Comparing Branch-and-price Algorithms for the Unsplittable Multicommodity Flow Problem

Filipe Alvelos, J. M. Valério de Carvalho
Universidade do Minho
Departamento de Produção e Sistemas
4710 053 Braga, Portugal
Phone: +351 253 604751, fax: +351 253 604741
Email: {falvelos,vc}@dps.uminho.pt

Abstract

In this paper we address branch-and-price algorithms for the unsplittable multicommodity flow problem. This problem is defined over a capacitated network in which we intend to route a set of commodities, each one with a given origin, destination and demand, at minimal cost, without exceeding the arc capacities. Furthermore, the flow of each commodity must be routed using a single path.

Formulating this problem with decision variables representing flows on each arc for each commodity gives rise to a large linear (integer) program with two types of constraints: flow conservation constraints and mutual capacity constraints. Based on the Dantzig-Wolfe principle we obtain two different decompositions (path and knapsack) depending on which type of constraints defines the subproblem. In order to solve the integer problem, we combine column generation and branch-and-bound, developing branching rules that preserve the structure of the subproblem in the branch-and-bound tree for both decompositions. For the path decomposition, we compare the developed branching rule with one previously presented by Barnhart, Hane and Vance (2000). We present preliminary computational results for the two decompositions, with different branching rules, and we compare them with the ones given by a general purpose integer programming solver.

Keywords: Integer programming, routing.

1 Introduction

Multicommodity flow problems (MFPs) are defined over a network in which we intend to route a set of commodities at minimal cost, subject to capacity constraints on the total flow that can traverse the arcs.
Multicommodity flow problems have been the subject of interest of the research community for its applications (namely in telecommunications networks, transportation/distribution systems and production planning) and for its role as a typical mathematical programming model with block-angular structure, thus being a representative problem (maybe the most used) for which several decomposition methods can be applied and tested.

Most of the work on MFPs is about its linear version, where the demand of a commodity can be splitted along different paths. However, for some applications it may be required that the demand of a commodity must be routed along a single path, which is the case in some telecommunications networks, where the commodities are associated with streams of traffic between different pairs of nodes and non-bifurcated routing is a requisite. That leads to the relevance of the version of the MFP that we consider in this paper, namely the unsplittable multicommodity flow problem (UMFP).

Most work on versions of the UMFP is related with approximation algorithms (for instance, [11]). Two exceptions are the heuristic approach presented in [14] and the exact algorithm presented in [2]. Part of our present work is related with this last reference, where a branch-and-price-and-cut algorithm is devised based on the path decomposition. In this paper we follow that work, but deriving a different branching rule. The other part is concerned with the knapsack decomposition (used before to obtain lower bounds to network design problems [4, 9] with solution approaches different from the one used here). Surveys about branch-and-price can be found in [3, 12, 15].

The main purpose of this work is to compare decomposition approaches (and also branching rules for the path decomposition) that can be applied to the UFMP, given that they possess inherently different properties. The main advantage of the path decomposition is that the size of the linear programs solved in a column generation scheme is, in general, considerably smaller than the size of the original linear program. As for the knapsack decomposition, the advantage of its use is that the lower bound it gives is, in general, tighter than the one given by the linear relaxation of the original formulation.

In our view the contributions of this paper include: (1) the development of a new decomposition algorithm for the UMFP, (2) the development of a new branching rule to the path decomposition, (3) the comparison of the practical behaviour of each algorithm based on preliminary computational tests and the identification of some important issues to improve their efficiency.

2 Original Formulation

We consider a network formed by a set of n nodes, represented by N, and a set of m arcs, represented by A. We use an index $i=\{1,...,n\}$ to represent a node and a pair of indexes ij to represent an arc which has origin in node i and destination in node j. We define a set K of h commodities, indexed by k. Each commodity k is characterised by an origin, o^k, a destination, d^k, and an integer demand, r^k, that is the number of units that are supplied at its origin and that are required at its destination. We also define an integer capacity u_{ij} associated with each arc of the network and a unit cost, c_{ij}^k, associated with the cost of unit of flow of commodity k traversing arc ij. We assume $c_{ij}^k > 0, \ \forall ij \in A, \ \forall k \in K$.

