Creating Advanced Bases For Large Scale Linear Programs Exploiting Embedded Network Structure

Nalân GÜLPINAR, Gautam MITRA*, István MAROS
Department of Computing,
Imperial College of Science, Technology and Medicine,
University of London, 180 Queen's Gate, London SW7 2BZ, UK.

*Department of Mathematical Sciences,
Brunel University, Uxbridge, Middlesex, London UB8 3PH, UK.

May 17, 2002

Abstract

In this paper, we investigate how an embedded pure network structure arising in many linear programming (LP) problems can be exploited to create improved sparse simplex solution algorithms. The original coefficient matrix is partitioned into network and non-network parts. For this partitioning, a decomposition technique can be applied. The embedded network flow problem can be solved to optimality using a fast network flow algorithm. We investigate two alternative decompositions namely, Lagrangean and Benders. In the Lagrangean approach, the optimal solution of a network flow problem and in Benders the combined solution of the master and the subproblem are used to compute good (near optimal and near feasible) solutions for a given LP problem. In both cases, we terminate the decomposition algorithms after a preset number of passes and active variables identified by this procedure are then used to create an advanced basis for the original LP problem. We present comparisons with unit basis and a well established crash procedure. We find that the computational results of applying these techniques to a selection of Netlib models are promising enough to encourage further research in this area.

1 Introduction

Large scale linear programming models which arise in many practical applications have sparse coefficient matrices and display alternative embedded structures such as Generalized Upper Bound (GUB), pure network and generalized network. It is well known that exploiting a special structure within a linear programming (LP) problem can lead to remarkable improvement

in the computational solution of LP problems; for instance see Brown and Olson [7]. When an LP model includes a subset of constraints and variables which together define a network flow problem [1], such a network structure appearing within the problem is called an *embedded network* and the remaining constraints are called *side constraints*. There is no restriction on the form of the rows and columns that do not possess a network property and prevent the LP problem from being directly solved by specialized network algorithms. For very large LP problems whose coefficient matrices are made up of a high proportion of rows and columns of the embedded network structure, the direct solution using the classical simplex method is generally expensive and may be impractical on a restricted computer platform. The idea of exploiting the network structure for solving linear programs has a growing importance in mathematical programming since network flow problems can be solved much faster using specialized network algorithms [35] rather than the state-of-the-art LP codes.

For Linear Programming problems with Embedded pure Network structures (LPEN), three classes of solution methods have been discussed in the literature. The first category is specialized simplex algorithms for solving embedded network linear programs. These methods are based on the partitioning of the basis. Glover and Klingman [20, 21] introduced a procedure called the Simplex Special Ordered Network (SON) algorithm. McBride [37] extended the SON procedure for solving embedded generalized network problems. Chen and Saigal also introduced a primal algorithm [11]. The second category consists of methods for solving network flow problems where the side constraints have a special structure. These are singly constrained network flow problems see [8, 19], multicommodity flow problems see [1] and linear relaxations of integer programming problems see [2]. Glover et al. [19] have reported that singly constrained transshipment network flow problems can be solved $25-30$ times faster than the state-of-the-art (at the time of publication of their paper) LP code APEX-III. The third category consists of methods which are based on a decomposition strategy which draws upon Lagrangean relaxation and surrogate constraint methods see [8, 9, 27, 15, 41]. Venkataraman [41] introduced a surrogate constraint approach to solve constrained network problems and compared his results with a subgradient optimization approach. Recently, Fourer and Hsu [15] have also investigated this approach which is based on the solution of the Lagrangean dual problem.

In this paper, our aim is to create an advanced basis for the LPEN problem by using two alternative decomposition methods. The rest of this paper is organized in the following way. In section 2, an embedded pure network flow problem is stated. In section 3, we present a broad algorithmic framework for creating an advanced basis and solving an LPEN problem. In section 4, we consider a Lagrangean relaxation of the LPEN problem and introduce a multiplier adjustment algorithm. In section 5, we consider a variation of the above approach in

which side constraints are aggregated prior to Lagrangean relaxation. A near optimal (and near feasible) solution yielded by the Lagrangean relaxation method is used to create an advanced basis for the original LP problem. Two procedures for creating such an advanced basis are described in section 6. In section 7, an alternative way of constructing such an advanced basis by using the Benders decomposition method is presented. The computational results are reported in section 8 and section 9 contains our conclusions in a summary form.

2 Problem Statement

A mathematical model of a network describes a system where the flow of some resource is conserved; the system is organized into a set of sites called nodes where a resource may be distributed or accumulated. The resource may be transferred within the system from one site to another following a set of directed arcs. A *pure network* is a network where for every arc, one unit of flow on the arc decreases the amount of resource at the origin node by one unit of resource and increases the amount of resource at the destination node by one unit. We consider the primal LP problem with simple upper bounds in the standard form:

$$\begin{aligned} &\text{Minimize } c^T x, \\ &\text{subject to} \\ &\qquad Ax = b, \\ &\qquad l \leq x \leq u, \end{aligned} \tag{2.1}$$

where $A \in R^{m \times n}$ and $c, x, l, u \in R^n$ and $b \in R^m$. An *embedded pure network structure* within an LP problem refers to a subset of rows of the coefficient matrix such that each column intersecting with these rows contains at most two non-zero entries (in these rows) and of opposite sign, that is at most one $+1$ and one -1 in each column. After such a subset has been identified, the LP problem can be interpreted as satisfying the conservation of flow at the nodes of a certain network defined by network rows and other conditions on flows as specified by the non-network rows. Assume that a submatrix of network rows and columns of A is detected by an appropriate embedded network detection method (for instance see [6, 24, 25]). Using the node-arc incidence matrix N to represent the pure network structure, the LP problem (2.1) can be restated in the decomposed form as,

P_0:

$$\begin{aligned} &\text{Minimize } z_0 = c'^T x' + c''^T x'', \\ &\text{subject to} \\ &\qquad Nx' = b', \\ &\qquad Sx' + Tx'' = b'', \\ &\qquad l' \leq x' \leq u', \\ &\qquad l'' \leq x'' \leq u''. \end{aligned} \tag{2.2}$$

In (2.2), subsets of rows and columns, submatrices and vectors are defined as

$m = m_1 + m_2,\ n = n_1 + n_2,$
$N = [n_{ij}] \in R^{m_1 \times n_1},\ S = [s_{ij}] \in R^{m_2 \times n_1},\ T = [t_{ij}] \in R^{m_2 \times n_2},$
$c', x', l', u' \in R^{n_1},\ c'', x'', l'', u'' \in R^{n_2},\ b' \in R^{m_1}$ and $b'' \in R^{m_2}$.

In many real life problems, l' and l'' are usually zero and inequalities $x' \leq u'$, $x'' \leq u''$ are turned into the equations $x' + s' = u'$, $x'' + s'' = u''$, where $s', s'' \geq 0$ and $s' \in R^{n_1}$, $s'' \in R^{n_2}$. The vector x' represents the network variables that have at least one non-zero element in the corresponding column of N. The second constraint set is referred to as side constraints and the vector x'' defines the non-network variables.

