Routing trains through a railway station based on a Node Packing model

Routing trains through a railway station based on a Node Packing model

Peter J. Zwaneveld* Leo G. Kroon* Stan P.M. van Hoesel‡

December 9, 1997

Abstract

In this paper we describe the problem of routing trains through a railway station. This routing problem is a subproblem of the automatic generation of timetables for the Dutch railway system.

The problem of routing trains through a railway station is the problem of assigning each of the involved trains to a route through the railway station, given the detailed layout of the railway network within the station and given the arrival and departure times of the trains. When solving this routing problem, several aspects such as capacity, safety, and customer service have to be taken into account.

In this paper we describe this routing problem in terms of a Weighted Node Packing Problem. Furthermore, we describe an algorithm for solving this routing problem to optimality. The algorithm is based on preprocessing, valid inequalities, and a branch-and-cut approach. The preprocessing techniques aim at identifying superfluous nodes which can be removed from the problem instance. The characteristics of the preprocessing techniques with respect to propagation are investigated.

We also present the results of a computational study in which the model, the preprocessing techniques and the algorithm are tested based on data related to the railway stations Arnhem, Hoorn and Utrecht in the Netherlands.

This research is sponsored by Railned and Nederlandse Spoorwegen (Netherlands Railways)

*Rotterdam School of Management, Erasmus University Rotterdam, P.O. Box 1738, NL–3000 DR Rotterdam, The Netherlands; e-mail: pzwaneveld@staf.fbk.eur.nl; lkroon@staf.fbk.eur.nl

‡Department of Quantitative Economics, Faculty of Economics, University of Limburg, P.O. Box 616, NL–6200 MD Maastricht, The Netherlands; e-mail: s.vanhoesel@ke.unimaas.nl.

1 Introduction

In this paper we consider the problem of routing trains through a railway station. This routing problem is a subproblem of the generation of a timetable for a railway company. In a highly utilized and intertwined railway network, such as the Dutch one, the latter problem is an important one, which is very hard to solve. Figure 1 gives a rough representation of the Dutch railway network.

Figure 1: The Dutch railway network.

In order to generate a timetable for a railway company, usually a hierarchical approach is followed. In a first step, a tentative timetable is generated, based on the rough layout of the railway network *between* the railway stations together with the desired lines, their frequencies and their connection requirements. Then, in a second step, it is checked whether the tentative timetable is feasible *within* the railway stations with respect to capacity, safety, and customer service. In order to carry out this feasibility check, detailed routes and schedules for the trains through the railway stations are generated.

Since the generation of a timetable is a task consuming quite a lot of time when carried out manually, the project DONS (Design Of Network Schedules) was initiated recently by Railned[1] and Netherlands Railways. The aim of this project is to develop a DSS (Decision Support System), also called DONS, that will assist the planners of Railned and Netherlands Railways in generating timetables. DONS contains two complementary optimization modules which are linked together by a database module and a graphical user-system-interface. The two optimization modules correspond to the two steps of the timetable generation process mentioned above.

The first optimization module, called CADANS, assists the planners in generating a tentative timetable based on the constraints deduced from the rough layout of the railway network between the stations, the line system, and the connection requirements at the railway stations. The timetable determined by CADANS is cyclic with a cycle length of one hour. CADANS is being developed by Schrijver and Steenbeek [**?**], see also [**?**].

[1]Railned has the task, amongst others, to advise the Dutch Ministry of Traffic with respect to the capacity of the Dutch railway infrastructure that will be necessary in the future.

The second optimization module, called STATIONS, assists the planners in solving the problem of routing trains through a railway station. STATIONS considers the stations one by one. The output of STATIONS is a detailed assignment of trains to routes and platforms within the observed station. Such an assignment serves as a local feasibility check for the tentative timetable generated by CADANS. If not all trains can be routed through the station, then STATIONS also points at the blocking trains. STATIONS is being developed by Zwaneveld et al. [**?**].

In this paper we discuss the model and the algorithm that is used within STATIONS for solving this routing problem to optimality. This paper is complementary to Kroon et al. [**?**] in which the computational complexity of several variants of this routing problem are discussed. Furthermore, this paper is a follow-up of Zwaneveld et al. [**?**]. In the current paper, we improve the model and the algorithm of [**?**] in several ways. In particular, we improve the model by including also shunting decisions and preferences of trains for platforms and routes, and we improve the algorithm by extending the preprocessing techniques. The algorithm described in [**?**] was not sufficient for solving the routing problem within the largest Dutch railway stations such as Amsterdam Central Station (CS) and Utrecht CS. The algorithm presented in the current paper can handle the routing problem for all railway stations in the Netherlands efficiently.

2 Description of operational processes

In this section we give a description of the operational processes within a railway station. This description is necessary for a complete understanding of the next sections. The characteristics of the operational processes that we describe here pertain to the railway system in the Netherlands, which is very similar to most European railway systems.

A railway station can be entered by a train at a number of *entering points*, and it can be left through a number of *leaving points*. We only consider the detailed layout of a railway station between these entering and 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 Utrecht CS are Amersfoort, Amsterdam, Arnhem, 's Hertogenbosch, and Rotterdam, see Figure 2.

The railway infrastructure of a railway station consists of a large number of *track sections* and a number of *platform tracks*. An inbound route is a sequence of sections linking an entering point to a platform track. Similarly, an *outbound* route is a sequence of sections linking a platform track to a leaving point. A platform track may be adjacent to a platform, but may also be a parking track or a track by-passing all platforms. Thus, given the definition of a platform track, each train visits at least one platform track. The platform tracks are part of the corresponding inbound and outbound routes. A complete route is a combination of an inbound route and an outbound route using the same platform track. There are often many different routes between an entering point or leaving point and a platform track, as can be observed in Figure 2.

The *arrival time* of a train is the time at which the train stops at a platform, after

Figure 2: Layout of the railway station Utrecht CS. NB: RICHTINGEN TOEVOEGEN IN PLAATJE

having travelled along an inbound route. Similarly, the *departure time* is the time at which the train leaves the platform along an outbound route. In practice, one can distinguish between the *scheduled* arrival and departure time, and the *actual* arrival and departure time. However, in the context of generating a timetable, only scheduled arrival and departure times are considered.

Clearly, the routing of a train through a railway station depends on the routing of other trains. Most importantly, the safety rules of the Dutch railway system dictate the following procedure, called the *route locking and sectional release system* [**?**, **?**]. As soon as a train arrives at a certain point in the neighbourhood of a station, it claims an inbound route to a platform. Since any track section can be claimed by only one train at the same time, a route is not feasible for a particular train if any section of the route has been claimed already by another train. When a train traverses its claimed route, it sequentially releases each of the track sections comprising the route. A similar procedure is followed for an outbound route, or for a complete route if a train does not stop at a platform.

If a railway station is one of the terminal stations of a train, and the length of the train's standstill interval at the arrival platform exceeds a certain (train-dependent) lower bound, then the train may be shunted towards a parking area in order to release the arrival platform. Later on, the train is shunted back towards its departure platform, which may be different from its arrival platform. Sometimes, a train's departure direction may be different from its arrival direction.

Finally, a number of service considerations have to be taken into account. First, convenience considerations towards the passengers may dictate that certain groups of trains all leave from the same platform. For instance, such a group 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 have to use platforms that are close to each other (preferably *cross-platform*). Moreover, there must be a certain minimum overlap in the time intervals spent at the corresponding platforms.

