Open access • Journal Article • DOI:10.1007/BF02614381

Solving nonlinear multicommodity flow problems by the analytic center cutting plane method — Source link

Jean-Louis Goffin, Jacek Gondzio, Robert Sarkissian, Jean-Philippe Vial

Institutions: McGill University, University of Geneva

Published on: 02 Jan 1997 - Mathematical Programming (Springer-Verlag)

Topics: Minimum-cost flow problem, Multi-commodity flow problem, Flow network, Nonlinear programming and Column generation

Article

Solving nonlinear multicommodity flow problems by the analytic center cutting plane method

GOFFIN, Jean-Louis, *et al*.

Mathematical Programming 76 (1996) 131–154

Solving nonlinear multicommodity flow problems by the analytic center cutting plane method [1]

J.-L. Goffin [a,b,*,2], J. Gondzio [b,3], R. Sarkissian [b], J.-P. Vial [b]

[a] *GERAD, Faculty of Management, McGill University, Montreal, Que., Canada H3A 1G5*
[b] *Logilab, HEC, Section of Management Studies, University of Geneva, 102 Bd.Carl Vogt, CH-1211 Genève 4, Switzerland*

Received 1 November 1994; revised manuscript received 1 November 1995

Abstract

The paper deals with nonlinear multicommodity flow problems with convex costs. A decomposition method is proposed to solve them. The approach applies a potential reduction algorithm to solve the master problem approximately and a column generation technique to define a sequence of primal linear programming problems. Each subproblem consists of finding a minimum cost flow between an origin and a destination node in an uncapacited network. It is thus formulated as a shortest path problem and solved with Dijkstra's d-heap algorithm. An implementation is described that takes full advantage of the supersparsity of the network in the linear algebra operations. Computational results show the efficiency of this approach on well-known nondifferentiable problems and also large scale randomly generated problems (up to 1000 arcs and 5000 commodities).

1. Introduction

The nonlinear multicommodity flow problem with separable increasing convex costs gives rise to very large nonlinear programs with linear constraints. It arises in the areas

* Corresponding author. e-mail: ma56@musica.mcgill.ca

[1] This research has been supported by the Fonds National de la Recherche Scientifique Suisse, grant #12-34002.92, NSERC-Canada and FCAR-Quebec.

[2] This research was supported by an Obermann fellowship at the Center for Advanced Studies at the University of Iowa.

[3] On leave from the Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland.

PII S0025-5610(96)00037-8

of transportation networks, traffic assignment problems, telecommunication or computer networks, multi-item production planning, to name but a few applications. The decomposition *principle* of Dantzig and Wolfe [8] is well suited to the primal block diagonal structure of the constraints, even though the decomposition *algorithm* may show slow convergence on some, but not all, formulations or instances. The granularity of the formulation may have a significant impact on the effectiveness of the approach as noted and studied by Jones et al. [24] and Minoux [32].

Three issues need to be settled to make for an effective algorithm:

- The formulation: how many commodities? Origin–destination problem, destination specific (or-origin specific) problem or product specific problem, or in fact a combination of these three cases [24]?
- How to deal with the nonlinear objective?
- What solution concept to use in the master problem, to compute a set of prices that will be used to generate new proposals (columns) from the subproblems (oracles).

The approach taken here uses the following answers to these three issues:

- The formulation: as fine a granularity (disaggregation) as the problem will allow.
- Deal with the nonlinear objective by defining a subproblem (oracle, block, convexity constraint) for each one-dimensional component of the objective.
- Do not solve the master problem to optimality but compute an analytical center of the set of localization (ACCPM: the analytic center cutting plane method) [15,3].

As in the Dantzig–Wolfe algorithm, our approach considers a restricted master program. This restriction is improved as new columns are added, either one at a time or in bunches. The analytic center of the set of localization is obtained by applying to the restricted master the de Ghellinck and Vial's variant [9] of Karmarkar's projective algorithm [25].

The purpose of our paper is to demonstrate that ACCPM is a viable tool for solving large non-linear multicommodity flow problems. Special attention has been paid to exploit structural sparsity in the restricted master. We conducted two different series of experiments. We first compared different levels of granularity on two well documented test problems; the finest granularity yields the best results by far. The second series of experiments pertained to random problems of various sizes – the formulation used the finest granularity. The results are very encouraging. The method is robust. Its behavior on the particular class of problems is very much alike what has been observed in different areas of application such as stochastic programming, multiregional planning, nonlinear programming, minmax problems. Our results provide further evidence that ACCPM is one of the method of choice for nondifferentiable optimization, especially for the nondifferentiable algorithms that arise in decomposition approaches to large scale programming.

Our specific contribution in this paper is twofold:

- our first contribution is to show that a full disaggregation has a dramatic effect on convergence; by this we mean a disaggregation on the OD pairs – the rationale here being that few shortest paths must appear in the optimal solution – *and* on each of the nonlinear cost arcs – the rationale being that cutting planes can

approximate well functions of *one* variable. The impact of this disaggregation is a considerable improvement not only of the ACCPM but also of Dantzig–Wolfe.

- Second we propose an advanced implementation of ACCPM with this particular application in mind. We refrained from undertaking a systematic comparison with other methods. We do not claim that our method is the best, but we claim that it is always competitive with the best, and that is is stable and robust, and not prone to slowing down as it is the case with some formulations of Dantzig–Wolfe decomposition.

There are, of course many other approaches to the linear or nonlinear multicommodity flow problem, many of them based upon variants of Dantzig–Wolfe decomposition, or dually, Kelley's cutting plane method or Benders's decomposition, *all of which* can and have been interpreted in terms of nondifferentiable optimization.

The fully aggregated formulation (which corresponds to *one* convexity row) has been used in many nondifferentiable optimization approaches: the subgradient method by Fukushima [12], the proximal bundle method of Lemaréchal [30] and Kiwiel [28], the bundle trust region method of Schramm and Zowe [36], the new bundle method of Lemaréchal, Nemirovskii and Nesterov [31], the affine scaling with centering bundle method of Hipolito [23], the dual ascent algorithm of Hearn and Lawphongpanich [21], and the restricted simplicial decomposition of Hearn, Lawphongpanich, and Ventura [22], among many others. Most of these papers report results on the small nonlinear multicommodity flow problem NDO22.

Some of these algorithms do not extend easily to, or have not been tried with, a disaggregated formulation; the methods of [24] and [32] use disaggregation, but only in the linear case.

2. Problem formulation

The formulation of the multicommodity flow problem follows that of Minoux [32]. We assume that the arcs are not directed (thus flows can traverse arcs in both directions). The flows must then be added up in absolute value as they represent different commodities. This is typical of telecommunication networks. We could also admit that some arcs could be directed (one way), as is usually the case in transportation networks. For clarity's sake we will ignore this possibility.

We are given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{A} \subset \{(s, t): s \in \mathcal{V}, t \in \mathcal{V}, s \neq t\}$ and $m = \text{card}(\mathcal{V})$ is the number of nodes; the arcs are directed, but the direction is selected arbitrarily. In order to formulate this as a nonlinear (or linear) program, the transpose of $\mathcal{A}$ is defined as $\mathcal{A}^{\mathrm{T}} \doteq \{(s, t): (t, s) \in \mathcal{A}\}$. We also define $T(a) \doteq \{(s, t): a = (t, s) \in \mathcal{A}\}$. This simply associates to every directed arc (s, t) the arc with the reverse orientation (t, s); the two directed arcs a and $T(a)$ represent the undirected arc $\{s, t\}$. Clearly $\mathcal{A}^{\mathrm{T}} = T(\mathcal{A})$. If the transposition map $T(a)$ were to be defined on a subset of $\mathcal{A}$, this would permit the modelling of one-way arcs.

To define a nonlinear (or linear) programming formulation we thus use the augmented graph $\overline{\mathscr{G}} = (\mathscr{V}, \overline{\mathscr{A}})$, where $\overline{\mathscr{A}} \doteq \mathscr{A} \cup \mathscr{A}^{\mathrm{T}}$. We denote by $n = \mathrm{card}(\overline{\mathscr{A}})$ the number of arcs and by N the $m \times n$ vertex-arc incidence matrix of $\overline{\mathscr{G}}$.

