# A LAGRANGIAN HEURISTIC BASED BRANCH-AND-BOUND APPROACH FOR THE CAPACITATED NETWORK DESIGN PROBLEM

**KAJ HOLMBERG and DI YUAN**

*Department of Mathematics, Linköping Institute of Technology S-581 83 Linköping, Sweden*
*kahol@mai.liu.se • diyua@mai.liu.se*

(Received February 1997; revisions received December 1997, May 1998; accepted July 1998)

The capacitated network design problem is a multicommodity minimal cost network flow problem with fixed charges on the arcs and is well known to be NP-hard. The problem type is very common in the context of transportation networks, telecommunication networks, etc. In this paper we propose an efficient method for this problem, based on a Lagrangian heuristic within a branch-and-bound framework. The Lagrangian heuristic uses a Lagrangian relaxation to obtain easily solved subproblems and solves the Lagrangian dual by subgradient optimization. It also includes techniques for finding primal feasible solutions. The Lagrangian heuristic is then embedded into a branch-and-bound scheme that yields further primal improvements. Special penalty tests and cutting criteria are developed. The branch-and-bound scheme can either be an exact method that guarantees the optimal solution of the problem or be a quicker heuristic. The method has been tested on networks of various structures and sizes. Computational comparisons between this method and a state-of-the-art mixed-integer code are presented. The method is found to be capable of generating good feasible solutions to large-scale problems within reasonable time and data storage limits.

The fixed charge network design model can be used in various real-life applications, such as construction of new streets in traffic networks, construction of new links in transportation networks, etc. A survey of network design problems is given in Magnanti and Wong (1984). Because of the rapid technological developments of telecommunication networks and computer networks during the last decade, the same model has applications in these networks as well.

Because of its usefulness in these applications, much research has been carried out in the area of network design. In particular, we found that the uncapacitated version of this type of problem has been quite well studied in several research reports, some of them being Magnanti et al. (1986), Balakrishnan et al. (1989), Hellstrand et al. (1992), and Lamar et al. (1990). In Holmberg and Hellstrand (1998) an efficient solution method based on a Lagrangian heuristic and branch-and-bound has been developed for it. Research has also been carried out on the capacitated model of the network design problem. In Gendron and Crainic (1994), for example, different relaxation schemes are studied and discussed with heuristics for yielding feasible solutions. However, compared to the uncapacitated case, very few references can be found in the literature for capacitated network design. While in some cases the network is, or can be assumed to be, uncapacitated, a capacitated model is more general and often much more suitable because capacity constraints often do arise in real-life applications.

In this paper we study the capacitated network design problem where it is assumed that only one origin and one destination exist for each commodity. Depending on the application, the commodities in the network model can be either goods in a transportation network or data signals in a computer network. The application that we have encountered is design and/or improvement of telecommunication networks where arcs are telephone links (copper cables or optical fiber links) and demand represents telephone calls (and in the future computer or video data). Moreover, in the application of telecommunication, networks tend to be very large, which poses challenges when efficient algorithms are to be developed.

In following sections, we describe a solution method based on a Lagrangian heuristic within a branch-and-bound framework. The Lagrangian heuristic provides both upper and lower bounds to the problem, and the branch-and-bound scheme ensures that the optimal solution is found. However, because the problem is NP-hard, we should not expect to be able to solve very large problems optimally in reasonable time. Thus we are also interested in a fast method that can generate near-optimal solutions for large problems. Therefore, we introduce the possibility of modifying the branch-and-bound method into a heuristic by including schemes to fix variables in order to improve the solution.

Lagrangian heuristics and branch-and-bound are often used for problems that are NP-hard. Because a method based on the same principle has been successfully applied to the uncapacitated model, it is interesting to investigate how the same principle behaves in the capacitated case. Basically, a Lagrangian heuristic is composed of a suitable Lagrangian relaxation of the problem, an efficient subgradient optimization procedure for solving the Lagrangian dual, and a primal heuristic for yielding feasible solutions. The Lagrangian relaxation must be designed in such a way that it results in an easily solved subproblem, but such that the solution of it still be yield lower bounds strong enough to be used efficiently in a branch-and-bound scheme. Clearly, a large part of this relies on the subgradient search procedure. If the

*Subject classifications:* Programming, integer, algorithms, relaxation/subgradient, branch-and-bound: for the capacitated network design problem. Networks/graphs, multicommodity: fixed charges.
*Area of review:* OPTIMIZATION.

subgradient search is inefficient, convergence of the method will be slow. Moreover, it is obviously vital to obtain good feasible solutions, yielding upper bounds, because the whole method relies heavily on lower and upper bounds. Finally, the branch-and-bound procedure should be based on the information generated by the Lagrangian heuristic. When a branch-and-bound scheme is combined with a Lagrangian heuristic, a large amount of subgradient iterations will probably reduce the gap between the lower and upper bounds and thus the size of the tree, but it will require extra computational time. On the other hand, too few subgradient iterations will most likely lead to a huge branch-and-bound tree. Therefore, a good balance must be achieved between the total number of iterations and the branch-and-bound tree size.

The most critical point of the method is to make all these parts work well together to obtain the best overall performance. However, this is not a simple task because there are various possibilities of constructing the Lagrangian relaxation, subgradient search procedure, or other approaches to improve the lower bound, the primal heuristic, and those design issues related to the branch-and-bound algorithm.

The rest of this paper is organized as follows. In §1 we present the mathematical model and some preview of our problem. The next section is dedicated to a detailed description of the Lagrangian heuristic used. In §3 we describe how a branch-and-bound algorithm can be constructed based on the Lagrangian heuristic and how such an algorithm can be used to improve the feasible solutions obtained as well as to obtain the optimum. Section 4 discusses the solution of the network design problem by using the state-of-the-art MIP-code CPLEX. In §5 computational results are given, and in §6 some conclusions are drawn.

## 1. PROBLEM FORMULATION AND PREVIEW

We consider a network that is represented by a set of nodes $\mathcal{N}$ and a set of directed arcs $\mathcal{A}$. Furthermore, a set of commodities, $\mathcal{C}$, is given. Each commodity, $k$, has one origin and one destination, denoted by $o(k)$ and $d(k)$. The quantity of commodity $k$ that is to be sent from $o(k)$ to $d(k)$ (the demand) is denoted by $r^k$.

There are two kinds of costs involved in the network. The routing costs increase linearly with the amount of flow, and we let $c_{ij}^k$ denote the cost of shipping one unit of commodity $k$ on arc $(i,j)$. Additionally, a fixed cost, $f_{ij}$, will be added when any amount of flow is sent on arc $(i,j)$. Furthermore, each arc $(i,j)$ has a limited capacity, $u_{ij}$, on the total flow on the arc.

Two sets of variables are introduced, the flow variables and the design variables, defined below:

$x_{ij}^k$ = the flow of commodity $k$ on an arc $(i,j)$,

$$y_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is used,} \\ 0 & \text{otherwise.} \end{cases}$$

Given these definitions with the objective to minimize the total cost, the mathematical model can be formulated as

$$\text{[CNDP]} \quad v^* = \min \sum_{k\in\mathcal{C}} \sum_{(i,j)\in\mathcal{A}} c_{ij}^k x_{ij}^k + \sum_{(i,j)\in\mathcal{A}} f_{ij} y_{ij}$$

$$\text{s.t.} \sum_{j:(i,j)\in\mathcal{A}} x_{ij}^k - \sum_{j:(j,i)\in\mathcal{A}} x_{ji}^k = b_i^k \quad \forall i \in \mathcal{N},\ \forall k \in \mathcal{C}, \tag{1}$$

$$\sum_{k\in\mathcal{C}} x_{ij}^k \leqslant u_{ij} y_{ij} \quad \forall (i,j) \in \mathcal{A}, \tag{2}$$

$$x_{ij}^k \leqslant d_{ij}^k y_{ij} \quad \forall (i,j) \in \mathcal{A},\ \forall k \in \mathcal{C}, \tag{3}$$

$$x_{ij}^k \geqslant 0 \quad \forall (i,j) \in \mathcal{A},\ \forall k \in \mathcal{C}, \tag{4}$$

$$y_{ij} \in \{0,1\} \quad \forall (i,j) \in \mathcal{A}, \tag{5}$$

where

$$d_{ij}^k = \min(r^k, u_{ij}) \quad \text{and} \quad b_i^k = \begin{cases} r^k & \text{if } i = o(k), \\ -r^k & \text{if } i = d(k), \\ 0 & \text{otherwise.} \end{cases}$$

Constraint set (1) contains the network flow conservation equations, constraint set (2) contains the arc capacity constraints, and constraint set (3) contains linking (forcing) constraints that ensure that no flow is allowed on arc $(i,j)$ unless the fixed charge $f_{ij}$ is paid. We assume that all coefficients are integral and nonnegative, and that $u$ and $r$ are positive. Obviously, CNDP is a linear mixed-integer programming problem with $|\mathcal{A}|$ binary design variables and $|\mathcal{A}||\mathcal{C}|$ continuous flow variables.

The case of already existing arcs in the network can be modeled by setting the corresponding $f_{ij}$ to 0, which implies that arc $(i,j)$ can be used free of charge. Without loss of generality we can set $y_{ij} = 1$ if $f_{ij} = 0$. Thus the network improvement problem, where the task is to improve the network design on a given network rather than design the whole network from scratch, can be treated as a special case of CNDP.

We notice that if a commodity has multiple origins *or* destinations (but not both), it can be treated as several commodities, each with a single origin and a single destination in order to use our formulation.

We also note that the presented model is a disaggregated version of the problem because the linking constraint set (3) is redundant. It is well-known fact that a disaggregated formulation often provides much better bounds in the context of relaxations. On the other hand, a disaggregated formulation requires a much larger number of constraints. The aggregated version of CNDP would have $|\mathcal{N}||\mathcal{C}| + |\mathcal{A}|$ constraints, while the model above has $|\mathcal{A}||\mathcal{C}|$ additional constraints. Furthermore, suppose that we perform LP-relaxation on the aggregated formulation. Because there is no integral restriction left on $y_{ij}$, it will attain its minimal feasible value, which is $\Sigma_{k\in\mathcal{C}} x_{ij}^k / u_{ij}$. Therefore, we can eliminate all the $y$-variables and obtain a multicommodity network flow problem.

Such a reformulation *cannot* be performed for the disaggregated formulation because in the optimal LP-solution, $y_{ij} = \max\{(\Sigma_{k\in\mathscr{C}} x_{ij}^k/u_{ij}), (\max_{k\in\mathscr{C}}(x_{ij}^k/d_{ij}^k))\}$, which would yield a nonlinear (and nondifferentiable) objective function. Because the LP-relaxation of the disaggregated version is more tightly constrained, the related lower bound is much sharper. Consequently, it is preferable to solve the relaxation of the disaggregated formulation despite its higher complexity. For detailed comparisons, see Gendron and Crainic (1994). Computational experience also shows that in the disaggregated formulation, it is essential to keep $y_{ij}$ in constraint set (2) to make the model as strong as possible.

For fixed $y$-variables, CNDP becomes a multicommodity network flow problem. It implies that even if all coefficients are integral, the optimal $x$-solution need not to be integral. This is unlike the uncapacitated case, where the problem is decomposed into a number of shortest path problems for fixed $y$-variables, which clearly yields integral $x$-solutions.

The fact that CNDP is NP-hard is quite obvious. The uncapacitated network design problem (UNDP) is NP-hard because it generalizes the Steiner Tree problem, and CNDP is clearly a generalization of UNDP. Also, CNDP seems to have a much higher practical difficulty than UNDP according to our computational experiences.

## 2. A LAGRANGIAN HEURISTIC

In this section we develop a Lagrangian heuristic for CNDP. Such a heuristic is composed by a Lagrangian relaxation, a dual search procedure for solving the Lagrangian dual, and techniques for yielding feasible solutions.

### 2.1. Lagrangian Relaxation

To perform a Lagrangian relaxation, a suitable constraint set is chosen to be relaxed. Considering the CNDP-formulation in the previous section, the two main alternatives are to relax either the linking constraints (2) and (3), or the flow conservation constraints (1). To maintain the network structure in the relaxation, one might prefer to choose the first relaxation. The network structure in the subproblem is then represented by a number of shortest path problems, or, if the upper bounds $d_{ij}^k$ on single variables are kept, a number of minimal cost flow problems. Theoretically, the two ways of relaxation provide the same best lower bound, on the condition that the Lagrangian dual is solved to optimality in both cases. The LP-relaxation also yields the same bound because the subproblems have the integrality property. For a proof of this result, see Gendron and Crainic (1994), where also the two relaxations are compared. In the computational tests in Gendron and Crainic, subgradient optimization methods are run a certain number of iterations for both relaxations. The conclusion is that the second relaxation, relaxing the flow conservation constraints, yields a quicker and weaker procedure; i.e., the subproblems are solved much faster, but the resulting bound is significantly weaker. Gendron and Crainic therefore choose to proceed with the first relaxation.

The subproblem in the first relaxation can be strengthened by adding capacity bounds, i.e., constraint set (2) without any $y$-variables. Instead of a shortest path problem, the subproblem then becomes a multicommodity network flow problem, which is clearly much harder to solve, especially for large networks.

In this paper we use the second type of Lagrangian relaxation, obtained by relaxing the flow conservation constraints. This is the same relaxation as used in Holmberg and Hellstrand (1998) for UNDP. The main advantage of this relaxation is that it yields a very simple separable subproblem, as shown below. This property enables a large number of subgradient iterations in order to solve the Lagrangian dual. As mentioned above, although the network structure is removed in the subproblem, it does not imply weaker bounds than the other relaxation if the corresponding Lagrangian dual is solved optimally. Furthermore, in subsequent sections we show that the solution of the subproblem provides valuable information for the branch-and-bound procedure, and that the separability is quite useful. Actually, the separable unit appearing in the subproblem, a single arc, will coincide with the unit directly affected by branching in our branch-and-bound procedure.

By relaxing the constraint set (1) using multipliers $w_i^k$, we obtain the following Lagrangian dual:

$$[\mathbf{LD}] \quad v_L = \max \varphi(w).$$

For fixed multipliers, $w = \bar{w}$, the Lagrangian relaxation takes the following form:

$$\begin{aligned}
[\mathbf{DS}] \quad \varphi(w) = \min \quad & \sum_{k\in\mathscr{C}} \sum_{(i,j)\in\mathscr{A}} \tilde{c}_{ij}^k x_{ij}^k \\
& + \sum_{(i,j)\in\mathscr{A}} f_{ij} y_{ij} + \sum_{k\in\mathscr{C}} \sum_{i\in\mathscr{N}} \bar{w}_i^k b_i^k \\
\text{s.t.} \quad & \sum_{k\in\mathscr{C}} x_{ij}^k \leqslant u_{ij} y_{ij} \quad \forall (i,j)\in\mathscr{A}, \\
& x_{ij}^k \leqslant d_{ij}^k y_{ij} \quad \forall (i,j)\in\mathscr{A},\ \forall k\in\mathscr{C}, \\
& x_{ij}^k \geqslant 0 \quad \forall (i,j)\in\mathscr{A},\ \forall k\in\mathscr{C}, \\
& y_{ij}\in\{0,1\} \quad \forall (i,j)\in\mathscr{A},
\end{aligned}$$

where, $\tilde{c}_{ij}^k = c_{ij}^k + \bar{w}_j^k - \bar{w}_i^k$, $\forall (i,j)\in\mathscr{A}$, $\forall k\in\mathscr{C}$. DS separates into $|\mathscr{A}|$ problems, one on each arc. For arc $(i,j)$ we need to solve the following subproblem:

$$\begin{aligned}
[\mathbf{DS_{ij}}] \quad g_{ij}(\bar{w}) = \min \quad & \sum_{k\in\mathscr{C}} \tilde{c}_{ij}^k x_{ij}^k + f_{ij} y_{ij} \\
\text{s.t.} \quad & \sum_{k\in\mathscr{C}} x_{ij}^k \leqslant u_{ij} y_{ij}, \\
& x_{ij}^k \leqslant d_{ij}^k y_{ij} \quad \forall k\in\mathscr{C}, \\
& x_{ij}^k \geqslant 0 \quad \forall k\in\mathscr{C}, \\
& y_{ij}\in\{0,1\}.
\end{aligned}$$

To solve $DS_{ij}$, we note that the value of $y_{ij}$ depends on the solution of flow variables. Moreover, the part that involves flow variables can be identified as a special case of the linear knapsack problem in which all variables have coefficient 1. This fact suggests that $DS_{ij}$ can easily be solved by using the greedy principle. The following algorithm describes such a solution procedure.

