Column generation algorithms for nonlinear optimization, II: Numerical investigations

Ricardo García-Ródenas [a], Angel Marín [b,*], Michael Patriksson [c,d]

[a] *Universidad de Castilla La Mancha, Escuela Superior de Informática, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain*
[b] *Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros Aeronáuticos, Plaza Cardenal Cisneros, 3, 28040 Madrid, Spain*
[c] *Department of Mathematical Sciences, Chalmers University of Technology, SE-412 96 Göteborg, Sweden*
[d] *Department of Mathematical Sciences, University of Gothenburg, SE-412 96 Göteborg, Sweden*

A B S T R A C T

García et al. [1] present a class of column generation (CG) algorithms for nonlinear programs. Its main motivation from a theoretical viewpoint is that under some circumstances, finite convergence can be achieved, in much the same way as for the classic simplicial decomposition method; the main practical motivation is that within the class there are certain nonlinear column generation problems that can accelerate the convergence of a solution approach which generates a sequence of feasible points. This algorithm can, for example, accelerate simplicial decomposition schemes by making the subproblems nonlinear. This paper complements the theoretical study on the asymptotic and finite convergence of these methods given in [1] with an experimental study focused on their computational efficiency.

Three types of numerical experiments are conducted. The first group of test problems has been designed to study the parameters involved in these methods. The second group has been designed to investigate the role and the computation of the prolongation of the generated columns to the relative boundary. The last one has been designed to carry out a more complete investigation of the difference in computational efficiency between linear and nonlinear column generation approaches.

In order to carry out this investigation, we consider two types of test problems: the first one is the nonlinear, capacitated single-commodity network flow problem of which several large-scale instances with varied degrees of nonlinearity and total capacity are constructed and investigated, and the second one is a combined traffic assignment model.

1. Introduction

Analyzing and solving combined traffic models for large-scale networks of practical interest requires computationally efficient solution methods for the following constrained optimization problem:

$$\operatorname*{minimize}_{x\in X} f(x), \qquad (\mathrm{CDP}(f,X))$$

where $X \subseteq \mathbb{R}^n$ is non-empty and convex, and $f : X \mapsto \mathbb{R}$ is continuously differentiable on X.

The class of simplicial decomposition methods successfully solves network equilibrium models formulated as CDP(f,X). At present, certain mathematical programming problems with equilibrium constraints, such as the capacity enhancement problem, the congestion toll pricing problem, the signal setting problem and the origin–destination matrix problem have led to the appearance of methods based on sensitivity analysis [2]. These new methods should solve many of these CDP(f,X) with great accuracy. This motivates an interest in accelerating classical simplicial decomposition methods, which is the objective of this work. For example, [3] analyses new ways of generating columns by exploiting the acyclicity of user equilibrium flows.

We consider here the class of simplicial decomposition methods for this problem, as represented by the work in Holloway [4], von Hohenbalken [5], Hearn et al. [6,7], Ventura and Hearn [8], Larsson et al. [9], Patriksson [10], Rosas et al. [11], Bertsekas and Yu et al. [12]. There are two main characteristics of the methods in this class: (i) an approximation of the original problem is constructed and solved, wherein the original feasible set is replaced by a compact subset, which is an *inner approximation*, and is the so-called *restricted master problem* (RMP); and (ii) this inner approximation is improved by generating a new vector (column) in the feasible set through the solution of another approximation of the original problem wherein the cost function is approximated. This stage is called the *column generation problem* (CGP).

* Corresponding author. Tel.: +34 91 3366323; fax:+34 91 3366324.
E-mail addresses: Ricardo.Garcia@uclm.es (R. García-Ródenas), angel.marin@upm.es (A. Marín), mipat@chalmers.se (M. Patriksson).

Concretely, we consider:

(1) The column generation problem is characterized by an iterative procedure. This iterative procedure is denoted by $\mathcal{A}_c^k$ and belongs to a finite collection $\mathcal{K}_c$. The assumptions on $\mathcal{A}_c^k$ are associated with an algorithm that operates as a descent algorithm on the original problem. In order to rule out the uninteresting case that the original problem is solved by means of only using the column generation problem an infinite number of times starting from a given iterate $x \in X$, we presume that the number of iterations performed from x is finite.
(2) The restricted master problem is assumed to be solved by the use of an iterative procedure, denoted $\mathcal{A}_r^k$ and belonging to a finite collection $\mathcal{K}_r$. It operates on $\hat{X} \subset X$ being equal to the current inner approximation of X. Also this algorithm will be presumed to be applied a finite number of times; we can still solve any given RMP exactly in this way, by making the proper choice of the procedure $\mathcal{A}_r^k$.

In Table 1, we summarize the different steps of this CG algorithm. The algorithm described is conceptual, but important algorithms are readily placed in its framework. The concept of the CG method was first outlined and established convergent in Larsson et al. [13]. García et al. [1] establish the global (asymptotic) convergence for the CG algorithm under the above assumptions on $\mathcal{A}_c^k$ with k belonging to $\mathcal{K}_c$ and $\mathcal{A}_r^k$ with k belonging to $\mathcal{K}_r$ similar to those utilized in the convergence analysis in Zangwill [14]. Finite convergence results with respect to the optimal face and the optimal solution are established for weak sharp minima.

The CG method offers a different viewpoint of the CGP from previous simplicial decomposition methods. There, the CGP is seen as an approximation to the original problem which provides a new column by means of its (truncated) solution. The approach taken in the CG algorithm is to construct profitable columns by roughly solving the *original* problem through the use of a limited number of iterations of a convergent algorithm. Now, the emphasis is placed on the algorithm used to generate a new column, and not on the definition of the approximation subproblem.

Previous simplicial decomposition methods, however, constitute instances of the CG algorithm. The classic *simplicial decomposition* method discussed earlier can be described by considering problems where X is a polyhedral set, $\mathcal{A}_c^k$ describes the Frank–Wolfe algorithm, and only one iteration is performed. The nonlinear simplicial decomposition (NSD) method of Larsson et al. [9], which is applied to the traffic assignment problem (see Sheffi [15], Patriksson [16]) can be described as an instance of the CG algorithm where $\mathcal{A}_c^k$ realizes one iteration of a truncated Newton algorithm, and in which the quadratic subproblems which appear in the Newton type algorithm are approximately solved using the Frank–Wolfe algorithm. The RSDCC method of Ventura and Hearn [8] follows from letting $\mathcal{A}_c$ describe the Topkis–Veinott algorithm, of which only one iteration is performed. Daneva et al. [17] provide a Frank–Wolfe type method for the stochastic transportation problem, which includes a multidimensional search.

The set augmentation rules applied in the original work on simplicial decomposition [4,5], as well as in the later developments in Hearn et al. [7] can be used in this framework as well. The role played by the extreme points is now assumed by the column generated. Since the columns generated by the CG algorithm are not necessarily extreme points or even boundary points, Larsson et al. [18] propose extending the column generated to the (relative) boundary. Under the realization of this operation the CG algorithm can be interpreted as a (restricted) simplicial decomposition method where the choice of columns belonging to the set of extreme points of the feasible region is enriched by the inclusion also of other points on the boundary.

While the CG method appears to have traditional, in particular primal, sub and master problems, one may derive through the CG framework also primal-dual types of methods. Among other methods, Larsson et al. [13] derive the Dantzig–Wolfe method in linear programming as a CG method, as well as a sequential quadratic programming algorithm with a side constrained master problem. Daneva et al. [19] similarly derive a sequential linear programming type method.

Table 2 summarizes these rules, which constitute realizations of Step 3 in the conceptual algorithm of Table 1. (The corresponding necessary initialization step is not included.)

1.1. Motivations

García et al. [1] have theoretically analyzed the asymptotic and finite convergence of the CG algorithm, but this study is not sufficient from the viewpoint of applications. This paper is a complementary study which computationally addresses important questions about the performance of the CG algorithms in terms of the elements that define them.

The following three main aspects of the CG algorithms are studied in this paper.

Performance of the CG algorithms: The SD algorithm alternates between the solution of two subproblems, the CGP and the RMP, in each iteration. The number of variables of the RMPs grows when this algorithm progresses, because in each iteration one new extreme point (a new variable) is stored and is usually not

Table 1
The conceptual algorithm.

0. (*Initialization*): Choose an initial point $x^0 \in X$, and let $t := 0$
1. (*Column generation problem*): Choose an algorithm $\mathcal{A}_c^{k_t}$, $k_t \in \mathcal{K}_c$. Apply a finite number of iterations on CDP(f,X), starting from x^t. Let the resulting point be y^t
2. (*Termination criterion*): If $x^t \in \text{SOL}(\nabla f, X) \rightarrow$ Stop. Otherwise, continue
3. (*Set augmentation*): Let $X^{t+1} \subset X$ be a non-empty, compact and convex set containing $[x^t, y^t]$
4. (*Restricted master problem*): Choose an algorithm $\mathcal{A}_r^{k_t}$, $k_t \in \mathcal{K}_r$. Apply at least one iteration of the algorithm on CDP (f, X^{t+1}), starting from x^t. Let the resulting point be x^{t+1}
5. (*Update*): Let $t := t+1$. Go to Step 1

Table 2
The set augmentation phase.