The set of commodities $\mathscr{I}$ is defined by exogeneous flow vectors (supplies and demands) $d^i = (d^i_s)_{s \in \mathscr{V}}$ that satisfy $e^{\mathrm{T}}_{\mathscr{V}} d^i = 0$ for $i \in \mathscr{I}$, with $e_{\mathscr{V}}$ a vector of ones. These flows must be shipped through $\overline{\mathscr{G}}$.

We denote by $x^i = (x^i_a)_{a \in \overline{\mathscr{A}}}$ the flow of commodity $i \in \mathscr{I}$. The feasible flows for commodity i are members of

$$\mathscr{F}^i = \{ x^i \geqslant 0 : Nx^i = d^i \}.$$

Following the terminology in Jones et al. [24], the flow problem for commodity i is a single origin–single destination (SOSD) problem if d^i has exactly one positive and one negative entry – the origin and the destination, respectively. If d^i has one positive entry and several negative entries, we have a single origin–multiple destination problem (SOMD). We similarly can have a multiple origin–single destination problem (MOSD). These three formulations truly represent the same problem, in the sense that it is possible to aggregate SOSD into SOMD (or MOSD) and conversely disaggregate SOMD (or MOSD) into SOSD. This is easy to see as every SOMD flow can be disaggregated into a convex combination of trees (basis), and every tree-flow disaggregates uniquely into a sum of path-flows, see [24] and Rockafellar [35].

The case of multiple origin-multiple destination problem (MOMD) is somewhat different though, as this formulates the transportation of a generic commodity (gas, oil, cable television, aircraft, etc.). We will restrict ourselves in our experimentation to the origin–destination case (SOSD), even though the algorithm would be applicable to the MOMD, or the mixed MOMD–SOSD cases.

A linear cost vector c^i is associated with the flows of each commodity. A nonlinear (or linear) cost is also charged on the total arc flow, where the total arc flow is $\sum_{i \in \mathscr{I}} (x^i_a + x^i_{T(a)})$. A set of coupling arcs $\mathscr{A}' \subset \mathscr{A}$ comprises those arcs for which there is either a nonlinear cost or a limited capacity on the total arc flow.

The nonlinear multicommodity flow problem can be formulated as

$$\min \quad \sum_{i \in \mathscr{I}} c^{i\mathrm{T}} x^i + \sum_{a \in \mathscr{A}'} f_a(y_a) \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in \mathscr{I}} \left(x^i_a + x^i_{T(a)} \right) \leqslant y_a \quad \forall a \in \mathscr{A}' \subset \mathscr{A}, \tag{2}$$

$$\tag{3}$$

$$x^i \in \mathscr{F}^i \quad \forall i \in \mathscr{I}, \tag{4}$$

$$0 \leqslant y_a \leqslant \gamma_a \quad \forall a \in \mathscr{A}' \subset \mathscr{A}.$$

In this formulation the vector $y = (y_a)_{a \in \mathscr{A}'}$ is meant to represent the joint total arc-flow of a and $T(a)$; in fact it is an upper bound to it, but of course $y_a = \sum_{i \in \mathscr{I}} (x^i_a + x^i_{T(a)})$ at the optimum if we assume that the cost function $f_a(y_a)$ is strictly increasing. We further assume that the function $f_a(y_a)$ is convex and the costs c^i are nonnegative. This will ensure that the Lagrange multipliers (the prices of the coupling constraints) are nonnegative, and in fact positive in the ACCPM; from this follows that the shortest paths routines (the oracles) need not worry about negative or zero cost cycles.

Constraints (3) and (4) are simple network and capacity constraints, respectively. Without (2), the problem is separable. Constraints (2) are named coupling since they coordinate the simple network problems. The nonlinear functions most used in practice are:

- a power function $f_a(y_a) = \alpha_a y_a^{\beta_a}$, with $\alpha_a > 0$ and $\beta_a \geqslant 1$;
- the delay function $f_a(y_a) = y_a/(\gamma_a - y_a)$.

Finally, we mention that the standard linear multicommodity flow corresponds to $f_a \equiv 0$. It is then possible to combine (2) and (4) into

$$\sum_{i \in \mathscr{I}} \left(x_a^i + x_{T(a)}^i \right) \leqslant \gamma_a \quad \forall a \in \mathscr{A}',$$

to obtain the usual formulation:

$$\min \quad \sum_{i \in \mathscr{I}} c^{iT} x^i \tag{5}$$

$$\text{s.t.} \quad \sum_{i \in \mathscr{I}} \left(x_a^i + x_{T(a)}^i \right) \leqslant \gamma_a \quad \forall a \in \mathscr{A}' \subset \mathscr{A}, \tag{6}$$

$$x^i \in \mathscr{F}^i \quad \forall i \in \mathscr{I}. \tag{7}$$

3. The decomposition principle

Real-life multicommodity flow problems leed to very large scale programming problems. For instance a problem with 500 nodes, 1000 arcs and 5000 commodities in the SOSD formulation would lead to a nonlinear program with 2.5×10^6 constraints and 5×10^6 variables; in the SOMD formulation it still remains a problem with 250 000 constraints and 500 000 variables. The difficulty in solving the problem as a single LP or NLP stems from the coupling constraints. The standard decomposition approach strives to separate the issues of finding flows for the individual commodities and of coordinating the individual commodity solutions. In this section we shall briefly review the various elements of this well-known approach.

3.1. Lagrangian dual

The partial Lagrangian is obtained by the dualization of the coupling constraints, using Lagrange multipliers $u = (u_a)_{a \in \mathscr{A}'}$ and letting $\gamma = (\gamma_a)_{a \in \mathscr{A}'}$:

$$\mathscr{L}(x, y; u) = \sum_{i \in \mathscr{I}} c^{iT} x^i + \sum_{a \in A'} f_a(y_a) + \sum_{a \in \mathscr{A}'} u_a \left(\sum_{i \in \mathscr{I}} \left(x_a^i + x_{T(a)}^i \right) - y_a \right). \tag{8}$$

We define

$$\mathscr{L}^P(x, y) := \max_{u \geqslant 0} \mathscr{L}(x, y; u) \tag{9}$$

and

$$\mathscr{L}^D(u) := \min_{\substack{x^i \in \mathscr{F}^i \\ 0 \leqslant y \leqslant \gamma}} \mathscr{L}(x, y; u). \tag{10}$$

The functions $\mathscr{L}^P(x, y)$ and $\mathscr{L}^D(u)$ are respectively convex and concave. From the minmax theorem for convex programming:

$$\min_{\substack{x^i \in \mathscr{F}^i \\ 0 \leqslant y \leqslant \gamma}} \mathscr{L}^P(x, y) = \max_{u \geqslant 0} \mathscr{L}^D(u).$$

So we can choose to solve (10) instead of (9), provided one is able to recover an approximate optimal primal solution (x^*, y^*) to (9) from an approximate optimal dual solution u^* to (10). As the methods used generate both primal and dual solutions, this follows quite naturally from the algorithm.

3.2. Polyhedral approximation and linear relaxation

Column generation – or cutting plane – algorithms use the fundamental subgradient inequality for convex functions. Let us briefly recall the definition of the subdifferential. Let f be a convex function and let x be in the interior of the domain of f. The subdifferential set $\partial f(x)$ of f at x is a nonempty compact convex set such that for any $\xi \in \partial f(x)$ and any y in the domain of f the following subgradient inequality holds:

$$f(y) \geqslant f(x) + \xi^T(y - x).$$

Since $-\mathscr{L}^D(u)$ is convex, we have that for each pair $(u, \bar{u})$ whose elements are in the domain of definition of $\mathscr{L}^D$ and each ξ in the subdifferential $\partial(-\mathscr{L}^D(\bar{u}))$ at $\bar{u}$, the following inequality holds

$$-\mathscr{L}^D(u) \geqslant -\mathscr{L}^D(\bar{u}) + \xi^T(u - \bar{u}).$$

It is convenient to introduce the set $-\partial(-\mathscr{L}^D(\bar{u}))$. So for each $\xi \in -\partial(-\mathscr{L}^D(\bar{u}))$, the reverse inequality holds:

$$\mathscr{L}^D(u) \leqslant \mathscr{L}^D(\bar{u}) + \xi^T(u - \bar{u}).$$

The elements of $-\partial(-\mathscr{L}^D(\bar{u}))$ are sometimes named supergradients.

