Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7, D-14195 Berlin-Dahlem, Germany

Andreas Löbel

Solving Large-Scale Real-World Minimum-Cost Flow Problems by a Network Simplex Method

Preprint SC 96–7 (Februar 1996)

Solving Large-Scale Real-World Minimum-Cost Flow Problems by a Network Simplex Method

Andreas Löbel*

Abstract

This paper presents a large-scale real-world application of the minimum-cost flow problem, describes some details of a new implementation of the network simplex algorithm, and reports on computational comparisons.

The real-world test sets include minimum-cost flow problems that are based on single-depot vehicle scheduling problems and on a Lagrangean relaxation of multiple-depot vehicle scheduling problems. Some of the problems are extremely large with up to 42,000 nodes and 20,000,000 arcs. The standard test problems are generated with NETGEN and include parts of the DIMACS standard problems. Our network simplex code is compared with RELAX-IV, Cost Scaling 2 version 3.4, and CPLEX's network solver NETOPT.

1 Introduction

Minimum-cost flow algorithms and, in particular, the network simplex algorithm have been investigated profoundly. Especially for the network simplex algorithm, there are many excellent publications that explain all the theoretical and algorithmic details such as basis characterization, data structures, computation of the dual and primal variables, basis update, etc.

The main motivation for this article was to report about very large minimum-cost flow problems arising in vehicle scheduling for public mass transportation and to demonstrate how to solve these real-world problems. (An overview about problem definitions and models in vehicle scheduling is given in Daduna and Paixão [1995] and Desrosiers, Dumas, Solomon, and Soumis [1995].)

We are currently working on *NP*-hard Multiple-Depot Vehicle Scheduling Problems (MDVSP) arising at large German public transportation companies. Our partners are the HanseCom GmbH, Hamburg, which is a subsidiary of the Hamburger Hochbahn AG and Siemens-Nixdorf Informationssysteme (SNI), and the world'CPLEX fourth largest public transportation company Berliner Verkehrsbetriebe (BVG) together with IVU GmbH,

*Konrad-Zuse-Zentrum für Informationstechnik Berlin, Heilbronner Straße 10, 10711 Berlin, Germany

Berlin. Our partners provided us with real-world data from the city of Berlin, the city of Hamburg, and the region Hamburg-Holstein.

In the case of just one depot, the MDVSP reduces to a Single-Depot Vehicle Scheduling Problem (SDVSP) that can be solved in polynomial time by minimum-cost flow algorithms. SDVSPs occur in practice for geographic reasons and, in particular, if special lines in a city are operated by a subsidiary company. We encountered single-depot subproblems of MDVSPs having up to 8,000 nodes and 7,200,000 arcs. When we apply a Lagrangean relaxation to a MDVSP and combine all depots into a single depot, we obtain "pure" SDVSP of extremely large-scale. We employ our network simplex code as a subroutine for the solution of this Lagrangean relaxation by a subgradient method. The instances arising here have up to 42,000 nodes and 20,000,000 arcs.

In addition, we have also tested some standard NETGEN minimum-cost flow problems proposed by a DIMACS algorithm challenge. The results of this implementation challenge and the related workshop in 1991 were published in Johnson and McGeoch [1993]. One of the workshop'CPLEX observations for minimum-cost flow problems is that there is no algorithm that performs best for all problems, see Bland, Cheriyan, Jensen, and Ladányi [1993]. A panel discussion at the DIMACS workshop about minimum-cost flow algorithms summarized the findings as follows, see Johnson [1993]:

- The challenge has no dominant winning algorithm for minimum-cost flow problems. Therefore, one should ask which code is the most robust implementation.
- Sometimes, the important question is not how fast an algorithm runs, but whether a code can solve a problem instance at all (especially when the minimum-cost flow code is used as a subroutine).
- There is only a small number of relatively small real-world instances.

Our paper has the following structure: First, we sketch the SDVSP, give a brief formulation of the minimum-cost flow problem, and describe our real-world minimum-cost flow problems. Second, we give a short outline of the primal network simplex algorithm and some implementation details of our network simplex code. Third, we report about our computational investigations of single-depot minimum-cost flow problems, which are defined by real-world one-depot problems and by a Lagrangean relaxation for the MDVSP, and of the NETGEN problems. Our implementation is compared with the two publically available minimum-cost flow solvers Cost Scaling 2 (Goldberg), RELAX-IV (Bertsekas and Tseng), and the network solver NETOPT from CPLEX [1995].

We assume the reader to be familiar with linear programming and network flow theory and algorithms, especially with the network simplex algorithm.

2 The Single-Depot Vehicle Scheduling Problem

The Single-Depot Vehicle Scheduling Problem is defined as follows: Given is a *depot* that represents a fleet of vehicles. The vehicles are all stationed at the same place and need not be distinguished. It is possible to consider depot capacities, i. e., a maximum number of vehicles. The depot has a start and an end point where the vehicles begin and terminate their daily scheduled run.

A set of lines and a given timetable define a set of so-called timetabled trips to carry passengers. Each *timetabled trip* has a first and a last stop, a departure and an arrival time, and must be serviced by exactly one vehicle from the first to the last stop.

There are further types of trips that all run without passengers: A *pull-out trip* connects the start point of the depot with the first stop of a timetabled trip, a *pull-in trip* connects the last stop of a timetabled trip with the end point of the depot, and a *deadhead* or *dead running trip* connects the last stop of a timetabled trip with the first stop of a succeeding timetabled trip. If it is possible to connect two timetabled trips in time, the corresponding deadhead trip is called *compatible*. Pull-out trips and pull-in trips, respectively, are used whenever a vehicle begins or terminates its daily scheduled run. A (compatible) deadhead trip is used when two successive timetabled trips are serviced by the same vehicle in sequence.

The use of a pull-out, pull-in, or deadhead trip causes some *cost* that depends on operational interests. The main objective is to minimize the total number of vehicle runs that are necessary to operate all timetabled trips and, secondary, to minimize the operating cost (given the minimum number of vehicles). We realize this two-stage objective by defining the cost of a pull-out, pull-in, or deadhead trip as its operational cost and increase the cost of each pull-out trip by a sufficiently large *bigM*. Another possible objective may be to find a cost minimal scheduling for a fixed or bounded fleet size. In this case, all costs are set to the operational costs.

The task of the SDVSP is to find a set of cost minimal vehicle runs such that each timetabled trip is contained in exactly one vehicle run and, for the fixed or capacitated case, such that the number of vehicle runs does not violate the depot capacities.

Bertossi, Carraresi, and Gallo [1987] and Branco and Paixão (1987,1988) give similar descriptions of the SDVSP; Branco, Costa, and Paixão [1995], Lamatsch (1988,1991), Bertossi, Carraresi, and Gallo [1987], and Mesquita and Paixão [1992] discuss Lagrangean relaxations of the MDVSP where SDVSPs appear.

3 The Minimum-Cost Flow Problem

Veldhorst [1993] compiled a bibliography containing 370 references to single-, multicommodity, and other classes of flow papers published by 1993. Two excellent books about network flows are Ahuja, Magnanti, and Orlin [1993] and Bazaraa, Jarvis, and Sherali [1990].

