ELSEVIER

Discrete Applied Mathematics 112 (2001) 73–99

DISCRETE APPLIED MATHEMATICS

Bundle-based relaxation methods for multicommodity capacitated fixed charge network design

Teodor Gabriel Crainic[a,b,*], Antonio Frangioni[c], Bernard Gendron[b,d]

[a] *Département des sciences administratives, Université du Québec à Montréal, Montreal, Canada*
[b] *Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montreal, QC, Canada H3C 3J7*
[c] *Dipartimento di informatica, Università di Pisa, Italy*
[d] *Département d'informatique et de recherche opérationnelle, Montreal, Canada*

Received 1 April 1998; revised 1 April 1999; accepted 1 August 2000

Abstract

To efficiently derive bounds for large-scale instances of the capacitated fixed-charge network design problem, Lagrangian relaxations appear promising. This paper presents the results of comprehensive experiments aimed at calibrating and comparing bundle and subgradient methods applied to the optimization of Lagrangian duals arising from two Lagrangian relaxations. This study substantiates the fact that bundle methods appear superior to subgradient approches because they converge faster and are more robust relative to different relaxations, problem characteristics, and selection of the initial parameter values. It also demonstrates that effective lower bounds may be computed efficiently for large-scale instances of the capacitated fixed-charge network design problem. Indeed, in a fraction of the time required by a standard simplex approach to solve the linear programming relaxation, the methods we present attain very high-quality solutions.

Keywords: Multicommodity capacitated fixed-charge network design; Lagrangian relaxation; Subgradient methods; Bundle methods

Résumé

La relaxation lagrangienne apparaît comme une technique prometteuse pour générer efficacement des bornes de qualité pour des exemplaires de grande taille du problème de conception de réseau avec coût fixe et capacité. Cet article présente les résultats d'expériences visant à comparer les méthodes de sous-gradients et de faisceaux, appliquées à l'optimisation des duaux lagrangiens issus de deux relaxations lagrangiennes. Cette étude démontre que les méthodes de faisceaux semblent supérieures aux méthodes de sous-gradients puisqu'elles convergent plus rapidement et sont plus robustes. Cet article montre également que des bornes inférieures de qualité pour des exemplaires de grande taille du problème de conception de réseau avec coût fixe et capacité peuvent être calculées de manière efficace. En effet, les approches de relaxation

* Corresponding author. Tel.: +1-514-343-7143; fax: +1-514-343-7121.
E-mail address: theo@crt.umontreal.ca (T.G. Crainic).

0166-218X/01/$ - see front matter
PII: S0166-218X(00)00310-3

lagrangienne obtiennent des bornes de très grande qualité en une fraction du temps requis par des méthodes de type simplexe.

Mots-clé: Problème de conception de réseau avec capacité et coût fixe; Relaxation lagrangienne; Méthodes de sous-gradients; Méthodes de faisceaux

1. Introduction

Network design models arise in various applications in telecommunications, transportation, logistics and production planning [2,3,16,31,32]. In many of these applications, the models are characterized as follows: given a network with arc capacities, it is required to send flows (which might be fractional) in order to satisfy known demands between origin-destination pairs. In doing so, one pays a price not only for routing flows, but also, in the form of fixed costs, for using arcs. Although deceptively simple to state, these *capacitated fixed-charge network design* problems are notoriously difficult. Not only they are NP-hard, but also, when one attempts to formulate and solve them as mixed-integer programs, several complications emerge. Indeed, because "simple" linear programming relaxations generally do not provide good approximations to the mixed-integer programs, one has to derive formulations with very large numbers of variables and constraints in order to obtain tight bounds. Therefore, traditional simplex-based branch-and-bound methods (which do not incorporate cutting plane or column generation procedures) are likely to solve only the simplest instances.

Lagrangian relaxation approaches offer an interesting alternative [17,18,26]. Several Lagrangian relaxations are possible, however, and researchers have shown that many yield the same theoretical lower bound. Furthermore, initial studies [17,18] have made clear that the implementation and calibration of Lagrangian-based methods have a significant impact on their behavior and performance. In these initial studies, the authors have used the traditional subgradient methods to optimize the Lagrangian duals and derive lower bounds. Other general techniques in the field of nondifferentiable optimization could be used, however; among them, *bundle methods* [25,29] appear especially promising. In contrast with subgradient methods, for which the most efficient variants [5,9] use subgradients from previous iterations in an aggregated form to avoid zigzagging, bundle approaches keep the subgradients generated so far in a disaggregated form (the so-called "bundle") to compute a tentative direction of ascent. A wide choice of alternatives thus appears available to design and implement Lagrangian-based bounding procedures for the capacitated fixed-charge network design problem and no experimental results have been presented yet, at least to our knowledge, to guide this choice. We aim to address this issue.

The main objective of this paper is therefore to present the results of comprehensive experiments aimed at calibrating and comparing bundle and subgradient methods applied to the optimization of the Lagrangian duals arising from the most promising relaxations presented in [17]. Two Lagrangian relaxations are compared: the first one,

called *shortest path relaxation*, relaxes the so-called "forcing" constraints and yields a Lagrangian subproblem that decomposes into a collection of shortest path problems; the second one, called *knapsack relaxation*, dualizes the flow conservation constraints, which allows to solve the Lagrangian subproblem as a collection of continuous knapsack problems.

The most significant contribution of this paper is a detailed analysis of the resulting relaxation methods and their implementations, based on experiments on a large set of test problems of various characteristics. The study offers insights into the behavior of each particular relaxation method and proposes strategies for their design and implementation. These experiments also demonstrate that bundle methods show two main advantages, when compared to subgradient approaches:

- *Bundle methods converge faster*. The increased complexity of bundle methods (both in theory and from an implementation viewpoint) is most often compensated by a faster convergence toward the optimal value of the Lagrangian dual. Indeed, our experiments show that, with the same computational effort (measured by CPU time), bundle methods generally provide better values than the best subgradient methods.
- *Bundle methods are more robust*. When efficient and portable software tools are developed [12,13], bundle methods require few parameters to adjust. Moreover, even when initial "bad" parameter settings are allowed, bundle methods still derive a relatively precise estimate of the optimal value of the Lagrangian dual. In contrast, subgradient methods usually require more parameter adjustments and are very sensitive to them. Indeed, it is not unusual to have a parameter setting for a particular relaxation that performs very well on some instances and very poorly on others.

Our study also shows that *effective* lower bounds can be computed *efficiently* for large-scale instances of the capacitated fixed-charge network design problem. Indeed, the Lagrangian duals corresponding to the two relaxations presented in this paper have the same optimal value, which is also the optimal value of the so-called "strong" linear programming relaxation [17]. For the instances we tested, the resulting "strong" lower bound on the optimal value of the mixed-integer program was reported to be within 9% of optimality on average, for those instances for which a feasible (but not necessarily optimal) solution could be obtained by a standard simplex-based branch-and-bound code [8]. This average optimality gap is very reasonable for such large-scale complex problems, as exemplified by the literature on solution methods for related capacitated network design models (see for example [4,6,30] and the survey [19]). Furthermore, the Lagrangian-based bounding procedures approximate the "strong" lower bound with very high accuracy and in a fraction of the time required by a competitive commercial simplex code to compute it.

The paper is organized as follows. In Section 2, we recall the formulation of the capacitated fixed-charge network design and then present the Lagrangian relaxations. A special case of the shortest path relaxation, shown to be very effective by our experiments, is (to our best knowledge) described for the first time. Section 3 gives an overview of the nondifferentiable optimization methods used in our implementation. In Section 4, we present and analyze results of experiments on a large set of randomly

generated problems of various characteristics. In the Conclusion, we summarize our work and propose extensions.

2. Problem formulation and relaxations

Given a directed graph $G = (N, A)$ and a set of commodities K which represent demands to satisfy between origin–destination pairs, the objective is to minimize the sum of arc transportation and design costs, the latter being charged whenever an arc is used. For each arc (i, j), the transportation cost per unit of commodity k is denoted c_{ij}^k, while the design cost is denoted f_{ij}. Both costs are assumed to be nonnegative. Let $d^k > 0$ denote the demand to satisfy between origin $O(k)$ and destination $D(k)$ for each commodity k. On each arc (i, j), there is a capacity $u_{ij} > 0$, and an upper bound $b_{ij}^k = \min\{d^k, u_{ij}\}$ may be imposed on the amount of flow of commodity k.

To formulate the problem, we introduce continuous flow variables x_{ij}^k, which reflect transportation decisions for each arc (i, j) and commodity k, and 0–1 design variables y_{ij}. We also define the sets of outward and inward neighbors of any node i: $N^+(i) = \{j \in N \mid (i, j) \in A\}$ and $N^-(i) = \{j \in N \mid (j, i) \in A\}$, respectively. The model is then given by

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij}, \tag{1}$$

$$\sum_{j \in N^+(i)} x_{ij}^k - \sum_{j \in N^-(i)} x_{ji}^k = \begin{cases} d^k, & i = O(k) \\ -d^k, & i = D(k) \\ 0, & i \neq O(k), D(k) \end{cases} \quad \forall i \in N,\ k \in K, \quad (v_i^k), \tag{2}$$

$$\sum_{k \in K} x_{ij}^k \leqslant u_{ij} y_{ij} \quad \forall (i, j) \in A, \quad (\alpha_{ij}), \tag{3}$$

$$x_{ij}^k \leqslant b_{ij}^k y_{ij} \quad \forall (i, j) \in A, \quad k \in K, \quad (\beta_{ij}^k), \tag{4}$$

$$x_{ij}^k \geqslant 0 \quad \forall (i, j) \in A, \quad k \in K, \tag{5}$$

$$0 \leqslant y_{ij} \leqslant 1 \quad \forall (i, j) \in A, \tag{6}$$

$$y_{ij} \quad \text{integer } \forall (i, j) \in A. \tag{7}$$

The *flow conservation constraints* (2) ensure that demands are satisfied between each origin–destination pair and that total incoming flow equals total outgoing flow at every transshipment node. To these flow constraints, we associate Lagrangian multipliers v, which are unrestricted in sign. Constraints (3) not only ensure that arc capacities are respected, but they force the flow of any commodity to be 0 if the arc is not chosen in the design. Constraints (4) achieve the same objective and are, therefore, completely redundant. However, they can significantly improve the lower bounds obtained through

relaxations, as demonstrated by computational experiments reported in [17] and in Section 4 of this paper. Constraints (3) and (4) are called weak and strong *forcing constraints*, respectively, and nonnegative multipliers α and β are associated to them.