*Step 1.* Set $\hat{g}_{ij} = f_{ij}$, $x_{ij}^k = 0$, $\forall k \in \mathscr{C}$.

*Step 2.* Select $k' = \text{argmin}_{k \in \mathscr{C}} (\tilde{c}_{ij}^k)$.

*Step 3.* If $\tilde{c}_{ij}^{k'} > 0$, go to 6.

*Step 4.* If $u_{ij} > r^{k'}$, set $x_{ij}^{k'} = r^{k'}, u_{ij} = u_{ij} - r^{k'}, \hat{g}_{ij} = \hat{g}_{ij} + \tilde{c}_{ij}^{k'} x_{ij}^{k'}$ and $\tilde{c}_{ij}^{k'} = +\infty$.

Otherwise, set $x_{ij}^{k'} = u_{ij}$, $u_{ij} = 0$, $\hat{g}_{ij} = \hat{g}_{ij} + \tilde{c}_{ij}^{k'} x_{ij}^{k'}$.

*Step 5.* If $u_{ij} > 0$, go to 2.

*Step 6.* If $\hat{g}_{ij} \leqslant 0$, set $y_{ij} = 1$, otherwise set $y_{ij} = 0$ and $x_{ij}^k = 0$, $\forall k \in \mathscr{C}$.

If $\hat{g}_{ij} = 0$, $y_{ij}$ can be either 0 or 1. However, because our primal heuristic is based on the solution of the Lagrangian relaxation, it is preferable to set $y_{ij} = 1$ to enable feasible solutions.

It is easy to show that the DS has the "integrality property." That is, the $y$-solution will obtain integral values even if the integrality constraints are removed. This property comes from the fact that if any flow is sent on an arc in the optimal solution, constraint (2) and/or constraint (3) will become active on that arc, which causes the design variable to be one. Otherwise, the design variable will be set to zero. This property implies that the optimal value of LD is the same as of the LP-relaxation.

## 2.2. Strengthening the Lagrangian Relaxation

There are several valid constraints, redundant in CNDP, that can be added to DS to strengthen it. Because all coefficients are assumed to be nonnegative in CNDP, the flow of a commodity will never contain a cycle in the optimal solution. Therefore, we can require that $x_{ij}^k = 0$ if $o(k) = j$ or $d(k) = i \; \forall (i,j) \in \mathscr{A} \; \forall k \in \mathscr{C}$. It requires almost no extra computational effort to take these constraints into consideration while solving DS because they can be established once and for all.

It is also clear that $x_{ij}^k = 0$ if $d(k)$ is unreachable from $j$ or $i$ is unreachable from $o(k)$ $\forall (i,j) \in \mathscr{A}$ $\forall k \in \mathscr{C}$. A large amount of flow variables can probably be eliminated from DS by this, if the network is very sparse. Furthermore, such variables can be easily detected by using a modified shortest path algorithm.

A common feature of these valid equalities is that they in some sense recreate what has been relaxed, namely the network structure, but *without* destroying the separability of the subproblem. This is an important general principle, that might be useful in many other methods of this type. The difficult task clearly is to find constraints of this type, recreating relaxed structures without losing the separability (as separability was the goal of the relaxation).

Other valid constraints are the so-called Gale-Hoffman inequalities. They are based on cuts that divide the set of nodes into two sets, such that for some commodities the origin is in one set and the destination is in the other. The inequalities ensure that the total capacity of such a cut must be greater or equal to the total demand of these commodities. In the special case when one set consists of a single node and only the commodities that have this node as origin are considered, the constraints (one for each node) contain disjunctive sets of variables.

With these inequalities added to DS, it requires the solution of a number of 0/1-knapsack problems. To reduce the additional computational effort, we have tested to use them in an LP-relaxed fashion. However, we found that the improvement of the lower bound and convergence are insignificant compared to the extra effort required to solve the knapsack problems. Therefore, these constraints are not added to DS in the final implementation.

## 2.3. Subgradient Search

To solve the Lagrangian dual, we use the well-known technique of subgradient optimization; see for example Poljak (1967, 1969), Held et al. (1974), Goffin (1977), and Shor (1985). A subgradient of the dual function $\varphi(w)$ is used to compose a search direction to update the multipliers (dual variables). In our case, a subgradient to $\varphi(\bar{w})$ is given as

$$\xi_i^k = b_i^k - \sum_{j:(i,j) \in \mathscr{A}} \bar{x}_{ij}^k + \sum_{j:(j,i) \in \mathscr{A}} \bar{x}_{ji}^k \quad \forall i \in \mathscr{N}, \; \forall k \in \mathscr{C},$$

where $\bar{x}$ denotes the optimal solution of DS at $\bar{w}$. Letting $\bar{w}^{(l)}$ denote the multiplier values in iteration $l$, we obtain the multipliers in the next iteration by setting

$$\bar{w}^{(l+1)} = \bar{w}^{(l)} + t^{(l)} d^{(l)},$$

where $t^{(l)}$ and $d^{(l)}$ are the step size and the search direction in iteration $l$. The search direction is usually set equal to the subgradient (i.e., $d^{(l)} = \xi^{(l)}$), but it is often more efficient to use a modified formula, such as suggested in Crowder (1976) or Gaivoronsky (1988). So our search direction $d^{(l)}$ is defined recursively as below, where $d^{(1)} = \xi^{(1)}$:

$$d^{(l)} = (\xi^{(l)} + \theta d^{(l-1)})/(1 + \theta) \quad \forall l > 1.$$

The choice of $t^{(l)}$ has always been a critical part in subgradient optimization because the practical convergence rate relies heavily on it. Several ways of choosing the step size have been suggested in Shor (1968), Poljak (1969), Held and Karp (1971), and Held et al. (1974). Here, we adopt the one suggested by Poljak, which is a very widely used one:

$$t^{(l)} = \lambda_l \frac{\tilde{v} - \varphi(\bar{w}^{(l)})}{\|\xi^{(l)}\|^2}.$$

Here, $\tilde{v}$ is supposed to be an upper bound of $v_L$ and $\lambda_l$ should be assigned a value in the interval $(\varepsilon_1, 2 - \varepsilon_1)$, where $\varepsilon_1 > 0$, to ensure convergence.

Given a set of problem instances, the subgradient search procedure contains several parameters, which can be adjusted to obtain as fast practical convergence as possible. The value of $\theta$ determines how much consideration is

taken to the previous direction. Empirical results indicate that 0.7 is an appropriate value. Another parameter used in the step size formula is $\lambda_l$. It is easy to show that $\lambda_l = 1$ is theoretically optimal if $\tilde{v} = v_L$. In practice, the value of $v_L$ is not known in advance and $\tilde{v}$ is often an upper bound of the optimal primal value $v^*$. Thus there are no simple rules to determine and justify this parameter. In our implementation, we choose $\lambda_1 = 1.1$ as a start value. If no improvement of the lower bound is obtained in $K$ successive iterations, we set $\lambda_l = 0.5\lambda_{l-1}$ and reset $\lambda_l$ back to 1.1 whenever we get an improved upper bound.

The convergence depends also on the value of $\tilde{v}$ used. As mentioned above, $\tilde{v}$ is usually an upper bound to $v^*$ and is often obtained from a primal feasible solution. Initially, however, such a bound may not be very good, and there might also be a quite large gap between $v_L$ and $v^*$. Rather than relying on an approximate upper bound, we update $\tilde{v}$ adaptively according to both our upper and lower bounds. Discussions of such techniques can be found in Bazaraa and Sherali (1981), Allen et al. (1987), and in a primal-dual context in Sen and Sherali (1986). We use the simplified formula:

$$\tilde{v} = (\eta \bar{v} + \varphi(\bar{w}^{(l)})/2.$$

Here $\bar{v}$ is the best obtained upper bound so far. $\tilde{v}$ converges towards $v_L$ if $\bar{v} = v_L$ and $\eta = 1$. However, it is not desirable that LD converges toward the global $v_L$ while using it in a branch-and-bound framework. In a node in the branch-and-bound tree, we would like LD to converge toward the $v_L$ that is associated with that specific node, which might be larger than $v^*$ because a number of variables are fixed. On the other hand, it is enough that $\varphi(\bar{w}^{(l)})$ is larger than $\bar{v}$ to cut off a node. Therefore, we choose $\eta$ to be slightly greater than one. Experiments shows that $\eta = 1.05$ is a good choice.

Another issue in subgradient optimization is the starting solution. A straightforward way is to start with $\bar{w}^{(1)} = 0$, which will yield $\varphi(\bar{w}^{(1)}) = 0$. To get a better initial value of the search procedure, we find the shortest path tree in the network for each commodity and use its node prices (the dual solution of the shortest paths). These node prices are multiplied with a weight $\tau \in [0, 1]$ to yield $\bar{w}^{(1)}$, which can be seen as a convex combination of the origin and the obtained nodeprices. Considering the choice of $\tau$, we find that $\tau = 0$ yields a bad starting solution but with a good convergence behavior, while $\tau = 1$ yields a good starting solution but much worse convergence behavior. In the last case, the objective function value immediately drops significantly, and a large number of subgradient iterations is needed to regain the starting value. We belive that this depends on the dual function $\varphi(w)$; $\tau = 1$ yields a point where $\varphi(w)$ is certainly not differentiable. In our implementation, $\tau$ is set to be 0.9. This leads to a much better initial lower bound than zero in most of the cases, and a reasonable convergence behavior. If, however, such an initial solution yields a worse value (i.e. $\varphi(\bar{w}^{(1)}) < 0$), we reset $\bar{w}^{(1)}$ back to zero.

Irrespective of the convergence rate, we have to terminate the subgradient search procedure sooner or later. Ideally, the procedure terminates when $\xi^{(l)} = 0$, which indicates that the dual optimum is found and that the solution is primal feasible. Because the relaxed constraint set involves only equalities, this also implies that the primal solution is optimal. However, this is unlikely to occur in most of the cases because the Lagrangian dual is solved approximately in practice, and a duality gap may exist. In practice, we choose to stop the search procedure when $\|d^{(l)}\| < \varepsilon$, $t^{(l)} < \varepsilon$, $l > M$ or $\bar{v} - \varphi(\bar{w}^{(l)}) \leqslant \varepsilon$.

To show the theoretical convergence of the basic subgradient search procedure, we refer to Poljak (1969) for a convergence proof if there is no duality gap and $\tilde{v} = v_L$. If $v_L < \tilde{v}$, we can use a result by Allen et al. (1987) to prove the convergence toward $v_L$.

In our implementation, we have used some nonstandard modifications, discussed earlier, to improve the performance. We can, however, still ensure the theoretical convergence by removing these modifications and stop reducing $\lambda$ after a certain number of iterations. In practice, we most often do not have the time to let the subgradient procedure converge with good accuracy. Instead we use a fairly small value of $M$, i.e., stop after a quite limited number of iterations.

### 2.4. Obtaining Feasible Solutions

The last part of a Lagrangian heuristic involves generation of upper bounds, i.e., feasible solutions. Upper bounds are used in the subgradient optimization to calculate the step size, as well as in our branch-and-bound algorithm to cut branches and to get the final solution when the algorithm terminates.

To obtain a feasible solution of CNDP, we consider a given subset of arcs, $\bar{\mathcal{A}} \subseteq \mathcal{A}$, where $\bar{\mathcal{A}} = \{(i,j) \in \mathcal{A}: \bar{y}_{ij} = 1\}$. Whenever $\bar{\mathcal{A}}$ (and $\bar{y}$) is fixed, CNDP becomes the following primal subproblem:

$$[\mathbf{PS}] \quad v(\bar{\mathcal{A}}) = \min \sum_{k \in \mathcal{C}} \sum_{(i,j) \in \bar{\mathcal{A}}} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in \bar{\mathcal{A}}} f_{ij}$$

$$\text{s.t.} \quad \sum_{j:(i,j) \in \bar{\mathcal{A}}} x_{ij}^k - \sum_{j:(j,i) \in \bar{\mathcal{A}}} x_{ji}^k = b_i^k$$

$$\forall i \in \mathcal{N}, \ \forall k \in \mathcal{C},$$

$$\sum_{k \in \mathcal{C}} x_{ij}^k \leqslant u_{ij} \quad \forall (i,j) \in \bar{\mathcal{A}},$$

$$x_{ij}^k \geqslant 0 \quad \forall (i,j) \in \bar{\mathcal{A}}, \ \forall k \in \mathcal{C},$$

where

$$b_i^k = \begin{cases} r^k & \text{if } i = o(k), \\ -r^k & \text{if } i = d(k), \\ 0 & \text{otherwise.} \end{cases}$$

There is no need to include constraints corresponding to the linking constraints (3) in CNDP because these are

redundant and will not affect the methods described below to solve PS.

If the partial network that is represented by $\bar{\mathcal{A}}$ does not contain "enough" arcs, no feasible solution will be found. On the other hand, if $\bar{\mathcal{A}}$ contains "too many" arcs, some of the arcs will not be used to send any flow. In the latter case, such arcs are eliminated by setting the corresponding $y_{ij}$ equal to 0 when calculating the upper bound, after PS is solved.

PS can be identified as a multicommodity network flow problem, which is a highly structured linear programming problem. In our method, we have combined two different approaches for solving PS: a simple primal heuristic and a commercially available linear programming code including a network flow solver.

The heuristic works according to the greedy principle and is a multicommodity version of the well-known Busacker-Gowen method for single commodity network flow problems; Busacker and Gowen (1961), Jewell (1958), Iri (1960). There have been similar heuristics developed for multicommodity network flow problem; see, for example, Barnhart (1993) and Holmberg (1995).

The heuristic selects one commodity at a time and applies the Busacker-Gowen procedure. This involves essentially solving of a number of shortest path problems. After finding the shortest path between the origin and the destination and sending as much flow as possible, capacities and feasible directions are updated, creating a residual network. (To allow decreasing a positive flow, an arc in the opposite direction is added.) The procedure is repeated until all flow of the commodity is sent. The same procedure is then repeated for the next commodity. If, however, no path can be found to send the remaining flow during the procedure, the heuristic fails to find a feasible solution.

The solution of PS generated by the heuristic is dependent on the order in which the commodities are considered. We found that instead of sending commodities in a random manner, better results are obtained by utilizing the values of the Lagrangian multipliers obtained in the last subgradient iteration. Each commodity $k$ is labeled with the difference of the multiplier values associated with $o(k)$ and $d(k)$, multiplied with its demand. The commodities are then considered in a descending order.

The heuristic consists mainly of a number of shortest path problems and is therefore very fast. To solve the shortest path problems, we use the algorithm L-threshold described in Gallo and Pallottino (1988).