Given a connected and directed graph $D = (V, A)$, a linear cost function $c : A \longmapsto \mathbb{Q}$, upper bounds $u : A \longmapsto \mathbb{Z}_+$, and node imbalances $b : V \longmapsto \mathbb{Z}$ such that $\sum_{i \in V} b_i = 0$. A node i is called a supply node when $b_i > 0$, a demand node when $b_i < 0$, and a transshipment node else. The minimum-cost flow problem is to find a vector $x^* : A \longmapsto \mathbb{Z}$

such that x^* is an optimal solution of the linear program

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1}$$

subject to

$$\begin{aligned} \sum_{j\,:\,(i,j)\in A} x_{ij} - \sum_{j\,:\,(j,i)\in A} x_{ji} = b_i & \qquad \text{for all } i \in V, & (2)\\ 0 \leqslant x_{ij} \leqslant u_{ij} & \qquad \text{for all } (i,j) \in A. & (3) \end{aligned}$$

The equations (2) are the so-called *flow conservation constraints* and the inequalities (3) are the *flow bounds* on x. A flow x is called a *feasible* flow, if it satisfies the flow conservation constraints and the flow bounds. In matrix notation, (1–3) reads

$$\min\{c^{\mathrm{T}} x \mid Ex = b,\ 0 \leqslant x \leqslant u\}, \tag{4}$$

where E is the node-arc incidence matrix of the digraph D.

Consider $\pi : V \longmapsto \mathbb{Q}$ (the so-called *node potentials*) and $\delta : A \longmapsto \mathbb{Q}$ as the dual multipliers for the flow conservation constraints (2) and the upper bounds (3), respectively. The dual problem of (4) is

$$\max\{\pi^{\mathrm{T}} b - \delta^{\mathrm{T}} u \mid \pi^{\mathrm{T}} E - \delta^{\mathrm{T}} \leqslant c^{\mathrm{T}},\ \delta \geqslant 0\}, \tag{5}$$

which is

$$\max \sum_{i\in V} \pi_i b_i - \sum_{(i,j)\in A} \delta_{ij} u_{ij} \tag{6}$$

subject to

$$\begin{aligned} \pi_i - \pi_j - \delta_{ij} \leqslant c_{ij} & \qquad \text{for all } (i,j) \in A, & (7)\\ \delta_{ij} \geqslant 0 & \qquad \text{for all } (i,j) \in A. & (8) \end{aligned}$$

Let $\overline{c}_{ij} := c_{ij} - \pi_i + \pi_j$ denote the *reduced cost* of an arc $(i,j) \in A$. With this definition, the well known optimality condition reads: A feasible flow x and feasible node potentials π are optimal for (4) and (5) if

$$\begin{aligned} \overline{c}_{ij} > 0 & \quad\Rightarrow\quad x_{ij} = 0, & (9)\\ \overline{c}_{ij} < 0 & \quad\Rightarrow\quad x_{ij} = u_{ij}, & (10)\\ 0 < x_{ij} < u_{ij} & \quad\Rightarrow\quad \overline{c}_{ij} = 0. & (11) \end{aligned}$$

4 The Single-Depot Vehicle Scheduling Network

Let d^+ and d^- denote the start and the end point of the depot. Let $\mathcal{T}$ denote the set of all timetabled trips. For each timetabled trip $t \in \mathcal{T}$ let t^- denote its first stop and t^+ denote its last stop,

$$\mathcal{T}^- := \{t^- \mid t \in \mathcal{T}\} \quad \text{and} \quad \mathcal{T}^+ := \{t^+ \mid t \in \mathcal{T}\}.$$

The SDVSP defines the following directed network $D = (V, A)$. The nodes V are the depot'CPLEX start and end point and the first and last stops of all timetabled trips, i. e.,

$$V := \mathcal{T}^- \cup \mathcal{T}^+ \cup \{d^+, d^-\}.$$

The arc set A contains one arc for each pull-out, pull-in, and compatible deadhead trip; parallel arcs (if we allow parallel deadhead trips) are possible. In addition, A contains one arc $\{(d^-, d^+)\}$ for the vehicle return from the end to the start point of the depot. Each arc $a \in A$ is unbounded, i. e., $u_{ij} = \infty$, unless we consider depot capacities; in this case, the upper bound $u_{(d^-,d^+)}$ is set to the depot capacity. If $u_{(d^-,d^+)} = \infty$, we shrink the two depot nodes d^- and d^+ to one single node d.

The nodes for the first stops are demand nodes, the nodes for the last stops are supply nodes, and the nodes for the start and end point (or the shrunk node d) of the depot are transshipment nodes, i. e., $b_{t^+} := 1$, $b_{t^-} := -1$, $b_{d^+} := 0$, and $b_{d^-} := 0$. Figure 1 illustrates a small SDVSP.

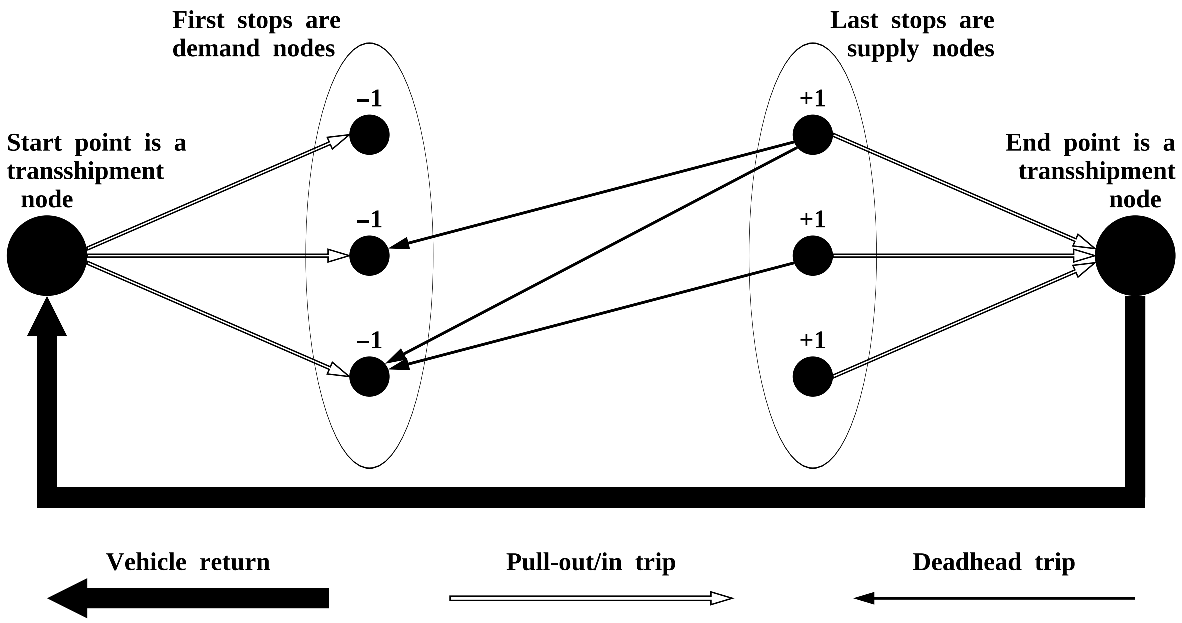


Figure 1: A network for the SDVSP.

