Almost 20 Years of Combinatorial Optimization for Railway Planning: from Lagrangian Relaxation to Column Generation

Alberto Caprara

DEIS, University of Bologna
Viale Risorgimento 2
40136 Bologna, Italy
`alberto.caprara@unibo.it`

Abstract

We summarize our experience in solving combinatorial optimization problems arising in railway planning, illustrating all of these problems as integer multicommodity flow ones and discussing the main features of the mathematical programming models that were successfully used in the 1990s and in recent years to solve them.

1998 ACM Subject Classification G.2.2 Graph Theory, Network problems; G.2.2 Graph Theory, Path and circuit problems; G.2.3 Applications

Keywords and phrases Railway Planning, Integer Multicommodity Flow, Integer Linear Programming Formulations, Lagrangian Relaxation, Column Generation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.1

1 Introduction

In this work we summarize our experience in solving combinatorial optimization problems arising in railway planning. Everything started in the early 1990s and has kept on going until today (and hopefully it will continue also later!) with various projects, some of which funded by the EC. The specific subjects of these projects, that we will briefly overview throughout the paper anyhow, are of limited interest here.

What we think may be interesting to a general audience, and so the real contribution of this work, are three things. First, we make an attempt to classify all of these problems within a common framework, in particular identifying two main categories under which these problems seem to fall. Second, besides noting that the best thing to be done to tackle these problems was to model them as *(Mixed-)Integer Linear Programs* (ILPs), which is what we tell our students a few minutes after the beginning of our introductory courses, we point out the big methodological change that we observed in the way these ILPs were approached, essentially related with the concurrent improvements in computing power and algorithmic technology for the solution of *Linear Programs* (LPs). Third, after having widely stressed the fact that the best way to approach these problems at present seems to be the use of very large ILP relaxations tackled with the combined use of general-purpose LP solvers and column generation (or pricing), we illustrate in a general context the main successful features of our approaches of this type.

All these problems have in common that they arise at the *planning level*, i.e., they have to be solved every once in a while (for instance twice a year, when a new passenger timetable is published) with plenty of computing power and time available. By *planning horizon* we mean the period to which the solution of the problems above applies (e.g., six months). On

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 1–12
OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

the other hand, the associated instances are generally of large size, so the models and solution approaches have to be defined with some care. Historically, these problems used to be solved (and in some cases are still solved) by hand by human experts. The associated solutions were generally of good quality (though "slightly infeasible in a tolerable way", a concept that has always been very hard to formalize), and often optimization-based approaches could not do any better. On the other hand, getting the hand-designed solution generally took all of the available time to the experts, and any what-if analysis varying some of the parameters was completely out of the question.

This work is organized as follows. In Section 2 we present the problems considered under a unique framework, limiting ourselves to considering their essential characteristics and leaving the details of the real-world cases to the surveys [6, 13, 16, 18, 21], and of course also to the specific papers considering each of them, listed in the references. In Section 3 we illustrate the changes in the most successful way to tackle these problems over the last 20 years, outlining the main features of the solution approaches. Finally, in Section 4 we discuss the main modelling issues and their implications on the solution methods.

2 Problem Classification: "Assignment" vs. "Timetabling" Problems

Regardless of the fairly different nature of the problems mentioned in the previous section, it seems to be possible to see each of them as the following *Integer Multi-Commodity Flow* (IMCF) problem [2] on a suitable graph, with additional side constraints. IMCF is formally defined by a set of commodities C and a directed multigraph $G = (V, A)$, whose vertex set V contains a *dummy source* s and a *dummy sink* t and whose arc set A is partitioned into different sets A_c, one for each commodity $c \in C$. The problem calls for finding a collection of paths from s to t, one for each commodity $c \in C$ composed by arcs in A_c, so that a suitable objective function is minimized and a suitable set of side constraints relating the paths in the collection is met. For convenience, the *dummy arc* (s, t) may be in A_c for some commodity $c \in C$, and the associated *dummy path* may be selected in the solution.

