A non-compact formulation for job-shop scheduling problems in traffic management

Leonardo Lamorgese * Carlo Mannino†

Abstract

A central problem in traffic management is that of scheduling the movements of vehicles so as to minimize the cost of the schedule. It arises in important applications such as train timetabling, rescheduling, delay and disruption management, airplane surface routing, runway scheduling, air traffic control and more. This problem can be modeled as a job-shop scheduling problem. We introduce a new MILP formulation for job-shop scheduling which is alternative to classical approaches, namely big-M and time-indexed formulations. It does not make use of artificially large coefficients and its constraints correspond to basic graph structures, such as paths, cycles and trees. The new formulation can be obtained by strengthening and lifting the constraints of a classical Benders' reformulation. Tests on a large set of real-life instances from train rescheduling show the new approach performs on average better than our previous approaches based on big-M formulations and particularly better on a class of instances with non-convex costs very common in the practice.

1 Introduction

A central problem in transportation is that of routing and scheduling the movements of vehicles so as to minimize the cost of the schedule. It arises, for instance, in train timetabling [10], rescheduling (or dispatching) [9], delay and disruption management [12], runway scheduling [6], airplane surface routing [16], and many more. These are both online (real-time) and off-line problems. For instance, train timetabling amounts to finding a long term (official) schedule of all trains movements, whereas in its online version, train rescheduling, the timetable is modified in real-time in order to react and recover from unexpected deviations from the plan. In both cases, the ability to find good or optimal schedules in short computing time is crucial: for real-time problems, because decisions must be taken and implemented in a few seconds; for off-line problems, because

*OptRail, Rome, Italy, e-mail: leonardo.lamorgese@optrail.com
†Sintef Digital and University of Oslo, e-mail: carlo.mannino@sintef.no

the scheduling problem must generally be solved repeatedly in the overall solution framework.

For the fixed routing case, the problem boils down to finding a minimum cost conflict-free schedule, i.e. a schedule where potential conflicts are prevented by a correct timing of the vehicles on the shared resources. Namely, whenever two vehicles want to access the same, non-sharable resource - e.g. a runway or a station platform - a decision about *who goes first* [28] must be taken. In [22] this problem is shown to belong to the class of job-shop scheduling problems with blocking and no-wait constraints, a class which generalizes canonical job-shop scheduling (see [26]).

This decision is modeled as a disjunction of two linear constraints in the scheduling variables [3, 5]. So, a basic mathematical representation of the scheduling problem involves continuous variables representing times, (time-precedence) linear constraints associated with single vehicles, and disjunctive (precedence) linear constraints associated with pairs of vehicles. In this paper we focus on this basic mathematical representation, which is the central building block for modelling and solving real-life scheduling problems. There are two major ways to linearize disjunctive formulations (see [27]), namely by means of big-M formulations or by *time-indexed* formulations[1]. In big-M formulations each disjunction is represented with two linear constraints and one (or two) binary variable(s), where the variables select which of the two constraints must be satisfied by the schedule (the other constraint becomes redundant). Although big-M formulations are generally very compact, notoriously they tend to produce weak relaxations and consequently large search trees. Time-indexed formulations were introduced precisely to address this major drawback: the time horizon is subdivided into periods, with a binary variable for each original scheduling variable and each time period ([14]). Disjunctive constraints are expressed by cardinality constraints in these variables. Time-indexed formulations typically return much stronger bounds than big-M formulations. However this comes at the cost of increasing the number of variables and constraints, at times dramatically, particularly for problems with large time horizons and fine time granularity (typically the case for traffic management problems in transportation). In our experience for train rescheduling [20], the trade-off is significantly in favour of big-M formulations. Indeed in [20], for time-indexed formulations, the increase in computing time to solve the linear relaxation due to larger instance size is not even closely compensated by the reduced size of the search tree. For this reason, in the recent development of a master/slave approach to train rescheduling ([17, 18]) we resorted to the big-M formulation for modeling and solving the master (job-shop scheduling) problem.

Despite the effectiveness of this approach, which led to successful real-life implementations, the "big-M trap" (i.e. the intrinsic limitation of the approach) remained behind the corner. To speed up the solution of the subproblems and/or tackle larger instances required strengthening the big-M formulation, which has been attempted for decades with little or no success. In very recent years, some authors have attempted to overcome this issue by exploring new paths. For instance, Bonami et al. [8] developed a

[1]See [8] for a discussion on how to handle generic indicator constraints

new, general method borrowed from Non-Linear Programming. However, according to the figures presented in [8], the application to scheduling problems appears to lead only to small improvements. A different approach, explicitly designed for train rescheduling, was presented in [15]. The authors show this approach can solve some real-life instances in the very short running times dictated by real-time. However, the approach is heuristic, and the exact delayed row generation algorithm presented in [17, 18] seems to lead to larger speeds up (although computational results cannot be directly compared as they relate to different sets of proprietary instances).

In this paper we develop a new, non-compact formulation for the job-shop scheduling problem with separable, non-decreasing non-negative cost function [31]. This new formulation allows to overcome the limitations of previous models and avoid the big-M trap (it does not make use of artificial large coefficients). The constraints in the formulation correspond to basic structures like paths, cycles and trees in an associated graph. We show how such constraints may be derived by strengthening and lifting from a classical Benders' reformulation [23]. We test the new approach on a large set of real-life instances from train rescheduling and show that it compares favorably with respect to the (effective) approach previously developed in [17, 18]. The average improvement in terms of computation time is small for convex costs, but becomes significant for the non-convex case (see Section 5). As mentioned, obtaining such speed-up is (clearly) particularly relevant in solving real-time traffic management problems (e.g. runway sequencing, train re-scheduling, etc.), but can also be important when attacking feasibility off-line problems, like the class of train timetabling problems presented in [19].

Summarizing, in ([17, 18]) we apply the so called Logic Benders' reformulation (see [11]) in order to decompose the original problem into a "line problem" (i.e. relative to the railway line) and a number of station problems (i.e. associated with each station), with the line problem being a big-M formulation (of a scheduling problem). In this paper we only focus on the line problem defined in the above decomposition, and proceeding from *the classic* Benders' reformulation we show how to get rid of the big-M constraints.

As mentioned earlier, our problem is a job-shop scheduling problem. Each vehicle is a job, each network resource (e.g. roads, crossings, waypoints, tracks) is a machine and the (sequence of) movements of each vehicle through the network are *operations*. Now, job-shop scheduling problems in traffic management have some common features for which the new approach is particularly suited. First, the number of disjunctions is limited, typically in the order of, or smaller, than the number of operations (whereas in principle this number could grow quadratically). Secondly, and perhaps more importantly, the objective function is associated with the schedule of few operations, sometimes even only one operation per job (i.e. the arrival time, or delay, of the vehicle at some specific target location). However, the approach developed in this paper can in principle be applied to a more general class of job-shop scheduling problems.

2 A big-M formulation for the job-shop scheduling problem in traffic management

In this section we formally introduce the job-shop scheduling problem considered in this paper. We then introduce a natural Mixed Integer Linear Program (MILP) for the problem. Finally we transform the natural formulation in a way which is suitable for the developments in the following sections.

The movement of a vehicle through a transportation network can be modeled as a sequence of *atomic movements*, corresponding to the occupation of a specific network resource. For example, a train in a railway network runs through a sequence of tracks; an airplane in the airdrome runs through a sequence of taxiways; a vehicle on a road through road segments, etc. The ordered sequence of atomic movements associated with a vehicle is called *route*. Let O be the set of atomic movements of all vehicles. With every atomic movement $u \in O$ we associate the quantity $t_u \in \mathbb{R}_+$ representing the time the vehicle starts atomic movement u (i.e. the vehicle starts occupying the corresponding network resource). The vector $t \in \mathbb{R}_+^{|O|}$ is called *schedule*. If we consider two successive atomic movements $u, v \in O$ on the route of a vehicle, then we have that the schedule satisfies the time-precedence (linear) constraint

$$t_v - t_u \geq \lambda_{uv} \tag{1}$$

where λ_{uv} is the minimum time necessary for the vehicle to complete the atomic movement u. Now, consider two distinct vehicles A and B, both going through a non-sharable network resource R_1, as in Figure 1.[2]

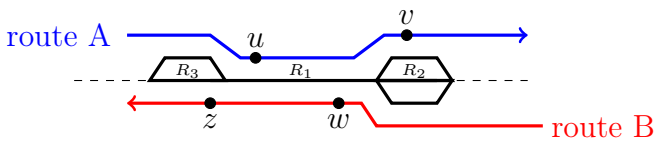

Figure 1: The routes of two trains running through the single track section R_1 in opposite directions.

Let $u \in O$ be the atomic movement on the route of A corresponding to the occupation of resource R_1, and let v be the next movement - corresponding to the occupation of a new resource R_2. Similarly, let w be the movement corresponding to resource R_1 on the route of B and let z be the next movement (corresponding to a resource R_3 which may be distinct from R_2). Since the resource R_1 is non-sharable, either vehicle A precedes B on R_1 or viceversa. In the first case, A enters the next resource R_2 before

[2]Note that here we consider a basic case of incompatible movements. More complicated cases can be easily handled in a similar fashion.

B enters R_1, that is $t_w \geq t_v$; in the second case B enters R_3 before A enters R_1, and we have $t_u \geq t_z$. Therefore, the schedule t must satisfy the following disjunctive constraint:

$$(t_w - t_v \geq 0) \bigvee (t_u - t_z \geq 0) \tag{2}$$

Observe that the above disjunction consists of two terms, each term being a (linear) precedence constraint in the schedule variables[3].

A job-shop scheduling problem in traffic management can be represented by a directed graph whose nodes correspond to atomic movements, whereas the edges correspond to linear precedence constraints and to disjunctions. In the graph of Figure 2 each disjunction is represented by a pair of directed edges (dotted edges in the picture), one for each term of the disjunction

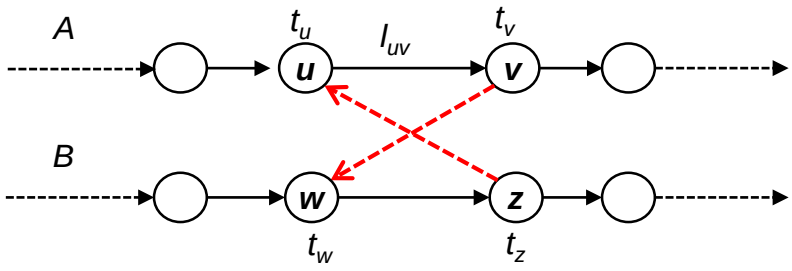

Figure 2: A directed graph representing a job-shop scheduling instance

We are now able to state the problem addressed in this paper:

Problem 2.1 *Let O be a set of atomic movements, let t_u be the starting time of movement $u \in O$ and let $O^2 = O \times O$ be the set of the ordered pairs of atomic movements. Let $A \subseteq O^2$ be a set of (indices of) linear precedence constraints with associated r.h.s. $\lambda \in I\!R_+^{|A|}$. Let $D \subseteq \{\{e, f\} : e, f \in O^2, e \neq f\}$ be the set of (indices of) disjunctive precedence constraints. Finally, let $c : I\!R_+^{|O|} \rightarrow I\!R$ be the cost of the schedule. Find a schedule $t \in I\!R_+^{|O|}$ satisfying all simple precedence constraint (1), for $(u, v) \in A$, and all disjunctive precedence constraints (2), for $\{(v, w), (z, u)\} \in D$, and such that $c(t)$ is minimized.*

Next, we introduce the set $F = \{(u, v) \in O^2 : \{(u, v), (w, z)\} \in D\}$ of the simple precedence constraints contained in one or more disjunctions of D. For the sake of simplicity, in the remainder of the paper we assume F to be a simple set - the generalization is immediate.

[3]In principle, the r.h.s. of each term may be non-zero. We prefer here this version in order to simplify the following discussion

Problem 2.1 can be easily represented as a Mixed Integer Program (MIP). For each $(u, v) \in F$ we introduce a binary variable y_{uv}, with $y_{uv} = 1$ if the schedule satisfies $t_v - t_u \geq 0$. Then Problem 2.1 can be written as the following MIP:

$$
\begin{array}{lll}
\min & c(t) & \\
s.t. & & \\
(i) & t_v - t_u \geq \lambda_{uv}, & (u,v) \in A \\
(ii) & t_v - t_u \geq M(y_{uv} - 1) & (u,v) \in F, \\
(iii) & y_{uv} + y_{wz} = 1, & \{(u,v),(w,z)\} \in D, \\
(iv) & t_u \geq 0, & u \in O, \\
 & t \in \mathbb{R}^{|O|} \quad y \in \{0,1\}^{|F|} &
\end{array}
\tag{3}
$$

where M is a suitably large constant and $\lambda_{uv} \geq 0$ for all $uv \in A$. Constraints $(3.ii)$ and $(3.iii)$ ensure that at least one simple precedence constraint in every disjunction is satisfied by t.

Program (3) is the "natural" formulation for our problem, adopted, in a form or another, by many studies on this topic. If $c(t)$ is linear, and all the binary variables are fixed to 0 or 1, (3) reduces to a linear program with non-negative variables and a very nice structure: all constraints are time-precedence constraints, of the form $t_y - t_x \geq \lambda_{xy}$. This very desirable property will be exploited in the remain of the paper. To this end, we need to manipulate (3) in order to reduce it to a form which is more convenient to our purposes. First, we will show how we can consider a convex piece-wise linear cost function and still maintain the wanted property (Program (8)). Then (8) is simply rewritten in the compact form (9). Finally, we apply a linear transformation to (the coefficient matrix of) (9) so to obtain precisely the dual (BF) of a max-cost flow problem.

2.1 Cost function and MILP formulation.

We assume the cost function to be non-negative, piecewise linear and convex, as depicted in Figure 3. Later on in the paper we show that convexity can be relaxed without challenging the validity of the approach.