It is also known, by convex duality, that under the usual regularity conditions,

$$\mathscr{L}^D(u) = \min_{\bar{u} \in \mathbb{R}^{n'}_+} \min_{\xi \in -\partial(-\mathscr{L}^D(\bar{u}))} \left(\mathscr{L}^D(\bar{u}) + \xi^T(u - \bar{u})\right),$$

where n' is the dimension of u or y, i.e., the cardinality of $\mathscr{A}'$, or the number of coupling constraints.

This equality also holds if the minima are restricted to a dense countable subset of

$$\left\{(\bar{u}, -\partial(-\mathscr{L}^D(\bar{u})) : \bar{u} \in \mathbb{R}^{n'}_+\right\}.$$

By using a finite subset of this countable dense set a polyhedral approximation to $-\mathscr{L}^D$ can be constructed and refined as this subset is expanded by further oracle calls.

It is thus possible to use the subgradient inequality to generate a polyhedral relaxation of the epigraph of the function $-\mathscr{L}^D$. For instance, let u^k, $k = 1, \ldots, \kappa$, be a collection

of points in the domain of $\mathscr{L}^D$ and let $\xi^k \in -\partial(-\mathscr{L}^D(\bar{u}^k))$ be an associated collection of supergradients. Then the set of pairs $(-z, u)$ satisfying

$$z - (\xi^k)^T u \leqslant \mathscr{L}^D(u^k) - (\xi^k)^T u^k, \quad k = 1, \ldots, \kappa,$$

is a polyhedral relaxation of the epigraph of $-\mathscr{L}^D$. Thus the polyhedral function

$$\mathscr{L}^D_\kappa(u) = \min_{k=1,\ldots\kappa} \left(\mathscr{L}^D(u^k) + \xi^{k^T}(u - u^k) \right)$$

is an upper bound on $\mathscr{L}^D(u)$, i.e.,

$$\mathscr{L}^D_\kappa(u) \leqslant \mathscr{L}^D(u) \quad \forall u \in \mathbb{R}^{n'}.$$

Hence, the optimum of the linear programming problem (the linear relaxation, or the Dantzig–Wolfe point)

$$\max\left\{ z : z - (\xi^k)^T u \leqslant \mathscr{L}^D(u^k) - (\xi^k)^T u^k,\ k = 1, \ldots, \kappa \right\}, \tag{11}$$

provides an upper bound to the optimal value of $\mathscr{L}^D$. We shall denote it $\theta^\kappa_{\text{sup}}$.

It is possible to localize more precisely the optimal point of $\mathscr{L}^D$. Let

$$\theta^\kappa_{\text{inf}} := \max_{0 \leqslant k \leqslant \kappa} \mathscr{L}^D(u^k)$$

be the largest value of $\mathscr{L}^D$ recorded in the sequence. Clearly θ^k_{inf} is a lower bound for the optimal value of $\mathscr{L}^D$. Moreover, any optimal point $(\mathscr{L}^D(u^*), u^*)$ lies in the so-called *localization set*

$$\mathscr{LOC}_\kappa = \left\{ (z, u) : z \geqslant \theta^\kappa_{\text{inf}} \text{ and } z - (\xi^k)^T u \right.$$
$$\left. \leqslant \mathscr{L}^D(u^k) - (\xi^k)^T u^k,\ k = 1, \ldots, \kappa \right\}. \tag{12}$$

3.3. Cutting planes: A prototype decomposition algorithm

The basic idea underlying cutting plane algorithms can be described as follows. Given a polyhedral outer approximation of the epigraph of the function, one selects a point u such that the pair (z, u), for some z, belongs to the localization set. The value $\mathscr{L}^D(u)$, and an element ξ of the supergradient set, are then computed; a new valid supergradient inequality is added to the definition of the localization set. This new inequality either "cuts" the localization set or is a support to it. The lower bound is updated and a new smaller localization set is obtained.

We can state formally the basic steps in the prototype cutting plane method. For the sake of simpler notations we drop the iteration index k. A localization set $\mathscr{LOC}$ and a lower bound θ_{inf} are given.

Step 1. Select $(\bar{z}, \bar{u}) \in \mathscr{LOC}$.

Step 2. Compute $\mathscr{L}^D(\bar{u})$ and $\xi \in -\partial(-\mathscr{L}^D(\bar{u}))$.

Step 3. Update $\theta_{\inf}$ by $\theta_{\inf} := \max\{\theta_{\inf}, \mathscr{L}^D(\bar{u})\}$ and add the inequality

$$z - \xi^T u \leqslant \mathscr{L}^D(\bar{u}) - \xi^T \bar{u}$$

to the definition of the localization set.

The process ends when the duality gap, $\theta_{\sup} - \theta_{\inf}$, falls below a given precision level.

Step 1 of the prototype algorithm is usually called the master program. The way $\bar{u}$ is chosen in the localization set characterizes the cutting plane method. Step 2 is problem dependent. It has been called an oracle, though the classical terminology in the Dantzig–Wolfe approach refers to subproblems.

4. Levels of disaggregation in the decomposition for multicommodity flow problems

In the case of the nonlinear multicommodity flow, the partial dual Lagrangian (10) is

$$\begin{aligned}\mathscr{L}^D(u) &= \min_{\substack{x^i \in \mathscr{F}^i \\ 0 \leqslant y \leqslant \gamma}} \left(\sum_{i \in \mathscr{I}} c^{iT} x^i + \sum_{a \in \mathscr{A}'} f_a(y_a) + \sum_{a \in \mathscr{A}'} u_a \left(\sum_{i \in \mathscr{I}} \left(x_a^i + x_{T(a)}^i \right) - y_a \right) \right) \\ &= \min_{x^i \in \mathscr{F}^i} \sum_{i \in \mathscr{I}} (c^i + u)^T x^i + \min_{0 \leqslant y \leqslant \gamma} \sum_{a \in \mathscr{A}'} (f_a(y_a) - u_a y_a) \\ &= \sum_{i \in \mathscr{I}} \min_{x^i \in \mathscr{F}^i} (c^i + u)^T x^i + \sum_{a \in A'} \min_{0 \leqslant y_a \leqslant \gamma_a} (f_a(y_a) - u_a y_a).\end{aligned}$$

The computation of the function $\mathscr{L}^D(u)$ and of a supergradient separates into easily computable functions:

$$\mathscr{L}^D(u) = \sum_{i \in \mathscr{I}} \mathscr{L}^i(u) + \sum_{a \in \mathscr{A}'} \mathscr{L}^a(u), \tag{13}$$

where each

$$\mathscr{L}^i(u) = \min_{x^i \in \mathscr{F}^i} (c^i + u)^T x^i \tag{14}$$

is the solution of a shortest path problem with nonnegative costs $c^i + u$, and where, abusing notation, $u^T x^i$ means $\sum_{a \in \mathscr{A}'} u_a (x_a^i + x_{T(a)}^i)$, while

$$\mathscr{L}^a(u) = \min_{0 \leqslant y_a \leqslant \gamma_a} (f_a(y_a) - u_a y_a) \tag{15}$$

is a one-dimensional convex program whose solution can usually be given analytically.

It should be noted that if the costs c^i do not depend on the commodity i, then only *one* shortest path problem between all pairs of nodes needs to be solved.

It is interesting to note that in the case of linear costs, $f_a \equiv 0$, the Lagrangian dual is

$$\mathscr{L}^D(u) = \sum_{i \in \mathscr{I}} \min_{x^i \in \mathscr{F}^i} (c^i + u)^T x^i + \sum_{a \in A'} \min_{0 \leqslant y_a \leqslant \gamma_a} (-u_a y_a). \tag{16}$$

It is important to note that, in the usual linear case (5), the function $\mathscr{L}^D$ differs in a very significant way from the partial dual associated to the (nonlinear) multicommodity flow formulation (1). Letting v_a be the Lagrange multipliers there one gets

$$\mathscr{L}_1^D(v) = \sum_{i \in \mathscr{I}} \min_{x^i \in \mathscr{F}^i} (c^i + v)^{\mathrm{T}} x^i - v^{\mathrm{T}}\gamma. \tag{17}$$

The functions $\mathscr{L}^D$ and $\mathscr{L}_1^D$ appear to be the sum of hopefully simpler functions. The simple structure can be exploited to yield various level of disaggregation. Each level generates its own set of subgradient inequalities.