The heuristic can quickly generate upper bounds for CNDP for a given $\bar{\mathcal{A}}$. However, such a heuristic alone may not be sufficient to obtain good feasible solutions since it solves PS approximately. Moreover, if we wish to solve CNDP exactly by branch-and-bound, we need the optimal solution of PS, at least in the leaves of the branch-and-bound tree.

To obtain the optimal solution to PS, we use the standard code CPLEX with the "hybnetopt" option for two reasons:

1. Because of its built-in network optimizer, CPLEX can find and solve the network part of PS and treat the capacities as side constraints, taken into account by dual simplex iterations. This approach is much more efficient than a standard linear programming code for PS. (In fact, it is even competitive comparing to specialized multicommodity network flow codes.)

2. In our branch-and-bound algorithm, PS needs to be solved iteratively. Two successive PSs have almost the same structure, except for small changes in the constraints. Therefore, the whole algorithm is accelerated considerably by reoptimization. This can be easily achieved in CPLEX by starting from the previous optimal basis.

In some cases when PS is solved by CPLEX, we notice that the efficiency can be increased dramatically by flow aggregation, if the network has the property that the routing cost $c_{ij}^k$ does not vary by commodity; i.e., for any arc $(i,j)$ in the network, we have $c_{ij}^k = c_{ij}$, $\forall k \in \mathcal{C}$. Although this need not be true in the general mathematical model, it is the case in many telecommunication applications. In this case flow aggregation can be done in PS whenever a number of commodities share the same origin or the same destination. Flow aggregation is, for example, described in Jones et al. (1993) and is done by aggregating all commodities with the same origin into one commodity. It requires addition of a supersink and one additional arc for each aggregated commodity. Analogous aggregation can be performed in PS if a number of commodities share a common destination.

One may actually use such aggregations on the original CNDP without changing the optimal value. However, because relaxation is performed (LP- or Lagrangian relaxation), an aggregated version leads to weaker lower bounds and will thus slow down the overall convergence.

Finally, we might mention that a recent implementation of a column generation method for the multicommodity flow problem, Holmberg and Yuan (1998), appears to be faster than CPLEX and could be used to solve PS to improve overall performance.

## 3. THE BRANCH-AND-BOUND ALGORITHM

In this section, we embed the Lagrangian heuristic in a branch-and-bound framework. We describe two versions of the method: one that finds the exact optimum and verifies it; and one heuristic, which in a shorter time finds near-optimal solutions. As mentioned before, one should not expect to be able to solve very large network design problems exactly within a reasonable time. In the heuristic, the algorithm is mainly used to improve the feasible solutions. The way to achieve this is to fix variables based on the information from the Lagrangian heuristic. In the following subsections we describe both cases (the exact and the heuristic) with their relevant design issues.

### 3.1. The Basic Scheme

In the branch-and-bound procedure, we use branching on the design variables. Two branches are generated, one with $y_{ij} = 1$ (the arc is fixed open) and the other with $y_{ij} = 0$ (the

arc is fixed closed). The effect of the branching is simply that some design variables are fixed either to 1 or 0, while solving the subproblems DS and PS.

At each node in the tree, the previously described Lagrangian heuristic is applied to generate lower and upper bounds. The Lagrangian heuristic starts with generating a feasible solution that is associated with the node. This is done by solving a PS with $\bar{\mathcal{A}}$ including all arcs except those that are fixed closed. The primal heuristic is applied to obtain a feasible solution. If the heuristic fails in finding a feasible solution, however, CPLEX is invoked to confirm that no feasible solution exists at this node, or alternatively, to find such a solution. This strategy implies that the same PS may be solved twice at certain nodes. The motivation here is that because PS becomes hard to solve for large problems, to solve PS exactly at each node is not desirable. Rather, the heuristic tries to obtain a feasible solution first, and if the Lagrangian heuristic shows later that the node can be cut off from the tree, there is no need to solve PS exactly. However, if this is not the case and further branching is needed, we solve PS exactly by CPLEX *before* the algorithm proceeds to the next node. To summarize, PS is solved exactly either to confirm that no feasible solution exists or before the algorithm enters new branches.

It should be mentioned that this approach does not mean that we need to solve PS exactly at all nodes in the tree where the algorithm has branched. In fact, PS needs to be solved exactly only at half of these nodes. This is because whenever there is a branching on a certain arc, there is no need to solve the PS that is associated with the 1-branch because the same PS has already been solved exactly at the current node, i.e., the subset $\bar{\mathcal{A}}$ will be the same.

Normally, a feasible solution is obtained in this way and a number of iterations of subgradient search is performed. During this search procedure, the primal heuristic is used periodically to obtain alternate feasible solutions based on the information from the Lagrangian dual. To achieve this, we use the $y$-solution of DS to compose the subset $\bar{\mathcal{A}}$ in PS. Let $\mathcal{A}(\bar{w}^{(l)})$ denote the arcs that are open in the solution of DS in iteration $l$. We then let $\bar{\mathcal{A}} = \mathcal{A}(\bar{w}^{(l)}) \cup \mathcal{A}(\bar{w}^{(l-1)})$, i.e., the union of two successive solutions. Our tests show that the $y$-solution of one iteration often does not contain enough arcs to enable a feasible solution. On the other hand, if one takes the union of the $y$-solutions of too many iterations, the obtained primal solution is often not good and tends to be identical during several iterations of the subgradient search.

Because of the relaxation that is chosen, we can construct penalty tests, used during the subgradient search to fix variables and thus reduce the branch-and-bound tree.

After a certain number of iterations, the Lagrangian heuristic at a certain node will either detect that the node can be cut off or decide to branch as a result of duality gap or slow convergence. Before a branching is done, the heuristic version of the algorithm tries to fix certain variables. (This is done in a heuristic manner, in contrast to penalty tests.) This makes the algorithm concentrate only on the most promising parts of the tree and hence saves a large portion of the computational effort required, which is necessary if the algorithm is going to deal with large problems in real-life applications.

More details of the branch-and-bound procedure are described in §§3.2 to 3.6. Section 3.2 gives the cutting and stopping criteria, and §3.3 describes the penalty tests. The two heuristic versions of the solution method using variable fixing are introduced in §3.4. Section 3.5 discusses branching and searching strategies. Finally, we give an algorithm summary in §3.6.

### 3.2. Cutting and Stopping Criteria

In this section, we describe the criteria for cutting off nodes in the branch-and-bound tree and the stopping criteria that are used to terminate the Lagrangian heuristic and perform a branching.

The standard cutting criteria are the following:

- Cut if the optimum of the node is found (which can be verified only if the solution of DS is feasible, i.e., if $\xi^{(l)} = 0$; but this rarely occurs for large problems).
- Cut if no feasible solution exists. (Before starting the Lagrangian heuristic at each node, the algorithm attempts to obtain a feasible solution.)
- Cut if the lower bound from the Lagrangian dual exceeds the best known upper bound, i.e., if $\varphi(\bar{w}^{(l)}) \geq \bar{v}$, where $\bar{v}$ is the best known upper bound.

Because of factors such as the existence of a duality gap, the convergence rate, the quality of the generated primal solutions, etc., it is possible that none of these criteria ever becomes satisfied during the subgradient search procedure.

In a problem that involves both routing costs and fixed charges, another criterion is available to check if a node in the branch-and-bound tree is an interesting one or not by utilizing information from primal solutions. The criterion is based on the relation between the routing costs and the fixed charges. Let us denote by $c(\mathcal{A})$ the optimal routing cost if all arcs in $\mathcal{A}$ are allowed to be used for routing and by PS($\mathcal{A}$) the corresponding primal subproblem. Hence given a subset of arcs $\bar{\mathcal{A}} \in \mathcal{A}$, $c(\bar{\mathcal{A}}) + \Sigma_{(i,j)\in\bar{\mathcal{A}}_+} f_{ij}$ is equal to the optimal value of PS($\bar{\mathcal{A}}$), where $\bar{\mathcal{A}}_+ = \{(i,j) \in \bar{\mathcal{A}} : y_{ij} > 0\}$.

LEMMA 1. $c(\mathcal{A}) \leq c(\bar{\mathcal{A}})$, $\forall \bar{\mathcal{A}} \subseteq \mathcal{A}$.

PROOF. Because $\bar{\mathcal{A}} \subseteq \mathcal{A}$, the optimal solution to PS($\bar{\mathcal{A}}$) is also feasible in PS($\mathcal{A}$). □

The lemma simply states that the optimal routing cost is lowest when all arcs are allowed to be used. Assume that $\hat{\mathcal{A}}$ is the set of all arcs used in the optimal primal solution. Because $\hat{\mathcal{A}} \subseteq \mathcal{A}$, it follows from this result that the routing cost obtained at the top node (root) is a lower bound of the routing cost in the optimal solution.

The result is also valid for any other node in the tree. For an arbitrary node, let $\mathcal{A}_1$ be the set of arcs that are fixed open, $\mathcal{A}_0$ be the set of arcs that are fixed closed and $\mathcal{A}_*$ be $\mathcal{A} \setminus (\mathcal{A}_0 \cup \mathcal{A}_1)$. The PS associated with the node is then

PS($\mathcal{A}_1 \cup \mathcal{A}_*$). Assume that another node $(\mathcal{A}_0', \mathcal{A}_1', \mathcal{A}_*')$ is obtained by further branching, i.e., $(\mathcal{A}_0', \mathcal{A}_1', \mathcal{A}_*')$ is a node in the subtree with $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_*)$ as the root.

LEMMA 2. $c(\mathcal{A}_1 \cup \mathcal{A}_*) \leqslant c(\mathcal{A}_1' \cup \mathcal{A}_*')$.

PROOF. Because $(\mathcal{A}_0', \mathcal{A}_1', \mathcal{A}_*')$ is obtained by further branching, we know that $\mathcal{A}_0 \subseteq \mathcal{A}_0'$. This is equivalent to $(\mathcal{A}_1' \cup \mathcal{A}_*') \subseteq (\mathcal{A}_1 \cup \mathcal{A}_*)$. The conclusion follows by Lemma 1. □

Lemma 2 states that the routing cost will not decrease if the algorithm makes a branching because a branching can only lead to fewer open (fixed open and not fixed) arcs in PS.

Based on these observations, we have Lemma 3.

LEMMA 3. *If a node* $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_*)$ *satisfies* $c(\mathcal{A}_1 \cup \mathcal{A}_*) + \Sigma_{(i,j)\in\mathcal{A}_1} f_{ij} \geqslant \bar{v}$, *then the node cannot lead to a feasible solution with objective function value strictly less than* $\bar{v}$.

PROOF. It follows from Lemma 2 that the routing cost obtained at the current node $c(\mathcal{A}_1 \cup \mathcal{A}_*)$ is a lower bound of the routing costs in the subtree with $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_*)$ as its root. The term $\Sigma_{(i,j)\in\mathcal{A}_1} f_{ij}$ is the sum of fixed charges of arcs that are fixed open at the current node. Obviously, this term is also nondecreasing while getting deeper in the subtree. Hence the conclusion. □

In other words, if the sum of the obtained routing cost and the fixed charges of those arcs that are fixed open exceeds the best known upper bound, then the current node is an uninteresting one.

The following cutting criterion is derived directly from Lemma 3.

- A node $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_*)$ can be cut off if $c(\mathcal{A}_1 \cup \mathcal{A}_*) + \Sigma_{(i,j)\in\mathcal{A}_1} f_{ij} \geqslant \bar{v}$.

By utilizing the information of the primal solution at each node, another cutting criterion can be obtained. The primal subproblem PS($\mathcal{A}_* \cup \mathcal{A}_1$) is solved exactly either when the Lagrangian heuristic starts or before a branching is made. Let $\mathcal{A}^0 \in (\mathcal{A}_* \cup \mathcal{A}_1)$ denote the set of arcs that are included but not used in this solution. We have the following result.

LEMMA 4. *If* $\mathcal{A}_* \subseteq \mathcal{A}^0$, *the current node cannot lead to better primal solutions by further branching.*

PROOF. As described in §3.1, the value of PS can be improved only in a 0-branch. Therefore, it is sufficient to consider optimal solutions to all PS in which one or several arcs in $\mathcal{A}_*$ are fixed to 0. In PS($\mathcal{A}_*' \cup \mathcal{A}_1$), where $\mathcal{A}_*' \subseteq \mathcal{A}_*$, some arcs are fixed closed, which is equivalent to fixing all flow variables on these arcs to zero. However, these variables are zero in the optimal solution of PS($\mathcal{A}_* \cup \mathcal{A}_1$). Because PS is a linear programming problem, the optimal solution to PS($\mathcal{A}_* \cup \mathcal{A}_1$) is still optimal to PS($\mathcal{A}_*' \cup \mathcal{A}_1$), hence the conclusion. □

The lemma says that no further branching is needed if none of the unfixed arcs is used in the optimal primal solution. Based on this result, the following cutting criterion is deduced.

- A node $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_*)$ can be cut off if $\mathcal{A}_* \subseteq \mathcal{A}^0$.

It is now quite obvious from Lemmas 1 to 4 that all the cutting criteria used are valid.

If none of the above criteria is satisfied, we have to terminate the Lagrangian heuristic at some point and branch. The stopping criteria for the Lagrangian heuristic are used to decide if a branching should be performed; see §2.3.

The choices of $\varepsilon$ and $M$ in the stopping criteria have a significant impact on the overall efficiency. If $\varepsilon$ is too small or $M$ too large, the algorithm will make too many iterations without significant improvement of the lower bound before a branching is made. On the other hand, a too large $\varepsilon$ or a too small $M$ introduces the risk that the algorithm will perform branching before good bounds are obtained.

### 3.3. Penalty Tests

We can use penalty tests to identify the correct values of some variables and thus reduce the size of the branch-and-bound tree. In a branch-and-bound algorithm based on a Lagrangian heuristic, it is not trivial to construct penalty tests because it depends both on the problem structure and the constraint set that is relaxed. Generally, we notice that if the relaxation is done in such a way that the dual subproblem is decomposed into a number of separate components and branching is made with respect to these components, then it becomes easier to construct penalty tests. Here we find a distinct advantage with our relaxation compared to the relaxation chosen in Gendron and Crainic (1994).

In our case, the DS is separable into one problem for each arc, while the branching also concerns single arcs. The optimal value of DS at $\bar{w}$ is

$$\varphi(\bar{w}) = \sum_{(i,j)\in\mathcal{A}} \hat{g}_{ij}\bar{y}_{ij} + \sum_{k\in\mathcal{C}}\sum_{i\in\mathcal{N}} \bar{w}_i^k b_i^k,$$

where $\hat{g}_{ij}$ is identified as the reduced cost on arc $(i, j)$, calculated when solving DS with the procedure described in §2.1. We would like to know the effect of fixing a certain design variable $y_{ij}$ to one or zero in the Lagrangian dual without solving it, which is not simple. However, we do know the effect of fixing a certain $y_{ij}$ in the Lagrangian objective function on condition that $\bar{w}$ remains unchanged, as shown below.

- $\hat{g}_{ij} > 0$: $\varphi(\bar{w})_{y_{ij}=1} = \varphi(\bar{w}) + \hat{g}_{ij}$, $\varphi(\bar{w})_{y_{ij}=0} = \varphi(\bar{w})$.
- $\hat{g}_{ij} < 0$: $\varphi(\bar{w})_{y_{ij}=1} = \varphi(\bar{w})$, $\varphi(\bar{w})_{y_{ij}=0} = \varphi(\bar{w}) + |\hat{g}_{ij}|$.