2.1 "Assignment" Problems

The first class of problems we addressed are *Train Unit (TU)* and *Crew Assignment* problems, see, e.g., [8, 12, 13]. In TU and Crew Assignment, the nodes in $V \setminus \{s, t\}$ in the IMCF graph correspond to services to be covered, referred to as *trips*, which are part of a train timetable that must be performed by the same composition of TUs/crews without changes (trips may be different for TUs and crews since changing a crew composition is generally much easier than changing a TU composition). Each commodity $c \in C$ is either a TU or a crew, and a path $P \subseteq A_c$ from s to t corresponds to the sequence of trips that the TU/crew has to perform within the planning horizon. The dummy path for a commodity represents the fact that the associated TU/crew is not used in the solution. Generally, each commodity $c \in C$ has a cost f_c which is paid if the corresponding TU/crew is used, and the objective function is the minimization of the total cost of the TUs/crews used. Finally, each arc $(u, v) \in A_c$ represents the fact that trip v can be performed right after trip u by TU/crew c. This means that there is enough time between the end of trip u and the start of trip v, and that either u ends at the same station at which v starts, or there are the possibility and the time to move the TU/crew c between the two stations. In the latter case, the associated *deadhead* cost for the movement between the two stations may be assigned to arc (u, v).

The side constraints require that each node is "covered" by the paths in the collection. To this purpose, in some cases it suffices to have at least one path visiting the node, i.e., at least one TU/crew performing the trip, which may be called *elementary composition.* Note that generally one does not require exactly one TU/crew since the trip, although already covered, may be used to transfer other TUs/crews to the departing station of their next trip. In some other cases, that may be referred to as *proper composition*, each node $v \in V$ has a demand d_v, each path associated with a commodity $c \in C$ has a capacity a_c, and the total capacity of the paths visiting v must be at least d_v. This happens, e.g., for TU Assignment to passenger trips, in which d_v is the (estimated) number of passengers travelling on the trip and a_c is the capacity of TU c. Clearly, a special case of proper composition is the one in which each trip must be covered by at least a given number of TU/crews.

2.2 "Timetabling" Problems

The second class of problems considered is the one of *Train Timetabling* and *Train Platforming* problems, see, e.g., [5, 7, 14, 15, 24]. In Train Timetabling and Platforming, the nodes in $V \setminus \{s, t\}$ in the IMCF graph correspond to resources of limited capacity, referred to as *events*, which are departures/arrivals of a train at a given point (a station track or a platform) and at a given time instant. Each commodity $c \in C$ is a train, and a path $P \subseteq A_c$ from s to t corresponds to the sequence of events associated with the itinerary followed by the train and the associated timetable. Note that trains may also be cancelled, i.e., not scheduled at all, in which case the dummy path is selected for the associated commodity. Finally, each arc $(u, v) \in A_c$ represents the fact that event v can follow event u for train c. This means that the time elapsing between u and v is either appropriate for train c to travel from the point of u to the point of v (in case u and v are associated with distinct points, e.g., two consecutive stations along a railway corridor), or for train c to stand at the point of u and v (if it is the same, e.g., u represents an arrival at a platform and v the departure from the platform, in which case the time must be appropriate for the operations to be performed for the train at the platform).

Each train should possibly be scheduled in a certain preferred way (e.g., using a given path). However, due to the side constraints illustrated below, not all trains can be scheduled in this way, and there are suitably-defined penalties for deviations (including cancellations). The objective function is the minimization of the total penalties. Penalties for deviations of a train $c \in C$ are either modelled by assigning costs to nodes $u \in V$, in case the event for the train does not take place at the preferred time instant, or to arcs $(u, v) \in A_c$, in case the time elapsing between u and v is different with respect to the preferred schedule. This is the way to model penalties as the cost of the path associated with a train, given by the sum of the costs of its nodes and arcs.

The side constraints require each node not to be contained in more than one path in the collection. More generally, for each node $v \in V$, there exists a set $I_v \subseteq V$ (with $v \in I_v$) of nodes that are *incompatible* with v, i.e., v can be visited by a path in the collection only if no other node in I_v is visited by some other path in the collection. This models the fact that trains cannot depart/arrive at the same point not only at the same time, but also too close in time. For tracks, this is a natural safety requirement, whereas for platforms this prevents the presence of more than one train simultaneously standing at the platform (besides imposing a minimum time between platform occupations by distinct trains). Even more generally, the incompatibility between two paths associated with different commodities may not be simply related with the fact that they visit two incompatible nodes, but with their overall structure, i.e., with the overall sequence of nodes visited by the two paths. In this more general case, there may be simply an *oracle* indicating, given the two paths, if they are incompatible or not.

2.3 A note on the objective function

Generally, speaking, for “assignment” problems the objective function tends to be well-defined as it is the sum of the costs of the TUs/crews employed, which are generally fairly high. So, although solutions that are robust towards disruptions are certainly very important in practice, it is highly desirable not to increase the nominal cost by a too large amount in order to achieve robustness. Viceversa, for “timetabling” problems, it tends to be less clear what would be the preferred schedule for a train, and to quantify the associated penalties for deviations with respect to this schedule. This makes the objective function mostly not well-defined, so robustness issues tend to play a major role in this case. In any case, robustness issues are out of the scope of this work.