4.1. No disaggregation: Single cuts

If we do not take advantage of the additive structure of $\mathscr{L}^D$, we obtain a supergradient of $\mathscr{L}^D$ at $\bar{u}$ as follows. Let $x(\bar{u})$ and $y(\bar{u})$ be optimal solutions of problems (14) and (15) for a value $\bar{u}$ of the Lagrange multiplier. The supergradient of $\mathscr{L}^D(u)$ at $\bar{u} = u$ is given by the coefficient of $\bar{u}_a$ in (13), i.e.,

$$\xi_a = \sum_{i \in \mathscr{I}} \left(x_a^i(\bar{u}) + x_{\mathrm{T}(a)}^i(\bar{u}) \right) - y_a(\bar{u}) \Big).$$

Note that at least one of the components of x_a^i or $x_{T(a)}^i$ is zero, as the subproblems compute shortest paths or trees.

In the case of linear costs it is interesting to compare the two formulations (16) and (17). In the former case the function $\sum_{a \in \mathscr{A}'} \mathscr{L}^a(u)$ has a superdifferential at $u = 0$ with $2^{n'}$ extreme points while in the latter case $-v^{\mathrm{T}}\gamma$ is differentiable and has a unique supergradient $-\gamma$.

Practitioners have always noticed that the additive structure of $\mathscr{L}_1^D$ can be used for a variant of Dantzig–Wolfe decomposition, which is sometimes effective.

4.2. Full disaggregation: Multiple cuts

The linear case

Recall that

$$\mathscr{L}_1^D(v) = \sum_{i \in \mathscr{I}} \mathscr{L}^i(v) - v^{\mathrm{T}}\gamma.$$

A disaggregated LP relaxation can be defined as

$$\mathscr{L}_1^D(v) = -v^{\mathrm{T}}\gamma + \sum_{i \in \mathscr{I}} \max_{x_i} \{ z_i : z_i \leqslant \mathscr{L}^i(v) \},$$

where

$$\mathscr{L}^i(v) = \min_{x^i \in \mathscr{F}^i} (c^i + v)^{\mathrm{T}} x^i. \tag{18}$$

Each function $\mathscr{L}^i$ has its own support $\xi^i = x^i$ at a point $\bar{v}$. Now the proposals ξ^i depend on whether the formulation is SOSD or SOMD; in the former case the proposals are path-flows, in the latter they are tree-flows.

As clearly shown in [24] the most disaggregated formulation, i.e., the SOSD formulation using path-flows, is the most effective one. Only a few proposals are needed for each of the component functions to define them accurately. The SOMD or tree-flow formulation creates exponentiality in the number of extreme points of the subdifferentials.

In this disaggregated formulation it is possible to introduce a cut for each $i \in \mathcal{I}$. To this end we introduce variables z_i, and $z = (z_i)_{i \in \mathcal{I}}$. We define the LP relaxation in the extended space $\mathbb{R}^{n'+|\mathcal{I}|+1}$ by

$$\begin{aligned} &\max \quad z \\ &\text{subject to} \\ &z = \sum_{i \in \mathcal{I}} z_i - v^{\mathrm{T}}\gamma, \\ &z_i - \left(\xi^{ik}\right)^{\mathrm{T}} v \leqslant \mathcal{L}^i(v^k) - \left(\xi^{ik}\right)^{\mathrm{T}} v^k, \quad i \in I, \quad k = 1, \ldots \kappa \end{aligned} \tag{19}$$

while the set of localization becomes

$$\begin{aligned} \mathcal{LOC}_\kappa = \Big\{ &(z, z_I, u) : z \geqslant \theta_{\inf}^\kappa, z = -v^{\mathrm{T}}\gamma + \sum_{i \in \mathcal{I}} z_i, \\ &z_i - \left(\xi^{ik}\right)^{\mathrm{T}} v \leqslant \mathcal{L}^i(v^k) - \left(\xi^{ik}\right)^{\mathrm{T}} v^k, \; i \in I, \; k = 1, \ldots \kappa \Big\}. \end{aligned} \tag{20}$$

The nonlinear case

In the nonlinear case the disaggregation needs to go further; we have

$$\mathcal{L}^D(u) = \sum_{i \in \mathcal{I}} \mathcal{L}^i(u) + \sum_{a \in A'} \mathcal{L}^a(u).$$

We may choose a partial disaggregation

$$\mathcal{L}^D(u) = \max_{z_0} \left\{ z_0 : z_0 \leqslant \sum_{a \in \mathcal{A}'} \mathcal{L}^a(u) \right\} + \sum_{i \in \mathcal{I}} \max_{z_i} \left\{ z_i : z_i \leqslant \mathcal{L}^i(u) \right\},$$

but the finer granularity of

$$\mathcal{L}^D(u) = \sum_{a \in \mathcal{A}'} \max_{z_a} \left\{ z_a : z_a \leqslant \mathcal{L}^a(u) \right\} + \sum_{i \in \mathcal{I}} \max_{z_i} \left\{ z_i : z_i \leqslant \mathcal{L}^i(u) \right\}$$

turns out to be *significantly* more effective.

We define the LP relaxation in the extended space $\mathbb{R}^{2n'+|\mathcal{I}|+1}$, as $n' = |\mathcal{A}'|$, by:

$$\begin{aligned} &\max \quad z \\ &\text{subject to} \\ &z = \sum_{i \in \mathcal{I}} z_i + \sum_{a \in A'} z_a, \\ &z_i - \left(\xi^{ik}\right)^{\mathrm{T}} u \leqslant \mathcal{L}^i(u^k) - \left(\xi^{ik}\right)^{\mathrm{T}} u^k, \quad i \in \mathcal{I}, \quad k = 1, \ldots \kappa, \\ &z_a - \left(\xi^{ak}\right)^{\mathrm{T}} u \leqslant \mathcal{L}^a(u^k) - \left(\xi^{ak}\right)^{\mathrm{T}} u^k, \quad a \in \mathcal{A}', \quad k = 1, \ldots \kappa, \end{aligned} \tag{21}$$

while the set of localization becomes:

$$\mathscr{LOC}_\kappa = \Big\{ (z, z_I, z_{A'}, u) : z \geqslant \theta_{\inf}^\kappa,$$

$$z = \sum_{i \in \mathscr{I}} z_i + \sum_{i \in A'} z_a,$$

$$z_i - (\xi^{ik})^{\mathrm{T}} u \leqslant \mathscr{L}^i(u^k) - (\xi^{ik})^{\mathrm{T}} u^k,\ i \in I,\ k = 1, \ldots \kappa,$$

$$z_a - (\xi^{ak})^{\mathrm{T}} u \leqslant \mathscr{L}^a(u^k) - (\xi^{ak})^{\mathrm{T}} u^k,\ a \in \mathscr{A}',\ k = 1, \ldots \kappa \Big\}. \quad (22)$$

The motivation here is that it is easy to approximate the functions $\mathscr{L}^i$ in the SOSD case, as few proposals (supergradients) are needed to approximate $\mathscr{L}^i$ accurately but that is also easy to approximate the functions $\mathscr{L}^a$ by cutting planes as they are functions of one variable only.

Note that the supergradient ξ^{ik} represents a shortest path between the origin and the destination of commodity i, that carries the total flow of commodity i. The supergradient ξ^{ak} has one nonzero component $\xi_a^{ak} = -y_a^{ak}$.

There is of course one drawback to this finest of granularities: the restricted master program – the dual of (22) – has as many convexity rows as there are coupling arcs *and* commodities ($|\mathscr{I}| + |\mathscr{A}'|$). Fortunately, those can be handled very efficiently by the GUB technique as will be shown in Section 6.1. Another point to note is the fact that the supergradients are sparse, as ξ^i are paths and ξ^a have only one nonzero component.

5. The analytic center cutting plane method (ACCPM)

There exist several reports on the principle of the method [15] and on its implementation aspects [16,4,5]. For clarity's sake we provide here a short description of ACCPM.

The problem of interest is the computation of the analytic center of the set of localization. This set is defined by a system of inequalities such as (21) or (22); in order to make it compact, we add artificial bounds $u \leqslant M$ on the Lagrange multipliers. We already noted that in our formulation of the multicommodity network flow problems the multipliers are nonnegative. (The box constraint $0 \leqslant u \leqslant M$ clearly compactifies the domain.) The analytic center is the unique maximizer of the product of the slacks to each of these inequalities (or equivalently, the sum of their logarithms).