0. (*Initialization*): choose an initial point $x^0 \in X$, let $t := 0$, $\mathcal{P}_s^0 = \emptyset$, $\mathcal{P}_x^0 = \{x^0\}$, $\mathcal{P}^0 = \mathcal{P}_s^0 \cup \mathcal{P}_x^0$ and $X^0 = \text{conv}(\mathcal{P}^0)$. Further, let r be a positive integer, and let $\{\varepsilon_1^t\} \rightarrow 0$ be a sequence of positive real numbers

3.1 *Column dropping rules*): let $x^t = \sum_{i=1}^m \beta_i p_i$, where $m = |\mathcal{P}^t|$ and $p_i \in \mathcal{P}^t$

3.1.a (Exact solution of RMP). Discard all elements p_i with weight $\beta_i = 0$

3.1.b (Truncated solution of RMP). Discard all elements p_i satisfying
$$\nabla f(x^t)^{\mathrm{T}}(p_i - x^t) \geq \varepsilon_1^t > 0 \tag{1}$$

3.2 (*Extension to the relative boundary of X*): Let $\hat{y}^t$ be the vector generated, and let y^t be defined by
$$y^t = x^t + \ell_t(\hat{y}^t - x^t); \quad \ell_t = \max\{\ell | x^t + \ell(\hat{y}^t - x^t) \in X\} \tag{2}$$

3.3. (*Set augmentation rules*):

3.3.a (Simplicial decomposition scheme). $\mathcal{P}^{t+1} = \mathcal{P}^t \cup \{y^t\}$. Set $X^{t+1} = \text{conv}(\mathcal{P}^{t+1})$

3.3.b (Restricted simplicial decomposition scheme). If $|\mathcal{P}_s^t| < r$, then set $\mathcal{P}_s^{t+1} = \mathcal{P}_s^t \cup \{y^t\}$, and $\mathcal{P}_x^{t+1} = \mathcal{P}_x^t$; otherwise, replace the element of $\mathcal{P}_s^t$ with minimal weight in the expression of x^t with y^t to obtain $\mathcal{P}_s^{t+1}$, and let $\mathcal{P}_x^{t+1} = \{x^t\}$. Finally, set $\mathcal{P}^{t+1} = \mathcal{P}_s^{t+1} \cup \mathcal{P}_x^{t+1}$ and $X^{t+1} = \text{conv}(\mathcal{P}^{t+1})$

dropped. On the other hand, the CPU time to generate the extreme points along the process is constant, and depends only on the size and type of the original problem. In the applications considered here, for example, this time is less for single-commodity network flow problems than for multi-commodity flow problems. The CPU time for solving the CGPs is approximately linear in the number of iterations but it is nonlinear for solving the RMPs.

In this paper, we study the concept of the "quality" of the columns in a CG algorithm, in particular its role in practice for large-scale, structured problems. For example, we will investigate the effect of increasing the number of times that $\mathcal{A}_c$ is applied in the CGP on the convergence of the overall algorithm compared to the cost of generating the corresponding columns.

The role of the prolongation to the relative boundary: As was described earlier, the column generation problem in the nonlinear SD method of Larsson et al. [9] is a nonlinear approximation of the original problem. They observed that rather than retaining the subproblem solution vector y^t as the new column, it is always an advantage to instead store the vector $y^t+\ell(y^t-x^t)$ for the maximum value of $\ell \geq 1$, that is, the prolongation of y^t to the relative boundary of X in the direction of y^t-x^t from x^t. The main motivation for this operation is to obtain the largest feasible regions of the RMPs possible, and thus the greatest improvements of the solution. The operation is simple in the case where X is polyhedral and does not increase the amount of work to solve the RMPs because the number of variables is equal in both cases (one per column).

Generalization of the methods of feasible directions: A further motivation for this work is to illustrate the potential of improving line search based feasible direction methods by means of its use within a CG scheme. The descent algorithm is then used as a means of generating the columns, that is, by defining instances of the mapping $\mathcal{A}_c$. There are two ways to do this:

- Multidimensional searches: The perhaps most obvious way to generalize a line search method in this way is, at a given iteration of the CG algorithm, to perform one or more iterations of the descent method, after which the resulting vector (or, rather its prolongation to the relative boundary) is stored as a column in the RMP. In the extreme case where only one iteration of the line search method is performed in each step results in a methodology in which, essentially, the line search in this algorithm is replaced by a multidimensional search. The main motivation for this algorithm is that the search directions are given a better opportunity to be used efficiently, since they are not used only once and discarded but instead kept in memory.
- Generalized PARTAN: In the RSD algorithm, the parameter r defines the maximal number of columns retained in the RMP, disregarding the current iterate, x^t. (With $r = 1$, RSD reduces to the Frank–Wolfe algorithm.) In our more general framework, the columns stored are the result of having applied a method for the original problem for a given number of iterations, and therefore choosing $r = 1$ may result in much more efficient algorithms than the Frank–Wolfe method. The effect of making this choice is that after having carried out n_c line searches in the current CGP, the next iterate is produced via yet another line search, but with one end point at the previous iterate. This type of method requires no special algorithm for RMPs, because only line searches are made, and it brings forward natural extensions of the PARTAN technique used previously together with the Frank–Wolfe algorithm.

In this work we have dealt with both extensions. We have conducted numerical tests with the classical Frank–Wolfe and Evans algorithms as $\mathcal{A}_c$, and a comparison with SD, RSD and NSD has been carried out.

2. Applications

In this section we describe the test problems and algorithms used for their resolution in our numerical experiments.

2.1. Test problems

The CG methods have been applied to single-commodity network flow problems, and a network equilibrium model with combined modes. There are several reasons for our choice of applications:

- The test problems are nonlinear network flow ones, and therefore they are amenable to solutions with methods based on the linearization of the objective function. The linear approximation of single-commodity nonlinear network flow problems is a minimal cost network flow problem. The linearization of the combined traffic problem separates into a number of shortest route calculations in two different networks, say A and B, and the computation of the shortest combined paths. Over the three optimal paths (the optimal path using only A, only B and both networks) the best path is chosen, taking into account a cost derived from the equilibrium formulation by modes of transport and transfer nodes between the networks.
- The main motivation for choosing nonlinear single-commodity network flow problems is that it is possible to explicitly compute the prolongation of any vector to the relative boundary of the feasible set. In our approach this step is important in order to produce competitive methods.

2.1.1. Single-commodity network flow problems [SNFP]

Consider a directed graph (N, A) with n nodes and m arcs. For each node $i \in \mathcal{N}$ a scalar s_i is given, where s_i is the strength of the source (if positive) or sink (if negative) flow at node i, and for each arc $(i,j) \in \mathcal{A}$ a convex and continuously differentiable cost function $f_{ij} : \mathbb{R} \mapsto \mathbb{R}$ is given. The nonlinear single-commodity network flow problem with separable arc cost functions is stated as that to

$$\text{minimize} f(x) := \sum_{(i,j) \in \mathcal{A}} f_{ij}(x_{ij}), \qquad \text{(SNFP)}$$

subject to

$$\sum_{\{j|(i,j) \in \mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i) \in \mathcal{A}\}} x_{ji} = s_i, \quad i \in \mathcal{N},$$

$$0 \leq x_{ij} \leq u_{ij}, \quad (i,j) \in \mathcal{A},$$

where the real variable x_{ij} is referred to as the *flow* on the arc (i,j) and the vector $x = (x_{ij})_{(i,j) \in \mathcal{A}}$ is referred to as the *flow vector*. The real numbers u_{ij} are the capacities on the arcs (i,j).

We have created three types of test problems. The first and second ones were developed using two of the public-domain generators: NETGEN, written by Klingman et al. [20], which generates linear-cost assignment/transportation/transhipment problems having a certain random structure, and GRIDGEN, written by Bertsekas [21], which constructs random problems with an underlying two-dimensional grid with wraparound structure. The grid arcs form a *skeleton*, guaranteeing problem feasibility. Additional arcs are added between randomly selected nodes. The third type was created by the authors. The topology of the graph consists of a complete and bipartite graph of size $n \times n$

plus two sets of links which connect an origin node with the nodes of the first set, and the nodes of the second set with a destination node such as is illustrated in Fig. 1.

The test problems have been entitled NET, GRI and AUT to indicate that they have been generated with NETGEN, GRIDGEN and the author's own generator, respectively.

The generators only create the topology of the graphs. We have considered nonlinear costs of the following types:

(1) $f_{ij}(x_{ij}) = a_{ij}(x_{ij}+0.2b_{ij}(x_{ij}^5/c_{ij}^4))$,
(2) $f_{ij}(x_{ij}) = a_{ij}x_{ij}^3+b_{ij}x_{ij}^2+c_{ij}x_{ij}$,
(3) $f_{ij}(x_{ij}) = a_{ij}x_{ij}^2+b_{ij}x_{ij}$,
(4) $f_{ij}(x_{ij}) = a_{ij}x_{ij}(\log(x_{ij}/b_{ij})-c_{ij})$,

where a_{ij}, b_{ij}, c_{ij} are parameters.

Table 3 describes the test problems in more detail. The name of the first problem type is encoded using the convention net+-number+letter. The number indicates a type of network topology and the letters indicate different parameterizations of the problem. With problems of type NET1 we have explored different cost functions. NET2 problems are those of larger size. NET3a and NET4a are ill-conditioned strictly convex quadratic problems as in Bertsekas et al. [22]. These problems were created by assigning to some of the arcs a much smaller (but nonzero) quadratic cost coefficient in comparison with other arcs. 50% of the links have a small coefficient of 1 and the other 50% of the coefficients have been generated by means of a uniform distribution on [5,10]. When the arc cost functions have this structure, ill-conditioning in the traditional sense of unconstrained nonlinear programming tends to occur.

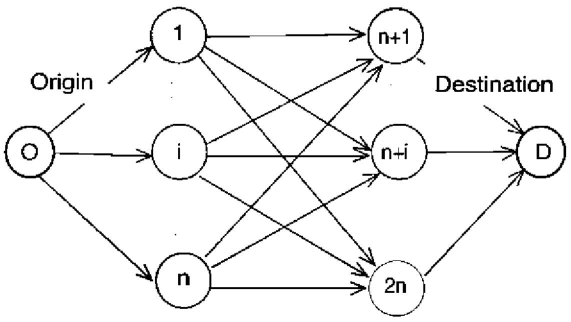

Fig. 1. Topology of the test graph AUT type.

NET3b and NET4b are mixed linear/quadratic cost problems. The linear costs have been obtained by means of replacing the small quadratic coefficients of the NET3a and NET4a problem instances with zero.

The problems generated with GRIDGEN have two kinds of links: the set of links of the grid and a set of random links. The cost coefficient of the $4n$ grid arcs is *MAXCOST*. All other arcs have cost coefficients selected according to a uniform distribution between *MINCOST* and *MAXCOST*. Thus, there is an incentive to avoid the grid arcs as much as possible in an optimal solution. The capacity of the links of the grid is the total demand. This choice guarantees the feasibility of the problem. The capacity of the random links is generated by a uniformly distributed random variable. We have generated two versions of the same problem. The first, whose title ends in *a*, creates the incentive of avoiding the grid by means of generating large numbers of random links. The grids which end in *b*, on the other hand, have fewer links by which to avoid the grid. This reduces the number of paths between the source node and the sink node.

