OPERATIONS RESEARCH
Vol. 53, No. 4, July–August 2005, pp. 632–645
ISSN 0030-364X | EISSN 1526-5463 | 05 | 5304 | 0632

informs®

DOI 10.1287/opre.1050.0222
© 2005 INFORMS

Dynamic Aggregation of Set-Partitioning Constraints in Column Generation

Issmail Elhallaoui
Mathematics and Industrial Engineering Department, École Polytechnique de Montréal and GERAD,
Montréal, Québec, Canada H3C 3A7, issmail.elhallaoui@gerad.ca

Daniel Villeneuve
Kronos Inc., 3535 Queen Mary, Suite 650, Montréal, Québec, Canada H3V 1H8, dvilleneuve@kronos.com

François Soumis, Guy Desaulniers
Mathematics and Industrial Engineering Department, École Polytechnique de Montréal and GERAD,
Montréal, Québec, Canada H3C 3A7 {francois.soumis@gerad.ca, guy.desaulniers@gerad.ca}

Column generation is often used to solve problems involving set-partitioning constraints, such as vehicle-routing and crew-scheduling problems. When these constraints are in large numbers and the columns have on average more than 8–12 nonzero elements, column generation often becomes inefficient because solving the master problem requires very long solution times at each iteration due to high degeneracy. To overcome this difficulty, we introduce a dynamic constraint aggregation method that reduces the number of set-partitioning constraints in the master problem by aggregating some of them according to an equivalence relation. To guarantee optimality, this equivalence relation is updated dynamically throughout the solution process. Tests on the linear relaxation of the simultaneous vehicle and crew-scheduling problem in urban mass transit show that this method significantly reduces the size of the master problem, degeneracy, and solution times, especially for larger problems. In fact, for an instance involving 1,600 set-partitioning constraints, the master problem solution time is reduced by a factor of 8.

Subject classifications: programming: linear; large-scale systems; column generation; degeneracy; dynamic constraint aggregation.
Area of review: Transportation.
History: Received July 2003; revisions received January 2004, April 2004; accepted June 2004.

1. Introduction

Column generation is an iterative process that solves, at each iteration, a restricted master problem (a linear program) and one or several subproblems. It is well known for solving linear relaxations in a variety of applications, especially in the vehicle-routing and crew-scheduling fields, where many problems can be formulated as set-partitioning-type integer problems. In practice, large instances of these problems involve more than one billion variables and one thousand constraints, including more than 90% of set-partitioning constraints. Such a large number of set-partitioning constraints and the presence of columns having on average more than 8–12 nonzero elements usually yield high degeneracy in the restricted master problems, considerably slowing down the column generation process. In fact, when the simplex algorithm is used for solving these linear programs, the basis contains a large percentage of degenerate variables and the algorithm executes many degenerate pivots.

In this paper, we propose a new exact column generation algorithm called the *dynamic constraint aggregation algorithm* that works with a sequence of linear programs containing significantly fewer constraints. These linear programs are much easier to solve than the standard restricted master problems, yielding a substantial reduction in the overall linear relaxation solution times.

Without loss of generality, we use the following terminology derived from crew-scheduling applications. A set-partitioning constraint is associated with a *task* (for instance, a flight or a bus trip segment) to be accomplished by a *commodity* (a pilot or a bus driver). The main variables are associated with *paths* (pilot or driver schedules) that are feasible with regards to a set of predefined rules. A path contains an ordered sequence of tasks and possibly other activities (breaks, rests, briefings, etc.). For the sake of clarity, we assume first that there exists a partial ordering of the tasks that is sufficient to order completely the tasks that can be part of the same feasible path. For instance, such an ordering might correspond to the chronological order of the task starting times. This assumption is lifted at the end of §4.2.2.

The motivation behind this work comes from two observations. First, in crew-scheduling problems, we observe that crews do not change their vehicles very often; their rotations thus have many parts in common with the vehicle routes. Second, for reoptimization of crew or vehicle

schedules during their operation, we notice that reoptimized solutions deviate slightly from planned ones. Consequently, many consecutive tasks on the initial paths (the vehicle or planned solution paths in the examples above) will remain grouped in an optimal solution. These consecutive tasks can then be seen as a single *aggregated task*, in which case their corresponding set-partitioning constraints can be replaced by a single set-partitioning constraint. Starting with an initial aggregation of the tasks, the dynamic constraint aggregation method relies on an *aggregated* master problem that contains a set-partitioning constraint for each aggregated task. Because the size of this master problem is reduced, it is easier to solve than the nonaggregated one. However, the task aggregation needs to be adjusted dynamically throughout the solution process because we do not know a priori which tasks will be consecutive in the optimal paths. Consequently, to prove the optimality of an aggregated master problem solution, columns that do not respect the current task aggregation also need to be priced out, requiring a dual solution for the nonaggregated master problem. The dynamic constraint aggregation algorithm uses an efficient shortest path algorithm to obtain the dual solution of the nonaggregated master problem.

The idea of dynamic constraint aggregation was first introduced in the thesis of Villeneuve (1999). He presented a theoretical framework of the dynamic constraint aggregation algorithm, but did not provide an implementation. This paper presents the first implementation of this approach and refines the theoretical framework based on the insight gained from the practical point of view. The results at the end of this paper show that, in some test cases, the total linear relaxation solution time is reduced by up to 80% compared to the standard column generation method.

This paper is organized as follows. The next section presents a generic but simplified mathematical formulation for the class of problems addressed by the dynamic constraint aggregation approach. It also briefly describes the standard column generation approach. Section 3 proposes a literature review on different ways of accelerating column generation. In §4, the dynamic constraint aggregation algorithm is presented. We discuss its basic concepts and analyze its convergence. In §5, we report computational results obtained with this approach on randomly generated instances of the simultaneous vehicle and crew-scheduling problem. Conclusions and future research directions are presented in §6.

2. Generic Formulation and Column Generation

The dynamic constraint aggregation approach addresses the linear relaxation of problems having a set of tasks to be covered exactly once with feasible paths at minimum cost. These problems can be modeled as set-partitioning-type problems, which can often be solved using a column generation method (Dantzig and Wolfe 1960, Gilmore and Gomory 1961) embedded in a branch-and-bound scheme. This decomposition method divides the problem into a master problem and a set of subproblems. The constraints linking the various commodities (such as task-covering constraints) are handled in the master problem, while the constraints separable by commodity (such as the total duration of a crew schedule) are treated in the subproblems.

We present below a generic formulation of a master problem. To lighten the text, this formulation is simplified and contains only the essential elements for understanding the dynamic constraint aggregation algorithm, that is, the set-partitioning linking constraints. The following notation is required.

Let W and K be the sets of tasks and commodities, respectively. The set of feasible paths for commodity $k \in K$ is denoted by P^k. With each path $p \in P^k$, $k \in K$, we associate a cost c_p^k and a parameter a_{wp}^k that takes value 1 if it covers task $w \in W$ and 0 otherwise. Without loss of generality, we assume that for each path p, at least one of its associated parameter a_{wp}^k takes value 1. The model relies on two types of variables: path flow variables θ_p^k that indicate the flow of commodity $k \in K$ along path $p \in P^k$ and artificial variables Y_w, $w \in W$, that guarantee problem feasibility. These variables are given a cost M large enough to ensure that their value is zero if there exists a solution involving only path variables. They are introduced in this model to simplify the proofs of §4.3. Note that, for real-world applications, each artificial variable could bear the real cost of executing the associated task by external means and would therefore be fully part of the model.

The generic master problem formulation, denoted by MP, is as follows:

$$\text{Minimize } \sum_{k\in K}\sum_{p\in P^k} c_p^k \theta_p^k + M \sum_{w\in W} Y_w \tag{2.1}$$

$$\text{subject to } \sum_{k\in K}\sum_{p\in P^k} a_{wp}^k \theta_p^k + Y_w = 1 \quad \forall\, w \in W, \tag{2.2}$$

$$\theta_p^k \geqslant 0 \quad \forall\, p \in P^k,\ k \in K, \tag{2.3}$$

$$Y_w \geqslant 0 \quad \forall\, w \in W. \tag{2.4}$$

Model (2.1)–(2.4) is a set-partitioning model. The objective function (2.1) seeks to minimize the total cost, which includes the penalties for violating the set-partitioning constraints (2.2). These constraints ensure that each task is covered exactly once. Constraint sets (2.3) and (2.4) impose nonnegativity on the θ_p^k and Y_w variables. It should be noted that the MP is feasible and bounded. A less-simplified model would contain additional linear constraints (for instance, constraints on vehicle availability), possibly involving additional variables, to reflect the specificity of the problem.