3 Solving the LPEN Problem with Advanced Basis

For large-scale LP problems, it is well known that using an advanced basis improves the performance of sparse simplex method [26], [32]. The calculation of the initial basis is of great importance as it determines to a large extent the amount of computation that is required to solve the problem to optimality. In this paper, our motivation is to construct an algorithm which exploits the embedded pure network structure in the LP problem to create an advanced basis. Once preprocessing and scaling procedures are applied, we extract the embedded network structure in the coefficient matrix of the LP problem (see [24, 25]). We then apply two different decomposition methods; Lagrangean relaxation and Benders decomposition to create an advanced starting point. The Lagrangean relaxation procedure creates a pure network flow model by adding the non-network constraints into the objective function with Lagrangean penalties. A series of minimum cost network flow problems are then solved iteratively by assigning trial values to the Lagrangean multipliers at each iteration. The Benders procedure decomposes the LP problem into a master and a subproblem. At each iteration, a cut obtained by solving the subproblem is introduced into the master problem and then solved again iteratively. The overall algorithmic framework is described below.

Solution of the LP with Embedded Pure Network Structure

Step 1 (Preprocessing and scaling)

Apply a preprocessing procedure to reduce the size of the problem and a scaling procedure to increase the number of essential rows and columns that have only ± 1 non-zero elements.

Step 2 (Network detection)

Detect the network structure out of the set of essential rows and columns. Decompose the problem into network and non-network structures as shown in (2.2).

Step 3 (Solve the decomposition problem and create a starting basis)

Either apply Lagrangean relaxation followed by the multiplier adjustment procedure (see section 4) and construct a triangular crash basis.

Or apply Benders decomposition (see section 7), then construct a starting basis by merging bases extracted from the solution of the master problem and the subproblem.

Step 4 (Complete sparse simplex solution)

Process the given LPEN problem applying the primal, dual or primal-dual simplex algorithm using the starting basis obtained above.

Having constructed a basis in the manner described in section 6, it is introduced as a starting basis to our experimental system FortMP [12] which is a general simplex solver. At this stage, the choice of the pivotal algorithms (primal, dual or primal-dual) plays an important role since one of these may be faster than the others for solving the given problem. Fourer and Hsu [15] suggested the dual-primal finishing strategy which avoids infeasibility thereby eliminating the need for Phase 1 in the two-phase simplex method. They claimed that this strategy is superior to the other simplex pivotal algorithms in terms of the number of iterations and CPU time. In our computational work, we have investigated the pivotal algorithms; primal, dual and primal-dual as sparse simplex completion strategies.

4 A Lagrangean Relaxation of the LPEN Problem

4.1 Lagrangean Relaxation and Lagrangean Dual Problem

From a computational point of view, decomposition is a well established approach for solving structured mathematical programming problems. In this approach, a large scale mathematical programming problem is decomposed into simpler problems where constraints are partitioned into two categories. Constraints in the first category are retained as binding constraints and constraints in the second category are removed (relaxed), grouped together as side constraints and penalized for violation. Relaxation of these side constraints makes the corresponding subproblem easier to solve than the original problem and this approach has become well known as Lagrangean relaxation. In the domain of combinatorial optimization, the Lagrangean relaxation method was first introduced by Held and Karp, who applied this technique to the travelling salesman problem [28]. Since then, this method has been widely used to solve other classes of constrained optimization problems [3, 13, 22, 23].

For the problem P_0, we group and relax side constraints $Sx' + Tx'' = b''$; weigh them us-

ing the Lagrangean multipliers λ. The problem is then restated as Lagrangean relaxation

$P_{L(\lambda)}$:

$$\text{Minimize } z_{L(\lambda)} = \lambda^T b'' + \left(c'^T - \lambda^T S\right) x' + \left(c''^T - \lambda^T T\right) x'',$$

subject to

$$\begin{aligned} & Nx' = b', \\ & l' \leq x' \leq u', \\ & l'' \leq x'' \leq u'', \\ & \lambda \in R^{m_2} \text{ and unrestricted.} \end{aligned} \tag{4.1}$$

In (4.1), the Lagrangean multipliers λ penalize the violation of the corresponding side constraints introduced in the objective function. It is easily seen that the relaxed problem $P_{L(\lambda)}$ is a pure network flow problem x' in which the feasibility of x'' can be trivially satisfied, whereas P_0 is a general LP problem. This network flow problem is solved efficiently by special algorithms such as the network simplex algorithm.

The Lagrangean function $z_{L(\lambda)}$ is convex, piecewise-linear and continuous; these important structural properties make the Lagrangean problem easy to solve. However, the Lagrangean function is not everywhere differentiable. It is differentiable whenever the optimal solution of the Lagrangean subproblem is unique. It is, however, subdifferentiable everywhere in the convex hull of the problem. Let $(\pi, \lambda, \sigma_1, \eta_1, \sigma_2, \eta_2)$ be dual variables corresponding to constraints in (2.2). The dual of the original LP problem P_0 is then formalized as

D_0 :

$$\text{Maximize } b'^T \pi + b''^T \lambda + l'^T \sigma_1 + l''^T \sigma_2 - u'^T \eta_1 - u''^T \eta_2,$$

subject to

$$\begin{aligned} & N^T \pi + S^T \lambda + \sigma_1 - \eta_1 = c', \\ & T^T \lambda + \sigma_2 - \eta_2 = c'', \\ & \sigma_1, \sigma_2, \eta_1, \eta_2 \geq 0. \end{aligned} \tag{4.2}$$

The dual problem of $P_{L(\lambda)}$ is the same as D_0 with different right hand side values. The Lagrangean dual problem of P_0 with respect to the side constraints is to find the set of Lagrangean multipliers λ^* that maximizes the Lagrangean function $z_{L(\lambda)}$. The objective function of Lagrangean dual problem is as follows

P_D:

$$z^*_{L(\lambda^*)} = \max_\lambda \left\{ \min_{(x', x'')} \left(c' - \lambda^T S\right) x' + \left(c'' - \lambda^T T\right) x'' \right\}, \tag{4.3}$$

where the Lagrangean multipliers λ are computed by solving the LP problem

$$
\begin{aligned}
z^*_{L(\lambda^*)} \;=\; & \text{Maximize } w, \\
& \text{subject to} \\
& \quad w \le f_j + \lambda^T g^j; \quad j = 1, \cdots, K.
\end{aligned} \tag{4.4}
$$

In (4.4), f_j is the objective value of P_0 and g^j is the subgradient of the jth basic solution

$$f_j = c'^T x'^j + c''^T x''^j,\; g^j = b'' - Sx'^j - Tx''^j \tag{4.5}$$

and K denotes the number of all basic solutions of the Lagrangean problem.

We may consider the Lagrangean problem $P_{L(\lambda)}$ and its relationship with the original problem P_0. For any vector λ of the Lagrangean multipliers, the optimum value $z^*_{L(\lambda)}$ of the Lagrangean function is a lower bound on the optimal objective function value z^*_0 of the original primal optimization problem P_0, that is, $z^*_{L(\lambda)} \le z^*_0$ for all λ. There exists a set of Lagrangean multipliers λ^* for which the objective value of the Lagrangean relaxation $z^*_{L(\lambda^*)}$ attains the optimal value of the original problem P_0, that is, $z^*_{L(\lambda^*)} = z^*_0$. It is worthwhile to note that when applying the Lagrangean relaxation method to linear programs, the first property has long been known, but Geoffrion [18] observed that the latter property does not hold for integer programs in general. For some choice of the Lagrangean multiplier vector λ, if the solution of the Lagrangean relaxation (x', x'') is feasible in the optimization problem and satisfies the complementary slackness conditions involving non-negative primal $\{x', x'', s', s''\}$, and non-negative dual $\{\sigma_1, \sigma_2, \eta_1, \eta_2\}$ variables with the property $x'\sigma_1 = x''\sigma_2 = s'\eta_1 = s''\sigma_2 = 0$, then (x', x'') is an optimal solution to P_0.