5 The Primal Network Simplex Algorithm

The network simplex algorithm with upper bound technique is a specialized revised simplex algorithm that exploits the structure of network flow problems (for a description of the revised simplex algorithm see Dantzig [1963] or Chvátal [1980]). The linear algebra of the simplex algorithm is replaced by simple network operations. Helgason and Kennington [1995] and Ahuja, Magnanti, and Orlin [1993] describe the primal network simplex algorithm and give pseudocodes, implementation hints, etc.

To apply the simplex algorithm to (4), we need a full rank constraint matrix. For a connected network D, the rank of the flow conservation constraints (2) is equal to $|V| - 1$ and the flow conservation constraint for one node (called the *root node*) can be eliminated. We will assume that we have chosen such a root node and have eliminated its flow conservation constraint, i. e., the reduced node-arc incidence matrix, which we also call E, has full rank. It is well known that every nonsingular basis matrix B of E corresponds to a spanning tree of A in D and vice versa.

If $T \subseteq A$ is a spanning tree (and thus the variables x_{ij}, $(i,j) \in T$, are the *basic variables* corresponding to the basis $B := E_{\cdot,T}$) and if L and U denote the arcs that correspond to the *nonbasic variables* whose values are at the lower and upper bound, respectively, than the triple (T, L, U) is called a *basis structure*. For given nonbasic arc sets L and U, the right hand side b transforms to

$$b' := b - \sum_{(i,j) \in U} E_{\cdot,ij} u_{ij}.$$

The associated basic solution is the solution of the system $Bx_T = b'$, the values of the node potentials are determined by the system $\pi^{\mathrm{T}} B = c_T^{\mathrm{T}}$, and the dual multipliers δ of the bounds $x \leqslant u$ by

$$\delta_{ij} := \begin{cases} -c_{ij} - \pi_j + \pi_i & \text{if } (i,j) \in U, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

A basis structure (T, L, U) is called *primal feasible* if x satisfies the flow bounds (3) and is called *dual feasible* if for all $(i,j) \in A \setminus T$:

$$\bar{c}_{ij} > 0 \quad \Rightarrow \quad (i,j) \in L, \tag{13}$$

$$\bar{c}_{ij} < 0 \quad \Rightarrow \quad (i,j) \in U. \tag{14}$$

A basis structure is called *optimal* if it is both primal and dual feasible. For a detailed description of the primal network simplex see Helgason and Kennington [1995].

6 Implementation Details

Many network flow textbooks contain (relatively similar) codes or pseudocodes of network flow algorithms. We started our implementation with such a code and tried to improve important algorithmic details to make the code more robust and efficient such that even truely large-scale problems can be solved routinely. We describe the key ingredients of our modifications of standard textbook codes. Most of our computational improvements result, in fact, from very efficient pricing strategies, which we describe at the end of this section. The importance of pricing follows from our experimental observations that our implementation still spends, on the average, more than 80 percent of the cpu time on pricing.

Computer Language and Data Structures.

Lustig [1990] investigates the influence of computer languages on an implementation of a primal network simplex code. His conclusions are that an addressed-based implementation (linked lists and pointers) is more efficient than a cursor-based implementation (vectors and indices) and that the performances of cursor-based implementations in C and Fortran are essentially the same. Our code is implemented in C with addressed-based data structures.

Over the last three decades, the basis tree representation and data structures for the network simplex algorithm have been investigated profoundly. Most of the network simplex implementations use similar data structures. We describe our version: All node and arc information, respectively, are stored in the following C structs:

```
struct node
{
    /* node potential */
    cost_t potential;

    /* rooted tree structure of basis tree */
    int subtreesize;             /* number of nodes in the
                                    subtree including this node  */
    struct node *pred;           /* predecessor node             */
    struct node *child;          /* first leave (or child) node  */
    struct node *right_sibling;  /* right sibling node           */
    struct node *left_sibling;   /* left sibling node            */
    struct arc *basic_arc;       /* basic arc between this node
                                    and the predecessor node     */
    int orientation;             /* zero if the basic_arc enters
                                    this node and one else       */
    flow_t flow;                 /* flow value of basic_arc      */
}
```

```
struct arc
{
    /* arc definitions */
    struct node *tail;
    struct node *head;
    cost_t cost;
    flow_t upper;           /* the upper bounds of the arc      */

    /* assignment to T, L, or U
       of the basis structure   */
    int ident;
}
```

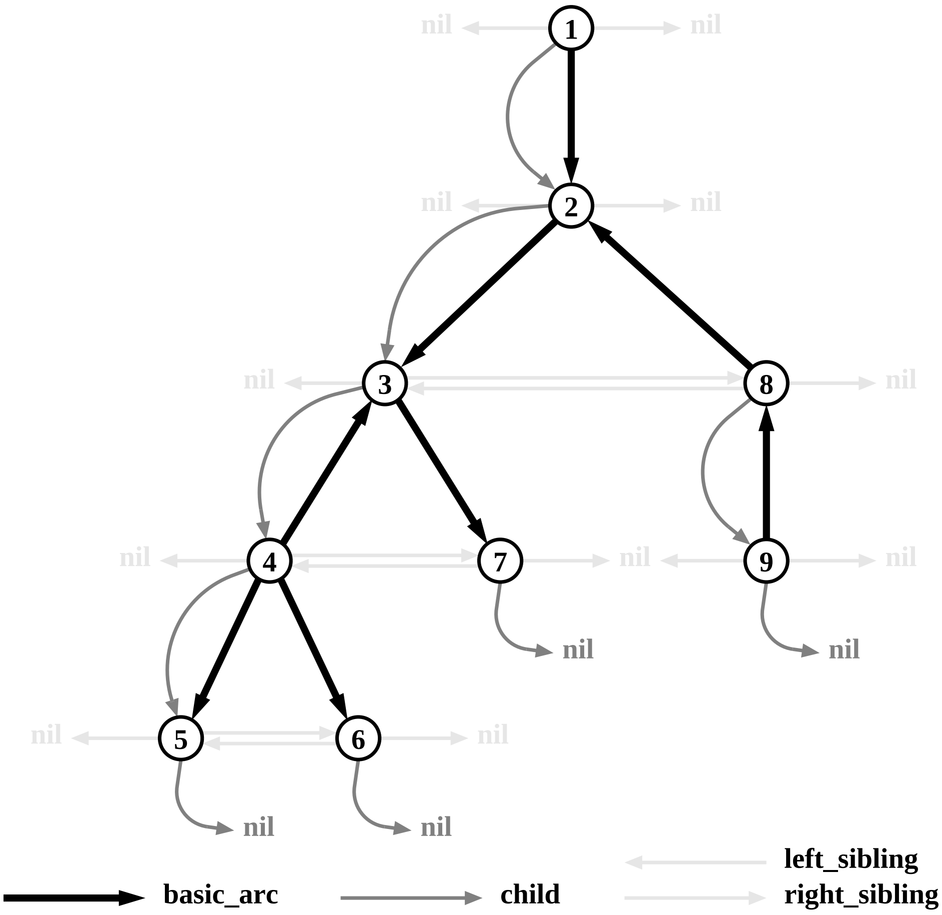
The variable types `cost_t` and `flow_t` are defined as integer or double variable types depending on the data input; integer types are default. Figure 2 shows a small example of a rooted basis tree for our data structures (the underlying network is a copy from Ahuja, Magnanti, and Orlin [1993]). The predecessor `pred` of a node is determined by its basic arc. Each node has at most one `child` node and the other children of a node can be reached by traversing the `right_sibling` links.