In practice, this formulation often contains a huge number of θ_p^k path variables and is solved using an iterative column generation method. At each iteration, such a method considers a restricted master problem (RMP) that involves

a small subset of the path variables. The RMP is usually solved by means of a standard linear programming algorithm, such as the primal simplex algorithm, to obtain a pair of primal and dual solutions. Then, taking into account this dual solution, a series of subproblems (one per commodity) are solved to generate negative reduced-cost path variables (columns), if any. If none are generated, the algorithm stops, as the computed primal solution of the current RMP is also optimal for MP. Otherwise, the negative reduced-cost columns (or a subset of them) are added to the RMP, which is reoptimized at the next iteration.

In general, for time-constrained vehicle-routing and crew-scheduling problems, the subproblems correspond to constrained shortest path problems (see Desrosiers et al. 1984, Desrochers and Soumis 1989, Barnhart et al. 1998, Desaulniers et al. 1998, Gamache et al. 1999). We assume that an oracle is available for efficiently solving them.

3. Literature Review

The overall solution time of a column generation method is equal to the sum of the time spent at each iteration. Two strategies can be used to reduce it: (i) use less time per iteration without increasing the number of iterations too much, or (ii) reduce the number of iterations without increasing the time spent at each iteration too much. We next review the main approaches proposed in the literature for both strategies.

To reduce the computational cost per iteration, partial pricing can often be used when a large proportion of the time is devoted to the solution of the subproblems. This acceleration technique consists of solving restricted subproblems or a subset of the subproblems at each iteration (see Gamache et al. 1999). However, the number of iterations might increase because the information used by the subproblems is poorer. Also, when the master problem contains a large number of constraints that have a high probability of being inactive at optimality, these constraints can be relaxed a priori and individually reintroduced at subsequent column generation iterations when the current solution violates them. This constraint generation approach was, for instance, successfully applied by Cordeau et al. (2001) for simultaneously assigning locomotives and cars to passenger trains. Constraint generation usually increases the number of iterations because the cascading effect of reintroducing a constraint on the other still relaxed constraints might be discovered only at future iterations. This approach can therefore be beneficial when a large number of constraints are set aside at the beginning and most of them are not reintroduced.

Freling et al. (1999) proposed to approximately solve each RMP using Lagrangean relaxation (Geoffrion 1974, Fisher 1981). In this approach, constraints are transferred in the objective function using multipliers, forming Lagrangean subproblems that provide lower bounds on the MP optimal value. Then, the problem of finding the multiplier values yielding the largest lower bound, called the Lagrangean dual problem, is solved by an algorithm for nonsmooth convex optimization. When using an algorithm whose convergence is asymptotic, such as the subgradient algorithm (Held and Karp 1971), the algorithm is stopped according to heuristic criteria. If the approximate (feasible but not necessarily optimal) dual solution to the RMP is sufficiently close to the set of optimal solutions, the number of column generation iterations needed to solve the MP stays more or less the same as with the column generation algorithm described in §2. However, the Lagrangean relaxation approach does not directly compute a primal solution for the RMP. Primal solutions are useful both for finding heuristic integer feasible solutions and for defining branching rules when column generation is used within a branch-and-price framework. To compensate for this shortcoming, Barahona and Anbil (2000) developed the so-called volume algorithm, an extension of the subgradient algorithm that also computes an approximate primal solution.

Lagrangean relaxation can also be applied for solving the whole MP, that is, not only the RMP at each column generation iteration. In this case, it is well known (see Nemhauser and Wolsey 1988) that it produces the same optimal value as column generation when these two methods rely on the same subproblems. Lagrangean relaxation is then a dual formulation of the column generation problem, where the set of all columns in the latter is replaced implicitly in the former by the use of optimization subproblems. This dual formulation has historically been associated with simple solution procedures at each iteration, such as a single step in the direction of the last subgradient produced by a subproblem, with asymptotic convergence as a consequence.

Solving the MP via a sequence of RMPs is a dual upper bounding algorithmic strategy first described by Kelley (1960) and having weak theoretical convergence properties. The major drawback of Kelley's method stems from the lack of significant guarantees on the amount of dual space being cut by the adjunction of new generated columns. This fact has motivated the development of central cutting-plane methods with better theoretical convergence properties, that is, yielding in theory a reduced number of iterations. Goffin and Vial (1990, 2002) describe an analytic center cutting-plane method (ACCPM) in which a logarithmic barrier function is maximized over a dual localization set to identify the next dual point to be used for solving the Lagrangean subproblems. They provide a convergence analysis and many references to problems solved either more efficiently or more reliably using ACCPM than Kelley's method.

Pursuing the same goal of accelerating the theoretical and practical convergence of the sequence of dual solutions toward a solution of the MP, bundle methods (see Hiriart-Urruty and Lemaréchal 1991) add to the linearized (column generated) model of the Lagrangean subproblems a parameterized quadratic penalty centered at the current dual iterate to prevent oscillations in the dual space. By

taking advantage of a priori information on optimal dual solutions to the MP, du Merle et al. (1999) could obtain important reductions in the overall solution time by using a three-piece linearization of the bundle methods idea.

The contribution of this paper is algorithmic and practical. We propose an original solution strategy for solving a special class of MPs by column generation, in which there is a large proportion of set-partitioning constraints. The solution strategy is a specialization of the usual column generation algorithm, where we exploit some (partial) knowledge of the structure of primal solutions to reduce the number of constraints to be considered simultaneously when solving an RMP. The originality of the approach lies in the definition of an auxiliary network flow problem used both to drive the selection of the subset of considered constraints and to efficiently produce values for the dual variables associated with unselected constraints. We then demonstrate the practical efficiency of our approach by solving large instances of a vehicle and crew-scheduling problem.

4. Dynamic Constraint Aggregation Algorithm

This section presents the proposed dynamic constraint aggregation method and analyzes its convergence. Beforehand, we discuss the basic concepts of this new methodology.

4.1. Basic Concepts

The dynamic constraint aggregation approach deals with reducing the number of set-partitioning constraints (2.2) in the MP by aggregating some of them. This aggregation is performed according to a partition of the tasks in W that is defined by the following equivalence relation:

Given a set of paths C, two tasks w_1 and w_2 in W are said to be equivalent with respect to C if every path in C covers both w_1 and w_2, or none of them.

This relation, which is denoted by $\mathcal{R}^C$, partitions the tasks into *equivalence classes*. Let L be the set of classes, W_l the subset of tasks in class $l \in L$, and $Q = \{W_l \colon l \in L\}$ the resulting task partition, where any reference to C is omitted to lighten the notation. At the beginning, partition Q is defined with respect to the initial set of paths C that is provided by an external heuristic or a planned solution. Then, throughout the solution process, Q is adjusted dynamically by modifying the composition of C until an optimal solution is found.

Now let us define some compatibility criteria between partition Q and the θ_p^k path variables. Let p be a path in P^k, $k \in K$, and T_p be the set of tasks it covers. Path p is said to be *compatible* with the equivalence class $l \in L$ if $W_l \cap T_p$ is empty or equal to W_l. It is also said to be *compatible* with partition Q if it is compatible with all equivalence classes in L. In this case, we also say that θ_p^k or its corresponding column is *compatible* with Q.

Instead of using the traditional RMP, the dynamic constraint aggregation algorithm relies on a so-called aggregated restricted master problem (ARMP) that considers smaller subsets of variables and constraints than the RMP. The partition Q plays a central role in defining the ARMP. First, it restricts the set of columns that can be added to the ARMP to the columns that are compatible with Q. Second, Q also restricts the set of partitioning constraints (2.2) considered in the ARMP to one *representative partitioning constraint* for each equivalence class $l \in L$. The task in W_l associated with that constraint is denoted by w_l and said to be the *representative task* for this class. The other set-partitioning constraints are removed from the ARMP as they are identical to their representative constraint with respect to the path variables considered by the ARMP. They might, however, reappear in the ARMP when Q is updated. Besides the columns compatible with Q, the ARMP also contains the artificial variables Y_{w_l}, $l \in L$, involved in the representative set-partitioning constraints. The cost of such a variable is, however, set to $M|W_l|$ to ensure a uniform cost structure when different partitions are used. We say that the ARMP is restricted to partition Q and, to be more explicit when necessary, we will denote it by ARMP_Q.