3 Problem definition

In this section we present a description of the routing problem that is solved by the DSS STATIONS. The involved timetables are cyclic in nature, and represent an operating plan for one hour. As was described in the introduction, these timetables have been generated by CADANS. STATIONS always considers one railway station at the same time. The problem that is solved for this railway station can be stated as follows:

> Given the detailed layout of the involved railway station, and given the scheduled arrival and departure times of a set of trains, STATIONS aims at routing as many trains as possible through the station, taking into account the capacity of the station, the safety system, and several service requirements. The routing of the trains should minimize the number of shunting operations, and it should maximize the total preference for the platforms and routes.

In the problem description a hierarchy of objectives is included. The first objective is to find a feasible route for as many trains as possible. Since we need to comply with the overall timetable, basically all trains have to be routed. However, the problem has been formulated as a maximization problem, because STATIONS should point at the blocking trains if a solution for all trains can not be obtained.

Furthermore, if all trains can be routed through the railway station, then the second objective is to minimize the number of shunting movements. A shunting movement is expensive, since personnel (a train driver and assisting personnel) must be allocated. Furthermore, a shunting movement also uses capacity of the railway station, because the routes towards and from the parking area need to be claimed by the safety system.

The last objective is to maximize the preferences of the trains for certain platforms or routes. The preference of a train for a certain route mainly depends on the *total number* of switches in the route, and on the total number of switches in the *non-preferred* direction.

4 Model formulation

In this section we describe the integer linear programming model that is used to solve the problem of routing trains through a railway station. We start with the characteristics of this problem and with some basic assumptions that are made to model these. We also show that the problem can be described in terms of a Weighted Node Packing Problem (WNPP).

4.1 Assumptions

A first assumption that is used in the model is that for all trains arriving at the railway station both the departure time and the destination are known. Of course, if the station is one of the intermediate stations of a train, then these data are provided by CADANS. However, the latter is not true if the station is one of the terminal stations of a train. In this case the train may be assigned to one of the return trips of the same line, or

the train may be assigned to a trip of a different line into a different direction. In our model, it is assumed that this joining of the outbound trips to the inbound trips has been made a priori. We are forced to make this assumption because, in practice, this joining is determined in a later stage than the routing of the trains through the railway station. However, it should be noted that the model may be modified in such a way that it also covers these joining decisions.

If the railway station is a terminal station of a train and the length of the train's standstill at the arrival platform is too long, then it may be decided to shunt the train temporarily towards a parking area. In our model, for each train that may be shunted, only the decision *whether or not to shunt it* is considered. The detailed shunting movements themselves are not taken into account explicitly. That is, if it has been decided that a train is to be shunted towards a parking area, then the train is assumed to have a standstill of a certain length at its arrival platform, to have a standstill of a certain length at its departure platform, and *to be at a certain parking area* between these standstills. On the other hand, if a train is not to be shunted, then it has a standstill at its arrival platform from its arrival time until its departure time. In this case, the train's departure platform is necessarily the same as its arrival platform.

4.2 Notation

The layout of the railway station consists of a set S of track sections. The set of sections included in a platform track is denoted by $P \subset S$. The set R of routes through the railway station can be determined from the set of sections. We distinguish the set $R^i \subset R$ of *inbound routes* leading from an entering point towards a platform, and the set $R^o \subset R$ of *outbound routes* departing from a platform towards a leaving point. Furthermore, we distinguish the set $R^p \subset R$ of *platform routes*. These platform routes represent the sections of the platform tracks.

The set of trains to be routed through the railway station is denoted by T. The set of trains that may be shunted is denoted by $T^S \subset T$. For each train $t \in T$, CADANS has determined a scheduled arrival time a_t and a scheduled departure time d_t, usually in minutes. We consider these arrival and departure times as given and fixed.

Each train has a known entering point and a known leaving point. The inbound, outbound and platform routes that may be chosen by train t are denoted by R^i_t, R^o_t, and R^p_t respectively. The complete set of all routes allowed for train t is denoted by $R_t = R^i_t \cup R^o_t \cup R^p_t$. The routes that are allowed for train t are selected based on attributes such as the involved arrival and departure direction and the length of the platform.

The safety rules described in the previous section are represented by defining a set $F_{t,t'}$ for each pair of trains $t, t' \in T$. These sets contain allowed combinations of routes (r, r'). In other words, $(r, r') \in F_{t,t'}$ means that, if train t is routed along route r and train t' is routed along route r', then there is no common section of the routes r and r' that is claimed by both trains at the same time. Thus, in order to check whether $(r, r') \in F_{t,t'}$, it is necessary to calculate the exact claim and release times of the track sections of routes r and r' by trains t and t'. These calculations are based on the well-known formulas from the theory of dynamics. For the details of the train dynamics we refer to Zwaneveld [**?**].

It is clear that this model can accommodate a large variety of safety rules (including the current Dutch ones). Actually, many other constraints can be modeled in the same way. First of all, we have to guarantee that for each train t the selected inbound, outbound, and platform route *fit together*. These constraints can be handled by the sets $F_{t,t}$. Indeed, only compatible pairs of routes are included in a set $F_{t,t}$.

Next, convenience considerations towards the passengers may request that all trains in a certain group leave from the same platform, or that certain transfer possibilities between trains are created. Obviously, these constraints and requests can also be modeled by adjusting the sets $F_{t,t'}$ appropriately.

Finally, the preferences of a train for certain platforms or routes are recorded as the preferences of the train for certain routes, since each route includes a platform. The preference of train t for route r is denoted by $\rho_{t,r}$.

4.3 Integer Linear Program

In this section the routing problem is formulated as an integer linear program. The model of the routing problem that we present in this section is an extension of the model described by Zwaneveld et al. [**?**]. In the latter model the decisions whether or not to shunt trains have not been included.

In order to model the routing problem as an integer linear program, a binary decision variable is introduced for each allowed combination of a train and a route. Thus the decision variables are the following:

$$X_{t,r} = \begin{cases} 1 & \text{if train } t \in T \text{ is routed along route } r \in R_t, \\ 0 & \text{otherwise.} \end{cases}$$

Recall that $R_t = R_t^i \cup R_t^o \cup R_t^p$. Thus, a train that may not be shunted should be assigned to an inbound route, a platform route, and an outbound route. A train that may be shunted should be assigned at least to an inbound route and an outbound route. If such a train is actually shunted, then it is assigned to an inbound route and an outbound route only. Otherwise, it is assigned to an inbound route, a platform route, and an outbound route. The following constraints have to be respected by the decision variables:

$$\sum_{r \in R_t^i} X_{t,r} \leq 1 \qquad \text{for all } t \in T, \tag{1}$$

$$\sum_{r \in R_t^o} X_{t,r} \leq 1 \qquad \text{for all } t \in T, \tag{2}$$

$$\sum_{r \in R_t^p} X_{t,r} \leq 1 \qquad \text{for all } t \in T, \tag{3}$$

$$X_{t,r} + X_{t',r'} \leq 1 \qquad \text{for all } t, t' \in T;\ r \in R_t;\ r' \in R_{t'};\ (r, r') \notin F_{t,t'}, \tag{4}$$

$$X_{t,r} \in \{0, 1\} \qquad \text{for all } t \in T;\ r \in R_t. \tag{5}$$

Constraints (1), (2) and (3) ensure that each train t is assigned to at most one inbound, one outbound, and one platform route, respectively. Constraints (4) guarantee that only routes are selected that are allowed with respect to the safety rules, the connection requirements, and the connections of inbound, outbound, and platform routes. Finally, constraints (5) declare the decision variables $X_{t,r}$ as binary.