Fourer and Hsu [15] used a master problem given in (4.4) with a trust region constraint to solve the Lagrangean dual problem in (4.3). In our procedure, we solve $P_{L(\lambda)}$ only by a multiplier adjustment approach which finds a good (near optimal) solution for the original LP problem as it progressively improves the lower bound. The multiplier adjustment method is a specialized procedure that solves the Lagrangean dual problem by exploiting the structure of a particular model. It is sometimes called Lagrangean dual ascent as it can be viewed as an ascent procedure which guarantees a monotone bound improvement. Fisher et al. [14] and Guignard et al. [22] gave applications of this method to an assignment problem and an allocation problem.

Developing a multiplier adjustment procedure is considered to be an art; different problems require different multiplier adjustment algorithms. It is an iterative method and starts with an initialized set of Lagrangean multipliers. The initialization of multipliers is dependent on the underlying model structure and affects the quality of the final bound. At each iteration, only a small subset of multipliers is examined and one or more multipliers of violated constraints is adjusted. The calculation of the multiplier adjustment amount is also problem specific. At iteration k of the multiplier adjustment method, a set of ascent directions is determined

such that the effect on the optimum value of the Lagrangean dual problem by a movement along a direction is evaluated. The common improvement on multipliers is made by the rule; $\lambda^{k+1} = \lambda^k + t_k g^k$, where g^k is an ascent direction and t_k is the step size. The step size, t_k, can be chosen either to maximize $z_{L(\lambda^k + t_k g^k)}$ or to take to the first point at which the directional derivative changes.

The ascent direction involves changes to multipliers corresponding to violated constraints which is made generally with the following rules:

- if ${g_i}^k < 0$, then reduce the multiplier ${\lambda_i}^k$
- if ${g_i}^k = 0$, then do not change the multiplier ${\lambda_i}^k$
- if ${g_i}^k > 0$, then increase the multiplier ${\lambda_i}^k$.

The determination of the set of directions and order in which directions are scanned are problem specific and affect the final bound. If the set of ascent directions is empty, in other words all constraints are satisfied, or the set has no improving direction, the procedure is terminated even though an optimal solution is not found.

4.2 Solving the Lagrangean Relaxation (Network) Problem

Consider the problem $P_{L(\lambda)}$ in (4.1). It actually consists of two subproblems; network and non-network. Thus, solving the kth Lagrangean relaxation problem involves a network part which provides the solution for network variables x' and a trivial non-network subproblem which finds the solution for non-network variables x''.

The non-network subproblem is solved simply by setting the side variables to bounds in their feasibility ranges according to their reduced costs as

$$\begin{aligned} ({x_j}'')^k &= ({l_j}'')^k \quad if \quad {c_j}'' - \left(\lambda^k\right)^T T_j \geq 0, \\ ({x_j}'')^k &= ({u_j}'')^k \quad if \quad {c_j}'' - \left(\lambda^k\right)^T T_j < 0, \end{aligned} \tag{4.6}$$

where T_j denotes the jth column of the matrix T. The network subproblem can be solved by the primal or dual network simplex algorithm. For our computational work, we use MINET a minimum cost network flow solver developed by Maros [31, 35].

The multiplier adjustment heuristic solves iteratively a sequence of network linear programs with different values of λ. Even though the solution $(x', x'')^k$ is an optimal solution for the Lagrangean relaxation problem at the kth iteration, it is not guaranteed that it is a feasible solution of the original LP problem. To improve the current lower bound, the algorithm finds another direction and updates the multipliers. The procedure is stated below.

Multiplier Adjustment Algorithm

Step 1 (Initialization)

Let $k = 0$, assign Lagrangean multipliers an initial value, say $\lambda^0 = 0$.

Step 2 (Solve subproblems)

Solve the network subproblem and find the network flows $(x')^k$. Apply rule (4.6) to bound restrictions on the non-network variables to determine their solution values. If there does not exist a feasible solution for the network subproblem, then terminate the algorithm and conclude that the original LP problem has no feasible solution.

Step 3 (Compute gradients)

Calculate gradients for each side constraint. Check gradients; if all side constraints are satisfied, that is ${g_i}^k = 0$, then stop the algorithm and conclude that a feasible solution to the original problem has been found.

Step 4 (Calculate the adjustment)

Compute $\Delta\lambda = \min\left\{c'^k - \left(\lambda^k\right)^T S_j,\ c''^k - \left(\lambda^k\right)^T T_j\right\}$.

Step 5 (Update the Lagrangean multipliers)

Choose a set of violated constraints. Update the Lagrangean multipliers ${\lambda_i}^k$ of the violated constraints as ${\lambda_i}^{k+1} = {\lambda_i}^k + \Delta\lambda$.

Step 6 (Update the problem)

Let $k = k + 1$, construct another minimum cost network flow problem with new multipliers and go to step 2.

It is worthwhile to mention here that, for $k = 0$, the problem represents the network components of the embedded network LP problem since the side constraints are all ignored ($\lambda = 0$). Hence, the objective value of the network subproblem and the objective value of the LP problem P_0 are the same. Fourer and Hsu [15] called this the zero-multiplier method.

5 Aggregation of the Side Constraints

In mathematical programming, aggregation techniques consist of a set of methods for solving optimization problems by combining data, using an auxiliary model which is reduced in size and complexity relative to the original model. They have been developed to help form the most appropriate reduced models that provide good approximations to the original problem [40].

In order to perform a row or a column aggregation, a set of constraints or variables are replaced with a single row or a single column. If a set of rows is multiplied by different weights and aggregated to a single constraint, this is known as *a weighted aggregation.* If a row is selected such that it dominates a set of rows, the choice of this row is called *an aggregation by dominance.* The same terminology applies to columns.

In this section, we consider another approach which quickly finds a near optimal solution of the LPEN problem by aggregating the side constraints. Instead of solving the problem with the original side constraints, the Lagrangean relaxation problem with respect to the aggregated side constraints is used to find a starting basis. The weight vector consisting of only ones is used to aggregate the non-network constraints. We apply a basic heuristic to aggregate side constraints in the given embedded network problem in (2.2). Variables which appear in a given side constraint i are either network variables or non-network variables and can be classified into two subsets as $I_i(N) = \{j|\ a_{ij} \neq 0,\ j \text{ is a network variable}\}$ and $I_i(\bar{N}) = \{j|\ a_{ij} \neq 0,\ j \text{ is a non-network variable}\}$.

The side constraints are aggregated into three groups using the following criteria;

- the group of rows which have only network columns, that is $I_i(\bar{N}) = \emptyset$. Let this be defined as the subset R_1 of the row indices i,
- the group of rows which have only non-network columns, that is $I_i(N) = \emptyset$. Let this be defined as the subset R_2 of the row indices i,
- the group of rows which have both network and non-network columns, that is $I_i(\bar{N}) \neq \emptyset$ and $I_i(N) \neq \emptyset$. Let this be defined as the subset R_3 of row indices i.

The aggregated embedded LP problem becomes a minimum cost network flow problem with only three side constraints and is set out below.