3 Solution Methods

A common feature of all real-world applications addressed here is that pretending a priori to find a provably optimal solution is generally hopeless, and one has to resort to heuristics. Moreover (and fortunately) the best heuristic methods known for all these problems are based on mathematical programming (mainly ILP) formulations, which are driven by appropriate (mainly LP) relaxations of these formulations.

IMCF calls for a min-cost collection of *paths* of G, each associated with arcs of the same commodity, with several constraints on the feasibility of single paths and on the compatibility between distinct paths. This can either be expressed by *arc ILP models* using binary variables $x_{c,(i,j)}$ equal to 1 if arc $(i,j) \in A_c$ is in the path for commodity c in the solution, or by *path ILP models* considering the whole list $\mathcal{P}_c$ of paths for commodity c and binary variables $x_{c,P}$ equal to 1 if $P \in \mathcal{P}_c$ is the path for commodity c in the solution. Side constraints can be written in a similar way in the two models (see below). Of course, path ILP models have a number of variables that may be exponential in the size of G, and should be solved by column generation (or sometimes simply by pricing, see below).

The LP relaxations of the arc and path ILP models are equivalent for IMCF, so the decision on whether to use the former or the latter depends on the solution approach. Here comes the main difference between the state-of-the-art approaches of today with respect to those of the 1990s.

3.1 Old days: Lagrangian relaxation for arc ILPs

Twenty years ago, solving the LP relaxations of both arc and path ILP formulations was sometimes possible but so much time consuming that it was advisable to avoid it. Therefore, it was natural to use methods capable of solving these LP relaxations approximately, both able to provide bounds on the integer optimum (though weaker than the LP optimum itself) and to drive successfully a diving heuristic search. Among these, the most successful appeared (and still appears) to be the combined use of *Lagrangian relaxation* (see, e.g., [19]) and *subgradient optimization* (see, e.g., [23]) applied to the arc ILP formulation. Compared to other alternatives to LP relaxation, Lagrangian relaxation has the advantage over *relaxation by elimination* that all constraints are somehow taken into account, and over *surrogate relaxation* that the relaxed problems are easier to solve. Moreover, the simplicity of the relaxed problem makes the easy-to-implement subgradient optimization a suitable solution method, since its slow convergence (in terms of number of iterations) with respect to more advanced variants such as *bundle* (see, e.g., [20]) is compensated by the very short time per iteration. Moreover, many iterations are an advantage for a heuristic since they mean many slightly different near-optimal Lagrangian multipliers that can be used to produce many slightly different candidate heuristic solutions out of which the best one is kept [1, 4, 10, 17].

In this work, for space reasons, we omit the description of specific successful features of the Lagrangian approaches. These approaches remain however the ones we suggest when the problem is so large that there seems to be no hope of solving the LP relaxation of whatever reasonable ILP formulation. The interested reader is referred, e.g., to [19, 20, 23].

3.2 These days: Column generation/pricing for path ILPs

Nowadays, while the solution of the LP relaxation of the arc ILP formulations still appears to be too time consuming (besides requiring a large amount of core memory), the LP relaxation of the path ILP formulations can generally be found quickly by column generation or pricing (see below). The huge advantage of path over arc ILP formulations was systematically true in all the applications we considered. It does not seem to be shared by all analogous applications, as testified by the surprised reaction we could note by many audiences and referees in these years. (From our side, we could only note that path ILP formulations do not seem to be suited to be used together with Lagrangian relaxation.)

In all cases, the number of variables in the path ILP formulations is generally too large to consider all of them explicitly in the LP relaxation, so one has to work with a *current LP* that is (much) smaller than the whole LP relaxation, iteratively adding variables to it. In some cases *column generation* is required, solving at each LP iteration an optimization problem to check if there is some variable with negative reduced cost (or positive reduced profit) to be added to the current LP. In some other cases, column generation is not strictly necessary as *pricing* suffices, corresponding to storing in memory the whole list of variables and explicitly computing their reduced costs at each iteration, adding (some of) those variables for which they are negative. Finally, there are cases in which the number of constraints to be considered is also very large, and *separation* has to be used, in order to keep only a small subset of constraints in the current LP, detecting and adding further violated constraints either by solving an optimization problem or by considering all missing constraints and checking if they are violated.

Fairly general issues concerning the definition of the path ILP formulations for the applications we considered and the solution of the associated LP relaxations are discussed in Section 4. To follow that section, some familiarity with the notions of column generation, pricing and separation is required. For a detailed discussion of these, we refer the interested reader to [3, 22].