The third type (AUT) of problems are designed to obtain test problems with a large number of links with optimal flows at the upper bounds. We have distinguished between the links with the beginning or the end at the origin or destination nodes, (the so-called exterior links) and the rest of the links (the so-called interior links). The exterior links have a capacity equal to the total demand, causing the maximum flow between the origin node and destination to be the sum of all the capacities of the interior links. This value, denoted by MAXFLOW, has been considered as a parameter in the generation of the graph. We have generated the uniform variables X_i, and Y_{ij} for all $i=1,\dots,n$ and $j=n+1,\dots,2n$. The capacity of the interior link (i,j) is calculated as

$$u_{ij} := \frac{X_i}{\sum_{s=1}^{n} X_s} \frac{Y_{ij}}{\sum_{k=1}^{n} Y_{ik}} \text{ MAXFLOW.}$$

The saturation of the network is monitored by the size of the demand in relationship to the parameter MAXFLOW.

Table 3
Description of the single-commodity test problems.

Problem	Nodes	Arcs	Func.	a_{ij}	b_{ij}	c_{ij}	u_{ij}	Demand
NET1a	500	2500	1	[1–10]	[1–10]	50	35	(250 [a],250 [b],5000 [c])
NET1b	500	2500	2	[1–50]	[1–10]	25	35	(250,250,5000)
NET1c	500	2500	3	[1–10]	[1–10]	50	35	(250,250,5000)
NET2a	1000	5000	1	[1–10]	[1000–10000]	[5–25]	15	(500,500,5000)
NET2b	1000	5000	2	[1–50]	[1–10]	25	15	(500,500,5000)
NET3a	200	1300	3	[5–10] or 1	[1–100]	0	[100–300]	(1,1,10000)
NET3b	200	1300	3	[5–10] or 0	[1–100]	0	[100–300]	(1,1,10000)
NET4a	400	4500	3	[5–10] or 1	[1–100]	0	[100–300]	(1,1,10000)
NET4b	400	4500	3	[5–10] or 0	[1–100]	0	[100–300]	(1,1,10000)
GRI1a	100	3000	1	[1–50]	20	1	[3–5]	(1,1,100)
GRI1b	100	1000	1	[1–10]	20	1	[1–3]	(1,1,100)
GRI2a	100	3000	2	1	0.05	0	1	(1,1,6000)
GRI2b	100	1000	2	1	0.05	0	1	(1,1,2000)
GRI3a	100	3000	4	1	1	1	1	(1,1,6000)
GRI3b	100	1000	4	1	1	1	1	(1,1,2000)
							MAXFLOW	
AUT1	42	440	4	1	2	1	425	(1,1,300)
AUT2	102	2600	4	1	2	1	2514	(1,1,100)
AUT3	102	2600	4	1	0.5	1	2514	(1,1,2400)

[a] Number of sinks.
[b] Number of sources.
[c] Total supply.

Table 4
Description of the accuracy, parameters and percentage of active bounds at the solution.

Network	Accuracy	% links at lower bound	% links at upper bound	Network	Accuracy	% links at lower bound	% links at upper bound
NET1a	10^{-5}	79.48	0.40	GRI1a	10^{-3}	80.36	0.90
NET1b	10^{-3}	43.04	0.04	GRI1b	10^{-3}	12.40	1.10
NET1c	10^{-4}	55.40	0.00	GRI2a	10^{-2}	23.36	14.06
NET2a	10^{-4}	40.24	0.04	GRI2b	10^{-2}	23.22	13.30
NET2b	10^{-3}	43.48	0.06	GRI3a	10^{-2}	15.30	39.53
NET3a	10^{-4}	71.38	0.54	GRI3b	10^{-2}	9.42	23.35
NET3b	10^{-4}	85.84	0.77	AUT1	10^{-4}	0.00	15.26
NET4a	10^{-4}	83.69	0.09	AUT2	10^{-3}	0.00	0.00
NET4b	10^{-4}	93.26	0.13	AUT3	10^{-5}	0.00	73.88

Table 4 presents the relative objective accuracy used in the experiments for every network and the percentage of the links whose flow are at the bounds for an approximate solution to the problem.

2.1.2. *Network equilibrium model with combined modes [TAP-M]*

Fernández et al. [23] and García and Marín [24] present a network equilibrium model with combined trips. We have used this model as the second test problem. The main elements are a demand model and a transport network. The first one is a nested logit model, wherein at the first level the users choose between three modes of transport: (a) car, (b) public transport and (c) park'n'ride (a combined trip) which implies taking the first part of the trip by car, and the second by public transport. At the second level of the demand model the users of park'n'ride choose the transfer node where they change their mode of transport. The transport supply model assumes a road network and a public transport network. The users choose the route in each subnetwork according to Wardrop's first principle. In this model there is a simultaneous equilibrium for the demand, where no user has the incentive to unilaterally change the mode of transport or the transfer point chosen, and another where no users have the unilateral incentive of changing the route used.

The equilibrium conditions for the assignment is obtained as the solution to the following differentiable convex problem:

$$\text{minimize}\, Z(x,g) := \sum_{i \in I} \int_0^{x_i} c_i(s)\, ds + \sum_{\omega \in W} G_\omega^{-1}(g_\omega), \qquad (\text{TAP}-\text{M})$$

subject to

$$A_\omega g_\omega = d_\omega, \quad \omega \in W,$$

$$B_\omega h_\omega = g_\omega, \quad \omega \in W,$$

$$\sum_\omega C_\omega^i h_\omega = x_i, \quad i \in I,$$

$$h_\omega \geq 0, \quad \omega \in W,$$

where I is the set of the links of the multimodal network; $\omega = (i,j)$ is a pair of demand between an origin i and a destination j; W is the set of all O–D pairs; x_i denotes the flow in the link $i \in I$; h_ω is the flow in the paths connecting O–D pair ω; (d_ω) is the origin–destination matrix of trips in all alternatives; (g_ω) is the number of users of the demand ω that travel by the considered modes; $c_i(x_i)$ is the unit travel cost in link $i \in I$ which is associated with the traffic flow v_i; $G_\omega^{-1}(x)$ is the inverted demand model (a nested logit model); and $A_\omega, B_\omega, C_\omega^i$ are incidence matrices.

The authors do not know of any previous instances of the model TAP-M in the literature. For this reason, test problems are defined from the basic traffic assignment problems: NgD (see Nguyen and Dupuis [25]), Sioux Falls (see LeBlanc et al. [26]) and Hull (see Florian [27]).

Table 5
Traffic network models with combined modes.

Problem	$\lvert\mathcal{N}\rvert$	$\lvert I\rvert$	$\lvert W\rvert$	# Centroids	# Demand variables	# Variables
NgD2	26	45	4	4	20	65
SiF2	48	224	528	24	3263	3487
Hul2	1002	1678	142	23	757	2435

Table 6
Parameters for the test networks.

Problem name	β_1	β_2	θ_a	θ_b	τ_ω	α_t^c	α^k
NgD2	2.00	4.00	1.0	1.0	1.0	1.0	1.0
SiF2	1.00	1.20	1.0	1.0	1.0	1.0	1.0
Hul2	1.00	1.50	1.0	1.0	1.0	1.0	1.0

The topology of the network is obtained by duplicating the basic network. Two new kinds of links are added. The first one joins centroids with their duplications. The direction of the link depends on whether it is an origin centroid or a destination centroid. The adjacent nodes from the origin centroids are the transfer nodes, and links that join these nodes to their duplications are added.

The test problems are small-size networks, data for which are given in Table 5. Here, $|\mathcal{N}|$ is the number of nodes, $|I|$ is the number of links and $|W|$ is the number of origin–destination pairs. The number of variables is the sum of the number of demand variables plus the number of flow link variables.

The main motivation for choosing small-size basic networks is that the duplications still define large-size networks. The total number of variables is approximately six times the number of O–D pairs plus twice the number of links in the original network.

We have duplicated the original demand. The modal split depends on the logit parameters and the cost of the journey in each alternative. The parameters of the test networks for TAP-M are presented in Table 6. We have chosen the parameters of the nested logit model, so that the users' choices of route, mode of transport and transfer node only depend on the cost of each alternative of making a journey. That is, if two alternatives for mode of transport, or in route or in used transfer node are different, but they have the same cost then they have the same disutility for the users. For this reason the parameters θ_a, θ_b, α^k, and α_t^c are equal to 1. For simplicity, we have assumed that the vehicle–occupancy rate is of one person per car, that is $\tau_\omega = 1$. The relevance of the cost in the choice of the mode of transport by the users is taken into account by means of the parameters β_1, β_2. We have chosen different parameters of β_1, β_2 for each test network.

All test problems employ the travel cost formula $c_i(x_i) = c_i^0 + (x_i/k_i)^{m_i}$, where c_i^0 is the free-flow travel time on link i, $m_i \geq 1$, and k_i is the *practical capacity* for the original links and their duplications. The link cost function is $c_i(f_i) = 0$ for the new links.

2.2. CG algorithms implemented

2.2.1. CGP algorithms implemented

Table 7 summarizes the algorithms for the CGP that have been used in our test problems. We have used the following feasible descent methods for $\mathcal{A}_c$.

The column generation problem (CGP) for the NSD class of Larsson et al. is defined by means of applying n iterations of a descent and closed algorithm A over an auxiliary problem of the form

$$\min_{y \in X} \Pi(y,x), \qquad \text{(CGP(II, X, x))}$$

where $\Pi(\cdot,x) : X \mapsto \mathbb{R}$ is an approximation of f at the point x. The combinations of both choices, the algorithm and the approximation problem, define the method $\mathcal{A}_c = (\Pi,A)$. In NSD, only one iteration of $\mathcal{A}_c$ is applied.

The objective function for TAP-M is of the separable form $Z(x,g) = f(x)+G(g)$, then an approximation function may be defined by

$$\Pi_{NSD}(y,q,x,g) := \Pi(x,y)+G(q).$$

For the multi-commodity flow problems considered, quadratic CGP problems are favourably solved with truncated versions of the Frank–Wolfe or Evans algorithms. The efficiency of these algorithms is based on the efficiency to solve their linear subproblems. When they are applied to the *original* problem, then the linear subproblem is essentially a shortest path problem, wherein all costs are *non-negative*. (This fact stems from the cost on each link being the derivate of the cost function of the link for the actual level of service. The link travel cost functions are increasing for every link, so its derivate is non-negative.) By the non-negativity of the link costs, these subproblems can be solved by Dijkstra's algorithm. If these algorithms are applied to quadratic subproblems, this property is lost however.