For every $u \in O$ we let the breakpoints of c_u in correspondence to (breakpoint) times $T_u^1, \ldots, T_u^{b(u)}$ - with $b(u)$ being the number of breakpoints. We let $c_u^1, \ldots, c_u^{b(u)}$ be the corresponding non-negative slope increments of the cost function. Namely, for $t_u \in [T_u^i, T_u^{i+1})$, $i = 1, \ldots, b(u)$, the slope will be $c_u^1 + \cdots + c_u^i$ (we let $T_u^{b(i)+1} = +\infty$). For every $u \in O$ and times $T_u^1, \ldots, T_u^{b(u)}$, we associate variables $\delta_u^1, \ldots, \delta_u^{b(u)}$ (referred to as *delay variables*) representing the delay with respect to each breakpoint time. Namely, we have

$$
\delta_u^i = \max(0, t_u - T_u^i), \qquad i = 1, \ldots, b(u) \tag{4}
$$

Then the cost of starting the atomic movement u at time t_u is defined as

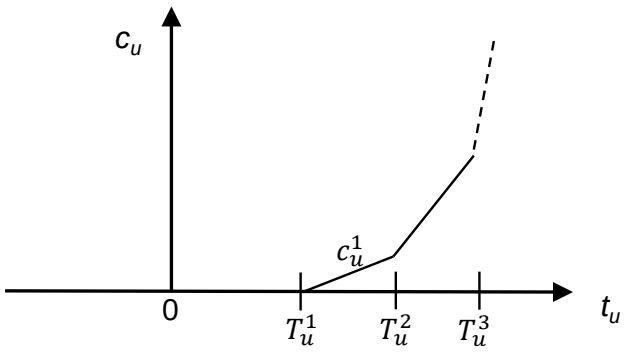

Figure 3: Convex piecewise linear cost function.

$$c_u^1 \delta_u^1 + \cdots + c_u^{b(u)} \delta_u^{b(u)} \tag{5}$$

Since all slope increments are non-negative and we want to minimize the overall cost, we can replace (4) with the following family of constraints in the delay variables:

$$\delta_u^i \geq 0, \qquad i = 1, \ldots, b(u) \tag{6}$$

$$\delta_u^i \geq t_u - T_u^i \qquad i = 1, \ldots, b(u) \tag{7}$$

Problem (3) can now be rewritten as a Mixed Integer Linear Program (MILP):

$$\begin{array}{lll}
\min & \sum_{u \in O} \sum_{q=1,\ldots b(u)} c_u^1 \delta_u^1 + c_u^2 \delta_u^2 + \cdots + c_u^{b(u)} \delta_u^{b(u)} & \\
s.t. & & \\
(i) & t_v - t_u \geq \lambda_{uv}, & (u,v) \in A \\
(ii) & t_v - t_u \geq M(y_{uv} - 1) & (u,v) \in F, \\
(iii) & y_{uv} + y_{wz} = 1, & \{(u,v),(w,z)\} \in D, \\
(iv) & \delta_u^i - t_u \geq -T_u^i, & u \in O, i = 1, \ldots, b(u), \\
(v) & \delta_u^i \geq 0, & u \in O, i = 1, \ldots, b(u), \\
(vi) & t_u \geq 0, & u \in O, \\
 & t \in \mathbb{R}^{|O|}, \quad \delta_u \in I\!R^{|b(u)|}, u \in O, \quad y \in \{0,1\}^{|F|} &
\end{array} \tag{8}$$

The next remarks follow immediately from the definitions.

Remark 2.2 *The following facts hold:*

1. *Only delay variables have non-zero cost coefficients in the objective of (8).*

2. *Each delay variable appears in only two constraints of (8), namely one constraint of type (8.iv) and one non-negativity constraint (8.v)*

We now simply rewrite (8) in a more compact form. In particular, since all the constraints involving continuous variables are either non-negative constraints, or time-precedence constraint we can consider a new variable vector $\bar{t} = (t, \delta)$. Renaming all constants accordingly, we can rewrite (8) as

$$\begin{array}{lll} \min & \sum_{u \in \bar{V}} \bar{k}_u \bar{t}_u & \\ s.t. & & \\ (i) & \bar{t}_v - \bar{t}_u \geq \bar{\lambda}_{uv}, & (u,v) \in \bar{A} \\ (ii) & \bar{t}_v - \bar{t}_u \geq M(y_{uv} - 1) & (u,v) \in F, \\ (iii) & y_{uv} + y_{wz} = 1, & \{(u,v),(w,z)\} \in D, \\ (iv) & \bar{t}_u \geq 0, & u \in \bar{V}, \\ & \bar{t} \in \mathbb{R}^{|\bar{V}|}, \quad y \in \{0,1\}^{|F|} & \end{array} \tag{9}$$

where $\bar{V}$ is the set of variable indexes, $\bar{k}_u \geq 0$ is the cost of variable $\bar{t}_u$ for $u \in \bar{V}$ and $\bar{A}$ is the (extended) family of simple precedence constraints, corresponding to constraints $(8.i)$ and $(8.iv)$. The r.h.s. vector $\bar{\lambda}$ extends vector λ to include the (negative of the) breakpoint times appearing in the r.h.s. of $(8.iv)$. With some abuse of notation we still denote by D the set of (renamed) disjunctive pairs and by F the corresponding set of simple precedence constraints appearing in the disjunctions. Observe that the original delay variables δ of (8) correspond to a subset $V_c \subset V$ of the new set of indices. Again, we have $\bar{k}_u \neq 0$ only for variables $\bar{t}_u$ with $u \in V_c$.

Finally, we further transform the above MILP by introducing a new fictitious movement o, the *origin*. We let $V = \bar{V} \cup \{o\}$, introduce real variables s_u for $u \in V$, and let $\bar{t}_u = s_u - s_o$, for $u \in V \setminus \{o\}$. The transformed problem reads as:

$$\begin{array}{lll} \min & \sum_{u \in V} k_u s_u & \\ s.t. & & \\ (i) & s_v - s_u \geq l_{uv}, & (u,v) \in E \\ (ii) & s_v - s_u \geq M(y_{uv} - 1) & (u,v) \in F, \\ (iii) & y_{uv} + y_{wz} = 1, & \{(u,v),(w,z)\} \in D, \\ & s \in \mathbb{R}^{|V|}, \quad y \in \{0,1\}^{|F|} & \end{array} \tag{BF}$$

where E is the resulting family of simple precedence constraints. It is not difficult to see that $k_u = \bar{k}_u$ for $u \in V \setminus \{o\}$ and $k_o = -(\sum_{u \in V \setminus \{o\}} k_u)$. Also, by construction, we have $k_u \geq 0$ for all $u \in V \setminus \{o\}$. Observe that the s variables are unconstrained in sign. Indeed, the original set of non-negativity constraints $(9.iv)$ is mapped into the family of simple precedence constraints $s_u - s_o \geq 0$, for $u \in V$. Finally, we include all simple precedence constraints in the set E and l extends the original vector λ accordingly. For

convenience, we let $l \in I\!R^{|E \cup F|}$, with $l_{uv} = 0$ for $(u, v) \in F$. Note that $V_c \subset \bar{V}$, and for $u \in V_c$, delay variable $\bar{t}_u$ corresponds to (delay) variable s_u. Next remark derives immediately from Remark 2.2:

Remark 2.3 *For $u \in V_c$ delay variable s_u appears in exactly two constraints of (BF):*

1. *constraint $s_u - s_w \geq -T_a^b$, where T_a^b is the original breakpoint time associated with s_u and s_w is the start time of the corresponding atomic movement a.*

2. *constraint $s_u - s_o \geq 0$, derived from an original non-negativity constraint.*

Finally, observe that through suitable scaling, we may assume that the cost vector k and the length vector l are integral vectors. As a consequence, (BF) is either empty or admits an optimal integer solution ([7]).

2.2 Disjunctive Graph

With program (BF) we associate in a standard fashion the directed graph $G = (V, E \cup F)$, with nodes corresponding to the variables and directed edges associated with simple precedence constraints, including those appearing in a disjunctive constraint. This type of graph was first introduced by Balas in [3], in the context of production scheduling. The vector $l \in I\!R^{|E \cup F|}$ can be regarded as the lengths of the edges of G. The nodes $V_c \subset V$ are called *delay nodes*. The edges in F are called *disjunctive edges* and the two disjunctive edges associated with an original disjunction are called *disjunctive pair* (of edges)[4].

A disjunctive graph associated with two vehicles A and B is depicted in Figure 4. Node 0 is the origin. Nodes 1,2 correspond to the atomic movements of vehicle A, whereas 3,4 are those of B. There is only one disjunction with corresponding pair of disjunctive edges, (2,4) and (4,2) (dotted in the picture). Nodes 5 and 6 are the delay nodes associated with atomic movement 2, whereas 7,8 are the delay nodes associated with atomic movement 4. Edge weights l_{12} and l_{34} are running times whereas $-l_{25}, -l_{26}$ $(-l_{47}, -l_{48})$ are the breakpoints on the time axis of the piecewise linear convex cost function associated with node 2 of train A (node 4 of train B).

The following facts are simple consequences of the previous definitions and are given without proof.

Property 2.4 *The following holds for the disjunctive graph G:*

1. *For every delay node $u \in V_c$, there are exactly two incoming edges and no outgoing edges.*

2. *There is a directed path from the origin o to all $v \in V$, even when restricted to the set E of non-disjunctive edges.*

[4] In the literature a disjunctive pair of edges is often denoted as a *disjunctive arc*

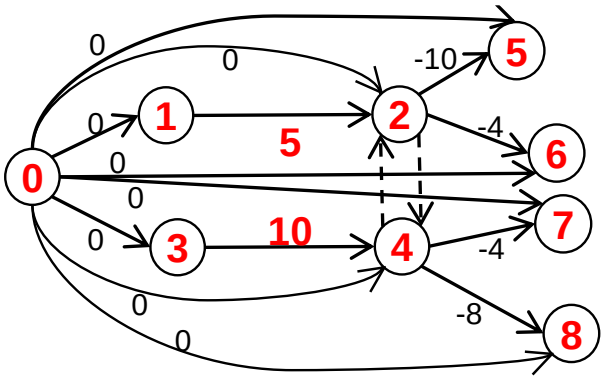

Figure 4: A disjunctive graph associated with two vehicles, A and B.

The second property follows from the fact that, for every $v \in V$, there is an edge from o to v of length 0, deriving from the original non-negativity constraint on the variables. Now, let $\bar{y} \in \{0,1\}^{|F|}$ satisfy constraints (BF.*iii*). Note that this corresponds to selecting exactly one simple precedence constraint per disjunction, as the other term becomes redundant. So program (BF) reduces to a linear program (BF($\bar{y}$)) in the s variables only and with only simple precedence constraints. A natural question is whether (BF($\bar{y}$)) admits a feasible solution $\bar{s}, \bar{y}$. A simple answer to this question exploits the properties of the disjunctive graph introduced above.

Note first that vector $\bar{y}$ can be regarded as the incidence vector of a subset $F(\bar{y})$ of disjunctive edges of G. Such set $F(\bar{y})$ is called *complete selection*, because it contains (exactly) one edge for each disjunction in D. Let $G(\bar{y}) = G(V, E \cup F(\bar{y}))$ be the subgraph of G obtained by dropping all the disjunctive edges in $F \setminus F(\bar{y})$. Observe that the linear program (BF($\bar{y}$)) is the dual of a max-cost network flow problem defined on graph $G(\bar{y})$ (we will come back to this in the next section). The following lemma and some other related facts exploited in the following are basic results from network optimization (see, e.g., [1, 7]).

Lemma 2.5 *Let $\bar{y} \in \{0,1\}^{|F|}$ satisfy (BF.iii). Then (BF($\bar{y}$)) is feasible if and only if the graph $G(\bar{y})$ contains no directed cycles C with strictly positive length $l(C) = \sum_{uv \in C} l_{uv}$.*

If $G(\bar{y})$ does not contain a strictly positive length directed cycle, then $G(\bar{y})$ contains a longest directed path $P^*_u(\bar{y})$ from o to u for every $u \in V$, and we let $L_u(\bar{y}) = l(P^*_u(\bar{y}))$ be its length. Note that $L_o(\bar{y}) = 0$. It is well known (see [7]) that $s_u = L_u(\bar{y})$, for $u \in V$, is a feasible solution to (BF($\bar{y}$)). Also, if $\tilde{s}$ is a feasible solution to (BF($\bar{y}$)), then $\bar{s}_u = \tilde{s}_u - \tilde{s}_o$, for $u \in V$ is also feasible for (BF($\bar{y}$)) and $\sum_{u \in V} k_u \tilde{s}_u = \sum_{u \in V} k_u \bar{s}_u$; also, $\bar{s}_u \geq L_u(\bar{y})$, for $u \in V$.[5]

[5]Indeed, if P_{uv} is a path from u to v in $G(\bar{y})$ and $\bar{s}$ is a feasible solution to (BF($\bar{y}$)), we have $\bar{s}_v \geq \bar{s}_u + l(P_{uv})$. Since $\bar{s}_o = 0$ by construction, then $\bar{s}_v \geq l(P^*_u(\bar{y})) = L_u(\bar{y})$

A *feasible complete selection* $\bar{F}$ is a complete selection of F such that $G(V, E \cup \bar{F})$ contains no strictly positive directed cycle. We denote by $Y_f \subseteq \{0,1\}^{|F|}$ the set of incidence vectors of the feasible complete selections of F.

Lemma 2.6 *Let $\bar{y} \in Y_f$. For $u \in V$, let $L_u(\bar{y})$ be the length of a longest path from o to u in $G(\bar{y})$. For $u \in V$, $s^*_u = L_u(\bar{y})$ is an optimal solution to (BF($\bar{y}$)).*

Proof. First recall that if (BF($\bar{y}$)) has an optimal solution, then it has an optimal solution $\tilde{s}$ with $\tilde{s}_o = 0$ ([7]). The cost of such solution will be $c(\tilde{s}) = \sum_{u \in V_c} k_u \tilde{s}_u$ since $k_u = 0$ for all $u \notin V_c \cup \{o\}$ and $\tilde{s}_o = 0$. Now, it is well known that $s^* = L(\bar{y})$ is feasible for (BF($\bar{y}$)) and we have $s^*_o = 0$. Moreover, let $\bar{s}$ be any feasible for (BF($\bar{y}$)) with $\bar{s}_o = 0$: then $\bar{s} \geq s^*$. The Lemma follows from the fact that for $u \in V_c$. $k_u \geq 0$ and thus $c(\bar{s}) = \sum_{u \in V_c} k_u \bar{s}_u \geq \sum_{u \in V_c} k_u s^*_u = c(s^*)$.

□

The cost $c(s^*)$ of such optimal solution is thus $\sum_{u \in V} k_u L_u(\bar{y})$. Also, since $L_u(\bar{y}) \geq 0$ for all $u \in V$ (due to the 0-length edge from o to u), then any feasible solution has non-negative cost. Now, from basic network optimization theory we know that $L_u(\bar{y})$ corresponds to the length of the unique path from o to u in a certain directed spanning tree of $G(\bar{y})$ rooted at o called *longest-path tree*. Summarizing:

Property 2.7 *For $y \in Y_f$, the optimal schedule for BF($\bar{y}$) is in correspondence to a longest-path tree (rooted in o) of $G(\bar{y})$.*

Such spanning trees are *branchings* (see [29]) since the in-degree of each node is 1, except for the root o (with in-degree 0).