The objective function of the model has the following form.

$$\max \sum_{t \in T} \sum_{r \in R_t} \rho_{t,r} X_{t,r} \tag{6}$$

As was mentioned already in the problem description, the first objective is to maximize the number of trains that can be routed through the railway station. The second objective is to minimize the number of shunting movements, and the third objective is to maximize the preferences for the routes and platforms. Thus, we consider the selection of a *necessary* route (that is, an inbound or an outbound route, or a platform route for a train that may not be shunted) for a certain train as more valuable than not shunting any number of other trains. Secondly, we prefer to select a necessary route for a certain train over selecting a more preferred route for all other trains. And, finally, we prefer to avoid shunting a certain train over selecting a more preferred route for all other trains. This hierarchical character of the objectives can be reflected in the objective function by choosing the coefficients $\rho_{t,r}$ appropriately, see Zwaneveld [**?**].

4.4 Weighted Node Packing Problem

In this section we show that the problem of routing trains through a railway station can be formulated as a *Weighted Node Packing Problem* (WNPP). Formally, the WNPP 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. Each node $v \in V$ has an weight ρ_v. Then a *node packing* is a set $S \subseteq V$ such that no edge in E joins two nodes in S. The total weight of a node packing S is given by $\sum_{v \in S} \rho_v$. Now the Weighted Node Packing Problem is the problem to find a node packing of maximum total weight.

The WNPP is thus characterized by an undirected graph and corresponding weights for the nodes. For the problem of routing trains through a railway station, we define a node for each variable $X_{t,r}$. The weight of the node $X_{t,r}$ is identical to the objective function value $\rho_{t,r}$. Then the following edges are added to the graph:

(i) Every node $X_{t,r}$ is connected with all nodes associated with the same train t and route type, i.e., inbound, outbound or platform.

(ii) Every pair of nodes $X_{t,r}$ and $X_{t',r'}$ with $(r, r') \notin F_{t,t'}$ is connected with each other.

Here (i) ensures that each train is assigned to at most one route of each type, and (ii) excludes conflicting combinations of trains and routes.

Obviously, a node packing represents a feasible routing of a number of trains through the railway station. Similarly, a node packing of maximum weight represents a feasible routing of trains with maximum value.

5 Relevant sections and routes

In this section we show that only a subset of the sections has to be considered in order to check whether a proposed assignment of trains to routes is feasible from a safety point of view. Furthermore, we also describe that it is not necessary to consider the so-called detour routes in a problem instance.

5.1 Relevant sections

A section is called *relevant* if it contains (i) a switch or (ii) an intersection of tracks, or if it corresponds to (iii) an entering point, (iv) a leaving point, or (v) a platform.

The set of relevant sections is denoted by S^*. Usually S^* contains significantly less elements than the set S of all sections. Note that, by definition of S^*, each section $s \in S \setminus S^*$ is located between two sections $s', s'' \in S^*$, since the first and the last section of each route is a relevant section.

The set of sections within route r is denoted by S_r. The time instants at which a section s is claimed and released by train t over route r is denoted by $S(t,r,s)$ and $F(t,r,s)$ respectively. We apply the following conventions for determining the claim and release times: $0 \leq S(t,r,s) < 60$, $S(t,r,s) \leq F(t,r,s)$ and the length of the reservation interval should be equal to $F(t,r,s) - S(t,r,s)$. These conventions are necessary since we consider a cyclical timetable with a cycle length of one hour. Note that it may happen that $F(t,r,s) \geq 60$. Note further that the length of each time interval has to be less than 60 minutes, due to the cycle length of 60 minutes.

As was mentioned before, the safety rules are represented using a set $F_{t,t'}$ for each pair of trains $t, t' \in T$. Such a set contains the pairs of allowable routes (r, r') for trains t and t'. That is, $(r, r') \in F_{t,t'}$ implies that route r for train t is *compatible* with route r' for train t'. By definition, route r for train t is compatible *from a safety point of view* with route r' for train t' if the following conditions are satisfied:

$\forall s \in S_r \cap S_{r'}$:

$$[S(t,r,s), F(t,r,s)) \cap [S(t',r',s), F(t',r',s)) = \emptyset. \quad (7)$$

$$[S(t,r,s), F(t,r,s)) \cap [S(t',r',s) - 60, F(t',r',s) - 60) = \emptyset. \quad (8)$$

$$[S(t,r,s) - 60, F(t,r,s) - 60) \cap [S(t',r',s), F(t',r',s)) = \emptyset. \quad (9)$$

Lemma 1 shows that only *relevant* sections have to be taken into account for determining the feasibility from a safety point of view.

Lemma 1 *Route $r \in R_t$ for train t is compatible from a safety point of view with route $r' \in R_{t'}$ for train t' if and only if $\forall s \in S_r \cap S_{r'} \cap S^*$ equations (7), (8), and (9) are satisfied.*

Proof. Since the 'only if' part of the statement is obvious, we only prove the 'if' part. To that end, suppose conditions (7), (8) and (9) are satisfied for all $s \in S_r \cap S_{r'} \cap S^*$, and choose a section $s_2 \in (S_r \cap S_{r'}) \setminus S^*$.

Without loss of generality $S(t', r', s_2) \geq S(t, r, s_2)$. Thus, condition (9) is valid as soon as condition (7) is valid. Let s_1 be the previous relevant section of route r before section s_2, and let s_3 be the next relevant section of this route. Note that route r' also contains the sections s_1 and s_3. Train t' may traverse the sections $\{s_1, s_2, s_3\}$ in the order (s_1, s_2, s_3), or in the order (s_3, s_2, s_1). These possibilities are considered separately:

Order (s_1, s_2, s_3): Conditions (7) and (9) are valid because $F(t, r, s_2) \leq S(t', r', s_2)$. This follows from $F(t, r, s_2) \leq F(t, r, s_3)$, $S(t', r', s_2) = S(t', r', s_3)$, and, by assumption, $F(t, r, s_3) \leq S(t', r', s_3)$. Condition (8) is valid because $F(t', r', s_2) - 60 \leq S(t, r, s_2)$. This follows from $F(t', r', s_2) \leq F(t', r', s_3)$, $S(t, r, s_2) = S(t, r, s_3)$, and, by assumption, $F(t, r, s_3) - 60 \leq S(t, r, s_3)$. The claim and release times are shown in Figure 3.

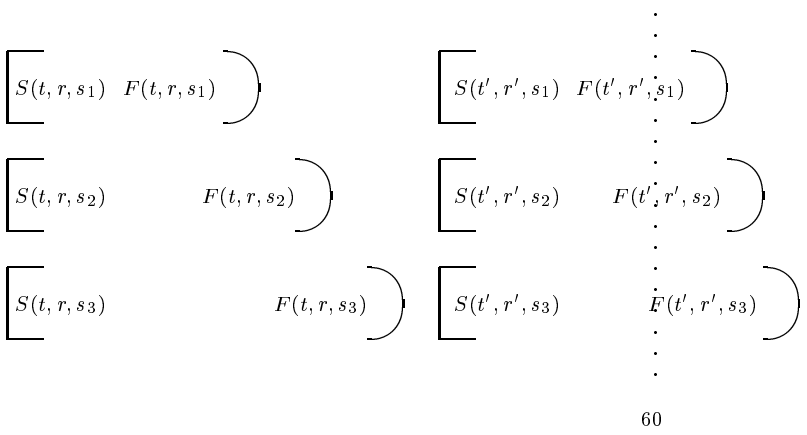

Figure 3: Claim and release times for routes in identical directions.