AP_0 :

Minimize $z_0 = {c'}^T x' + {c''}^T x''$

subject to

$$Nx' = b',$$

and the following aggregated constraints

R_1:

$$\sum_{i \in R_1} \left(\sum_{j \in I_i(N)} s_{ij} \right) {x_j}' = \sum_{i \in R_1} {b_i}'',$$

In vector notation, this can be written as $S_1 x' = \beta_1$, where

$$S_1 = \left[\sum_{i \in R_1} s_{i1}, \cdots, \sum_{i \in R_1} s_{in_1} \right] \text{ and } \beta_1 = \sum_{i \in R_1} b_i''.$$

R_2:

$$\sum_{i \in R_2} \left(\sum_{j \in I_i(\bar{N})} t_{ij} \right) x_j'' = \sum_{i \in R_2} b_i'',$$

In vector notation, this can be written as $T_2 x'' = \beta_2$, where

$$T_2 = \left[\sum_{i \in R_2} t_{i1}, \cdots, \sum_{i \in R_2} t_{in_2} \right] \text{ and } \beta_2 = \sum_{i \in R_2} b_i''.$$

R_3:

$$\sum_{i \in R_3} \left\{ \sum_{j \in I_i(N) \cup I_i(\bar{N})} (s_{ij} x_j' + t_{ij} x_j'') \right\} = \sum_{i \in R_3} b_i'',$$

In vector notation, this can be written as $S_3 x' + T_3 x'' = \beta_3$, where

$$S_3 = \left[\sum_{i \in R_3} s_{i1}, \cdots, \sum_{i \in R_3} s_{in_1} \right], \; T_3 = \left[\sum_{i \in R_3} t_{i1}, \cdots, \sum_{i \in R_3} t_{in_2} \right] \text{ and } \beta_3 = \sum_{i \in R_3} b_i''.$$

$$l' \leq x' \leq u',$$

$$l'' \leq x'' \leq u''.$$

The multiplier adjustment procedure explained in section 4.2 can now be applied to solve the following Lagrangean relaxation of the aggregated embedded network flow problem in which there are only three multipliers to be adjusted.

$AP_{L(\lambda)}$:

Minimize $z_{L(\lambda)} = \lambda_1 \beta_1 + \lambda_2 \beta_2 + \lambda_3 \beta_3 + c'^T x' + c''^T x'' - \lambda_1 S_1 x' - \lambda_2 T_2 x'' - \lambda_3 (S_3 x' + T_3 x'')$,

subject to

$$Nx' = b',$$

$$l' \leq x' \leq u',$$

$$l'' \leq x'' \leq u'',$$

and vector $\lambda^T = \{\lambda_1, \lambda_2, \lambda_3\}$ is unrestricted.

6 Creating An Advanced Basis

6.1 Discussion of Crash Procedures

The role of an advanced basis and the framework for solving LPEN problems have been introduced in section 3. In this section, we review some well known approaches for creating

advanced bases. In general, crash procedures are designed to create an initial basis to provide an advanced starting point [10, 32]. The simplest way to find an initial basis is to enlarge the problem by adding artificial and slack variables and create a basis which is made up of these logical variables. This leads to well known all-logical (unit) basis. The objective at this stage is to drive artificial variables to zero (make as many as possible non-basic) in order to obtain a feasible solution to the original problem. It is well known that a starting basis with multiple structural variables needs fewer iterations and less time to find an optimal solution compared to the all-logical basis. Alternative heuristic procedures for creating an advanced basis have been described in the literature (see for instance Carstens [10], Bixby [5], Maros and Mitra [32] and Gould and Reid [26]).

The triangular crash procedures find a basis matrix which has as many structural variables as possible in such a way that the resulting basis matrix has a triangular form with a zero free diagonal. There are two types of triangular bases which can be extracted from the LP constraint matrix. The first one is the lower triangular basis in which non-zeros are located in and below the main diagonal. The other one is the upper triangular basis in which non-zeros are located in and above the main diagonal. If the matrix found by the triangular crash procedures does not satisfy the full row rank property, then it is usually augmented by logical variables as necessary to form a non-singular triangular matrix that can be used as a starting basis for the simplex algorithm. The triangularity of this basis matrix ensures that a factored inverse representation of the basis with a minimum number of non-zeros can be trivially created.

The triangular crash procedure was first introduced by Carstens in [10]. He defined the improvement in the objective value as GAIN and introduced some heuristics for choosing the possible pivots which lead to improvement in GAIN. The block triangular crash procedure was first introduced by Gould and Reid in [26]. Maros and Mitra introduced a few crash heuristics which have been used in the solver FortMP [32, 33, 34]. Our network based crash procedure uses one of them, namely, Lower Triangular Symbolic Crash designed for Feasibility (*CLTSF*), but customizes it by restricting the choice of basic columns to the optimum network variables.

6.2 Network Based Crash Procedure: CNET1, CNET2

In this section, we consider the embedded network LP problem given in (2.2) and describe our network based crash procedure for creating an advanced basis. For a given set of trial values of Lagrangean multipliers, the Lagrangean relaxation problem is itself a minimum cost network flow problem. Therefore, the basis corresponding to a feasible or an optimal solution to this problem has a triangular form because of the natural structure of network flow problems. This leads us to use a lower triangular crash procedure by considering only variables

which are basic in the network optimal solution. As there are generally less basic variables in the optimal solution of network problems than the basis size of the original problem, the logical variables associated with side constraints are introduced to gain the full row rank and to construct a non-singular basis for the original problem.

We construct two crash procedures; in the first one our choice is restricted to only network variables and network rows. The starting basis is initialized as the one which consists of the basic variables of the network problem and the logical variables of the non-network rows. In other words, all non-network columns and network columns which are not in the optimal basis of the network problem are excluded. We call this crash procedure CNET1. In the second method, we take into account the remaining rows and apply the CLTSF procedure to the non-network rows that are not processed in the CNET1 procedure. Therefore, as many logical variables corresponding to non-network rows as possible are replaced by structural variables. We call this procedure CNET2. The main steps of the crash CNET2 procedure are set out below.

Network Based Crash Algorithm: CNET2

Step 1 Initialize the starting basis for the original problem. It contains all basic variables of the optimal solution of the network problem and the logical variables of the non-network rows.

Step 2 Define the set of active rows as all non-network rows and the set of active columns as all nonbasic columns of the optimal network solution. Exclude the free rows and fixed columns.

Step 3 Calculate the row and column counts of non-zero entries for active rows and columns of the coefficient matrix.

Step 4 Make the row selection on the basis of minimum row count. If there is a tie, then break it as in [32]. If the row selection is successful, then select the pivot column, based on minimum column count. In case of a tie, break it by [32]. If there is no row to select, then terminate the algorithm with the current basis that may contain some logical variables of the non-network rows.

Step 5 Update the basis by exchanging the basic column corresponding to the pivot row with the selected pivot column.

Step 6 Delete the selected row and column from the active sets and also delete any other active columns intersecting with the selected row.

Step 7 Update the row and column counts and go to Step 4 to select the next pivot row.

7 Benders Decomposition of the LPEN Problem

7.1 Theoretical Framework

This decomposition was introduced by Benders [4] to solve mixed integer programming problems. Since then, it has been applied to many large scale problems in mathematical programming, especially, for solving two stage as well as multistage stochastic programming problems. The embedded network flow problem may be considered as a two stage problem in which the network part is the first stage and the non-network part is the second stage. This has motivated us to use Benders decomposition to create an advanced basis for solving the original LPEN problem. In this section, we describe our algorithm based on the Benders decomposition procedure. We consider the embedded pure network problem P_0 given in (2.2) and split the original problem into a master P_{master} and a subproblem P_{sub}. The latter is used to generate cuts as in the Benders decomposition method. Initially, the P_{master} problem is a pure network problem stated as

P_{master}:

Minimize $z_M = c'^T x'$,

subject to

$Nx' = b'$,

$l' \leq x' \leq u'$.

