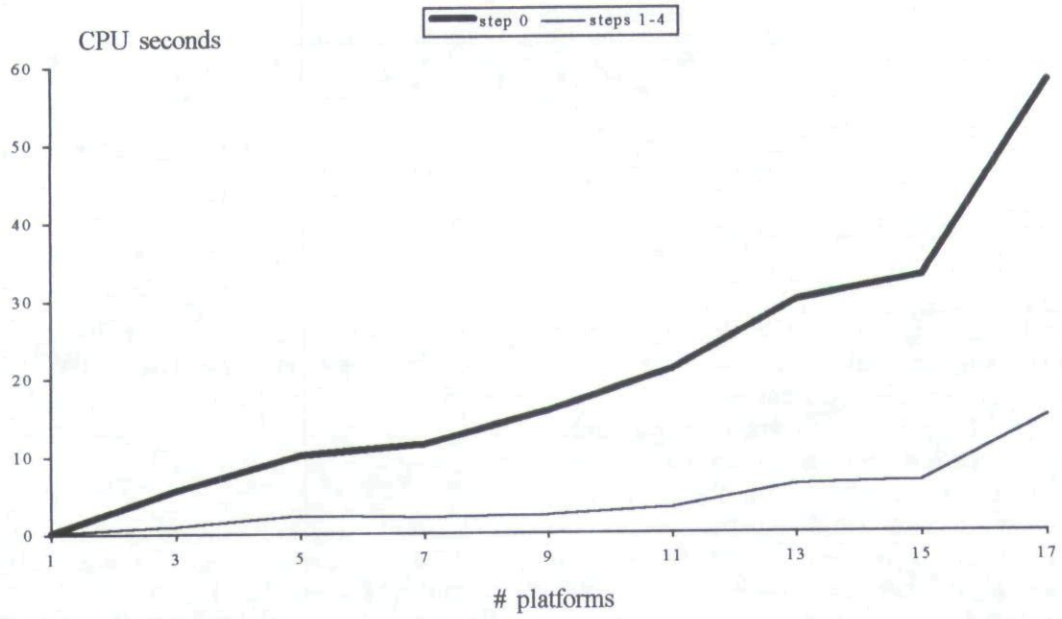

Fig. 3. Average CPU time for different layout designs.

are summarized in, respectively, Figures 3 and 4. We were able to solve each of the problem instances to optimality within 85 CPU seconds. This represents the total computation time for Steps 0–4. For any of the instances, no more than 61 subproblems had to be investigated and no more than 2 useful violated clique inequalities were found during the branch-and-cut phase.

The CPU times reported in both figures are divided into the time required for formulating the problem instances (Step 0) and the time required for solving the problem instances (Steps 1–4). The reported computing times are averages over 20 randomly generated problem instances.

4.4. Buffer Times

The capacity of a railway station depends, apart from the infrastructure and the safety system, on the buffer time that is required. Increasing this buffer time improves the robustness of the resulting

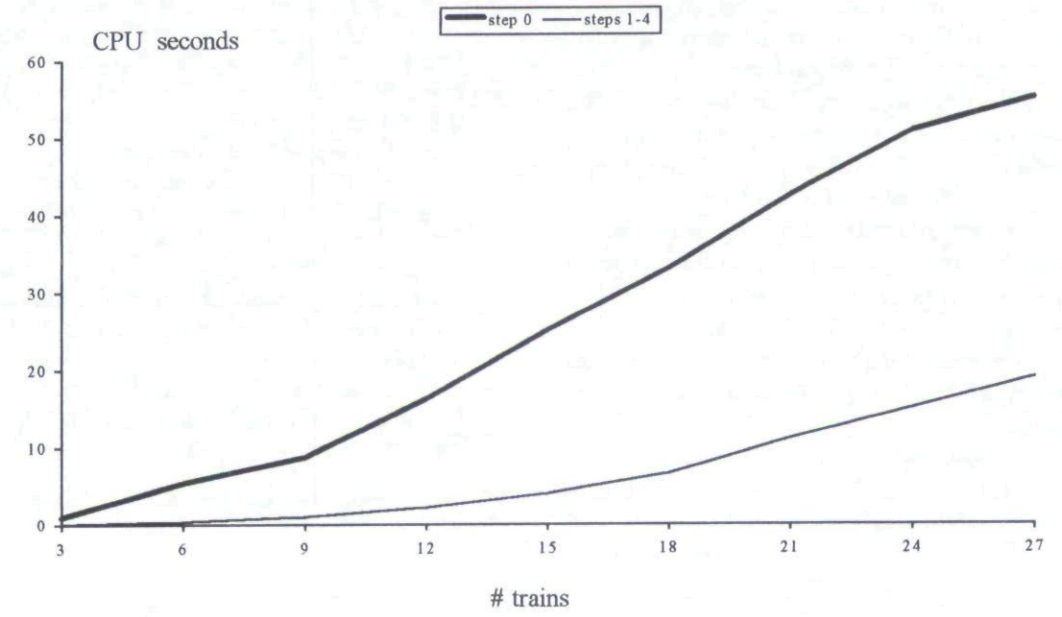

Fig. 4. Average CPU time for different numbers of trains.