Our technique to store the rooted basis tree results from personal discussions with R. E. Bixby, whose CPLEX'CPLEX network simplex solver NETOPT has the same data structure for the basis tree. Helgason and Kennington [1995] describe a very similar basis structure, where the `child`, `right_sibling`, and `left_sibling` variables are replaced by a `thread` and `last_successor` index. Their representation is a little bit more compact, but it is easy to show that the two representations are equivalent.

The `subtreesize` and `pred` variables are necessary for the ratio test. The `child`, `right_sibling`, and `orientation` variables are necessary for the computation of the node potentials. The algorithm also works without the `left_sibling` variables, but they allow a more efficient basis update. For a detailed description of all these single network simplex steps, the reader is referred to Helgason and Kennington [1995].

Initial Basis Structure.

We assume that the considered network D is connected and that we have chosen a root node $r \in V$. The easiest way to find an initial primal feasible basis structure is as follows: Consider some node $i \in V \setminus \{r\}$. Depending on the value of the node imbalance b_i, we add to the arc set A either an artificial arc (i, r) if $b_i \geqslant 0$ or an artificial arc (r, i) if $b_i < 0$; we denote this larger network by D'. Each artificial arc has a lower bound of 0, an upper bound of infinity, and a sufficiently large cost coefficient *bigM*. The initial basis tree consist of all artificial arcs, all original arcs become nonbasic at their lower bound, and no arc is nonbasic at its upper bound. Obviously, this initial basis structure is primal feasible and the original network D is feasible if the network D' has a feasible solution where no artificial arc has a positive flow value. The use of an aritificial basis tree has



node	1	2	3	4	5	6	7	8	
subtreesize	9	8	5	2	1	1	1	2	1
pred	nil	1	2	3	4	4	3	2	8
child	2	3	4	5	nil	nil	nil	9	nil
right_sibling	nil	nil	8	7	6	nil	nil	nil	nil
left_sibling	nil	nil	nil	nil	nil	5	4	3	nil
orientation	-	0	0	1	0	0	0	1	1

Figure 2: Rooted basis tree.

several advantages. First, it has a simple structure and can be generated quickly. Second, the ratio test and the basis update are quite fast for the first iterations. We have also tried to generate an initial basis structure using a crash procedure. The performance, however, was always slower than starting with an aritificial basis tree. The only exceptions occur for special applications where particular problem knowledge can be exploited, which we describe in the next paragraph.

Delayed Column Generation and Sensitivity Analysis.

For large-scale networks, the performance may benefit from a delayed column generation approach. This means that, in a first step, only a subset $A' \subset A$, containing at least a feasible solution, is considered and that the flow value of each arc $a \in A \setminus A'$ is fixed to zero. When the smaller network (V, A') has been solved to optimality, all fixings of the ignored arcs are removed. Then, the reduced costs according to the last node potentials are computed. As long as there exist arcs that violate the optimality conditions, we add at least one and at most a (parameter controlled) maximum number of such arcs to A', reoptimize for the new arc set A', and iterate until optimality can be proved for the complete arc set A.

We have implemented delayed column generation only for our vehicle scheduling application. For problems of this type, our special network simplex code starts with an artificial basis structure as described above only for the first subnetwork. For all subsequent subnetworks, our network simplex code restarts with the optimal basis structure of the previous subnetwork.

Pricing.

The pricing rule has a significant influence on the performance. Ahuja, Magnanti, and Orlin [1993] describe some pricing rules such as Dantzig'CPLEX rule, first eligible arc rule, and a candidate list rule. We have implemented and tested these pricing rules in slightly modified ways. It turned out that our by far fastest rules are special candidate list rules, called *multiple partial pricing* (e. g., see Bixby(1992,1994)). Given two natural numbers K and J, the arc set A is divided into K candidate lists, each of size at least $\lfloor \frac{|A|}{K} \rfloor$. There is a "hot-list" of at most $J + K$ arcs, which is initially empty. The candidate lists are indexed from 1 to K. The candidate list number $next$, which defines the next examined candidate list in a pricing call, is set to 1. Candidate lists are always examined in a wraparound fashion. For one pricing call, the following steps are done: As long as the hot-list includes less than J arcs and not all candidate lists have been examined in this pricing step, we price out all arcs of the $next$ candidate list, add all nonbasic arcs that violate the optimality condition to the hot-list, and increment the $next$ variable by 1 (if $next > K$, we reset $next$ to 1). If all candidate lists have been examined and if the hot-list is still empty, the current basis structure is optimal. Otherwise, an arc of the hot-list that violates the reduced cost criteria most is selected as the basis entering arc. The last step of a pricing call is the preparation of the hot-list for the next pricing call: The new hot-list for the next pricing call contains at most $J - 1$ arcs among those arcs of the current hot-list that are not the basis entering arc and that have the most invalid reduced cost.

Multiple partial pricing is very sensitive to the number of arcs; this makes a fine tuning for every problem class necessary. We use the following default choices for K and J depending on the number of arcs:

Number of arcs	K	J
$\|A\| < 10,000$	30	5
$10,000 \leqslant \|A\| \leqslant 100,000$	50	10
$\|A\| > 100,000$	200	20

Compared to the multiple partial pricing with the default values of K and J as above, pricing rules such as first eligible arc rule or Dantzig'CPLEX rule need about 14 to 75 times more cpu time for the large-scale problems of our test set (e. g., the network problems from Tab. 4).

7 Test Data

We evaluate our code on different classes of networks: some are obtained from Klingman, Napier, and Stutz [1974] and created with their network generator NETGEN, some have been generated by ourselves with NETGEN, and some are real-world data from our application.

NETGEN Networks.

To make computational studies of network flow implementations comparable, some problem generators were developed to generate standard test sets. *The first DIMACS international algorithm implementation challenge: Problem definitions and specifications* (DIMACS [1990]) presents standard problem definitions including input and output formats. These formats are widely accepted and supported by almost all publicly available problem generators such as NETGEN (Klingman, Napier, and Stutz [1974]), GRIDGEN (Lee and Orlin), GRIDGRAPH (Goldberg), RMFGEN (Goldfarb and Grigoriadis), etc. All these problem generators are available from DIMACS [1993]. Our random test set was generated with NETGEN. The random number *seed*, which is an input parameter of NETGEN, is for all networks 13502460.