Let x'^* denote an optimal solution of P_{master}, then the subproblem P_{sub} is defined as

P_{sub}:

Minimize $z_S = c''^T x''$,

subject to

$Tx'' = b'' - Sx'^*$,

$l'' \leq x'' \leq u''$.

The corresponding dual linear program D_{sub} of the subproblem P_{sub} is stated using dual variables $\pi^T = (\pi_1, \pi_2, \pi_3)^T$ as

D_{sub}:

Maximize ${\pi_1}^T(b'' - Sx'^*) - {\pi_2}^T u'' + {\pi_3}^T l''$,

subject to

${\pi_1}^T T \leq c''$,

π_1 free,

$\pi_2, \pi_3 \geq 0$.

The feasibility condition of D_{sub} is ${\pi_1}^T T - c'' \leq 0$ and $\pi_2, \pi_3 \geq 0$. The assumption that P_0 is feasible requires the feasibility of P_{sub} for all values of x' satisfying $l' \leq x' \leq u'$ and $Nx' = b'$. In addition, from duality theory, problem D_{sub} is finite if and only if

${\pi_1}^T(b'' - Sx'^*) - {\pi_2}^T u'' + {\pi_3}^T l'' \leq 0$. This constraint can be appended to the first master problem to ensure that the solution to the new master problem leads to a feasible solution of the original problem; this constraint is called a *feasibility cut.*

By considering an upper bound on the objective function of D_{sub}, the smallest value of this upper bound is denoted by θ and used to formulate the master problem in the following way.

$$\begin{aligned}
&\text{Minimize } z_M = {c'}^T x' + \theta, \\
&\text{subject to} \\
&\qquad Nx' = b' \\
&\qquad \theta - ({\pi_1}^*)^T(b'' - Sx') + ({\pi_2}^*)^T u'' - ({\pi_3}^*)^T l'' \geq 0, \qquad (7.1) \\
&\qquad ({\pi_1}^*)^T(b'' - Sx') - ({\pi_2}^*)^T u'' + ({\pi_3}^*)^T l'' \leq 0, \qquad (7.2) \\
&\qquad l' \leq x' \leq u', \\
&\qquad \theta \text{ free},
\end{aligned}$$

where $\pi^* = \{{\pi_1}^*, {\pi_2}^*, {\pi_3}^*\}$ is the optimal solution of D_{sub}. At each iteration of the decomposition procedure, either an optimality cut (7.1) or a feasibility cut (7.2) is added to the master problem; if primal infeasibility (dual unboundness) is found for the subproblem, the feasibility cut is added, if the primal problem is feasible (dual bounded) the optimality cut is then appended to the master problem. Each optimal solution of the master problem (x'^*, θ^*) is suboptimal and gives a lower bound on the objective value of the LPEN problem, that is $LB = {c'}^T x'^* + \theta^* \leq z_0^*$. When the master and subproblem are both feasible, the solution (x'^*, x''^*) is a feasible solution for the problem (2.2) and creates an upper bound, that is $UB = {c'}^T x'^* + {c''}^T x''^* \geq z_0^*$. At each pass of the decomposition, the lower bound is updated, at the kth pass the lower bound is

$$LB^k = {c'}^T \left(x'^*\right)^k + \theta^{*k}. \qquad (7.3)$$

Initially, the upper bound is set to infinity. If the subproblem at iteration k is solved to optimality, then a new upper bound may be found by the relation

$$UB^k = \min\left\{UB^{k-1}, {c'}^T \left(x'^*\right)^k + {c''}^T \left(x''^*\right)^k\right\}. \qquad (7.4)$$

When the relative gap satisfies the following property, then the problem is considered to have been solved with sufficient accuracy [29],

$$\frac{UB^k - LB^k}{|LB^k| + 1} \leq TOL, \qquad (7.5)$$

and the algorithm is terminated. We use $TOL = 10^{-6}$ in our computational experiments.

7.2 Algorithm

It is well known that Benders decomposition converges to an optimal solution in a finite number of iterations. Instead of solving the entire embedded network flow problem with Benders decomposition, our aim is to take advantage of a good intermediate solution obtained by the decomposition algorithm and create an advanced starting point. Our procedure is set out below.

Step 1 (Construct a master and a subproblem)

Decompose the LPEN problem into a master and a subproblem.
Initialize the maximum number of pass, $MAXP$.
Repeat for $k = 0, \ldots,$ $MAXP$
{ *Step 2 (Solve the master problem)*

Solve the master problem by a simplex solver. (When $k = 0$, use a network solver). If the solution is infeasible, then conclude that the entire LP problem is infeasible and go to step 7.

Step 3 (Construct a new subproblem and solve)

By fixing the solution x'^* of the master problem and revising the right hand side, construct P_{sub} and then solve this subproblem by the primal, dual or primal-dual simplex algorithm.

Step 4 (Obtain the lower bound)

Calculate the lower bound using relation (7.3).

Step 5 (Create a cut)

If an optimal solution is found for the subproblem, then
calculate the upper bound using relation (7.4),
check the optimality conditions shown in (7.5).
If the optimality condition is satisfied, then
go to step 7.
Else
create an optimality cut (7.1).
Endif

Elseif the subproblem does not have a feasible solution, then
create a feasibility cut (7.2).
Endif

Step 6 (Update the problem)

Add this cut to the master problem. }

Step 7 (Terminate the algorithm)

7.3 Constructing an Advanced Basis

Having applied Benders decomposition with a preset number of passes, we use the combined solution of the master (the embedded network flow problem) and the subproblem to compute a good (near optimal and near feasible) solution for the given LP problem. Let $x^k = \left((x')^k, (x'')^k\right)$ denote the solution vector of the master and the subproblem in the kth pass. This solution may be

- an infeasible,
- a feasible or
- a feasible as well as optimal

solution of the LPEN problem. We create a starting basis for the original problem in the following way. If the variable ${x_i}^k$ for the ith component appears as a basic variable in the solution of the master or the subproblem, then we mark its status as basic. If not, it means that the variable is non-basic, then we analyse the solution values;

Lower bound: If the solution value ${x_i}^k$ for the ith component is at its lower bound, that is, ${x_i}^k = l_i$, then we set its status to non-basic at lower bound.

Upper bound: If the solution value ${x_i}^k$ for the ith component is at its upper bound, that is, ${x_i}^k = u_i$, then we set its status to non-basic at upper bound.

The basis factorization procedure INVERT uses this information to create an initial factorization of this basis as a simplex starting point for solving the LPEN problem.

8 Computational Results

8.1 The Collection of Test Problems

We have investigated the computational performance of our algorithms for a number of industrial benchmark problems which are readily available to the scientific community. We have, therefore, chosen models from `Netlib\lp\data` [16] and `Kennington` library which is also a part of Netlib as well as a collection of multicommodity flow problems [17]. The characteristics of the test problems in terms of the number of constraints, variables and non-zero entries are summarized in Table 1.

We have detected the embedded pure network structure by using a multi-stage GUB based algorithm developed by us and described in [24, 25]. The number of network rows detected by this algorithm and the corresponding network columns are also displayed in Table 1.