The dynamic constraint aggregation algorithm is made up of two types of iterations: minor and major ones. In a minor iteration, the ARMP_Q is optimized and columns are generated by the subproblems, whereas in a major iteration a series of minor iterations are executed before adjusting the partition Q. To generate columns, the subproblems require a complete dual solution, that is, a dual variable value for each set-partitioning constraint (2.2) in the MP. Therefore, a procedure is needed to disaggregate the dual solution of the ARMP_Q. Finally, let us mention that the subproblems can generate columns that are compatible or incompatible with Q. The incompatible columns are obviously not added to the ARMP_Q, but they are used to decide when and how to modify the task partition.

4.2. Algorithm Description

The dynamic constraint aggregation algorithm constitutes a new version of the standard column generation method in which some constraints of the RMP are aggregated. This new version speeds up the solution process. It can also be easily adapted to adequately exploit specific structures of a given problem.

The algorithm pseudocode is given in Algorithm 1 and commented upon in the subsequent paragraphs. Comprehensive details are further provided in §§4.2.1 and 4.2.2. We begin by giving some useful notation:

- Q denotes a partition of the tasks.
- For a given set P of feasible columns, $P_Q \subset P$ denotes the subset of columns compatible with partition Q.
- P' denotes the set of columns generated by the oracle since the beginning of the algorithm.

• oracle(α) is a function that solves the subproblems and returns negative reduced cost columns, if any, according to a disaggregated dual variable vector α.

• P'' denotes the set of columns generated by the oracle at a given minor iteration.

• modify(α) is a predicate that, based on a disaggregated dual variable vector α, decides whether or not it is better to switch to another partition.

• Z_{old} stores the objective value at the end of each major iteration and is initially set to ∞.

• B denotes the set of basic columns taking a positive value at a given minor iteration.

In Step 1 of Algorithm 1, an initial set of columns C is chosen. This set may be found by a heuristic procedure such as the simple one presented in §5.3. It is used to define an initial partition according to the equivalence relation $\mathcal{R}^C$. On the one hand, we shall highlight the importance of accurately choosing the initial partition to maximize the effectiveness of the linear programming optimizer and to minimize the number of column generation iterations. This initial partition should group together, as much as possible, tasks having a high probability of being grouped together in an optimal solution of the linear relaxation. The design of this partition is therefore problem dependent. On the other hand, it is not essential that these initial columns form a feasible solution to the MP, and even that they be associated with valid paths, because they are only used to partition the tasks. The set of path variables considered for the first ARMP_Q is then initialized to the empty set in Step 2. Obviously, if some of the columns in C are valid, they can be added to P'_Q.

Algorithm 1. Dynamic Constraint Aggregation Algorithm

1. Choose an initial set of columns C and create an initial partition Q from C.
2. $P'_Q \leftarrow \varnothing$
3. $Z_{\text{old}} = \infty$
4. **while** True **do**
5. **repeat**
6. Solve the ARMP_Q restricted to P'_Q to obtain a primal solution x of value Z and a dual solution $\hat{\alpha}$.
7. Compute a vector of disaggregated dual variables α from $\hat{\alpha}$.
8. $P'' \leftarrow$ oracle(α)
9. **if** $P'' = \varnothing$ **then**
10. STOP {x is an optimal solution}
11. **end if**
12. $P' \leftarrow P' \cup P''$
13. **until** $P''_Q = \varnothing$ or modify(α)
14. Let $I \subseteq P'$ be a nonempty set of negative reduced cost columns incompatible with Q.
15. **if** $Z = Z_{\text{old}}$ or $\exists l \in L$ such that $Y_{w_l} > 0$ **then**
16. $C \leftarrow C \cup I$
17. **else**
18. $C \leftarrow B \cup I$
19. **end if**
20. Redefine Q and the ARMP_Q according to C.
21. $Z_{\text{old}} \leftarrow Z$
22. **end while**

Major iterations begin at Step 4 and end at Step 22, while minor ones go from Step 5 to Step 13. In Step 6, the ARMP_Q is solved using a linear programming optimizer to yield a primal solution x and a vector of dual variables $\hat{\alpha} = \{\hat{\alpha}_l \mid l \in L\}$ associated with the aggregated constraints.

In Step 7, this vector of dual variables is disaggregated to obtain one dual variable for each set-partitioning constraint (2.2) of the original problem. To do so, the following linear system, which possesses an infinite number of feasible solutions when at least two constraints are aggregated, needs to be solved:

$$\sum_{w \in W_l} \alpha_w = \hat{\alpha}_l \quad \forall\, l \in L. \tag{4.1}$$

An efficient strategy for disaggregating dual variable vectors is discussed in §4.2.2.

In Step 8, the oracle returns a set of feasible columns with negative reduced costs. If this set is empty, then the fractional solution x is optimal and the algorithm is stopped. Otherwise, this set is added to the set P' in Step 12. In Step 13, the algorithm decides if it is desirable to change the partition by completing a major iteration or to keep it unchanged and perform another minor iteration. The partition is changed if none of the columns generated in Step 8 are compatible with Q or if the predicate modify(α) decides so. Such a predicate can be very simple, as shown by the example given in §4.2.1.

In Steps 14–20, partition Q is redefined according to a new set of columns C. This new set is formed either by adjoining a set of incompatible columns to the previous set C or by merging this set of incompatible columns with the set B of positive-valued columns in the current basis. The first alternative necessarily increases the number of equivalence classes in the partition, while the second one aims at reducing it.

4.2.1. Partition Handling. After every minor iteration, the algorithm must decide at Step 13 if partition Q requires an update. Obviously, when no columns compatible with Q have been generated in the current iteration (that is, P''_Q is empty), Q is updated. Otherwise, the predicate modify(α) decides whether or not such an update is deemed profitable. An example for such a predicate is as follows:

$$\texttt{modify}(\alpha) = \begin{cases} \text{true} & \text{if } \{p \in \bar{P}'_Q\colon \bar{c}_p(\alpha) < 0\} \neq \varnothing \text{ and} \\ & \quad r = \dfrac{\min_{p \in P''_Q} \bar{c}_p(\alpha)}{\min_{p \in \bar{P}'_Q} \bar{c}_p(\alpha)} < \lambda, \\ \text{false} & \text{otherwise,} \end{cases}$$

where $\bar{P}'_Q = P' \backslash P'_Q$ and $\bar{c}_p(\alpha)$ denotes the reduced cost of path $p \in P^k$, $k \in K$, with respect to the disaggregated dual-variable vector α, and λ is a nonnegative parameter. This predicate returns a *true* value (which triggers a partition update) whenever there exists at least one incompatible path with a negative reduced cost and the ratio r is less than λ. When λ is given a very large value (∞), the algorithm completes a major iteration as soon as there exists one incompatible column with a negative reduced cost. The other extreme is to fix λ at 0. This case dictates executing a minor iteration every time there is a compatible column with a negative reduced cost. Neither of these extremes is a good value for this parameter. In fact, a value between 0 and 1 seems to be a more appropriate choice because modifying the partition can substantially change the structure of the dual domain of the ARMP, yielding an increase in its solution time at the next minor iteration. Note that this strategy is similar to the partial pricing mechanism used in the simplex algorithm.

Two ways are used to update partition Q. On the one hand, when the last partition did not succeed in improving the objective function value $Z = Z_{\text{old}}$, or when the current ARMP solution involves positive-valued artificial variables, the set C is augmented in Step 16 by adding to it all columns in I. This refines the partition by breaking some of its elements to make them compatible with the columns in I. To avoid increasing the partition size too much, I is restricted to a maximum of γ columns, favoring the columns with the most negative reduced costs. On the other hand, when partition Q was successful in decreasing the objective function value, the partition is aggregated to prevent its explosion into singletons. This is done by redefining the set C in Step 18 as a union of I and the nondegenerate basic columns of B. For practical purposes, we also impose another condition for aggregating the partition: $|Q|/|W|$ must exceed a ratio δ. This condition ensures that the partition is not too aggregated.

Finally, note that the columns incompatible with partition Q at a given iteration are removed temporarily from the ARMP and kept in memory for future use, namely, in the dual variables disaggregation strategy discussed in the next section.