Note that the proof of the above Lemma relies on the crucial property that every feasible solution s to (BF($\bar{y}$)) such that $s_o = 0$ is such that each component s_u is not smaller than $L_u(\bar{y})$. This allows for an immediate generalization to any non-decreasing non-negative cost function:

Corollary 2.8 *Let $c : I\!R^{|V|} \to I\!R$ be a non-decreasing non-negative function, let $\bar{y} \in Y_f$ and consider a problem $Q(\bar{y})$ obtained from (BF($\bar{y}$)) by replacing the original objective function with $c(s)$ and including the constraint $s_o = 0$. Let $T(\bar{y})$ be a longest path tree in $G(\bar{y})$. Then $s^*_u = L_u(\bar{y})$ is an optimal solution for $Q(\bar{y})$, where $L_u(\bar{y})$ is the length of the unique path from o to u in $T(\bar{y})$.*

So, let $T = T(\bar{y})$ be a longest-path tree (from now on all our longest path trees will be directed spanning trees rooted at o) of $G(\bar{y})$. We define the cost $c(T)$ of T as the cost of the solution associated with BF($\bar{y}$), namely $c(T) = \sum_{u \in V_c} k_u L_u(\bar{y})$. Since $L_u(\bar{y})$ is the length $L_u(T)$ of the unique path $P_u(T)$ from o to u in T, we can rewrite

$$c(T) = \sum_{u \in V_c} k_u L_u(T) = \sum_{u \in V_c} c(P_u(T)) \tag{10}$$

where we define $c(P_u(T)) = k_u L_u(T)$ as the cost of path $P_u(T)$.

Finally, observe that, because a delay node $u \in V_c$ has in-degree 2 and out-degree 0 (see Property 2.4), then the following holds.

Property 2.9 *Let T be a longest path tree of $G(\bar{y})$ rooted at o and let $u \in V_c$ be a delay node. Then u is a leaf of T.*

In Figure 5 we give an example. The solid edges are the edges of a longest-path tree T of $G(\bar{y})$ for $\bar{y}_{24} = 1, \bar{y}_{42} = 0$. Delay nodes are (the only) leaves of T. The associated solution in the delay nodes is $s_6 = 1, s_7 = 0, s_8 = 7, s_9 = 3$. Assuming $k_6 = 3, k_7 = 4, k_8 = 5, k_9 = 3$, we have $c(T) = 47$.

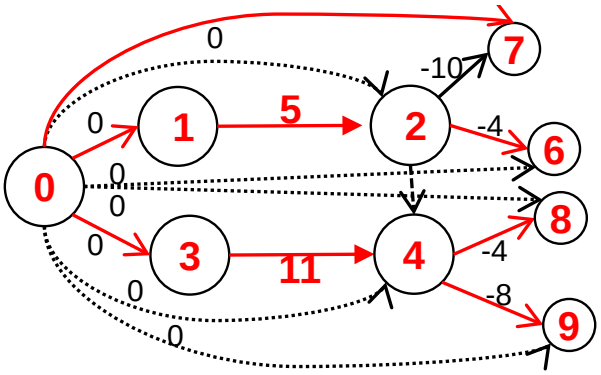

Figure 5: Longest-path tree T of $G(\bar{y})$.

3 Benders' decomposition and reformulation

We now rewrite program (BF) by applying the classical Benders' reformulation ([23]). For any vector $y \in \{0,1\}^{|F|}$ satisfying (BF.*iii*), program (BF) reduces to the dual of the following max-cost flow problem:

$$
\begin{array}{lll}
\max & \sum_{(u,v)\in E} l_{uv} x_{uv} + (\sum_{(u,v)\in F} M(y_{uv} - 1) x_{uv} & \\
s.t. & & \\
(i) & \sum_{uv\in\delta^-(v)} x_{uv} - \sum_{vu\in\delta^+(v)} x_{vu} = k_v, & v \in V \\
(ii) & x_{uv} \geq 0, & uv \in E \cup F
\end{array}
\qquad (MF(y))
$$

where $\delta^-(v)$ and $\delta^+(v)$ are the incoming and outgoing stars of $v \in V$ in graph $G = (V, E \cup F)$. We let $P(G)$ be the polyhedron of the feasible points of $(MF(y))$.

When y is the incidence vector of a complete selection and $G(y)$ does not contain strictly positive length cycles (i.e. $y \in Y_f$), the following holds:

Lemma 3.1 *Let $y \in Y_f$, and let H be a longest path tree of $G(y)$ of cost $c(H)$. Then*

1. *$MF(y)$ has an optimal feasible solution $x(y) = p^*$, where p^* is an extreme point of $P(G)$ and $p^*_e = 0$ for all $e \notin H$.*
2. *The value of the optimal solution is $v(p^*) = \sum_{(u,v)\in H\cap E} l_{uv} p^*_{uv}$*
3. *We have $v(p^*) = c(H)$*

Proof. 1.) and 3.) derive directly from classical duality results (see, e.g., [7]). As for 2.), let y be the incidence vector of $F(y) \subseteq F$ and so by construction $H \subseteq E \cup F(y)$. By 1.) we have that, for $e \in F$, $p^*_e > 0$ implies $e \in F(y)$, and thus $y_e = 1$. So, for $e \in F$ we have $p^*_e(1 - y_e) = 0$ and $\sum_{(u,v)\in F} M(y_{uv} - 1)p^*_{uv} = 0$. □

Let K and J be the set of extreme points p^k ($k \in K$) and extreme rays r^j ($j \in J$) of $P(G)$, respectively. Observe that since they are feasible flows, all extreme points and rays are non-negative vectors. Also, since we assume the r.h.s. k_v to be integer for all $v \in V$, then $P(G)$ is an integral polyhedron and all its vertices are integral points ([7]). Benders' reformulation of (BF) writes as

$$
\begin{array}{lll}
\min & \eta & \\
s.t. & & \\
(i) & y_{uv} + y_{wz} = 1, & \{(u,v),(w,z)\} \in D, \\
(ii) & \sum_{e\in E} l_e r^j_e + \sum_{e\in F} M(y_e - 1) r^j_e \le 0, & j \in J, \\
(iii) & \sum_{e\in E} l_e p^i_e + \sum_{e\in F} M(y_e - 1) p^i_e \le \eta, & i \in K, \\
 & y \in \{0,1\}^{|F|}, \eta \in \mathbb{R}_+ &
\end{array}
\tag{11}
$$

Constraints (11.*ii*) are called *feasibility cuts*, whereas constraints (11.*iii*) are the *optimality cuts*. We remark that, since any feasible solution to (BF) has non-negative cost, we have that also η is non-negative in every feasible solution to (11), and we explicitly stipulate this.

Note that the reformulation only contains the original binary variables y (one for each disjunctive edge), plus one continuous variable η representing the cost of the solution. However, the notorious big-M coefficients are still present in the constraints. In the remaining of this section we exploit classical results from polyhedral combinatorics and network optimization to finally get rid of such coefficients. In particular, the following definitions and results may be found in [4]. Let $S = \{z \in \{0,1\}^{|K|} : \sum_{i\in K} a_i z_i \le b\}$ be the feasible solution set to the knapsack constraint $\sum_{i\in K} a_i z_i \le b$, where $a_i \in Z_+$. A *cover* (of the knapsack constraint) is a subset $Q \subseteq K$ such that $\sum_{i\in Q} a_i > b$. The *cover inequality* associated with a cover Q is $\sum_{i\in Q} z_i \le |Q| - 1$ and is satisfied by every $z \in S$. A cover Q is *minimal* if no proper subset of Q is a cover. The cover inequality associated with a minimal cover is facet defining for $conv(S)$. A

stronger formulation for S can be obtained by replacing the original knapsack constraint with the cover constraints associated with the set $\mathcal{Q}$ of its minimal covers, namely $S = \{z \in \{0,1\}^{|K|} : \sum_{i \in Q} z_i \leq |Q| - 1, \quad Q \in \mathcal{Q}\}$.

Lemma 3.2 *The set of constraints (11.ii) can be replaced by the following set of constraints:*

$$\sum_{e \in F \cap C} y_e \leq |F \cap C| - 1 \qquad C \in \mathcal{C} \tag{12}$$

where $\mathcal{C}$ is the set of strictly positive length directed cycles of G.

Proof. It is well known (see e.g. [7]) that each extreme ray $r^C \in P(G)$ is the incidence vector of a directed cycle C in G. Consider the constraint (11.*ii*) associated with C. Now, if $l(C) = \sum_{e \in C} l_e = \sum_{e \in E \cup F} l_e r_e^C = \sum_{e \in E} l_e r_e^C \leq 0$, then (11.*ii*) is trivially satisfied, because $\sum_{e \in F}(y_e - 1) \leq 0$ for any $y \in \{0,1\}^{|F|}$. So, we can restrict (11.*ii*) to the set $\mathcal{C}$ of strictly positive length directed cycles of G. Now, let $C \in \mathcal{C}$ and let $\sum_{e \in E \cap C} l_e + \sum_{e \in F \cap C} M(y_e - 1) \leq 0$ be the knapsack constraint (11.*ii*) associated with (the incidence vector of) C. Then it is not difficult to see that the knapsack has a unique minimal cover $F \cap C$. □

Incidentally, constraint (12) can also be interpreted as a "combinatorial Benders' cut" (see [11]). It has a direct interpretation: for each strictly positive directed cycle C of G and each selection vector y, at least one of the disjunctive edges contained in C cannot be selected by y.

Next, we focus on constraints (11.*iii*) associated with the extreme points of $P(G)$. Each extreme point of $P(G)$ is in one-to-one correspondence with the set $\mathcal{T}$ of spanning trees of G corresponding to the feasible basis of $(MF(y))$, for $y \in Y_f$ (see [7]). In particular, by Lemma 3.1 if $p^H \in P(G)$ is the extreme point corresponding to spanning tree $H \in \mathcal{T}$, we have that $p_e^H = 0$ for all $e \notin H$. In order to highlight this mapping, we rewrite the set (11.*iii*) as:

$$\sum_{e \in E \cap H} l_e p_e^H + \sum_{e \in F \cap H} M(y_e - 1) p_e^H \leq \eta, \qquad H \in \mathcal{T} \tag{13}$$

We will now show that the set $\mathcal{T}$ can be restricted to the family of longest path trees of $G(y)$, for $y \in Y_f$. Observe that if $y \in \{0,1\}^{|F|}$ satisfies (11.*i*) and (11.*ii*), then y is the incidence vector of a complete selection and $G(y)$ does not contain strictly positive length cycles, so $y \in Y_f$. In other words, for $y \notin Y_f$, constraints (11.*iii*) become redundant. When $y \in Y_f$ we have the following:

Lemma 3.3 *The set of constraints (11.iii) is dominated by the set of valid inequalities*

$$\sum_{e \in E \cap H} l_e p_e^H + \sum_{e \in F \cap H} M(y_e - 1) p_e^H \leq \eta, \qquad H \in \mathcal{T}^* \tag{14}$$

where $\mathcal{T}^* = \{H^*(y) : y \in Y_f, H^*(y)$ any longest path tree in $G(y), \}$.

Proof. We assume $Y_f \neq \emptyset$ (otherwise problem (11) is trivial or infeasible). Let $\bar{y} \in Y_f$ and let $\mathcal{T}(\bar{y})$ be the set of spanning trees associated with the feasible basis of $(MF(y))$. For $H \in \mathcal{T}(\bar{y})$, the l.h.s. of constraint (13) is dominated by

$$\max_{H \in \mathcal{T}(\bar{y})} \left(\sum_{e \in E \cap H} l_e p_e^H + \sum_{e \in F \cap H} M(\bar{y}_e - 1) p_e^H \right) \leq \eta,$$

By Lemma 3.1 the maximum above is obtained for H^* longest path tree of $G(\bar{y})$. □

By statement 3.) of Lemma 3.1, for $H \in \mathcal{T}^*$, each constraint (14) can also be written as

$$c(H) + \sum_{e \in F \cap H} M(y_e - 1) p_e^H \leq \eta, \tag{15}$$

It is not difficult to see that, for $y \in \{0,1\}^{|F|}$, the above constraint is satisfied if and only if y satisfies the following inequality:

$$c(H)(\sum_{e \in F \cap H} y_e - |F \cap H| + 1) \leq \eta, \tag{16}$$

In fact, one can show a stronger result still. Let $H \in \mathcal{T}^*$. For $e \in F \cap H$, let $M_e = p_e^H \cdot M$. Note that $p_e^H > 0$ for some $e \in F \cap H$, implies $M_e \geq M$, because p^H is integral. Let $N = \{e \in F \cap H : M_e > 0\}$.

Lemma 3.4 *Let $Y^H = \{(y, \eta) \in \{0,1\}^{|F \cap H|} \times I\!R_+^1 : (y, \eta)$ satisfies (15)$\}$. Then the constraint*

$$c(H)(\sum_{e \in N} y_e - |N| + 1) \leq \eta, \tag{17}$$

is facet defining for conv(Y^H).

The proof of Lemma 3.4 can be found in the appendix.