A big advantage of working with path ILPs and column generation/pricing with respect to Lagrangian relaxation and subgradient optimization is that in the latter a few parameters have to be tuned in a proper way (at the risk of not converging at all otherwise), whereas the former appears to be a fairly robust method both with respect to the initial set of variables inserted in the LP and with the policy used to add columns at each iteration (many versus few, the most violated versus any violated). Moreover, also with column generation one has many similar near-optimal LP solutions to drive a heuristic.

The main limitation of column generation is that it restricts the formulation of the side constraints, since the structure of the column generation problem itself depends on the form of these constraints, as we will discuss in Section 4. Therefore, in addition to the classical compromise between the strength of the LP relaxation and the ability to handle the associated constraints, possibly by separation, one has to take into account also the ability to generate the columns.

Note that branching is not a real issue in this context, as one is limited to diving heuristics (see Section 3.3) that fix to 1 variables that are fractional in the LP relaxation, imposing a path in the solution. Such a fixing poses no problem to column generation. (Fixing a variable to zero, instead, as a branching dichotomy would naturally do, gives a lot of troubles to column generation, see, e.g., [3].)

3.3 Evergreen: diving heuristics

The most successful heuristics used in combination with whatever relaxations of the ILP formulations above are *diving* heuristics, that solve a suitable relaxation, fix some of the variables according to the relaxed solution, solve again the relaxation subject to the fixing constraints, and so on, until a feasible solution is found. Besides outperforming metaheuristics on all the applications we considered, these diving heuristics have the advantage of certifying the quality of the solutions through bounds obtained by solving the initial relaxation. In fact, it may turn out that for some (or most of) the instances considered the bounds certify a posteriori that the solution found is indeed optimal.

Plenty of details on possible heuristic implementations are given in the references for the interested reader. In any case, the main theme is the diving scheme, that has to be properly adapted to the specific application at hand.

4 Defining and Solving Path ILP Formulations of IMCF

As already mentioned, for all the applications we addressed we always found that the path ILP formulation of IMCF was the best, if not the only viable, choice for approaches based on solving the LP relaxation by general-purpose solvers. All these formulations deal with binary variables $x_{c,P}$ for each commodity $c \in C$ and path $P \in \mathcal{P}_c$, with an associated cost $f_{c,P}$ paid if $x_{c,P} = 1$. This ILP calls for the minimization of the total cost:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_c} f_{c,P}\; x_{c,P} \tag{1}$$

subject to the constraint that one path is selected for each commodity:

$$\sum_{P \in \mathcal{P}_c} x_{c,P} = 1, \quad c \in C. \tag{2}$$

How to model the missing side constraints depends on the specific application. For each commodity $c \in C$ and vertex $v \in V$, let $\mathcal{P}_{c,v}$ denote the sublist of paths for commodity c that visit vertex v. Moreover, for a path $P \in \mathcal{P}_c$ for some commodity $c \in C$, let V_P denote the set of nodes visited by P.

4.1 Assignment Problems

The side constraints in TU/crew assignment must ensure that all the trips are covered. As we will see, this poses no problem to column generation, which is a great advantage with respect to the timetabling case. The unique issue to be addressed, as for regular ILP formulations (with a reasonable number of variables), is to have a strong LP relaxation, yielding lower bounds close to the integer optimum and possibly integer solutions quickly in a diving heuristic, after having fixed a relatively small number of binary variables to 1.

Set Covering constraints

From an LP relaxation strength viewpoint, the best situation is the case of elementary composition, since the side constraints are expressed in the classical form of Set Covering constraints, and are already as strong as possible:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} x_{c,P} \geq 1, \quad v \in V. \tag{3}$$

In this case, there is no real modelling issue to be addressed, and the best way to proceed is to combine the solution of the LP relaxation with column generation or pricing [11].

Knapsack constraints

For proper composition the situation is more complex. Recalling that d_v is the demand of each vertex v and a_c the capacity of each commodity c, the natural way to write these constraints is:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} a_c \, x_{c,P} \geq d_v, \quad v \in V. \tag{4}$$

Unfortunately, the constraints in this form tend to be fairly weak, as it is typically the case for knapsack-type constraints with large coefficients. Weakness does not only mean poor lower bounds obtained by solving the LP relaxations, but also bad quality solutions for heuristics driven by these relaxations, as discussed in [8].