The original formulation is obtained using decision variables that represent the proportion of the demand of each commodity that flows in each arc. Forcing those variables to be binary is the same as forcing every flow of every commodity to be routed along a single path.

The decision variables are represented as x_{ij}^k. The original formulation is as follows:

$$Min \sum_{k \in K} \sum_{ij \in A} c_{ij}^k r^k x_{ij}^k \tag{O}$$

$$\sum_{ij \in A} x_{ij}^k - \sum_{ji \in A} x_{ji}^k = \begin{cases} 1 \ if \ i = o^k \\ -1 \ if \ i = d^k \\ 0 \ if \ i \neq o^k, i \neq d^k \end{cases}, \forall i \in N, \forall k \in K \tag{1}$$

$$\sum_{k \in K} r^k x_{ij}^k \leq u_{ij}, \forall ij \in A \tag{2}$$

$$x_{ij}^k \in \{0,1\}, \forall k \in K, \forall ij \in A. \tag{3}$$

Constraints (1) are flow conservation constraints. They state that, for each commodity, the difference between the flow that enters a node and the flow that leaves that node is equal to the supply/demand of that node. Constraints (2) are capacity constraints. They state that the total flow on each arc must be less than or equal to its capacity.

3 Path Decomposition

We apply the Dantzig-Wolfe decomposition principle [5] to the original formulation, defining the subproblem with constraints (1) and (3). In this way, the subproblem is a set of h shortest path problems, one for each commodity, thus it has no extreme rays, and each extreme point of subproblem k is associated with a path from o^k to d^k. We denote the set of indexes p of all the extreme points of the subproblem of commodity k by P^k. We represent a given extreme point, with index p and associated with the subproblem of a commodity k, by δ^{pk}. The entry in that vector corresponding to the original variable x_{ij}^k is denoted by δ_{ij}^{pk}, that takes value *1* if arc ij belongs to the path p of commodity k, and *0* otherwise. The cost of one extreme point, c^{pk}, is given by

$$c^{pk} = \sum_{ij \in A} \delta_{ij}^{pk} c_{ij}^k .$$

At last, we define the weight variable associated with each extreme point, δ^{pk}, as y^{pk}. According to the Dantzig-Wolfe decomposition principle and relaxing the binary requirements we get the following master problem:

$$Min \sum_{k \in K} \sum_{p \in P^k} c^{pk} r^k y^{pk} \tag{P}$$

$$\sum_{p \in P^k} y^{pk} = 1, \forall k \in K \tag{4}$$

$$\sum_{k \in K} \sum_{p \in P^k} \delta_{ij}^{pk} r^k y^{pk} \leq u_{ij} \tag{5}$$

$$y^{pk} \geq 0, \forall k \in K, \forall p \in P^k .$$

A decision variable of (P), y^{pk}, can be seen as the proportion of the demand of the commodity k that is routed in path p. Constraints (4) are convexity constraints. Constraints (5) are the capacity constraints after the redefinition of

variables subjacent to the Dantzig-Wolfe decomposition principle (when there are no extreme rays): a solution to (O) can be represented as a convex combination of the extreme points of the subproblem(s). Given a solution of (P) we can recover a solution of (O) by using:

$$x_{ij}^{k} = \sum_{p \in P^{k}} \delta_{ij}^{pk} y^{pk}, \forall ij \in A, \forall k \in K \ . \tag{6}$$

The linear relaxation of the path decomposition just presented gives a lower bound that is equal to the one given by the original formulation, since the subproblem has the integrality property [7] (that is, all extreme points are integer). However, the efficiency of the solution methods of the linear relaxation can be very different. The dimension of the basis in the original formulation is $n.h+m$ while, in the path decomposition, it is $h+m$. Since the basis dimension is the main factor to simplex methods efficiency, we can expect the path decomposition to be more efficient in larger instances, given that we use a column generation scheme to deal with the exponential number of columns, with respect to the dimension of the network.