So, when $N = F \cap H$, we have that (16) is facet defining for *conv*(Y^H); otherwise it is the trivial lifting of a facet defining inequality. Incidentally, note that one can easily prove that, for $e \in F \cap H$, we have $p_e^H > 0$ (i.e. $e \in N$) if and only if e is on the unique path in H from o to some delay node $v \in V_c$, that is to a node v such that $k_v > 0$. The variables corresponding to these disjunctive edges are the only ones appearing in the constraint (17). We somehow exploit this observation in our main result, stated in Theorem 3.5. To this end, recall that for $H \in \mathcal{T}^*$, its cost $c(H)$ can be computed as $\sum_{u \in V_c} c(P_u(H))$ where V_c is a set of leaves of H and $c(P_u(H)) = k_u L_u(H)$ is the cost of the unique path $P_u(H)$ from o to u in H. The following theorem holds:

Theorem 3.5 *Let $H \in \mathcal{T}^*$ and let $\mu_u \in I\!R$, for $u \in V_c$. Then constraint (16) is dominated*[6] *by the following system of valid inequalities:*

$$\begin{array}{lll} (i) & \eta = \sum_{u \in V_c} \mu_u, & , \\ (ii) & c(P_u(H))(\sum_{e \in F \cap P_u(H)} y_e - |F \cap P_u(H)| + 1) \leq \mu_u, & u \in V_c. \end{array} \tag{18}$$

Before proving our main theorem, we prove an intermediate lemma. Let L_1 and L_2 be a partition of V_c and let H_1 (H_2) be the subtree of H containing only the edges in the paths $P_u(H)$, from the origin o to every node $u \in L_1$ ($u \in L_2$) (see Figure 6). Let $c(H_1) = \sum_{u \in L_1} c(P_u)$ be the cost of H_1 and $c(H_2) = \sum_{u \in L_2} c(P_u)$ be the cost of H_2. Clearly $c(H) = c(H_1) + c(H_2)$. Note that H_1 and H_2 may intersect. Finally, let $F_0 = H \cap F$, $F_1 = H_1 \cap F$ and $F_2 = H_2 \cap F$ be the set of disjunctive edges in H, H_1 and H_2, respectively. Note that

$$F_0 = F_1 \cup F_2 = (F_1 \setminus F_2) \cup (F_2 \setminus F_1) \cup (F_1 \cap F_2) \tag{19}$$

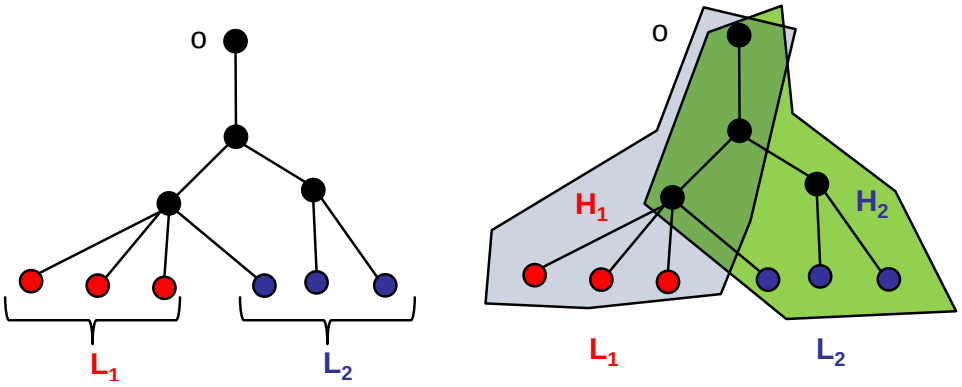

Figure 6: The bipartition of the leaves of H and the corresponding trees H_1 and H_2

Lemma 3.6 *Constraint (16) is dominated by the following system of valid inequalities:*

$$\begin{array}{lll} (i) & \eta = \mu_1 + \mu_2, & , \\ (ii) & c(H_1)(\sum_{e \in F_1} y_e - |F_1| + 1) \leq \mu_1, & i \in L_1, \\ (iii) & c(H_2)(\sum_{e \in F_2} y_e - |F_2| + 1) \leq \mu_2, & i \in L_2, \end{array} \tag{20}$$

Proof. The validity of (20) is trivial. Next, note that system (20) dominates the surrogate constraint

[6] By "dominated" we mean here that if the real vector $(\bar{y}, \bar{\eta}, \bar{\mu})$ is feasible for (18) then its projection $(\bar{y}, \bar{\eta})$ is feasible for (16)

$$c(H_1)(\sum_{e\in F_1} y_e - |F_1| + 1) + c(H_2)(\sum_{e\in F_2} y_e - |F_2| + 1) \leq \mu_1 + \mu_2 = \eta.$$

We show now that the above constraint dominates (16) by proving that, for any $y \in [0,1]^F$, we have:

$$c(H_1)(\sum_{e\in F_1} y_e - |F_1| + 1) + c(H_2)(\sum_{e\in F_2} y_e - |F_2| + 1) \geq (c(H_1) + c(H_2))(\sum_{e\in F_0} y_e - |F_0| + 1)$$

or, after simplification:

$$c(H_1)(\sum_{e\in F_1} y_e - |F_1|) + c(H_2)(\sum_{e\in F_2} y_e - |F_2|) \geq (c(H_1) + c(H_2))(\sum_{e\in F_0} y_e - |F_0|) \quad (21)$$

Observe that F_1 can be partitioned in $F_1 \setminus F_2$ and $F_1 \cap F_2$; similarly F_2 can be partitioned in $F_2 \setminus F_1$ and $F_1 \cap F_2$: then the left-hand side of (21) can be written as:

$$c(H_1)(\sum_{e\in F_1\setminus F_2} y_e - |F_1\setminus F_2|) + c(H_2)(\sum_{e\in F_2\setminus F_1} y_e - |F_2\setminus F_1|) + (c(H_1)+c(H_2))(\sum_{e\in F_1\cap F_2} y_e - |F_2\cap F_1|) \quad (22)$$

While, by using (19), the right-hand side of (21) writes as:

$$(c(H_1) + c(H_2)) \left(\sum_{e\in F_1\setminus F_2} y_e - |F_1 \setminus F_2| + \sum_{e\in F_2\setminus F_1} y_e - |F_2 \setminus F_1| + \sum_{e\in F_1\cap F_2} y_e - |F_2 \cap F_1| \right) \quad (23)$$

Substituting (22) and (23) in (21) and simplifying, we obtain:

$$0 \geq c(H_2)(\sum_{e\in F_1\setminus F_2} y_e - |F_1 \setminus F_2|) + c(H_1)(\sum_{e\in F_2\setminus F_1} y_e - |F_2 \setminus F_1|)$$

which is trivially satisfied for any $y \in [0,1]^F$ since $c(H_1), c(H_2) \geq 0$ □

In order to prove Theorem 3.5, we can use induction: if both L_1 and L_2 are singletons, we are done; otherwise we can (recursively) apply Lemma 3.6 to H_1 and H_2 separately. □

We can finally rewrite problem (11) as follows

$$
\begin{array}{lll}
\min & \sum_{u\in V_c}\mu_u & \\
s.t. & & \\
(i) & y_{uv}+y_{wz}=1, & \{(u,v),(w,z)\}\in D,\\
(ii) & \sum_{e\in C\cap F} y_e \le |C\cap F|-1, & C\in\mathcal{C},\\
(iii) & c(P_u(H))(\sum_{e\in F\cap P_u(H)} y_e - |F\cap P_u(H)|+1)\le \mu_u, & u\in V_c, H\in\mathcal{T}^*\\
 & y\in\{0,1\}^{|F|}, \eta\in\mathbb{R}_+ & \\
\end{array}
\tag{Bend}
$$

Extension to separable, non-decreasing non-negative cost functions. Now, consider non decreasing non-negative functions $c_u : \mathbb{R} \to \mathbb{R}$, for $u \in V_c$ and, for $s \in I\!R^{|V|}$, let $c(s) = \sum_{u\in V_c} c_u(s_u)$. In other words, we assume c to be separable, and the non-zero components of c correspond to the nodes in V_c. As in corollary 2.8, consider problem $Q(\bar{y})$ obtained from (BF($\bar{y}$)) by replacing the original objective function with $c(s)$. Then it is not difficult to see that problem $Q(\bar{y})$ can be reformulated as (Bend), where $c(P_u(H))$ is defined as $c_u(L_u(H))$ and $L_u(H)$ is the length of $P_u(H)$.

A final observation, even if c is a non-separable (non-decreasing non-negative) function we can still reformulate the original problem by using the weaker inequalities (16) instead of (Bend.*iii*) (and considering the original objective $\min \eta$).

4 Solution Algorithm

To solve MILP (Bend) we use classical delayed row and column generation [2, 13]. Recall that, given an optimal solution y^* to (Bend), the associated optimal schedule $s^* = s(y^*)$ can be computed by a longest path tree computation in graph $G(y^*) = (V, E \cup F(y^*))$, where $F(y^*)$ is the set of disjunctive edges "selected" by y^*.

Our algorithm solves a sequence of subproblems (Bend^1), (Bend^2), ..., until an optimal solution is found or the problem is (proven) infeasible. Problem (Bend^k) is associated with the triple $(D^k, \mathcal{C}^k, \mathcal{VT}^k)$ representing the current subset of (indices of) constraints (Bend.*i*), (Bend.*ii*) and (Bend.*iii*), respectively (with $\mathcal{VT}^k \subseteq V_c \times \mathcal{T}^*$). Also, we let y^k be the optimal solution to (Bend^k), with $y_{uv} = y_{wz} = 0$ for all $\{(u,v),(w,z)\} \in D \setminus D^k$. So, if $D^k \neq D$, the selection $F(y^k)$ is not a complete selection, since some of the original disjunctive pairs are not taken into account as the corresponding pairs of variables are fixed to 0. We denote by F^k the set of disjunctive edges associated with the disjunctions of D^k and by $G(y^k) = G(V, E \cup F^k)$. Also, we let $s^k = s(y^k)$ be the associated optimal schedule, calculated by solving (BF(y^k)), namely problem (BF)(y)) restricted to the constraints in $E \cup F^k$ and to the disjunctions of D^k.

The algorithm proceeds by growing sets $D^k, \mathcal{C}^k, \mathcal{VT}^k$. Namely we have $\emptyset = D^0 \subseteq D^1 \subseteq \ldots$, $\emptyset = \mathcal{C}^0 \subseteq \mathcal{C}^1 \subseteq \ldots$ and $\emptyset = \mathcal{VT}^0 \subseteq \mathcal{VT}^1 \subseteq \ldots$.

At each iteration we first check if y^k violates a cycle inequality C. If so, we add C to the set of cycle inequalities and iterate. If not, we compute a longest path tree T^k in $G(y^k)$ and check whether some path inequalities associated with T^k are violated. If so, we add them to the set of path inequalities and iterate. Otherwise we perform a macro-iteration, namely we check if solution s^k associated with T^k violates any of the disjunctions in the set $D \setminus D^k$. If not, we are done: we let $s^* = s^k$, we (trivially) extend y^k to a feasible complete selection y^* respecting s^* and we have that (s^*, y^*) is an optimal solution to our original problem. Otherwise, we add the violated disjunction $\{(u, v), (w, z)\}$ to the current set D^k and iterate.

Paths&Cycles Algorithm (PC)

0. Set $k = 0$; Set $D^1 = \emptyset$, $\mathcal{C}^1 = \emptyset$, $\mathcal{VT}^1 = \emptyset$.

   Micro Iterations

1. $k = k + 1$. **Solve** (Bend^k). If (Bend^k) infeasible, (Bend) is infeasible and **STOP**.

2. Let y^k be the optimal solution to (Bend^k).

3. If y^k violates cycle inequality C:
   Let $\mathcal{C}^{k+1} = \mathcal{C}^k \cup \{C\}$, $D^{k+1} = D^k$, $\mathcal{VT}^{k+1} = \mathcal{VT}^k$. **GoTo 1**.

4. Solve (BF(y^k)). Let $T^k \in \mathcal{T}^*$ be the longest path tree of $G(y^k)$ and s^k be the associated schedule.

5 If y^k violates path inequalities $VT^k = \{(v, T^k) : v \in V_c^k \subseteq V_c\}$:
   Let $\mathcal{VT}^{k+1} = \mathcal{VT}^k \cup VT^k$, $D^{k+1} = D^k$, $\mathcal{C}^{k+1} = \mathcal{C}^k$. **GoTo 1**.

   Macro Iterations

6. If s^k violates disjunction d, let $D^{k+1} = D^k \cup \{d\}$. $\mathcal{C}^{k+1} = \mathcal{C}^k$, $\mathcal{VT}^{k+1} = \mathcal{VT}^k$. **GoTo 1**.

7. **STOP**: extend y^k to a complete selection y^*, and $(y^*, s^* = s^k)$ is an optimal solution for (BF).

The solution of the 0,1 program (Bend^k) at Step 1 can be obtained by invoking any MILP solver (see Section 5). Problem (BF(y^k)) amounts to a longest path tree computation in $G(y^k)$, which in turn can be obtained in $O(|E \cup F(y^k)|)$ for acyclic graphs (which is the case for the real-life instances presented in Section 5, and more generally for many relevant traffic management problems). Note that, in each macro iteration k, only a subset of the y variables can be different from 0, namely variables y_e for $e \in F^k$. One can then neglect all other variables y_e for $e \in F \setminus F^k$. So, at iteration k only variables y_e for $e \in F^k$ are actually included in the current MILP (Bend^k) (*delayed column generation*).

5 Computational Results

In this section we present computational results for the PC algorithm. We test the algorithm on real-life instances from *train rescheduling*, an operational problem that arises in managing real-time railway traffic. This is in fact the application which originally motivated this new study: the necessity to improve on a successful but big-M based approach (referred to as BM) presented in [18] (which extends and completes the formulation introduced in [17]). In BM, the big-M formulation (3) is solved repeatedly following the macroscopic/microscopic decomposition approach described in [30]. In this paper, we compare BM with the new PC algorithm. BM is also based on delayed row generation as the PC algorithm (see Section 4); however, at Step 1, formulation (Bend) is replaced by the initial big-M formulation (3) introduced in Section 2. Consequently, Steps 2, 3, 4, and 5 of the PC Algorithm are skipped, while 6 and 7 are executed. For more detail regarding BM we refer to ([18]).

The instances are provided by a train TMS (traffic management system) operative on a railway line in Norway, in the Stavanger region (see Section 5.1). In the rest of the section, we first present information about the test instances and the implementation details, then present comparisons between BM and PC for different types of non-decreasing non-negative objective functions that typically arise in this kind of traffic management problems.

5.1 Test instances

First, we give some context regarding the instances used in this paper for our tests. All instances refer to real-life trains running on a railway line at a certain moment in time. More precisely, a snapshot of the state of the system (train position, network status) is taken each time an event occurs on the line (train reaches signaling point, delays are registered etc.). The instances are provided by an optimization-based train rescheduling system implemented in Norway in February 2014. The system was developed at SINTEF and funded by Norway's infrastructure manager (Jernbaneverket), train operating companies (NSB, Flytoget, CargoNet) and Research Council (Forskningsrådet). The system was implemented on the main line of the Stavanger region (the *Jærbane*) on the west coast of Norway (also including the railway between Egersund and Moi). The Jærbane (123 km, 16 stations) is one of the few lines in Norway with both single- and double-tracks and is the most trafficked line in Norway outside of the Oslo area.

As mentioned in the introduction, the problem can be decomposed into "station problems" and a "line problem", and in this paper we only focus on the latter (in [18] we show how the two problems can be re-coupled exactly by means of Benders' logic decomposition). The line problem can be viewed as a problem where each station is collapsed into a single, capacitated node (and thus we call it a *macroscopic* representation of the railway). So, in the line problem the railway is essentially reduced to a sequence of nodes representing the original stations, connected by single or double-track sections. In single track sections, trains can only meet or pass in stations. In double

track sections, each track is devoted to one of the two directions, so meets can happen anywhere, whereas passes can only occur in stations. In other words, the line problem consists in finding a schedule for the trains running during the planning horizon such that meets and passes occur only in feasible regions of the network and some cost associated with the delays is minimized. In our instances, trains traveling along the entire line have approximately 50 (macroscopic) atomic movements.[7] Even though neglected in this paper, it is worth mentioning that each station problem is a feasibility problem that, besides scheduling, also involves routing trains. Indeed, in many stations trains can choose among several routes and platforms. In [18] we show how this problem can be modelled and solved. However, we remark here that, in our instances, the bottleneck of the overall solution process is the solution to the line problem, requiring significantly higher computational effort.