Note that solving DS at $\bar{w}$ always yields a lower bound in any branch, regardless of $\bar{w}$. Thus we can evaluate the Lagrangian dual at $\bar{w}$ in both of the two new branches $y_{ij} = 1$ and $y_{ij} = 0$ without actually branching on the arc $(i, j)$. The value of $|\hat{g}_{ij}|$ gives a lower bound of the increase in one branch. Because such an evaluation can be performed by one single addition, we can easily estimate the effect of branching on a certain arc, without changing the multipliers.

Therefore, the following penalty test is used each time DS is solved, in an attempt to detect uninteresting branches.

For each arc $(i, j)$:

1. If $\hat{g}_{ij} > 0$ and $\varphi(\bar{w}) + \hat{g}_{ij} \geq \bar{v}$, fix $y_{ij}$ to 0.
2. If $\hat{g}_{ij} < 0$ and $\varphi(\bar{w}) + |\hat{g}_{ij}| \geq \bar{v}$, fix $y_{ij}$ to 1.

In the first case, we detect that the Lagrangian dual value will exceed the best known upper bound if $y_{ij}$ is set to one. According to our cutting criteria in §3.2, the 1-branch can be cut off if we branch on this arc. The variable $y_{ij}$ can consequently be fixed to zero. Similarly, $y_{ij}$ is fixed to one in the second case.

At each node in the tree, we wish to determine values for as many design variables as possible by penalty tests. Because the maximal number of allowed subgradient iterations is different at different nodes (see §3.5), we increase the maximal number of iterations for the current node if a variable is fixed by the penalty test. Because different limits exist in the tree, we denote the maximal limit $M1$ and set $M = \min(M1, M + 1)$ in an iteration in which the penalty tests succeeds in fixing variables.

We notice that it is possible to apply these dual-based penalty tests by the following two facts:

1. The relaxation that is adopted enables the subproblem to be decomposed on each arc, and the dual objective function value is determined by these components.
2. The branching in the branch-and-bound algorithm is done on design variables, which coincides with the decomposition effect in the subproblem.

It should be pointed out that the penalty tests above are not limited just to CNDP. As long as these two conditions hold, the same tests can be carried out. This means that we can apply the same penalty tests in the uncapacitated case as an improvement to the algorithm in Holmberg and Hellstrand (1998) or to similar problems—for example, for networks with both partial capacities for commodities and a mutual capacity on each arc—as long as the same relaxation is adopted.

If the Lagrangian relaxation of constraint sets (2) and (3) is used, the dual subproblem does not decompose on arcs. The solution of the subproblem contains flows between origins and destinations, which is not the components on which branching is made. The solution to the dual subproblem can be totally different when a certain arc is fixed, even if the same multipliers are used, so the penalty tests would be much more computationally expensive.

### 3.4. Heuristic Fixing of Variables

For very large network design problems, we cannot expect an exact branch-and-bound method to terminate within a reasonable time limit. In this section, we describe two heuristic approaches developed to speed up the branch-and-bound procedure. The idea is to guide the branch-and-bound scheme to investigate only those nodes that seem to be most promising. This is achieved by fixing a certain number of design variables before a branching is made, using the information that is generated by the Lagrangian dual. Obviously this turns the method into a heuristic.

In the first approach, which is denoted by "$\alpha$-fixing", we consider the $y$-solution of the relaxation. During the subgradient search procedure, if a certain $y_{ij}$ is constantly set to one in DS, it indicates that arc $(i, j)$ is most likely to be included in the optimal solution. Similarly, arc $(i, j)$ is probably not included if the relaxation repeatedly suggests that the value of $y_{ij}$ is zero. Fixation rules can thus be designed based on the frequency that a certain $y_{ij}$ is set to one or zero. Ideally, $y_{ij}$ should be fixed only if it attains the same value in all subgradient iterations. However, to make the approach more flexible, we introduce a parameter $\alpha \in [0, 0.5]$ to allow deviations. If $\alpha = 0$, we have the situation above. The value $(1 - \alpha) * M$ is used as a threshold to determine whether a design variable is to be fixed after $M$ subgradient iterations. Below we summarize this fixation rule (the value of $y_{ij}$ in the $l$th solution of DS is denoted by $y_{ij}^{(l)}$):

- $y_{ij}$ is fixed to one if $\Sigma_{l=1}^{M} y_{ij}^{(l)} \geq (1 - \alpha) * M$.
- $y_{ij}$ is fixed to zero if $\Sigma_{l=1}^{M} y_{ij}^{(l)} \leq \alpha * M$.

By fixing variables in this way, only promising nodes are generated in the branch-and-bound tree. In a real application, it is almost always necessary to make a trade-off between the quality of the final solution generated and the computational effort required. This can be done by starting with a relatively large value of $\alpha$ to quickly obtain a solution and a lower bound, which will give an indication of how good the solution is. A good trade-off can be then reached by successively reducing $\alpha$.

It is worth noticing that even if $\alpha = 0$, the method is still a heuristic because $y_{ij}$ might be fixed to a value that is not optimal, although the probability is small. To obtain an exact solution method, this step has to be removed from the algorithm. Moreover, it is obvious that there is no explicit guarantee on the speed-up factor by fixing variables in this way. Also, the algorithm is still exponential, independently of $\alpha$. This drawback can, however, be avoided by using the alternative approach below to fix variables.

In the second approach, denoted by "$\beta$-fixing," we concentrate on the reduced costs $\hat{g}_{ij}$ obtained in DS. Notice the $\hat{g}_{ij}$ is highly related to the solution itself because the value of $y_{ij}$ is determined by the sign of $\hat{g}_{ij}$. Thus $\hat{g}_{ij} \gg 0$ or $\hat{g}_{ij} \ll 0$ indicates that $y_{ij}$ is zero and one, respectively, in the optimal solution, which suggests how to fix variables based on the value of $|\hat{g}_{ij}|$. To achieve this, we introduce another parameter $\beta \in [0, 1]$, which simply specifies the portion of the design variables to be fixed at each node in the branch-and-bound tree. Letting $n$ denote the initial number of unfixed arcs (which is often equal to $|\mathcal{A}|$), the number of arcs that will be fixed is exactly $\lceil \beta n \rceil$, or $|\mathcal{A}_*|$ if $|\mathcal{A}_*| < \lceil \beta n \rceil$. To determine the design variables to be fixed, we select those $y_{ij}$ with largest values of $|\hat{g}_{ij}|$, in a descending order. In our algorithm, we use the accumulated reduced cost from several iterations rather than one iteration in the following fashion. We let $R_{ij} = \hat{g}_{ij}^{(1)}$, and during the subgradient procedure, if a better lower bound is obtained in iteration $l$, we accumulate the corresponding reduced costs by setting $R_{ij} = \gamma * R_{ij} + \hat{g}_{ij}^{(l)}$ with $\gamma \in [0, 1]$. Because we consider the

reduced cost associated with an improved lower bound to be more reliable, the old value of $R_{ij}$ should have less weight. In our implementation, we set $\gamma = 0.5$. When the Lagrangian heuristic terminates, we fix a number of variables in the following manner:

*Step 1*. Set $m = 1$.

*Step 2*. Select an arc $(i', j') = \text{argmax}_{(i,j)\in\mathcal{A}_*}|R_{ij}|$.

*Step 3*. If $R_{i'j'} < 0$, fix $y_{i'j'}$ to 1, otherwise fix $y_{i'j'}$ to 0. Set $\mathcal{A}_* = \mathcal{A}_* \backslash (i', j')$ and $m = m + 1$.

*Step 4*. If $m \geqslant \min(|\mathcal{A}_*|, \lceil \beta n \rceil)$, stop. Otherwise go to 2.

The described two approaches are similar in the sense that both utilize the information generated by the relaxation. The difference is that using $\beta$-fixing, the maximal size of the branch-and-bound tree is known if $\beta$ is fixed. For example, $\beta = 0.1$ means that 10 percent of all design variables are fixed at each level in the tree. Consequently, the maximum depth of the tree is 10, and at most $2^{11}-2$ nodes (excluding the root) will be generated by the algorithm. Therefore, it is more convenient to control the time needed by the algorithm by varying the value of $\beta$ than $\alpha$. Moreover, the algorithm turns into an exact solution method for $\beta = 0$.

According to our computational tests, we can obtain a significant speed-up of the algorithm and simultaneously keep the final solution fairly close to optimum by using either one of the heuristic techniques $\alpha$-fixing or $\beta$-fixing.

### 3.5. Branch and Search Strategies

Even if we know when to branch, there are still several questions left to be answered. First of all, we have to decide on which variable the branching should be made. Recall that in §3.2 we show that no further branching is needed if all unfixed design variables are zero in the primal optimal solution of the current node. While this might happen when the algorithm gets deeper in the tree, it is rarely useful at the top. Nevertheless, the result indicates that when we choose the variable to branch on, we should avoid a variable $y_{ij}$ that is zero in the obtained primal solution. If we branch on an arc that is not used in the primal solution, none of the two branches that are generated will lead to an improved primal solution.

Still, there might be a large amount of candidate arcs. One choice is to fix a $y_{ij}$ that seems to be uncertain in the solution of our relaxation. Recall that the value of a design variable $y_{ij}$ is determined by its reduced cost $\hat{g}_{ij}$ in DS. If $|\hat{g}_{ij}|$ is large, the value of it is quite certain. Therefore, to branch on an arc that seems to be uncertain, we should choose an arc which has a small value of $|\hat{g}_{ij}|$, for example the arc with minimal $|\hat{g}_{ij}|$. (This approach would not be possible using the relaxation proposed in Gendron and Crainic 1994.)

Such a strategy is also very natural in the heuristic version of the method because before a branching is done, the algorithm attempts to fix some design variables. These variables will have the largest values of $|\hat{g}_{ij}|$ in DS; see §3.4. The remaining variables are considered as uncertain by the algorithm, which suggests that we should choose the most uncertain one (i.e., the arc with least $|\hat{g}_{ij}|$) to branch on.

However, this may not be suitable if the algorithm is used as an exact solution method. Our experiments show that better results are achieved by using another strategy, namely to choose the arc that has the maximal $|\hat{g}_{ij}|$ in the latest solution of DS and then start with the most promising branch; see below. The reason for this is quite different from the reasoning above and lies in the continued search for better solutions, while getting deeper in the tree. We now develop this argument.

After having chosen an arc, we still need to choose which of the two branches to search first. Assume that branching is made on arc $(i, j)$ and let $\tilde{y}_{ij}$ denote its value in the latest solution of DS. The following basic alternatives are possible:

- **[S1]** Start with $y_{ij} = \tilde{y}_{ij}$.
- **[S2]** Start with $y_{ij} \neq \tilde{y}_{ij}$.
- **[S3]** Start with $y_{ij} = 0$.
- **[S4]** Start with $y_{ij} = 1$.

While S1 and S2 are related to information of the relaxation, no such consideration is taken in S3 or S4. We know that the Lagrangian dual function $\varphi(w)$ is concave and piecewise linear. Furthermore, different linear supports at and around the point $\bar{w}$ can be identified by its solution in DS. In particular, there are two linear supports corresponding to $y_{ij} = 0$ and $y_{ij} = 1$, respectively. The value of $y_{ij}$ simply indicates which linear support is currently active. We can either keep the active cut by adopting S1 or use S2 to remove it. In the first case, we will continue the subgradient search on the same linear segment while in the second case we obtain an immediate improvement of the dual value at point $w = \bar{w}$ (if $\hat{g}_{ij} \neq 0$).

Our computational results indicate that S1 performs more efficiently than the other strategies listed above, so we begin our search with the branch in which $y_{ij}$ is fixed to the same value as $\tilde{y}_{ij}$.

Note that in the subproblem solution, $y_{ij} = 1$ if $\hat{g}_{ij} \leqslant 0$, otherwise $y_{ij} = 0$. In case that the algorithm is used as an exact method, we branch on an arc with the largest $|\hat{g}_{ij}|$; so if $\hat{g}_{ij}$ is negative, it is probably much less than zero, and $y_{ij}$ is likely to be one even in the solution of PS. We have parallel results in case $\hat{g}_{ij} > 0$. Therefore, if the algorithm follows this indication, the chance to obtain a good feasible solution is increased. That means that if $\hat{g}_{ij} \leqslant 0$ we start with the 1-branch, otherwise we start with the 0-branch. Obviously, this is the same as strategy S1.

Another issue in the search strategy concerns in which order the generated tree is to be searched. In our method, a depth-first strategy is chosen. One motivation here is that while getting deeper in the tree, it is very natural to start with the latest obtained multipliers, enabling a continuous improvement of the Lagrangian dual. This is especially suitable when coupled to the branching and searching strategies of the exact method because the first branch very much coincides with what would have been achieved without branching. Of course, if a backtracking is performed, the latest multipliers become a heuristic starting solution. However, this starting solution is found to perform better than starting from the same dual solution after each backtracking.

When a Lagrangian heuristic is combined with a branch-and-bound algorithm, it is quite common to associate the maximal number of subgradient iterations with the status of the current node. At the root, one should try to make the subgradient optimization method converge as much as possible by performing a relatively large number of iterations, denoted by $M1$. Because the depth-first search strategy is used, the subgradient search procedure will continue to deal with almost the same network after a branching, it is thus preferable to perform $M2$ iterations where $M2 < M1$. However, this might not be the case when backtracking occurs. Therefore, an intermediate number $M3$ is then used.

### 3.6. Algorithm Summary

In this section, the proposed method is summarized in an algorithmic fashion. Beside the notation used to describe the method in previous sections, we let $\mathcal{T}$ be the set of uninvestigated nodes in the branch-and-bound tree and $\hat{v}$ the objective value each time PS is solved.

*Step 1. Initialize B&B*: Set $\mathcal{A}_0 = \mathcal{A}_1 = \emptyset$, $\mathcal{A}_* = \mathcal{A}$, $\mathcal{T} = \{(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_*)\}$, $M = M1$ and $\bar{v} = \infty$.

*Step 2. Initialize a node*:

(a) Select a node from $\mathcal{T}$ according to the depth-first strategy. Update $\mathcal{A}_0, \mathcal{A}_1$ and $\mathcal{A}_*$. Get a starting dual solution $\bar{w}^{(l)}$ for the current node.

(b) If all $y_{ij}$ are fixed, i.e., $\mathcal{A}_* = \emptyset$, solve PS($\mathcal{A}_1$) exactly. If a feasible solution is obtained and $v(\mathcal{A}_1) < \bar{v}$, set $\bar{v} = v(\mathcal{A}_1)$. Go to 9.

(c) If $\mathcal{A}_* \neq \emptyset$, solve PS($\mathcal{A}_1 \cup \mathcal{A}_*$) by the heuristic. If no feasible solution is obtained, solve PS($\mathcal{A}_1 \cup \mathcal{A}_*$) exactly. If there is still no feasible solution, go to 9. Otherwise, if $v(\mathcal{A}_1 \cup \mathcal{A}_*) < \bar{v}$, set $\bar{v} = v(\mathcal{A}_1 \cup \mathcal{A}_*)$.

(d) If PS($\mathcal{A}_1 \cup \mathcal{A}_*$) is solved exactly: If $c(\mathcal{A}_1 \cup \mathcal{A}_*) + \Sigma_{(i,j)\in\mathcal{A}_1} f_{ij} \geq \bar{v}$, go to 9.

(e) Set $\lambda_1 = \lambda_0$, $l = 1$ and $\underline{v}$ to be the best lower bound obtained at the parent node.