In the resolution of linear relaxation of the path formulation using column generation, a restricted master problem (RMP), a problem where not all paths are included, is considered. In each iteration, the RMP is optimised and the attractiveness of the paths that are not present in the RMP is evaluated by solving a subproblem (more precisely, one subproblem for each commodity) that uses the values of the dual variables. This subproblem corresponds to a shortest path problem for each commodity. A detailed description of a similar procedure can be found in [1].

In order to obtain the optimal binary solution, we apply branch-and-bound. However, since column generation is used to solve the linear relaxations of each node of the tree, it is necessary to combine branch-and-bound and column generation, thus using a branch-and-price algorithm. The main issue when using branch-and-price is to derive branching rules that are compatible with the subproblem structure: after branching, the subproblem should not become significantly more difficult to solve. A general principle to achieve this is to perform branching on the original variables. In the case of the UMFP, we note that the value of each original variable can be obtained through (6).

The first branching rule that we implemented is taken from [2]. The main idea is based on identifying a node where the flow of a commodity is first split to different paths and then branch by forbidding the flow on subsets of arcs that leave that node. This branching rule is motivated by the difficulty in solving the shortest path (sub)problem where some arcs are forced to belong to the solution, while forbidding a set of arcs only requires their exclusion from the network. A disadvantage of this branching rule is that a feasible solution may belong to the solution space of different nodes of the tree.

The main idea of the branching rule that we developed is to explicitly include branching constraints in RMP and modifying the subproblem accordingly. Thus, after selecting a fractional original variable, x_{ij}^{k}, two branches are created by inserting, in the RMP of each of the new nodes, one of following constraints

$$\sum_{p \in P^{k}} \delta_{ij}^{pk} y^{pk} \leq 0 \text{ and } \sum_{p \in P^{k}} \delta_{ij}^{pk} y^{pk} \geq 1 .$$

The dual variables of these new constraints are then used to modify the costs of the arcs when solving the shortest path subproblems.

In [2] a way of incorporating lifted cover inequalities (LCIs) in the branch-and-price algorithm (thus turning it into a branch-and-price-and-cut algorithm) is presented. The LCIs used there are simple LCIs. We implemented the same approach but using general LCIs. In every node of the branch-and-price tree the corresponding problem is solved to optimality. Before branching, we use the heuristic proposed in [8] to try to detect a violated LCI for each saturated arc (what implies solving a set of knapsack problems). That is done based on the original variables values. If LCIs are found, then the corresponding constraints are inserted in the RMP, after translated to path variables, and the problem is reoptimised using column generation, taken into account the dual variables of the LCIs constraints of the RMP in the subproblems. If no LCIs are found and the node requires branching, new nodes are created.

4 Knapsack Decomposition

In the knapsack decomposition, we apply the Dantzig-Wolfe decomposition principle to the original formulation, defining the subproblem with constraints (2) and (3). In this way, the subproblem is a set of m knapsack problems, one for each arc, thus it has no extreme rays, and each extreme point of subproblem of arc ij is associated with a set of commodities that, together, can use arc ij without exceeding its capacity.

We denote the set of indexes q of all the extreme points of the subproblem of arc ij by Q^{ij}. We represent a given extreme point, with index q and associated with the subproblem of an arc ij, by ρ^{qij}. The entry in that vector corresponding to the original variable x_{ij}^{k} is denoted by ρ_{k}^{qij}. Thus, the cost of one extreme point, c^{qij}, is given by

$$c^{qij} = \sum_{k \in K} \rho_{k}^{qij} c_{ij}^{k} \ .$$

At last, we define the weight variable associated with each extreme point, ρ^{qij}, as z^{qij}. According to the Dantzig-Wolfe decomposition principle and relaxing the binary requirements we get the following master problem:

$$Min \sum_{ij \in A} \sum_{q \in Q^{ij}} c^{qij} r^k z^{qij} \tag{K}$$

$$\sum_{q \in Q^{ij}} z^{qij} \leq 1, \forall ij \in A \tag{7}$$

$$\sum_{ij \in A} \sum_{q \in Q^{ij}} \rho_k^{qij} z^{qij} - \sum_{ji \in A} \sum_{q \in Q^{ji}} \rho_k^{qji} z^{qji} = \begin{cases} 1 \text{ if } i = o^k \\ -1 \text{ if } i = d^k \\ 0 \text{ if } i \neq o^k, i \neq d^k \end{cases}, \forall i \in N, \forall k \in K \tag{8}$$

$$z^{qij} \geq 0, \forall ij \in A, \forall q \in Q^{ij}.$$

Constraints (7) are convexity constraints. We note that the origin is an extreme point for each subproblem, what justifies their sense. Constraints (8) are the flow conservation constraints after the redefinition of variables subjacent to the Dantzig-Wolfe decomposition principle. Given a solution of (K) we can recover a solution of (O) by using:

$$x_{ij}^k = \sum_{ij \in Q^{ij}} \rho_k^{qij} z^{qij}, \forall ij \in A, \forall k \in K. \tag{9}$$

The basis dimension of (K) is the same as the basis dimension of the original formulation (O). The main advantage of its use is that, in general, it gives a better lower bound to the binary problem than the linear relaxation of (O). The subproblems are binary knapsack problems that do not have the integrality property.

We now briefly describe the branching rule that was implemented for this decomposition. In a given node of the branch-and-price tree, optimised by column generation, the original variables values can be computed through (9). If it is necessary to perform branch, we select one original variable, x_{ij}^k, with fractional value and derive two new branches by forcing, in one, $x_{ij}^k = 0$ and, in the other, $x_{ij}^k = 1$. In both cases we have to remove some columns relative to the commodity k and arc ij prior to optimizing the resulting node. In the branch $x_{ij}^k = 0$ we remove the columns associated with arc ij that include commodity k and we also exclude commodity k from the subproblem associated with that arc. In this way we guarantee that the optimal solution of the node will have $x_{ij}^k = 0$. In the branch $x_{ij}^k = 1$ we remove the columns associated with arc ij that do not include commodity k and we force the solution of the subproblem of arc ij to include commodity k. Also, we modify the sense of the convexity constraint of arc ij to equality.

5 Computational Results

We implemented the proposed algorithms and developed a program that solves the UMFP directly using Cplex 7.5 [10] in C++ using the programming environment Microsoft Visual Studio 6.0. The same general purpose solver was used to solve the restricted master problems of both decompositions and to solve all the knapsack problems (in finding LCIs in the path decomposition and in solving the subproblems in the knapsack decomposition). For the shortest path problems we implemented a L-2queue algorithm [6]. We also developed a random instance generator that uses the LEDA class library [13].

We decided to generate instances with 32 nodes that can be divided in two main classes of 24 instances each, here denoted by I and II. For the instances of class I the ratio mean capacity of arcs / mean demand of commodities is 1,5. For the instances of class II that ratio is 10. We combined two network densities and three number of commodities (as showed in Table 1) thus obtaining 6 groups of instances for each class. For each group, we generated four instances by choosing a small or large cost variance and by choosing if the cost of the arc is the same for all commodities or not. All these instances are available at *http://sarmento.eng.uminho.pt/dps/falvelos*.

All the reported results were obtained on a personal computer with a Pentium 4, 2 GHz processor, 1 GB of RAM, running Windows XP Professional Edition.

The meaning of the notation used in Table 1 is as follows: ‘c’ stands for the cplex program, ‘p1’ for the path decomposition with the branching rule developed (with a best search strategy and general LCIs), ‘p2’ for the path decomposition with the Barnhart et al. rule (depth first search strategy and simple LCIs as in the original description of the algorithm), ‘k’ for the knapsack decomposition (best first strategy). The percentual gap refers to the value obtained in the root of the branch-and-bound tree (that includes the use of LCIs in the path decomposition algorithms).