We have considered a modification of the quadratic subproblem of CGP in order to avoid this disadvantage, which could otherwise result in negative cycles occurring. To explain the modification in short, we consider a multi-commodity nonlinear network problem, and we assume that the objective function is separable, that is, of the form $f(x) : = \sum_{i \in I} f_i(x_i)$, where f_i is a function of one variable. The quadratic approximation of f at a point x is $\Pi(y,x) = f(x) + \nabla f(x)^{\mathrm{T}}(y-x) + 1/2(y-x)H(x)(y-x)$ where $H(x)$ is a symmetric and positive definite matrix. This function is also separable and is expressed as $\Pi(y,x) = \sum_{i \in I} \Pi_i(y_i,x_i)$ where

$$\Pi_i(y_i,x_i) = f_i(x_i) + f'_i(x_i)(y_i - x_i) + 1/2 H_i(x_i)(y_i - x_i)^2,$$

where $H_i(x_i) = f''_i(x_i)$ holds for Newton's method, and $H_i(x_i) = \gamma > 0$ for the gradient projection method.

We denote by $T_i(y_i,x_i)$ the equation of the tangent straight line of the curve $\Pi(y_i,x_i)$ at a point ω for which $f_i(0) = T_i(0,x_i)$ holds. We have replaced the approximation $\Pi_i(y_i,x_i)$ with

$$\hat{\Pi}_i(y_i,x_i) := \begin{cases} \Pi_i(y_i,x_i) & \text{if } y_i \geq w, \\ T_i(y_i,x_i) & \text{otherwise,} \end{cases}$$

where w is the abscise of the tangent point. This point is given by

$$w(x_i) = \sqrt{\frac{2(f_i(x_i) - f'_i(x_i)x_i - f_i(0))}{H_i(x_i)} + x_i^2}.$$

This function is convex, differentiable and increasing over $\mathbb{R}$. If the symbol $\hat{}$ is placed over the name of an approximation, it indicates that this modification is used. Here, it applies to two methods, denoted by $\hat{\mathrm{N}}$E and GL$\hat{\mathrm{P}}$E.

To fully define these algorithms we must state the number of iterations performed in each iteration, and we must also choose the value of the parameter n_c^t. For the algorithms N, GLP, $\hat{\mathrm{N}}$E and GL$\hat{\mathrm{P}}$E we have used the parameter $n_c^t = 1$ for every iteration t. They lead to CG algorithms belonging to the NSD class. For the algorithms FW, E, RSD($\tilde{r}$), we have used sequences $\{n_c^t\}$ defined in two ways. The first one takes a fixed value of $n_c^t = n_c \geq 1$ in all iterations t, and the second one is defined by an adaptive tool for choosing the value of n_c^t (see García [31]), which has been devised to allow the method to choose the best trade-off between

Table 7
Algorithms used in CGP.

Problem	$\mathcal{A}_c$	Algorithm	Definition
SNFP	RSD($\tilde{r}$)	Restricted simplicial decomposition	Hearn et al. [7]
	FW/RSD(1)	Frank–Wolfe Algorithm	Frank and Wolfe [28]
	SD/RSD(∞)	Simplicial decomposition method	Holloway [4] and von Hohenbalken [5]
	N(n)	Truncated Newton method	$\Pi_N(y,x) := f(x)^T y + (1/2) y^T \nabla^2 f(x) y,$ $A :=$ n iterations of FW
	GLP(n)	Gradient projection	Goldstein [29] and Levitin and Polyak [30] $\Pi_{GLP}(y,x) := \nabla f(x)^T y + (\gamma/2) y^T y, \gamma > 0$ $A :=$ n iterations of FW
TAP-M	FW	Frank–Wolfe Algorithm	Frank and Wolfe [28] and García [31]
	E	Evans method	Evans [32] and García [31]
	$\hat{\mathrm{N}}$E	A modified Newton–Evans method	$\Pi_{NE}(y,x) := \hat{\Pi}_N(y,x) + G(q)$ $A :=$ n iterations of FW
	GL$\hat{\mathrm{P}}$E	A modified GLP–Evans method	$\Pi_{GLPE}(y,x) := \hat{\Pi}_{GLP}(y,x) + G(q)$ $A :=$ n iterations of FW

the generation of columns of high quality and the resulting computational efficiency.

2.2.2. *Description of the RMP*

The second main contribution of the CG algorithms to the state-of-the-art in solving the problems considered is the generality by which we may define the feasible set of the RMP. We have used the set augmentation rule given in Larsson et al. [9] to define X^t, and which is given in Step 3.2 in Table 2. Note that to fully define the set augmentation rule, we must also specify the parameter r.

We have throughout used the projected Newton method of Bertsekas [33] to solve the RMPs. This algorithm has a superlinear convergence rate, and it is advantageous especially because of the simple constraint structure of RMP.

We have not considered other algorithms because in this phase of CG the key is the convergence speed of $\mathcal{A}_r$. A sublinear convergence rate could make the CG algorithm inefficient, and a different algorithm with a superlinear convergence rate would have similar behavior, and would perhaps only change the amount of time spent in the RMP.

The truncation of the RMP is monitored by the number of iterations (number of projections) used with the algorithm $\mathcal{A}_r$; this number is denoted by n_r^t. In all iterations we have used the same number of iterations, and this number is denoted by n_r.

2.2.3. *Notation*

We have introduced the following notation to refer to a specific CG algorithm:

$$(\{\mathcal{A}_r^{k_t}\}_r^{n_r^t}, \{\mathcal{A}_c^{k'_t}\}^{n_c^t}),$$

where, at the main iteration t, $\mathcal{A}_c^{k'_t}$ describes the algorithm used in the CGP, n_c^t is the number of iterations that is performed by using the algorithm $\mathcal{A}_c^{k'_t}$ on this problem, $\mathcal{A}_r^{k_t}$ describes the algorithm used in the RMP, n_r^t is the number of iterations performed on the RMP using this method, and r is the parameter used in the RMP. We can shorten this notation, because in every test problem only one algorithm is used for the RMP, as was remarked above, and only one algorithm $\mathcal{A}_c^{k'_t}$ is used in each experiment. Therefore, the title of $\mathcal{A}_r$ is implicitly defined and the index k'_t can be omitted. The following notation hence is sufficient to specify an algorithm:

$$\{\mathcal{A}_c\}_r^{n_r^t, n_c^t}.$$

2.3. *Implementational details*

2.3.1. *The single-commodity network flow problem [SNFP]*

Table 8 presents the parameter γ used in the GLP algorithm for each subproblem. This parameter has been calibrated by means of performing five trials, and taking the best value.

The implementation of the projected Newton method of Bertsekas [33] to solve the RMP is identical to that used by RSDNET of Hearn et al. [34]. This algorithm includes an Armijo-type line search and two parameters σ and β. We have used the values of 1.0D–4 and 0.5, respectively, which are adequate in most applied problems (see Bertsekas [33]). The number of iterations n_r^t has been variable in the computational experiments, but this number is fixed in all the iterations of the same experiment.

All codes were compiled in FORTRAN and run on a PC with 64 megabytes of RAM and a 200 MHz CPU. All computations are done in double precision. All the considered algorithms use as a subroutine the RSD algorithm. We have used the code RSDNET of Hearn et al. [34] to implement this method. This code uses a primal simplex algorithm for networks, the NETFLO code, which is implemented as in Kennington and Helgason [35] to solve the linear minimum cost flow problem.

Table 8
Values of the parameter γ used in every problem.

Network	γ
NET1a	3
NET1b	75
NET1c	10
NET2a	2000
NET2b	50
NET3a	50
NET3b	15
NET4a	50
NET4b	25
GRI1a	100
GRI1b	50
GRI2a	100
GRI2b	25
GRI3a	100
GRI3b	25
AUT1	75
AUT2	50
AUT3	200

2.3.2. *The combined traffic assignment problem [TAP-M]*

The adaptation of the Frank–Wolfe and Evans algorithms for TAP-M is given in García and Marín [31,36]. We have used the algorithm L2QUE of Gallo and Pallotino [37] to solve the minimum path problem in order to obtain the direction search. A main iteration of these algorithms is completed by means of the solution to a one-dimensional minimization problem, which determines the optimal step size that minimizes the value of the objective function, given the current solution and the descent direction obtained in the CGP. The unidimensional search is made by using the Golden Section Method.

The same Frank–Wolfe subroutine is used in order to generate the column in the CGP, and as a truncated solver for the quadratic approximation of the Newton method.

The RMPs for TAP-M are also solved by the projected Newton method of Bertsekas [33]. This algorithm has been coded as for the SNFP.

Program sources are written in the FORTRAN Visual Workbench programming language and double precision has been employed in the representation and in all the operations where roundoff error might arise or when accuracy is important for algorithm behavior. Whenever roundoff errors might not affect performance, single precision is employed. For example, single precision is used in the shortest path tree computations. Double precision is employed in operations involved in the master problem resolution, because projections are very sensitive to roundoff errors. The codes are run on a PC with 384 megabytes of RAM and 400 MHz.

3. Numerical experiments

We have designed three computational experiments to investigate the behavior of the CG algorithms. The first one has the objective of studying the influence of some factors on the performance of CG algorithms. The second one has the objective of studying the relevance of prolonging the columns obtained in the CGP to the relative boundary. The last experiment is designed to compare the new CG algorithms with classic ones.

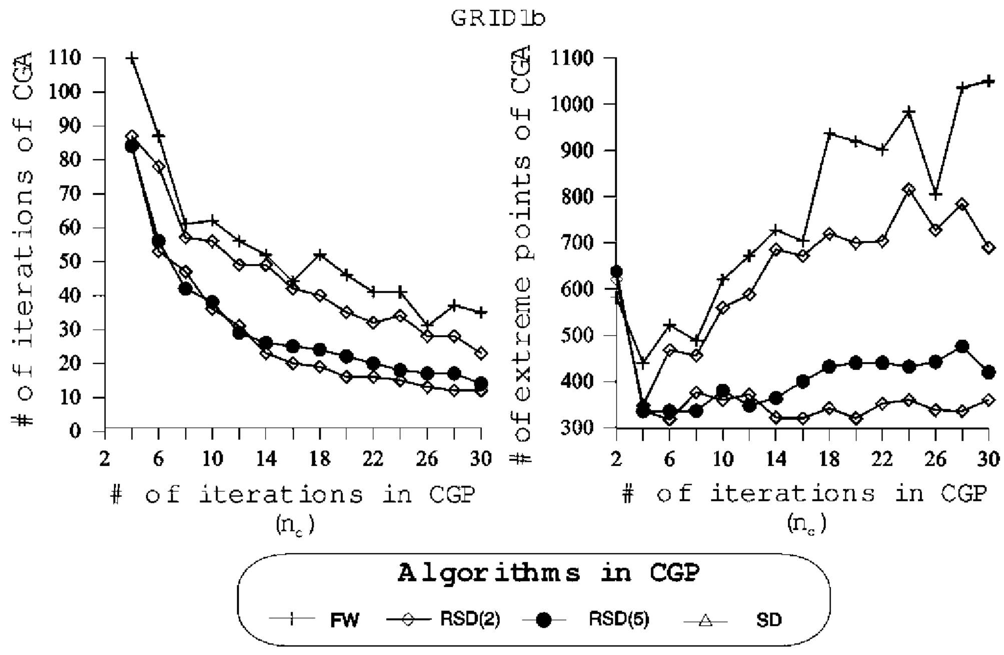

Fig. 2. Number of main iterations of CG algorithm and number of extreme points versus n_c.