Order (s_3, s_2, s_1): Conditions (7) and (9) are valid because $F(t, r, s_2) \leq S(t', r', s_2)$. This follows from $F(t, r, s_2) \leq F(t, r, s_3)$, $S(t', r', s_2) = S(t', r', s_3)$, and, by assumption, $F(t, r, s_3) \leq S(t', r', s_3)$. Condition (8) is valid because $F(t', r', s_2) - 60 \leq S(t, r, s_2)$. This follows from $F(t', r', s_2) \leq F(t', r', s_1)$, $S(t, r, s_2) = S(t, r, s_1)$, and, by assumption, $F(t, r, s_1) - 60 \leq S(t, r, s_1)$. The claim and release times are illustrated in Figure 4. It follows that equations (7), (8), and (9) are satisfied for all $s \in S_r \cap S_{r'}$. □

5.2 Detour routes

A route r is called a *detour route* if it is a detour in comparison with one of the other routes. The latter route is called the corresponding *straight route*. The detour routes are easily identified with the use of the relevant sections that were defined in the previous section. By definition, route r is a detour route if the following condition holds:

$$\exists r' \ : \ ((S_{r'} \cap S^*) \subset (S_r \cap S^*)) \ \wedge \ (\forall t \in T \ : \ \rho_{t,r} \leq \rho_{t,r'}).$$

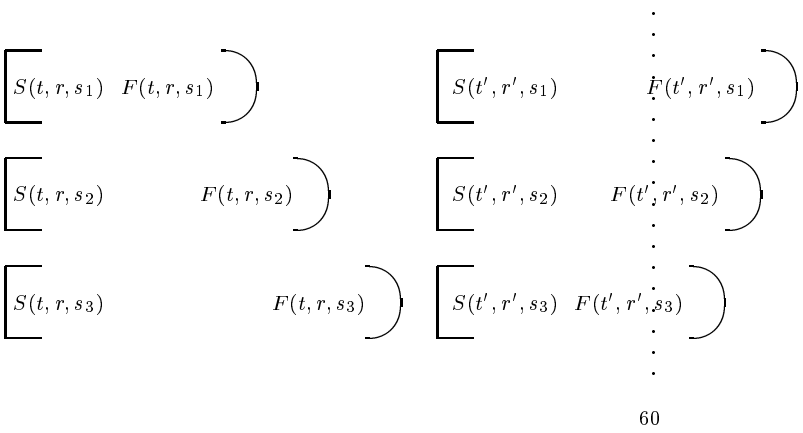

Figure 4: Claim and release times for routes in opposite directions.

In other words, route r is a detour route in comparison with route r' if route r' contains only a subset of the relevant sections of route r, and if all trains prefer route r' over route r. An example of a detour route and a corresponding straight route is given in Figure 5.

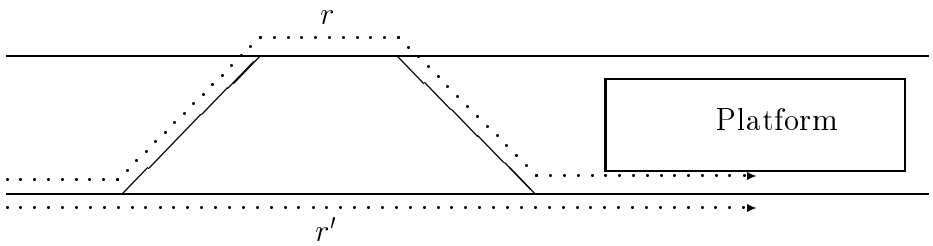

Figure 5: An example of a detour route r and a corresponding straight route r'.

In practice, trains should not be assigned to a detour route, since the reservation of a detour route puts a greater claim on the capacity of the relevant sections than the reservation of a corresponding straight route. Therefore, the decision variables involving a detour route are superfluous and can be deleted from the model. As will be explained in Section 6.1.1, a detour route is also *node-dominated* by a straight route. Therefore the detour route condition is a special case of the node-dominance condition.

6 An LP-based algorithm for the WNPP

In this section, we present a solution method for the problem of routing trains through a railway station. This method is based on the formulation of the problem in terms of the Weighted Node Packing Problem (WNPP) as described earlier.

Thus, we have a graph $G = (V, E)$ in which each node represents a combination of a train and a (partial) route. An edge between two nodes indicates that the two

corresponding combinations of routes and trains conflict with each other. Furthermore, we have a positive weight function on the nodes: $\rho : V \to I\!R^+$. In this graph G we have to find a maximum weight node packing or independent set.

In the sequel we also use the following notation. For any subset I of the nodes, $\rho(I) = \sum_{i \in I} \rho_i$. Furthermore, for a set S and an element i, let $S + i = S \cup \{i\}$ and $S - i = S \backslash \{i\}$. Finally, $G - i$ is the graph G from which node i has been removed, together with the edges incident to node i. The neighbours of node i are denoted by $N(i) = \{j \in V : \{i, j\} \in E\}$. Moreover, $N_d(i)$ is the set of nodes at distance at most d, where distance is measured as the number of edges on the shortest path. Thus, $N_0(i) = \{i\}$, $N_1(i)$ is the set of neighbours of i, i included, etcetera. These definitions are extended to sets; for example, $N(S)$ is the set of nodes connected to some node of S.

We solve the WNPP with a branch-and-cut method using the LP- relaxation of the problem, strengthened with valid inequalities, as an upper bound in each of the subproblems that are created in the branching process. Before this process starts, we try to remove as many variables from the model formulation as possible, using several preprocessing techniques. Although all developed techniques may be used in all subproblems of the branch-and-cut tree, only one of the techniques is applied to all subproblems.

In the sequel, the components of the algorithm are described. We start with the ideas developed for preprocessing, then we describe the branch-and-cut procedure. Illustrative and real-life computational results are added to each of the described components.

6.1 Preprocessing

Instance reduction by preprocessing receives more and more attention as an indispensable part of optimization methods. It is usually cheap in computation time and it may be very effective on specific (practical) problems. For example, instance reduction for the (unweighted version of the) Node Packing Problem has been applied by Hoffman and Padberg [**?**] in their crew-scheduling application. More complicated ideas have been used by Mannino and Sassano [**?**], who show that certain substructures of a graph can be replaced by smaller structures with a simultaneous reduction in stability number, the maximum size of a node packing. The conditions on such substructures, however, are not only restrictive, they are also not easily generalized to the weighted version of the Node Packing Problem. Therefore, we decided to develop some preprocessing methods that are applicable to our instances of the WNPP.

All methods presented here aim at removing nodes from the graph, i.e., fixing variables at the value zero, based on combinatorial arguments. The basic idea is to show that a certain node i is dominated, i.e., for each solution of the WNPP containing node i we can find an alternative solution that is at least as good and that does not contain node i. Although this dominance is hard to check in general, we consider some special cases, which drastically reduce the number of variables in our instances. And, equally important, the linear programming relaxation can be solved much faster, implying a really better bound.

6.1.1 Node-dominance

The node-dominance technique determines whether a node i can be replaced by a single other node j in all node packings containing node i. If this is possible, then the variable corresponding to node i can be removed from the problem instance. The definition of node-dominance is as follows:

Definition 2 *Node $i \in V$ is node-dominated in the graph $G = (V, E)$ by node $j \in V$ if for each node packing S containing node i, node i can be replaced by node j without weight reduction. That is, the node packing $S' = S + j - i$ is feasible and $\rho(S) \leq \rho(S')$.*

Clearly, if node j node-dominates node i, then we can remove node i from G without reducing the optimal value of the underlying WNPP. This simple idea of node-dominance turns out to have a dramatic effect on the size of the instances of the problem of routing trains through a railway station. Due to the special structure of these instances, simple node-dominance reduces the number of variables with more than 70%. Moreover, it is easily checked, since there are transparent necessary and sufficient conditions specifying which node node-dominates another. This is described in the following theorem.