5.2 Implementation details

Tests were run on a Dell laptop with an Intel (R) Core (TM) i7-5600U CPU @2.60 GHz 16GB RAM. IBM Cplex 12.5 32 bit was used for all experiments, with the exception of some of the instances in Tables 3 and 6 for which the 64-bit version was used due to memory issues when solving the MILPs (more info is given in describing the tables).

5.3 Convex non-decreasing (non-negative) piecewise linear cost functions

Convex non-decreasing (non-negative) objective functions are commonly adopted for job-shop scheduling problems in traffic management. Such objectives can typically be expressed or approximated with piece-wise linear functions (see Figure 3). Broadly speaking, if the system is in a state that requires recovery, e.g. train i is delayed, a convex objective function enforces that the delay will be somehow spread out on the trains encountered by i (assuming, for instance, these are not themselves already delayed). Convex functions are chosen to model recovery decisions for a combination of reasons. First of all, they model the behaviour/recovery strategies of planners (or controllers, dispatchers etc.) reasonably well, particularly in situations that are not unusually critical (like when one or more vehicles are extremely delayed or there are large scale disruptions). Secondly, they are deemed preferable on the modeling side due to their computational tractability. Modeling a convex objective function for BM only requires adding an extra continuous variable for each interval (or "piece") of the objective and some linear constraints. In this study we show however that this type of objective function can also present some non-negligible drawbacks, which will be discussed later in this section.

[7] Aside from 16 stations, the line also has 7 stops where passengers can load and unload but no train meets and passes are possible, and 5 tracks which are divided in multiple block sections.

In Table 1 we present results for 5 days of real-life data from the railway line in Stavanger, aggregated over four-hour time periods[8]. A 1 hour time limit was set (and in no instance reached). The objective, agreed with practitioners from the Norwegian infrastructure manager, is of the type shown in Figure 3, with 6 delay intervals (or classes) which are penalized increasingly: less than 1 minute delay, between 1 and 3, between 3 and 5, between 5 and 10, between 10 and 15 and more than 15 minutes delay. In column 'Periods' we indicate the day and hour range (format: *dd hh-hh*). In columns 'Trains' and 'Late' we indicate the average number of trains controlled by the system and the average number of these running late, respectively, within the time period. Column 'Runs' indicates the number of instances considered for the given period in these experiments.[9] The remaining columns give information about the performance of the two algorithms, namely the average run time ('Time'), the average number of algorithm iterations ('Iterations'), the average time taken to solve each iteration ('It. time'), the average aggregated number of branching nodes made by Cplex in solving each instance ('Branchings') and the average number of initial and generated rows and columns. Note that, for both algorithms, the column 'Iterations' refers to the number of times the algorithm performs the most-time consuming task, namely solving the MILP at hand with Cplex. For BM, basically all the overall computation time is taken up by this task. For PC, the proportion of time used by Cplex w.r.t. to the other steps of the algorithm (namely 3, 4 and 5, see Section 4) is less skewed but still very significant. Notably, this proportion grows with the size of the problem and number of iterations (an overview is given in Figure 7). Column 'Branchings' in Table 1 is calculated by summing the number of branching nodes explored by Cplex each time it is called during an algorithm run.

As can be noted in Table 1, PC performs better than BM in terms of computation time in 22 periods out of 25 ('Time'). Column 'It. time' shows that the time required by PC for each iteration is, on average, somewhere between 15% and 30% of the corresponding iteration time for BM. The average number of iterations for PC is (quite naturally) always higher than BM, due to the additional *micro-iterations* that have to be performed by PC. Indeed, as described in Section 4, PC requires a series of steps (namely 3, 4 and 5) to check for violated cycle and path inequalities. Whenever at least one such inequality is violated, the corresponding constraint is separated and added to the MILP, and Cplex is invoked to find a new optimal selection y^k. PC spends a non-negligible amount of time to perform these steps (depending on the instance, between 5 and 9 out of 10 iterations are of this kind). Therefore, using strategies to skip at least a part of these micro-iterations (e.g. by smart heuristic generation of constraints and variables) could further increase the improvement over BM.[10]

Thus, to summarize the performance of the two algorithms: a PC run solves a

[8]Note that the data is spread out over 6 days, and that time periods where on average less than 1 train is late were removed

[9]Note that here the instances were randomly sampled at a rate of between 1 and 2 per minute, while, in the real-life setting, on average an instance every 20 seconds has to be solved.

[10]These questions will be matter of future studies.

Period	Trains	Late	Runs	Time (seconds)		Iterations		It. time (milliseconds)		Branchings		Initial				Generated			
												Rows		Columns		Rows		Columns	
				BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC
1 12-16	84	4	216	8.01	8.87	12	44	691	202	7258	2204	3858	0	3233	0	350	822	97	233
1 16-20	62	5	240	2.25	1.60	6	12	381	133	794	5	2766	0	2402	0	372	226	125	135
1 20-24	66	5	240	2.17	1.32	7	9	314	147	55	0	2911	0	2527	0	691	173	256	128
2 04-08	115	7	222	26.77	23.84	16	117	1705	204	84009	11180	5044	0	4263	0	1590	2621	401	399
2 08-12	109	5	231	12.41	12.55	13	54	933	232	14267	1296	4824	0	4060	0	785	972	217	319
2 12-16	84	3	240	7.74	13.36	12	60	650	223	4007	794	3874	0	3249	0	425	1440	115	261
2 16-20	62	4	240	2.05	1.38	7	10	315	138	206	0	2758	0	2395	0	366	185	124	132
2 20-24	66	5	224	6.22	3.87	9	21	723	184	6844	574	2889	0	2509	0	780	367	276	165
3 04-08	113	3	240	5.20	3.60	10	19	536	189	912	65	4957	0	4164	0	478	379	165	215
3 08-12	105	4	163	2.27	1.74	7	10	329	174	270	14	4696	0	3924	0	84	197	29	155
3 12-16	86	2	240	3.16	2.15	8	12	405	179	156	0	3963	0	3318	0	117	226	51	170
3 16-20	57	4	238	2.49	1.83	6	14	422	131	633	0	2562	0	2208	0	198	202	62	133
3 20-24	38	3	240	1.21	0.34	5	7	257	49	22	0	1836	0	1615	0	228	91	84	70
4 08-12	40	1	240	0.77	0.24	4	5	183	48	4	0	2338	0	1945	0	63	68	24	60
4 12-16	30	1	240	0.72	0.10	4	4	200	25	17	0	1738	0	1444	0	77	49	29	45
4 16-20	23	1	237	0.39	0.04	3	3	134	13	2	0	1180	0	1000	0	74	36	29	32
4 20-24	15	1	240	0.05	0.01	2	1	28	10	0	0	745	0	649	0	2	19	1	17
5 08-12	37	1	240	0.76	0.24	4	6	205	40	28	0	2346	0	1976	0	50	63	15	55
5 12-16	34	1	240	1.76	0.52	6	11	289	47	80	0	2128	0	1798	0	89	94	32	71
5 16-20	35	2	240	1.70	0.86	5	11	327	78	81	0	1875	0	1628	0	181	108	59	76
5 20-24	59	3	240	3.21	1.50	7	14	459	107	236	0	2694	0	2316	0	318	156	107	115
6 04-08	112	3	239	4.58	3.30	9	18	492	183	657	39	4923	0	4125	0	263	330	84	203
6 08-12	111	3	107	13.25	10.89	14	48	946	227	8032	716	4843	0	4053	0	350	804	106	292

Table 1: Algorithmic information for the convex objective function

significantly higher number of MILPs very fast, while BM requires solving fewer MILPs at each run but is slower. Accordingly, the number of branchings is higher for BM in all periods, while for PC in most periods Cplex is able to solve at root node. In addition, PC always terminates with a significantly smaller MILP (see columns 'Initial' and 'Generated' variables and constraints).

Figure 8 shows the frequency of instances in Table 1 solved within certain time ranges by the two algorithms. The first two columns show how many instances were solved within 1 second, the second two columns between 1 second and 3 seconds and so on. Figure 9 shows the cumulative distribution of the instances solved by the two algorithms, i.e. the percentage of instances solved within a given time. For instance, we notice that PC solves 50% of instances within 1 second and 72% of instances within 2 seconds, against 33% and 56% for BM, respectively. PC reaches the 90% mark of solved instances after approximately 7 seconds, while BM requires around 9.4 seconds. The chart is truncated at 90% due to the outliers in the remaining 10% that hinder its readability.

In Table 2 we present aggregated results with respect to computation time for the same 5 days of data from March 2014. For results presented in this table however, we use a much more stringent time limit for the algorithms, one that is in line with what happens in the practical setting for train traffic management (and other real-time applications). The time limit in this case was set to 30 seconds, as agreed with dispatchers in Stavanger.[11] Column 'Day' indicates the day of the month the instances were generated. Column 'Runs' indicates the number of instances taken into account

[11]Depending on the line this time limit may actually range from a few seconds to minutes.

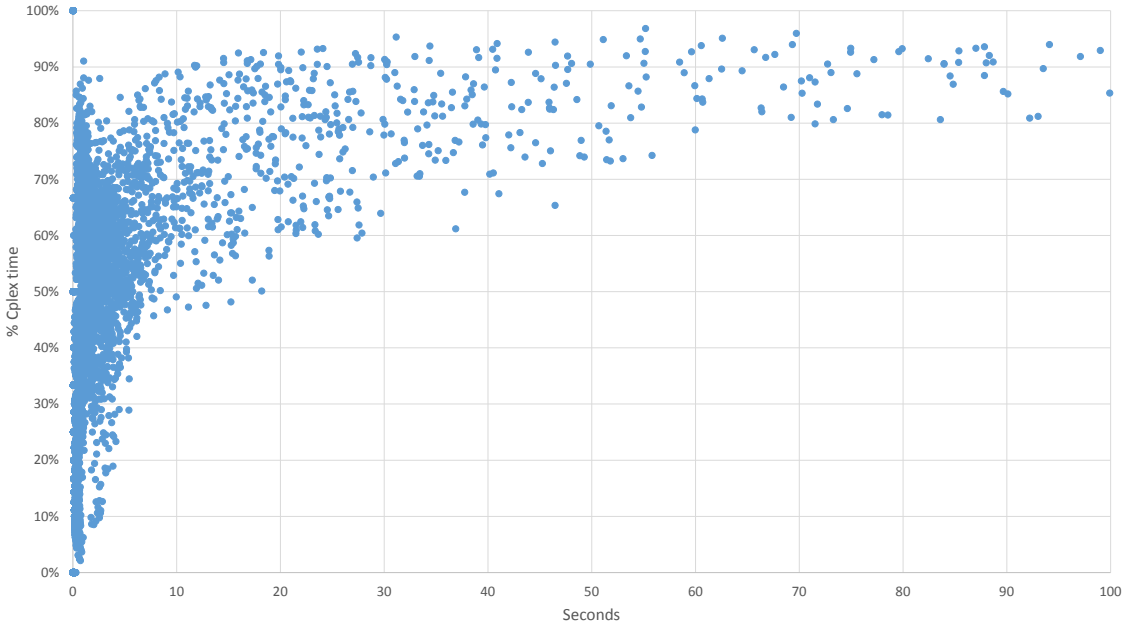

Figure 7: Instances solved by PC represented in terms of overall solution time (horizontal axis) and percentage of solution time used by Cplex respect to the overall solution time (vertical axis).

in this experiment for that day. 'Optimal (%)' expresses the percentage of instances solved in seconds. We note that the instances for which the optimal solution was not found within the time limit are also counted in the average run time as being solved in 30 seconds. This models what happens in the real-life setting, where the algorithm is given T seconds (in this case $T = 30$) to find a solution, before some other solution method must come in play to find a solution (a fast, possibly sub-optimal heuristic method or "manual" intervention by the operators). Table 2 shows that on average PC solves faster than BM in all ranges. While the speed up is small for days 1 and 2 ($1.07x$ and $1.01x$, respectively), it increases to as much as $4.45x$ on day 4.

In Table 3 we compare the algorithms on 10 representative instances. These were selected by dividing all our instances in two classes, one with less trains (< 100) and one more trains ($>= 100$) and picking 5 at random from each class, with a higher probability placed on instances with more trains late. These instances are available in anonymised form as part of this paper's online companion in Operations Research. In this test, the 1 hour time out ($t.o.$) was set. In addition to BM and PC, we attempt to solve the instances using the full big-M formulation[12] and feeding the resulting program to Cplex. We denote this approach as FBM. For the columns we use the same headings as in

[12]The big-M formulation without using the delayed generation approach, i.e. with all variables and constraints that model the possible disjunctive pairs.

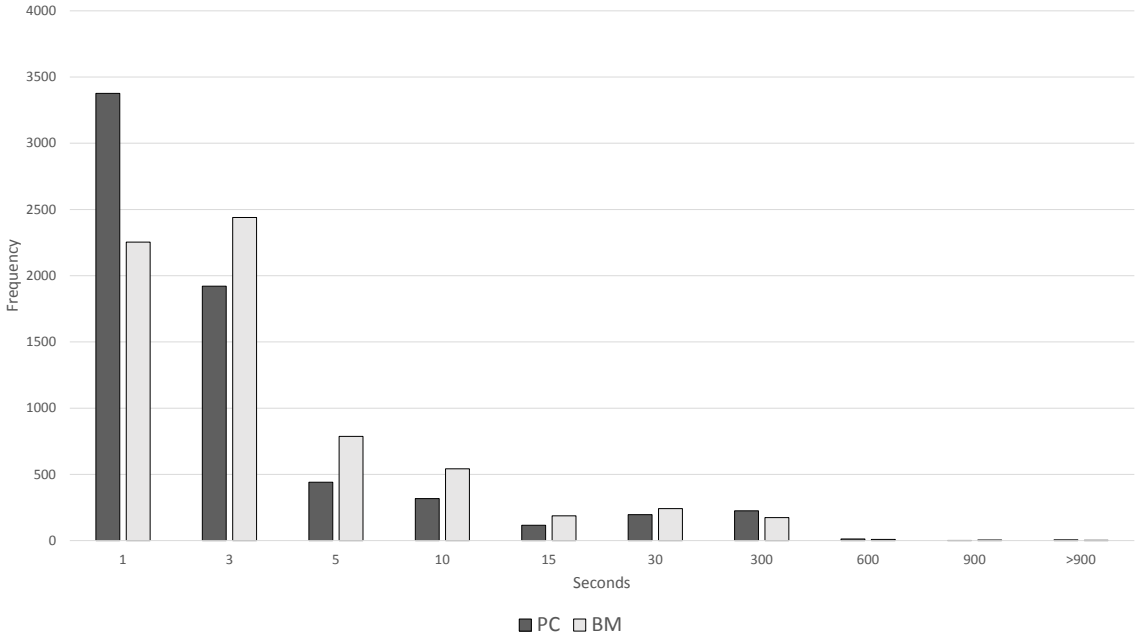

Figure 8: Histogram of the frequency of instances with respect to computational time presented in Table 1.