The first Lagrangian relaxation, called *shortest path* (or *flow*) relaxation, is obtained by dualizing the forcing constraints (3) and (4) [17,18]. The resulting Lagrangian subproblem is

$$Z(\alpha,\beta) = \min \sum_{k\in K}\sum_{(i,j)\in A} (c_{ij}^k + \alpha_{ij} + \beta_{ij}^k)x_{ij}^k + \sum_{(i,j)\in A}\left(f_{ij} - u_{ij}\alpha_{ij} - \sum_{k\in K} b_{ij}^k\beta_{ij}^k\right) y_{ij} \tag{8}$$

subject to constraints (2) and (5)–(7). It decomposes into $|K|$ shortest path problems and a problem, in y variables only, solvable by simple inspection of the cost signs. Since the Lagrangian subproblem has the *integrality property* (e.g., its optimal value is the same whether we relax the integrality constraints (7) or not) [20], the Lagrangian dual – $\max_{\alpha,\beta\geqslant 0} Z(\alpha,\beta)$ – has the same optimal value as the *strong* linear programming relaxation obtained by dropping the integrality requirements (7).

This observation suggests an interesting alternative approach to compute the same lower bound which can be seen as a particular case of the shortest path relaxation. The idea follows from the fact that, assuming design variables are continuous, these may be eliminated by *projecting* them over the space of the flow variables. More precisely, assume that the integrality constraints (7) are dropped and the strong forcing constraints (4) are relaxed in a Lagrangian way; then, add the capacity constraints

$$\sum_{k\in K} x_{ij}^k \leqslant u_{ij} \quad \forall (i,j)\in A. \quad (\gamma_{ij}) \tag{9}$$

In the resulting formulation, it is easy to see that the y variables can be dropped since an optimal solution to this subproblem must satisfy

$$y_{ij} = \begin{cases} \sum_{k\in K} x_{ij}^k/u_{ij} & \text{if } f_{ij}(\beta)\geqslant 0, \\ 1 & \text{otherwise,} \end{cases} \qquad \forall (i,j)\in A, \tag{10}$$

where $f_{ij}(\beta) = f_{ij} - \sum_{k\in K} b_{ij}^k\beta_{ij}^k$. Following this projection of the y variables over the x variables, we can dualize the capacity constraints (9) by using the nonnegative multipliers γ. The resulting subproblem

$$Z(\gamma,\beta) = \min \sum_{k\in K}\sum_{(i,j)\in A} (c_{ij}^k + \gamma_{ij} + f_{ij}(\beta)^+/u_{ij} + \beta_{ij}^k)x_{ij}^k + \sum_{(i,j)\in A} (f_{ij}(\beta)^- - u_{ij}\gamma_{ij}), \tag{11}$$

subject to (2) and (5), where $f_{ij}(\beta)^+ = \max\{0, f_{ij}(\beta)\}$ and $f_{ij}(\beta)^- = \min\{0, f_{ij}(\beta)\}$, also decomposes into $|K|$ shortest path problems. This *projected shortest path* relaxation is easily seen to be a special case of the shortest path relaxation (8), obtained by setting $\alpha_{ij} = \gamma_{ij} + f_{ij}(\beta)^+/u_{ij}$ $\forall (i,j)\in A$. Our computational results (see Section 4) suggest that this projected variant of the shortest path relaxation is very effective,

as it usually converges faster to the optimal value of the Lagrangian dual than the "ordinary" shortest path relaxation. Intuitively, this is due to the fact that the projected variant produces "better" y, and therefore better subgradients to drive the search (see Section 3).

An interesting special case of the projected shortest path relaxation arises when one fixes β to 0, which is equivalent to remove the valid inequalities (4) from the formulation. In this case, the Lagrangian subproblem can be interpreted as the relaxation of the capacity constraints (9) of a multicommodity minimum cost network flow (MMCF) problem, where the transportation costs are defined as $c_{ij}^k + f_{ij}/u_{ij}\ \forall (i,j) \in A, k \in K$. It is easy to see that the resulting Lagrangian dual has the same optimal value as the *weak* linear programming relaxation, which is obtained from the problem formulation by dropping the integrality requirements (7) and the strong forcing constraints (4). Although this weak bound can be computed *exactly* and efficiently by a bundle method based on the relaxation of the capacity constraints (as suggested by our computational experiments reported in Section 4 and the results presented in [15]), it is probably too weak to serve as a basis for branch-and-bound methods.

The second Lagrangian relaxation we study, called *knapsack* relaxation, is based on dualizing the flow conservation constraints [17,26]. The Lagrangian subproblem may be written as

$$Z(v) = \min \sum_{k\in K} \sum_{(i,j)\in A} (c_{ij}^k + v_i^k - v_j^k)x_{ij}^k + \sum_{(i,j)\in A} f_{ij}y_{ij} + \sum_{k\in K} d^k(v_{D(k)}^k - v_{O(k)}^k), \quad (12)$$

subject to constraints (3)–(7). This subproblem can be easily solved by inspection of the signs of the costs of the following problem:

$$Z(v) = \min_{y\in\{0,1\}^{|A|}} \sum_{(i,j)\in A} (f_{ij} + g_{ij}(v))y_{ij} + \sum_{k\in K} d^k(v_{D(k)}^k - v_{O(k)}^k), \quad (13)$$

where $g_{ij}(v)$, for each $(i,j) \in A$, is the optimal value of the continuous knapsack problem:

$$g_{ij}(v) = \min \sum_{k\in K} (c_{ij}^k + v_i^k - v_j^k)x_{ij}^k, \quad (14)$$

$$\sum_{k\in K} x_{ij}^k \leqslant u_{ij}, \quad (15)$$

$$0 \leqslant x_{ij}^k \leqslant b_{ij}^k \quad \forall k \in K. \quad (16)$$

Since this Lagrangian subproblem has the integrality property, the Lagrangian dual – $\max_v Z(v)$ – has the same optimal value as the strong linear programming relaxation. Therefore, it also gives the same bound as the Lagrangian dual of the shortest path relaxation, provided one can solve the Lagrangian duals to optimality. Although the two relaxations are theoretically equivalent (e.g., they provide the same lower bound), their relative performances, both in terms of computational efficiency and speed of convergence, have not been fully characterized yet. But these performances are clearly

dependent upon the method chosen to optimize the Lagrangian duals. A description of the methods used in our implementation is the topic of the next section.

3. Nondifferentiable optimization methods

The Lagrangian duals related to the relaxations presented in the previous section may be cast in the form of nondifferentiable optimization problems schematically described as

$$\max_{w\in\Omega} \phi(\omega), \tag{17}$$

where ϕ is a concave nondifferentiable function, finite everywhere on Ω, a set which represents either the whole real space (of a dimension corresponding to the number of multipliers) or its nonnegative orthant. For each $\omega \in \Omega$, the value of $\phi(\omega)$ can be obtained by solving the Lagrangian subproblem; then, one *subgradient* $g(\omega)$ of ϕ in ω can be easily retrieved from the optimal primal solution (x, y) of the subproblem. For the shortest path relaxation, the subgradient takes the form

$$\left[\sum_{k\in K} x_{ij}^k - u_{ij}y_{ij} \quad \forall (i,j)\in A; \quad x_{ij}^k - b_{ij}^k y_{ij} \quad \forall (i,j)\in A, \quad k\in K\right], \tag{18}$$

while for the knapsack relaxation, the expression of the subgradient is

$$\left[\sum_{j\in N^+(i)} x_{ij}^k - \sum_{j\in N^-(i)} x_{ji}^k - \begin{cases} d^k, & i = O(k) \\ -d^k, & i = D(k) \\ 0, & i \neq O(k), D(k) \end{cases} \quad \forall i \in N, k \in K\right]. \tag{19}$$

Several iterative methods have been designed to solve problem (17) with the sole help of this kind of information. Among them, we have tested a *bundle-based algorithm* and several variants of the *subgradient method*. All these algorithms start from an initial estimate $\bar{\omega}$ of the solution and iteratively repeat the following five basic steps: (1) select a *tentative ascent direction* d; (2) select a *stepsize* t; (3) evaluate $\phi(\bar{\omega} + td)$ and the corresponding subgradient; (4) eventually, move the current point to $\bar{\omega} \leftarrow \bar{\omega} + td$; (5) check some stopping criteria.

Despite sharing the same abstract algorithmic structure, bundle and subgradient methods are very different under many viewpoints (some of which are highlighted by our computational experiments, reported in Section 4):

- Bundle approaches are *ascent* methods, since they only move the current point $\bar{\omega}$ when a (sufficiently) better point is found, while subgradient methods update the current point at every iteration, even if the newly obtained point has a worse (smaller) value of the function.
- Subgradient methods are (almost) oblivious of the search history, in the sense that, at any iteration, they only use the current subgradient and the previous direction to compute the new one, while bundle methods (in principle) retain all the previously obtained subgradients in a *disaggregated* form.

- Most variants of the subgradient methods do not necessarily converge with a "practical" choice of stepsize [24], hence usually they give no guarantee of reaching the optimal solution; bundle methods *finitely* converge when maximizing a polyhedral function, and also give a *proof* of the (ε)-optimality of the obtained solution when they stop.
- When used to optimize a Lagrangian dual, most variants of the subgradient methods only convey *dual* information, while bundle methods provide satisfactory *primal* information as well.
- Subgradient methods are rather "unstable", their performances being highly dependent on the setting of several parameters (whose best setting for a class of instances may well be very poor for another class, up to the point that the method can "diverge"), while the performances of bundle methods are much more stable and depend on less parameters (which, though, may have a significant impact).
- Subgradient methods are very easy to implement and their computational cost is almost always dominated by the cost of evaluating ϕ, while bundle methods require a much more sophisticated implementation, their computational bottleneck in practice often being the computation of the direction d.