Theorem 3 *Let i and j be two nodes in $G = (V, E)$. Node i is node- dominated by node j, if and only if*

1. *$\rho_i \leq \rho_j$ (weight condition).*
2. *$N_1(j) \subseteq N_1(i)$ (neighbour condition).*

Proof. The necessity of both conditions is easily established. If $\rho_i > \rho_j$, then any node packing containing node i deteriorates by replacing node i by node j. Furthermore, if $k \in N_1(j) \backslash N_1(i)$ then $\{i, k\} \notin E$. Therefore, the solution consisting of the node packing $S = \{i, k\}$ does not have the property that node i can be replaced by node j, since the distance of node j and node k is at most 1: if $k = j$, then there is only one node left in the new node packing and thus it has smaller weight; otherwise $\{j, k\} \in E$, and thus $S' = \{j, k\}$ is not a feasible node packing.

Both conditions are also sufficient. Suppose that nodes i and j satisfy them. Consider any node packing S containing node i. Then $S' = S - i + j$ is a feasible node packing: indeed, suppose that for $k \in S - i$ we have $\{j, k\} \in E$. Then $k \in N_1(j) \subseteq N_1(i)$, and therefore $\{i, k\} \in E$. This is a contradiction with the feasibility of S. Finally, $\rho(S') = \rho(S) - \rho_i + \rho_j \geq \rho(S)$. So, we can replace node i by node j. □

Note that the neighbour condition implies that $\{i, j\} \in E$, since $j \in N_1(j) \subseteq N_1(i)$.

By the removal of a node-dominated node, other nodes, that were not node-dominated earlier, may become node-dominated. This effect is called propagation. Propagation of node-dominance is illustrated in the example shown in Figure 6.

In this example all nodes have the same weight. Thus, the node-dominance relation is equivalent to the neighbour condition. Node 1 node-dominates node 2, and there is no

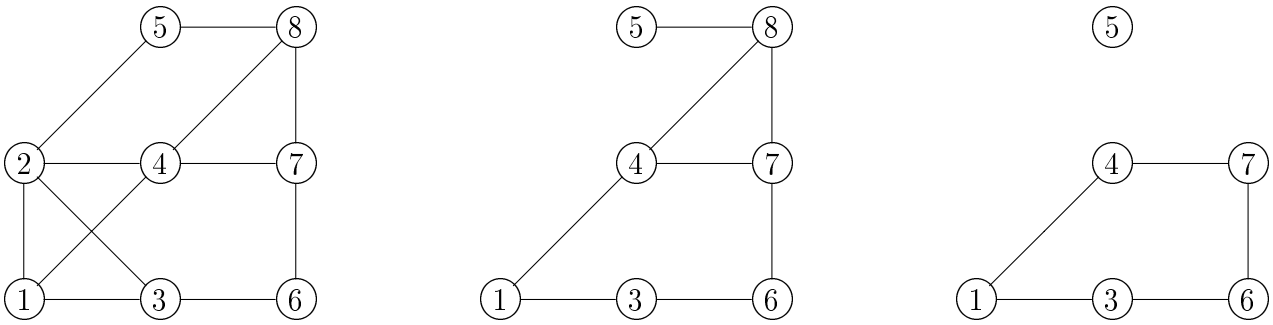

Figure 6: Propagation of node-dominance.

other pair involved in a node-dominance relation. However, after the removal of node 2 from the graph, node 5 node-dominates 8, and thus node 8 can be removed as well.
Next, we will prove that the final result of the node-dominance technique is not dependent on the order in which node-dominated nodes are removed from the graph. We will prove this using the following two lemmas. The first lemma proves the transitivity of node-dominance, and the second one shows that a node-dominated node loses this property during the process of deleting nodes only under very special circumstances.

Lemma 4 *Consider three different nodes i, j, and k. If node i node-dominates node j and node j node- dominates node k, then node i node-dominates node k.*

Proof. $\rho_i \geq \rho_j \geq \rho_k$, and $N_1(i) \subseteq N_1(j) \subseteq N_1(k)$. □

The following lemma plays a key role in the proof of the order independence.

Lemma 5 *Consider two nodes i and j, which are both node- dominated in a graph G. Then node i is node-dominated in $G - j$, or nodes i and j are identical in G, i.e., $\rho_i = \rho_j$ and $N_1(i) = N_1(j)$.*

Proof. If $\exists k \neq j$ such that node k node-dominates node i in G, then clearly node i is node-dominated in $G - j$ by node k. Otherwise, node i is only node-dominated by node j. Moreover, node j is only node-dominated by node i, since otherwise, by transitivity, node i would have been node- dominated by a node other than node j. However, this implies that $\rho_i = \rho_j$ and $N_1(i) = N_1(j)$. □

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with weight functions ρ_1 and ρ_2 on the nodes are said to be *weight-isomorphic*, if there is a one-to-one mapping f of the nodes from G_1 to G_2 that preserves adjacency and weight, i.e., for all $i \in V_1 : \rho_i = \rho_{f(i)}$ and for all $i, j \in V_1 : \{i, j\} \in E_1$ if and only if $\{f(i), f(j)\} \in E_2$. Note that weight-isomorphism is a symmetric relation.

Theorem 6 *Consider a problem instance of the WNPP on a graph G. Consider two maximal sequences of node-dominated nodes $a = (a_1, \ldots, a_l)$ and $b = (b_1, \ldots, b_m)$, i.e., $G - a$ and $G - b$ do not contain any node-dominated nodes. Then $G - a$ and $G - b$ are weight-isomorphic.*

Proof. Consider a node-minimal counter example (G, ρ). We will show that we can transform the sequence b to a sequence b' such that a_1 is the first node in b', and $G - b$ and $G - b'$ are weight-isomorphic. This proves the theorem, since $G - a_1$ contains fewer nodes than G, and thus $G - a$ and $G - b'$ are weight-isomorphic.

First, suppose that $a_1 \notin b$. Since a_1 is node-dominated in G and not in $G - b$, there is an index i with $1 \leq i \leq m$ such that a_1 is node-dominated in $G - (b_1, \ldots, b_{i-1})$ and not in $G - (b_1, \ldots, b_i)$. By Lemma 5, this means that a_1 and b_i are identical in $G - (b_1, \ldots, b_{i-1})$. Therefore, if we exchange b_i and a_1 in b, then the resulting graph is weight-isomorphic with $G - b$. Thus we may suppose that $a_1 \in b$.

If $a_1 = b_1$, then we are done since we assumed that (G, ρ) was a node-minimal counter example. Therefore, suppose that $a_1 \neq b_1$, say $a_1 = b_j$ for some $j > 1$. We will show that we can exchange b_j and b_i for some $i < j$. Doing this repeatedly leads to $a_1 = b_1$, which finishes the proof by the argument given above. If b_j is not node-dominated in $G - (b_1, \ldots, b_{j-2})$, then there is an index i with $1 \leq i \leq j - 2$ such that a_1 is node-dominated in $G - (b_1, \ldots, b_{i-1})$ and not node-dominated in $G - (b_1, \ldots, b_i)$. By Lemma 5, this means that a_1 and b_i are identical in $G - (b_1, \ldots, b_{i-1})$. Therefore, we can exchange b_i and b_j in b without making any actual changes. Finally, suppose that b_j is node-dominated in $G - (b_1, \ldots, b_{j-2})$. Then both b_{j-1} and b_j are node-dominated. If b_{j-1} is node-dominated in $G - (b_1, \ldots, b_{j-2}, b_j)$, then we can exchange b_{j-1} and b_j. Otherwise, by Lemma 5, b_{j-1} and b_j are identical, and again we can exchange b_{j-1} and b_j. Concluding, we can rearrange the order of the nodes in b so that a_1 is the first one. □

Implementing node-dominance is fairly straightforward. If we order the nodes in order of non-increasing weight, breaking ties by ordering the nodes in order of non- decreasing degree, then it suffices to check whether a node node-dominates later nodes in the list. Checking node-dominance for each node pair takes $\mathcal{O}(|V|)$ time. The overall running time is $\mathcal{O}(|E|\ |V|)$, since we only need to check neighbouring pairs of nodes. As this may still be quite a lot of time, and propagation may force us to restart the process, we reduce the running time by comparing only nodes corresponding to the same train, the same bound (inbound, outbound, or platform), and the same platform.