Different interior point methods can be used to compute the analytic center. In ACCPM we choose a variant [9] of Karmarkar's projective algorithm [25] and apply it to the dual of (21) (or (22)). The analytic center, or its approximation, is obtained by simple duality.

Let us stress that the dual of (21) (or (22)) is the restricted master program of the Dantzig–Wolfe algorithm. This linear program has a special structure. Individual columns in it correspond to cuts (supergradients) generated by a commodity or by a

coupling arc. The columns associated with the same commodity are linked by a convexity constraint and so are the columns corresponding to a given coupling arc; hence a GUB structure. In addition to these columns, there are unit vectors corresponding to the box constraints $0 \leqslant u \leqslant M$.

In a typical iteration, new columns are added after the analytic center of the current localizaton set has been computed. The localization set is updated and a new analytic center is computed. In order to cut down the computational effort, one should be able to make use of the previous analytic center as a warm start. This is known to be a challenge for any interior point method. In the projective method we use, the addition of the cuts in the localization set amounts to the introduction of new columns, i.e., variables that must be given an initial value zero to maintain feasibility, thereby violating the fundamental interior property requirement of the method. To handle this case a special technique has been devised proposed in [33]. (See [15] for the principle and [4],[5] for a detailed description.) Note that for a nonlinear multicommodity flow problem, the number of potential new columns at each iteration is the number of commodities plus the number of coupling arcs, a considerable number for problems with many commodities or many coupling arcs.

6. Implementation of ACCPM

The ACCPM approach presented in previous sections has been implemented and applied to solve two practical (and well documented in the literature) nondifferentiable optimization (NDO) problems: NDO22 and NDO148 and several large randomly generated problems. As we were encouraged by the results of earlier experiments [15,16], we have decided to prepare a specialized sparsity-exploiting implementation of the method dedicated to real-life large scale multicommodity network flow problems. This section addresses several issues of our implementation.

6.1. Projective algorithm in the master problem

As explained before a single outer iteration consists of finding an approximate analytic center for a (modified) set of localization. This is done by the projective algorithm. The bulk of the work in a single iteration of it (as in a single iteration of any other interior point method [19]) is clearly computing the orthogonal projection of a vector onto the null space of a scaled linear operator. Let A denote this linear operator (the LP constraint matrix) and Y denote a diagonal scaling matrix. We need to compute the orthogonal projection of a vector onto the null space of AY and this effort is dominated by the inversion of the matrix

$$AY^2A^{\mathrm{T}}. \tag{23}$$

The matrix A shows a lot of special structure in itself resulting from box constraints imposed on dual variables, our approach to dealing with nonlinear objective term and, finally, GUB constraints coming from the decomposition approach.

$$A = \left(\begin{array}{c|cc|ccc|ccc} & & & & & & h_1^{\mathrm{T}} & & \\ & I & -I & G_1 & \cdots & G_p & & \ddots & \\ & & & & & & & & h_n^{\mathrm{T}} \\ \hline & & & e_1^{\mathrm{T}} & & & & & \\ \gamma & & & & \ddots & & & & \\ & & & & & e_p^{\mathrm{T}} & & & \\ \hline & & & & & & f_1^{\mathrm{T}} & & \\ & & & & & & & \ddots & \\ & & & & & & & & f_n^{\mathrm{T}} \end{array}\right). \tag{24}$$

Here γ is a (presumably dense) column resulting from the transformation of the problem to the form required by the projective algorithm [9], each column of G_i characterizes a feasible path (one supergradient ξ^{ik} in the notation of (22)) for the demand of commodity i, $i = 1,\ldots,p$, h_j, $j = 1,\ldots,n$, are vectors built of supergradients corresponding to nonlinear part in the objective (ξ^{ak} in (22)). Next, e_i, $i = 1,\ldots,p$ and f_j, $j = 1,\ldots,n$, are now vectors of all ones of appropriate dimensions (GUB constraints). In order to solve large scale problems and to reach maximum efficiency in the implementation, we have to exploit the special structure of A. We discuss below how this can be done. Assume the scaling matrix Y has been partitioned accordingly to (24):

$$Y = \operatorname{diag}\left(Y_\gamma, Y_{\eta_1}, Y_{\eta_2}, Y_{G_1},\ldots,Y_{G_p}, Y_{h_1},\ldots,Y_{h_n}\right). \tag{25}$$

Then (23) becomes

$$AY^2A^{\mathrm{T}} = H + \gamma Y_\gamma^2\gamma^{\mathrm{T}} = \begin{pmatrix} C & B_1 & B_2 \\ B_1^{\mathrm{T}} & D_1 & 0 \\ B_2^{\mathrm{T}} & 0 & D_2 \end{pmatrix} + \gamma Y_\gamma^2\gamma^{\mathrm{T}}, \tag{26}$$

where

$$C = \sum_{i=1}^{p} G_i Y_{G_i}^2 G_i^{\mathrm{T}} + Y_{\eta_1}^2 + Y_{\eta_2}^2 + \operatorname{diag}\left(h_j^T Y_{h_j}^2 h_j\right), \tag{27}$$

$$B_1 = \left(G_1 Y_{G_1}^2 e_1, G_2 Y_{G_2}^2 e_2,\ldots,G_p Y_{G_p}^2 e_p\right), \tag{28}$$

$$B_2 = \operatorname{diag}\left(h_j^T Y_{h_j}^2 f_j\right), \tag{29}$$

$$D_1 = \operatorname{diag}\left(e_i^{\mathrm{T}} Y_{G_i}^2 e_i\right), \tag{30}$$

$$D_2 = \operatorname{diag}\left(f_j^{\mathrm{T}} Y_{h_j}^2 f_j\right). \tag{31}$$

It is also convenient to introduce the matrices

$$B = (B_1 \quad B_2) \text{ and } D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}. \tag{32}$$

Column γ of (24) is supposed to be dense. If not taken into account separately, it would completely destroy the sparsity of AY^2A^T. It is best to view it as a rank-one correction to H and handle it by the Schur complement mechanism [7,14]. Assume that we have computed a symmetric (sparse) factorization

$$H = L\Lambda L^T. \tag{33}$$

From the Sherman–Morrison–Woodbury formula we obtain the following inverse of (26):

$$(AY^2A^T)^{-1} = L^{-T}\Lambda^{-1/2}\left(I - \frac{1}{1 + v^T\Lambda^{-1}v}vv^T\right)\Lambda^{-1/2}L^{-1}, \tag{34}$$

where

$$v = L^{-1}Y_\gamma\gamma. \tag{35}$$

The above formula thus yields that, having done the preparation step (35), we can solve any equation with AY^2A^T: this involves one forward transformation with L, one backsolve with L^T and a few scalar products. Before we move on to the description of our approach to the computation of the factorization (33), let us comment on the dimension of matrix H in (26). C is an almost completely dense square matrix which size is the number of arc flow constraints. D_1 and D_2 are diagonal matrices of sizes equal to the number of commodities p and the number of arc flow constraints n, respectively. NDO148, for example, (see numerical results section) has 148 arc flow constraints and 122 commodities, so H has dimension $2n + p = 418$; this could be handled by any general purpose sparse Cholesky factorization code. However, we expect computational advantages as well as obvious storage savings from avoiding its explicit formulation. Instead, we propose a specialized block decomposition technique for its inversion. The whole diagonal block D is pivoted out first, as this operation introduces no fill-in:

$$H = \begin{pmatrix} I & BD^{-1} \\ 0 & I \end{pmatrix}\begin{pmatrix} C - BD^{-1}B^T & 0 \\ 0 & D \end{pmatrix}\begin{pmatrix} I & 0 \\ D^{-1}B^T & I \end{pmatrix}. \tag{36}$$