First, we have generated the 40 benchmark networks, which Klingman et al. propose in their article, and called them *Klingman et al. networks*. Second, we have generated networks whose upper capacities of the arcs are relatively small compared to the total supply such that an optimal basis contains many arcs at their upper bound. Third, we have generated transportation networks. The tables in Appendix B, Appendix C, and Appendix D show the NETGEN input parameter of our NETGEN problems.

Single-Depot Vehicle Scheduling Networks.

The second part of our test data are networks from our real-world application. These networks are based on 31 single-depot problems (dep1, dep2, etc.) and on Lagrangean relaxations of three multiple-depot problems (lagr1, lagr2, and lagr3). The main objective for these problems is to find a fleet minimal solution, this implies that the upper bound of the vehicle return arc is set to infinity and that the two nodes for the depot are shrunk to one node. The single-depot problems range from 1.291 nodes and 175.533 arcs to 8.193 nodes and arcs; the Lagrangean relaxation problems have nodes and arcs, nodes and arcs, and nodes and arcs (see Tab. 1).

Prob.	$\|V\|$	$\|A\|$	Sub-nets
dep1	8193	7224283	6
dep2	4583	2238585	4
dep3	1505	216483	3
dep4	4475	2008034	4
dep5	1297	175533	4
dep6	6577	4634374	4
dep7	1391	200757	3
dep8	1507	217058	3
dep9	5021	2577951	4
dep10	2731	771481	4
dep11	1583	270603	3
dep12	2923	921440	4
dep13	4567	2209154	4
dep14	2139	467469	4
dep15	1503	227876	4
dep16	2955	865660	4
dep17	1559	252055	3
dep18	2667	694181	4

Prob.	$\|V\|$	$\|A\|$	Sub-nets
dep19	1701	288994	4
dep20	3827	1415590	5
dep21	3103	952824	6
dep22	3841	1447401	5
dep23	1879	352459	3
dep24	5237	2688675	5
dep25	3121	967201	4
dep26	3995	1640499	8
dep27	1583	259148	4
dep28	1503	226089	3
dep29	2739	778733	6
dep30	1333	183727	3
dep31	2183	468763	4
lagr1	3669	1186270	4
lagr2	17127	19116767	8
lagr3	42007	10434409	6

Table 1: Vehicle scheduling problems.

We apply the delayed column generation approach, as described in Sec. 6, to all our vehicle scheduling problems. To guarantee a feasible solution for the first subnetwork, we add at least all pull-out and pull-in trips to the arc set A'. We limit the column generation between two subsequent subnetworks to at most 30,000 arcs. The numbers of subnetworks that are generated by our delayed column generation rule are listed in the columns “Subnets” of Tab. 1.

The minimum-cost flow problems, which we use to evaluate our code, are the complete single-depot problems (dep1, dep2, etc.), the first and the last subnetwork from each single-depot problem (dep$i$.a and dep$i$.b, $i = 1, \ldots, 31$) and some subnetworks from the subgradient method of each Lagrangean relaxation problem (lagr$i$.a, lagr$i$.b, etc., $i = 1, 2, 3$). The complete Lagrangean relaxation problems are only evaluated for our delayed column generation approach because the problems need too much memory for the other codes. The problems lagr2.e and lagr2.f (see Tab. 4) have the same number of variables but different objective functions because lagr2.f was generated after a change of the Lagrangean multipliers without applying column generation. The same applies to the problems lagr2.g and lagr2.h.

8 Computational Results

Our network simplex code, called MCF, is compared with RELAX-IV written by Bertsekas and Tseng [1994], Cost Scaling 2 version 3.4 (CS2-3.4) written by Goldberg [1992], and CPLEX'CPLEX network solver NETOPT.

The RELAX-IV code was compiled with the SUN Fortran compiler f77 SC3.0.1 using the compiler optimization options "-fast -O4 -cg92 -dalign -libmil". The CS2-3.4 code was compiled with the Gnu gcc compiler 2.6.3 using the compiler optimization option "-O3". The MCF code was compiled with the SUN C compiler cc[1] SC3.0.1 using the compiler optimization options "-fast -xO4 -xcg89 -xlibmil".

The reported running times are in cpu seconds on a SUN SPARCstation 20 with a "Model 71 SuperSPARC-II SPARCmodule" CPU and 384 MByte main memory (1.8 GByte virtual memory). All running times are measured without any read time and without the transformation time for the RELAX-IV input format[2]. For NETOPT, only the "Network Time", which CPLEX gives, is measured. Tables 5, 6, 7, 4, 9, 11, and 13 in the appendices show the cpu times of MCF, CS2-3.4, RELAX-IV (default and auction version), and NETOPT for our different benchmark problems. A summary of all running time tables is shown in Tab. 2.

Our MCF code is the fastest code for almost all test instances except for the capacitated NETGEN problems, which the auction version of RELAX-IV solves faster. The most competitive algorithm for the NETGEN problems is RELAX-IV (both default and auction version) and the most competitive algorithm for the vehicle scheduling problems is CS2-3.4. In addition, MCF (default) has the lowest memory requirements; code and data segment need at most 227 MByte memory (for the largest network dep1). On the average, RELAX-IV[3] requires a factor of 1.5, CS2-3.4 requires a factor of 2, and the network solver NETOPT requires a factor of 3 to 5 times the memory needed by MCF.

The delayed column generation approach was only compared to the other codes for the complete vehicle scheduling problems dep1, dep2, etc. The default version of RELAX-IV was not able to solve two of these problems, namely the largest problem instance dep1 and the third largest problem instance dep24. We are not able to give the NETOPT running

[1]The gcc compiler would create a 5 percent more efficient code on the average!

[2]For the problem dep1, the program "dimacsconv", which is part of the RELAX package, needs 45 minutes cpu time to transform the dimacs format into the RELAX.INP format and fills more than 865 MByte on the hard disk (280 MByte for the file RELAX.INP and 585 MByte for the file TEMP, both together more than six times the dimacs input file).

[3]RELAX-IV is the only Fortran implementation and, thus, has an inflexible memory management.

Sums of Running Times in Seconds						
All problems	MCF		CS2-3.4	RELAX-IV		CPLEX NETOPT
	column gener.	default		default	auction	
dep1, dep2, etc.	746.2	4392.4	5778.6	—	12612.5	—
dep*.a, dep*.b		231.9	686.7	898.4	614.5	493.3
lagr1.*	(9.0)	17.2	68.0	31.1	44.5	41.0
lagr2.*	(325.3)	840.7	2293.4	3017.1	4560.3	1735.3
lagr3.*	(391.5)	1948.1	2031.1	7889.0	16418.3	3099.5
Klingman et al.		10.22	23.88	15.45	15.23	26.91
Capacitated		183.3	250.1	185.3	160.7	444.8
Transportation		10.0	29.9	12.5	14.3	27.7

Normalized Sums of Running Times						
All problems	MCF		CS2-3.4	RELAX-IV		CPLEX NETOPT
	column gener.	default		default	auction	
dep1, dep2, etc.	0.17	1.0	1.32	—	2.87	—
dep*.a, dep*.b		1.0	2.96	3.87	2.65	2.13
lagr1.*	(0.52)	1.0	3.95	1.81	2.59	2.38
lagr2.*	(0.38)	1.0	2.73	3.59	5.42	2.06
lagr3.*	(0.20)	1.0	1.04	4.05	8.43	1.59
Klingman et al.		1.0	2.34	1.51	1.49	2.63
Capacitated		1.0	1.36	1.01	0.88	2.43
Transportation		1.0	2.99	1.25	1.43	2.77

Table 2: Running times.

times for these problems because NETOPT[4] needs more than 400 MByte memory for many of them (this is due to the generality of the LP-solver CPLEX). The cpu time savings for the MCF delayed column generation are significant.