*Step 3. Dual subproblem*: Solve DS with $\bar{w}^{(l)}$ and $\mathcal{A}_0, \mathcal{A}_1$. If $\varphi(\bar{w}^{(l)}) > \underline{v}$, set $\underline{v} = \varphi(\bar{w}^{(l)})$ and $\lambda_{l+1} = \lambda_l$. If $\underline{v} \geq \bar{v}$, go to 9. Otherwise, if $\underline{v}$ has not been improved after $K$ successive iterations, set $\lambda_{l+1} = \lambda_l/2$. Compute a subgradient $\xi^{(l)}$.

*Step 4. Penalty test*:

(a) For any $(i,j) \in \mathcal{A}_*$, if $\hat{g}_{ij} > 0$ and $\varphi(\bar{w})_{y_{ij}=1} = \varphi(\bar{w}) + \hat{g}_{ij} \geq \bar{v}$, fix $y_{ij}$ to zero and set $M = \min(M1, M+1)$.

(b) For any $(i,j) \in \mathcal{A}_*$, if $\hat{g}_{ij} < 0$ and $\varphi(\bar{w})_{y_{ij}=0} = \varphi(\bar{w}) + |\hat{g}_{ij}| \geq \bar{v}$, fix $y_{ij}$ to one and set $M = \min(M1, M+1)$.

(c) Update sets $\mathcal{A}_0, \mathcal{A}_1$ and $\mathcal{A}_*$ if any $y_{ij}$ is fixed. If all $y_{ij}$ are fixed, solve PS($\mathcal{A}_1$) exactly, set $\bar{v} = v(\mathcal{A}_\infty)\hat{v}$ if $v(\mathcal{A}_\infty)\hat{v} < \bar{v}$ and go to 9.

*Step 5. Primal subproblem*: If $(l \bmod L) = 0$, let $\bar{\mathcal{A}} = \mathcal{A}(\bar{w}^{(l)}) \cup \mathcal{A}(\bar{w}^{(l-1)})$. Solve PS($\bar{\mathcal{A}}$) with the heuristic. If $v(\bar{\mathcal{A}}) < \bar{v}$, set $\bar{v} = v(\bar{\mathcal{A}})$ and $\lambda_{l+1} = \lambda_0$. If $\underline{v} \geq \bar{v}$, go to 9.

*Step 6. Optimality check*: If $\|\xi^{(l)}\| = 0$, go to 9.

*Step 7. Subgradient search*: Calculate the search direction $d^{(l)} = (\xi^{(l)} + \theta d^{(l-1)})/(1+\theta)$ and a step size $t^{(l)} = \lambda_l(\tilde{v} - \varphi(\bar{w}^{(l)}))/\|\xi^{(l)}\|^2$, where $\tilde{v} = (\eta\bar{v} + \varphi(\bar{w}^{(l)}))/2$. Calculate the new multipliers $\bar{w}^{(l+1)} = \bar{w}^{(l)} + t^{(l)}d^{(l)}$.

*Step 8. Termination check*: If $\|d^{(l)}\| \leq \varepsilon$, $t^{(l)} \leq \varepsilon$ or $l \geq M$, go to 10. Otherwise, set $l = l + 1$ and go to 3.

*Step 9. Cut*: Set $\mathcal{T} = \mathcal{T} \setminus \{\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_*)\}$. If $\mathcal{T} = \emptyset$, stop. Otherwise, perform a backtracking. Set $M = M3$ and go to 2.

*Step 10. Branching preparation*:

(a) Solve PS($\mathcal{A}_1 \cup \mathcal{A}_*$) exactly if it was not done in step 2. If $v(\mathcal{A}_1 \cup \mathcal{A}_*) < \bar{v}$, set $\bar{v} = v(\mathcal{A}_1 \cup \mathcal{A}_*)$.

(b) If $\mathcal{A}_* = \emptyset$ or no arc in $\mathcal{A}_*$ is used in the exact solution of PS, go to 9.

(c) If $c(\mathcal{A}_1 \cup \mathcal{A}_*) + \Sigma_{(i,j)\in\mathcal{A}_1} f_{ij} \geq \bar{v}$, go to 9.

(d) Otherwise, select a variable $y_{ij}$, $(i,j) \in \mathcal{A}_*$, for branching.

*Step 11. Variable fixing*:

(a) Among the remaining unfixed variables, use $\alpha$-fixing or $\beta$-fixing to fix variables. Update sets $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_*$ and $M$ if any variable is fixed.

(b) If all $y_{ij}$ are fixed, solve PS($\mathcal{A}_1$) exactly. If $v(\mathcal{A}_1) < \bar{v}$, set $\bar{v} = v(\mathcal{A}_1)$ and go to 9.

*Step 12. Branching*: Generate two new branches and put them into $\mathcal{T}$ according to the branch and search strategy chosen. Set $M = M2$, go to 2.

The values of the parameters in our implementation are $M1 = 100, M2 = 5, M3 = 10, K = 4, L = 2, \lambda_0 = 1.1, \theta = 0.7$, $\eta = 1.05$, and $\varepsilon = 0.01$.

Because $M$ is finite, we have the following result for the above method, denoted by LHBB (Lagrangian heuristic with branch-and bound).

THEOREM 1. *LHBB will generate at least one feasible solution to CNDP (if there is any) and will terminate in a finite number of steps.*

PROOF. The first PS that is solved in the algorithm contains all arcs in the network. This primal subproblem is solved by the heuristic and, if necessary, by the exact method. Therefore, if there is a feasible solution to the problem, at least one of them will be found. If not, the algorithm is terminated. Secondly, in the worst case the algorithm will branch on all $y$-variables. Therefore, the number of nodes generated is finite. Furthermore, at most $M$ subgradient iterations will be carried out at each node. Hence the whole algorithm is finite. □

The above algorithm is designed to generate good solutions to large scale problems with a computational time that is acceptable for practical applications. It is not an exact solution method because nonoptimal values may be assigned to some design variables in step 11. However, if we wish, we can make the algorithm an exact solution method by simply dropping this step from the algorithm or setting $\beta = 0$, which is stated by Theorem 2. We call the exact method LHBB0.

THEOREM 2. *If step 11 is removed from the above algorithm, it will correctly solve CNDP in a finite number of steps.*

PROOF. As shown in §3.2, all cutting criteria are valid. Therefore the algorithm will not terminate before it verifies that the optimum has been found. To show that the optimal solution will be found, assume that all $y$-variables are fixed to their optimal values. Because all $y$-variables are fixed, PS will be solved exactly, yielding the optimal $x$-solution and $\hat{v} = v^*$. That the exact algorithm is finite follows from the same argument as in the proof of Theorem 1. □

Thus the algorithm can be both a fast heuristic and an exact method, without major modifications. Furthermore, if $\beta$-fixing is used, and if PS is solved with a polynomial method (which rules out the primal heuristic discussed in §2.4, since it is only pseudo-polynomial), we have the following result.

THEOREM 3. *The algorithm has a polynomial complexity if $\beta$-fixing (with $\beta > 0$) is used, and if PS is solved with a polynomial method.*

PROOF. Obviously, all other parts in the Lagrangian heuristic at each node have a polynomial complexity. These include the method used to solve DS and to make a subgradient step. Moreover, the penalty tests as well as the variable fixing procedure also have a polynomial complexity. Therefore, the algorithm complexity is polynomial at each branch-and-bound node, if a polynomial method is used whenever PS is to be solved (which is possible because PS is a linear programming problem). All these operations will not be performed more times than the maximal number of subgradient iterations. Furthermore, because at each level, $\beta$ percent of all arcs are fixed (at least one design variable will be fixed at each level since $\beta > 0$), there will be at most $\lceil 1/\beta \rceil$ levels in the tree. Thus the Lagrangian heuristic will be carried out at most $2^{\lceil 1/\beta \rceil + 1} - 1$ nodes, which is independent of the problem size. Hence the conclusion. □

In our implementation the algorithm is not theoretically polynomial because PS is solved exactly by a simplex method with a network optimizer. This is, however, not a drawback in practice because the way PS is solved is found to be very efficient. Indeed, because the size of the tree is determined by $\beta$, $\beta$-fixing provides a convenient way to adapt the practical difficulty to the size of the problem.

## 4. SOLVING CNDP WITH CPLEX

We have also, as a comparison, used CPLEX to solve CNDP by using the LP-relaxation of the problem together with branch-and-bound. Even if CPLEX with default settings of all parameters is a quite efficient way of solving mixed integer linear problems, there is still much that can be done to improve the performance. The key ingredient is to have a proper formulation of the problem. In CNDP, it matters whether the disaggregated linking constraints (3) are used or not and whether $y_{ij}$ is included in the capacity constraints (2) or not. The formulation presented in §1 is referred to as the *strong formulation*, and we obtain the weak formulation if the constraint set (3) is removed. While for small problems it seems quite obvious that one should use the strong formulation, the LP-relaxation becomes hard to solve when the problem size is getting large. For a problem that contains 100 arcs and 100 commodities, the strong formulation will have 10,000 more linking constraints. However, our experiments show that although it takes considerably longer time to solve the LP-relaxation, the strong formulation is still computationally preferred. The test results indicate that the size of the branch-and-bound tree can be reduced by a factor of at least two by using the strong formulation and that this factor increases fast with increasing problem size. Even for rather small problems, we obtain a speed-up factor of about 5 in the execution time. Note that it is crucial to have $d_{ij}^k = \min(r^k, u_{ij})$ because the same improvements will not be obtained using a larger value of $d_{ij}^k$. We also notice that the lower bound can, in many cases, be improved by including $y_{ij}$ in the capacity constraints.

A further consideration is the method used to solve the LP-relaxation. Several approaches are available in CPLEX. Because CNDP contains a network structure, we use the built-in hybrid network optimizer in CPLEX, which first solves the network part with a network optimizer and treats the remaining parts as side constraints, taken into account by a number of dual simplex iterations. Comparing this approach to the standard primal simplex method, we found that the simplex method requires considerably more iterations and time to solve the LP-relaxation. In some cases, the initial LP-relaxation cannot be solved within the time limit (one hour) by the standard simplex method, and hence no solution is found at all. With the network optimizer, the efficiency is increased dramatically. Because in the branch-and-bound tree only minor changes of the LP-problem will occur at successive nodes, the best performance is obtained by using the network optimizer at the top node and the dual simplex method at all other nodes.

Regarding the branch-and-bound scheme, we use the default settings in CPLEX. For instance, a best-first search strategy is adopted which generally gives a better performance than other search orders.

By utilizing all these features, our experiments show that CPLEX is quite efficient in solving CNDP, especially for small problems. However, when the problem size is getting larger, it become unpractical to solve CNDP in this way, both with respect to computational time and storage requirement. Our tests show that if the problem size exceeds approximately 100,000 variables, it becomes too large even to initialize the solving process because of the total memory size required to solve the LP-relaxation and to start the branch-and-bound procedure. This suggests that methods with less memory requirement are needed for large applications.

## 5. COMPUTATIONAL RESULTS

To test our algorithm, we have used a total of 65 test problems in 7 groups of different structures. The ranges of

**Table 1.** The characteristics of the test problems.

| Group | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| $\#P$ | 10 | 8 | 8 | 13 | 15 | 7 | 4 |
| $\lvert\mathcal{N}\rvert_{\max}$ | 17 | 103 | 100 | 148 | 84 | 150 | 61 |
| $\lvert\mathcal{A}\rvert_{\max}$ | 272 | 814 | 980 | 686 | 1,000 | 415 | 1,000 |
| $\lvert\mathcal{C}\rvert_{\max}$ | 272 | 20 | 50 | 235 | 282 | 16 | 50 |

the numbers of nodes, arcs and commodities are 7–150, 42–1,000, and 5–282, respectively. The smallest problem contains 1,260 continuous variables and 30 design variables ($y$-variables with $f_{ij} = 0$ are not counted), while for the largest one these numbers are 191,196 and 1,000. Most of these test problems are based on problems used in Holmberg and Hellstrand (1998), but are significantly modified. Capacities are added and adjusted to affect the solution significantly but still avoid infeasibility. The relations between the numbers of nodes, arcs, and commodities are kept similar to some real-life problems we have encountered (but which, unfortunately, are unavailable for these tests). After solving a preliminary set of test problems, we have removed or modified problems that turned out to be almost uncapacitated, and also problems where the costs for the continuous part (the flow costs) dominate, i.e., problems that in effect were almost linear.

The first group, A, contains networks of complete graphs. Arcs have very large, almost identical fixed charges, and there is (almost) one commodity for each pair of nodes. All commodities have a demand of one unit. Without capacity constraints, these problems would be close to modeling the traveling salesman problem.

Group B contains some randomly generated networks, structured in levels. The commodities are sent from the first level to the last level.

The problems in the next three groups, C, D, and E, have randomly generated unstructured networks. The problems in group C have relatively few commodities and some arcs have fixed costs equal to zero, while the number of commodities are considerably larger in groups D and E, and practically all arcs have nonzero fixed costs.

Group F contains problems with grid structured networks, many nodes, and rather few commodities, sent across the network.

The test problems in group G are basically obtained by replacing the facility nodes in capacitated facility location problems by capacitated arcs, and adding fixed costs on all arcs. All commodities are sent from one single origin (the super source) to the nodes corresponding to the demand nodes of the location problem, i.e., all commodities use exactly two arcs.

Table 1 gives an overview of these problem groups. For each group, we give the number of problems it contains, $\#P$, and the maximal number of nodes, $\lvert\mathcal{N}\rvert_{\max}$, arcs, $\lvert\mathcal{A}\rvert_{\max}$, and commodities, $\lvert\mathcal{C}\rvert_{\max}$.

The implementation is done in Fortran and the tests are carried out on a Sun UltraSparc workstation. Moreover, we limit the execution time to one hour both for CPLEX and for LHBB. Our method contains a number of parameters, which are possible to fine-tune to fit a certain problem structure, (i.e., a certain problem group). However, this was avoided in our tests, and the parameters were chosen to give a reasonably good performance for all the groups. Furthermore, because the tests show that our heuristics often generate near-optimal solutions quite fast, we have utilized the heuristic solutions in our exact solution method, LHBB0, by using LHBB with $\alpha = 0.05$ as the first phase and the exact method as the second phase. Both the best upper bound obtained in the heuristic and the multipliers at the starting node are used in the second phase. The value of $\lambda_0$ is reset to 0.5 in the second phase.

Comparing the test results of the 65 test problems, CPLEX solves 8 problems exactly within 1 hour and finds feasible solutions, without completing the branch-and-bound search, for another 35 problems. For 13 of the remaining problems, CPLEX does not manage to find a feasible integer solution within the prescribed time, while 9 problems are too large for CPLEX to initialize the solution process. LHBB0 solves 9 problems exactly within 1 hour and provides feasible solutions for all the remaining problems. None of the test problems is too large for LHBB0 to start the solution process and find a feasible integer solution.

The computational results are given in Tables 2 and 3, which contain test results for CPLEX, the exact method LHBB0, and the two heuristics with $\beta = 0.1$ and $\alpha = 0.05$. For CPLEX and LHBB0, the tables contain the optimal objective function value (indicated by $^*$), or the best upper bound found if optimality could not be verified within one hour, the number of nodes in the branch-and-bound tree and the solution time in seconds. For $\text{LHBB}_{\beta=0.1}$ and $\text{LHBB}_{\alpha=0.05}$ the best upper bound found and the execution time are given. We use '—' to denote a case where CPLEX does not find a feasible integer solution.

Considering the results for the different groups of problems, we find the following. Of the 10 problems in group A, only one is solved exactly by CPLEX and LHBB0 (with LHBB0 somewhat quicker than CPLEX). CPLEX fails completely to solve five of the problems and manages to solve four approximately, while LHBB0 manages to solve all the nine approximately and significantly better than CPLEX. These problems are also quite difficult for the heuristics. The difficulty of the problems comes mostly from the TSP-structure because they are not very large.