6.1.2 Set-dominance

Set-dominance determines whether a node i can be replaced by one of the nodes of a given set R in all node packings containing node i. If this is possible, then node i can be removed from the problem instance. This idea generalizes node-dominance in the sense that the node that replaces node i can be taken arbitrarily from the set R.

Definition 7 *A node $i \in V$ is set-dominated in graph $G = (V, E)$ by a set $R \subset V - i$ if for each feasible node packing S containing node i, there is a node $j \in R$ such that the*

set $S' = S - i + j$ is a feasible node packing and has weight greater than or equal to the weight of S, i.e., $S' = S + j - i$ is a feasible node packing, and $\rho(S) \leq \rho(S')$.

Obviously, node-dominance is a special case of set-dominance, where the set R consists of one node only. Below we give an example that shows that set-dominance truly generalizes node-dominance.

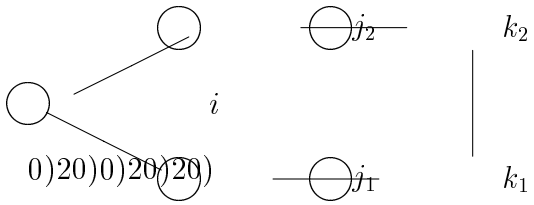

Figure 7: Set-dominance.

In this example all nodes have the same weight. There are two maximal node packings containing node i, namely, $\{1, 4\}$ and $\{1, 5\}$. In $\{1, 4\}$ we can replace node 1 by node 3 and in $\{1, 5\}$ we can replace node 1 by node 2. Thus in both cases we do not need node 1. After node 1 has been removed, node 4 is node-dominated by node 2, and node 5 is node-dominated by node 3. So, this instance can be solved completely by dominance techniques, although in the initial graph no node is node- dominated by another one.

Clearly, if the set R set-dominates node i, then we can remove node i from G, without reducing the optimal value of the WNPP. Set-dominance may reduce the size of the instances of the problem of routing trains through a railway station with a substantial amount, even after the application of node- dominance. However, it is more difficult to find conditions that are necessary and sufficient for a node to be set-dominated.

On the other hand, we can restrict the number of sets to be taken into account for checking set-dominance of a node i. The following lemma shows that the weights of the elements of the set R should be sufficient.

Lemma 8 $\forall_{j \in R: \rho_j < \rho_i}$ *if R set-dominates node i, then $R - j$ set-dominates node i.*

In the sequel we consider only sets R in which all elements have weight at least ρ_i. Using this result, the definition of set-dominance becomes equivalent with the following statement: for each node packing $S \subseteq N(R) - N_1(i)$, there is a node $j \in R$ such that $N_1(j) \cap S = \emptyset$. The following lemma shows that we can restrict ourselves to sets containing only neighbours of node i.

Lemma 9 $\forall_{j \in R \setminus N(i)}$ *if R set-dominates node i, then $R - j$ set-dominates node i.*

Proof. Suppose that R set-dominates node i and that $R - j$ does not set-dominate node i, for some $j \in R \setminus N(i)$. Then, according to the definition of set-dominance, there is a node packing S containing node i, such that $S' = S - i + j$ is a feasible node packing. Moreover, there is no other node k in $R - j$ such that $S - i + k$ is a feasible node packing.

However, since nodes i and j are not connected, $S + j$ is a feasible node packing as well. By the fact that R set-dominates node i, there should be a node $k \in R - j$ that can replace node i in $S + j$. This is a contradiction, since this particular node k would then be able to replace node i in S as well. □

From this lemma it follows that we can restrict set-dominant sets to contain only neighbours of the set-dominated node. The following lemma shows that we can restrict set-dominant sets to all neighbours of node i with sufficient weight, i.e., to $R^i_{\max} = \{j \in N(i) : \ \rho_j \geq \rho_i\}$.

Lemma 10 $\forall_{j \in N(i):\ \rho_j \geq \rho_i}$ *if R set- dominates node i, then $R + j$ set-dominates node i.*

Proof. The condition that for each node packing S including node i this node can be replaced by an element of R holds trivially for $R + j$ if it holds for R. Note that $j \notin S$ since node j is connected to node i. □

Contrary to the node-dominance technique, the final result of the set-dominance technique may be sequence dependent. However, the result of the set-dominance technique is sequence independent if all weights of connected nodes are different. The proof of this result can be found in Zwaneveld [**?**]. An example of a sequence dependent problem instance for set- dominance is given in Figure 8.

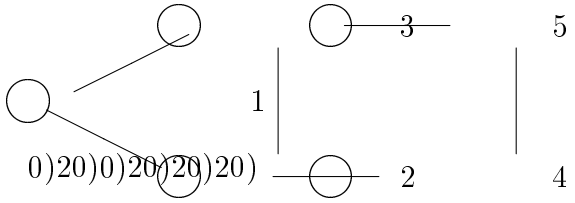

Figure 8: Sequence dependent propagation.

Figure 9: The results.

In the above example all nodes have the same weight. Clearly, node 1 node-dominates nodes 2 and 3. After the removal of nodes 2 and 3, node 4 node-dominates node 5, so that node 5 can be removed as well, and we are left with two unconnected nodes. On the other hand, the set $\{2, 3\}$ set-dominates node 1. If we remove node 1 we are finished. Thus

the different orders result in one of the graphs shown in Figure 9, in which no further set-dominance can be detected.

We conclude the description of set-dominance with an algorithm that determines for a node whether it is set- dominated by a subset of its neighbours. Consider an arbitrary node i for which we want to determine whether it is set-dominated. Let $R^i_{\max} = \{j \in N_1(i) : \rho_j \geq \rho_i\}$. The recursive algorithm described below determines whether or not node i is set-dominated by $R^i_{\max}$.

The algorithm maintains a partial node packing S in $N_2(i) - N_1(i)$. If S is a so-called *blocking* node packing, i.e., a node packing in which each node is connected with a node from $R^i_{\max}$, then the algorithm stops: we have a proof that node i is not set-dominated by $R^i_{\max}$, since it can not be replaced by a node of $R^i_{\max}$ in the node packing $S+i$. Otherwise, we try to show that any extension of S has no edge with at least one node from $R^i_{\max}$, i.e., no extension of S is blocking.

In this algorithm we use three sets. Besides the set S, the set $R \subseteq R^i_{\max}$ is used to show that any extension of S is set-dominated. Finally, we maintain a set O which contains the nodes from $N_2(i) - N_1(i)$ that need not be considered in extensions of S.

The procedure AUGMENT(S, O, R) determines if any extension of S with nodes from $N_2(i) - N_1(i) - O$ is a blocking node packing. Its first call is with empty sets only: AUGMENT($\emptyset$, $\emptyset$, $\emptyset$).