The easiest possibility to strengthen (4) is to re-compute the coefficients so that they are minimal, i.e., for each commodity c there is always a combination of other commodities such that the sum of the associated capacities is exactly d_v. Formally, this amounts, for each vertex v, to first redefine its demand d_v so that it can be met exactly by a subset of commodities, i.e., as:

$$\min_{S \subseteq C} \left\{ \sum_{c \in S} a_c : \sum_{c \in S} a_c \geq d_v \right\},$$

Then, one may ensure that all coefficients are minimal (not only for the commodities belonging to subsets whose total capacity is exactly d_v) by defining initially $a_{c,v} := a_c$ for $c \in C$ and then, iteratively for $c \in C$, by redefining $a_{c,v}$ as:

$$\max_{S \subseteq C \setminus \{c\}} \left\{ d_v - \sum_{d \in S} a_{d,v} : a_{c,v} + \sum_{d \in S} a_{d,v} \geq d_v \right\},$$

iterating the replacement until the coefficient does not change for all commodities (note that the final result depends on the order in which the coefficients have been considered). The resulting improved inequalities read:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} a_{c,v} \, x_{c,P} \geq d_v, \quad v \in V. \tag{5}$$

However, having minimal coefficients often does not lead to a strong enough LP relaxation.

The only alternative to get stronger constraints is to add more than one constraint for each vertex v, getting say m constraints of the form:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} b^i_{c,v} \, x_{c,P} \geq d_v, \quad v \in V, \ i = 1, \ldots, m. \tag{6}$$

The ideal case is when the associated inequalities

$$\sum_{c\in C} b^i_{c,v}\; y_c \ge d_v, \quad i=1,\dots,m, \tag{7}$$

define the convex hull of the Knapsack polytope:

$$\operatorname{conv}\left\{ y \in \{0,1\}^C : \sum_{c\in C} a_c\; y_c \ge d_v \right\}. \tag{8}$$

This is unfortunately impractical unless the number of commodities is small, or there are other side constraints that simplify the situation (e.g., in [8] the convex hull has an elementary structure since each vertex must be covered by at most two commodities). In any case, it is much better if the inequalities are at least a subset of the facets of the Knapsack polytope (8).

Column generation

The structure of the column generation problem is the same whatever the form of the covering constraints. Namely, consider the most general form (5) and let β^i_v denote the dual variables associated with these constraints. Letting α_c be the dual variables associated with constraints (2), the reduced cost of variable $x_{c,P}$ is given by $f_{c,P} - \alpha_c - \sum_{v\in V_P}\sum_{i=1}^m b^i_{c,v}\; \beta^i_v$, so the column generation problem amounts to finding a minimum-cost path from s to t in G with arcs in A_c and costs associated with the nodes in V (and possibly also with the arcs in A_c, depending on the way cost $f_{c,P}$ is defined).

4.2 Timetabling Problems

The side constraints in train timetabling/platforming impose that no two incompatible nodes are visited by some path. The easiest (but also essentially the weakest) way to force this is to consider all pairs of incompatible nodes $u, v \in V$ and state that at most one path visiting one of these two nodes is selected:

$$\sum_{c\in C}\sum_{P\in\mathcal{P}_{c,v}} x_{c,P} + \sum_{c\in C}\sum_{P\in\mathcal{P}_{c,u}} x_{c,P} \le 1, \quad v\in V, u\in I_v \setminus \{v\}. \tag{9}$$

The number of these constraints is polynomial, so even if they are too many to be handled explicitly by the LP solver at hand, they can be separated by simple enumeration.

Node incompatibility constraints

In order to formalize the incompatibility relation, which is extremely common in combinatorial optimization, one can represent it by an auxiliary undirected *node incompatibility graph* $I = (V, E)$, in which the neighbors of each node $v \in V$ are precisely its incompatible nodes I_v. The side constraints can then be imposed in a much stronger form by considering maximal subsets of pairwise incompatible nodes and stating that at most one path visiting one of the nodes in a subset is selected. Formally, let $\mathcal{K}$ denote the list of the maximal *cliques* of I, i.e., $K \in \mathcal{K}$ if K is a maximal vertex subset so that $(u, v) \in E$ for each pair $u, v \in K$. The side constraints can be modelled as:

$$\sum_{v\in K}\sum_{c\in C}\sum_{P\in\mathcal{P}_{c,v}} x_{c,P} \le 1, \quad K\in\mathcal{K}. \tag{10}$$

In general these constraints are exponential in number but can often be handled conveniently by separation. In particular, the complexity of their separation is the same as the complexity of finding a maximum-weight clique in I, where the weight of each vertex $v \in V$ is given by $\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} x_{c,P}$. Even if finding a maximum-weight clique in I is computationally too heavy, one may resort to heuristics, making sure that the final violated constraint added to the current LP relaxation is associated with a maximal clique of I by possibly adding nodes if the clique found by a heuristic is not maximal. Moreover, the use of (10) can easily be combined with column generation, as explained below. A discussion on various ways to define and handle these constraints in the Train Timetabling case can be found in [7, 9].