On the level structured problems in group B, CPLEX performs quite well, with results similar to those of LHBB0 (although none of the problems is solved exactly). However, the heuristics perform very well and find comparable solutions in much shorter time, in some cases in a few seconds compared to one hour.

In group C, CPLEX and LHBB0 solve three problems exactly, with LHBB0 quicker on two of them. For the remaining problems, LHBB0 yields better solutions than CPLEX in all cases but one, while CPLEX fails in two cases. Also here the heuristics are very quick.

**Table 2.** Computational results, Part 1.

| Name | Size | | | CPLEX | | | LHBB0 | | | $LHBB_{\beta=0.1}$ | | $LHBB_{\alpha=0.05}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\lvert\mathcal{A}\rvert$ | $\lvert\mathcal{N}\rvert$ | $\lvert\mathcal{C}\rvert$ | $\bar{v}$ | Nodes | Time | $\bar{v}$ | Nodes | Time | $\bar{v}$ | Time | $\bar{v}$ | Time |
| A1 | 42 | 7 | 30 | *4101.00 | 606 | 190.09 | *4101.00 | 9552 | 183.20 | 4565.00 | 3.91 | 4421.00 | 11.50 |
| A2 | 56 | 8 | 42 | 5493.00 | 2293 | 3587.19 | 5192.00 | 93291 | 3600.06 | 5762.00 | 21.20 | 5401.00 | 67.73 |
| A3 | 56 | 8 | 42 | 5311.00 | 2211 | 3581.09 | 5245.00 | 94049 | 3600.05 | 5664.00 | 18.79 | 5318.00 | 52.55 |
| A4 | 90 | 10 | 72 | 9012.00 | 298 | 3573.43 | 7404.00 | 21867 | 3600.04 | 8047.00 | 79.16 | 7404.00 | 473.15 |
| A5 | 132 | 12 | 110 | — | — | — | 9612.00 | 11871 | 3600.25 | 10469.00 | 429.61 | 9612.00 | 3600.25 |
| A6 | 210 | 15 | 182 | — | — | — | 13423.00 | 3835 | 3600.79 | 14459.00 | 1059.40 | 13423.00 | 3600.79 |
| A7 | 272 | 17 | 240 | — | — | — | 14726.00 | 2099 | 3603.09 | 16843.00 | 1956.47 | 14726.00 | 3603.09 |
| A8 | 90 | 10 | 90 | 10138.00 | 98 | 3505.61 | 8227.00 | 10371 | 3600.20 | 9090.00 | 99.84 | 8227.00 | 1078.33 |
| A9 | 210 | 15 | 210 | — | — | — | 14562.00 | 3188 | 3600.57 | 15638.00 | 1683.20 | 14562.00 | 3600.57 |
| A10 | 272 | 17 | 272 | — | — | — | 15277.00 | 1633 | 3603.54 | 17568.00 | 2428.26 | 15277.00 | 3603.54 |
| B1 | 357 | 56 | 10 | 143801.00 | 33133 | 3583.48 | 144436.00 | 34065 | 3600.03 | 144926.00 | 20.19 | 146279.00 | 1.23 |
| B2 | 290 | 41 | 10 | 16446.00 | 34142 | 3592.29 | 16152.00 | 59237 | 3600.03 | 16601.00 | 2.45 | 16601.00 | 0.97 |
| B3 | 814 | 103 | 12 | 135194.00 | 4855 | 3577.55 | 134960.00 | 14035 | 3600.18 | 135815.00 | 19.56 | 135815.00 | 3.78 |
| B4 | 370 | 51 | 20 | 259865.00 | 10496 | 3578.35 | 259558.00 | 29738 | 3600.02 | 259940.00 | 6.24 | 260391.00 | 2.79 |
| B5 | 365 | 56 | 10 | 233048.00 | 49122 | 3577.58 | 233337.00 | 36759 | 3600.23 | 233648.00 | 12.22 | 234298.00 | 1.21 |
| B6 | 290 | 41 | 10 | 14761.00 | 60897 | 3580.83 | 15491.00 | 68127 | 3600.05 | 15889.00 | 2.12 | 15889.00 | 0.93 |
| B7 | 814 | 103 | 12 | 55143.00 | 3786 | 3583.08 | 55274.00 | 13095 | 3600.15 | 56323.00 | 6.93 | 56290.00 | 4.43 |
| B8 | 370 | 51 | 20 | 228207.00 | 11133 | 3580.21 | 228176.00 | 28092 | 3600.04 | 228415.00 | 10.40 | 228814.00 | 2.85 |
| C1 | 300 | 51 | 30 | 322938.00 | 388 | 3581.63 | 311352.00 | 22624 | 3600.55 | 314046.00 | 75.84 | 311413.00 | 59.61 |
| C2 | 586 | 100 | 50 | — | — | — | 2008785.00 | 5763 | 3600.16 | 1979307.00 | 139.60 | 2037130.00 | 18.94 |
| C3 | 980 | 100 | 50 | — | — | — | 1021361.00 | 2939 | 3602.45 | 1113928.00 | 3649.06 | 2031132.00 | 23.12 |
| C4 | 290 | 50 | 40 | 547561.00 | 277 | 3578.03 | 477021.00 | 16992 | 3600.09 | 534841.00 | 57.47 | 535873.00 | 18.21 |
| C5 | 430 | 30 | 20 | *243114.00 | 20 | 23.24 | *243114.00 | 152 | 15.56 | 243114.00 | 2.09 | 243114.00 | 2.96 |
| C6 | 360 | 31 | 20 | 388135.00 | 12980 | 3578.39 | 431021.00 | 19830 | 3600.18 | 405410.00 | 21.21 | 443259.00 | 2.77 |
| C7 | 340 | 21 | 15 | *137598.00 | 155 | 90.36 | *137598.00 | 544 | 39.76 | 137977.00 | 3.04 | 137706.00 | 5.02 |
| C8 | 271 | 25 | 20 | *220414.00 | 8 | 5.82 | *220414.00 | 80 | 5.83 | 220414.00 | 1.33 | 220414.00 | 1.55 |
| D1 | 72 | 10 | 90 | *722089.00 | 452 | 672.04 | *722089.00 | 6979 | 384.89 | 726802.00 | 9.47 | 723291.00 | 15.47 |
| D2 | 68 | 10 | 90 | 695073.00 | 1104 | 3535.18 | 689616.00 | 60414 | 3600.01 | 693309.00 | 19.62 | 700195.00 | 9.20 |
| D3 | 148 | 33 | 134 | *1098362.00 | 259 | 1034.78 | *1098362.00 | 1848 | 308.90 | 1115833.00 | 19.33 | 1325866.00 | 3.46 |
| D4 | 146 | 33 | 185 | 2459763.00 | 596 | 3528.43 | 2506161.00 | 16325 | 3600.04 | 2548429.00 | 16.69 | 2894607.00 | 7.04 |
| D5 | 470 | 92 | 215 | — | — | — | 1336035.00 | 5911 | 3600.51 | 1361945.00 | 56.16 | 1378931.00 | 19.61 |
| D6 | 180 | 15 | 157 | 1096691.00 | 410 | 3523.54 | 1081088.00 | 24569 | 3600.19 | 1092356.00 | 69.31 | 1086297.00 | 108.20 |
| D7 | 614 | 72 | 198 | — | — | — | 1479209.00 | 4928 | 3600.58 | 1506709.00 | 86.37 | 1478624.00 | 2778.51 |

**Table 3.** Computational results, Part 2.

| Name | Size | | | CPLEX | | | LHBB0 | | | $LHBB_{\beta=0.1}$ | | $LHBB_{\alpha=0.05}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\lvert\mathcal{A}\rvert$ | $\lvert\mathcal{N}\rvert$ | $\lvert\mathcal{C}\rvert$ | $\bar{v}$ | Nodes | Time | $\bar{v}$ | Nodes | Time | $\bar{v}$ | Time | $\bar{v}$ | Time |
| D8 | 686 | 76 | 235 | — | — | — | 1541466.00 | 2808 | 3600.40 | 1550952.00 | 76.39 | 1540786.00 | 2249.11 |
| D9 | 484 | 22 | 215 | — | — | — | 1310727.00 | 9940 | 3601.81 | 1343058.00 | 371.67 | 1609211.00 | 25.04 |
| D10 | 324 | 18 | 235 | 1543134.00 | 285 | 3596.27 | 1541466.00 | 9866 | 3600.07 | 1556138.00 | 173.37 | 1540786.00 | 1079.85 |
| D11 | 174 | 68 | 90 | 756581.00 | 742 | 3543.75 | 755826.00 | 17623 | 3600.07 | 769447.00 | 57.11 | 761866.00 | 71.06 |
| D12 | 159 | 63 | 90 | 736146.00 | 801 | 3535.00 | 710580.00 | 13113 | 3600.17 | 739309.00 | 211.51 | 722106.00 | 18.44 |
| D13 | 399 | 148 | 157 | 1142435.00 | 118 | 3546.75 | 1139080.00 | 6483 | 3600.90 | 1169328.00 | 381.55 | 1155254.00 | 880.15 |
| E1 | 90 | 10 | 90 | 872304.00 | 995 | 3581.77 | 834621.00 | 43900 | 3600.16 | 856597.00 | 59.66 | 844654.00 | 20.84 |
| E2 | 50 | 10 | 90 | *798318.00 | 319 | 276.46 | *798318.00 | 4213 | 342.60 | 812254.00 | 9.52 | 800288.00 | 13.65 |
| E3 | 678 | 84 | 282 | — | — | — | 1806887.00 | 3615 | 3601.69 | 1833724.00 | 114.29 | 1810983.00 | 3601.29 |
| E4 | 686 | 76 | 231 | — | — | — | *1481573.00 | 130 | 806.56 | 1495545.00 | 67.69 | 1481573.00 | 750.66 |
| E5 | 586 | 76 | 239 | — | — | — | *1529770.00 | 415 | 615.94 | 1550482.00 | 81.88 | 1529770.00 | 1393.82 |
| E6 | 500 | 40 | 78 | — | — | — | 951138.00 | 8451 | 3601.66 | 975379.00 | 224.86 | 951138.00 | 34.64 |
| E7 | 1000 | 40 | 78 | — | — | — | 872198.00 | 2558 | 3606.71 | 933251.00 | 1109.58 | 872198.00 | 65.33 |
| E8 | 594 | 76 | 211 | — | — | — | 1509182.00 | 4338 | 3601.45 | 1544685.00 | 75.97 | 1563318.00 | 26.37 |
| E9 | 1000 | 50 | 98 | — | — | — | 838799.00 | 2853 | 3601.87 | 903694.00 | 1123.70 | 838799.00 | 111.50 |
| E10 | 358 | 50 | 98 | 665830.00 | 35 | 3550.66 | 644279.00 | 12548 | 3601.13 | 669398.00 | 76.20 | 647213.00 | 35.68 |
| E11 | 421 | 60 | 111 | — | — | — | 1682178.00 | 4954 | 3600.11 | 1743821.00 | 100.74 | 1744303.00 | 9.67 |
| E12 | 492 | 70 | 136 | — | — | — | 708890.00 | 5328 | 3602.05 | 723136.00 | 138.60 | 714882.00 | 570.05 |
| E13 | 546 | 75 | 148 | — | — | — | 2264311.00 | 2255 | 3600.81 | 2330146.00 | 182.15 | 2376015.00 | 24.17 |
| E14 | 296 | 19 | 211 | 1509182.00 | 443 | 3594.06 | 1509182.00 | 13022 | 3600.17 | 1520440.00 | 98.99 | 1548067.00 | 15.85 |
| E15 | 400 | 20 | 282 | — | — | — | 1814226.00 | 5900 | 3600.10 | 1818310.00 | 164.70 | 1809645.00 | 3600.05 |
| F1 | 270 | 100 | 10 | 681014.00 | 1944 | 3554.22 | 671153.00 | 45550 | 3600.05 | 670489.00 | 14.72 | 673386.00 | 1.50 |
| F2 | 270 | 100 | 10 | 789901.00 | 1937 | 3550.87 | 778940.00 | 56045 | 3600.26 | 775976.00 | 7.52 | 786609.00 | 1.17 |
| F3 | 270 | 100 | 10 | 793528.00 | 2066 | 3552.34 | 776165.00 | 47544 | 3600.04 | 779683.00 | 24.39 | 776165.00 | 1.35 |
| F4 | 255 | 100 | 5 | 400715.00 | 10789 | 3550.57 | 390321.00 | 95906 | 3600.06 | 390338.00 | 1.39 | 390338.00 | 0.82 |
| F5 | 270 | 100 | 16 | 1380708.00 | 1493 | 3578.57 | 1367226.00 | 33882 | 3600.00 | 1368728.00 | 6.94 | 1377054.00 | 1.73 |
| F6 | 410 | 150 | 10 | 1304046.50 | 976 | 3555.79 | 1268747.00 | 33207 | 3600.23 | 1260779.00 | 41.38 | 1287462.00 | 2.68 |
| F7 | 415 | 150 | 15 | 1280114.00 | 297 | 3557.71 | 1264171.00 | 20550 | 3600.12 | 1264856.00 | 141.90 | 1283643.00 | 4.60 |
| G1 | 1000 | 61 | 50 | 187414.00 | 143 | 3578.23 | 189112.00 | 5472 | 3600.04 | 188089.00 | 71.65 | 191122.00 | 8.40 |
| G2 | 820 | 61 | 40 | 148246.00 | 94131 | 3454.06 | 149380.00 | 9568 | 3600.78 | 149239.00 | 51.01 | 157357.00 | 5.04 |
| G3 | 840 | 41 | 20 | 57229.00 | 1299 | 3577.56 | 57643.00 | 16084 | 3600.18 | 58614.00 | 42.11 | 74785.00 | 2.38 |
| G4 | 930 | 61 | 30 | *96122.00 | 52538 | 2453.21 | 96781.00 | 11803 | 3600.10 | 96963.00 | 31.30 | 113064.00 | 4.26 |

In group D, two problems are solved to optimality by CPLEX and LHBB0 (somewhat quicker), while CPLEX fails in four cases. In most cases LHBB0 yields better solutions than CPLEX. In group E, CPLEX fails on 11 problem and solves 1 exactly. This problem, and two of the problems that CPLEX fails on, are solved to optimality by LHBB0. Again, LHBB0 in general yields better solutions than CPLEX. The results of the heuristics on groups D and E vary significantly, and on some of the problems they take a quite long time.

Group F contains grid structured problems, which are surprisingly difficult. No problem is solved exactly by any method, but all are solved approximately. LHBB0 yields better solutions than CPLEX for all problems. Here we can note that the heuristics actually often yield better solutions in a couple of seconds than CPLEX in one hour.

In group G CPLEX solves one problem exactly, while LHBB0 does not solve any problem to optimality. CPLEX yields better solutions than LHBB0 also for the remaining three problems. Here CPLEX performs better than on the other groups. We should, however, keep in mind the very special structure of these problems.

LHBB0 uses as a first phase the heuristic (with $\alpha = 0.05$), slightly modified by solving PS somewhat less frequently. In 15 cases LHBB0 does not find a better solution than the heuristic. Of these, four solutions are optimal, but for some others the heuristic uses all or much of the allowed hour, leaving very little time for the second phase of LHBB0. The slight difference between the heuristic and the first phase of LHBB0 has some effects, namely that the heuristic in some cases actually yields better solutions than LHBB0, but in another case, E5, takes much longer time.