4.2.2. Dual Variables Disaggregation Strategy. The optimal primal solution x, of value Z, computed for the ARMP in Step 6 is also optimal for the MP if there exists a dual solution α for the MP that is dual feasible and whose cost is Z. One difficulty with the dynamic constraint aggregation method is that it does not provide a complete dual solution when solving the ARMP. It rather provides an aggregated dual solution $\hat{\alpha}$ that needs to be disaggregated.

To do so, one can find a feasible solution α to the linear system (4.1). This solution has a cost of Z and guarantees that the reduced costs of the basic variables of the current ARMP remain zero. Note that setting $\alpha_w = \hat{\alpha}_l/|W_l|$ $\forall w \in W_l$, $l \in L$, defines a feasible solution to this system. However, preliminary tests have shown that it is more efficient to find a solution that also satisfies a large number of dual constraints

$$\sum_{w \in W} a_{wp} \alpha_w \leqslant c_p \quad \forall p \in \bar{P}^*_Q \subseteq \bar{P}'_Q, \tag{4.2}$$

where $\bar{P}^*_Q$ denotes the subset of generated incompatible columns for which we want these constraints to be satisfied, and the commodity index k was ignored for notational simplification. In fact, these constraints must all be satisfied at optimality. Consequently, instead of only looking for a feasible solution to system (4.1), it is preferable to search for a feasible solution to the linear system of equalities and inequalities composed of (4.1) and (4.2).

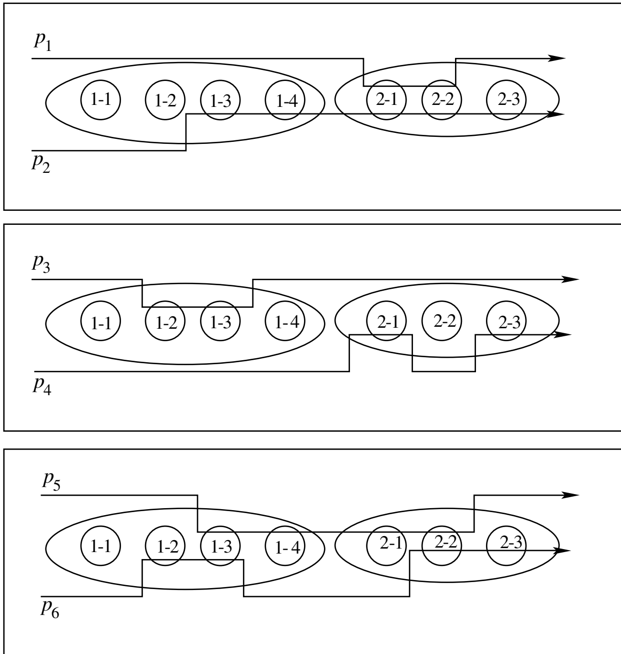
This enlarged linear system represents the intersection between a hyperplane and a polyhedron. Solving this system can be as difficult as solving the original MP itself. Because the main goal of the dynamic constraint aggregation algorithm is to answer precisely the lack of efficient methods for solving it, we propose an alternative that consists of restricting the set $\bar{P}^*_Q$ to some interesting incompatible columns (as explained below) so that this linear system can be transformed into a shortest-path problem that is easier to solve.

The rest of this section shows how to derive this shortest-path problem. First, the set of incompatible columns is classified into six categories to identify the interesting incompatible columns. Then, a variable substitution is proposed to transform the linear system (4.1) and (4.2) into another linear system whose inequalities correspond to the optimality conditions of the sought shortest-path problem. These conditions allow the definition of the nodes and arcs of this problem.

Given the assumption on the partial ordering of the tasks, the tasks within each equivalence class of Q can be ordered. Thus, we refer to a task $w_j \in W$ as w_l^h if $w_j \in W_l$ and h is its order number in this set. Given a partition Q, incompatible columns are grouped into six categories. Figure 1 illustrates an example for each category. In this figure, which is divided into three parts to enhance visibility, seven tasks (numbered from 1-1 to 1-4 and from 2-1 to 2-3) are partitioned into two equivalence classes according to partition $Q = \{W_1, W_2\}$, where $W_1 = \{\text{1-1}, \text{1-2}, \text{1-3}, \text{1-4}\}$ and $W_2 = \{\text{2-1}, \text{2-2}, \text{2-3}\}$. Incompatible columns (numbered from p_1 to p_6) are illustrated by polygonal lines that pass through the tasks they cover. The six incompatibility categories are defined as follows.

- A column p is said to be *S-incompatible* with Q if it is incompatible with only one class i of L and covers the starting (S) tasks of this class, i.e., if there exist $i \in L$ and a positive integer $m < |W_i|$ such that:
  1. p is compatible with $l \in L \backslash \{i\}$, and
  2. $W_i \cap T_p = \{w_i^1, \ldots, w_i^m\}$.
- A column p is said to be *E-incompatible* with Q if it is incompatible with only one class i of L and covers the ending (E) tasks of this class, i.e., if there exist $i \in L$ and

Figure 1. Column incompatibilities with a given partition.



Legend. p_1: S-incompatible, p_2: E-incompatible, p_3: M-incompatible, p_4: SE-incompatible, p_5: ES-incompatible, p_6: O-incompatible.

a positive integer $m > 1$ such that:

1. p is compatible with $l \in L \backslash \{i\}$, and
2. $W_i \cap T_p = \{w_i^m, \ldots, w_i^{|W_i|}\}$.

• A column p is said to be *M-incompatible* with Q if it is incompatible with only one class i of L and covers the middle (M) tasks of this class, i.e., if there exist $i \in L$ and positive integers m and n with $1 < m \leqslant n < |W_i|$, such that:

1. p is compatible with $l \in L \backslash \{i\}$, and
2. $W_i \cap T_p = \{w_i^m, \ldots, w_i^n\}$.

• A column p is said to be *SE-incompatible* with Q if it is incompatible with only one class i of L and covers the starting (S) and the ending (E) tasks of this class, i.e., if there exist $i \in L$ and positive integers m and n with $n \geqslant 2$ and $m + n \leqslant |W_i|$, such that:

1. p is compatible with $l \in L \backslash \{i\}$, and
2. $W_i \cap T_p = \{w_i^1, \ldots, w_i^m, w_i^{m+n}, \ldots, w_i^{|W_i|}\}$.

• A column p is said to be *ES-incompatible* with Q if it is incompatible with two classes i and j of L and covers the ending (E) tasks of class i followed by the starting (S) tasks of class j, i.e., if there exist $i \in L$, $j \in L$ with $i \neq j$, and positive integers $m > 1$ and $n < |W_j|$, such that:

1. p is compatible with $l \in L \backslash \{i, j\}$,
2. $W_i \cap T_p = \{w_i^m, \ldots, w_i^{|W_i|}\}$, and
3. $W_j \cap T_p = \{w_j^1, \ldots, w_j^n\}$.

• Any other incompatible column with Q is said to be *O-incompatible*.

The subset $\bar{P}_Q^*$ of incompatible columns in (4.2) is restricted to the columns that fall within the first five categories. The O-incompatible columns are not considered because they would destroy the structure of the shortest-path problem described below. Note, however, that these columns, which represent less than 18% of the incompatible columns in the tests we made, might have the opportunity to be considered in subsequent iterations when the partition Q is readjusted.

Given a set $\bar{P}_Q^*$ of generated incompatible columns, the linear system (4.1)–(4.2) is transformed using a new set of variables π_l^h, $h \in \{1, \ldots, |W_l|\}$ and $l \in L$, defined by

$$\pi_l^h = \sum_{j=1}^{h} \alpha_l^j \quad \forall\, h \in \{1, \ldots, |W_l|\},\ l \in L, \tag{4.3}$$

where α_l^j is the dual variable of the set-partitioning constraint (2.2) associated with task $w_l^j \in W$. For the example above, these variables correspond to

$$\begin{aligned} &\pi_1^1 = \alpha_1^1, && \pi_2^1 = \alpha_2^1,\\ &\pi_1^2 = \alpha_1^1 + \alpha_1^2, && \pi_2^2 = \alpha_2^1 + \alpha_2^2,\\ &\pi_1^3 = \alpha_1^1 + \alpha_1^2 + \alpha_1^3, && \pi_2^3 = \alpha_2^1 + \alpha_2^2 + \alpha_2^3.\\ &\pi_1^4 = \alpha_1^1 + \alpha_1^2 + \alpha_1^3 + \alpha_1^4, && \end{aligned}$$

Using these new variables, equalities (4.1) can be written as

$$\pi_l^{|W_l|} = \hat{\alpha}_l \quad \forall\, l \in L, \tag{4.4}$$

while inequalities (4.2), depending on the column incompatibility category, can be expressed as in Table 1, where L_p denotes the set of equivalence classes completely covered by path p and the notation (i, j, m, and n) used for each category corresponds to the one used earlier to define these categories. Using (4.4) for replacing the $\pi_i^{|W_i|}$ variables in the right-hand side of these inequalities, we obtain constant right-hand sides. Table 2 presents the resulting constraints for the example above (assuming that there are no other tasks in the problem).