Next, we build the dense symmetric matrix S, and compute its triangular factorization

$$S = C - BD^{-1}B^T = L_0D_0L_0^T. \tag{37}$$

We then obtain the required representation

$$H = \begin{pmatrix} L_0 & BD^{-1} \\ 0 & I \end{pmatrix}\begin{pmatrix} D_0 & 0 \\ 0 & D \end{pmatrix}\begin{pmatrix} L_0^T & 0 \\ D^{-1}B^T & I \end{pmatrix}, \tag{38}$$

which is basically the same as (33), except that L and Λ are decomposed into smaller easily invertible blocks. Summing up, the solution of an equation with AY^2A^T needs two solves with L and one with L^T, where

$$L = \begin{pmatrix} L_0 & BD^{-1} \\ 0 & I \end{pmatrix} \tag{39}$$

is a block upper triangular matrix. It also needs several additional inner products. The main computational effort in finding representation (38) consists in creating and decomposing matrix S in (37). To avoid explicit formulation of B_1 in S we represent the term $B_1 D_1^{-1} B_1^T$ as the weighted sum of the outer products of columns b_i of B_1. Thus we have for S,

$$S = C - \sum_{i=1}^{p} \frac{1}{d_{1_i}} b_i b_i^T - B_2 D_2^{-1} B_2^T. \tag{40}$$

(Recall that S is dense so we do not bother about the order in which simple products in $b_i b_i^T$ are accumulated.) A column of B_1 is thus built and forgotten just after its contribution to (40) has been added. B_2 is a diagonal matrix and its contribution to (40) is the same. Implicit handling of B_1 leads to clear storage savings. The only part of H that has to be stored is matrix C which is later rewritten with S and finally replaced with a factorization (37). Its size is equal to the number of arc flow constraints, and thus, is known in advance.

In addition to the construction of S, the matrix B is used again in solves with L and L^T. The implicit handling of B_1 here also leads to computational time savings, due to the special structure of the G_i matrices. This will not necessarily be the case in other ACCPM applications as will become clear after the presentation of our approach to the handling G_i matrices in Section 6.3.

Let us finally address the issue of stability in computing orthogonal projections. In our experiments we observed excellent accuracy, which seems to be a structural feature of the approach. First, the matrix (24) has full row rank even after removal of the first column γ. Second, all GUB constraints are orthogonal to each other. The presence of diagonal term $Y_{\eta_1}^2 + Y_{\eta_2}^2$ in (27) prevents excessive spread of the pivots in the factorization (37). Consequently, the factorization (33) is stable and so is the rank-one update (26) used to deal with dense column γ. This situation is in contrast to general linear programming [19] for which a removal of dense columns may lead to a rank deficiency of Sherman–Morrison–Woodbury update.

6.2. The oracle

Each subproblem consists in finding a minimum cost flow between an origin and a destination node on an uncapacited network and it can be formulated as a shortest path problem. As the network is expected to be sparse, Dijkstra's d-heap algorithm is a suitable solution method. Our implementation follows the description of [1]. The basic

step of Dijsktra's algorithm consists of an update of two independent data structures: a set of nodes $\mathscr{S}$ for which the shortest path has already been determined; and a set of all neighbours of $\mathscr{S}$ (nodes that are adjacent to a node in $\mathscr{S}$). In a single iteration, Dijsktra's algorithm chooses one node from the set of neighbours and adds it to $\mathscr{S}$. To accelerate the search in a subset of neighbours, nodes are stored in a form of a tree (d-heap). Once a node is chosen, the list is updated accordingly.

The reader interested in a more detailed description of Dijktra's algorithm and its efficient implementation is referred to [1]. Let us mention, however, that its application undoubtly contributed to the overall efficiency of the ACCPM approach on this particular class of multicommodity network flow problems. In fact, even with a large number of subproblems to be solved at every cutting plane iteration (see the numerical results in Section 7), the time needed to find a minimum cost flow seldom exceeded 10% of the total computational time.

6.3. Data structures for cuts

Every call to the ith shortest path subproblem can add a new cut, i.e., a new column to the matrix G_i. Each subproblem consists of finding a minimum cost flow between two nodes. In many practical applications, this path involves only a few arcs. As a result, every column has only a few nonzero elements. These nonzeros have all the same value g_i for each commodity i. Thus we may write

$$G_i = g_i G_{i0}, \tag{41}$$

where G_{i0} is a 0–1 matrix that can be stored in a compact form involving half length integers. Moreover G_{i0} is usually very sparse. We refer to it in every solve with L of (39) and in the computation of (40). Each such reference creates a single column of B_1,

$$b_i = G_i Y_{G_i}^2 e_i, \tag{42}$$

possibly scaled by some factor. We take advantage of the form (41) and replace the above equation with

$$b_i = G_{i0}\left(g_i Y_{G_i}^2 e_i\right). \tag{43}$$

We thus reduce the number of necessary multiplications from the number of nonzero entries of G_i to the number of its columns (these multiplications need to be done only once in a single iteration of the projective algorithm). The remaining arithmetic operations in (43) are additions that would also have to be done if (42) had been used. The matrices G_i grow in subsequent iterations of the ACCPM algorithm so the most suitable data structure to remember them is a collection of sparse columns (see, e.g., [10, Chapter 2]). Let us finally observe that replacing (42) with (43) is advantageous only if these columns are very sparse and their number is not excessive. Fortunately, both these conditions were always satisfied when solving multicommodity network flow problems.

7. Numerical results

An implementation of ACCPM algorithm has been written in C (except for the Cholesky factorization that comes from LAPACK of Andersen et al. [2] and is written in FORTRAN 77). The code has been run on a POWER PC workstation (type 7011, model 25T) with 64MB of virtual memory. It was compiled with the AIX XL C + + and FORTRAN 77 compiler with a default optimization level (-O option). Several multicommodity network flow problems have been solved with ACCPM, each to a six-digits relative accuracy of the optimal solution.

7.1. Impact of disaggregation

Our first series of tests were performed on two practical, well documented, standard nondifferentiable problems NDO22 and NDO148 from [13]. See Table 1 for their statistics.

We used two different levels of granularity: cuts are fully aggregated and fully disaggregated.

In Tables 2 and 3, we report detailed information on the influence of various levels of aggregation on the efficiency of ACCPM and Dantzig–Wolfe (DW) implementations, respectively. The tables report: the number of outer iterations, NITER, the number of inner iterations, Newton steps (or simplex pivots in the case of DW), the number of cuts (supergradients) added and the solution time. A simple test was used to check not to duplicate the existing cuts. (It was applied to the linear parts only.) Note, that in a case of full aggregation, each iteration generates one cut while in the case of maximum disaggregation, each iteration generates p cuts for the nonlinear part ($p = 22$ and $p = 148$, respectively) and at most n cuts for the shortest path part ($n = 23$ and $n = 122$, respectively).

As mentioned earlier, the efficiency of ACCPM depends mainly on the number of interior point (Newton) iterations which does not seem to vary much with the number of subgradients added at every outer iteration. The disaggregated version of ACCPM accumulates about one cut per subproblem at every outer iteration, many more than in classical aggregate NDO approaches. Note, that the number of outer iterations or, equivalently, calls to oracle, corresponds to the number of objective function evaluations (a usual measure of efficiency in nondifferentiable optimization [27]).

Summing up, ACCPM did pretty well, finding an accurate solution in a number of iterations proportional to the number of coupling constraints n', in the case of aggregated formulation and O(1) in the disaggregated formulation.

Table 1
NDO problems statistics

Problem	Nodes	Arcs	Comm.	Optimum
NDO22	14	22	23	−103.412017
NDO148	61	148	122	−151.926870

Table 2
Efficiency of ACCPM on nonlinear NDO problems

Disaggr.	NDO22				NDO148			
	NITER	Newton	CUTS	Time	NITER	Newton	CUTS	Time
None	138	186	138	1.37	857	1011	857	558.08
Maximum	14	50	353	0.51	16	81	2649	16.15

In contrast, the behavior of Dantzig–Wolfe algorithm depends much on the aggregation of cuts. For an advanced implementation of Dantzig–Wolfe algorithm see, e.g., [11] and [24]. When multiple cuts are used, both ACCPM and DW work similarly, although ACCPM turns out to be about two times faster on larger, NDO148 problem. When fully aggregated cuts are added, DW looses much of its efficiency and performs poorer than ACCPM. In particular, DW was unable to solve NDO148 problem in a reasonable time. It stalled with a relative precision of about 20%.