The column "column gener." in Tab. 2 shows in the rows lagr1.*, lagr2.*, and lagr3.* the running times for the complete Lagrangean relaxation problems lagr1, lagr2, and lagr3 from Tab. 3 without the times needed for the delayed column generation part. (i. e., we give only the running times for the solution of all considered subnetworks).

[4]Before CPLEX calls NETOPT, the network is stored as a sparse row and as a sparse column matrix, which explains the extremely large memory requirements.

9 Conclusions

We presented a minimum-cost flow application to a real-world vehicle scheduling problem containing many real-world network flow instances, some of extremely large-scale. Although the default RELAX-IV version could not solve two of our largest problems and, for some problems, NETOPT needs too much main memory (due to the generality of the LP-solver CPLEX), all evaluated codes (CS2-3.4, RELAX-IV, NETGEN, and our MCF) are robust implementations and able to solve even large-scale minimum-cost flow problems.

What does MCF distinguish from the other codes? First, at least for the real-world and some NETGEN problems, MCF performs on the average always better than the other codes and is always among the two fastest codes for each problem instance. Especially for the extremely large real-world problems dep1, dep2, etc., the MCF delayed column generation approach outperforms by far all the other codes including the default version of MCF. Second, MCF has the lowest memory requirements, which may be advantageous when the main memory is limited or when the code is only a module in a large program package. Third, the most important advantage is the possibility of a delayed column generation and sensitivity analysis approach, which is indispensable for an efficient solution of very large-scale problems.

Our robust and very efficient minimum-cost flow solver MCF (with delayed column generation) is integrated in several subroutines within a large program package for the solution of multiple-depot vehicle scheduling problems. This program package, in which the CPLEX LP-solver is also an important part, enables us to solve real-world multiple-depot vehicle scheduling problems – even from cities as large as Hamburg and Berlin – to optimality.

References

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1989). *Network Flows.* In Nemhauser, Rinnooy Kan, and Todd [1989], chapter IV, pages 211–369.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., editors (1995b). *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science.* Elsevier Science B.V., Amsterdam.

Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., editors (1995a). *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science.* Elsevier Science B.V., Amsterdam.

Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D. (1990). *Linear programming and network flows.* John Wiley & Sons, Inc., 2nd edition.

Bertossi, A. A., Carraresi, P., and Gallo, G. (1987). On some matching problems arising in vehicle scheduling models. *Networks*, 17:271–181.

Bertsekas, D. P. and Tseng, P. (1994). RELAX-IV: A faster version of the RELAX code for solving minimum cost flow problems. Technical report.

Bixby, R. E. (1992). Implementing the simplex method: The initial basis. *ORSA Journal on Computing*, 4(3):267–284.

Bixby, R. E. (1994). Progress in linear programming. *ORSA Journal on Computing*, 6(1):15–22.

Bland, R. G., Cheriyan, J., Jensen, D. L., and Ladányi, L. (1993). An empirical study of min cost flow algorithms. In Johnson and McGeoch [1993].

Branco, I., Costa, A., and Paixão, J. M. P. (1995). Vehicle scheduling problem with multiple type of vehicles and a single depot. In Daduna, Branco, and Paixão [1995].

Branco, I. M. and Paixão, J. P. (1987). A quasi-assignment algorithm for bus scheduling. *Networks*, 17:249–269.

Branco, I. M. and Paixão, J. P. (1988). Bus scheduling with a fixed number of vehicles. In Daduna and Wren [1988], pages 28–40.

Chvátal, V. (1980). *Linear programming*. W. H. Freeman and Company, New York.

CPLEX (1995). *Using the CPLEX Callable Library*. CPLEX Optimization, Inc., Suite 279, 930 Tahoe Blvd., Bldg 802, Incline Village, NV 89451, USA. URL: http://www.cplex.com/.

Daduna, J. R., Branco, I., and Paixão, J. M. P., editors (1995). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems. Springer Verlag.

Daduna, J. R. and Paixão, J. M. P. (1995). Vehicle scheduling for public mass transit – an overview. In Daduna, Branco, and Paixão [1995].

Daduna, J. R. and Wren, A., editors (1988). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems. Springer Verlag.

Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton University Press, Princeton.

Desrochers, M. and Rousseau, J.-M., editors (1992). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems. Springer Verlag.

Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F. (1995). *Time Constrained Routing and Scheduling*. In Ball, Magnanti, Monma, and Nemhauser [1995a], chapter 2, pages 35–139.

DIMACS (1990). The first DIMACS international algorithm implementation challenge: Problem definitions and specifications. Available from anonymous ftp from dimacs.rutgers.edu, in the directory /pub/netflow/general-info.

DIMACS (1993). The first DIMACS international algorithm implementation challenge. available from anonymous ftp from dimacs.rutgers.edu, in the directory /pub/netflow.

Du, D.-Z. and Pardalos, P. M., editors (1993). *Network Optimization Problems: Algorithms, Applications and Complexity*, volume 2 of *Series on Applied Mathematics*, Singapore, New York, London. World Scientific Publishing Co. Pte. Ltd.

Goldberg, A. V. (1992). An efficient implementation of a scaling minimum-cost flow algorithm. Report No. STAN-CS-92-1439, Department of Computer Science, Standford University, Stanford, California 94305.

Helgason, R. V. and Kennington, J. L. (1995). *Primal Simplex Algorithms for Minimum Cost Network Flows*. In Ball, Magnanti, Monma, and Nemhauser [1995b], chapter 2, pages 85–133.

Johnson, D. S. (1993). Appendix B: Panel discussion highlights. In Johnson and McGeoch [1993].

Johnson, D. S. and McGeoch, C. C., editors (1993). *Network Flows and Matching*, volume 12 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Klingman, D., Napier, A., and Stutz, J. (1974). NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20(5).

Lamatsch, A. (1988). *Wagenumlaufplanung bei begrenzten Betriebshofkapazitäten*. PhD thesis, Universität Fridericiana zu Karlsruhe (TH).

Lamatsch, A. (1991). An approach to vehicle scheduling with depot capacity constraints. In Desrochers and Rousseau [1992].

Lustig, I. J. (1990). The influence of computer language on computational comparisons: An example from network optimization. *ORSA Journal on Computing*, 2(2):152–161.

Mesquita, M. and Paixão, J. (1992). Multiple depot vehicle scheduling problem: A new heuristic based on quasi-assignment algorithms. In Desrochers and Rousseau [1992].

Nemhauser, G. L., Rinnooy Kan, A. H. G., and Todd, M. J., editors (1989). *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*. Elsevier Science B.V., Amsterdam.