MODEL NAME	ROWS	COLUMNS	NONZEROS	NETWORK ROWS	NETWORK COLUMNS
bnl2	2325	3489	16124	595	1826c
cre-a	3517	4067	19054	579	3446
cre-b	9649	72447	328542	675	26033
cre-c	3069	3678	16922	546	2881
cre-d	8927	69980	312626	608	19888
cycle	1904	2857	21322	178	1377
degen3	1503	1818	24646	579	1654
d2q06c	2172	5167	21322	376	1662
energy	2236	9799	29063	1009	6017
greenbea	2392	5405	31499	235	1404
ken7	2426	3602	11981	788	2258
pilot	1442	3652	43220	119	463
pilot87	2030	4883	73152	82	346
sctap3	1480	2480	10734	620	1860
ship12l	1151	5427	21597	490	3501
sierra	1227	2036	9252	618	1945
stocfor2	2157	2031	9492	780	1560
stocfor3	16676	15695	74004	6072	12144
pds-1	1473	3729	8052	876	3322
pds-2	2953	7535	16390	1227	5631
pds-3	4593	12287	26796	2141	10872
pds-4	6372	18194	39523	2728	14465
pds-5	8099	23639	51425	3660	20969

Table 1: The characteristics of test problems and the number of network rows and network columns.

8.2 Experimental Results

We have developed a FORTRAN implementation of the algorithm described in section 3 and the alternative advanced bases. FortMP [12] is used as a callable subroutine. FortMP has been developed by the mathematical programming group at Brunel University. It is an industrial strength mathematical programming system used for both algorithmic research and in collaborative projects with industry. The computational experiments have been carried out on a SUN Sparc 10 computer with 256MB memory. We have used the CPU time and simplex iterations as alternative performance measures of our procedures. In the network exploitation procedures, the total solution time is calculated by including the time spent in

1. the network extraction including scaling,
2. the Lagrangean multiplier adjustment procedure (solving a series of network flow problems) or Benders decomposition procedure (solving a series of master and subproblems),
3. the basis construction, and
4. solving the entire LP problem by FortMP.

However, the time to input the original LP problem and pre-processing time, which are common to all runs, are excluded. We present our results for each decomposition method separately and discuss consolidated results.

Results with Lagrangean Relaxation

The results obtained by the Lagrangean relaxation method are set out in Table 2. We first apply the multiplier adjustment method to the Lagrangean relaxation of the LPEN problem and display results under the heading *LR: Adjusted Multiplier*. We then consider an alternative relaxed problem in which all non-network side constraints are aggregated. We apply the same multiplier adjustment method to solve the corresponding network flow problem with at most three side constraints and the results are displayed under the heading *LR: Row Aggregation*. We also consider the zero-multiplier method mentioned in section 4 which is a special case of the multiplier adjustment procedure where all multipliers are fixed to zero and display results in the second column, *LR: Zero Multiplier*.

In these cases, the advanced bases are constructed by applying two network based crash procedures *CNET1, CNET2* discussed in section 6. The test problems are then solved using these starting bases by applying primal, dual or primal-dual sparse simplex as a finishing strategy. The results in Table 2 are chosen as the best out of all alternative advanced bases and different finishing strategies. The maximum number of passes to solve the Lagrangean relaxation problem is limited to 50 in both cases of applying the multiplier adjustment procedure. We observe that for some LP models, the solution obtained using the basis constructed after optimizing only the network problem without making any multiplier adjustment (zero multiplier case) decreases the number of iterations as well as the CPU time comparing with multiplier adjustment method, for example see `energy, stocfor2` and `stocfor3`. However, for other cases, for instance models `cre-d, d2q06c, pilot87`, and `cycle, degen3, pilot, pds1-5`, LR: Adjusted Multiplier and LR: Row Aggregation methods lead to better performance.

The results in Table 2 reveal that an LP with an advanced starting point constructed from the solution of Lagrangean relaxation of the aggregated embedded network problem is solved

to optimality with fewer iterations and less time than the one from the solution of the original embedded network problem in most models; for example, see models `degen3, pilot` and `pds1-5`. Since the multiplier adjustment procedure is carried out unless the side constraints are satisfied or the maximum iteration number is reached, the total solution time is increased in the Lagrangean relaxation procedure.

	LR: ZERO MULTIPLIER		LR: ADJUSTED MULTIPLIER		LR: ROW AGGREGATION	
MODELS	TIME	ITER	TIME	ITER	TIME	ITER
bnl2	38.84	2810	*37.16	2731	37.77	2810
cre-a	*34.65	2624	35.15	2624	35.16	2624
cre-b	1063.95	23987	*1062.76	23900	1140.6	21346
cre-c	28.32	2375	28.15	2375	*27.64	2375
cre-d	1215.96	19702	*646.15	16552	654.95	13577
cycle	14.47	1517	15.04	1517	*11.06	1221
degen3	98.82	4904	104.61	5176	*93.69	4706
d2q06c	443.67	20118	*429.69	19443	437.47	20118
energy	*85.32	9295	93.52	9640	88.8	9295
greenbea	103.8	7076	85.72	5494	*84.66	5494
ken7	12.03	1890	12.44	1927	*10.84	1591
pilot	506.01	9562	509.27	9562	*489.96	9425
pilot87	2301.03	14135	*2296.41	14135	2480.01	15720
sctap3	2.61	372	*2.55	372	3.11	372
ship12l	4.82	822	5.75	1075	*3.57	726
sierra	3.13	644	*3.07	619	3.41	644
stocfor2	*8.12	674	14.74	1281	9.84	816
stocfor3	*790.19	7232	1044.07	10322	797.69	7232
pds-1	2.92	361	3.57	453	*1.24	449
pds-2	19.81	1168	10.62	1168	*7.41	1181
pds-3	46.05	1716	45.65	1716	*16.65	1698
pds-4	134.43	3348	133.38	3348	*52.86	3348
pds-5	219.59	4204	216.83	4208	*86.20	4208

TIME: CPU time in seconds, **ITER**: The number of simplex iterations to solve the LP with an advanced basis, *: The best solution time obtained out of all procedures.

Table 2: Results with the Lagrangean relaxation method.

It is worthwhile to mention that even though increasing the number of multiplier adjustments might give a better bound, it is still time consuming. In contrast, the aggregated side constraint can be satisfied without reaching the preset limit on the number of passes. In some cases, however, considerable computational time is required to satisfy the aggregate side constraint, for example see models `cre-b` and `pilot87`.

Results with Benders Decomposition

In Table 3, we present the results of applying Benders decomposition described in section 7.2 to test problems. Since at each pass a master and a subproblem are solved and repeated passes can be time consuming for large models, we have preset the maximum number of passes to one. Therefore, only one cut is introduced into the master problem. In Table 3, we break up the results for total solution time and total iteration number for the master and subproblems as well as the simplex finishing strategy.

In the last column in Table 3, the total solution time includes the time taken to solve the master and subproblems, to create an advanced basis and to solve the original LP problem as well as the time taken to extract the network structure. Considering the results shown in Table 3, we find that the time spent in solving two master problems does not take the major proportion of the total solution time. However, for model `stocfor3`, even though the network simplex solver can be used for the first pass of the master problem (the pure network flow problem), solving the master problem with one cut is relatively time consuming.