We now substantiate these points by describing in detail how the basic operations are accomplished within each class of methods.

3.1. Subgradient methods

As already mentioned, subgradient methods move their *current point* at each iteration: if we let $\bar{\omega}^i$ and d^i denote the current point and the direction at the ith iteration, respectively, ϕ is calculated at the *tentative point* $\omega^i = \bar{\omega}^i + t^i d^i$ and the current point is moved to $\omega^i (\bar{\omega}^{i+1} \leftarrow \omega^i)$. Since $\bar{\omega}^{i+1}$ cannot be guaranteed to be better (in terms of its ϕ value) than $\bar{\omega}^i$, the best value of ϕ found so far (and maybe the corresponding ω) must be retained; let ϕ^i_* denote this value.

Early versions of the subgradient algorithm [24,33] used only the subgradient to compute a direction, that is $d^i = g^i = g(\bar{\omega}^i)$. It was quickly realized, however, that taking into account the direction d^{i-1} of the previous iteration could lead to performance improvements. The more general formula

$$d^i = g^i + \theta^i d^{i-1} \tag{20}$$

(of which the previous one is the special case for $\theta^i = 0$) has been usually reported as being more effective for "clever" choices of θ^i. The simplest choice, called *Crowder rule* [9], used θ^i fixed to some value < 1. Another approach used the more sophisticated *Camerini–Fratta–Maffioli rule* [5]

$$\theta^i = \begin{cases} -\mu g^i d^{i-1} / ||d^{i-1}||^2 & \text{if } g^i d^{i-1} < 0, \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

where μ is a parameter to be selected through computational experiments (the authors indicate that 1.5 usually constitutes a good choice). The rationale for (21) is that a

proper choice of μ in the interval $[0,2]$ guarantees that d^i is at least as good a direction as g^i. The need for the hand-tuned parameter μ may be overcomed by using the self-adjusting rule: $\mu = -||g^i||||d^{i-1}||/g^i d^{i-1}$, which is based on geometrical arguments. This yields the following *modified Camerini–Fratta–Maffioli rule* [5]:

$$\theta^i = \begin{cases} ||g^i||/||d^{i-1}|| & \text{if } g^i d^{i-1} < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

Feasibility issues that may arise when using the subgradient algorithm are tackled by means of projection. The actual direction is the projection over the active constraints of the direction obtained by using one of the previous formulae (this is almost costless for the case of nonnegativity constraints). The tentative point must be projected as well to ensure its feasibility, since the stepsize is usually selected without taking the constraints into consideration. A variant of the standard stepsize formula [33], used in our implementation is

$$t^i = \lambda^i(\bar{\phi}^i - \phi(\bar{\omega}^i))/g^i d^i \tag{23}$$

where λ^i is an iteration-dependent scaling factor and $\bar{\phi}^i$ an estimate of the maximum of ϕ.

But for the selection of the rule for computing d^i, the most critical issues in subgradient methods come from the adjustment of the parameters in the stepsize formula (23). The fundamental issues are: (1) how λ^i is updated; (2) how $\bar{\phi}^i$ is estimated.

Typically, λ^i is divided by a constant factor $\gamma_1 > 1$ (such as 2) every time ϕ^i_* has not been improved for γ_2 consecutive iterations. Note that this exponential decrease of λ may theoretically cause the subgradient to converge to a non optimal point: in fact, the usual convergence theorems [1,25,29] require the overall stepsize t^i to converge to zero slowly enough that the series $\sum_{i=0}^{\infty} t^i$ diverges. In practice, however, the subgradient always seems to converge "near enough" to an optimal solution, whenever it converges.

A good $\bar{\phi}^i$ may sometimes be provided, typically when the computation of the Lagrangian bound is associated with heuristic or enumerative approaches to the original problem that produce feasible solutions. However, this is not always the case. The usual way for providing such an estimate is then to multiply ϕ^i_* by a constant $\gamma_3 > 0$. This is arguably a coarse method, but more sophisticated ones would call for other parameters in order to dynamically adjust γ_3. Furthermore, our experience has shown that providing a tight bound, when available, may sometimes deteriorate the performances of the method.

Hence, the choice of the stepsize is influenced by at least four parameters: $\gamma_1, \gamma_2, \gamma_3$ and λ^0, the initial value of λ^i. In practice, all these parameters may have an impact, but the most critical is λ^0, followed by γ_2. As shown by our computational experiments, the setting of λ^0 usually makes a large part of the difference between obtaining good performances and having the method "diverge", never being able to improve on the initial estimate. Unfortunately, no settings appear to provide at least reasonable performances on all problem classes: depending on the particular instance and the other parameters,

the "good" numerical values for λ^0 may vary. Thus extensive experimentation appears to be the only way to guess a reasonable value.

Another possible choice appears when the multipliers are nonnegative. When calculating the stepsize formula (23), the scalar product $d^i g^i$ in the denominator may either use the *original* or the *projected* direction d^i. In [28], sound theoretical arguments (based on the notion of conditional subgradients) as well as computational results have been offered supporting the choice of the projected direction. Our results have generally confirmed the validity of such a choice, but still computational results were necessary in order to establish it. This is especially true since the original development in [28] was limited to the "basic" subgradient method using $d^i = g^i$, while we extended it to the more general case of (20). This also suggested the possibility to use the projected versions of d^{i-1} and g^i in (21) and (22), that is for computing the direction, but the results were generally worse than those of the non-projected variants.

Finally, as far as the stopping criteria are concerned, the theoretical stopping criterion of the subgradient is that $||g^i||$ is "small enough". This is a sound criterion, since finding a zero (ε-)subgradient at $\bar{\omega}^i$ suffices to prove that $\bar{\omega}^i$ is (ε)-optimal. In the constrained case (e.g., with nonnegative multipliers), $||g^i||$ has to be replaced with the norm of the projected subgradient. Unfornately, this stopping criterion almost never applies in practice (it would require that the optimal solution of the Lagrangian subproblem be feasible for the original problem): hence, an alternative criterion is to stop after ϕ^i_* has not been improved for $\gamma_4 > \gamma_2$ consecutive iterations. In order to avoid stopping "too early", γ_4 must not be chosen "too small". The result is that, even for "easy" instances, subgradient methods almost always stop after having performed the maximum number of iterations.

3.2. Bundle methods

The main idea of bundle methods is to use the information generated previously to build a *model* of the function ϕ to be maximized, and use this model to drive the search for a better point. It makes use of the *information transport property* [25,29]: if d is a σ-subgradient of ϕ in $\bar{\omega}$ ($\phi(\omega) \leqslant \phi(\bar{\omega}) + d(\omega - \bar{\omega}) + \sigma \; \forall \omega \in \Omega$), then d is also σ'-subgradient of ϕ in ω' where

$$\sigma' = \sigma + d(\omega' - \bar{\omega}) - (\phi(\omega') - \phi(\bar{\omega})). \tag{24}$$

It follows from this property that at each iteration i, one can associate to each previously generated subgradient g^h, $h = 1, \ldots, i-1$, a *linearization error* $\alpha^h \geqslant 0$ such that g^h is an α^h-subgradient of ϕ at the current point $\bar{\omega}^i$. Therefore, the polyhedral concave function

$$\phi_B(\omega) = \min_{i \in B} \{g^i \omega + \alpha^i\} \tag{25}$$

called the *cutting plane model* of ϕ, is an *upper approximation* of $\phi - \phi(\bar{\omega}^i)$ for each set – or *bundle* – $B \subseteq \{1, \ldots, i-1\}$. Using the maximizer of ϕ_B as the next trial point results in the well-known *cutting plane algorithm* [27], which suffers, however, from several drawbacks, in particular: (1) the maximizer of ϕ_B is most likely to be

undefined when only few subgradient are known; (2) the algorithm has no *locality* property, i.e., it tends to generate iterates that can be very far from the current point even if the latter is "near" to the optimum.

Bundle methods can be viewed as "stabilized" versions of the cutting plane algorithm, where ϕ_B is maximized subject to some stabilizing device that enforces the locality properties of the iterates [25,29] and also, usually, addresses the unboundedness problem. In our implementation, we have chosen the "standard" stabilizing device, i.e., the direction is selected via the following quadratic problem (QP):

$$\phi_B(\omega) = \max_d \left\{ \phi_B(d) - \frac{1}{2t^i}||d||^2 \right\}, \tag{26}$$

where $t^i > 0$ is called the *trust region parameter*. Problem (26) always attains a (unique) finite solution d^i that is used as the next direction with constant stepsize 1. However, t^i may also be interpreted as a stepsize. Indeed, the quadratic dual of (26) is

$$\min_{\theta \geqslant 0} \left\{ \frac{t^i}{2} \left\| \sum_{j \in B} g^j \theta_i \right\|^2 + \sum_{j \in B} \alpha_j \theta_j \middle| \sum_{j \in B} \theta_j = 1 \right\} \tag{27}$$

and $d^i = t^i z^i$, where $z^i = \sum_{j \in B} g^j \theta_j^i$ and θ^i is the optimal solution of (27). Hence, the new trial point is found along direction z^i, a *convex combination* of the previously obtained subgradients using t^i as a predefinite stepsize.

Our bundle method requires the solution of (27) at each iteration in order to compute the next trial point. Since this can easily become the computational bottleneck of the approach, especially as the number of variables grows, specialized QP codes [12] are instrumental in order to obtain an efficient implementation. This is especially true in the *constrained* case: in fact, in order to ensure the feasibility of the next current point, the constraints $d \geqslant -\bar{\omega}^i$ must be added to (26), possibly making it harder to solve.