3.1. A study of some factors determining the performance of CG algorithms

Experiment 1—*A study of the quality of the columns versus the performance of CG algorithms*: The CG algorithms have two means of improving the quality of the columns generated in an CGP. The first is by means of increasing the number of times that the algorithm $\mathcal{A}_c$ is applied to CGP, that is, increasing the number n_c, and the second is by choosing a more efficient algorithm to represent $\mathcal{A}_c$.

In our first experiment, we solve the network GRI1b with the algorithms RSD $(\tilde{r})_{\infty}^{n_r,n_c}$ for several combinations of the parameters n_c and $\tilde{r}$. The value of n_r is here fixed to 10. We have considered the following CGP algorithms: Frank–Wolfe ($\tilde{r}=1$), RSD ($\tilde{r}=2$ and $\tilde{r}=5$) and SD ($\tilde{r}=\infty$), as procedures to generate the columns.

Fig. 2 shows the number of main iterations and the total number of extreme points generated by the CG algorithm as a function of the number of iterations done in the CGP for GRI1b. Note that the number of extreme points generated by the CG algorithm is computed as the number of main iterations multiplied by n_c.

The conclusion drawn from this experiment is that if the quality of the columns is improved by means of increasing the value of the parameter n_c or by choosing a better algorithm $\mathcal{A}_c$ (in our example this is obtained by increasing the value of $\tilde{r}$), then the number of RMPs and CGPs required to obtain a given accuracy is reduced. On the other hand, the computational cost in the CGP is increased or decreased according to the trade-off between a reduction of the number of CGP and an increase of the total number of inner iterations (that is n_c) to obtain the column (compare in Fig. 2 the values of $\tilde{r}=5$ and ∞ with respect to $\tilde{r}=1$ and 2).

Our experiment has the objective of evaluating the effect of the quality of the columns on the CPU time of a CG algorithm. An improvement in the quality of the columns always reduces the CPU time spent in the RMPs because a smaller number of RMPs is carried out, and they also have fewer variables, since in general the number of variables of the RMP is augmented by one in each iteration. Furthermore, the CPU time spent in the CGPs is proportional to the total number of iterations carried out by using the CG algorithm multiplied by the CPU time for the generation of one column.[1] An improvement in the quality of the columns reduces the number of iterations of the CG algorithm but it increases the CPU time spent in performing each iteration. This may indicate that an improvement of the quality of the columns could reduce the CPU time spent in CGP, but this it is not guaranteed.

The CPU time of a CG method is the sum of the CPU time for solving the CGPs plus the CPU time for solving the RMPs. The first is roughly proportional to the number of iterations of the CG algorithm, but the second is proportional to the square of the number of main iterations. For this reason, a small increase in the quality of the columns will reduce the overall computational cost, especially for problems with high dimension, dimF^*, of the optimal face, and which require greater accuracy, but if this increment is too large, the total CPU time begins to grow, because the CPU time for solving the CGPs increases and it is not compensated by the saving of CPU time in the RMPs.

The experiment consists of solving the test networks NET1a, and AUT1 using the algorithm RSD $(\tilde{r})_{\infty}^{n_r,n_c}$ for several values of the parameters n_c and $\tilde{r}$, and computing the total CPU time spent. The values of the parameter n_r used for each network are 8 and 15, respectively.

Fig. 3 shows the results obtained. The behavior of the CG algorithm agrees with the above discussion. The range of the optimal values for the parameter n_c is in the interval [5–10]. This illustrates the potential improvement of the CG class over the original simplicial decomposition method (which corresponds to $n_c = 1$, and which is too slow to even be included in the figure).

In the NET1a example (the picture to the left) we can observe that the best algorithm to use in the CGP is always defined by the choice of $\tilde{r}=1$, that is, the Frank–Wolfe algorithm, while in the AUT1 example (the picture to the right) the best choice corresponds to $\tilde{r}=\infty$, that is, the simplicial decomposition

[1] Assuming that the value of n_c^t is constant the complexity of each iteration of $\mathcal{A}_c$ algorithm is approximately the same throughout the procedure.

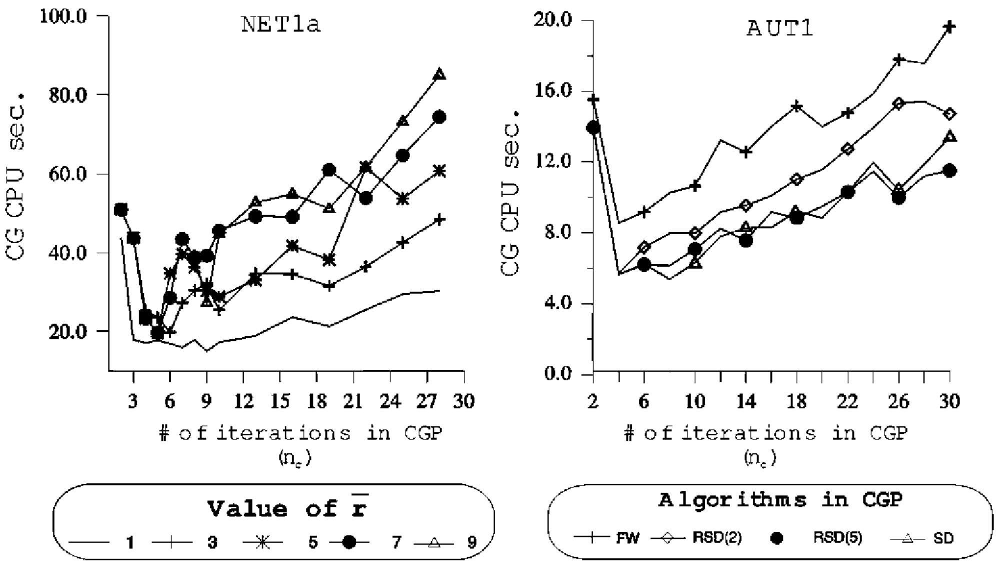

Fig. 3. Algorithm $\text{RSD}(\bar{r})_{\infty}^{n_r,n_c}$ applied to NET1a and AUT1. CPU time versus value of n_c.

algorithm. (For NET1a, an algorithm such as RSD($\bar{r}$) with $\bar{r} > 1$ is always worse.) This perhaps surprising result is explained by the fact that while the columns generated by the algorithm RSD($\bar{r}$) have a better quality then the columns produced by the Frank–Wolfe algorithm, the quality is not good enough to significantly reduce the number of main iterations of the CG algorithm. As the mean CPU time used in one iteration of the Frank–Wolfe algorithm is less than that of RSD($\bar{r}$), this implies that the total CPU time is also lower. For AUT1, this trade-off is, however, reversed.

The dimension of the optimal face of AUT1 is dim$F^* = 399$, while for NET1a it is not possible to compute exactly. (The dimension of the optimal face is upper bounded by the number of links with a positive flow, that is, by 513. Moreover, all feasible solutions have at least 250 links with a positive flow, thus dimF^* is upper bounded by 263.) The linear terms of NET1a dominate the nonlinear terms; for this reason the linear approximation of the objective functions is very accurate. On the other hand, AUT1 is significantly nonlinear at the optimal solution. This explains why for NET1a it is optimal to generate the columns using the Frank–Wolfe algorithm but that this is not so for AUT1.

We have carried out additional tests where we initialize the RMPs of the algorithm RSD($\bar{r}$) with the extreme points retained in the previous CGP. This is, however, a bad choice compared with an empty set of extreme points as the initialization for this test problem. This type of initialization could, however, be of interest when the computational cost to obtain the extreme points is much higher.

Experiment 2—*A study of the effect of accuracy required on the performance of CG algorithms*: According to the arguments given earlier nonlinear versions of the CG algorithm should be better than an (R)SD algorithm for finding highly accurate solutions. In this experimented we investigate this issue numerically.

This experiment consists of solving SiF2 with the algorithms $\text{E}_r^{n_r,n_c}$ for the relative accuracies of 1.0D+00,1.0D–01,1.0D–02,1.0D–03,1.0D–04 and several combinations of n_c and n_r. There exist interactions between the parameter n_c and the parameters n_r and r used in the RMPs. Computationally, it is very intensive to study all combinations. We have therefore considered three small and three large values of the parameter n_c (2, 3, and 4, and 10, 20, and 30, respectively). We have chosen $r = \infty$ as being the best choice and we have dealt with two values of the parameter n_r. The first one is $n_r = 100$, which means that the RMP is solved almost exactly, and the other one is $n_r = 10$, which is (in general) much less accurate.

When applied to the combined traffic model TAP-M, a natural improvement over SD is to replace its linear CGP by an only partially linearized subproblem, as in Evans' algorithm. We denote this choice by $\text{E}_r^{n_r,1}$. We have compared the algorithm $\text{E}_r^{n_r,1}$ with algorithms for which $n_c > 1$ by means of the CPU time ratio

$$\text{CPU time ratio} = \frac{\text{CPU time spent by } \text{E}_r^{n_r,1}}{\text{CPU spent time by } \text{E}_r^{n_r,n_c}},$$

as a function of n_c and the level of accuracy of the solution of CDP(f,X). This rate measures the relative improvement of a CG scheme with respect to a modification of the original scheme of SD.

Fig. 4 displays the results obtained from this experiment. It is shown that the optimal quality of the columns generated depends on the accuracy that we wish to achieve. For a low (respectively, high) degree of accuracy a small (respectively, large) value of n_c is advisable. These results confirm that the original scheme ${}_{\infty}^{n_r,1}$ is improved when the demanded accuracy is higher.