Table 1, with the exception of 'Last it. time' (which expresses the time taken by Cplex to solve the last MILP) and 'Rows' and 'Columns' (which here represent the total number of rows and column of the final MILP, i.e. initial plus generated). Note that for FBM we omit columns 'Iterations' and 'Last it.time' as there is no delayed generation, thus it solves a single MILP/iteration. In column 'Time' we notice how FBM takes orders of magnitude more computation time respect to the other two approaches, timing out on 3 occasions (more than 1 hour running). Moreover, again for FBM, the 32-bit version of Cplex crashed in solving instances 5 to 10 due to lack of memory, requiring the use of the 64-bit version. This results clearly demonstrate that solving "directly" the full big-M would be impracticable, at least for our test-bed of real-life traffic management instances. As shown for the aggregate results, PC solves a higher number of MILPs w.r.t. BM but spending significantly less time. This can be seen as consequence of the smaller size of the MILPs (see 'Rows' and 'Columns') and more importantly of the stronger formulation which allows to explore a reduced search tree (see 'Branchings'). In addition, the time taken to solve the last iteration/MILP is considerably smaller for PC, again pointing to the fact that, with an effective strategy to generate the "right" constraints and variables (and skipping as many micro-iterations as possible), one could further improve the overall performance of the algorithm.

Convex functions are, in many cases, a reasonable proxy of a planner's decision process. However, in many cases the effect on decisions can become increasingly distant

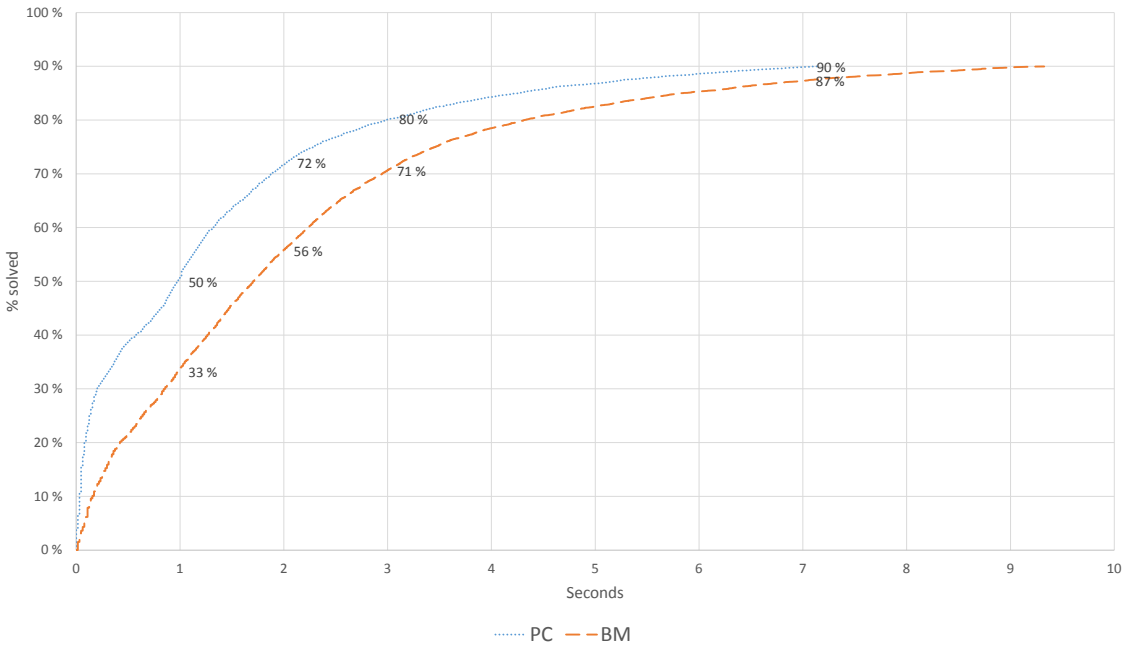

Figure 9: Cumulative distribution of the percentage of instances solved in Table 1 with respect to computation time.

from the wish or behaviour of the planners, operators, stakeholders. We will elaborate on this more in the following subsection.

5.4 Non-convex non-decreasing (non-negative) cost functions

A class of cost functions frequently adopted by practitioners in traffic management problems, is that of non-convex non-decreasing (non-negative) functions. In many cases, these functions model the recovery strategies of planners more accurately than convex ones. In public transport for instance, heavily delayed vehicles often actually *lose* priority over other vehicles after a certain threshold (the opposite of what the corresponding convex objective function would enforce). Controllers, dispatchers, or the operators themselves, may prefer avoiding the recovery of such a vehicle should it come at the price of delaying others. In other words, they may favour the strategy of not further disturbing the system, rather than attempting the recovery of one (or more) heavily delayed vehicles. In mathematical terms, this can be expressed with a function of the type depicted in Figure 10.

The "loss of convexity" is generally seen as a drawback, as the computational complexity of traditional models tends to increase as a consequence (e.g. for BM a function of this kind requires adding suitable binary variables and constraints).

In the following we compare the two algorithms using a non-convex non-decreasing

Day	Runs	Optimal (%)		Run time (s)		Speed up
		BM	PC	BM	PC	
1	696	99.6	97.6	4.01	3.76	1.07x
2	1188	88.6	82.9	10.93	10.85	1.01x
3	1360	100	100	2.92	1.86	1.57x
4	1434	100	100	0.49	0.11	4.45x
5	1378	100	100	1.34	0.55	2.44x

Table 2: Aggregate comparisons for 5 days of computations using the convex objective function.

Id	Trains	Late	Time (seconds)			Iterations		Last it. time (milliseconds)		Branchings			Rows			Columns		
			FBM	BM	PC	BM	PC	BM	PC	FBM	BM	PC	FBM	BM	PC	FBM	BM	PC
1	77	0	88.34	0.41	0.15	2	2	200	23	43	0	0	126361	2912	84	14562	2858	81
2	96	2	276.72	13.80	3.88	14	29	620	30	1627	1723	0	203894	4669	500	22658	3815	242
3	96	1	196.46	6.77	6.16	14	31	1654	21	903	1700	0	201583	4678	525	22487	3812	240
4	92	3	204.01	11.25	2.71	14	34	942	154	1129	3561	0	182640	4555	640	20696	3660	260
5	88	4	238.35	6.77	6.31	9	32	901	334	1585	2699	0	171706	4332	520	19706	3534	230
6	105	3	549.97	42.92	22.61	17	101	3846	540	5398	24517	2184	223957	5962	2174	23910	4358	341
7	105	3	2025.69	137.68	88.71	18	175	30018	736	27691	115475	26258	229178	6477	3493	28035	4421	407
8	119	7	t.o.	96.68	78.37	20	136	18037	513	-	84144	7267	290091	7088	3609	35173	4920	493
9	118	5	t.o.	95.39	117.55	19	180	38582	615	-	86802	15493	286200	6981	3642	34600	4854	518
10	120	7	t.o.	510.96	373.24	20	229	51564	5758	-	606519	36170	288642	6884	7008	34812	4885	522

Table 3: Algorithmic information for 10 representative instances with the convex objective function

(non-negative) objective function. In particular, we use a cost function with a similar structure to that used for the convex case (the same 6 delay intervals), only with a "plateau" for the last interval (more than 15 minutes delay). In other words, any train delayed more than 15 minutes, whose delay cannot be brought under such threshold, loses all dispatching priority. Comparisons are presented in Tables 4, 5 and 6, for the same instances and indicators used previously in Tables 1, 2 and 3, respectively.

Table 4 confirms and adds to the results obtained with the convex objective. Indeed, the trade-off between shorter (average) iteration time and higher number of iterations is even more favourable to PC in this case, as it performs better than BM in terms of average computation time in 23 periods out of 25. This improvement can be explained as the consequence of PC solving even smaller MILPs than in the convex case, which can be noted by the size of the MILPs (columns 'Initial' + 'Generated') and the number of branchings (PC on average solves every sub-problem at node 0 for 19 periods of 25).

Aside from PC performing proportionally better with this objective than BM, an improvement w.r.t to the convex case was registered for both algorithms, which may seem counter-intuitive given the increased computational complexity of using such a non-convex objective. However, this can be explained intuitively analyzing the behaviour of the algorithms in situations of disruption. Indeed, in the convex case, the number of "similar" solutions (in terms of objective function) is often very high. There-

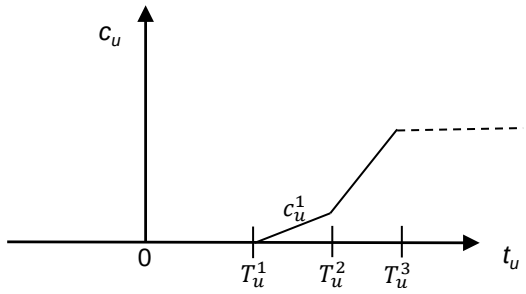

Figure 10: Example of non-convex non-decreasing (non-negative) function.

fore algorithms such as BM and PC that follow a "delayed" approach will tend to generate many rows and columns associated with potential conflicts to explore such solutions.

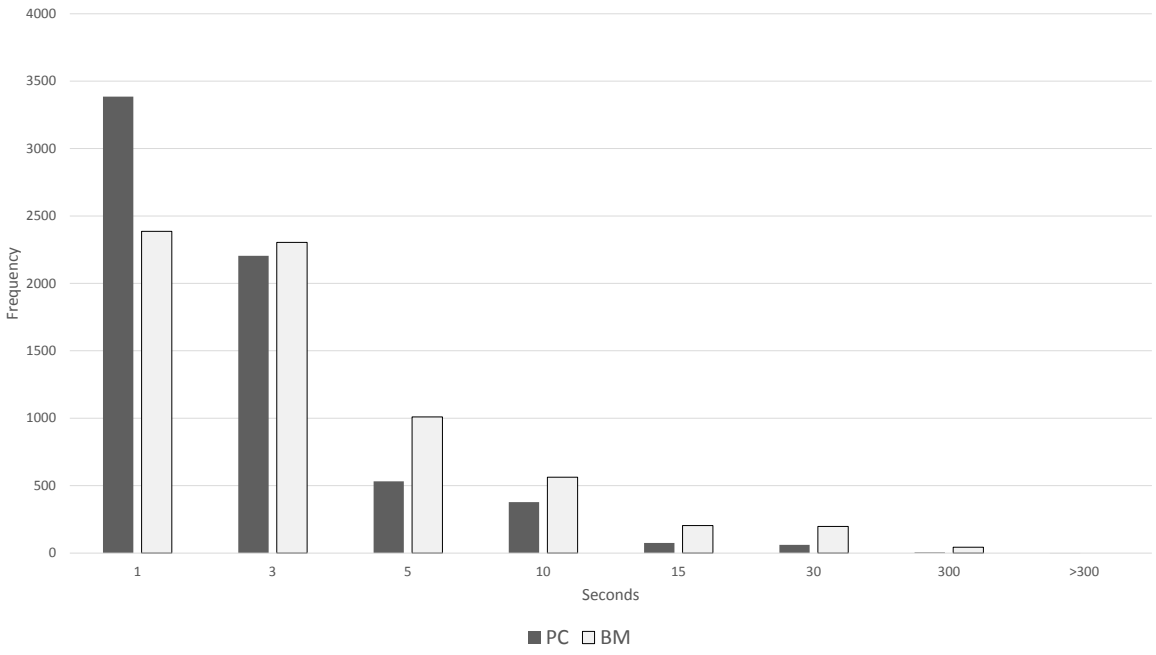

Figure 11: Histogram of the frequency with respect to computational time for instances in Table 4.

In Figure 11 we show the frequency of instances in Table 4 solved within certain time ranges. The superior performance of PC in terms of computation time stands out as PC has less instances in each range except for the less than 1 second case. Similarly, in Figure 12 we show that PC solves a higher percentage of instances than BM by any given computation time. For example, PC solves 50% of the instances within 1 second,

Period	Trains	Late	Runs	Time (seconds)		Iterations		It. time (milliseconds)		Branchings		Initial				Generated			
												Rows		Columns		Rows		Columns	
				BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC
1 12-16	84	4	216	4.75	6.35	11	31	440	207	2904	172	3941	0	3233	0	324	569	93	233
1 16-20	62	5	240	2.09	1.69	6	12	327	147	247	0	2827	0	2402	0	383	222	127	135
1 20-24	66	5	240	3.53	1.30	7	9	527	141	126	0	2977	0	2527	0	698	173	260	128
2 04-08	115	7	222	17.77	4.77	21	18	858	273	2708	0	5160	0	4263	0	2751	420	614	310
2 08-12	109	5	231	11.49	5.16	15	21	787	246	4067	12	4933	0	4060	0	1393	434	320	278
2 12-16	84	3	240	6.76	5.14	11	25	609	210	4514	207	3958	0	3249	0	809	466	156	212
2 16-20	62	4	240	2.20	1.26	7	10	333	125	94	0	2819	0	2395	0	376	184	124	131
2 20-24	66	5	224	4.94	2.90	8	16	633	177	5307	32	2955	0	2509	0	780	276	275	161
3 04-08	113	3	240	4.17	2.22	8	12	502	193	66	0	5070	0	4164	0	881	252	220	198
3 08-12	105	4	163	2.66	1.76	6	10	416	181	259	11	4801	0	3924	0	101	197	39	154
3 12-16	86	2	240	2.97	2.15	8	12	386	173	47	0	4050	0	3318	0	142	227	58	170
3 16-20	57	4	238	2.16	1.83	6	14	338	133	417	0	2615	0	2208	0	216	202	66	133
3 20-24	38	3	240	1.01	0.34	5	7	210	50	5	0	1874	0	1615	0	235	91	88	70
4 08-12	40	1	240	1.04	0.24	6	5	182	53	0	0	2378	0	1945	0	104	68	32	60
4 12-16	30	1	240	0.44	0.10	4	4	119	29	1	0	1768	0	1444	0	81	49	31	45
4 16-20	23	1	237	0.32	0.04	3	3	107	14	0	0	1203	0	1000	0	75	35	29	32
4 20-24	15	1	240	0.05	0.01	2	1	28	9	0	0	760	0	649	0	2	10	1	8
5 08-12	37	1	240	0.77	0.23	5	6	171	40	15	0	2383	0	1976	0	55	63	18	55
5 12-16	34	1	240	0.97	0.52	6	11	156	47	10	0	2162	0	1798	0	92	94	33	71
5 16-20	35	2	240	1.07	0.87	5	11	214	78	8	0	1910	0	1628	0	182	108	59	76
5 20-24	59	3	240	2.63	1.49	6	14	424	110	22	0	2753	0	2316	0	318	155	109	115
6 04-08	112	3	239	3.73	2.32	8	13	448	186	85	0	5035	0	4125	0	679	240	142	192
6 08-12	111	3	107	10.96	12.08	14	50	766	242	9724	1028	4954	0	4053	0	399	845	114	295

Table 4: Algorithmic information for the non-convex objective function.