The new inequalities are called *difference inequalities* because they involve a difference of two variables on the left-hand side, where one of these two variables might be missing. Note that O-incompatible columns cannot be written like difference inequalities.

As one can see, the difference inequalities correspond to the optimality conditions of a shortest-path problem

Table 1. Transformation of inequalities (4.2).

Category	Transformed constraint
p is S-incompatible	$\pi_i^m \leqslant c_p - \sum_{l \in L_p} \hat{\alpha}_l$
p is E-incompatible	$-\pi_i^{m-1} \leqslant c_p - \hat{\alpha}_i - \sum_{l \in L_p} \hat{\alpha}_l$
p is M-incompatible	$\pi_i^n - \pi_i^{m-1} \leqslant c_p - \sum_{l \in L_p} \hat{\alpha}_l$
p is SE-incompatible	$\pi_i^m - \pi_i^{m+n-1} \leqslant c_p - \hat{\alpha}_i - \sum_{l \in L_p} \hat{\alpha}_l$
p is ES-incompatible	$\pi_j^n - \pi_i^{m-1} \leqslant c_p - \hat{\alpha}_i - \sum_{l \in L_p} \hat{\alpha}_l$

Table 2. Transformed inequalities for the example.

Category	Dual constraint	Transformed constraint
p_1 is S-incompatible	$\alpha_2^1 + \alpha_2^2 \leqslant c_{p_1}$	$\pi_2^2 \leqslant c_{p_1}$
p_2 is E-incompatible	$\alpha_1^3 + \alpha_1^4 + \alpha_2^1 + \alpha_2^2 + \alpha_2^3 \leqslant c_{p_2}$	$-\pi_1^2 \leqslant c_{p_2} - \hat{\alpha}_1 - \hat{\alpha}_2$
p_3 is M-incompatible	$\alpha_1^2 + \alpha_1^3 \leqslant c_{p_3}$	$\pi_1^3 - \pi_1^1 \leqslant c_{p_3}$
p_4 is SE-incompatible	$\alpha_2^1 + \alpha_2^3 \leqslant c_{p_4}$	$\pi_2^1 - \pi_2^2 \leqslant c_{p_4} - \hat{\alpha}_2$
p_5 is ES-incompatible	$\alpha_1^3 + \alpha_1^4 + \alpha_2^1 + \alpha_2^2 \leqslant c_{p_5}$	$\pi_2^2 - \pi_1^2 \leqslant c_{p_5} - \hat{\alpha}_1$

(see Ahuja et al. 1993, pp. 135–136) defined on the directed network $G = (V, E)$, where V and E denote its node and arc sets, respectively. Node set V contains a source node S and a node N_l^j for each task w_l^j, $j \in \{1, \ldots, |W_l| - 1\}$, $l \in L$. No node is created for $j = |W_l|$ because the value of $\pi_l^{|W_l|}$ is given by (4.4). In arc set E, an arc is defined for each difference inequality as described in Table 3, where the same notation as in Table 1 is again used. The cost of such an arc is equal to the right-hand term of the inequality it represents.

Arc set E also contains artificial arcs from S to the nodes for which no path from S exists. These arcs are assigned a cost ζ, which is large enough to ensure that the artificial arcs cannot be part of a negative directed cycle. An artificial arc connecting S to a node N_l^j is equivalent to adding the difference inequality $\pi_l^j \leqslant \zeta$ to the linear system. The network built for our example is shown in Figure 2.

To compute the π variable values, the shortest-path problem from S to all the other nodes in V is solved using Bellman's (1958) dynamic programming algorithm because network G can contain cycles and arc costs can be negative. The value of π_l^j, $j \in \{1, \ldots, |W_l| - 1\}$, $l \in L$, corresponds to the length of the computed shortest path between node S and node N_l^j. When network G contains a negative cost cycle, the system of difference inequalities is infeasible because the dual problem of an unbounded primal problem is always infeasible. In this case, the detected cycle is simply broken up by removing some of its arcs (that is, by omitting the corresponding constraints (4.2)), and the shortest-path problem is solved again.

Note that once the π variables have been computed, the α variables can easily be computed as follows: $\alpha_l^1 = \pi_l^1$ and $\alpha_l^j = \pi_l^j - \pi_l^{j-1}$ $\forall j \in \{2, \ldots, |W_l|\}$, $l \in L$. Recall that the values of $\pi_l^{|W_l|}$, $l \in L$, are provided by Equation (4.4).

Table 3. Arcs created from difference inequalities.

Category	Arc	Cost
p is S-incompatible	(S, N_i^m)	$c_p - \sum_{l \in L_p} \hat{\alpha}_l$
p is E-incompatible	(N_i^{m-1}, S)	$c_p - \hat{\alpha}_i - \sum_{l \in L_p} \hat{\alpha}_l$
p is M-incompatible	(N_i^{m-1}, N_i^n)	$c_p - \sum_{l \in L_p} \hat{\alpha}_l$
p is SE-incompatible	(N_i^{m+n-1}, N_i^m)	$c_p - \hat{\alpha}_i - \sum_{l \in L_p} \hat{\alpha}_l$
p is ES-incompatible	(N_i^{m-1}, N_j^n)	$c_p - \hat{\alpha}_i - \sum_{l \in L_p} \hat{\alpha}_l$

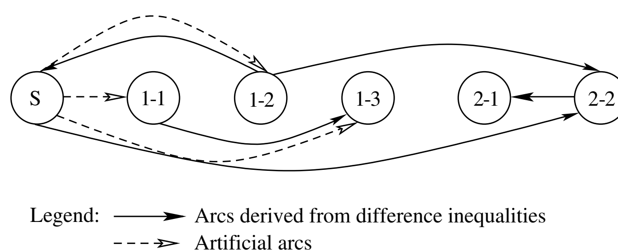
To conclude this section, let us discuss the assumption on the partial ordering of the tasks that is only needed to classify the incompatible columns and to transform inequalities (4.2) into difference inequalities. When this assumption does not hold, the dynamic constraint aggregation method can still be applied. Indeed, on the one hand, if the order of the tasks in a path is totally irrelevant for the problem as in the binary cutting-stock problem (see Vance et al. 1994), then an arbitrary task ordering can be defined at the beginning of the solution process and the tasks in a path can always be ordered according to it. On the other hand, if this order is important as in the vehicle-routing problem with (large) time windows, where the cost of a path depends on the order of the tasks it contains (see Desrochers et al. 1992), an arbitrary task ordering can be defined at the beginning of the solution process and revised each time that the partition is modified. In this case, a path that covers in disorder two tasks belonging to the same equivalence class is simply associated with an O-incompatible column.

4.3. Algorithm Convergence

In this section, we show that the dynamic constraint aggregation method presented above converges to an optimal solution of the MP (2.1)–(2.4) in a finite number of minor iterations, that is, a finite number of times where the ARMP is solved in Step 6. The demonstration relies on three propositions that are stated and proven below. We conclude this section by highlighting the fact that this convergence is similar to the convergence of the primal simplex algorithm.

Observe first that the artificial variables Y_w, $w \in W$, ensure the feasibility of the MP as well as the feasibility of the ARMP_Q for any partition Q. The following proposition provides a relationship between the optimal basic solutions

Figure 2. Directed network G for the example.



computed by the simplex algorithm for the ARMP and the solutions of the MP.