Path incompatibility constraints

Constraints (10) can also be seen as associated with cliques of the following *path incompatibility graph* $J = (\mathcal{P}, F)$, where $\mathcal{P} := \bigcup_{c \in C} \mathcal{P}_c$, which provides more information, but is also much larger, than the node incompatibility graph I above. The path incompatibility graph contains a node for each path $P \in \mathcal{P}$, i.e., for each variable $x_{c,P}$, and an edge in F joining each pair of incompatible paths, corresponding to two variables that cannot both take the value 1 due to constraints (10). Namely, two paths are incompatible if they either are associated with the same commodity or visit a pair of incompatible nodes. Moreover, with the path incompatibility graph one may represent incompatibilities that are not associated with nodes, specified by an oracle as discussed at the end of Section 2.2.

Letting $\mathcal{L}$ denote the list of the maximal cliques of J, alternatively to (10) the side constraints can be expressed by:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_c \cap L} x_{c,P} \leq 1, \quad L \in \mathcal{L}. \tag{11}$$

Even in case all incompatibilities are associated with nodes of G, constraints (11) may be much stronger than (10), although this does not seem to happen in practice. The first evident disadvantage of using (11) rather than (10) is that their separation may be much more complex. Even worse, their use is generally not compatible with column generation, so it is essentially limited to the cases in which all variables can be listed explicitly and pricing can be used. In the latter case, the only issue to be addressed is separation, which becomes easy in case one considers the relaxation of (11) involving paths associated with only two commodities, as discussed below.

Column generation

The structure of the column generation problem heavily depends on the form of the incompatibility constraints. For the form (10) let β^K denote the dual variables associated with these constraints and, as before, α_c the dual variables associated with constraints (2). Moreover, let $\mathcal{K}_v$ be the sublist of the maximal cliques of I containing node $v \in V$. The reduced cost of variable $x_{c,P}$ is given by $f_{c,P} - \alpha_c - \sum_{v \in V_P} \sum_{K \in \mathcal{K}_v} \beta^K$, and again the column generation problem amounts to finding a minimum-cost path from s to t in G with arcs in A_c and costs associated with the nodes in V and the arcs in A_c. The case of constraints (9) is analogous.

On the other hand, for constraints (11), the associated dual variables γ^L should be associated with all paths belonging to clique L in J. Therefore, in the column generation problem, one should charge the cost γ^L to the path being generated only if it belongs to L, which is problematic in general since this condition depends on the path itself. Note that the tempting trivial trick to consider all variables not belonging to the current LP as having coefficient 0 in all inequalities (11), so as to forget about the γ^L values in the generation, *does not work!* Indeed, one may end-up generating an already existing variable, having a nonnegative reduced cost in the current LP, since this reduced cost was underestimated by the (wrong) trick.

Oracle incompatibilities without column generation

As already mentioned, incompatibilities defined by an oracle and not associated with nodes of G can generally be handled, by using constraints (11), only if column generation is not required. The issue to be addressed in this case is the separation of (11), corresponding (in optimization version) to the determination of a maximum-weight clique in J. This itself is generally very complex, but can be greatly simplified if one considers the relaxation of (11) in which only two commodities are involved:

$$\sum_{P_1 \in \mathcal{P}_{c_1} \cap L} x_{c_1,P_1} + \sum_{P_2 \in \mathcal{P}_{c_2} \cap L} x_{c_2,P_2} \leq 1, \quad L \in \mathcal{L}, c_1, c_2 \in C. \tag{12}$$

In this case, the separation calls for a maximum-weight clique in the subgraph of J induced by the paths in $P_{c_1} \cup P_{c_2}$. Given that all paths in P_{c_1} (or P_{c_2}) are pairwise incompatible, this subgraph is the complement of a bipartite graph. Complementing everything, the separation problem calls for a maximum-weight stable set in a bipartite graph, which can be found efficiently by flow techniques. For details, see [15].