AUGMENT(S, O, R)

if $\forall_{j \in R^i_{\max} - R} : S \cap N(j) \neq \emptyset$
then STOP: S is a blocking node packing
fi

Let $j \in R^i_{\max} - R : S \cap N(j) = \emptyset$
Let $K = \{k \in (N_2(i) - N_1(i)) \cap N(j) : k \notin N(S) \cup O\}$

while $K \neq \emptyset$
do
    Select $k \in K$;
    AUGMENT($S + k$,O,$R + j$);
    $O+ = k$; $K- = k$;
od

return The extensions of S (not using O) are not blocking.

In this algorithm, we select a node j from $R^i_{\max}$ and we consider all extensions of S using neighbours of node j. If all neighbours of node j have been checked, then we are finished since any other extension of S contains only nodes that are not connected to node j. Note

that the selection of node $j \in R^i_{\max} - R$ is not specified. If we select node j such that K is minimal, then we determine immediately whether a node is node-dominated by another one. Thus, node-dominance is a special case of set-dominance, as was noted earlier.

The running time of the algorithm to determine set-dominance for a single node by a set R is $\mathcal{O}(|E|^{|R|})$. We implemented two versions of the set-dominance technique. In the first implementation we restrict the set R of alternatives to nodes corresponding to the same train, the same bound and the use of the same platform. In the other implementation the set R holds nodes corresponding to the same train and the same bound. In the latter implementation we allow a maximum CPU time of 0.01 seconds per variable. For test instances the obtained number of set-dominated variables was not increased when we allowed a maximum of 300 seconds per variable.

6.1.3 Iterative set-dominance

If the set-dominance algorithm fails to find set-dominated nodes, then for each node i and corresponding set $R^i_{\max}$, there is at least one blocking node packing S. We may, however, be able to alter such a blocking node packing without changing its weight or extendibility, in such a way that the resulting node packing is not blocking anymore. Consider the following example.

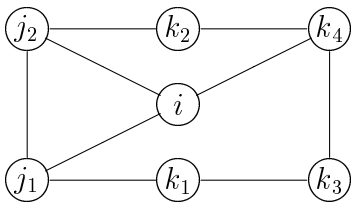

Figure 10: Iterative set-dominance.

In the figure above, the nodes 1, 2, and 3 have weight 2; the other nodes have weight 1. No node is set-dominated by some set of other nodes. If one tests the set-dominance of node 1 by the set $\{2, 3\}$, then the node packing S causing trouble would be $\{4, 5\}$. However, suppose that we want to replace node 1 by node 2. In the node packing $\{4, 5\}$, we may replace node 4 by node 6. Thereafter we may replace node 1 by node 2. Thus it may be concluded that the node packing $\{4, 5\}$ is not a problem after all, and thus node 1 can be removed from the graph.

In general, we can check for each blocking node packing S whether it is possible to replace some of its nodes by other nodes until the obtained node packing is not blocking anymore. Although we do use this technique in the implementation of our algorithm, we refer the reader interested in the further details to Zwaneveld [**?**].

6.1.4 Combining nodes

Two nodes are combined into one node if the fact that one of the nodes is selected in the node packing implies that the other node must be selected in the node packing as well,

and vice versa. The arguments for combining nodes are based upon the interpretation of the problem as the problem of routing trains through a railway station. We identify the following situations:

- If only one inbound and one outbound route lead to and from a platform for a specific train, then the nodes corresponding to the inbound, outbound and platform route can be combined into one new variable.

- Each remaining node corresponding to a platform route for a specific train can be combined with a compatible node corresponding to an inbound route for the same train.

- If only one node corresponding to an inbound (outbound) route towards (from) a specific platform for a particular train remains, then this node can be combined with all compatible nodes corresponding to an outbound (inbound) route for the considered platform and train.

6.1.5 Effects of Preprocessing

In this section we present computational results for the preprocessing techniques that were described in the previous sections. The computational results are obtained on a SUN LX workstation with 50 Mhz, and we used ANSI C for programming. The computational results are presented for three railway stations in the Netherlands, namely Arnhem, Hoorn, and Utrecht CS.

- Arnhem (Ah) is a medium sized railway station in the eastern part of the Netherlands. This station is visited by about 40 trains per hour. Arnhem has 16 platform and through-going tracks and consists of 102 track sections of which 74 are relevant.

- Hoorn (Hn) is a small railway station in the north- western part of the Netherlands. This station is visited by about 12 trains per hour. Hoorn has 6 platform and through-going tracks and consists of 61 track sections of which 32 are relevant.

- Utrecht CS (Ut) is the largest railway station in the Netherlands, located in the center of the country. This station is visited by about 80 trains per hour. Utrecht CS has 40 platform and through-going tracks, and consists of 264 track sections of which 201 are relevant, see Figure 2.

The problem instances are taken from studies of the Capacity Planning department of Railned, which involve future railway network designs for the year 2005. For each railway station three instances (timetables) have been generated and analyzed. It turned out that for each railway station the sizes of these problem instances were almost identical.

The sequence in which the preprocessing techniques are applied is as follows:

1. Detour routes.

2. Node-dominance.

3. Combining variables.

4. Set-dominance with the set R restricted to nodes representing the same train, the same bound, and the use of the same platform. We refer to this set R as R^{small}.

5. Iterative set-dominance.

6. Set-dominance with the set R restricted to nodes representing the same train and the same bound. We refer to this set R as R^{large}. The maximum CPU time spent on each variable is set to 0.01 seconds.

All preprocessing techniques are repeated whenever a variable has been removed. Before any technique is repeated, all previously mentioned techniques are executed as well.

It turns out that the order in which the preprocessing techniques are executed influences the number of removed variables. However, the size of the remaining problem instances after applying the techniques in different orderings did not differ more than 4% from each other. The sequence that we apply is determined by the effect of each preprocessing technique and the CPU time necessary for executing each technique.

Table 1 presents the effects of the preprocessing techniques with their corresponding CPU times. For each railway station the results are an average over the three studied problem instances. The percentages in this table are in comparison with the average initial problem size.

Station	Ah		Hn		Ut		Average	
	nodes	*CPU*	*nodes*	*CPU*	*nodes*	*CPU*	*perc.*	*CPU*
initial	5930		165		9328		100%	
detour	-3053	3	-9	0.01	-1025	3	-26%	2
node	-2389	22	-38	0.02	-6695	117	-59%	46
comb. var	-168	1.6	-71	0.2	-779	10	-7%	4
set, small	-47	0.2	-0.3	0.00	-55	1	-1%	0.4
iter. set	-3	0.1	-0	0.00	-36	1	-0.3%	0.4
set, large	-33	0.4	-2	0.01	-32	10	-0.4%	3
final	215	27	45	0.2	706	142	6%	56

Table 1: Cumulative effect on the number of variables by each preprocessing technique and the CPU times in seconds.

In most cases the preprocessing techniques are able to reduce the initial problem size by over 90%. Furthermore, the effects of the preprocessing techniques differ over the railway stations. In general, set-dominance and iterating set-dominance have the largest effect on the larger problem instances. Node-dominance is the largest consumer of CPU time, because it has to consider much more nodes than the techniques which are applied later on. Propagation is very important for node-dominance: about 25% of the node-dominated nodes are node-dominated due to propagation. Node-dominance is repeated up to 10 times to remove all node-dominated variables. Table 1 shows that the increase of the CPU time for the larger problems is acceptable.

6.2 The complete algorithm

In this section we describe the complete algorithm for solving the problem of routing trains through a railway station, based on the formulation of the problem as a WNPP. Very generally, the algorithm reads as follows:

The complete algorithm

1. Initialization: generate all routing possibilities (t, f), and determine all conflicting combinations of routing possibilities.