Class	*m/n*	*h/n*	Gap (%)				Number of nodes				Time (in seconds)				Obs.
			c	p1	p2	k	c	p1	p2	k	c	p1	p2	k	
I	3	1,5	7,2	1,3	1,6	0,1	12,3	22,0	141,5	1,5	2,9	16,2	50,4	409,2	(1)
I	3	6	3,1	0,9	1,2	–	9565	–	–	–	2983,6	–	–	–	(2)
I	3	10	0,7	0,1	0,2	–	4861	–	–	–	1262,9	–	–	–	(2)
I	10	1,5	16,8	2,1	2,2	0,2	25,3	47,0	169,3	8,5	6,0	7,2	54,5	7,3	(1)
I	10	6	9,7 8,2 11,1	0,7 0,4 0,3	0,8 0,6 0,8	0,3 0,1 –	1538 531 –	935 49 159	– – –	– – –	1108,2 254,4 –	1739,7 109,7 352,7	– – –	– – –	(3)
I	10	10	6,0	0,5	0,6	–	965,0	–	–	–	2375,7	–	–	–	(4)
II	3	1,5	11,1	3,9	4,4	1,6	103,5	355,8	–	–	16,4	335,8	–	–	(5)
II	3	6	2,5 1,4 1,7	1,1 0,1 0,3	1,2 0,2 0,4	–	669 282 2663	– 238 1019	– – –	– – –	2324,8 40,7 1239,8	– 862,7 3281,6	– – –	– – –	(3)
II	3	10	–	–	–	–	–	–	–	–	–	–	–	–	(6)
II	10	1,5	4,2	0,4	0,4	0,0	1,0	2,5	3,0	1,3	0,3	0,4	0,5	490,1	(1)
II	10	6	7,6	0,3	0,4	–	605,8	349,8	–	–	226,2	384,7	–	–	(5)
II	10	10	5,5 4,3 6,6	0,3 0,5 0,4	0,4 0,7 0,4	– – –	231 556 1968	267 – 331	– – –	– – –	629,2 1214,2 1160,8	1366,4 – 1461,3	– – –	– – –	(3)

Table 1 - Instance characteristics and computational results.

(1) Average values for four instances. (2) An integer optimal solution was found in less than one hour only for one instance, with Cplex. The presented values are for that instance. The other algorithms could not found an optimal integer solution. For the knapsack relaxation, one hour was not enough to solve the linear relaxation. (3) One instance could not be solved in one hour for all algorithms. Presented results are for the other three instances, where '–' signals that the given instance could not be solved in one hour by the given algorithm. (4) An integer optimal solution was found in less than one hour only for two instances, with Cplex. The presented values are the average of those two instances. The other algorithms could not found an optimal integer solution. For the knapsack relaxation, one hour was not enough to solve the linear relaxation. (5) Average values for four instances for the algorithms that solved them all. (6) None of the four instances could be solved in the time limit given by any algorithm.

We must make one remark before taking conclusions about the comparative behaviour of the different approaches: all the knapsack problems (for both decompositions) were solved by Cplex; given that we may expect specialised algorithms to be much more efficient, some care should be taken when comparing the computational time. Furthermore, some improvement may be done in the data structures used. In fact, taking the path decomposition as an example, the average (for all 48 instances) time spent in solving the RMPs and the shortest path (sub)problems is less than 10%.

In general, Cplex gives better results than both decomposition approaches. However, for some instances, in particular the dense ones with medium number of commodities, the path decomposition with the developed branching rule is more effective.

We note that, given the significant difference in the gap values, more sophisticated search strategies and the combination of the two branching rules presented for the path decomposition (that is using our branching strategy but on subsets of variables as in the Barnhart et al. rule, which may lead to more balanced search trees) can produce better results.

Only for one group of instances the results of the knapsack decomposition are slightly better than the ones of the other algorithms (I-10–1,5). However, in general, the behaviour of the knapsack decomposition is very poor when compared with the other algorithms, although the quality of the lower bound given by the linear relaxation is effectively much better (for the instances for which it could be computed). Besides the remark made above about the algorithm used to solve its subproblems, the RMPs are much difficult to solve and the number of iterations of the column generation procedure is much bigger than for the path decomposition.