4.3 Linearizing quadratic objective functions

In some cases the objective function happens to be quadratic in the variables (if not multilinear in general). This happens, e.g., for Timetabling Problems when two paths $P_1 \in \mathcal{P}_{c_1}$ and $P_2 \in \mathcal{P}_{c_2}$ are "slightly" incompatible and, rather than forbidding the selection of both in the solution, one wants to penalize it with a penalty g_{c_1,P_1,c_2,P_2}. The resulting quadratic objective function reads:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_c} f_{c,P}\ x_{c,P} + \sum_{c_1 \in C} \sum_{P_1 \in \mathcal{P}_{c_1}} \sum_{c_2 \in C} \sum_{P_2 \in \mathcal{P}_{c_2}} g_{c_1,P_1,c_2,P_2}\ x_{c_1,P_1}\ x_{c_2,P_2}. \tag{13}$$

The textbook approaches to linearize such an objective function, with the introduction of additional variables y_{c_1,P_1,c_2,P_2} to model the product $x_{c_1,P_1} \cdot x_{c_2,P_2}$, appear to be fairly unsuccessful in this context, given that even the $x_{c,P}$ variables are so many to have to be handled with care. On the other hand, taking into account constraints (2), rather than these product variables one may simply introduce variables z_{c_1,c_2} to model the whole term $\sum_{P_1 \in \mathcal{P}_{c_1}} \sum_{P_2 \in \mathcal{P}_{c_2}} g_{c_1,P_1,c_2,P_2}\ x_{c_1,P_1}\ x_{c_2,P_2}$, knowing in advance that exactly one of the $x_{c_1,P_1} \cdot x_{c_2,P_2}$ products in the summation will take the value 1 in the optimal integer solution. The number of these variables is only quadratic in the number of commodities, so they can generally be all inserted explicitly in the current LP relaxation. The linearized objective function reads:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_c} f_{c,P}\ x_{c,P} + \sum_{c_1 \in C} \sum_{c_2 \in C} z_{c_1,c_2}, \tag{14}$$

and the issue is to link the z_{c_1,c_2} and the $x_{c,P}$ variables appropriately. While it is not clear how to do this in general when column generation is required for the $x_{c,P}$ variables, and we are not aware of successful applications of this, the task is fairly easy when column generation is not necessary, as illustrated next.

For a given c_1, c_2 pair, consider the polytope defined by the convex hull $\mathcal{H}$ of the following vectors with $1 + |\mathcal{P}_{c_1}| + |\mathcal{P}_{c_2}|$ components. The first component is associated with z_{c_1,c_2}. The subsequent $|\mathcal{P}_{c_1}|$ components are associated with x_{c_1,P_1}, $P_1 \in \mathcal{P}_{c_1}$. The last $|\mathcal{P}_{c_2}|$ components are associated with x_{c_2,P_2}, $P_2 \in \mathcal{P}_{c_2}$. There are $|\mathcal{P}_{c_1}| \cdot |\mathcal{P}_{c_2}|$ vectors defining the convex hull $\mathcal{H}$, namely those associated with a unique component $x_{c_1,P_1} = 1$, a unique component $x_{c_2,P_2} = 1$, and the first component given by $z_{c_1,c_2} = g_{c_1,P_1,c_2,P_2}$. These vectors correspond to all possible values of the associated variables in the solution. Accordingly, as link inequalities one may impose all those valid (and facet-defining) for the convex hull $\mathcal{H}$, having the generic form:

$$\sum_{P_1 \in \mathcal{P}_{c_1}} \alpha_{c_1,P_1}\ x_{c_1,P_1} + \sum_{P_2 \in \mathcal{P}_{c_2}} \alpha_{c_2,P_2}\ x_{c_2,P_2} + \beta_{c_1,c_2}\ z_{c_1,c_2} \leq \gamma. \tag{15}$$

Even restricting attention to the facet-defining ones, not only the number of these constraints is exponential, but also their structure is unknown (to the best of our knowledge). On the other hand, these constraints may be separated in time polynomial in $|\mathcal{P}_{c_1}|\cdot|\mathcal{P}_{c_2}|$ by explicitly setting up a separation LP to test if the current LP solution is a convex combination of the $|\mathcal{P}_{c_1}|\cdot|\mathcal{P}_{c_2}|$ vectors defining the convex hull. If this is not the case, the dual of this separation LP yields a violated inequality (15). The reader is referred to [15] for details.

5 Conclusions

In this work we have briefly illustrated the mathematical formulations and the solution approaches that we found most successful for a few real-world optimization problems arising in railway planning. In a forthcoming full version of this work, we plan to provide further details and some theoretical foundations and justifications of the various qualitative and vague notions of "good", "bad", "strong" and "weak" given here, that so far were only motivated by computational evidence.