Time to solve subproblems depends on the size of network structures detected; if the proportion of the network structure is not large, it means that the subproblem is relatively large and cannot be solved fast. The results for models `pilot` and `pilot87` support this observation. Since the subproblem is the same as the previous one with only different right hand side values, the previous basis of the subproblem for a warm start is used.

We also observe that an advanced starting point can always be constructed from the solution of the entire master (pure network flow problem) and subproblem as in the zero multiplier method. However, we wish to test whether the advantage of adding the cut can be established here. Therefore we present the results of restricted Benders decomposition.

	MASTER PROBLEM		SUB PROBLEM		SIMPLEX BEND BASIS		TOTAL SOLUTION
MODELS	TIME	ITER	TIME	ITER	TIME	ITER	TIME
bnl2	0.90	337	0.05	197	55.77	3407	58.23
cre-a	0.48	201	0.50	40	37.56	3108	39.85
cre-b	5.28	430	3.81	72	934.44	21977	970.46
cre-c	0.42	307	0.36	44	33.80	3401	35.83
cre-d	4.93	541	3.15	56	695.69	18475	726.20
cycle	0.01	2	0.03	5	8.81	477	10.15
degen3	0.83	590	0.03	5	83.36	4063	89.37
d2q06c	0.59	316	10.49	1213	481.95	20533	496.07
energy	6.31	1399	1.17	269	85.91	7794	96.84
greenbea	0.54	39	0.65	61	109.35	6922	111.00
ken7	3.34	861	0.07	47	9.62	1404	14.86
pilot	0.39	30	31.68	1804	331.22	6322	364.18
pilot87	0.77	35	254.45	3020	2011.73	10627	2268.15
sctap3	0.76	574	0.16	48	11.18	1928	12.98
ship12l	0.76	299	0.11	13	3.68	764	5.46
sierra	1.18	612	0.02	0	3.35	960	5.44
stocfor2	1.63	685	0.14	3	19.16	2047	21.93
stocfor3	101.50	5389	1.79	15	1676.65	17416	1833.75
pds-1	1.43	533	0.01	0	1.14	209	3.72
pds-2	6.34	1053	0.03	0	9.68	926	20.50
pds-3	20.59	1944	0.05	0	34.80	2127	66.21
pds-4	47.33	2884	0.07	0	110.79	4306	181.06
pds-5	70.94	3271	0.11	0	153.21	4562	261.52

TIME: CPU time in seconds, **ITER**: The number of simplex iterations to solve the master, the subproblem, and the original LP problem.

Table 3: Results with Benders decomposition with one cut.

Consolidated Results

We compare our network based crash procedures with the unit basis and *CLTSF* procedure which has one of the best performances currently reported in the literature, see [32, 34]. We therefore set out the consolidated results in Table 4 and display the time and the iteration number to solve the LPEN problem with the advanced basis chosen as the best performance of the Lagrangean relaxation and Benders decomposition, the crash *CLTSF* and the unit basis.

	LAGRANGEAN RELAXATION		BENDERS DECOMPOSITION		CRASH CLTSF		UNIT BASIS	
MODELS	TIME	ITER	TIME	ITER	TIME	ITER	TIME	ITER
bnl2	*37.16	2731	58.23	3407	57.63	3447	74.42	5409
cre-a	*34.65	2624	39.85	3108	51.98	3073	56.38	4514
cre-b	1062.76	23900	*970.46	21977	1124.0	21346	1153.62	23987
cre-c	*27.64	2375	35.83	3401	43.71	3151	44.88	4256
cre-d	*646.15	16552	726.20	18475	1181.74	19702	690.47	16552
cycle	11.06	1221	*10.15	477	14.57	1149	11.78	1316
degen3	93.69	4706	*89.37	4063	118.66	6724	170.90	10270
d2q06c	*429.69	19443	496.07	20533	463.60	20504	509.57	22952
energy	*85.32	9295	96.84	7794	88.80	9295	87.03	9902
greenbea	*84.66	5494	111.0	6922	85.79	5162	116.68	7536
ken7	10.84	1591	14.86	1404	*7.58	1207	16.89	2790
pilot	489.96	9425	*364.18	6322	505.91	9411	486.79	9220
pilot87	2296.41	14135	*2268.15	10627	2732.42	19894	2588.02	15225
sctap3	*2.55	372	12.98	1928	3.08	761	5.50	1370
ship12l	3.57	726	5.46	764	*3.11	726	5.66	1075
sierra	*3.07	619	5.44	960	5.99	1363	4.79	1248
stocfor2	*8.12	674	21.93	2047	13.49	1153	18.38	2056
stocfor3	*790.19	7232	1833.75	17416	1178.61	10123	1375.86	16330
pds-1	*1.24	449	3.72	209	4.02	829	4.95	814
pds-2	*7.41	1181	20.50	926	32.58	2902	25.81	1785
pds-3	*16.65	1698	66.21	2127	120.74	5318	67.22	2656
pds-4	*52.86	3348	181.06	4306	743.91	18560	183.92	5144
pds-5	*86.20	4208	261.52	4562	1811.53	34958	326.74	6733
RP	16B, 21W	—	5B, 14W	—	2B, —	—	0B, 10W	—

RP: Relative performance, **B**: Best out of all methods, **W**: Winner against CLTSF crash procedure.

TIME: CPU time in seconds, **ITER**: The number of simplex iterations to solve the LP problem with different advanced bases and the unit basis. *: The best solution time.

Table 4: The consolidated results.

In Table 4, the results displayed in column two under the heading Lagrangean relaxation are obtained as the best out of the two crash procedures *CNET1* and *CNET2* which can be claimed to be good crash procedures. However, we cannot make any general conclusion as to which of the network based crash procedures fully dominates all others. The performance gain of network based crash procedures appears to be problem specific.

Results in Table 4 reveal that for some models, such as `pilot, cre-b, degen3`, the solution time and the number of simplex iterations obtained by Benders decomposition of the

LPEN with one cut are reduced. However, the crash procedures based on Lagrangean relaxation outperform the restricted Benders decomposition. We observe that the best of the Lagrangean relaxation and Benders decomposition performs better than the crash *CLTSF* procedure.

Considering these computational results, we make the following broad observations.

1. Exploiting the embedded pure network structure within large scale LP problems improves the total solution time and number of iterations in most of the cases compared with the sparse simplex method with the advanced basis and unit basis.

2. The sparse simplex solution time and iteration number are decreased by solving the LP problem with the advanced basis obtained by our start up procedure which uses Benders decomposition in a restricted form. However, overall the Lagrangean relaxation procedure dominates the other procedures.

3. If we apply the best network based crash procedure chosen out of *CNET1* and *CNET2*, then this result dominates the performance of the *CLTSF* in most cases.

9 Conclusions

In this paper, we have shown how the embedded pure network structure in LP problems can be used to create an advanced starting point to solve the original LP problem by the simplex method. The computational results show that even for general classes of LP problems this can be an effective procedure for creating an advanced basis. One way forward is to explore further the decomposition scheme for creating an advanced basis and the obvious step is to further refine the Benders decomposition method.

Acknowledgments

We are grateful for the financial support provided by the Turkish Educational Council and Muğla University to Miss Nalân Gülpınar; this grant had enabled her to undertake research at Brunel University.

References

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

[2] Ali, A.I., Hemalatha, T., A Network Relaxation Based Enumeration Algorithm for Set Partitioning, *European Journal of Operational Research*, Vol. 38, (1989), p. 76–85.