One very important feature that the QP solver must possess is to extensively support reoptimization, in particular by allowing the on-line creation and destruction of variables. This so-called *variable generation strategy* has already been shown critical in order to keep low the cost of solving the QP when applied to a Lagrangian relaxation approach for MMCF problems [15]. The importance of this technique is confirmed in the present setting, as the results presented in Section 4 clearly show.

Another possibility for reducing the cost of solving the QP is to keep small the size of the bundle. Usually, subgradients are discarded after having been "inactive" (i.e., with $\theta_j^i = 0$) for a fixed number of consecutive iterations. Quite "conservative" rules work best: after being inactive for 20 iterations, an old subgradient is unlikely to turn up useful again, but selecting much smaller values often deprives the algorithm of potentially still useful information, deteriorating its speed of convergence. It is also possible to fix the bundle to a maximum size (down to two): this can be done without impairing convergence provided that the *aggregated subgradient* z^i (and its *aggregated linearization error* $\sigma^i = \sum_{j \in B} \alpha^j \theta_j^i$) is added to the bundle each time "active"

subgradients are eliminated. "Aggressive" aggregation, corresponding to a very limited bundle size, usually leads to poorer performances in terms of convergence, but it may significantly reduce the overall cost of the algorithm, both in time and memory, up to making it comparable with that of the subgradient method. This is illustrated by the computational results described in Section 4.

The choice of the trust region parameter t^i, that may well be regarded as the choice of the stepsize, is potentially critical for the performance of the bundle algorithm. It is also intimately tied to the choice between moving the current point $\bar{\omega}^i$ to the trial point ω^i, called a *serious step* or SS, or leaving it unchanged, called a *null step* or NS. Typically, a SS is performed when $\phi(\omega^i)$ "sufficiently improves" over $\phi(\bar{\omega}^i)$, that is $\phi(\omega^i) - \phi(\bar{\omega}^i) \geqslant m_1 \phi_B(d^i)$ for some fixed parameter $0 \leqslant m_1 < 1$. The usual value for m_1 is 0.1 although, rarely, $m_1 = 0$ happens to work slightly better. Note that $\phi_B(d^i)$ is the increase predicted by the cutting plane model for a movement to ω^i: hence, a SS is performed when ϕ_B predicts the behavior of the actual function ϕ "well enough". On the other hand, a NS is performed when ϕ_B is a "poor" model of ϕ. In this case, however, new information (g^i, α^i) is gathered that will "refine" ϕ_B as a model of ϕ in the current point ω^i, hopefully leading to a better d^{i+1}.

The bundle algorithm will work with t^i fixed to any finite value throughout all the execution. However, a too small t^i would make it perform short steps, each yielding little improvement, while a too large t^i will make it perform too many NS between two consecutive SS. Therefore, increasing t^i after a SS or decreasing it after a NS appears to be a sensible choice. In the former case, heuristics have been developed [13] that approximate the restriction of ϕ along d^i with a quadratic function and choose t^i as its maximizer (provided that it is larger than t^i, which turns out to be the case only when the slope of ϕ in ω^i is positive). In the latter case, a typical check involves the size of α^i and that of the aggregated linearization error σ^i: the rationale for this is that α^i is a sort of "measure of accuracy" of g^i as first-order information about ϕ in $\bar{\omega}^i$, so that g^i can be believed to carry "accurate" information only if $\alpha^i \leqslant m_2 \sigma^i$ for some fixed parameter $m_2 > 0$. Otherwise, a t^i decrease is preferred, in the hope that ϕ_B be a better model of ϕ "nearer" to the current point. The actual value of t^{i+1} is selected via a heuristic rule similar to the one for the increasing case.

The value usually suggested for m_2 is 0.9 [34]. This appears to be a sensible choice, because z^i itself has just shown to be "inaccurate", so that "better" information should be required. However, in our experience, such a value invariably leads to a sequence of t^i reductions that, in the long run, seriously slows down the overall convergence, a phenomenon commonly referred to as *tailoff effect*. Higher settings, such as $m_2 = 3.0$, prevent t^i from changing during almost all the run and have proved to be quite effective in preventing tailoff effects. However, a correct choice of the initial value t^0 could then become important, although the heuristics are usually capable of correcting wrong choices of t^0. A good value for t^0 depends on the "scaling" of ϕ and can be usually guessed by observing the run of just one instance in a class and choosing the order of magnitude of the t^i produced on average by the heuristics. Furthermore, this kind of

"scaling" information is also useful for the appropriate setting of the stopping criterion, to be discussed shortly.

To counter the tailoff effect, more sophisticated self-adjusting rules for choosing t^i, called *long-term t-strategies*, have been developed in [13]. Although their detailed description is outside the scope of the present work, we briefly recall their ingredients. Long-term t-strategies are based on the idea that $\phi_B(d^i)$ quantifies the maximum improvement that can be expected by the next step, and this quantity is increasing with t^i. Hence, by guessing what a "reasonable improvement" is, it is possible either to increase t^i (*hard long-term t-strategy*) or to inhibit its decrease (*soft long-term t-strategy*) if it turns out to produce too small values of $\phi_B(d^i)$. Obviously, the main issue is then how the "reasonable improvement" estimate may be obtained. In our implementation, we define an initial target and allow it to decrease only when the stopping criterion of the algorithm is satisfied for that target.

Given a required relative accuracy ε, the stopping criterion of our bundle algorithm is $t^*||z^i||^2 + \sigma^i \leqslant \varepsilon\phi(\bar{\omega}^i)$, where t^* is a user-provided parameter. It is easy to see that any stopping criterion for a bundle algorithm should require both $||z^i||$ and σ^i to be "small". In the literature, it is suggested to test them against two separate thresholds. However, figuring out the numerical value to be used as a threshold for $||z^i||$ is not easy. Using the parameter t^* is more convenient, since it is in the same "units of measure" than t^i. In fact, a good choice of t^* is usually just one order of magnitude larger than a well-chosen t^0: this reduces the "critical" parameters to be sought for by one, and almost always provides solutions that are within the required precision (as long as the bundle is given time enough to converge). Furthermore, this form of stopping criterion is critical for implementing the long-term t-strategies.

An important characteristic of the bundle is that, unlike the subgradient, its stopping criterion is *effective*: indeed, on "easy" instances the bundle usually reaches convergence before the maximum iteration limit is exceeded. This does not mean that the bundle stops as soon as a good enough solution has been found: "certifying" the optimality (that is, finding a z^i of small enough norm) can be costly. Still, in some cases, a bundle method may require significantly less running time than a subgradient method, simply because it is able to stop much earlier.

4. Computational results

The combination of a particular Lagrangian dual problem – resulting from one of the shortest path or knapsack relaxations of Section 2 – and of a nondifferentiable optimization method – subgradient or bundle – together with a choice of implementation criteria (Section 3), yields many *relaxation methods*, each one with possibly significantly different behavior and performance. These characteristics have first to be qualified prior to any rigorous comparison.

The computational experiments we performed were thus guided by two objectives: (1) to determine a promising set of parameters and rules for each relaxation method and

for different classes of instances (the *calibration phase*); (2) to compare the relaxation methods, as well as a number of other well-known bounding procedures (e.g., linear programming relaxations), by analyzing their performances with respect to various problem characteristics.

In order to achieve these objectives, we have run our tests on 196 problem instances (also used in [8] to test a tabu search procedure for the same problem) obtained from a network generator similar to the one described in [17,18]. When provided with target values for $|N|$, $|A|$, and $|K|$, this generator creates arcs by connecting two randomly selected nodes (no parallel arcs are allowed). It proceeds similarly to create commodities. Costs, capacities, and demands are then generated, uniformly distributed over userprovided intervals. Capacities and costs can then be scaled to obtain networks with various degrees of capacity tightness and relative importance of fixed costs. Two ratios are used for this purpose: the *capacity ratio* $C = |A|T/\sum_{(i,j)\in A} u_{ij}$ and the *fixed cost ratio* $F = |K|\sum_{(i,j)\in A} f_{ij}/T\sum_{k\in K}\sum_{(i,j)\in A} c_{ij}^k$, where $T = \sum_{k\in K} d^k$. Capacities and fixed costs are adjusted so that these ratios come close to user-provided values. In general, when C approaches 1, the network is lightly capacitated and becomes more congested as C increases. When F is close to 0, the fixed costs are low compared to the transportation costs, while their relative importance increases with F.

The instances are divided into three classes. Class I consists of instances with many commodities (significantly more than the number of nodes), while Class II is made of instances with few commodities (usually less than the number of nodes). Class III instances have been specifically generated to make problem characteristic versus performance analyses easier. They are divided into two subclasses, III-A and III-B, corresponding to 10-node and 20-node problem instances, respectively. Nine networks are generated in each subclass, by combining three arc-densities, roughly 25%, 50% and 75%, with three commodity-densities, roughly 10%, 25% and 50% (the density is the ratio with respect to $|N||N-1|$). For each of these nine networks, nine problem instances are created by combining three values of F – 0.01, 0.05, and 0.1 – with three values of C: 1, 2, and 8. Several problem instances were also created for each network in classes I and II to represent various F and C ratio values. Table 1 summarizes the characteristics of the 196 problem instances in the three classes according to problem

Table 1
Classification of instances according to problem dimension

Class I	(31)	Class II	(12)	Class III-A	(72)	Class III-B	(81)
20,230,40	(3)	25,100,10	(3)	10,35,10	(6)	20,120,40	(9)
20,230,200	(4)	25,100,30	(3)	10,35,25	(6)	20,120,100	(9)
20,300,40	(4)	100,400,10	(3)	10,35,50	(6)	20,120,200	(9)
20,300,200	(4)	100,400,30	(3)	10,60,10	(9)	20,220,40	(9)
30,520,100	(4)			10,60,25	(9)	20,220,100	(9)
30,520,400	(4)			10,60,50	(9)	20,220,200	(9)
30,700,100	(4)			10,85,10	(9)	20,320,40	(9)
30,700,400	(4)			10,85,25	(9)	20,320,100	(9)
				10,85,50	(9)	20,320,200	(9)

dimension represented by a triplet $|N|, |A|$, and $|K|$. The number of instances is displayed between parentheses (infeasible problem instances have been discarded). The test problems and the generator used for class III can be obtained from the authors.