Proposition 1. *Let Q be the partition at a given minor iteration, x_Q the optimal basic solution computed for the $ARMP_Q$ at this iteration, and z_Q the cost of this solution. Then, x_Q can be extended to form a basic feasible solution x of the MP, whose cost is also z_Q.*

Proof. Let R_Q be the set of representative tasks associated with the rows of the ARMP_Q. It is easy to prove that the solution x can be obtained by assigning to all variables considered in the ARMP_Q their corresponding values in x_Q, to all artificial variables Y_w, $w \in W\backslash R_Q$, the values of their corresponding representative task artificial variables, and to all other variables the zero value. The basis associated with x is then composed of all the positive-valued variables and a subset of the artificial variables, including all the artificial variables Y_w, $w \in W\backslash R_Q$. □

Next, denote by Z_i the optimal value of the ARMP computed in Step 6 at minor iteration i. The following proposition shows that the optimal value of the ARMP cannot increase from one minor iteration to the next.

Proposition 2. *For any two consecutive minor iterations i and $i+1$ of the dynamic constraint aggregation algorithm, $Z_i \geqslant Z_{i+1}$.*

Proof. First, assume that no partition adjustment is performed after iteration i (the test in Step 13 is false). In this case, a standard column generation iteration is executed for an MP that involves a restricted number of set-partitioning constraints. Hence, $Z_i \geqslant Z_{i+1}$.

Next, assume that a partition adjustment is performed after iteration i, where the set C is updated in Step 16 or in Step 18. Denote by Q_i and Q_{i+1} the partitions used in Step 6 at minor iterations i and $i+1$, respectively. Observe that all path variables taking a positive value in the solution of the ARMP_{Q_i} are also compatible with the partition Q_{i+1}. Thus, they are part of $P'_{Q_{i+1}}$ and considered in the $\text{ARMP}_{Q_{i+1}}$. Consequently, a feasible solution for the $\text{ARMP}_{Q_{i+1}}$ can be built by keeping the same values for these variables, setting the other path variables to zero and duplicating appropriately the values of the artificial variables as in the proof of Proposition 1. Because this feasible solution has a cost of Z_i, the optimal value Z_{i+1} of the $\text{ARMP}_{Q_{i+1}}$ cannot exceed Z_i. □

The following proposition completes the proof on the finite convergence of the algorithm. The proof proceeds by analyzing the sequence of blocks of minor iterations defined by the sequence of major iterations: We show that the objective cannot increase between major iterations, that the number of consecutive major iterations with the same objective value is bounded, and that the number of different objective values is finite. Here we assume that the primal simplex algorithm used to solve the ARMP includes an anticycling strategy that takes care of degeneracy and that the oracle always solves the subproblems in finite time. Also, recall from §2 that the MP is bounded. It therefore possesses an optimal solution because it is also feasible.

Proposition 3. *The dynamic constraint aggregation algorithm requires a finite number of minor iterations to find an optimal solution to the MP.*

Proof. As mentioned earlier, minor iterations correspond to standard column generation iterations on the ARMP. Therefore, due to the finite convergence of the column generation method, there can only be a finite number of minor iterations within a major iteration. The proof thus consists of showing that there cannot be an infinite number of major iterations.

First, Proposition 2 specifies that the optimal value Z of the ARMP cannot increase from one major iteration to the next. Second, let us show that there can only be a finite number of major iterations where the optimal value Z of the ARMP decreases. By Proposition 1, we know that every computed solution for the ARMP corresponds to a basic feasible solution of the MP that bears the same cost. From linear programming theory, the MP has a finite number of basic solutions and, consequently, a finite number of basic solutions with different costs. Putting these two remarks together, we deduce that an infinite number of ARMP basic solutions with different costs cannot exist. Because Z is nonincreasing between two consecutive major iterations, the same ARMP solution cannot be computed more than once when Z decreases, and there can only be a finite number of such decreases.

Third, there can only be a finite number of consecutive major iterations without a decrease in the optimal value Z of the ARMP. Indeed, when there is no decrease between two major iterations (that is, $Z = Z_{\text{old}}$ in Step 15), partition Q is disaggregated to enlarge the set of compatible columns. Because Q becomes fully disaggregated after a maximum of $|W|-1$ consecutive disaggregations, at most $|W|-1$ consecutive major iterations can be performed without a decrease of Z. Note that when Q is fully disaggregated, all columns are compatible with it, and either the algorithm stops or a decrease must occur.

The proof is complete because we have shown that there are no two consecutive minor iterations with an increase in Z, and a finite number of consecutive minor iterations with a decreased or constant Z value. □

To conclude this section, we would like to point out that the dynamic constraint aggregation algorithm can have a convergence that is similar to the convergence of the simplex algorithm in the sense that the objective value of the solutions computed after every simplex pivot does not increase. Proposition 2 shows that the optimal value of the ARMP is not increasing from one iteration to the next, while the simplex algorithm ensures such a property when solving the ARMP in Step 6. Now consider what happens to the objective value between the last simplex pivot of an

iteration and the first pivot of the next iteration. When no partition adjustment is performed, this transition is smooth and the objective does not increase. However, when the partition changes, one can always build (as discussed in the proof of Proposition 2) from the solution of the last iteration a feasible basic solution for the next ARMP that has the same cost. This basic solution can be supplied to the simplex algorithm to avoid executing a feasibility phase that would inevitably cause the objective value to increase.

5. Computational Experimentation

A series of computational experiments were conducted to test the efficiency of the dynamic constraint aggregation algorithm proposed in this paper. We performed these tests on instances of the vehicle and crew-scheduling problem (VCSP) in urban mass transit systems, an NP-hard problem that has recently been addressed by Freling et al. (1999) and Haase et al. (2001) using column generation approaches. The VCSP was chosen because good initial task partitions can easily be obtained for this type of problem. Before presenting the results, we briefly define the VCSP and discuss details on the implementation of the dynamic constraint aggregation approach for the VCSP.

5.1. VCSP Definition

The following VCSP definition is a summary of the definition presented in Haase et al. (2001). The VCSP consists of determining bus and crew schedules simultaneously to accomplish at minimum cost a set of timetabled trips running along a set of bus lines while satisfying a variety of crew-scheduling constraints imposed by collective agreement and internal regulations. Bus schedules correspond to alternating sequences of *deadhead* and timetabled trips. Deadhead trips are empty of passengers and used to reposition buses. We assume that all buses are identical and based in a single depot.

Driver schedules, called *duties*, are more complex because driver exchanges can occur at various locations along the timetabled trips. These locations are called *relief points* and divide each trip into consecutive segments. A duty is composed of a sequence of segments, deadhead trips, and breaks. In general, there are several valid duty types. For our tests, two duty types were considered: straight duties that contain no breaks and split duties that include a single break. Note that a driver can only change buses after a break. Therefore, it is quite obvious that most consecutive segments along the trips will also be consecutive in optimal VCSP driver schedules, hence the idea of task aggregation.

The main constraints of this problem are as follows: A bus must be assigned to each timetabled trip, a driver must be assigned to each segment, and a driver must be assigned to each deadhead trip that is part of a bus schedule. The cost structure involves operational costs proportional to the mileage traveled by the buses and to the time worked by the drivers, as well as bus and driver fixed costs.

5.2. Implementation Details

As proposed by Haase et al. (2001), we formulated the VCSP as a set-partitioning-type model solvable by branch and price. In the corresponding MP, a commodity is defined for each type of duty, a θ_p^k variable for each feasible driver duty, and a set-partitioning constraint for each following task: drive a bus to the start location of each timetabled trip, operate a bus along each segment, and drive a bus away from the end location of each trip. This MP also includes an additional variable and a supplementary set of constraints to count the buses and ensure the optimality of the bus schedules, which are derived a posteriori. A subproblem is defined for each duty type. These subproblems correspond to constrained shortest-path problems that are solved by a generalization of the dynamic programming algorithm of Desrochers and Soumis (1988).

The dynamic constraint aggregation algorithm was customized as follows for the VCSP. First, the partial ordering of the tasks is given by the chronological order of the task start times. Second, an initial set of paths C is created by constructing for each timetabled trip a path that starts from the depot with a deadhead trip, covers all the segments in this trip, and returns to the depot with another deadhead trip. Indeed, it is very likely that the consecutive segments along each timetabled trip remain consecutive in an optimal solution. Note, however, that some of these paths might not be feasible with respect to the working rules but, as mentioned earlier, this is not necessary. An initial partition Q is then defined with respect to the equivalence relation $\mathcal{R}^C$.