[3] Beasley, J.E., Lagrangean Relaxation, in *Modern Heuristic Techniques for Combinatorial Problems*, Ed. Reeves, C.R., Blackwell Scientific Publications, Oxford, (1993).

[4] Benders, J.F., Partitioning Procedures for Solving Mixed-Variables Programming Problems, *Numerische Mathematik,* 4, (1962), p. 238–252.

[5] Bixby, R., Implementing Simplex Method: The Initial Basis, *ORSA Journal On Computing*, Vol 4, No. 3, (1992), p. 267–284.

[6] Bixby, R., Fourer, R., Finding Embedded Network Rows in Linear Programs I. Extraction Heuristics, *Management Science*, Vol. 34, No. 3, (1988), p. 342–376.

[7] Brown, G.G., Olson, M.P., Dynamic Factorization in Large-Scale Optimization, *Mathematical Programming*, Vol. 64, (1994), p. 17–51.

[8] Bryson, N., Parametric Programming and Lagrangian Relaxation: The Case of the Network Problem with a Single Side Constraint, *Computers and Operations Research*, Vol. 18, No. 2, (1991), p. 129–140.

[9] Bryson, N., A Parametric Programming Methodology to Solve the Lagrangian Dual for Network Problems with Multiple Side-Constraints, *Computers and Operations Research*, Vol. 20, No. 5, (1993), p. 541–552.

[10] Carstens, D.M., Crashing Techniques, in Orchard-Hays, W., *Advanced Linear Programming Computing Techniques*, McGraw-Hill, (1968), p. 131–141.

[11] Chen, S., Saigal, R., A Primal Algorithm for Solving a Capacitated Network Flow Problem with Additional Constraints, *Networks*, Vol. 7, (1977), p. 59–79.

[12] Ellison, E.F.D., Hajian, M., Levkovitz, R., Maros, I., Mitra, G., *A Fortran Based Mathematical Programming System: FortMP*, Brunel University, London and NAG Ltd, Oxford, (1995).

[13] Fisher, M., The Lagrangean Relaxation Method for Solving Integer Programming Problems, *Management Science*, Vol. 27, No. 1, (1981), p. 1–18.

[14] Fisher, M., Jaikumar, R., Wassenhove, L.N.V, A Multiplier Adjustment Method for the Generalized Assignment Problem, *Management Science*, Vol. 32, No. 9, (1986), p. 1095–1103.

[15] Fourer, R., Hsu, A.C., Exploiting Network Structure for Solving Large-Scale Linear Programming Models, *Working Paper*, (1996).

[16] Gay, D.M., Electronic Mail Distribution of Linear Programming Test Problems, *Mathematical Programming Society Coal, Newsletter,* Vol. 13, (1985), p. 10–12.

[17] Frangioni, A., Multicommodity Min Cost Flow Problems, `http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html`

[18] Geoffrion, A.M., Lagrangean Relaxation for Integer Programming, *Mathematical Programming Study*, Vol. 2, (1974), p. 82–114.

[19] Glover, F., Karney, D., Klingman, D., Russell, R., Solving Singly Constrained Transshipment Problems, *Transportation Science*, Vol. 12, No. 4, (1978), p. 277–297.

[20] Glover, F., Klingman, D., The Simplex SON Algorithm for LP/Embedded Network Problems, *Mathematical Programming Study*, Vol. 15, (1981), p. 148–176.

[21] Glover, F., Klingman, D., Basis Exchange Characterizations for the Simplex SON Algorithm for LP/Embedded Networks, *Mathematical Programming Study*, Vol. 24, (1985), p. 141–157.

[22] Guignard, M., Rosenwein, B., An Application Oriented Guide for Designing Lagrangean Dual Ascent Algorithms, *European Journal of Operational Research*, Vol. 43, (1989), p. 197–205.

[23] Guignard, M., A Lagrangean Dual Ascent Algorithm for Simple Plant Location Problems, *European Journal of Operational Research,* Vol. 35, (1988), p. 193–200.

[24] Gülpınar, N., Gutin, G., Mitra, G., Maros, I., Detecting Embedded Pure Network Structures Using GUB and Independent Set Algorithms, *Computational Optimization and Applications*, Vol. 15, (2000), p. 235–247.

[25] Gülpınar, N., Gutin, G., Mitra, G., Detecting Embedded Pure Network Structures By Using Independent Set Algorithm, Brunel University, (1997), TR/12/97.

[26] Gould, N.I.M., Reid, J.K., New Crash Procedures for Large Systems of Linear Constraints, *Mathematical Programming*, Vol. 45, (1989), p. 475–501.

[27] Held, M., Karp, R.M., The Travelling Salesman Problem and Minimum Spanning Trees, *Operations Research*, Vol. 18, (1970), p. 1138–1162.

[28] Held, M., Karp, R.M., The Travelling Salesman Problem and Minimum Spanning Trees, Part II. *Mathematical Programming*, Vol. 1, (1971), p. 6–25.

[29] Infanger, G., *Planning Under Uncertainty: Solving Large Scale Stochastic Linear Program*, Boyd and Fraser, Danvers, MA, (1994).

[30] Mamer, J. W., McBride, R.D., A Decomposition-Based Pricing Procedure for Large-Scale Linear Programs: An Application to Linear Multicommodity Flow Problem, *Management Science*, Vol. 46, No. 5, (2000), p. 693–709.

[31] Maros., I., A Practical Anti-Degeneracy Row Selection Technique in Network Linear Programming, *Annals of Operations Research*, Vol. 47, (1993), p. 431–442.

[32] Maros, I., Mitra, G., Strategies for Creating Advanced Bases for Large-Scale Linear Programming Problems, *Informs Journal on Computing*, Vol. 10, No. 2, (1998), p. 248–260.

[33] Maros, I., Mitra, G., Simplex Algorithms, in *Advances in Linear and Integer Programming*, Ed. Beasley, J., Oxford University Press, (1996), p. 1–46.

[34] Maros, I., Mitra, G., Finding Better Starting Bases for Simplex Method, Ed. Kleinschmidt, P., *Operations Research Proceedings 1995*, Springer, Berlin, (1996), p. 7–12.

[35] Maros, I., Performance Evaluation of the MINET Minimum Cost Netflow Solver, *DIMACS Series in Discrete Mathematics and Computer Science*, Vol. 2, (1993), p. 199–217.

[36] Mathies, S., Mevert, P., A Hybrid Algorithm for Solving Network Flow Problems with Side Constraints, *Computers and Operations Research*, Vol. 25, (1998), p. 745–756.

[37] McBride, R.D., Solving Embedded Generalised Network Problems, *European Journal of Operational Research*, Vol. 21, (1985), p. 82–92.

[38] McBride, R.D., Mamer, J. W., Solving Multicommodity Flow Problems with an Embedded Network Simplex Algorithm, *Journal on Computing*, Vol. 9, No. 2, (1997).

[39] McBride, R.D., Advances in Solving Multicommodity Flow Problem, *Interfaces*, Vol. 28, No. 2, (1998), p. 32–41.

[40] Rogers, D.F., Plante, R.D., Wong, R.T., Evans, J.R., Aggregation and Disaggregation Techniques and Methodology in Optimisation, *Operations Research*, Vol. 39, No. 4, (1991).

[41] Venkataramanan, M.A., Dinkel, J.J., Mote, J., A Surrogate and Lagrangian Approach to Constrained Network Problems, *Annals of Operations Research*, 20, (1989), p. 283–302.