The next subsection presents the calibration results of each relaxation method on the three classes of instances, while the second is dedicated to comparative analyses. Two performance measures are used:

- The *GAP* $(Z_* - Z)/Z_*$ between the lower bound Z obtained by the given method and the best available lower bound Z_*. This last corresponds to the optimal value of the strong linear programming relaxation except for some difficult problem instances for which technological limitations (indicated in Section 4.2) do not permit its computation.
- The *CPU* time on a Sun Ultra 1/140 workstation (4.66 SPECint95, 7.90 SPECfp95) with 64 MB of RAM memory. The code is programmed in C++ and compiled with the CC compiler using the -O option.

4.1. Calibration

The methods were fine-tuned for each class independently through the analysis of a large number of strategies and parameter sets. However, the available space does not allow a detailed presentation of the results. In the following, we summarize the main findings and present a number of aggregated performance measures that support them. A larger set of analyses and aggregated measures can be found in [7] (the complete experimental results can be obtained from the authors). The aggregated measures are obtained through averaging the performance measures over all instances in a class. Note that, although averaging over such large sets of instances with so varied characteristics might appear coarse, it proves remarkably reliable, as more detailed instance-by-instance analyses have revealed similar results and tendencies, whatever method is used. Also note that the maximum number of iterations for both the subgradient and the bundle methods was fixed to 500. Such a number allows both methods to come reasonably close to the optimal value for most instances, although it is important to mention that it could have been significantly diminished for the bundle, as it usually converges much more rapidly than the subgradient (see Section 4.2).

4.1.1. Shortest path–bundle method

The development of a bundle-based method for the shortest path relaxation requires the careful management of the bundle size due to the large number of multipliers $(|A| + |A||K|)$ generated by this relaxation. Indeed, in the course of our experiments, it became rapidly clear that discarding "inactive" subgradients is not enough for an efficient resolution, and that one is also required to fix the maximum size of the bundle to a relatively small value. When using a "conservative" value for the maximum bundle size (such as 100), all the RAM memory was quickly consumed for large-scale problem instances. This resulted in the need to access secondary memory devices, yielding prohibitive computation times.

An interesting question then arises: what is the tradeoff between numerical accuracy and computation time when the maximum bundle size varies? Other interesting issues that we aim to address with our experiments concern the impact on solution quality and computation performance resulting from the utilization of: (1) the *projected* shortest path relaxation rather than the "ordinary" one; (2) the *variable generation strategy* (e.g., on-line creation and destruction of multipliers); (3) the *long-term t-strategies*.

For all variants of the shortest path–bundle strategy that we tested, the parameters of the bundle method, such as m_1, m_2 and, especially, t^0, have been calibrated extensively for each class of instances. Typically, $m_1 = 0.1$, $m_2 = 3.0$ ($m_2 = 0.9$ when the long term t-strategy is used), and $t^0 = 1$ is a very good setting for all instances, even though $t^0 = 0.1$ leads to slightly better results for problems in class I. It is noteworthy, however, that other "reasonable" settings of t^0 (such as $t^0 = 0.01$ or 10) lead to very similar results: this is a clear demonstration of the robustness of the bundle method.

Several interesting conclusions emerged from our calibration experiments. First, note that 400-commodity instances in class I cannot be solved when the bundle size is fixed to 100 because the RAM memory limitation is quickly exceeded (the same occurs when a straightforward simplex-based approach is used to solve the strong linear programming relaxation, as explained in Section 4.2). Even some 200-commodity instances require the use of secondary memory devices. All instances can be treated, however, when the maximum bundle size is set to 10, with no significant impact on convergence: the gaps slightly increase on average, but remain in the same order of magnitude (10^{-4} for problems in classes I and II, and 10^{-3} for class III instances). The CPU times, however, are reduced significantly (by 37%, on average).

Another important conclusion concerns the projected version of the shortest path relaxation, which significantly and consistently improves over the "ordinary" version. For a slightly smaller computational effort, the projected version decreases the gap by an order of magnitude, from 10^{-3} to 10^{-4} for problems in classes I and II, and from 10^{-2} to 10^{-3} for class III instances. This behavior was constantly confirmed when other parameter settings were selected, for bundle and subgradient methods alike.

The comparisons also show the computational benefits of the variable generation strategy: for basically the same accuracy, CPU times are reduced on average by 64%, 47%, and 62% for problems in classes I, II, and III, respectively. Finally, one may observe that the hard long-term t-strategy is slightly superior to the soft variant and generally fastens convergence (other computational evidence is given in [13]). Therefore, the most promising variant of the shortest path–bundle method that we selected for further comparisons makes use of the projected version with a maximum bundle size of 10, plus the variable generation strategy and the hard long-term t-strategy.

4.1.2. Shortest path–subgradient method

We have already indicated that the projected shortest path relaxation is much more effective than the ordinary one. Hence, three main issues remain when designing

subgradient-based methods for the shortest path relaxation:

- The selection of the rule for computing the direction, Crowder or Camerini–Fratta–Maffioli, and the adjustment of the corresponding parameter, θ or μ.
- The adjustment of the stepsize, in particular:
  1. the setting of λ^0;
  2. the adjustment of λ^i; after testing several settings, one of the most promising we found is to halve λ^i after 10 consecutive iterations without improvement, with λ^i kept larger than some minimum, fixed to 10^{-3}; this setting was adopted for the remaining tests;
  3. the determination of the estimate $\bar{\phi}^i$; we have tried an upper bound provided by a tabu search heuristic [8], but the rough estimate $2 \times \phi^i_*$ was preferable in most cases.
- The choice between *projecting* the direction onto the nonnegative orthant (as in [28]) or not.

One of the biggest surprises we encountered when running the experiments was the performance of the modified Camerini–Fratta–Maffioli rule. It is not tested in [5], but only mentioned as a promising alternative to formula (21). To the best of our knowledge, it has been very seldom used since. In our experiments, it completely outperformed the Camerini–Fratta–Maffioli rule with the suggested setting $\mu = 1.5$ [5]. It is also competitive with Crowder rule, as it displays very similar gap and CPU results, with a slight edge for the modified Camerini–Fratta–Maffioli rule. Note that this conclusion is true whether the direction is projected or not.

Another important conclusion from our calibration phase concerns the projection of the direction. It appears that for both Crowder and modified Camerini–Fratta–Maffioli rules, projecting the direction improves the gap by an order of magnitude, for essentially the same computational effort: from an order to 10^{-3} to 10^{-4} for problems in classes I and II, and from 10^{-2} to 10^{-3} for class III instances. It is interesting to note that the two options, projecting the direction or not, require different "optimal" settings of the parameters.

We expected to run into the well-known "jamming" phenomenon: the method is unable to improve on the value of the objective function and, consequently, the parameter λ^i is continually decreased. The result is a series of very short steps, which makes the method "jam" far away from an optimal solution. Indeed, most variants we tested suffer to some degree from this problem and display a similar tendency to "diverge". In fact, the experiments illustrated what can happen when a subgradient method suggested in the literature is run without carefully adjusting the parameters. For example, using the Camerini–Fratta–Maffioli rule with the suggested $\mu = 1.5$, without projecting the direction and calibrating λ^0 (1.0 was determined to be the most promising for all classes), produced average gaps of two orders of magnitude larger than those obtained by the best finetuned variants. This is a clear demonstration of the *lack of robustness* of the subgradient method: "good" settings can be found, but most "reasonable" settings one might try are unbearably "bad". Following our experiments, we selected the following variant of the shortest path–subgradient method for further

comparisons: use the modified Camerini–Fratta–Maffioli rule by projecting the direction and $\lambda^0 = 1.0$.

4.1.3. Knapsack–bundle method

Compared to the shortest path relaxation, the knapsack relaxation has a relatively small number of multipliers ($|N||K|$). Consequently, more flexibility is allowed in the development of a bundle method for the optimization of its Lagrangian dual. In particular, the variable generation strategy is generally slightly inferior in this case: it slows down convergence, as usual, but also sometimes increases computation time. Keeping low the maximum bundle size is not necessary either. However, it is still interesting to qualify the tradeoff between accuracy and computation time when the maximum bundle size varies: a small maximum bundle size reduces the burden of solving the QP at each iteration, which might significantly improve the efficiency, at the cost of losing some effectiveness. Another interesting issue is the effect of the long-term t-strategies, shown to be marginally superior for the shortest path-bundle method.

In light of these remarks, we experimented with bundle sizes of 100 and 10, implementing or not the hard long-term t-strategy (again shown to be superior to the soft one). The parameters of the bundle method were calibrated for each class of problem instances. It turned out that the values selected as the most promising were the same as in the case of the shortest path–bundle method. Furthermore, other "reasonable" settings displayed very similar results. This is another indication of the robustness of the bundle method.

The size of the bundle does not appear to influence significantly the quality of the solution for problems in classes I and III where the average gaps (in the order of 10^{-4} and 10^{-3}, respectively) are almost identical for all variants. The difference is only a little bit more significant for class II problems: the average gaps vary from approximately 8×10^{-4} to 1×10^{-3} when the maximum bundle size is decreased from 100 to 10. Overall, these results show that the method is capable of generating high-quality solutions even when the maximum bundle size is significantly reduced. On the other hand, computation times are cut by a factor of 2 for problems in classes I and III, and almost 4 for class II instances. Similarly to the shortest path–bundle method, the long-term t-strategy slightly improves the performances of the bundle method. As it represents a fair balance between numerical precision and computation time, we selected for further comparisons a strategy with a low maximum bundle size (10), using the hard long-term t-strategy.