To make more detailed comparisons between the performances of the different methods, we divide all the test problems into three categories, depending on the test results. The test problems in the first category are solved exactly by CPLEX and LHBB0 within one hour, while the second category contains those problems for which CPLEX and LHBB0 find feasible solutions but do not solve completely in one hour. Finally, the last category is composed by those problems for which CPLEX fails to find any feasible integer solution. We can call the problems in these three categories "easy," "difficult," and "large." Table 4 contains the numbers of problems of each group that these categories contain. (The second group also contains one problem solved to optimality by CPLEX, but not by LHBB0, and the third group contains two problems solved to optimality by LHBB0.)

**Table 4.** Relationship between problem groups and categories.

| | A | B | C | D | E | F | G | Total |
|---|---|---|---|---|---|---|---|---|
| Category 1 ("easy") | 1 | 0 | 3 | 2 | 1 | 0 | 0 | 7 |
| Category 2 ("difficult") | 4 | 8 | 3 | 7 | 3 | 7 | 4 | 36 |
| Category 3 ("large") | 5 | 0 | 2 | 4 | 11 | 0 | 0 | 22 |

Comparing the exact methods, we find that the number of nodes in the branch-and-bound tree is much higher in LHBB0 than in CPLEX. (This comparison is not relevant for problems not solved exactly, since one of the methods might be much closer to finishing than the other.) For the seven problems that are solved exactly by both methods, LHBB0 yields much larger trees. One example is problem A1, where CPLEX yields 606 nodes and 190 seconds, while LHBB0 yields 9,552 nodes and 183 seconds. In this case CPLEX spends in average 0.31 seconds on each node in the tree, while LHBB0 spends 0.019 seconds on each node. The tree size makes it crucial that the subproblems are solved very quickly. (Using the relaxation preferred in Gendron and Crainic (1994), this would probably be different, because then the subproblem takes a longer time to solve but probably yields better bounds.)

In average LHBB0 yields more than 10 times as many nodes as CPLEX on these 7 problems. In preliminary computational tests, the limits on the number of subgradient iterations in each node were increased very much, to check if the reason for the large number of nodes is that LD is not solved sufficiently well. However, even in that case, LHBB0 yields much larger trees.

The average computational results for the first category are summarized in Figure 1, which displays the relative error and the execution time. The two different versions of LHBB are denoted $\text{LHBB}_\alpha$ and $\text{LHBB}_\beta$, depending on which heuristic is used for fixation of variables. (Recall that LHBB0 is $\text{LHBB}_\beta$ with $\beta = 0$.)

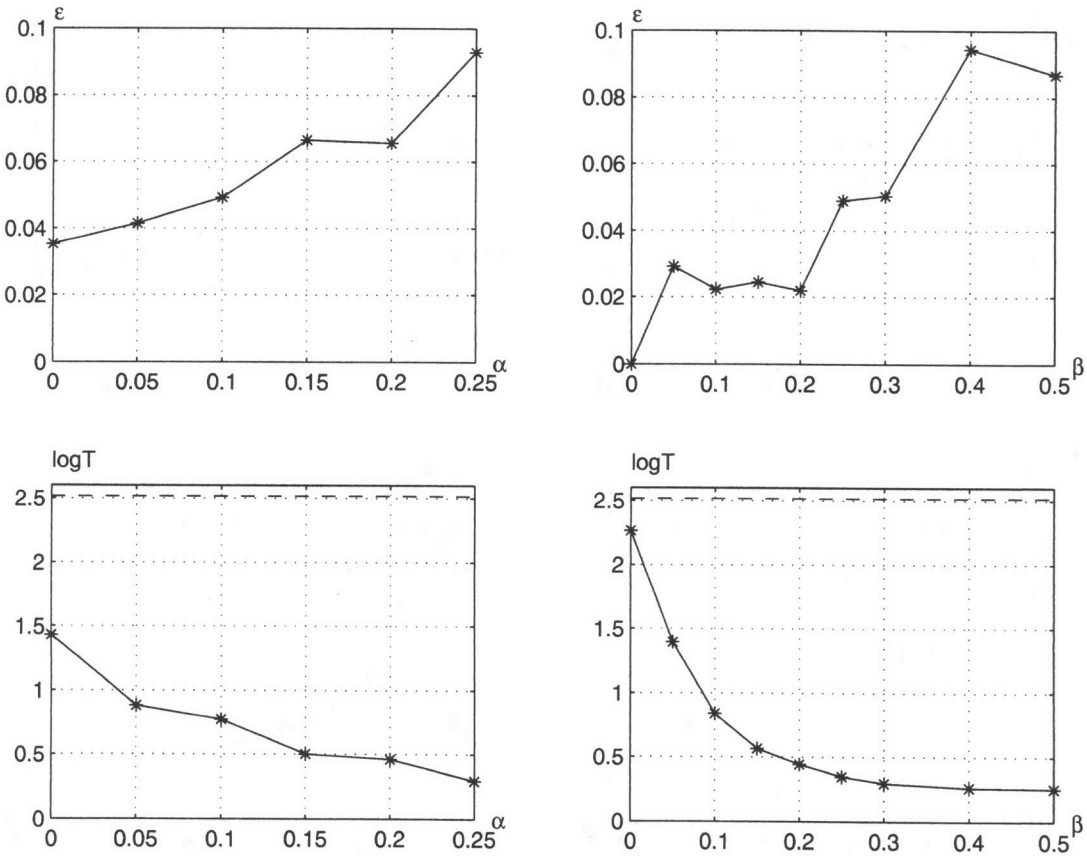
The first problem category contains the seven "easy" test problems that are solved exactly by both CPLEX and LHBB0. The solution times of LHBB0 are shorter than those obtained by CPLEX in five cases, practically equal in one case, and longer in one case. In average, LHBB0 uses 183 seconds, while CPLEX used 260 seconds.

Figure 1 confirms that the average solution time T for our exact method is smaller than for CPLEX. If either of the heuristics is used, we obtain a fairly small $\varepsilon$ even for quite large values of $\alpha$ or $\beta$. For example, $\varepsilon$ is less than 4% for $\beta \leqslant 0.2$. At this stage, T is about a few seconds, which is approximately 1.5% of the effort required to solve the problem exactly. Notice that the figure is not mainly a comparison between CPLEX and our heuristics because these problems are solved exactly by CPLEX. Rather, the figure shows how $\varepsilon$ and T are related to each other with respect to different values of $\alpha$ and $\beta$, and it suggests that our heuristics are capable of generating good solutions with small computational effort.

Based on Figure 1, we can compare $\text{LHBB}_\alpha$ to $\text{LHBB}_\beta$ by simply observing the corresponding values of $\varepsilon$ given a certain value of T, and vice versa. For small values of $\alpha$, $\text{LHBB}_\alpha$ is weaker and quicker, but otherwise $\text{LHBB}_\beta$ seems preferable.

For the second category, both CPLEX and LHBB0 yield feasible solutions (CPLEX solves one of them to optimality) within one hour. Of the 36 problems, LHBB0 finds better

**Figure 1.** Average computational results for the 7 problems in category 1 with $LHBB_\alpha$ on the left and $LHBB_\beta$ to the right. For each algorithm, $\varepsilon$ is the relative difference between the best solutions found and the optimum, log T is the 10-based logarithm of the execution time in seconds. The dashed lines show the average time required for CPLEX.




solutions in 25 cases, worse in 10 cases, and the same in one case. The test results are summarized in Figure 2.

It is clear that $LHBB_\beta$ can find better solutions for the problems in category 2 than CPLEX can, with a considerable speed-up factor. Better solutions are obtained for $\beta$ up to 0.1, which yields a quite short solution time. For small values of $\alpha$, $LHBB_\alpha$ yields solutions somewhat worse, but close to what CPLEX yields, in a much shorter time. To generate a solution at least as good as the one found by CPLEX during one hour, it takes about 57 seconds for $LHBB_\beta$, which confirms that the proposed method is very fast in obtaining good feasible solutions.

The last category contains 22 problems for which CPLEX does not find a feasible solution. Among these problems, two are solved exactly within one hour by LHBB0. The test results are shown in Figure 3.

The only available values for the problems in this category are the upper and lower bounds generated by our algorithm. Because no extra attempts have been made to improve the lower bound found at the top node of the branch-and-bound tree, the lower bounds obtained are usually not very close to optimum.

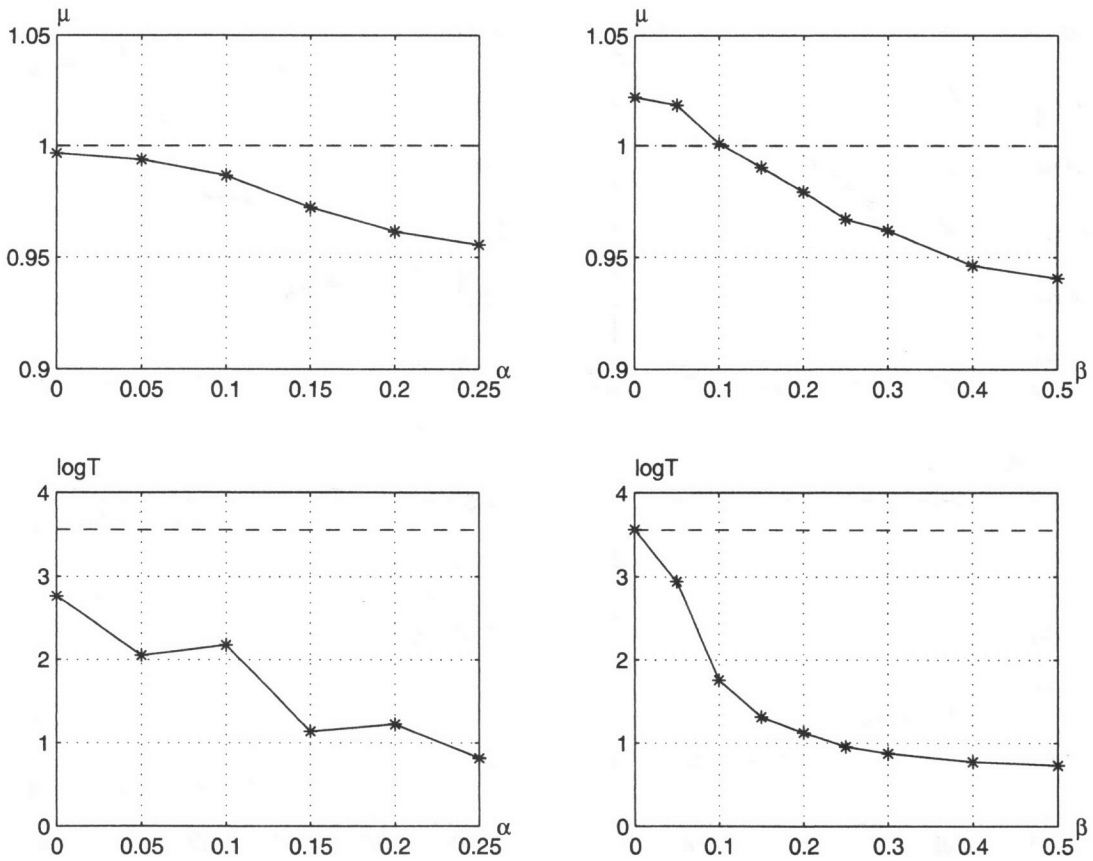
To investigate the capability of our algorithm, we have applied $LHBB_\alpha$ and $LHBB_\beta$ to a large problem of the same structure as those in group E, containing 1,000 design variables and 600,000 flow variables. Both $LHBB_\alpha$ and $LHBB_\beta$ can start to solve this problem. The test results for some parameter values are shown in Table 5.

Table 5 shows that good feasible solutions are obtained by both $LHBB_\beta$ and $LHBB_\alpha$ because for all the parameter values, $\varepsilon$ is less than 3%, and often less than 2%. Notice that while $\alpha$ or $\beta$ increases, $\varepsilon$ does not increase monotonously. This can be explained by the fact that for such a large problem, only few nodes can be searched for small values of the parameters, while better solutions are obtained if more nodes are searched. Another observation is that because the problem is very large, it seems easier to adjust the computational efforts in $LHBB_\beta$ than $LHBB_\alpha$ because an explicit number of variables are fixed in $LHBB_\beta$.

Summarizing the computational results, we find that LHBB0 is better than CPLEX in 52 cases (better solutions in the same time or shorter time for the same solution), worse in 11 cases, and practically equivalent in 2 cases. The heuristics in an absolute majority of cases yield fairly good solutions in a much shorter time.

In Table 6 some statistics for the solutions obtained by CPLEX are given (including solutions not shown to be optimal). We give the number of arcs, $|\mathcal{A}|$, the number of

**Figure 2.** Average test results for the 36 problems in category 2. The value $\mu$ is the quotient between the objective function values found by CPLEX and LHBB, i.e. the improvement of LHBB over CPLEX (CPLEX corresponds to $\mu = 1.0$). The lower images show the 10-based logarithm of the average execution time T in seconds. The dashed lines are the execution time ($\approx$ 1 hour) for CPLEX.




opened arcs, $|\mathcal{A}_O|$, and the number of saturated arcs (where the total flow is equal to the capacity), $|\mathcal{A}_S|$. We also give the proportion between the total fixed costs and the total costs, $f_P$, the average utilization of the opened arcs (i.e., the total flow divided by the total capacity of the opened arcs), $U_O$, and the average total utilization (i.e., the total flow divided by the total capacity), $U_{TOT}$.

Finally, we give the gap between the optimal objective function value of the LP-relaxation and the upper bound obtained by "intelligent rounding," i.e., by making this solution integer and feasible (by paying the correct fixed costs), $G_1$, in %, the gap between the best feasible integer solution found by CPLEX and the optimal objective function value of the LP-relaxation, $G_2$, in %, and the gap between the upper and lower bounds in the first node of LHBB, $G_3$, in %.

In some cases $G_1$ is less than $G_2$, which indicates that CPLEX may perform worse than "intelligent rounding." The gaps $G_3$ are in general quite large, up to 930%, but in some cases smaller than $G_1$. The gaps $G_3$ are also available for the problems where CPLEX fails to find a feasible solution, even though these problems are not included in Table 6. Of these problems, A5,6,7,9 have initial gaps between 124% and 183%, and C3 an initial gap of 248%. Apart from these cases, the problems not included in Table 6 do not have significantly larger gaps than those included.

We have also made tests using LHBB with $\beta = 0.1$ (instead of $\alpha = 0.05$) as first phase of LHBB0. This modification enables verification of optimal solution of 11 problems (instead of 9) within 1 hour but does not otherwise yield a general improvement.

A large part of the time used in LHBB is spent in solving PS exactly, which is done with CPLEX. A recent implementation of a column-generation method (Holmberg and Yuan 1998) seems to be able to solve problems similar to PS faster than CPLEX, so it is possible that LHBB can be made faster by inclusion of that method. It is also possible to make the branching even smarter, which might decrease the size of the trees. Finally, another idea to be pursued in the future is to improve that lower bounds by including valid inequalities for the convex hull of the feasible integer solutions.

## 6. CONCLUSIONS

In this paper we propose a method based on a combination of a Lagrangian heuristic and branch-and-bound for the

**Figure 3.** Test results for the 22 problems in category 3. For each value of $\alpha$ and $\beta$, the figure displays the numbers of problems for which the obtained relative differences between the upper and lower bounds are in the ranges $[0, 0.05]$, $(0.05, 0.1]$ and $(0.1, \infty]$. It also shows the 10-based logarithm of the average solution time T in seconds.