Acknowledgment

This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

References

1 E. Balas and M.C. Carrera, "A Dynamic Subgradient-Based Branch-and-Bound Procedure for Set Covering", *Operations Research* 44 (1996), 875–890.

2 C. Barnhart, C.A. Hane, P.H. Vance, "Integer Multicommodity Flow Problems", in W.H. Cunningham, T.S. McCormick, M. Queyranne (eds.), *Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization (IPCO'96)*, Lecture Notes in Computer Science 1084, Springer-Verlag (1996) 58–71.

3 C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance, "Branch-and-Price: Column Generation for Solving Huge Integer Programs", *Operations Research* 46 (1998) 316–329.

4 J.E. Beasley, "A Lagrangian Heuristic for Set Covering Problems", *Naval Research Logistics* 37 (1990), 151–164.

5 U. Brännlund, P.O. Lindberg, A. Nõu and J.E. Nilsson, "Railway Timetabling Using Lagrangian Relaxation", *Transportation Science* 32 (1998), 358–369.

6 M. Bussieck, T. Winter, U. Zimmermann, "Discrete Optimization in Public Rail Transport", *Mathematical Programming* 79 (1997), 415–444.

7 V. Cacchiani, A. Caprara, P. Toth, "A Column Generation Approach to Train Timetabling on a Corridor", *4OR* 6 (2008), 125–142.

8 V. Cacchiani, A. Caprara, P. Toth, "Solving a Real-World Train Unit Assignment Problem", *Mathematical Programming* 124 (2010), 207–232.

9 V. Cacchiani, A. Caprara, P. Toth, "Non-cyclic Train Timetabling and Comparability Graphs", *Operations Research Letters* 38 (2010), 179–184.

10 A. Caprara, M. Fischetti, P. Toth, "A Heuristic Method for the Set Covering Problem", *Operations Research* 47 (1999), 730–743.

11 A. Caprara, M. Fischetti, P. Toth, "Algorithms for the Set Covering Problem", *Annals of Operations Research* 98 (2000), 353–371.

12 A. Caprara, M. Fischetti, P. Toth, D. Vigo, "Modeling and Solving the Crew Rostering Problem", *Operations Research* 46 (1998), 820–830.

13 A. Caprara, M. Fischetti, P. Toth, D. Vigo, P. Guida, "Algorithms for Railway Crew Management", *Mathematical Programming* 79 (1997), 125–141.

14 A. Caprara, M. Fischetti, P. Toth, "Modeling and Solving the Train Timetabling Problem", *Operations Research* 50 (2002), 851–861.

15 A. Caprara, L. Galli, P. Toth, "Solution of the Train Platforming Problem", C. Liebchen, R.K. Ahuja, J.A. Mesa (eds.) *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*, IBFI, Schloss Dagstuhl, Germany (2007).

16 A. Caprara, L. Kroon, M. Monaci, M. Peeters, P. Toth, "Passenger Railway Optimization", in: C. Barnhart, G. Laporte (eds.), *Transportation*, Handbooks in Operations Research and Management Science 14, Elsevier (2007), 129–187.

17 S. Ceria, P. Nobili and A. Sassano, "A Lagrangian-Based Heuristic for Large-Scale Set Covering Problems", *Mathematical Programming* 81 (1998), 215–228.

18 J. Desrosiers, Y. Dumas, M.M. Solomon and F. Soumis, "Time Constrained Routing and Scheduling", in M.O. Ball et al. (Eds.), *Handbooks in OR & MS*, Vol. 8, Elsevier Science, 1995, 35–139.

19 M.L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science* 27 (1981), 1–18.

20 G. Gruber F. Rendl, "The Bundle Method for Hard Combinatorial Optimization Problems", in M. Jünger, G. Reinelt, G. Rinaldi (eds.), *Combinatorial Optimization – Eureka, You Shrink!*, Springer-Verlag (2003), 78–88.

21 D. Huisman, L. Kroon, R. Lentink, M. Vromans, "Operations Research in Passenger Railway Transportation", *Statistica Neerlandica* 59 (2005), 467–497.

22 G.L. Nemhauser, L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley (1999).

23 H.D. Sherali, D.C. Myers, "Dual Formulations and Subgradient Optimization Strategies for Linear Programming Relaxations of Mixed-Integer Programs", *Discrete Applied Mathematics* 20 (1988), 51–68

24 P. Zwaneveld, L. Kroon, C. van Hoesel. "Routing Trains Through a Railway Station Based on a Node Packing Model", *European Journal of Operational Research* 128 (2001), 14–33.