4.1.4. Knapsack–subgradient method

In light of previous experience, making the subgradient method converge when applied to the knapsack relaxation appeared to be a difficult task. Indeed, the authors presented in [17] disappointing results for the knapsack–subgradient method: for most instances, the method was "diverging" to the point that the bound it computed was worse than the weak linear programming bound. In [26], better results are obtained

with a similar procedure which uses a variant of Crowder rule for updating the direction. Following our present computational experiments, we can claim that the fundamental difference between the procedures described in [17,26] lies in the way the multipliers are initialized: in [17], they are set to 0, while in [26], they assume the values of the node potentials obtained after solving $|K|$ shortest path problems with costs $c_{ij}^k \ \forall (i,j) \in A, \ k \in K$ (actually, a convex combination of 0 and the vector of node potentials is used). In fact, prior to the publication of [17], a similar *warmstart* procedure was tested but no improvement of the bound was observed. Based on our latest computational experiments, we can attribute this negative result to a bad adjustment of the parameters, especially λ^0.

For the present tests, we have used a warmstart procedure where the multipliers are initialized to the values of the node potentials resulting from the solution of $|K|$ shortest path problems with costs $c_{ij}^k + f_{ij}/u_{ij} \ \forall (i,j) \in A, k \in K$ (a choice experimentally shown better than the above one). Then, by carefully adjusting the parameters, we were able to obtain very accurate results. Without this warmstart procedure, however, we were unable to make the method converge, irrespective of the parameter setting used (thus confirming the bad results reported in [17]). Note that the same warmstart procedure was tested for the knapsack–bundle method, but then shown to be inferior to the straightforward initialization of the multipliers. A similar warmstart method, based on solving continuous knapsack problems, was also tested for the shortest path relaxation: it did not improve neither the bundle nor the subgradient.

The results from this calibration phase support our previous conclusions about the subgradient method: (1) the Camerini–Fratta–Maffioli rule with $\mu = 1.5$ is consistently outperformed by other rules; (2) the modified Camerini–Fratta–Maffioli and Crowder rules are both competitive, with a slight edge for the former. The variant selected for further comparisons thus makes use of the modified Camerini–Fratta–Maffioli rule with λ^0 adjusted for each class: $\lambda^0 = 1.0$ for class I, $\lambda^0 = 0.7$ for class II, and $\lambda^0 = 0.3$ for class III. The remaining issues, i.e., the adjustment of λ^i and the determination of the estimate $\bar{\phi}^i$, are settled in the same way as for the shortest path–subgradient method.

It is important to point out that the relatively good results we obtained for the knapsack–subgradient method (average gaps of the order of 10^{-4} for class I and 10^{-3} for the others) are due to a very careful adjustment of λ^0 for each strategy and class of problems. Values different than the ones mentioned above might lead to disastrous results. For example, when using the modified Camerini–Fratta–Maffioli rule for class III, if $\lambda^0 = 0.7$ is used instead of $\lambda^0 = 0.3$, an average gap of the order of 10^{-1} is obtained. This further illustrates the lack of robustness of the subgradient method.

4.2. Comparison

In this section, we compare the most promising variants of the four relaxation methods identified in Section 4.1, as well as three other bounding procedures. We characterize the performances of the methods with respect to problem dimension and relative importance of capacities and fixed costs (measured by the F and C ratios). The

bounding procedures are:

- *CPXW*: It solves the weak linear programming relaxation formulated as an MMCF problem (with costs $c_{ij}^k + f_{ij}/u_{ij}\ \forall (i,j) \in A, k \in K$), by a well-known and competitive simplex solver, CPLEX (version 4.0). It uses the "netopt" option, which provides a good starting basis by solving the minimum cost network flow problem resulting from the relaxation of the capacity constraints, and then switches to the dual simplex method.
- *WB*: It solves the Lagrangian dual of the projected weak shortest path relaxation (with $\beta = 0$) by the same bundle method used for other relaxations. As explained in Section 2, the optimal value of this Lagrangian dual gives the same bound as the weak linear programming relaxation.
- *CPXS*: It solves the strong linear programming relaxation with CPLEX using the "netopt" option. It is undoubtedly a very straightforward approach (e.g., a cutting-plane procedure based on relaxing the forcing constraints and introducing them gradually in the formulation would certainly be more efficient). However, it illustrates the difficulty of solving these large-scale formulations by using standard methods, and highlights the degeneracy issue faced by all simplex-based methods.
- *SS*: The most promising variant of the shortest path–subgradient method identified in the calibration phase (Section 4.1).
- *SB*: The most promising variant of the shortest path–bundle method identified in the calibration phase (Section 4.1).
- *KS*: The most promising variant of the knapsack–subgradient method identified in the calibration phase (Section 4.1).
- *KB*: The most promising variant of the knapsack–bundle method identified in the calibration phase (Section 4.1).

Tables 2–5 display the results obtained by these methods when applied to all problem instances in the three classes. Problems are grouped according to their dimension, the number of instances in each group being written between parentheses. The first figure indicates the average GAP (an "X" signals that the corresponding method was unable to solve the instances in the group) with respect to the best value obtained, while the figure below shows the average CPU time.

These results demonstrate that the weak bound is really weak and certainly not viable as a basis for branch-and-bound methods, since it displays an average gap of roughly 20% with respect to the strong bound. However, it is interesting to note that method WB converges *exactly* (within a tolerance of $\varepsilon = 10^{-6}$) to the weak bound for all instances and it is up to an order of magnitude faster than a standard simplex-based method for large-scale MMCF problems (other computational evidence is given in [15]). Since the bundle method is capable of extracting an *optimal primal solution* once it has converged [15], this is especially appealing in the context of designing heuristics for our problem, since finding a feasible solution amounts to solving an MMCF problem.

As the number of commodities increases, the resources – time and memory – required by method CPXS become prohibitive, to the point that 400-commodity instances cannot

Table 2
Method comparisons with respect to problem dimension – Class I

Problems	CPXW	WB	CPXS	SS	SB	KS	KB
20,230,40	6.9e-2	6.9e-2	0.0	5.9e-5	7.0e-5	1.2e-4	6.7e-5
(3)	0.5	0.1	4.9	9.9	10.7	2.5	3.7
20,230,200	2.4e-1	2.4e-1	0.0	1.8e-3	6.7e-4	9.7e-4	1.4e-3
(4)	2.7	0.3	2448.3	56.6	63.6	24.9	29.7
20,300,40	8.8e-2	8.8e-2	0.0	2.4e-4	7.9e-5	2.0e-4	7.7e-5
(4)	0.7	0.1	7.2	12.7	11.7	3.2	3.7
20,300,200	1.9e-1	1.9e-1	0.0	1.1e-3	4.0e-4	6.0e-4	6.1e-4
(4)	4.6	0.5	10298.1	70.8	74.6	29.0	32.1
30,520,100	1.7e-1	1.7e-1	0.0	8.8e-4	9.3e-4	1.0e-3	9.5e-4
(4)	4.6	0.7	1086.1	62.9	64.1	17.8	21.1
30,520,100	1.4e-1	1.4e-1	X	6.6e-4	1.9e-5	1.1e-4	3.8e-4
(4)	63.8	6.5	11112.8	264.8	257.6	139.9	166.1
30,700,100	1.6e-1	1.6e-1	0.0	9.2e-4	5.2e-4	7.3e-4	8.8e-4
(4)	5.1	0.9	857.5	82.3	83.1	22.5	24.9
30,700,400	1.6e-1	1.6e-1	X	6.5e-4	0.0	1.5e-4	4.0e-4
(4)	76.9	9.3	11700.6	343.8	334.7	179.8	199.3
Average	1.6e-1	1.6e-1	0.0	8.0e-4	3.4e-4	5.0e-4	6.1e-4
(31)	20.5	2.4	4840.5	116.3	115.8	54.1	61.9

Table 3
Method comparisons with respect to problem dimension – Class II

Problems	CPXW	WB	CPXS	SS	SB	KS	KB
25,100,10	2.3e-1	2.3e-1	0.0	5.3e-4	1.8e-4	7.6e-4	2.7e-4
(3)	0.1	0.0	1.3	1.1	1.0	0.6	0.8
25,100,30	2.2e-1	2.2e-1	0.0	4.0e-4	1.4e-4	9.8e-4	5.5e-4
(3)	0.6	0.2	11.3	3.0	3.5	1.2	2.3
100,400,10	2.8e-1	2.8e-1	0.0	1.1e-3	6.7e-4	1.6e-3	1.3e-3
(3)	0.3	0.1	35.9	4.2	4.8	1.7	3.0
100,400,30	2.9e-1	2.9e-1	0.0	1.0e-3	1.1e-3	1.9e-3	2.5e-3
(3)	5.9	2.3	351.9	14.3	16.7	4.5	9.1
Average	2.5e-1	2.5e-1	0.0	7.6e-4	5.1e-4	1.3e-3	1.2e-3
(12)	1.7	0.7	100.1	5.7	6.4	2.0	3.8

even be solved due to lack of memory. In comparison, our implementations of the relaxation methods are very efficient, especially on large-scale problem instances, where they run in a fraction of the time taken by CPXS, provided the size of the bundle is controlled (as mentioned above, if a maximum bundle size of 100 is used, the shortest path–bundle method fall into the same memory problems as CPXS).