2. Preprocessing: try to reduce the problem instance in advance, thereby using the techniques that were described earlier in this section.

3. Formulate the routing problem as an integer programming problem and tighten its LP-relaxation by adding valid inequalities.

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

In Step 0 of the algorithm the (useful) routing possibilities for all trains as well as the admissible pairs of them are determined. This step involves many travel time calculations and searching for overlapping reservation intervals of sections. For the implementation details of this step we refer to Zwaneveld [**?**]. The computing time for performing this step is linear in the number of conflicting routing possibilities or, equivalently, in the number of edges of the node packing formulation.

In Step 1 we apply the previously explained preprocessing techniques in the order as described in Section 6.1.5.

Step 2 involves the derivation of many *clique inequalities*. As is well-known, the LP-relaxation of a WNPP is very bad in general, and needs improvement by the addition of valid inequalities. Standard valid inequalities for the Node Packing Problem are clique inequalities and (lifted) odd-hole inequalities, see Padberg [**?**]. Hoffman and Padberg [**?**], and Nemhauser and Sigismondi [**?**] give implementations based on these inequalities. Due to the special structure of our problem (a high edge- density) we will be concerned with generating (violated) clique inequalities in successive LP-relaxations only. For more details on the generation of these clique inequalities we refer to Zwaneveld et al. [**?**].

The corresponding cliques cover all edges, such that the inequalities (4) (and (1), (2), (3)) can be removed from the problem formulation. Cliques are initially based on nodes representing two trains with the same direction. Each clique is lifted by extending it to a maximal clique. The sequence in which the connected variables are considered is chosen randomly, based on a uniform distribution over these variables.

The branch-and-cut procedure in Step 3 may require an exponential amount of time. The main aspects of this procedure are:

1. Applying set-dominance with the set R^{small}.

2. Solving the LP-relaxation.

3. Applying a rounding heuristic to the solution of the LP-relaxation to obtain a feasible solution.

4. Searching for violated clique inequalities.

5. Selecting the branching variable.

All LP-relaxations are solved from scratch by the LP-solver of CPLEX 3.0 [**?**]. Since the objective function coefficients of the variables differ strongly from each other, degeneracy hardly occurs. Moreover, the strong differences between these coefficients help the LP-solver to select the optimal set of basic variables and therefore they speed up the calculation of the LP-relaxation.

The rounding heuristic greedily constructs a node packing by considering the variables in order of decreasing value. When all variables with a strictly positive value have been considered, the constructed node packing is lifted to a maximal node packing by considering all other variables (i.e. with value zero in the solution of the LP-relaxation) in random order.

The search for violated clique inequalities can be restricted to the set of variables with a fractional value. The fractional variables are ordered in order of non- increasing value. Starting with the first variable, we try to find a violated clique inequality containing this variable by adding the variables in the given order. If a clique of fractional variables is found with total value exceeding one, then we have found a violated clique inequality. Subsequently, we lift this clique randomly to a maximal clique, remove the fractional variables from the list, and continue the search. If no violated clique with the first variable from the list exists, then we remove this variable from the list and we also continue our search. When all variables from the list have been considered as part of a violated clique, then we add the corresponding constraints to the LP formulation and recalculate the LP-relaxation.

The branching scheme follows the depth-first-search strategy. The branching variable is the variable with a fractional value in the solution to the LP-relaxation and with the smallest index.

6.3 Computational results of the complete algorithm

The results of the complete algorithm with respect to the problem instances that were described in Section 6.1.5 are presented in Table 2. Again the computational results are obtained on a SUN LX workstation with 50 Mhz.

On average, the clique inequalities reduce the number of constraints from 4733 initial constraints, see m in Table 2, to 532, see m_v. The initial constraints are the constraints (1), (2), (3), and (4). The resulting LP-formulation is quite tight, as can be observed from the small difference, see $\lfloor \text{LP} \rfloor$ - Opt, between the truncated solution value of the LP- relaxation and the optimal solution. We are allowed to use the truncated value of the LP- solution, since we restricted the objective function coefficients to integer values. In the table, 'size tree' indicates the number of branches which are investigated in the branch-and-cut tree. The maximum depth of the tree is denoted by 'depth tree' and the

number of useful violated clique inequalities, which existed in the branch-and-cut tree is indicated by 'cuts'. The total number of set-dominated nodes is indicated by '# set in tree'. The number of trains in the timetable is denoted by '$|T|$'. The number of routed trains in the optimal solution is denoted by '$|T|$ routed'. The CPU time for the branch-and-cut algorithm is indicated by 'CPU b&c'. The total CPU time, see 'CPU total', includes the time to read all input data from file, as well as the time to generate the problem instance and the clique inequalities.

Timetable	Ah	Hn	Ut	Average
m	2452	118	11629	4733
m_v	324	40	1231	532
size tree	4	1	21	9
depth tree	2	1	3	2
cuts	1	0	4	2
# set in tree	1	0	50	0.3%
$\lfloor$LP$\rfloor$ - Opt	6995	2664	23993	11217
Opt	2103514	581879	13899361	5528251
$\|T\|$	39	15.3	79	44
$\|T\|$ routed	29	14.6	75	40
CPU prep. (s)	27	0.2	142	56
CPU b&c (s)	4	0.1	116	40
CPU total (s)	37	0.4	287	108

Table 2: Results of the Cutting-plane algorithm.

Furthermore, it can be mentioned that the described model formulation and algorithm have been implemented in the DSS STATIONS that was described in Section 1. This DSS is in operation within the Dutch organizations for railway transport: Railned and Netherlands Railways. Up to the mid of 1997, STATIONS has been used to solve over 500 practical problem instances. The planners reported an average computing time of about 1 minute for the larger railway stations. These computing times are quite acceptable for the planners when they use STATIONS interactively.

Altogether, it can be concluded that the complete algorithm is very well able to solve even the largest practical problem instances that occur in the Dutch railway network.

7 Summary and conclusion

In this paper we explained the problem of routing trains through a railway station as well as its practical context. We described the relevant aspects of the operational processes that have to be taken into account to solve the routing problem in practice. We showed that these aspects can adequately be taken into account in the formulation of the problem as a Weighted Node Packing Problem (WNPP). Furthermore, we developed a branch-and-cut algorithm to solve the problem to optimality.

We described several preprocessing techniques to reduce the problem size a priori. Three techniques, namely node-dominance, set-dominance, and iterating set-dominance, are applicable to all WNPPs. These techniques aim to prove that a certain node can be replaced by some other node in all feasible solutions. We derived several characteristics of the techniques with respect to propagation. The techniques had a large effect on the sizes of the problem instances for which we presented computational results in this paper. Due to the high effectiveness of the preprocessing techniques, all problem instances that we studied could be solved to optimality. An interesting direction for further research is the application of these techniques to other problems that can be formulated as a WNPP.

Based on their experiences with the DSS STATIONS so far, the planners have formulated several desirable extensions of the system. One extension involves the possibility to solve the routing problem for two railway stations at the same time. This is particularly interesting if two railway stations are located close to each other. Another extension involves the possibility to use different safety systems for route locking and sectional release. A final extension involves the possibility that STATIONS provides the planners with suggestions for the modification of the arrival and departure times of trains if a feasible solution for all trains with the given arrival and departure times does not exist, see Zwaneveld [**?**].

Although the mentioned extensions do not change the mathematical formulation of the routing problem, they may enlarge the instance of the WNPP to be solved. Therefore, it will be interesting to further investigate the range of the sizes of the routing problem that can be handled by the described techniques.