Experiment 3—*A study of the RMPs on the efficiency of the CG algorithm*: The parameters n_r and r determine the number of main iterations of the CG algorithm and the complexity of the RMPs, respectively. The value of n_r monitors the accuracy of the RMPs and r their sizes. With smaller values of r the total number of RMPs is larger, but they are easily solvable, and the relative importance of the parameter n_r is small. On the other hand, when r is larger the number of iterations of the CG method is decreased, but the computational time spent on each RMP is larger. Our next experiment investigates this trade-off.

Experiment 1 shows that the CG methods with $n_c > 1$ solve less RMPs which hence have less variables than those in the (R)SD method. A consequence of this is that it may not be necessary to introduce a strategy to keep the size of the RMP in such CG methods small. To investigate this reasonable assumption we have considered the evolution of the CPU time spent for the

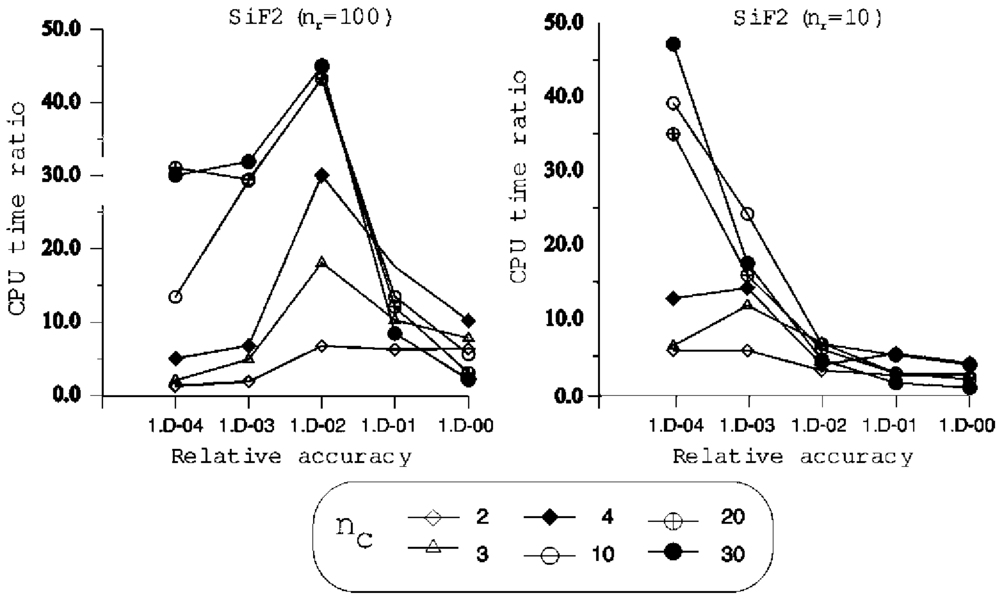

Fig. 4. Ratio between the CPU time spent by E_{∞}^{100,n_c} and $E_{\infty}^{n_r,1}$ versus the relative accuracy for $n_r = 100$ and 10.

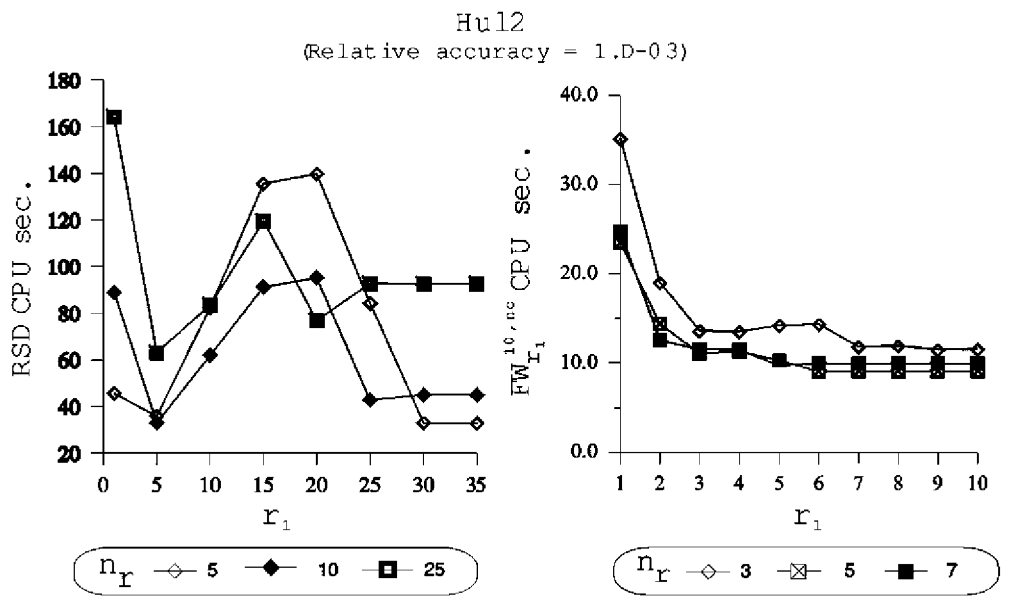

Fig. 5. RSD and FW_r^{10,n_c} CPU time versus the parameter r.

algorithms $FW_r^{n_r,1}$ (which is the original RSD) and FW_r^{10,n_c} as a function of the parameter r. This experiment consists of solving the test problem Hul2 for a relative accuracy of 1.0D–03 by means of these algorithms and for several values of n_c, n_r and r. In the $FW_r^{n_r,1}$ algorithm, we have used three (eight) values of n_r (r), and correspondingly, three (eight) values of n_c (r) in the FW_r^{10,n_c} algorithm.

The results of this experiment are shown in Fig. 5. The conclusion of this experiment is that the best choice for this example is, not surprisingly, $r = \infty$ (a simplicial decomposition scheme) for high quality columns (the picture on the right). On the other hand, the algorithm $FW_r^{n_r,1}$ is sensitive to the parameters r and n_r. In this case the better performance is achieved for a finite value of the parameter r (a restricted simplicial decomposition method). Moreover, the performance of the RSD algorithm depends largely on the accuracy of the RMPs, that is, on the value of n_r.

3.2. On the prolongation to the relative boundary

This subsection deals with the role of the prolongation to the relative boundary given in (2) played in the performance of the CG algorithm.

Experience with the RSD method has shown that it initially makes rapid progress, quickly reaching a near-optimal solution, especially when relatively large values of r are used and when second-order methods are used for the solution of the RMP; experience also shows that it slows down close to an optimal solution.

If $r \geq \dim F^* + 1$, where F^* is the optimal face, then the number of RMP is finite, and the local rate of convergence is governed by the local convergence rate of the method chosen for the solution of the RMP; thus, a superlinear or quadratic convergence rate may be attained. If $r < \dim F^* + 1$, then the algorithm is only asymptotically convergent, and the rate of convergence is the same as that of the *Frank–Wolfe* algorithm, that is, the convergence rate is sublinear. In this work we formulate the following conjecture based on experimental results:

If one does not prolong the columns to the relative boundary, then the convergence rate of the CG algorithm is governed by that of the algorithm used in CGP $\mathcal{A}_c$. Otherwise, and if $r \geq \dim(F^*)+1$, then the convergence rate is governed by that of the algorithm $\mathcal{A}_r$.

This result generalizes that for RSD to the more general CG algorithm. It postulates that it is also true for the CG algorithm under a new assumption: the realization of the prolongation of the columns to the relative boundary. When RSD is used the columns are extreme points and the prolongation is therefore unnecessary.

A main difference between SNFP and TAP-M is that for the single-commodity flow problem SNFP it is possible to easily compute the prolongation based on the flow vector of the column. The TAP-M problem is a multi-commodity flow problem, where this is more complicated; the reason is that the columns stored are aggregated (that is, stored in terms of the total link flows), whereas the prolongation must be calculated taking disaggregated, commodity link flow, information into account. More comments on the latter is made below.

We have carried out two numerical experiments. Experiment 4 is designed to show the role of the prolongation to the relative boundary in the performance of the CG methods. Experiment 5 is designed to numerically compute the speed of convergence of CG methods.

Experiment 4—*On the effect of a prolongation to the relative boundary*: This experiment consists of solving NET2b with the algorithms FW, $N_\infty^{10,1}$, $GLP_\infty^{10,1}$ and $FW_\infty^{10,10}$ with and without a prolongation of the columns to the relative boundary. The quadratic subproblems of NSD have been solved using 10 iterations of the Frank–Wolfe algorithm in both trials.

The computational results are shown in Fig. 6. The efficiency of the algorithms $N_\infty^{10,1}$ and $FW_\infty^{10,10}$ are dramatically reduced when the columns generated are not prolonged to the relative boundary. Moreover, the convergence speed of $FW_\infty^{10,10}$ and $N_\infty^{10,1}$ algorithms then are to that of the Frank–Wolfe algorithm. The method $GLP_\infty^{10,1}$ has similar behavior in both approaches.

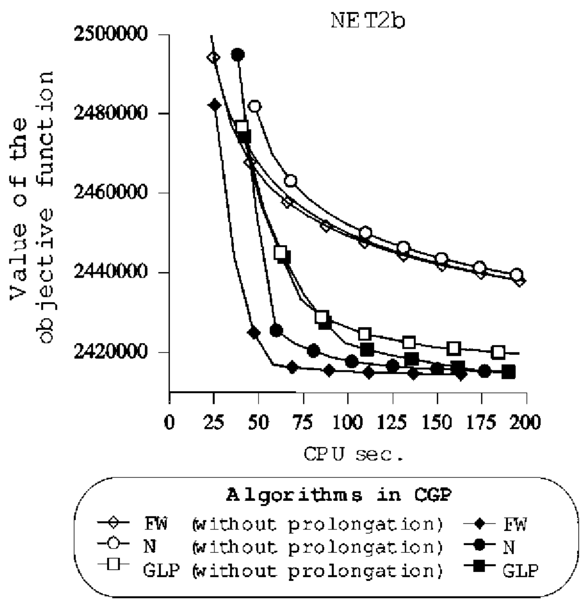

Fig. 6. Influence on efficiency of the prolongation to the relative boundary.

We have observed that if we do not prolong the columns to the relative boundary, then an integer τ exists, so that

$$x^t = \hat{y}^{t-1} \quad \text{for all } t \geq \tau.$$

For the case of the NSD algorithm where the CGP is based on a quadratic approximation (that is, a method similar to $N_\infty^{10,1}$), this was also observed by Larsson et al. [9]. This means that eventually the RMPs in fact reduce to line searches, and the CG method here reduces to a pure Newton method. An interesting conclusion is that by prolonging the columns to the relative boundary, the RMPs become non-trivial, with the result of an improvement over Newton's method which can be quite substantial.