Third, for computing the disaggregated dual variables, the linear system (4.1) and (4.2) is transformed into a shortest-path problem, as explained in §4.2.2. In the VCSP context, the set $\bar{P}_Q^*$ used to define the underlying network is restricted to a subset of the previously generated incompatible columns, namely, the S-, M-, and ES-incompatible columns. The SE- and E-incompatible columns are ignored to ensure that this network is free of negative directed cycles, a rare situation when they were considered in the preliminary tests that we performed. Furthermore, ignoring these columns yields an acyclic network. The corresponding shortest-path problem can then be solved by sorting the nodes in topological order before executing a single pass of Bellman's (1958) shortest-path algorithm. The solution provides the values of the π variables, which allows us to compute the values of the disaggregated α variables through the lower triangular linear systems (4.3).

Fourth, the set I in Step 14 of the algorithm is restricted to a subset of disjoint incompatible columns, where two columns p_1 and p_2 are said to be *disjoint* if $T_{p_1} \cap T_{p_2} = \varnothing$. This restriction originates from the fact that any feasible solution of a set-partitioning problem is formed of disjoint columns.

Finally, let us specify that the parameters λ, δ, and γ used for partition handling (see §4.2.1) were set for the computational experiments to 0.5, 0.6, and 6, respectively.

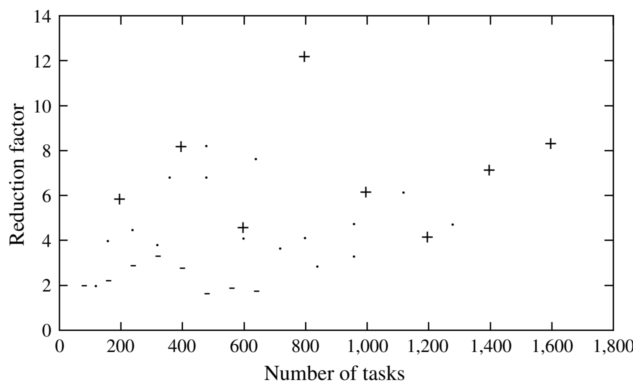
5.3. Computational Results

The tests on the VCSP instances were limited to solving the linear relaxations of these instances. This way, the dynamic constraint aggregation algorithm and the standard column generation method can be fairly compared without the interference of the branch-and-bound process. All tests were performed on a DELL i386 single processor Redhat Linux machine (Intel Pentium 4, Type i686 CPU, 1.8 GHZ) using GENCOL 4.3, a software developed at the GERAD research center in Montreal and now owned by AD OPT Technologies. GENCOL supports the standard column generation method and relies on the CPLEX optimizer (release 7.5) for solving the restricted MPs. It also constitutes a benchmark for this research.

Test instances were randomly generated using the same generator as in Haase et al. (2001). Overall, 32 instances that differ from the number of timetabled trips to cover (20, 40, 60, 80, 100, 120, 140, or 160) and the number of segments per trip (2, 4, 6, or 8) were generated, one for each possible combination. For all instances, the trips were generated using the same bus lines. Therefore, instances with a high (respectively, low) number of segments per trip have short (respectively, long) segments. From §5.1, we can deduce that the number of tasks (or equivalently, the number of set-partitioning constraints in the MP) is equal to the number of trips times the number of segments per trip plus two.

For the 32 instances studied, the dynamic constraint aggregation algorithm succeeded in reducing the MP solution time when compared to the standard column generation method. Figure 3 presents the reduction factor obtained as a ratio of the MP solution times (standard column generation time over dynamic constraint aggregation time) for all 32 instances. In this graph, points are illustrated using three symbols ($-$, $+$, and $\cdot$). The $-$ signs correspond to the instances with two segments per trip. These instances with longer segments have a mean of 5.8 tasks per column in the optimal solution. Instances with eight segments per trip are represented by the $+$ points. These short segment instances exhibit a mean of 12.2 tasks per column in the optimal solution. Finally, the $\cdot$ points correspond to instances with four and six segments per trip. The optimal solutions of these problems contain a mean of 8.8 tasks per column when the mean is taken over all instances, while this mean varies between 6.3 and 12.2 for individual instances.

Figure 3. Reduction factor of the master problem solution time.



One can observe that, for these instances, the reduction factor ranges between 1.7 and 12.2. This factor grows significantly with the size of the instances. In fact, for instances with 700 tasks or more the reduction factor is at least 3, for those with 1,000 tasks or more the reduction factor is at least 4, and for those with 1,300 tasks or more the reduction factor is at least 7. Also, we observe that the reduction factor grows as the number of tasks per column increases.

Three tables of results are presented (Tables 4–6). Each table is divided into two parts: The size of the instances are presented in the top part, while the bottom part shows the results. In these tables, we refer to the dynamic constraint aggregation method with the acronym “`dca`” and to the standard column generation method with the acronym “`std`.” Because the time consumed by the initial partition heuristic is not significant (<0.001 second), it is not reported. Note that, as shown in these tables, the MP solution time often represents more than 70% of the total solution time (the average is 79%) when the standard column generation method is used. This fact motivates dynamically aggregating some MP constraints.

Table 4 presents the results for 40-trip instances when the number of segments per trip varies. This table shows that the dynamic constraint aggregation algorithm reduces the MP solution time by a percentage ranging from 66% to 88%. This is due to a substantial reduction in the mean number of constraints in the MP at each column generation (minor) iteration. On the other hand, the number of minor iterations is increased because the new algorithm performs several iterations with a nonoptimal partition. This increase yields higher subproblem solution times. Nevertheless, the overall solution times are reduced by percentages that range from 21% to 62%.

Better results for the 80-trip instances with a varying number of segments are reported in Table 5. For these larger instances, the MP solution times are reduced by percentages ranging from 66% to 92% when the dynamic constraint aggregation approach is used. Gains on the total solution time vary between 33% and 77%. Note that the amount of memory used is also reduced by up to 40%.

Table 6 provides the results for instances with eight segments per trip and different numbers of trips. Again, the dynamic constraint aggregation method succeeds in reducing the MP solution time by up to 90%, as well as the total solution time by up to 80%. For these instances, the mean number of MP constraints is diminished by an average of 39% in the dynamic constraint aggregation algorithm. Note also that the algorithm updates the partition about once every two minor iterations. Most of the partition

Table 4. Results for 40-trip instances.

Instance size								
Number of segments per trip	2		4		6		8	
Number of tasks	160		240		320		400	
Results	dca	std	dca	std	dca	std	dca	std
Number of column generation iterations	38	36	69	41	99	53	121	76
Number of partition changes	16	—	33	—	48	—	56	—
Mean number of constraints	114	175	156	255	192	335	228	415
Memory (Mb)	1.9	1.8	4.2	4.4	7.6	8.8	11.2	15.3
Dual variable disaggregation (sec.)	<0.1	—	<0.1	—	0.1	—	0.3	—
Partition adjustment (sec.)	<0.1	—	0.1	—	0.2	—	0.4	—
Subproblem time (sec.)	0.5	0.4	2	2	6	4	12	8
Master problem time (sec.)	0.2	0.7	1	5	3	9	5	42
Total time (sec.)	1.0	1.3	4	7	11	14	20	52

changes occur at the beginning of the solution process, and the number of partition updates (major iterations) decreases progressively. For example, for the 40-trip instances with eight segments per trip, 20 partition changes are performed in the first 40 column generation iterations, 19 in the next 40 iterations, and only 8 in the remaining 41 iterations (see Figure 4). This shows that on the road to the optimal solution, the algorithm updates the partition less frequently towards the end, that is, the partition stabilizes. Figure 4 also shows that, in this example, the partition is only aggregated twice after the initial aggregation.

Table 6 also provides the mean number of degenerate basic variables throughout the solution process and the number of fractional variables at the end of it. The dynamic constraint aggregation algorithm again produces very interesting results with regard to those two statistics. Indeed, the mean number of degenerate basic variables at each iteration is reduced by an average of 58%. In percentage of the number of constraints, this mean number represents 38% when using the dynamic constraint aggregation algorithm and 56% when using the standard column generation method. This reduction comes from the diminution of the number of nonzero elements per column with the aggregation process. Consequently, the effects of degeneracy are reduced. We observe also that the number of fractional-valued variables at the end of the solution process is less when dynamic constraint aggregation is used for solving large instances (⩾120 trips). Therefore, we expect that the dynamic constraint aggregation algorithm will also reduce the number of branch-and-bound nodes for large instances.