TABLE IV

Maximum Number of Trains that Can Be Routed

		Buffer Time									
Timetable	Deviations	0.0	0.3	0.5	1.0	1.5	2.0	2.5	3.0	3.5	5.0
Current	no	18	18	17	15	15	14	12	11	11	10
Random18	no	16.3	15.9	15.1	14.6	14.2	13.9	13.4	13.0	12.6	11.5
Current	yes	18	18	18	18	17	16	15	14	13	11
Random18	yes	17.8	17.8	17.6	17.1	16.5	15.9	14.9	14.2	13.9	12.7

timetable with respect to operational disruptions. However, increasing the buffer time also decreases the capacity of the railway station, since each section on the route of a train is reserved longer by that train. As an illustration we investigated the impact of varying the buffer time on the number of trains that can be routed. The result of this analysis is summarized in Table IV. Railway station Zwolle was used for this analysis. The decrease in capacity can clearly be observed in the results.

As far as the performance of the algorithm is concerned, in about 1% of the 420 problem instances we need to resort to the branch-and-cut procedure. In none of those cases more than 13 subproblems (corresponding to nodes in the branch-and-cut tree) need to be considered. The maximum number of violated clique inequalities, which were generated during the branch-and-cut procedure, is 5.

5. SUMMARY AND CONCLUSIONS

IN THIS PAPER we considered the problem of constructing feasible routings of trains through station yards. We presented a comprehensive description of the problem, and a formulation of the problem as a Node Packing Problem. Next, a branch-and-cut algorithm was outlined and implemented to solve the problem to optimality. The procedure proved effective in solving the routing problem for one of the largest stations in the Dutch railway network.

The numerical experiments showed that the initialization step (i.e., the step of actually formulating the problem instance) forms the major computational burden. Future research should be directed towards improving this initialization step. Another direction for future research involves the inclusion of shunting movements of trains, more complicated service aspects, and other performance indicators into the problem.

Finally, the incorporation of our solution procedure into the system STATIONS, and the interaction mechanism, within DONS, between CADANS (which generates arrival and departure times for each train at each station) and STATIONS requires further investigation.

REFERENCES

Arkin, E. M. and E. B. Silverberg, "Scheduling Jobs with Fixed Start and End Times," *Discrete Applied Mathematics,* **18,** 1–8 (1987).

Carey, M., "A Model for Train Pathing with Choice of Lines, Platforms and Routes," *Transportation Research* **28B,** 333–353 (1994).

Garey, M. R. and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco (1979).

Hoffman, K. L. and M. Padberg, "Solving Airline Crew Scheduling Problems by Branch-and-Cut," *Management Science,* **39,** 657–682 (1993).

Kroon, L. G., H. E. Romeijn, and P. J. Zwaneveld, "Routing Trains Through Railway Stations: Complexity Issues," Working paper 209, Erasmus University Rotterdam, Rotterdam School of Management (1995).

Kroon, L. G. and P. J. Zwaneveld, "STATIONS: Final Report of Phase 1," Working paper 201, Erasmus University Rotterdam, Rotterdam School of Management (1995).

Nemhauser, G. L. and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York (1988).

Padberg, M. and G. Rinaldi, "An Efficient Algorithm for the Minimum Capacity Cut Problem," *Mathematical Programming,* **47,** 19–36 (1990).

Padberg, M. W., "On the Facial Structure of Set Packing Polyhedra," *Mathematical Programming* **5,** 199–215 (1973).

Schrijver, A. and A. Steenbeek, "Dienstregeling Ontwikkeling voor Railned (Timetable Development for Railned," Report Cadans 1.0, C.W.I. Amsterdam, The Netherlands (1994).

Serafini, P. and W. Ukovich. "Mathematical Model for Periodic Scheduling Problems," *SIAM Journal on Discrete Mathematics* **2,** 550–581 (1989).

(Received: April 1995; revision received: August 1995; accepted: September 1995)