We provide a formula for this computation when X is a closed and convex, and a feasible descent direction method, $\mathcal{A}_c$, is used for the CGP. Within this algorithm we assume that the line search has the form

$$\min_{\lambda \in [0,1]} f(\lambda x + (1-\lambda)p),$$

where x is the current iteration point and p is an extreme point of X. Then, the prolongation to the relative boundary of the resulting vector is given by

$$\ell_t = \frac{1}{1-\prod_{i=1}^{n_c^t} \lambda_i}, \tag{3}$$

where λ_i for $i=1,\ldots,n_c^t$ are the step lengths provided by the line searches at iteration t in CGP.

Experiment 5—*Speed of convergence of the CG methods*: The following experiment is designed to evaluate the speed of convergence of CG methods with and without using a prolongation of the columns to the relative boundary. First, we have computed a solution to the problem Hul2 with a relative objective accuracy of 5.8×10^{-6}. This solution is denoted by $\hat{x}$. We have then calculated the sequence

$$\text{fAbsErr}_t := f(x^t) - f(\hat{x}),$$

which is an estimation of the absolute error of the objective function.

We have solved Hul2 with the methods $FW_\infty^{10,5}$ and $E_\infty^{10,5}$, with and without prolongation to the relative boundary of the columns generated. We have computed a fit of the sequences fAbsErr_t of the type $\text{fAbsErr}_t = \alpha/t^\beta$, and the results are presented in Table 9. We have estimated the parameters α and β by using the least square method. The goodness-of-fit has been evaluated by means of R^2, a coefficient of determination, that gives the proportion of the variance of fAbsErr explained by the regression. (These values are very close to 1.)

The main conclusion is that if the prolongation is not used then the rate of convergence of the methods is similar to that of the algorithm used in CGP. That is, $FW_\infty^{10,5}$ is similar to performing five iterations of the Frank–Wolfe algorithm, and $E_\infty^{10,5}$ to performing five iterations of Evans' algorithm. Note that if the prolongation is used $\beta > 1$, otherwise $\beta < 1$.

Table 9
Fit $\text{fAbsErr}_t = \alpha/t^\beta$ of methods.

Algorithm	With prolongation			Without prolongation		
	α	β	R^2	α	β	R^2
$FW_\infty^{10,5}$	2799.83	1.7468	0.9676	2013.24	0.7112	0.985
$E_\infty^{10,5}$	2726.18	2.1362	0.9978	886.305	0.8622	0.9971
FW	2271.94	0.5469	0.9914			
E	3746.75	0.8718	0.9943			

3.3. Further comparisons between CG methods

We can say that the class of CG algorithms generalizes methods of feasible descent directions by means of their use in CGP, and obtain them from a CG method by choosing $r = 1$ and $n_c = 1$. We have investigated three ways to improve feasible direction methods by using its algorithm description $\mathcal{A}_c$ within a CG method:

I: Generalized PARTAN ($n_c \geq 2$ and $r = 1$). This extension consists of making n_c line searches within the algorithm $\mathcal{A}_c$, thus making a new line search on the direction defined by the column and the current iterates of RMP. These methods are new except for the case $n_c = 2$, which includes the method of *parallel tangents* (PARTAN) credited to Shah et al. [38].

II: Multidimensional searches ($n_c = 1$ and $r \geq 2$). The choice of $n_c = 1$ and $r \geq 2$ is a means to generalize line search methods to multidimensional searches on simplices (when RMP is solved exactly). Examples of these are the NSD algorithms of Larsson et al. [9].

III: Multidimensional searches ($n_c \geq 2$ and $r \geq 2$). This type of algorithm is new, and could be said to properly define the class of CG algorithms.

This experiment has been designed to compare some algorithms of feasible directions with their extensions of type I, II, and III. To this end we have solved the SNFP and TAP-M problems using the algorithms presented in Table 10. Some of these methods have lower bounds available for the optimal value of the problem. Differences exist between their quality. To eliminate this factor in the measure of the performance of the methods we use the same lower bound for all of them. The accuracy used for the SNFP problems are presented in Table 4; two degrees of accuracy (10^{-3} and 10^{-4}) for the TAP-M problems have been used.

We have always used $n_r = 8$ projections to solve the RMPs for the problems of the NET type, and $n_r = 25$ for the other types of SNFP problems. We have used the adaptive tool developed in Garzía [31] to dynamically choose the parameter n_c and n for all the algorithms used in SNFP problems. On the other hand, we have used a fixed value of n_c for TAP-M problems. We have used the value of $n_c = 15$ for Sif2 and the value of $n_c = 5$ for the problems: Hul2 and NgD2. We have here always used $n_r = 10$. The quadratic subproblems that appear in $\hat{\mathrm{NE}}_{\infty}^{n_r,1}$ and $\hat{\mathrm{NE}}$ were solved by means of the Frank–Wolfe algorithm and the number of iterations was then equal to n_c.

The results obtained for SNFP problems are presented in Table 11 and those for the TAP-M problems in Table 12. Table 11 shows the CPU times. Table 12 presents in the first block the method of feasible directions and in the second one their generalizations to searches on simplices. The report for this experiment consists of the CPU times.

We analyze below the results derived from Tables 11 and 12.

I: It is not really possible to compare the "crude" FW algorithm with its extensions FW_1^{8,n_c^t} or $\mathrm{FW}_1^{n_r,n_c}$ because the CPU time spent by FW to solve the test problems is prohibitive. If we compare the latter two with the SD methods for NET-SNFP problems and with the RSD method $\mathrm{FW}_{100}^{10,1}$ for the TAP-M problems, then the harmonic mean of the ratios between CPU times is 17.2 and 1.5, respectively.
On the other hand, it is possible to compare the algorithm $\mathrm{E}_1^{n_r,n_c}$ with its original algorithm E. The harmonic mean of the ratios of CPU times is 3.4 for the accuracy 10^{-3} and 5.6 for the accuracy 10^{-4}. Note that $\mathrm{E}_1^{n_r,n_c}$ also improves upon the simplicial decomposition with Evans' subproblems, that is, the method $\mathrm{E}_{100}^{10,1}$. This mean rate of improvement is 1.9 for the accuracy 10^{-3} and 1.2 for the accuracy 10^{-4}. The SD method is satisfactory for small or medium accuracy, but highly computationally intensive when a large accuracy is demanded. The CG methods are much better here to reach high accuracies.

II: We have computed the harmonic mean of the ratios between the unidimensional methods and their NSD extension. We

Table 10
Algorithms.

Type	SNFP	TAP-M
I	FW_1^{8,n_c^t}	$\mathrm{FW}_1^{n_r,n_c}$, $\mathrm{E}_1^{n_r,n_c}$
II	SD, $\mathrm{N}_{\infty}^{8,1}$, $\mathrm{GLP}_{\infty}^{8,1}$	$\mathrm{FW}_{100}^{n_r,1}$, $\mathrm{E}_{100}^{n_r,1}$, $\hat{\mathrm{NE}}_{\infty}^{n_r,1}$
III	$\mathrm{FW}_{\infty}^{8,n_c^t}$	$\mathrm{FW}_{\infty}^{n_r,n_c}$, $\mathrm{E}_{\infty}^{n_r,n_c}$
$\mathcal{A}_c$	N, GLP,	FW, E $\hat{\mathrm{NE}}$

Table 11
CPU times for the SNFP problems.

Network	SD	FW_1^{8,n_c^t}	N	GLP	$\mathrm{FW}_{\infty}^{8,n_c^t}$	$\mathrm{N}_{\infty}^{8,n_c^t}$	$\mathrm{GLP}_{\infty}^{8,n_c^t}$
NET1a	260.9[a]	20.2	$\underline{7.5}$	33.6	18.7	10.5	20.2
NET1b	2056.3	585.8	607.1	1624.6	$\underline{37.1}$	56.4	104.2
NET1c	4686.1	544.1	=	663.0	$\underline{31.6}$	=	45.2
NET2a	> 11 000	1106.8	4003.7	1434.3	92.5	$\underline{80.5}$	111.9
NET2b	> 44 000	1685.5	2360.4	3552.3	$\underline{93.4}$	207.6	293.5
NET3a	6407.9	136.5	=	264.0	$\underline{21.2}$	=	24.9
NET3b	665.9	21.6	=	103.0	$\underline{13.3}$	=	52.9
NET4a	> 17 110	471.7	=	922.6	$\underline{98.5}$	=	110.5
NET4b	> 15 550	154.6	=	1442.6	$\underline{81.6}$	=	918.7
Network		FW_1^{25,n_c^t}	N	GLP	$\mathrm{FW}_{\infty}^{25,n_c^t}$	$\mathrm{N}_{\infty}^{25,n_c^t}$	$\mathrm{GLP}_{\infty}^{25,n_c^t}$
GRI2a		227.1	266.2	1604.8	102.6	126.2	$\underline{94.6}$
GRI2b		93.7	116.2	256.6	37.2	62.8	$\underline{35.4}$
GRI3a		76.6	$\underline{43.0}$	263.7	59.7	297.9	1668.0
AUT1		29.8	24.9	22.9	12.8	6.8	$\underline{6.6}$
AUT2		$\underline{45.0}$	92.3	47.1	59.5	81.8	71.3
AUT3		31.9	$\underline{18.7}$	24.3	33.3	20.3	27.5

Table 12
CPU times for the TAP-M problems.

Accuracy	Problem	FW	E	$\hat{\text{NE}}$	FW_1^{10,n_c}	E_1^{10,n_c}	$\text{FW}_{100}^{10,1}$	$\text{E}_{100}^{10,1}$	$\hat{\text{NE}}_\infty^{10,1}$	$\text{FW}_\infty^{10,n_c}$	E_∞^{10,n_c}
	NgD2	3.62[a]	1.37	0.33	0.60	0.28	0.33	0.22	$\underline{0.11}$	0.22	$\underline{0.11}$
$\mathbf{10^{-3}}$	SiF2	>72.5	49.8	310.3	313.1	11.3	>1616.9	56.2	34.9	13.4	$\underline{11.7}$
	Hul2	89.2	16.6	20.9	22.5	8.6	44.1	16.8	10.2	10.1	$\underline{6.2}$
	NgD2	144.56	39.32	15.22	24.44	6.59	0.60	1.04	$\underline{0.55}$	0.77	0.72
$\mathbf{10^{-4}}$	SiF2	?	394.9	?	>2346.5	48.5	?	222.3	1682.7	942.6	$\underline{18.9}$
	Hul2	?	639.1	990.3	1291.2	185.5	>30000	423.4	67.8	67.3	$\underline{25.9}$

have dealt with the algorithms N and GLP for the SNFP problems and $\hat{\text{NE}}$ for the TAP-M problems. These values for the type NET are 5.6 and 6.4 for N and GLP, respectively, and 1.1 and 1.8 for the GRID-AUT problems.