The knapsack-based methods, due to the simplicity of the Lagrangian subproblem, are approximately 2–3 times faster than the shortest path-based methods. Note, however, that method SB is remarkably effective, as it displays the best average gaps over all classes, except class III-B. The very good performances of procedure KS came as a surprise: it is the fastest and its average gaps are very comparable to those obtained by method KB. It should be pointed out, however, that strategy KB could offer a better performance if less iterations (say 100) were allowed and the maximum bundle size was

Table 4
Method comparisons with respect to problem dimension – Class III-A

Problems	CPXW	WB	CPXS	SS	SB	KS	KB
10, 35, 10	9.7e-2	9.7e-2	0.0	5.6e-4	0.0	2.2e-4	1.3e-5
(6)	0.0	0.0	0.1	0.6	0.3	0.3	0.3
10, 35, 25	1.6e-1	1.6e-1	0.0	1.6e-4	6.0e-5	3.3e-4	1.3e-4
(6)	0.1	0.0	0.4	1.1	1.0	0.5	0.9
10, 35, 50	1.4e-1	1.4e-1	0.0	3.4e-4	1.2e-4	9.4e-4	5.7e-4
(6)	0.2	0.1	2.0	2.1	2.3	0.9	1.6
10, 60, 10	1.7e-1	1.7e-1	0.0	7.4e-4	3.5e-5	1.9e-4	5.1e-5
(9)	0.0	0.0	0.2	0.8	0.6	0.3	0.4
10, 60, 25	1.3e-1	1.3e-1	0.0	8.3e-4	5.3e-4	8.5e-4	5.3e-4
(9)	0.1	0.1	1.3	1.6	1.6	0.7	1.1
10, 60, 50	1.7e-1	1.7e-1	0.0	1.1e-3	1.1e-3	1.7e-3	1.2e-3
(9)	0.3	0.1	6.5	3.1	3.5	1.2	2.0
10, 85, 10	1.3e-1	1.3e-1	0.0	4.1e-4	2.0e-5	1.4e-4	2.8e-5
(9)	0.0	0.0	0.2	0.9	0.7	0.4	0.4
10, 85, 25	1.6e-1	1.6e-1	0.0	7.1e-4	1.7e-4	3.7e-4	1.1e-4
(9)	0.1	0.1	1.6	2.1	1.7	0.8	1.0
10, 85, 50	2.0e-1	2.0e-1	0.0	7.6e-4	4.2e-4	5.1e-4	3.2e-4
(9)	0.3	0.1	10.2	4.1	4.6	1.5	2.2
Average	1.5e-1	1.5e-1	0.0	6.6e-4	2.9e-4	5.9e-4	3.4e-4
(72)	0.1	0.1	2.7	1.9	2.0	0.8	1.1

Table 5
Method comparisons with respect to problem dimension – Class III-B

Problems	CPXW	WB	CPXS	SS	SB	KS	KB
20, 120, 40	1.8e-1	1.8e-1	0.0	4.6e-3	5.3e-3	2.1e-3	1.5e-3
(9)	1.0	0.4	24.3	4.8	5.6	1.7	2.9
20, 120, 100	1.8e-1	1.8e-1	0.0	1.1e-3	1.3e-3	2.3e-3	1.5e-3
(9)	8.0	1.6	217.2	12.9	15.1	5.1	8.5
20, 120, 200	1.6e-1	1.6e-1	0.0	6.2e-3	9.9e-4	1.5e-3	1.7e-3
(9)	35.8	4.5	1164.6	26.2	30.3	13.9	22.3
20, 220, 40	2.7e-1	2.7e-1	0.0	1.7e-3	1.3e-3	1.3e-3	1.0e-3
(9)	0.5	0.1	63.1	9.3	11.3	2.5	3.7
20, 220, 100	2.5e-1	2.5e-1	0.0	9.3e-3	8.8e-3	8.7e-3	8.8e-3
(9)	3.1	0.7	558.1	24.4	28.1	7.5	10.8
20, 220, 200	2.2e-1	2.2e-1	0.0	1.3e-3	8.7e-4	1.3e-3	1.4e-3
(9)	40.5	6.1	3572.9	50.3	55.5	19.7	27.3
20, 320, 40	3.0e-1	3.0e-1	0.0	8.3e-3	1.0e-2	2.1e-3	2.8e-3
(9)	0.6	0.1	102.8	13.4	15.9	3.3	4.6
20, 320, 100	2.8e-1	2.8e-1	0.0	2.5e-3	2.2e-3	1.4e-3	1.4e-3
(9)	2.2	0.3	999.5	36.8	41.3	10.6	14.3
20, 320, 200	2.5e-1	2.5e-1	0.0	2.0e-3	1.1e-3	1.0e-3	1.1e-3
(9)	12.0	2.7	5789.3	75.8	79.6	26.0	35.6
Average	2.3e-1	2.3e-1	0.0	4.1e-3	3.6e-3	2.4e-3	2.4e-3
(81)	11.5	1.9	1388.0	28.2	31.4	10.0	14.5

increased (say to 100). In fact, for most instances, the bundle method achieves most progress in the first iterations, while the last iterations only bring a minor contribution in terms of the bound quality. The subgradient method has a significantly different behavior, as it converges much more slowly (this point is further substantiated below).

The relative gaps displayed in these tables point to the conclusion that, once properly calibrated for each method and problem class, subgradient-based methods offer similar performances in terms of solution quality over the whole range of problem dimensions. This observation confirms that reported by Gondran and Minoux [23] relative to implementations using the settings proposed in [24]. Our results also show, however, that the same apparent insensitiveness to problem dimension characterizes bundle-based implementations as well (further theoretical and experimental investigations into this matter are required but fall outside the scope of the present paper).

We have also analyzed the performances of the seven methods with respect to the importance of fixed costs and capacities. The results (detailed in [7]) show that the gap between the weak and the strong bounds increases when fixed costs get higher and capacities less tight. The weak bound can, in general, be easily computed, but it is even more easier to compute for instances with large F and small C. This is true for both CPXW and WB methods, but WB is significantly faster. For the CPXS procedure, the CPU time increases not only with the importance of the fixed costs, but also with the tightness of the capacities. Our relaxation methods generally exhibit larger gaps when fixed costs are more important. No clear tendency emerges however with respect to the capacity ratio. Overall, these results demonstrate that the performances of our implementations of Lagrangian relaxation methods are much less dependent on the relative importance of fixed costs and capacities than simplex-based methods.

The previous analyses allow to qualify the performances of the various methods with respect to different problem characteristics and to discriminate between relaxation and simplex-based approaches. They may be misleading, however, in suggesting that the four relaxation methods are more or less "equivalent": for almost all instances, their relative accuracy is similar (with an edge for method SB) and their computation times improve significantly over a straightforward simplex implementation (method KS being the fastest). However, these results are obtained with a fixed maximum number of iterations of 500 for all methods, a maximum that is almost always attained: hence, they give no indication of the *speed of convergence*. We illustrate the performance of the methods with respect to this characteristic by using two representative instances: one in class I of dimension 20,300,200 and one in class II of dimension 100,400,30. For each problem instance, Figs. 1 and 2 show the evolution of the bound with time (in CPU seconds) for the four relaxation strategies, as well as for method CPXS (each dot in the figure corresponds to an iteration of the method). For the first instance, it is clear that the two bundle methods converge faster than the subgradient approaches, the fastest being SB with KB a close second, while SS is by far the slowest of the four. For the second instance, the hierarchy is about the same, but this time the two knapsack-based relaxations are closer to each other. The figures also highlight the degeneracy problem faced by method CPXS: for both instances, the objective remains

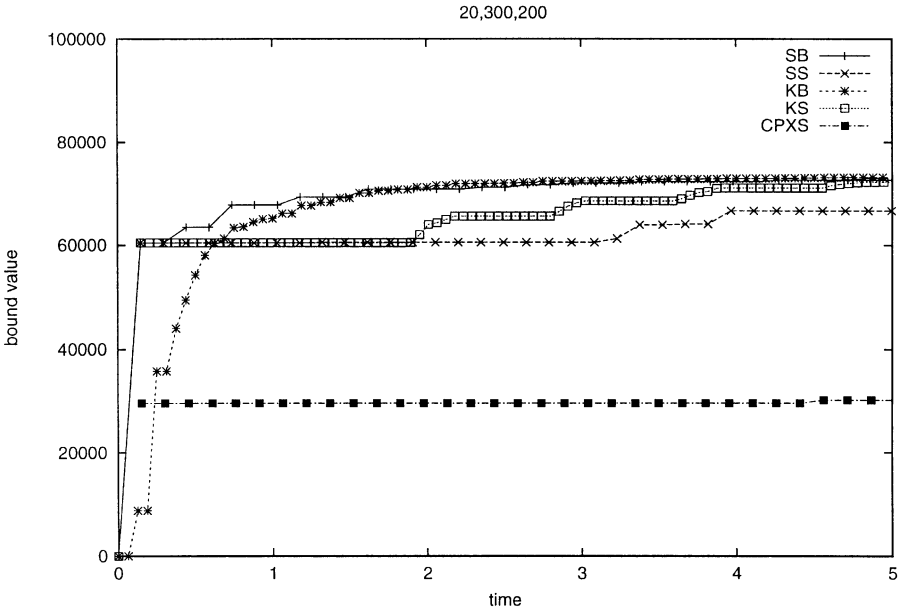

Fig. 1. Comparison of the speed of convergence for one problem in class I.

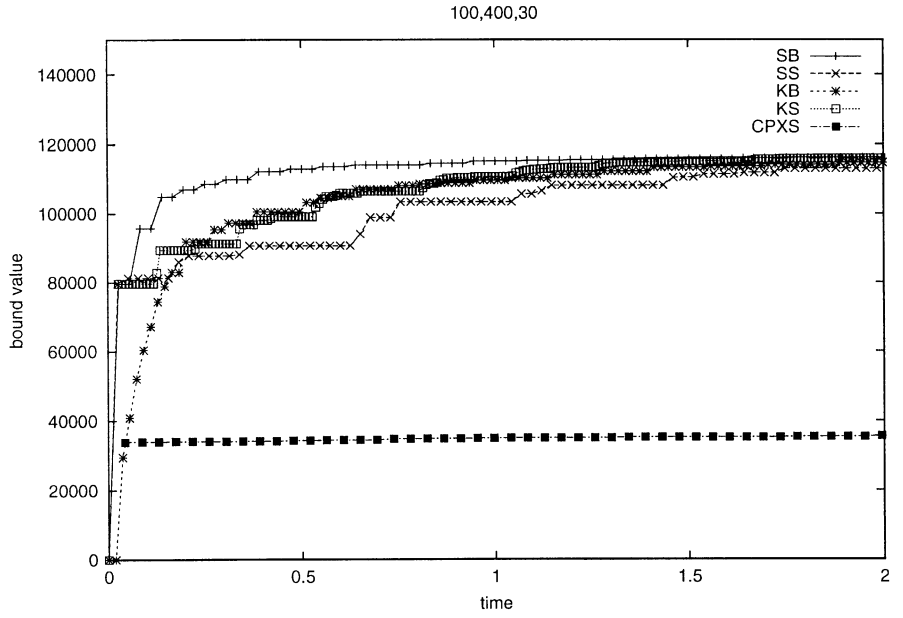

Fig. 2. Comparison of the speed of convergence for one problem in class II.

the same for several consecutive iterations. Consequently, the method converges very slowly; the first instance took 1742 s to reach the optimal value, the second one 408 s.