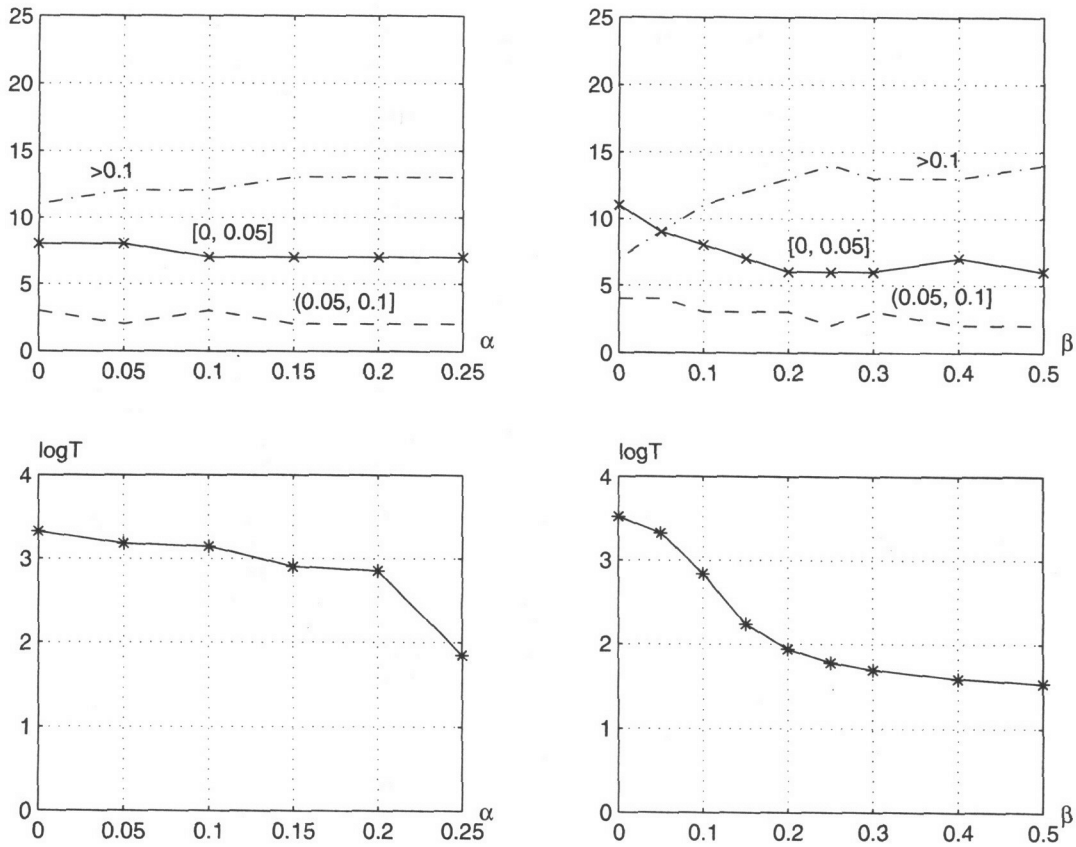



**Table 5.** Test results for a problem with 1,000 design variables and 600,000 continuous variables.

| $\alpha$ | 0 | | 0.1 | | 0.15 | | 0.2 | | 0.3 | |
|---|---|---|---|---|---|---|---|---|---|---|
| $LHBB_\alpha$ | T | $\varepsilon$ | T | $\varepsilon$ | T | $\varepsilon$ | T | $\varepsilon$ | T | $\varepsilon$ |
| | 3601.09 | 2.00 | 3604.52 | 1.02 | 3601.97 | 1.10 | 3601.87 | 0.98 | 1315.91 | 0.98 |
| $\beta$ | 0 | | 0.1 | | 0.2 | | 0.3 | | 0.4 | |
| $LHBB_\beta$ | T | $\varepsilon$ | T | $\varepsilon$ | T | $\varepsilon$ | T | $\varepsilon$ | T | $\varepsilon$ |
| | 3602.70 | 2.04 | 3605.01 | 1.93 | 562.98 | 2.54 | 335.15 | 2.19 | 271.85 | 2.73 |

*Note:* $\varepsilon$ is the relative difference between the upper and lower bound generated in percent and T is the solution time in seconds. The limit of execution time is approximately 1 hour = 3,600 seconds.

capacitated network design problem. The basic scheme and various enhancements are discussed. Especially, we can use the fact that the separability of the Lagrangian relaxation coincides with the effects of the branching to construct efficient penalty tests. We also develop new cutting criteria. The method can either be an exact solution method, or with the introduction of heuristic variable fixation techniques, a faster heuristic, with a parameter controlling the quality of the approximation (trading solution time for solution quality). The computational results suggest that the proposed methods provides a quite efficient way in obtaining near-optimal solutions with small computational effort, especially for large-scale problems. This makes it interesting for practical applications. Furthermore, it provides means to adapt the algorithm complexity to the problem size.

Tested on 65 problems, our method is better and/or faster in 52 cases, compared to the state-of-the-art code CPLEX.

It is also interesting to notice that although we are mostly interested in obtaining good *primal* solutions, it is achieved by effectively utilizing information generated by the *dual* procedure. This confirms the advantages of combining primal and (Lagrangian) dual procedures for structured mixed integer problems.

**Table 6.** Statistics for the solutions obtained by CPLEX.

| Name | $\|\mathcal{A}\|$ | $\|\mathcal{A}_O\|$ | $\|\mathcal{A}_S\|$ | $f_P$ | $U_O$ | $U_{TOT}$ | $G_1$ (%) | $G_2$ (%) | $G_3$ (%) |
|---|---|---|---|---|---|---|---|---|---|
| A1 | 42 | 12 | 9 | 0.878 | 1.000 | 0.228 | 226.20 | 9.69 | 49.93 |
| A2 | 56 | 16 | 5 | 0.874 | 0.916 | 0.209 | 273.50 | 16.99 | 102.92 |
| A3 | 56 | 16 | 6 | 0.904 | 0.888 | 0.202 | 275.74 | 14.81 | 77.58 |
| A4 | 90 | 24 | 9 | 0.888 | 0.813 | 0.176 | 327.74 | 29.13 | 117.08 |
| A8 | 90 | 23 | 10 | 0.868 | 0.777 | 0.189 | 392.78 | 31.99 | 103.16 |
| B1 | 357 | 54 | 19 | 0.062 | 0.691 | 0.099 | 1.93 | 1.16 | 207.92 |
| B2 | 290 | 54 | 10 | 0.667 | 0.597 | 0.099 | 23.02 | 18.65 | 47.78 |
| B3 | 814 | 80 | 27 | 0.109 | 0.762 | 0.071 | 2.75 | 2.58 | 346.28 |
| B4 | 370 | 60 | 7 | 0.034 | 0.523 | 0.079 | 0.56 | 0.64 | 211.08 |
| B5 | 365 | 48 | 7 | 0.277 | 0.655 | 0.070 | 0.98 | 1.03 | 226.98 |
| B6 | 290 | 39 | 6 | 0.628 | 0.731 | 0.096 | 17.81 | 7.68 | 44.87 |
| B7 | 814 | 108 | 55 | 0.136 | 0.782 | 0.104 | 6.07 | 5.62 | 929.88 |
| B8 | 370 | 56 | 5 | 0.037 | 0.548 | 0.080 | 0.56 | 0.60 | 195.84 |
| C1 | 300 | 54 | 4 | 0.444 | 0.455 | 0.110 | 38.97 | 7.12 | 11.93 |
| C4 | 290 | 111 | 22 | 0.511 | 0.627 | 0.240 | 33.41 | 16.96 | 30.35 |
| C5 | 430 | 15 | 5 | 0.293 | 0.632 | 0.062 | 6.31 | 1.21 | 1.49 |
| C6 | 360 | 8 | 9 | 0.229 | 0.623 | 0.085 | 31.43 | 16.37 | 142.83 |
| C7 | 340 | 29 | 4 | 0.491 | 0.555 | 0.048 | 22.15 | 4.49 | 5.72 |
| C8 | 271 | 15 | 2 | 0.225 | 0.530 | 0.074 | 8.66 | 1.29 | 3.08 |
| D1 | 72 | 30 | 10 | 0.340 | 0.846 | 0.439 | 11.53 | 1.10 | 9.76 |
| D2 | 68 | 42 | 17 | 0.394 | 0.885 | 0.542 | 13.74 | 3.12 | 16.80 |
| D3 | 148 | 54 | 6 | 0.180 | 0.462 | 0.171 | 5.05 | 0.56 | 40.10 |
| D4 | 146 | 68 | 7 | 0.212 | 0.393 | 0.176 | 4.04 | 0.62 | 24.71 |
| D6 | 180 | 56 | 4 | 0.307 | 0.536 | 0.149 | 15.14 | 3.71 | 11.39 |
| D10 | 324 | 62 | 4 | 0.215 | 0.318 | 0.038 | 5.72 | 1.13 | 3.72 |
| D11 | 174 | 52 | 14 | 0.321 | 0.863 | 0.288 | 9.07 | 0.95 | 141.79 |
| D12 | 159 | 55 | 24 | 0.399 | 0.847 | 0.376 | 12.85 | 5.34 | 17.55 |
| D13 | 399 | 93 | 8 | 0.277 | 0.692 | 0.127 | 7.91 | 2.58 | 53.22 |
| E1 | 90 | 55 | 17 | 0.480 | 0.801 | 0.480 | 14.51 | 5.83 | 41.82 |
| E2 | 50 | 34 | 17 | 0.328 | 0.843 | 0.573 | 11.09 | 1.41 | 20.34 |
| E10 | 358 | 118 | 4 | 0.310 | 0.332 | 0.060 | 13.28 | 4.09 | 35.01 |
| E14 | 296 | 74 | 12 | 0.211 | 0.530 | 0.093 | 4.28 | 0.44 | 8.58 |
| F1 | 270 | 113 | 4 | 0.140 | 0.585 | 0.176 | 11.02 | 6.42 | 15.64 |
| F2 | 270 | 101 | 9 | 0.113 | 0.581 | 0.172 | 6.04 | 4.92 | 11.08 |
| F3 | 270 | 96 | 9 | 0.114 | 0.560 | 0.172 | 8.05 | 5.00 | 12.97 |
| F4 | 255 | 140 | 22 | 0.334 | 0.574 | 0.279 | 17.50 | 15.26 | 33.79 |
| F5 | 270 | 117 | 11 | 0.070 | 0.544 | 0.192 | 3.36 | 2.76 | 8.09 |
| F6 | 410 | 204 | 20 | 0.143 | 0.613 | 0.227 | 6.89 | 7.33 | 16.77 |
| F7 | 415 | 182 | 22 | 0.118 | 0.583 | 0.189 | 5.24 | 5.35 | 16.39 |
| G1 | 1000 | 62 | 5 | 0.195 | 0.259 | 0.021 | 1.99 | 0.79 | 7.36 |
| G2 | 820 | 65 | 9 | 0.217 | 0.476 | 0.036 | 4.14 | 1.32 | 8.27 |
| G3 | 840 | 50 | 10 | 0.322 | 0.508 | 0.031 | 12.42 | 2.13 | 35.32 |
| G4 | 930 | 50 | 6 | 0.257 | 0.577 | 0.031 | 2.67 | 1.55 | 20.77 |

## ACKNOWLEDGMENT

This work has been financed by the Swedish Transport and Communication Research Board (KFB), the National Board for Industrial and Technical Development (NUTEK), Telia AB, and Ericsson Telecom AB.

## REFERENCES

Allen, E., R. Helgason, J. Kennington, B. Shetty. 1987. A generalization of Polyak's convergence result for subgradient optimization. *Math. Programming* **37** 309–317.

Balakrishnan, A., T. L. Magnanti, R. T. Wong. 1989. A dual-ascent procedure for large-scale uncapacitated network design. *Oper. Res.* **37** 716–740.

Barnhart, C. 1993. Dual-ascent methods for large-scale multicommodity flow problems. *Naval Res. Logist.* **40** 305–324.

Bazaraa, M. S., H. D. Sherali. 1981. On the choice of step size in subgradient optimization. *European J. Oper. Res.* 7 380–388.

Busacker, R., P. Gowen. 1961. A procedure for determining a family of minimum-cost network flow patterns. ORO Technical Report 15, Johns Hopkins University, Baltimore, MD.

Crowder, H. 1976. Computational improvements for subgradient optimization. *Symposia Mathematica*, Vol. XIX. Academic Press, London, 357–372.

Gaivoronsky, A. 1988. Stochastic quasigradient methods and their implementation. Y. Ermoliev et al., eds. *Numerical Techniques for Stochastic Optimization*, Vol. 10 of *Springer*

*Series in Computational Mathematics*. Chapter 16, Springer-Verlag, Berlin, Germany 313–351.

Gallo, G., S. Pallottino. 1988. Fortran codes for network optimization: Shortest path algorithms. *Ann. Oper. Res.* **13** 3–79.

Gendron, B., T. G. Crainic. 1994. Relaxations for multicommodity capacitated network design problems. Research Report CRT-965, Centre de Recherche sur les Transports, Université de Montréal, Canada.

Goffin, J. L. 1977. On convergence rates of subgradient optimization methods. *Math. Programming* **13** 329–347.

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

——, P. Wolfe, H. P. Crowder. 1974. Validation of subgradient optimization. *Math. Programming* **6** 62–88.

Hellstrand, J., T. Larsson, A. Migdalas. 1992. A characterization of the uncapacitated network design polytope. *Oper. Res. Letters* **12** 159–163.

Holmberg, K. 1995. Lagrangian heuristics for linear cost multicommodity network flow problems. Working Paper LiTH-MAT/OPT-WP-1995-01, Division of Optimization, Department of Mathematics, Linköping Institute of Technology, Sweden.

——, J. Hellstrand. 1998. Solving the uncapacitated network design problem by a Lagrangian heuristic and branch-and-bound. *Oper. Res.* **46** 247–259.

——, D. Yuan. 1998. A multicommodity network flow problem with side constraints on paths solved by column generation. Research Report LiTH-MAT-R-1998-05, Department of Mathematics, Linköping Institute of Technology, Sweden.

Iri, M. 1960. A new method for solving transportation-network problems. *J. OR Soc. Japan* **3** 27–87.

Jewell, W. 1958. Optimal flow through networks. Technical Report 8, OR Center, MIT, Cambridge, MA.

Jones, K. L., I. J. Lustig, J. M. Farvolden, W. B. Powell. 1993. Multicommodity network flows: The impact of formulation on decomposition. *Math. Programming* **62** 95–117.

Lamar, B. W., Y. Sheffi, W. B. Powell. 1990. A capacity improvement lower bound for fixed charge network design problems. *Oper. Res.* **38** 704–710.

Magnanti, T. L., P. Mireault, R. T. Wong. 1986. Tailoring Benders decomposition for uncapacitated network design. *Math. Programming Study* **26** 112–154.

——, R. T. Wong. 1984. Network design and transportation planning: Models and algorithms. *Transp. Sci.* **18** 1–55.

Poljak, B. T. 1967. A general method of solving extremum problems. *Soviet Math. Doklady* **8** 593–597.

——. 1969. Minimization of unsmooth functionals. *USSR Computational Math. and Math. Physics* **9** 14–29.

Sen, S., H. D. Sherali. 1986. A class of convergent primal-dual subgradient algorithms for decomposable convex problems. *Math. Programming* **35** 279–297.

Shor, N. Z. 1968. On the rate of convergence of the generalized gradient method. *Kibernetika* **4** 98–99.

——. 1985. *Minimization Methods for Non-Differentiable Functions*. Springer-Verlag, Berlin.