Veldhorst, M. (1993). A bibliography on network flow problems. In Du and Pardalos [1993], pages 301–331.

Appendix A Vehicle Scheduling Networks

Prob.	$\|V\|$	$\|A\|$	Sub-nets	Column generation
lagr1	3669	1186270	4	14.78
lagr2	17127	19116767	8	686.94
lagr3	42007	10434409	6	544.15

Table 3: MCF column generation cpu times (sec) for the Lagrangean relaxation problems.

Prob.	$\|V\|$	$\|A\|$	MCF default	CS2-3.4	RELAX-IV default	RELAX-IV auction	CPLEX NETOPT
lagr1.a	3669	94600	2.74	14.57	5.97	5.35	7.57
lagr1.b	3669	122265	5.01	17.68	11.02	16.56	9.83
lagr1.c	3669	124733	4.84	18.18	7.07	11.33	11.82
lagr1.d	3669	124735	4.57	17.60	7.04	11.22	11.78
All lagr1.*			17.2	68.0	31.1	44.5	41.0
lagr2.a	17127	1060154	94.82	202.43	321.86	384.11	169.53
lagr2.b	17127	1278550	138.43	287.23	378.80	509.18	244.29
lagr2.c	17127	1308213	119.38	289.43	258.37	574.24	225.50
lagr2.d	17127	1310269	106.55	278.87	399.79	474.30	221.91
lagr2.e	17127	1310400	111.17	305.25	318.62	512.91	218.29
lagr2.f	17127	1310400	97.28	277.62	519.52	674.74	210.79
lagr2.g	17127	1313161	84.42	291.95	472.09	868.97	218.43
lagr2.h	17127	1313161	88.66	360.60	348.03	561.85	226.62
All lagr2.*			840.7	2293.4	3017.1	4560.3	1735.3
lagr3.a	42007	254082	128.17	91.13	216.83	443.29	190.64
lagr3.b	42007	1195791	314.99	315.87	2456.13	4923.19	556.42
lagr3.c	42007	1469435	366.13	421.35	1674.40	2507.09	782.55
lagr3.d	42007	1518992	429.43	458.43	1650.13	3613.42	541.93
lagr3.e	42007	1359195	343.17	360.52	934.99	2394.21	518.84
lagr3.f	42007	1359202	366.25	383.78	956.56	2537.08	509.09
All lagr3.*			1948.1	2031.1	7889.0	16418.3	3099.5

Table 4: Cpu times (sec) for the Lagrangean relaxation subproblems.

Prob.	$\lvert V \rvert$	$\lvert A \rvert$	MCF		CS2-3.4	RELAX-IV	
			column gener.	default		default	auction
dep1	8193	7224283	178.00	1301.51	1506.60	$>$ 17000	3027.38
dep2	4583	2238585	49.00	197.31	377.63	553.62	1045.57
dep3	1505	216483	3.27	10.50	22.45	13.34	32.06
dep4	4475	2008034	22.27	225.44	269.47	205.30	705.08
dep5	1297	175533	3.72	4.77	16.92	6.84	10.25
dep6	6577	4634374	90.77	645.37	740.38	689.00	1685.02
dep7	1391	200757	4.42	8.70	21.65	12.54	20.23
dep8	1507	217058	3.10	9.86	24.08	10.96	28.94
dep9	5021	2577951	46.17	315.76	381.35	571.58	1501.78
dep10	2731	771481	14.76	35.79	100.22	69.18	239.11
dep11	1583	270603	5.22	10.47	28.90	18.93	51.33
dep12	2923	921440	19.90	60.83	131.07	152.41	467.07
dep13	4567	2209154	42.27	340.89	321.13	268.67	894.81
dep14	2139	467469	7.65	18.07	53.03	34.02	78.65
dep15	1503	227876	5.99	9.34	29.53	11.12	37.29
dep16	2955	865660	10.47	60.60	113.47	126.63	114.92
dep17	1559	252055	3.63	11.55	31.37	23.49	30.05
dep18	2667	694181	13.84	37.81	100.12	158.74	221.14
dep19	1701	288994	3.83	13.32	29.10	8.55	10.01
dep20	3827	1415590	23.49	154.97	186.72	265.48	145.78
dep21	3103	952824	24.04	62.48	124.40	232.29	424.37
dep22	3841	1447401	17.46	126.88	206.15	304.96	121.56
dep23	1879	352459	5.35	13.10	40.32	12.86	9.29
dep24	5237	2688675	41.77	376.51	380.15	$>$ 200000	634.60
dep25	3121	967201	12.57	73.93	126.98	151.69	48.27
dep26	3995	1640499	50.92	159.35	206.28	145.63	584.44
dep27	1583	259148	3.96	11.19	26.65	10.51	39.69
dep28	1503	226089	3.71	9.89	25.80	17.35	14.62
dep29	2739	778733	21.76	51.62	85.38	57.97	255.27
dep30	1333	183727	3.42	6.95	20.02	6.08	24.77
dep31	2183	468763	9.51	27.62	51.20	52.49	109.12
All dep1, dep2, etc.			746.24	4392.38	5778.52	—	12612.5

Table 5: Cpu times (sec) for the complete single-depot vehicle scheduling problems.

Prob.	$\|V\|$	$\|A\|$	MCF default	CS2-3.4	RELAX-IV default	RELAX-IV auction	CPLEX NETOPT
dep1.a	8193	442381	37.55	60.73	138.27	33.06	44.73
dep1.b	8193	535088	21.29	101.42	344.68	50.87	82.81
dep2.a	4583	133394	7.07	18.83	19.12	6.80	10.50
dep2.b	4583	187140	10.34	32.40	15.12	43.46	18.16
dep3.a	1505	10915	0.30	0.73	0.59	0.37	0.32
dep3.b	1505	36915	0.87	3.23	0.76	2.67	1.59
dep4.a	4475	63442	3.89	7.23	8.71	3.36	6.53
dep4.b	4475	101816	4.58	13.32	7.46	16.61	8.02
dep5.a	1297	14136	0.21	0.90	0.16	0.48	0.58
dep5.b	1297	41077	0.96	3.43	0.81	1.05	3.03
dep6.a	6577	293812	15.70	41.63	68.07	14.47	25.08
dep6.b	6577	374541	22.62	60.57	65.73	76.51	55.44
dep7.a	1391	17778	0.36	1.17	0.36	0.51	0.67
dep7.b	1391	47822	1.52	4.63	2.08	1.84	3.20
dep8.a	1507	10967	0.24	0.68	0.59	0.19	0.53
dep8.b	1507	36944	0.97	3.17	1.36	2.39	3.02
dep9.a	5021	100517	3.75	12.58	26.33	4.14	10.71
dep9.b	5021	150134	6.69	23.20	24.87	26.83	13.09
dep10.a	2731	56988	2.27	6.47	2.36	2.27	3.76
dep10.b	2731	88376	3.48	11.77	4.53	5.66	7.40
dep11.a	1583	22865	0.46	1.77	1.24	0.42	1.04
dep11.b	1583	51478	1.19	5.53	0.85	1.86	4.17
dep12.a	2923	63693	2.17	7.02	7.59	3.58	4.28
dep12.b	2923	98188	3.55	12.78	7.65	10.48	10.08
dep13.a	4567	125596	5.12	12.93	4.33	4.29	11.33
dep13.b	4567	162640	9.07	23.22	10.39	26.53	14.29
dep14.a	2139	8454	0.21	0.88	0.35	1.10	0.50
dep14.b	2139	50086	1.52	5.70	2.23	2.24	3.66
dep15.a	1503	4489	0.16	0.57	0.52	0.35	0.34
dep15.b	1503	49035	1.49	5.83	1.87	3.11	3.93
dep16.a	2955	10623	0.39	1.37	1.98	0.44	0.85
dep16.b	2955	70831	2.85	9.45	6.54	16.10	5.57

Table 6: Cpu times (sec) for the single-depot vehicle scheduling subproblems.