6 Conclusions

In this paper we presented two branch-and-price algorithms for the unsplittable multicommodity flow problem. The first one is based on a path decomposition for which a branching rule, that gave better computational results for the tested instances than the one presented in [2], was developed. The second one is based on a knapsack decomposition,

which allows obtaining better lower bounds in the nodes of the branch-and-price tree. The quality of the lower bounds was proved empirically by the computational tests performed. However, it turned out that significant improvements must be done for it to become computationally attractive, at least for the instances that were tested.

For the most part of the tested instances, the general integer programming solver Cplex 7.5 was more efficient than the decomposition approaches presented. We noted, however, that several improvements can be done in the path decomposition implementation, some of them described in the previous section. Also, these preliminary tests were done with small instances for which the original formulation, the one solved by Cplex, does not require a prohibitive amount of memory, what can be the case for larger instances. Given this and noting that, for some instances, it gave better results, path decomposition, with the branching rule presented in this paper, deserves further testing, in particular for larger instances.

In a near future we expect to improve the implementations here succinctly described and extend the computational tests in order to perform a more detailed analysis of what approach is more suitable for different types of instances.

Acknowledgments

We thank Carina Pimentel for programming the random instance generator that were used and part of the code, and also for the execution of the computational tests. This work was partially supported by Fundação para a Ciência e Tecnologia (Projecto POSI / 1999 / SRI / 35568) and Centro de Investigação Algoritmi da Universidade do Minho, and was developed in the Grupo de Engenharia Industrial e de Sistemas.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, "Network Flows". Prentice Hall, 1993.

[2] C. Barnhart, C.A. Hane, P.H. Vance, "Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems". Operations Research, 48, 318-326, 2000.

[3] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance, "Branch-and-price: column generation for solving huge integer programs". Operations Research, 46, 316-329, 1998.

[4] T.G. Crainic, A. Frangioni, B. Gendron, "Bundle-based relaxation methods for multicommodity capacitated fixed charge network design". Discrete Applied Mathematics, 112, 73-99, 2001.

[5] G.B. Dantzig, P. Wolfe, "Decomposition principle for linear programs". Operations Research, 8, 101-111, 1960.

[6] G. Gallo, S. Pallottino, "Shortest path algorithms". Annals of Operations Research, 13, 3-79, 1988.

[7] A.M. Geoffrion, "Lagrangean relaxation for integer programming". Mathematical Programming Study, 2, 82-114, 1974.

[8] Z. Gu, G.L. Nemhauser, M.W.P. Savelsbergh, "Lifted cover inequalities for 0-1 integer programs: computation". INFORMS Journal on Computing, 10, 427-437, 1998.

[9] K. Holmberg, D. Yuan, "A Lagrangean heuristic based branch-and-bound approach for the capacitated network design problem". Operations Research, 48, 461-481, 2000.

[10] ILOG, "CPLEX 7.1, User's Manual". 2001.

[11] S.G. Kolliopoulos, C. Stein. "Experimental evaluation of approximation algorithms for single-source unsplittable flow". Proc. 7th International Integer Programming and Combinatorial Optimization Conference, Graz, Austria, 1999, 328-344. Lecture Notes in Computer Science, 1610, Springer-Verlag.

[12] M.E. Lübbecke, J. Desrosiers, "Selected topics in column generation", Les Cahiers de GERAD G-2002-64, 2002.

[13] K. Mehlhorn, S. Näher, "LEDA - A platform for combinatorial and geometric computing". Cambridge University Press, 1999.

[14] Y. Wang, Z. Wang. "Explicit routing algorithms for internet traffic engineering". Proc. International Conference on Computer Communication Networks, Boston, USA, 1999,

[15] W.E. Wilhelm, "A Technical Review of Column Generation in Integer Programming". Optimization and Engineering, 2, 159-200, 2001.