74% after 2 seconds, 94% after 5 seconds and 98% after 10 seconds, against the 35%, 56%, 85% and 94% of BM, respectively.

Day	Runs	Optimal (%)		Run time (s)		Speed up
		BM	PC	BM	PC	
26	696	99.9%	99.9%	3.41	3.00	1.14x
27	1188	97%	100%	8.47	3.79	2.23x
28	1360	100%	100%	2.78	1.62	1.72x
29	1434	100%	100%	0.52	0.10	5.20x
30	1378	100%	100%	0.94	0.55	1.71x

Table 5: Aggregate comparisons for 5 days of computations using the non-convex function

In Table 5 we show the performance of the algorithms aggregated over each day of instances. Similarly to Table 2, we set a 30 second time limit for the algorithms to find the optimal solution. We notice that the average speed up achieved using PC increases compared to the convex case (Table 2) on the first 4 days.[13] In particular,

[13]On the last day, the average computation time for PC is actually the same as the convex case (0.55 seconds), while BM performs better (0.94 seconds against 1.34). Still, PC performs better than BM (0.55 against 0.94 seconds).

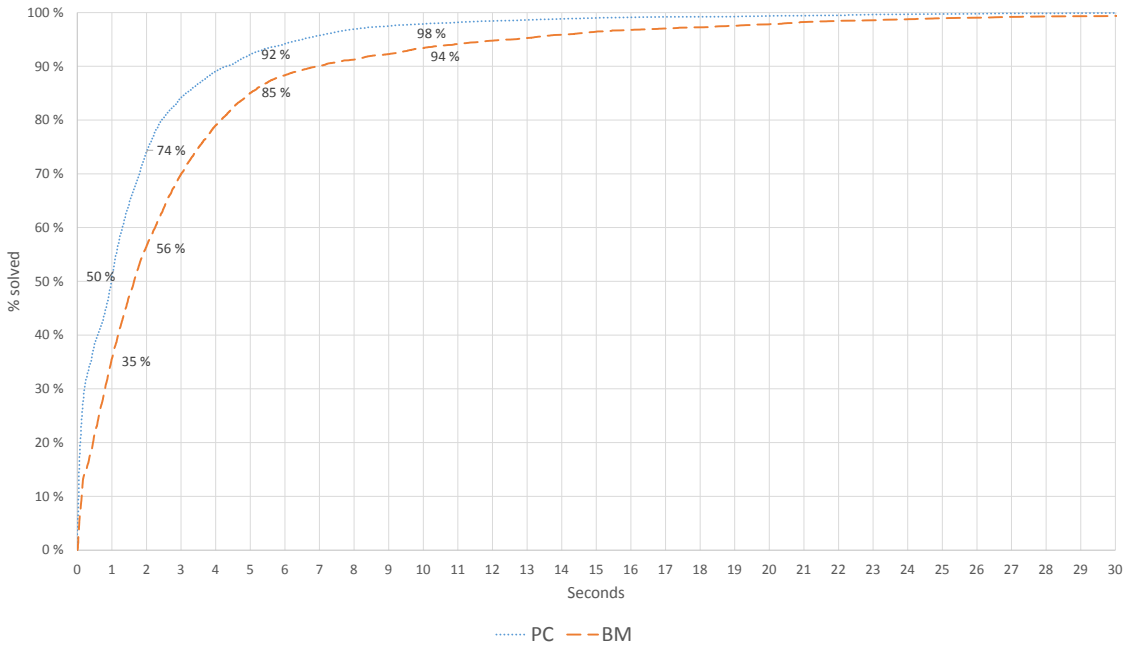

Figure 12: Cumulative distribution of the percentage of instances solved in Table 4 with respect to computation time.

this improvement is most noticeable on day 2, where PC is 2.23 times faster than BM on average, against the convex case where the algorithms basically performed the same ($1.01x$ speed up, see Table 2).

Id	Trains	Late	Time (seconds)			Iterations		Last It. time (milliseconds)		Branchings			Rows			Columns		
			FBM	BM	PC	BM	PC	BM	PC	FBM	BM	PC	FBM	BM	PC	FBM	BM	PC
1	77	0	9.35	1.03	0.07	4	1	162	20	0	0	0	126438	2999	84	14562	2863	81
2	96	2	52.61	8.55	5.07	15	28	387	141	281	1306	0	208387	4869	500	26385	3824	242
3	96	1	44.56	6.94	5.61	15	30	570	225	423	1395	0	206067	4868	519	26207	3819	240
4	92	3	33.53	4.28	7.26	14	32	241	140	595	2163	0	186921	4713	628	24254	3661	260
5	88	4	28.39	7.13	5.24	13	29	508	165	283	1748	0	175847	4498	478	23149	3535	230
6	105	3	67.42	9.47	1.69	9	16	730	10	444	0	0	228616	6820	307	27838	4482	257
7	105	3	85.35	6.75	3.44	12	16	582	285	411	80	0	229283	8387	305	28035	4743	267
8	119	7	3257.91	22.03	2.29	17	13	1663	20	9309	4408	0	290144	9136	610	35173	5243	371
9	118	5	2301.93	29.47	2.07	14	13	1795	10	3475	2736	0	288766	9002	428	34812	5193	322
10	120	7	t.o.	22.12	1.33	14	13	2754	10	-	3722	0		9366	619		5278	368

Table 6: Algorithmic information for 10 representative instances with the non-convex objective function

In Table 6 we compare the algorithms on the same 10 representative instances in Table 6 using the non-convex objective. Again, we additionally attempt to solve the instances using the full big-Mformulation and feed the resulting program to Cplex (approach FBM). What stands out from Table 6 is that PC performs significantly better than BM, in particular for the hard instances 8-10, for which the improvement

is drastic. Again, in searching for a rationale behind this improvement it is natural to point to the size of the MILPs being solved and the number of branchings required to solve them (significantly smaller than for the big-Mapproaches). Also to be noted is that FBM performs significantly better than in the convex case, at least for some instances, but remains very distant from BM and PC in terms of overall performance.

These experiments somehow confirm the intuitive fact that the choice of objective function, even reasonably similar ones, can non-negligibly impact the performance of an algorithm. Therefore, it is important, particularly when dealing with real-life problems for which the definition of the objective can be fuzzy, to carefully elicitate and analyze the specifications, rules of thumb, wishes and goals of the stakeholders. In some cases, real-life may be simpler than the models. To give an example, the performance of the traffic management department of the Danish train operating company DSB is evaluated on a single indicator, namely the percentage of trains delayed over 3 minutes ([24]). Thus, practically speaking, their only true objective is to comply with this requirement. The resulting (binary) cost function is shown in Figure 13.

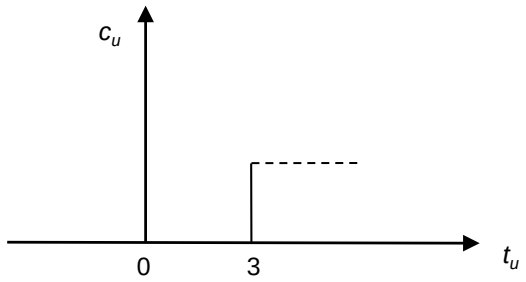

Figure 13: Example of binary function: 3-minutes-or-die.

We finally test and compare the performance of the two algorithms using the DSB cost function in Figure 13. We present the results in analogous Tables to those presented for the previous objectives. In Table 7 we give algorithmic information regarding the solution process when using the binary objective function. In general, both algorithms perform competitively, solving on average under 3 seconds for all time periods. Again however PC performs better than BM in 23 periods out of 25, "losing" by only decimals of a second in periods 2 04-08 and 2 20-24.

Like for the previous two objectives, in Figures 14 and 15 we show the frequency of instances being solved within certain time ranges and their cumulative distribution. Again, it can be noted that for any given computation time PC has cumulatively solved more, or at least as many, instances as BM. Finally, in Table 8 we show the performance of the algorithms aggregated over each day of instances also for the binary objective. PC performs better than BM on all 5 days, with the performance being very similar on day 2 and noticeably better on all the other days.

Period	Trains	Late	Instances	Run time (seconds)		Iterations		Iteration time (milliseconds)		Branchings		Initial				Generated			
												Rows		Columns		Rows		Columns	
				BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC
1 12-16	84	4	216	1.44	0.89	10	19	145	46	65	0	3944	0	2900	0	867	476	172	307
1 16-20	62	5	240	0.82	0.38	8	11	98	33	20	0	2827	0	2155	0	784	199	201	204
1 20-24	66	5	240	0.62	0.55	6	7	100	77	0	0	2977	0	2263	0	1441	134	372	187
2 04-08	115	7	222	2.91	3.04	10	23	294	135	70	0	5160	0	3805	0	3821	465	653	446
2 08-12	109	5	231	2.18	1.74	10	17	229	101	7	0	4933	0	3623	0	1574	370	345	388
2 12-16	84	3	240	1.07	0.96	9	17	126	57	7	0	3958	0	2914	0	922	343	184	298
2 16-20	62	4	240	0.63	0.48	7	10	91	47	3	0	2819	0	2149	0	652	166	178	196
2 20-24	66	5	224	0.86	0.83	7	9	130	91	0	0	2955	0	2246	0	1881	170	384	215
3 04-08	113	3	240	1.09	1.16	8	9	140	127	1	0	5070	0	3713	0	1415	222	292	313
3 08-12	105	4	163	0.74	0.47	7	7	104	69	0	0	4801	0	3502	0	495	164	114	258
3 12-16	86	2	240	0.71	0.52	8	10	95	53	0	0	4050	0	2973	0	470	209	121	259
3 16-20	57	4	238	0.75	0.40	8	10	91	39	5	0	2615	0	1978	0	627	154	140	196
3 20-24	38	3	240	0.39	0.11	6	5	70	20	0	0	1874	0	1464	0	528	77	123	109
4 08-12	40	1	240	0.21	0.08	5	3	45	24	0	0	2378	0	1785	0	163	60	54	100
4 12-16	30	1	240	0.29	0.04	5	3	58	13	0	0	1768	0	1322	0	123	45	45	75
4 16-20	23	1	237	0.30	0.02	6	2	53	10	0	0	1203	0	908	0	192	33	48	56
4 20-24	15	1	240	0.08	0.01	3	1	30	1	0	0	760	0	588	0	12	19	4	33
5 08-12	37	1	240	0.32	0.04	6	3	53	14	0	0	2383	0	1829	0	131	54	43	91
5 12-16	34	1	240	0.31	0.08	7	5	47	15	0	0	2162	0	1661	0	231	67	67	101
5 16-20	35	2	240	0.34	0.10	7	8	53	12	0	0	1910	0	1488	0	612	79	119	106
5 20-24	59	3	240	0.69	0.25	7	9	95	29	0	0	2753	0	2080	0	1314	121	237	170
6 04-08	112	3	239	1.48	0.69	8	9	197	73	0	0	5035	0	3678	0	1658	213	243	307
6 08-12	111	3	107	2.40	1.26	10	23	233	55	0	0	4954	0	3610	0	886	445	170	396

Table 7: Algorithmic information for binary objective function.

Day	Runs	Optimal (%)		Run time (s)		Speed up
		BM	PC	BM	PC	
26	696	100%	100%	0.95	0.60	1.58x
27	1398	100%	100%	1.41	1.40	1.01x
28	1362	100%	100%	0.75	0.52	1.44x
29	1434	100%	100%	0.24	0.04	6.00x
30	1440	100%	100%	0.32	0.09	3.56x

Table 8: Aggregate comparisons for 5 days of computations using the binary objective.

6 Final remarks and future developments.

The non-compact formulation introduced in this paper for certain job-shop scheduling problems arising in transportation networks favorably compares with the classic big-Mformulation, in particular on a class of non-convex cost functions which are commonly used in a real-life setting. A merit of the new approach is to open a number of potential research paths.

A first candidate for investigation is the fact that the constraints in the new formulation correspond to basic graph structures (such as paths, cycles and trees). This opens the way for strengthening the formulation through polyhedral work, identifying classes of valid, possibly facet-defining inequalities. Other interesting modeling developments to look into are the integration of this approach with routing, which was not considered in this paper, and the relaxation of the non-decreasing cost function assumption.

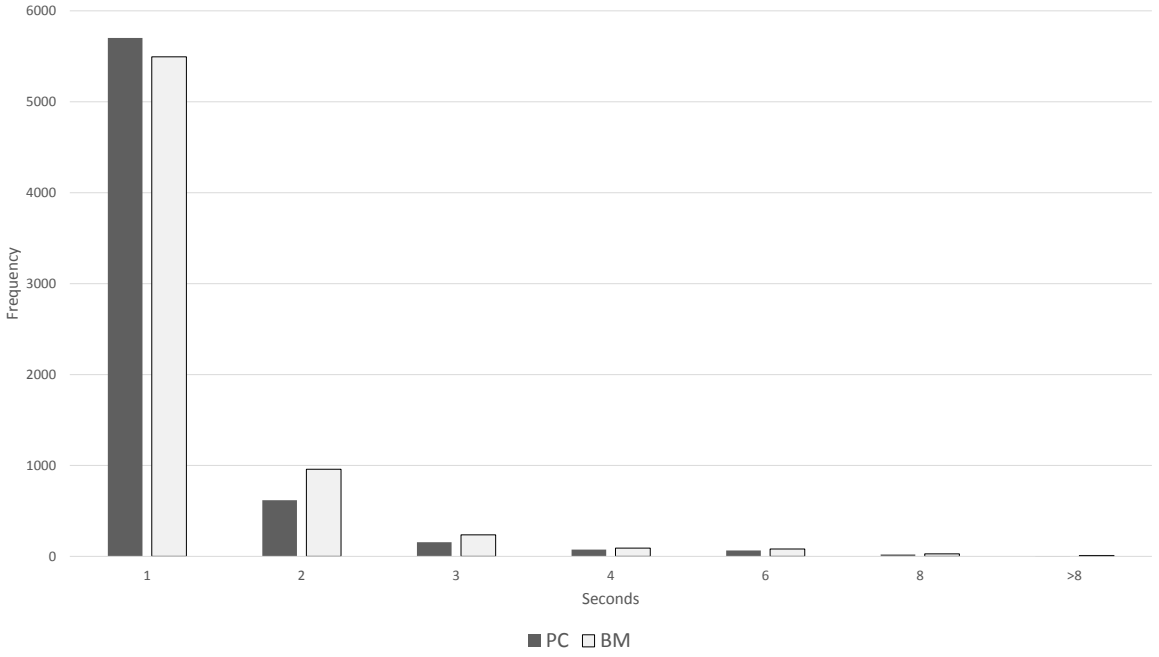

Figure 14: Histogram of the frequency with respect to computational time for instances in Table 7.