5. Conclusion

We have presented and analyzed the results of a comprehensive study of Lagrangian-based bounding methods for capacitated fixed-charge network design problems. The relaxation methods differed by the dual formulation solved (based on relaxing the flow or the forcing constraints), by the nondifferentiable optimization approach used (subgradient or bundle), as well as by the choice of a number of important rules and parameters. Linear programming relaxations, solved by a competitive commercial simplex code, were also included in our study, in order to correctly characterize the performances of the various methods. Experimentation has been performed on a large set of test problems of various characteristics.

The calibration phase has allowed to gain precious insights into the behavior of each method for the class of instances studied, to single out critical parameters, and to identify recommended strategies and parameter settings. The experiments have also allowed to substantiate the fact that, although more complex to implement, bundle methods appear superior to subgradient approaches: they converge faster and are more robust relative to different relaxations, problem characteristics, and selection of the initial parameter values. Note that our codes are based on generic C++ bundle and subgradient classes that have been used in several projects by different research groups. Although they are not yet in the public domain, they can be obtained from the second author for controlled experiments.

As mentioned in Section 3.2, our bundle implementation uses the standard quadratic "stabilizing" device. Other choices are possible however. For instance, a generalization of standard bundle methods has been recently proposed [14], which allows the "stabilizing" device to be chosen from a large class of functions. Other methods from the field of nondifferentiable optimization can also be used in the context of Lagrangian relaxation, the most notable being the proximal level method [25] and the analytic center cutting-plane method [22]. In particular, the latter has been used successfully in two applications closely related to ours: linear and nonlinear MMCF [21,11] and capacitated lot-sizing [10].

Our study has demonstrated that effective lower bounds may be computed efficiently for large-scale instances of the capacitated fixed-charge network design problem. Indeed, in a fraction of the time required by a standard simplex approach to solve the linear programming relaxation (if it may be solved at all), the methods we presented attain very high-quality solutions. This should prove of prime importance when optimal solutions are sought for by branch-and-bound methods.

Several fascinating research avenues are now inviting. One is the integration of strong valid inequalities into the relaxations. To preserve the structure of the Lagrangian sub-problems, these valid inequalities could be relaxed in a Lagrangian way, which

would, however, introduce large quantities of multipliers and subgradients related to these constraints. Our implementation of the bundle method has two features that allow an efficient treatment of such relaxations: (1) the size of the bundle of subgradients can be limited by the user; (2) it is possible to restrict the direction-finding subproblem to the set of multipliers corresponding to violated constraints, which can be revised dynamically in an efficient way. Our experiments show indeed that these two features were essential to obtain good results for the shortest path relaxation. Another issue concerns the determination of tight feasible solutions; in particular, bundle-based Lagrangian heuristics appear promising. Finally, the combination of relaxation methods and heuristics with reoptimization, variable fixing techniques and branch-and-bound procedures constitutes the goal of one of our major research trusts.

Acknowledgements

We want to acknowledge the efforts of Benoît Bourbeau who helped us in testing our code. We also want to thank two anonymous referees: their comments helped as improve the presentation of the paper. Financial support for this project was provided by N.S.E.R.C. (Canada) and the Fonds F.C.A.R. (Québec). In particular, Antonio Frangioni's sojourn at the Centre for Research on Transportation in Montréal was supported by the *Network for Computing and Mathematical Modeling* from its N.S.E.R.C. partnership grant.

References

[1] E. Allen, R. Helgason, J. Kennington, B. Shetty, A generalization of Polyak's convergence result for subgradient optimization, Math. Programming 37 (1987) 309–317.
[2] A. Balakrishnan, T.L. Magnanti, P. Mirchandani, Network design, in: M. Dell'Amico, F. Maffioli, S. Martello (Eds.), Annotated Bibliographies in Combinatorial Optimization, Wiley, New York, 1997, pp. 311–334 (Chapter 18).
[3] A. Balakrishnan, T.L. Magnanti, A. Shulman, R.T. Wong, Models for planning capacity expansion in local access telecommunication networks, Ann. Oper. Res. 33 (1991) 239–284.
[4] D. Bienstock, O. Günlük, Capacitated network design – polyhedral structure and computation, INFORMS J. Comp. 8 (3) (1996) 243–259.
[5] P.M. Camerini, L. Fratta, F. Maffioli, On improving relaxation methods by modified gradient techniques, Math. Programming Study 3 (1975) 26–34.
[6] S.-G. Chang, B. Gavish, Lower bounding procedures for multiperiod telecommunications network expansion problems, Oper. Res. 43 (1) (1995) 43–57.
[7] T.G. Crainic, A. Frangioni, B. Gendron, Bundle-based relaxation methods for multicommodity capacitated fixed charge network design problems, Publication CRT-98-45, Centre de recherche sur les transports, Université de Montreal, 1998.
[8] T.G. Crainic, M. Gendreau, J. Farvolden, Simplex-based tabu search for the multicommodity capacitated fixed charge network design problem, INFORMS J. Comput. 12(3) (2000).
[9] H. Crowder, in: Computational improvements for subgradient optimization, Symposia Mathematica, Vol. XIX, Academic Press, London, 1976.
[10] O. du Merle, J.-L. Goffin, C. Trouiller, J.-P. Vial, A Lagrangian relaxation of the capacitated multi-item lot sizing problem solved with an interior point cutting plane algorithm, Research Report 97-5, Logilab, University of Geneva, 1998.

[11] O. du Merle, J.-L. Goffin, J.-P. Vial, On improvements to the analytic center cutting plane method, Comput. Optim. Appl. 11 (1998) 37–52.
[12] A. Frangioni, Solving semidefinite quadratic problems within nonsmooth optimization algorithms, Comput. Oper. Res. 23 (1) (1996) 1099–1118.
[13] A. Frangioni, Dual-ascent methods, and multicommodity flow problems, Ph.D. Thesis, Dipartimento di informatica, Università di Pisa, 1997.
[14] A. Frangioni, Generalized bundle methods, Technical Report TR-98-04, Dipartimento di informatica, Universitá di Pisa, 1998.
[15] A. Frangioni, G. Gallo, A bundle type dual-ascent approach to linear multicommodity min cost flow problems, INFORMS J. Comp. 11 (4) (1999) 370–393.
[16] B. Gavish, Topological design of telecommunications network – local access design methods, Ann. Oper. Res. 33 (1991) 17–71.
[17] B. Gendron, T.G. Crainic, Relaxations for multicommodity capacitated network design problems, Publication CRT-965, Centre de recherche sur les transports, Université de Montréal, 1994.
[18] B. Gendron, T.G. Crainic, Bounding procedures for multicommodity capacitated fixed charge network design problems, Publication CRT-96-06, Centre de recherche sur les transports, Université de Montréal, 1996.
[19] B. Gendron, T.G. Crainic, A. Frangioni, Multicommodity capacitated network design, in: B. Sansó, P. Soriano (Eds.), Telecommunications Network Planning, Kluwer Academics Publishers, Dordrecht, 1998, pp. 1–19.
[20] A.M. Geoffrion, Lagrangean relaxation for integer programming, Math. Programming Study 2 (1974) 82–114.
[21] J.-L. Goffin, J. Gondzio, R. Sarkissian, J.-P. Vial, Solving nonlinear multicommodity flow problems by the analytic center cutting plane method, Math. Programming 76 (1996) 131–154.
[22] J.-L. Goffin, A. Haurie, J.-P. Vial, Decomposition and nondifferentiable optimization with the projective algorithm, Management Sci. 38 (2) (1992) 284–302.
[23] M. Gondran, M. Minoux, Graphes et Algorithmes, Eyrolles, Paris, 1979.
[24] M. Held, P. Wolfe, H.P. Crowder, Validation of subgradient optimization, Math. Programming 6 (1974) 62–88.
[25] J.B. Hiriart-Urruty, C. Lemaréchal, in: Convex analysis and minimization algorithms II, A Series of Comprehensive Studies in Mathematics, Vol. 306, Springer, Berlin, 1993.
[26] K. Holmberg, D. Yuan, A Lagrangean heuristic based branch-and-bound approach for the capacitated network design problem, Research Report LiTH-MAT-R-1996-23, Department of Mathematics, Linköping Institute of Technology, Oper. Res. (1996) forthcoming.
[27] J.E. Kelley, The cutting-plane method for solving convex programs, J. SIAM 8 (1960) 703–712.
[28] T. Larsson, M. Patriksson, A.-B. Strömberg, Conditional subgradient optimization – theory and applications, European J. Oper. Res. 88 (1996) 382–403.
[29] C. Lemaréchal, Nondifferentiable optimization, in: G.L. Nemhauser, A.H.G. Rinnoy Kan, M.J. Todd (Eds.), Handbooks in Operations Research and Management Science, Chapter 7, Optimization, Vol. 1, 1989, pp. 529–572.
[30] T.L. Magnanti, P. Mirchandani, R. Vachani, Modeling and solving the two-facility capacitated network loading problem, Oper. Res. 43 (1) (1995) 142–157.
[31] T.L. Magnanti, R.T. Wong, Network design and transportation planning: models and algorithms, Transportation Sci. 18 (1) (1984) 1–55.
[32] M. Minoux, Network synthesis and optimum network design problems: models, solution methods and applications, Networks 19 (1989) 313–360.
[33] B.T. Polyak, Minimization of nonsmooth functionals, USSR Comput. Math. and Math. Phys. 9 (1969) 14–29.
[34] H. Schramm, J. Zowe, A version of the bundle idea for minimizing a nonsmooth function: conceptual idea, convergence analysis, numerical results, SIAM J. Optim. 2 (1) (1992) 121–152.