7.2. ACCPM on large scale problems

Let us now analyse the efficiency of the method on large scale nonlinear multicommodity network flow problems. According to our knowledge, there is no public domain collection of large scale multicommodity network flow problems. Most of problems reported in the literature come from practical applications and are proprietary. We thus decided to generate randomly a wide class of such test problems. Below we briefly describe the way in which it is done.

The program generates a multicommodity flow problem with n_c commodities on a graph that has n_n nodes and n_a arcs. (The numbers n_c, n_n and n_a are user supplied.) The graph has a two-level structure. Its lower level consists of $n_b - 1$ independent (connected) subgraphs. Each of them has roughly speaking the same number of nodes and arcs but their structure is randomly generated. The total number of nodes in these subgraphs is n_n.

For every lower level subgraph a node is chosen (call it connector) and another, connected, randomly generated graph is spanned on the set of connectors. The number of arcs in this higher level subgraph is chosen in such a way that the total number of arcs in the network is equal to a prescribed value n_a. From this construction, the resulting graph is connected. It contains n_b blocks: $n_b - 1$ of them are completely

Table 3
Efficiency of DW on nonlinear NDO problems

Disaggr.	NDO22				NDO148			
	NITER	Pivots	CUTS	Time	NITER	Pivots	CUTS	Time
None	582	4239	582	23.35	> 10000	?	10000	> 50000
Maximum	14	343	630	0.46	16	7350	4320	27.66

Table 4
Statistics of randomly generated problems

Problem	Blocks	Nodes	Arcs	Comm.
Random1	4	60	140	100
Random2	4	60	140	500
Random3	4	60	140	2000
Random4	4	100	150	120
Random5	5	100	300	200
Random6	10	100	300	200
Random7	10	200	400	200
Random8	10	200	400	500
Random9	10	200	500	500
Random10	10	200	500	1000
Random11	10	200	500	3000
Random12	10	300	600	1000
Random13	10	300	800	1000
Random14	10	400	800	2000
Random15	15	300	1000	4000
Random16	10	300	1000	1000
Random17	10	400	1000	2000
Random18	15	400	1000	3000
Random19	15	400	1000	4000
Random20	15	400	1000	5000
Random21	15	500	1000	3000
Random22	15	500	1000	4000

independent lower level subnetworks and the last one defines links between them. Each block has the same numbers of nodes and arcs, n_n/n_b and n_a/n_b, respectively (unless n_n and n_a are not multiples of n_b).

In the next step, n_c origin–destination pairs of nodes are randomly chosen. A single path flow is then generated between each origin–destination pair. Next, the sum of all flows passing through a given arc is computed and increased with some (safety) positive number to define a capacity of the arc. For such capacities the problem has well defined feasible and optimal solutions.

Let us mention that our generator of nonlinear multicommodity network flow problems is available for research purposes. [4]

We have used this generator to produce a wide class of test problems that differ in the size and in the structure of the network. As we have observed consistent good efficiency of the ACCPM approach on these problems, we do not report all results obtained but restrict ourselves to a representative subset of them. Table 4 collects information on these test problems. For every problem, it reports several characteristics of the graph structure, namely: the number of subgraphs, the number of nodes, the number of arcs and the number of commodities.

The problems collected in Table 4 are listed in increasing order of the number of arcs n. Recall that this number determines the size of matrix S of (37) so it is the most

[4] contact Robert Sarkissian: sarkissi@divsun.unige.ch

Table 5
Efficiency of ACCPM on randomly generated problems

Problem	NITER	Newton	CUTS		CPU time		
			total	paths	t_F	t_O	total
Random1	18	107	2573	289	14.73	1.68	17.06
Random2	18	159	3838	1392	38.73	9.24	49.18
Random3	23	273	9398	6288	206.77	51.96	264.16
Random4	17	99	2687	270	12.95	3.19	16.84
Random5	20	134	6117	774	95.90	7.51	105.35
Random6	19	139	5403	775	94.21	7.46	103.46
Random7	18	159	7062	812	251.78	14.78	268.70
Random8	20	202	9577	2396	403.07	41.19	447.79
Random9	20	171	10366	1718	545.02	46.62	595.68
Random10	21	274	14384	5031	761.48	47.80	824.10
Random11	28	345	28280	15635	1943.10	357.19	2321.07
Random12	23	243	18288	5554	1531.65	154.50	1694.84
Random13	23	268	21314	4925	3405.28	167.15	3582.93
Random14	22	281	26329	9529	4107.81	448.61	4570.70
Random15	26	388	43243	21251	9306.64	794.43	10132.44
Random16	23	256	24429	5430	5277.92	176.60	5467.75
Random17	25	341	34739	12016	8351.45	502.32	8874.58
Random18	25	354	41340	19040	9293.14	771.58	10090.58
Random19	28	375	49123	24718	10135.82	1142.64	11315.36
Random20	29	479	59432	33614	13769.69	1454.98	15275.07
Random21	27	384	43741	19042	10701.53	1009.40	11744.53
Random22	28	397	49599	23721	11650.10	1405.78	13097.43

important factor in evaluating ACCPM's computational effort (every inner iteration of the method requires computing one Cholesky decomposition of S; this is done in about $\frac{1}{6}n^3$ *flops* [18]).

Let us observe that the numbers given in Table 4 "hide" the large size of the problems solved. The linear version of the Random10 problem, for example, in an equivalent compact LP formulation, would involve 200 blocks of 1000 constraints of commodity flow balance at each node and 500 coupling constraints of total flow capacity on the arcs. This formulation comprises $200 * 1000 + 500 = 200\,500$ constraints and $2 * 500 * 1000 = 1\,000\,000$ variables. The reader interested in the influence of the multicommodity flow problem formulation on the efficiency of different solution methods is referred to [24].

Table 5 collects data on the solution of randomly generated problems. We report in it: the number of outer iterations, NITER, the number of inner iterations, Newton, the total number of cuts (subgradients) added through the whole solution process, the number of shortest path type cuts and the CPU time (to reach a 6-digit accurate solution on a POWER PC computer). To give a bit of an insight into the ACCPM's behavior, Table 5 additionally reports the time spent in the factorizations of S (dominating term in the master), t_F, and the time spent in the oracle, t_O.

An analysis of Table 5 results clearly indicates that the most computationally expensive part of ACCPM consists in building and factorizing matrix S (see Eqs. (37)

and (40)). Apart from the predicted cubic dependence of this effort on the number of arc flow constraints n, we also note considerable influence of the number of commodities, specially if n is not excessively large. Such an influence is observed on problems 1, 2 and 3 which differ only in the number of commodities (100, 500 and 2000, respectively). It is considerably less important for larger n (compare, for example, times for problems Random16 through Random20) although the total number of cuts varies linearly with the number of commodities. The time spent in subproblems usually varies between 5 and 10% of the total CPU effort and depends little on the number of commodities, which proves the high efficiency of Dijkstra's d-heap algorithm for this class of problems.

Let us observe that except for problems with a very large number of commodities, the average number of Newton steps required to approach a new approximate analytic center of the localization set is about 10. For problems with a large number of commodities many more cuts are added at every call to the oracle. Consequently, more significant changes are made to the localization set and the projective algorithm needs more iterations (up to 20, in the average) to approach the new approximate analytic center. This is a possible place for further improvements.

Finally, we would like to comment on the number of subgradients added during the solution process. The number of subgradients corresponding to the shortest paths shows uniform linear dependance on the number of commodities: from 3 cuts per commodity on smaller problems up to 7 cuts per commodity on larger ones. The total number of subgradients counts also the cuts resulting from the nonlinear term in the objective. It thus depends much more on the structure of the network and on how tight the arc capacity constraints are. Our experience indicates that, in practice, this number also grows linearly with the number of commodities.

Our last experiment aims at showing the trade-off between the required relative precision of the optimum and the time needed to reach it for the problem Random10. In Table 6 we report numbers of outer and inner iterations, the number of subgradients (cuts) added, and the CPU time required to reach a given number of the exact digits of the optimum.

The results collected in Table 6 show that it takes much time to build the first complete description of the localization set and to reach at least one digit exact solution.