From an algorithmic standpoint, a natural enhancement is to separate the the path and cycle constraints during the Branch&Bound process, rather than only at the end.

Finally, the approach can be extended to tackle other related applications such aircraft scheduling and routing.

On a final note, as mentioned we expect this new formulation to be particularly suited for the class of job-shop scheduling problems that arise in transport traffic management because of their common structure. In particular, the fact that the number of "necessary" disjunctions is limited, typically in the order of the number of operations, and that the objective function is generally associated with the timing of few operations (e.g. the delay of a vehicle at a given location). It seems natural to extend the approach to handle other, similar traffic management problems, such as train delay management, runway scheduling and airport ground movement management. At a later stage, it may be interesting to also investigate its performance on other job-shop scheduling problems in possibly non-transport related application areas.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows*, Prentice-Hall, 1993.

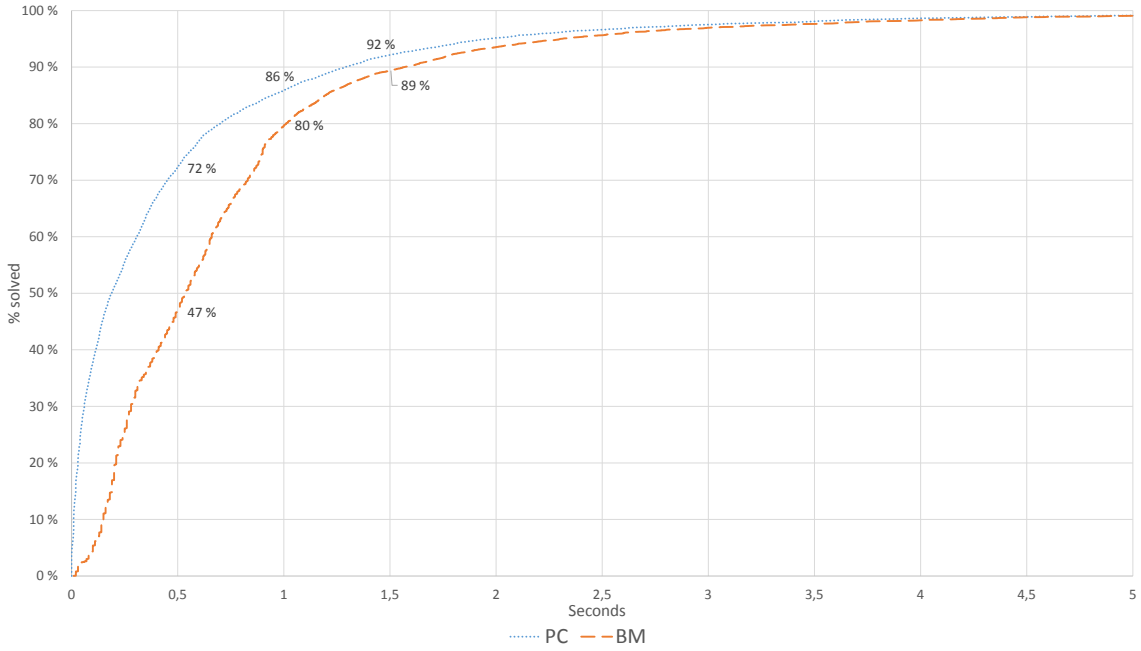

Figure 15: Cumulative distribution of the percentage of instances solved in Table 7 with respect to computation time.

[2] D. Alvras, M.W. Padberg, *Linear Optimization and Extensions: Problems and Soluzions*. Springer-Verlag, Berlin, Germany, 2001.

[3] E. Balas, Machine sequencing via disjunctive graphs, Operations Research 17 (1969) pp. 941–957.

[4] E. Balas, *Facets of the knapsack polytope*, Mathematical Programming 8 (1), pp. 146–164, 1975.

[5] E. Balas, *Disjunctive programming*, Annals of Discrete Mathematics, **5**, pp. 3–51, 1979.

[6] J.A. Bennell, M. Mesgarpour, C.N. Potts, *Airport runway scheduling*, *4OR* **9**(2) 115–138, 2011.

[7] D. Bertsimas, J. Tsitsiklis, *Introduction to linear optimization*. Vol. 6. Belmont, MA: Athena Scientific, 1997

[8] P. Bonami, A. Lodi, A. Tramontani, A., S. Wiese, 2015, *On mathematical programming with indicator constraints*, Mathematical Programming, 151(1), pp. 191–223, 2015.

[9] V. Cacchiani, D. Huisman, M. Kidd, L. Kroon, P. Toth, L. Veelenturf, J. Wagenaar, *An overview of recovery models and algorithms for real-time railway rescheduling*, Transportation Research Part B, 63, pp. 15–37, 2014.

[10] V. Cacchiani, P. Toth *Nominal and robust train timetabling problems*, European Journal of Operational Research 219, pp 727-737, 2012.

[11] G. Codato, M. Fischetti, *Combinatorial Benders' Cuts for Mixed-Integer Linear Programming*, Operations Research, 54 (4), pp. 756-766, 2006.

[12] F. Corman, L.Meng, *A review of online dynamic models and algorithms for railway traffic management*, Intelligent Transportation Systems, IEEE Transactions on 16 (3), pp. 1274-1284, 2015.

[13] G. Desaulniers, J. Desrosiers, M.M. Solomon, eds. *Column generation*, Vol. 5. Springer Science & Business Media, 2006.

[14] M. Dyer., L. Wolsey, *Formulating the single machine sequencing problem with release dates as a mixed integer program*, Discrete Applied Mathematics, no. 26 (2-3), pp. 255-270, 1990.

[15] M. Fischetti, M. Monaci, *Using a general-purpose MILP solver for the practical solution of real-time train rescheduling*, Tech Rep. DEI, Univ. Of Padova, 2016.

[16] D. Kjenstad, C. Mannino, P. Schittekat, M. Smedsrud, Integrated surface and departure management at airports by optimization, IEEE Xplore Digital Library, 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO), Hammamet 2013.

[17] L. Lamorgese, C. Mannino, An exact decomposition approach for the real-time Train Dispatching problem, *Operations Research*, 63 (1), pp. 48-64, 2015.

[18] L. Lamorgese, C. Mannino, M. Piacentini, Optimal Train Dispatching by Benders'-like reformulation, *Transportation Science*, published on line (http://pubsonline.informs.org/doi/10.1287/trsc.2015.0605), 2016.

[19] L. Lamorgese, C. Mannino, E. Natvig, An exact micro-macro approach to cyclic and non-cyclic train timetabling, *Omega*, 72, 59–70, 2017.

[20] C. Mannino, A. Mascis, *Real-time Traffic Control in Metro Stations*, Operations Research, 57 (4), pp 1026-1039, 2009

[21] H. Marchand, L.A. Wolsey, *The 0-1 knapsack problem with a single continuous variable*, Mathematical Programming 85 (1), pp. 15–33, 1999.

[22] A. Mascis, D. Pacciarelli, *Job shop scheduling with blocking and no-wait constraints*, European Journal of Operational Research, 143 (3), pp. 498–517, 2002.

[23] G.L. Nemhauser, L.A. Wolsey, Integer and Combinatorial Optimization, Wiley-Interscience, 1999.

[24] M.N. Nielsen, *Danish State Railways*, presentation at the Dagstuhl seminar on Algorithmic Methods for Optimization in Public Transport, available at http://materials.dagstuhl.de/index.php?semnr=16171 (last access 28.06.2016).

[25] https://www.forskningsradet.no/prosjektbanken/#/project/NFR/267554/Sprak=en.

[26] M.L. Pinedo, *Scheduling: theory, algorithms, and systems*, Springer Science & Business Media, 2012.

[27] Queyranne, M., A.S. Schulz, *Polyhedral Approaches to Machine Scheduling*, Tech Rep. 408/1994, Technische Universitat Berlin, 1994.

[28] M. Schachtebeck and A. Schöbel, *To Wait or Not to Wait - And Who Goes First? Delay Management with Priority Decisions*, Transportation Science, 44 (3), pp. 307–321, 2010.

[29] A. Schrijver, *Combinatorial Optimization*, Springer, 2003.

[30] T. Schlechte, R. Borndörfer, B. Erola, T. Graffagnino, E. Swarat, *Micro-macro transformation of railway networks*, Journal of Rail Transport Planning & Management, 1, pp. 38–48, 2011.

[31] T. Vidal, T.G. Crainic, M. Gendreau, C. Prins, *Timing problems and algorithms: Time decisions for sequences of activities*, Networks, 65(2), pp. 102–128, 2015.

[32] E. Zemel, *Lifting the facets of zero-one polytopes*, Mathematical Programming 15, pp. 268–277, 1978.

7 Appendix: proof of Lemma 3.4

To prove Lemma 3.4 we use the following result by Marchand and Wolsey ([21]):

Lemma 7.1 *Let* $Y = \{(y,\eta) \in \{0,1\}^{|N|} \times I\!R^1_+ : \sum_{j\in N} a_j y_j \leq b + \eta\}$, *where* $0 < a_j \leq b$ *for* $j \in N$. *Let* $Y_0 = \{(y,\eta) \in Y : \eta = 0\}$.

If $\sum_{j\in N} \pi_j y_j \leq \pi_0$ *with* $\pi \geq 0$, $\sum_{j\in N} \pi_j > \pi_0 > 0$ *defines a facet of* $conv(Y_0)$, *then* $\sum_{j\in N} \pi_j y_j \leq \pi_0 + \frac{\eta}{\beta}$ *defines a facet of* $conv(Y)$, *where* $\beta = \min_{\eta>0} \frac{\eta}{\sigma(\eta)-\pi_0}$, $\sigma(\eta) = \max\{\sum_{j\in N} \pi_j y_j : \sum_{j\in N} a_j y_j \leq b + \eta, y \in \{0,1\}^{|N|}\}$.

Proof of Lemma 3.4. We first show that (17) is facet defining for the $conv(Y^N)$, where

$$Y^N = \{(y,\eta) \in \{0,1\}^{|N|} \times I\!R^1_+ : \sum_{e\in N} M_e y_e \leq \sum_{e\in N} M_e - c(H) + \eta\} \tag{24}$$

If $N = \emptyset$, both (17) and the knapsack defining Y^N reduces to $c(H) \leq \eta$ and the lemma follows trivially.

If $N = \{e\}$, the knapsack defining Y^N reduces to $M_e y_e \leq M_e - c(H) + \eta$. It is not difficult to see that $Y^N = \{\{(y_e = 0, \eta) : \eta \geq 0\} \cup \{(y_e = 1, \eta) : \eta \geq c(H)\}\}$, for which $conv(Y^N)$ is defined by the trivial facets $y_e \geq 0$, $y_e \leq 1$ and $c(H)y_e \leq \eta$, the latter coinciding with (17).

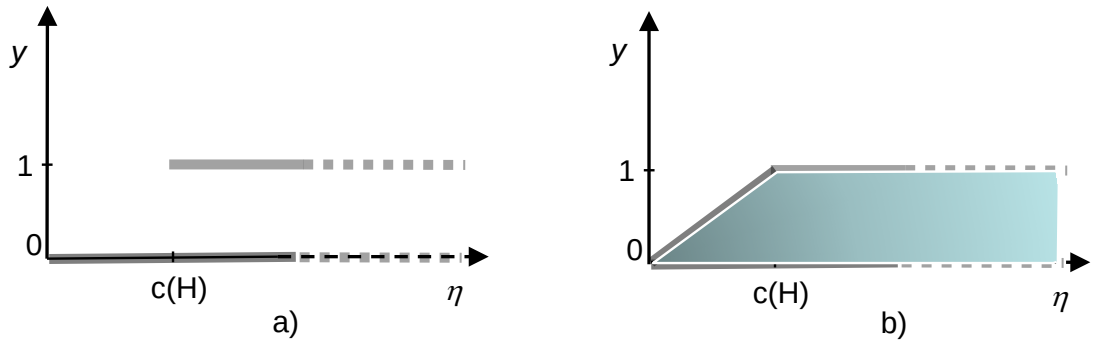

Figure 16: a) Y^N for $|N| = 1$ (in gray) b) $conv(Y^N)$

The remaining case is $|N| \geq 2$, for which we will use the same notation of Lemma 7.1. For $\eta = 0$, the knapsack defining Y^N reduces to the following binary knapsack:

$$\sum_{e \in N} M_e y_e \leq \sum_{e \in N} M_e - c(H) \tag{25}$$

Observe that, for $e \in N$, we have $0 < M_e \leq \sum_{e \in N} M_e - c(H) = b$ and the conditions of Lemma 7.1 are satisfied. The only (minimal) cover of the above knapsack constraint is N. Therefore, the cover inequality

$$\sum_{e \in N} y_e \leq |N| - 1 \tag{26}$$

is facet defining for the polyhedron $conv(Y_0^N)$.

So, we have

$$\sigma(\eta) = \max\{\sum_{e \in N} y_e : \sum_{e \in N} M_e y_e \leq \sum_{e \in N} M_e - c(H) + \eta\} = \begin{cases} |N| - 1, & 0 \leq \eta < c(H) \\ |N|, & c(H) \leq \eta \end{cases}$$

and thus

$$\beta = \min_{\eta>0} \frac{\eta}{\sigma(\eta) - (|N| - 1)} = \min_{\eta>0} \begin{cases} \frac{\eta}{|N|-1-(|N|-1)}, & 0 \leq \eta < c(H) \\ \frac{\eta}{|N|-(|N|-1)}, & c(H) \leq \eta \end{cases} = c(H).$$

Having computed β we can substitute in the expression in the statement of Lemma 7.1 and the constraint

$$\sum_{e \in N} y_e \leq |N| - 1 + \frac{\eta}{c(H)} \tag{27}$$

is facet defining for $conv(Y^N)$. To complete the proof we need to show that (27) is also facet defining for $conv(Y^H)$, as N may be a proper subset of $F \cap H$. But since $M_e = 0$ for $e \in (F \cap H) \setminus N$, this derives from simple lifting results (see [32]). □