Within these classes of problems there exist networks where the multidimensional searches do not improve the method of feasible directions. This is due to two factors. The first one is that these examples have a solution with a great number of active bounds, see Table 4, which make the expansion of the simplex inefficient. The second factor is that the RMPs should be solved more exactly.

The algorithm $\hat{\text{NE}}$ is improved around a mean value of a factor 3.8 for the accuracy 10^{-3} and around 20.35 for the accuracy of 10^{-4}.

We conclude that CG methods improve upon the unidimensional search methods. Moreover, this improvement is larger for larger accuracies demanded.

III: The algorithms of type III, $\text{FW}_\infty^{8,n_c^t}$, $\text{FW}_\infty^{n_r,n_c}$, and $\text{E}_\infty^{n_r,n_c}$ have a quite satisfactory computational behavior for the NET and TAP-M problems. Table 11 presents that the speed-up over the SD algorithm is improved by the $\text{FW}_\infty^{8,n_c^t}$ algorithm by a mean factor of 113.2 for the NET problems. This factor is 10.3 for the TAP-M problems. If we compare $\text{FW}_{100}^{10,1}$ (restricted simplicial decomposition) with $\text{E}_\infty^{n_r,n_c}$ we obtain that this factor is 24.9. (In fact, we should compare the $\text{E}_\infty^{n_r,n_c}$ algorithm with $\text{E}_{100}^{10,1}$ and this factor is 4.3.)

We have also seen that the algorithms of type III generate the best lower bounds throughout the process.

4. Conclusions

In this paper two important aspects of column generation methods for large-scale nonlinear network flow problems have been studied computationally. The first is the validation of this methodology in order to improve the performance of feasible direction and simplicial decomposition methods used in equilibrium assignment models. The second is to study the role of the prolongation to the relative boundary and of the parameters n_c, r, n_r of CG methods determining the accuracy of computations.

When a high accuracy is required, then the size of the RMPs for the SD algorithm begins to grow and their CPU time to solve them becomes impractical. The CG algorithms maintain a small number of variables in RMP by means of improving the quality of the columns; this opens up a way to utilize algorithms with a quadratic rate of convergence for large-scale problems in a reasonable time of computation.

The improvements of the CG algorithms with respect to SD come from three sources: first, less RMPs are solved; second, on average, the RMPs are smaller and therefore less demanding computationally; and third, if we use the Frank–Wolfe algorithm then the number of extreme points generated by the CG algorithm may be less than those of the SD algorithm. The last factor depends on the accuracy demanded by the solution and the characteristics of dimF^*.

The computational experiments have also established the crucial role of the prolongation of the columns generated to the relative boundary. In the case that this prolongation is not used, the speed of convergence is monitored by the algorithm used in the CGP.

We stress finally that in the CG methods tested, with the exception of the column prolongation, the computations performed are of the same type as in the (R)SD methods, namely, linear network flow problems and multidimensional searches over simplices. The fact that the well-studied (R)SD methods can be substantially improved within the same framework of algorithms suggests, in our opinion, that "unorthodox" column generation methods such as the ones investigated here should be studied further.

Acknowledgments

This research was supported under Projects PT2007-003-08CCPP from the *Ministerio de Fomento, Spain*, under Project TRA2008-06782-C02-01 from the *Ministerio de Ciencia e Innovación, Spain* and under the Project PCC08-0081-4388-2 from the *Junta de Comunidades de Castilla-La Mancha.*

References

[1] García R, Marín A, Patriksson M. Column generation algorithms for nonlinear optimization, I: convergence analysis. Optimization 2003;52:171–200.
[2] Patriksson M. Sensitivity analysis of traffic equilibria. Transportation Science 2004;38:258–81.
[3] Nie Y. A class of bush-based algorithms for the traffic assignment problem. Transportation Research—B 2010;44:73–89.
[4] Holloway CA. An extension of the Frank and Wolfe method of feasible directions. Mathematical Programming 1974;6:14–27.
[5] von Hohenbalken B. Simplicial decomposition in nonlinear programming algorithms. Mathematical Programming 1977;13:49–68.
[6] Hearn DW, Lawphongpanich S, Ventura JA. Finiteness in restricted simplicial decomposition. Operations Research Letters 1985;4:125–30.
[7] Hearn DW, Lawphongpanich S, Ventura JA. Restricted simplicial decomposition: computation and extensions. Mathematical Programming Study 1987;31:99–118.
[8] Ventura JA, Hearn DW. Restricted simplicial decomposition for convex constrained problems. Mathematical Programming 1993;59:71–85.
[9] Larsson T, Patriksson M, Rydergren C. Simplicial decomposition with nonlinear column generation. Technical report, Department of Mathematics, Linköping Institute of Technology, Linköping Institute of Technology, Linköping, Sweden; 1996.
[10] Patriksson M. Nonlinear programming and variational inequality problems. A unified approach. Dordrecht: Kluwer Academic Publishers; 1999.
[11] Rosas D, Castro J, Montero L. Using ACCPM in a simplicial decomposition algorithm for the traffic assignment problem. Computational Optimization and Applications 2009;44:289–313.
[12] Bertsekas DP, Yu H. A unifying polyhedral approximation framework for convex optimization. LIDS report 2820, M.I.T., September 2009.
[13] Larsson T, Migdalas A, Patriksson M. A generic column generation scheme. Report LiTH-MAT-R-94-18, Department of Mathematics, Linköping Institute of Technology, Linköping, Sweden, Under revision for Optimization Methods and Software; 1994.

[14] Zangwill W. Nonlinear programming: a unified approach. Englewood Cliffs, NJ: Prentice-Hall; 1969.
[15] Sheffi Y. Urban transportation networks: equilibrium analysis with mathematical programming methods. Englewood Cliffs, NJ: Prentice-Hall; 1984.
[16] Patriksson M. The Traffic assignment problem. Models and methods. VSP, Utrecht, The Netherlands; 1994.
[17] Daneva M, Larsson T, Patriksson M, Rydergren C. A comparison of feasible direction methods for the stochastic transportation problem. Computational Optimization and Applications 2010;46(3):451–66.
[18] Larsson T, Patriksson M, Rydergren C. Applications of simplicial decomposition with nonlinear column generation to nonlinear network flows. In: Pardalos PM, Hager WW, Hearn DW, editors. Network optimization. Lecture notes in economics and mathematical systems, vol. 450. Berlin: Springer-Verlag; 1997. p. 346–73.
[19] Daneva M, Göthe-Lundgren M, Larsson T, Patriksson M, Rydergren C. A sequential linear programming algorithm with multi-dimensional search: derivation and convergence. Technical report, Department of Mathematics, Linköping Institute of Technology, Linköping Institute of Technology, Linköping, Sweden; 2007. Under revision for Optimization Methods and Software.
[20] Klingman D, Napier A, Stutz J. NETGEN: a program for generating large scale assignment, transportation, and minimum cost flow network problems. Management Science 1974;20:814–22.
[21] Bertsekas DP. Mincost grid problem generator. Fortran code, located at the internet web page: ⟨http://web.mit.edu/afs/athena.mit.edu/user/d/i/dimitrib/www/lopnet.txt⟩.
[22] Bertsekas DP, Polymenakos LC, Tseng P. $\varepsilon-$relaxation and auction methods for separable convex cost network flow problems. In: Pardalos PM, Hager WW, Hearn DW, editors. Network optimization. Lecture notes in economics and mathematical systems, vol. 450. Berlin: Springer-Verlag; 1997. p. 103–26.
[23] Fernández E, Cea JD, Florian M, Cabrera E. Network equilibrium models with combined modes. Transportation Science 1994;28:182–92.
[24] García R, Marín A. Network equilibrium models with combined modes: models and solution algorithms. Transportation Research—B 2005;39:223–54.
[25] Nguyen S, Dupuis C. An efficient method for computing traffic equilibria in networks with asymmetric transportation costs. Transportation Science 1984;18:185–202.
[26] LeBlanc L, Morlok EK, Pierskalla WP. An efficient approach to solving the road network equilibrium traffic assignment problem. Transportation Research—B 1975;9:309–18.
[27] Florian M. Private communication with M. Patriksson; 1990.
[28] Frank M, Wolfe P. An algorithm for quadratic programming. Naval Research Logistics Quarterly 1956;3:95–110.
[29] Goldstein A. Convex programming in Hilbert space. Bulletin of the American Mathematical Society 1964;70:709–10.
[30] Levitin E, Polyak BT. Constrained minimization methods. USSR Computational Mathematics and Mathematical Physics 1966;6:1–50.
[31] García R. Metodología para el diseño de redes de transporte y para la elaboración de algoritmos en programación matemática convexa diferenciable. PhD thesis, Escuela Técnica Superior de Ingenieros Aeronaúticos, Universidad Politécnica de Madrid; 2001.
[32] Evans SP. Derivation and analysis of some models for combining trip distribution and assignment. Transportation Research 1976;10:37–57.
[33] Bertsekas DP. Projected Newton methods for optimization problems with simple constraints. SIAM Journal on Control and Optimization 1982;20:221–46.
[34] Hearn DW, Lawphongpanich S, Ventura J, Yang K. RSDNET user manual. Research Report 87-17, Department of Industrial and Systems Engineering, University of Florida, Florida; 1987.
[35] Kennington A, Helgason R. Algorithms for network programming. New York, NY: John Wiley & Sons; 1980.
[36] García R, Marín A. Using restricted simplicial decomposition within partial linearization methods. In: Proceedings of 16th international symposium on mathematical programming. École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1997.
[37] Gallo G, Pallotino S. Shortest paths algorithms. Annals of Operations Research 1988;13:3–79.
[38] Shah BV, Beuhler RJ, Kempthorne O. Some algorithms for minimizing a function of several variables. SIAM Journal on Applied Mathematics 1964;12: 74–92.