Table 6
Trade-off between an accuracy and an efficiency of ACCPM for Random10 problem

Accuracy	NITER	Newton	CUTS		CPU time
			total	paths	
1 digit	11	140	10195	4946	433.59
2 digits	13	180	11145	5030	550.53
3 digits	15	208	11974	5031	628.97
4 digits	17	230	12784	5031	692.02
5 digits	18	245	13186	5031	739.71
6 digits	21	274	14384	5031	824.10
7 digits	22	293	14783	5031	896.93
8 digits	23	335	15182	5031	1016.69

(This part of the optimization process takes about 50% of the effort to solve the problem to 6 digits.) However, once the optimum has been approximately localized, the next digits (up to 8) can be achieved relatively fast with an effort that is almost linear with the number of digits required in the optimum. The method deteriorates if the academic precision 10^{-9} is required. Recall that practitioners are normally satisfied with a two or three digit solution.

Up to the level of accuracy of 10^{-6}, the projective algorithm involves from 10 to 15 iterations per outer iteration. For higher level of accuracy ($< 10^{-6}$), this number increases to 20 and 40. We believe that this is essentially due to insufficient accuracy in the computation of the search direction. Despite an iterative refinement procedure the Cholesky factorization attains its limits.

8. Conclusions

We have presented a specialized version of the analytic center cutting planes method for nonlinear multicommodity flow problems. We have discussed the influence of different aggregation/disaggregation techniques on the behaviour of the method and presented a detailed description of its sparsity-exploiting linear algebra kernel that ensures its high efficiency.

Computational experience showed the method's ability to solve fast even very large scale problems on a workstation with 64MB of memory. The method has been tested on public domain collection of large scale problems and two small, nondifferentiable optimization problems known from the literature.

The analysis of numerical results shows promise for ACCPM to become a competitive method for nonlinear multicommodity flow problems.

Acknowledgement

The authors thank Olivier du Merle for many helpful suggestions concerning the efficient implementation of ACCPM and for running the Dantzig–Wolfe algorithm.

References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows* (Prentice-Hall, Englewood Cliffs, NJ, 1993).

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen, *LAPACK Users' Guide*, (SIAM, Philadelphia, PA, 1992).

[3] D.S. Atkinson and P.M. Vaidya, "A cutting plane algorithm that uses analytic centers," *Mathematical Programming* 69 (1995) 1–43.

[4] O. Bahn, J.-L. Goffin, J.-P. Vial and O. du Merle, "Experimental behaviour of an interior point cutting plane algorithm for convex programming: An application to geometric programming," *Discrete Applied Mathematics* 49 (1994) 3–23.

[5] O. Bahn, O. du Merle, J.-L. Goffin and J.-P. Vial, "A cutting plane method from analytic centers for stochastic programming," *Mathematical Programming* 69 (1995) 45–73.
[6] I.C. Choi and D. Goldfarb, "Exploiting special structure in a primal–dual path following algorithm," *Mathematical Programming* 58 (1993) 33–52.
[7] R.W. Cottle, "Manifestations of the Schur complement," *Linear Algebra and its Applications* 8 (1974) 189–211.
[8] G.B. Dantzig and P. Wolfe, "The decomposition algorithm for linear programming," *Econometrica* 29 (4) (1961) 767–778.
[9] G. de Ghellinck and J.-P. Vial, "A polynomial Newton method for linear programming," *Algorithmica* 1 (1986) 425–453.
[10] I.S. Duff, A.M. Erisman and J.K. Reid, *Direct Methods for Sparse Matrices* (Oxford University Press, New York, 1987).
[11] J.M. Farvolden, W.B. Powell and I.J. Lustig, "A primal partitioning solution for the arc-chain formulation of a multicommodity network flow problem," *Operations Research* 41 (1993) 669–603.
[12] M. Fukushima, "On the dual approach to the traffic assignment problems," *Transportation Research B* 18 (1984) 235–245.
[13] E.M. Gafni and D.P. Bertsekas, "Two-metric projection methods for constrained optimization," *SIAM Journal on Control and Optimization* 22 (6) (1984) 936–964.
[14] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright, "Sparse matrix methods in optimization," *SIAM Journal on Scientific and Statistical Computing* 5 (1984) 562–589.
[15] J.-L. Goffin, A. Haurie and J.-P. Vial, "Decomposition and nondifferentiable optimization with the projective algorithm," *Management Science* 38 (2) (1992) 284–302.
[16] J.-L. Goffin, A. Haurie, J.-P. Vial and D.L. Zhu, "Using central prices in the decomposition of linear programs," *European Journal of Operational Research* 64 (1993) 393–409.
[17] J.-L. Goffin, Z. Luo, and Y. Ye, "On the complexity of a column generation algorithm for convex or quasiconvex feasibility problems," in W.W. Hager, D.W. Hearn and P.M. Pardalos, eds., *Large Scale Optimization: State of the Art* (Kluwer Academic Publishers, Dordrecht).
[18] G.H. Golub and C. Van Loan, *Matrix Computations*, 2nd ed. (Johns Hopkins University Press, Baltimore, MD, 1989).
[19] J. Gondzio and T. Terlaky, "A computational view of interior point methods for linear programming," in J. Beasley, ed., *Advances in Linear and Integer Programming* (Oxford University Press, Oxford) 1996, 103–144.
[20] C.C. Gonzaga, "Path following methods for linear programming," *SIAM Review* 34 (1992) 167–227.
[21] D. Hearn and S. Lawphongpanich, "A dual ascent algorithm for traffic assignment problems," *Transportation Research B* 24 (1990) 423–430.
[22] D. Hearn, S. Lawphongpanich, and J. Ventura, "Restricted simplicial decomposition: Computation and extensions," *Mathematical Programming Study* 31 (1987) 99–118.
[23] A.L. Hipolito, "A weighted least squares approach to direction finding in mathematical programming," Ph.D. Thesis, Dept. of Industrial Engineering, University of Florida, Gainesville, FL (1993).
[24] K.L. Jones, I.J. Lustig, J.M. Farvolden and W.B. Powell, "Multicommodity network flows: The impact of formulation on decomposition," *Mathematical Programming* 62 (1993) 95–117.
[25] N. Karmarkar, "A new polynomial time algorithm for linear programming," *Combinatorica* 4 (1984) 373–395.
[26] J.E. Kelley, "The cutting plane method for solving convex programs," *Journal of the SIAM* 8 (1960) 703–712.
[27] K.C. Kiwiel, *Methods of Descent for Nondifferentiable Optimization*, Lecture Notes in Mathematics, Vol. 1133 (Springer, Berlin, 1985).
[28] K.C. Kiwiel, "Proximity control in bundle methods for convex nondifferentiable optimization," *Mathematical Programming* 46 (1990) 105–122.
[29] L.S. Lasdon, *Optimization Theory for Large Scale Systems* (Macmillan, New York, 1970).
[30] C. Lemaréchal, "Nondifferentiable optimization", in G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd, eds., *Optimization*, Handbooks in Operations Research and Management Science, Vol. 1, (North-Holland, Amsterdam, 1989) pp. 529–572.
[31] C. Lemaréchal, A. Nemirovskii and Yu. Nesterov, "New variants of bundle methods," *Mathematical Programming* 69 (1995) 177–204.

[32] M. Minoux, *Mathematical Programming: Theory and Algorithms* (Wiley, New York, 1986).
[33] J.E. Mitchell and M.J. Todd, "Solving combinatorial optimization problems using Karmarkar's algorithm," *Mathematical Programming* 56 (1992) 245–284.
[34] Yu. Nesterov, "Cutting plane algorithms from analytic centers: Efficiency estimates," *Mathematical Programming. Series B* 69 (1995) 149–176.
[35] R.T. Rockafellar, *Monotropic Flows and Network Optimizations* (Wiley, New York, 1984).
[36] H. Schramm and J. Zowe, "A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results," *SIAM Journal on Optimization* 2 (1992) 1211–1252.
[37] G. Sonnevend, "New algorithms in convex programming based on a notion of 'centre' (for systems of analytic inequalities) and on rational extrapolation," in: K.H. Hoffmann, J.B. Hiriat-Urruty, C. Lemarechal, and J. Zowe, eds., *Trends in Mathematical Optimization: Proceedings of the 4th French–German Conference on Optimization in Irsee, West-Germany, April 1986*, Vol. 84 of International Series of Numerical Mathematics (Birkhäuser, Basel, 1988), pp. 311–327.
[38] Y. Ye, "A potential reduction algorithm allowing column generation," *SIAM Journal on Optimization* 2 (1992) 7–20.