In all of the test cases above, the computational times of dual variable disaggregation and partition adjustment were not significant. Indeed, they were less than 1.5% and 3% of the total time, respectively. This illustrates the efficiency of the dual variable and partition handling strategies. In particular, it shows that it is not too time consuming to compute dual variables that satisfy a large number of constraints (4.2).

The dynamic constraint aggregation approach outperforms the standard column generation method in all test cases. They behave in the same way for small test instances, with a slight advantage for the dynamic constraint aggregation method, but for large instances, the ratio of time reduction becomes very significant. We can say that the dynamic constraint aggregation algorithm is very useful when the size of the problem is large.

Table 5. Results for 80-trip instances.

Instance size								
Number of segments per trip	2		4		6		8	
Number of tasks	320		480		640		800	
Results	dca	std	dca	std	dca	std	dca	std
Number of column generation iterations	82	45	137	79	191	121	270	174
Number of partition changes	30	—	66	—	95	—	123	—
Mean number of constraints	241	353	311	513	397	673	459	833
Memory (Mb)	6.5	6.7	17.3	20.1	30.4	43.1	47.7	79.2
Dual variable disaggregation (sec.)	0.1	—	0.4	—	0.9	—	1.7	—
Partition adjustment (sec.)	0.3	—	1.1	—	2.2	—	4.4	—
Subproblem time (sec.)	5	3	23	14	61	41	144	101
Master problem time (sec.)	4	12	12	80	40	291	72	884
Total time (sec.)	10	15	40	96	108	337	230	993

Table 6. Results for instances with eight segments per trip.

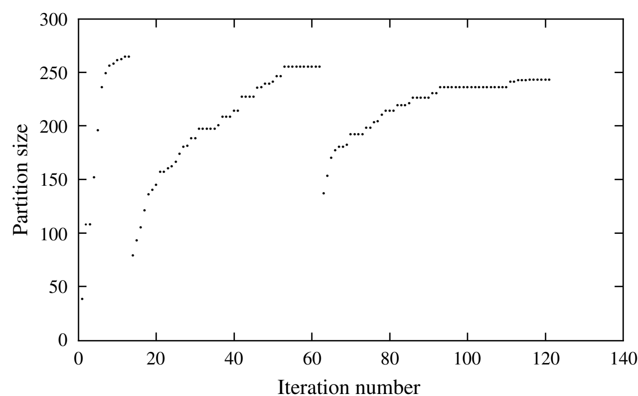
Instance size								
Number of trips	40		80		120		160	
Number of tasks	400		800		1,200		1,600	
Results	dca	std	dca	std	dca	std	dca	std
Number of column generation iterations	121	76	270	174	528	343	640	547
Number of partition changes	56	—	123	—	194	—	304	—
Mean number of constraints	228	415	459	833	832	1,252	1,017	1,662
Mean number of degenerate basic variable	98	248	176	496	313	622	375	895
Number of fractional variable	37	40	73	71	117	173	151	212
Memory (Mb)	11	15	48	79	140	190	245	314
Dual variable disaggregation (sec.)	0.3	—	1.7	—	8.9	—	15.2	—
Partition adjustment (sec.)	0.4	—	4.3	—	17.2	—	42.5	—
Subproblem time (sec.)	12	8	144	101	861	547	1,952	1,698
Master problem time (sec.)	5	42	72	884	1,405	5,835	2,373	19,771
Total time (sec.)	20	52	230	993	2,312	6,402	4,421	21,508

6. Conclusion

Dynamic constraint aggregation is a novel approach that reduces the number of set-partitioning constraints in column generation MPs, allowing shorter solution times. The algorithm defines and modifies an equivalence relation over the tasks (associated with the set-partitioning constraints), solves linear programs of reduced size, and uses a shortest-path algorithm to recover the nonaggregated dual information needed by the column generation subproblems.

Results for the VCSP show that the dynamic constraint aggregation approach outperforms the standard column generation method in all test cases. In fact, the dynamic constraint aggregation algorithm reduces the number of constraints by an average of 39% and the MP time by up to 90%. We have also shown that the linear relaxation solution time is reduced by up to 80% for large instances. This technique is especially useful for solving very large instances not yet solved efficiently by the present methods.

Figure 4. Number of set-partitioning constraints at each iteration.



Even though we have only used basic strategies, results are very significant. Future research will focus on designing and analyzing more sophisticated strategies, on merging this new methodology with interior point methods, and on integrating it within a branch-and-bound scheme.

References

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows*. Prentice Hall, Englewood Cliffs, NJ.

Barahona, F., R. Anbil. 2000. The volume algorithm: Producing primal solutions with a subgradient method. *Math. Programming* **87** 385–399.

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* **46** 316–329.

Bellman, R. E. 1958. On a routing problem. *Quart. Appl. Math.* **16** 87–90.

Cordeau, J.-F., G. Desaulniers, N. Lingaya, F. Soumis, J. Desrosiers. 2001. Simultaneous locomotive and car assignment at VIA Rail Canada. *Transportation Res. B* **35** 767–787.

Dantzig, G. B., P. Wolfe. 1960. Decomposition principle for linear programs. *Oper. Res.* **8** 101–111.

Desaulniers, G., J. Desrosiers, I. Ioachim, F. Soumis, M. M. Solomon, D. Villeneuve. 1998. A unified framework for time constrained vehicle routing and crew scheduling problems. T. G. Crainic, G. Laporte, eds. *Fleet Management and Logistics*. Kluwer, Norwell, MA, 57–93.

Desrochers, M., F. Soumis. 1988. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR* **26** 191–212.

Desrochers, M., F. Soumis. 1989. A column generation approach to the urban transit crew scheduling problem. *Transportation Sci.* **23** 1–13.

Desrochers, M., J. Desrosiers, F. Soumis. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.* **40** 342–354.

Desrosiers, J., F. Soumis, M. Desrochers. 1984. Routing with time windows by column generation. *Networks* **14** 545–565.

du Merle, O., D. Villeneuve, J. Desrosiers, P. Hansen. 1999. Stabilized column generation. *Discrete Math.* **194** 229–237.

Fisher, M. L. 1981. The Lagrangian relaxation method for solving integer programming problems. *Management Sci.* **27** 1–18.

Freling, R., A. P. M. Wagelmans, J. M. P. Paixão. 1999. An overview of models and techniques for integrating vehicle and crew scheduling. N. H. M. Wilson, ed. *Computer-Aided Transit Scheduling. Lecture Notes in Economics and Mathematical Systems*, Vol. 471. Springer-Verlag, Heidelberg, Germany, 441–460.

Gamache, M., F. Soumis, G. Marquis, J. Desrosiers. 1999. A column generation approach for large scale aircrew rostering problems. *Oper. Res.* **47** 247–263.

Geoffrion, A. M. 1974. Lagrangean relaxation for integer programming. *Math. Programming Stud.* **2** 82–114.

Gilmore, P. C., R. E. Gomory. 1961. A linear programming approach to the cutting stock problem. *Oper. Res.* **9** 849–859.

Goffin, J.-L., J.-P. Vial. 1990. Cutting plane and column generation techniques with the projective algorithm. *J. Optim. Theory Appl.* **65** 409–429.

Goffin, J.-L., J.-P. Vial. 2002. Convex nondifferentiable optimization: A survey focussed on the analytic center cutting plane method. *Optim. Methods Software* **17** 805–867.

Haase, K., G. Desaulniers, J. Desrosiers. 2001. Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Sci.* **35** 286–303.

Held, M., R. M. Karp. 1971. The traveling salesman problem and minimum spanning trees: Part II. *Math. Programming* **1** 6–25.

Hiriart-Urruty, J., C. Lemaréchal. 1991. *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods. A Series of Comprehensive Studies in Mathematics*. Springer, New York.

Kelley, J. E. 1960. The cutting-plane method for solving convex programs. *J. SIAM* **8** 703–712.

Nemhauser, G. L., L. A. Wolsey. 1988. *Integer and Combinatorial Optimization*. Wiley, New York.

Vance, P. H., C. Barnhart, E. L. Johnson, G. L. Nemhauser. 1994. Solving binary cutting stock problems by column generation and branch-and-bound. *Comput. Optim. Appl.* **3**(2) 111–130.

Villeneuve, D. 1999. Logiciel de génération de Colonnes. Ph.D. dissertation, Université de Montréal, Montreal, Quebec, Canada.

Copyright 2005, by INFORMS, all rights reserved. Copyright of Operations Research is the property of INFORMS: Institute for Operations Research and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.