Prob.	$\lvert V \rvert$	$\lvert A \rvert$	MCF default	CS2-3.4	RELAX-IV		CPLEX NETOPT
					default	auction	
dep17.a	1559	5472	0.13	0.65	0.79	0.34	0.29
dep17.b	1559	38344	1.24	4.22	2.47	2.33	2.61
dep18.a	2667	15413	0.59	1.93	2.10	1.56	1.17
dep18.b	2667	88903	5.27	12.65	9.85	10.91	9.48
dep19.a	1701	4678	0.09	0.57	0.80	0.16	0.38
dep19.b	1701	37312	1.05	3.65	0.53	1.96	2.41
dep20.a	3827	17818	0.83	2.63	2.28	1.79	1.75
dep20.b	3827	109890	5.73	16.45	7.76	10.24	10.50
dep21.a	3103	13187	0.39	1.93	2.17	1.72	0.98
dep21.b	3103	112244	4.24	15.60	21.16	46.37	10.27
dep22.a	3841	15219	0.85	2.15	3.21	1.83	1.47
dep22.b	3841	85235	4.59	14.17	10.47	7.20	9.90
dep23.a	1879	7958	0.15	1.03	1.58	0.86	0.35
dep23.b	1879	52901	1.69	7.07	1.77	1.09	4.64
dep24.a	5237	19553	1.30	3.17	2.74	2.98	2.72
dep24.b	5237	110732	5.10	18.50	13.05	16.46	12.38
dep25.a	3121	12515	0.48	1.52	2.07	1.05	1.01
dep25.b	3121	72895	2.16	9.87	9.78	5.16	5.84
dep26.a	3995	17556	0.83	2.42	3.11	1.50	1.79
dep26.b	3995	173676	8.53	28.15	6.51	79.21	21.48
dep27.a	1583	4073	0.14	0.52	0.54	0.38	0.22
dep27.b	1583	36873	1.16	4.20	3.20	2.70	2.91
dep28.a	1503	4935	0.14	0.58	0.64	0.34	0.34
dep28.b	1503	38605	1.15	3.88	1.82	4.05	3.25
dep29.a	2739	10688	0.37	1.28	0.63	0.56	0.70
dep29.b	2739	131122	5.74	18.60	4.69	34.52	14.91
dep30.a	1333	3274	0.07	0.43	0.24	0.17	0.18
dep30.b	1333	48218	1.56	4.62	1.41	1.56	3.41
dep31.a	2183	6313	0.22	0.77	0.69	0.39	0.49
dep31.b	2183	66468	3.31	8.42	2.84	11.06	6.21
All dep*.a and dep*.b			231.9	686.7	898.4	614.5	493.3

Table 7: Cpu times (sec) for the single-depot vehicle scheduling subproblems.

Appendix B Klingman et al. Networks

No.	$\lvert V \rvert$	Sources	Sinks	$\lvert A \rvert$	Cost Range		Total Supply	Transshipments		Percent of High Cost	Percent of Cap. Arcs	Upper Bound Range	
					Min	Max		Sources	Sinks			Min	Max
1	200	100	100	1300	1	10000	100000	0	0	0	0	0	0
2	200	100	100	1500	1	10000	100000	0	0	0	0	0	0
3	200	100	100	2000	1	10000	100000	0	0	0	0	0	0
4	200	100	100	2200	1	10000	100000	0	0	0	0	0	0
5	200	100	100	2900	1	10000	100000	0	0	0	0	0	0
6	300	150	150	3150	1	10000	150000	0	0	0	0	0	0
7	300	150	150	4500	1	10000	150000	0	0	0	0	0	0
8	300	150	150	5155	1	10000	150000	0	0	0	0	0	0
9	300	150	150	6075	1	10000	150000	0	0	0	0	0	0
10	300	150	150	6300	1	10000	150000	0	0	0	0	0	0
11	400	200	200	1500	1	10000	200	0	0	0	0	0	0
12	400	200	200	2250	1	10000	200	0	0	0	0	0	0
13	400	200	200	3000	1	10000	200	0	0	0	0	0	0
14	400	200	200	3750	1	10000	200	0	0	0	0	0	0
15	400	200	200	4500	1	10000	200	0	0	0	0	0	0
16	400	8	60	1306	1	10000	400000	0	0	30	20	16000	30000
17	400	8	60	2443	1	10000	400000	0	0	30	20	16000	30000
18	400	8	60	1306	1	10000	400000	0	0	30	20	20000	120000
19	400	8	60	2443	1	10000	150000	0	0	30	20	20000	120000
20	400	8	60	1416	1	10000	400000	5	50	30	40	16000	30000
21	400	8	60	2836	1	10000	400000	5	50	30	40	16000	30000
22	400	8	60	1416	1	10000	400000	5	50	30	40	20000	120000
23	400	8	60	2836	1	10000	400000	5	50	30	40	20000	120000
24	400	4	12	1382	1	10000	400000	0	0	30	80	16000	30000
25	400	4	12	2676	1	10000	400000	0	0	30	80	16000	30000
26	400	4	12	1382	1	10000	400000	0	0	30	80	20000	120000
27	400	4	12	2676	1	10000	400000	0	0	30	80	20000	120000
28	1000	50	50	2900	1	10000	1000000	0	0	0	0	0	0
29	1000	50	50	3400	1	10000	1000000	0	0	0	0	0	0
30	1000	50	50	4400	1	10000	1000000	0	0	0	0	0	0
31	1000	50	50	4800	1	10000	1000000	0	0	0	0	0	0
32	1500	75	75	4342	1	10000	1500000	0	0	0	0	0	0
33	1500	75	75	4385	1	10000	1500000	0	0	0	0	0	0
34	1500	75	75	5107	1	10000	1500000	0	0	0	0	0	0
35	1500	75	75	5730	1	10000	1500000	0	0	0	0	0	0
36	8000	200	1000	15000	1	10000	4000000	100	300	0	0	30	30
37	5000	150	800	23000	1	10000	4000000	50	100	0	0	0	0
38	3000	125	500	35000	1	10000	2000000	25	50	0	0	0	0
39	5000	180	700	15000	1	10000	4000000	100	300	0	1	3000	5000
40	3000	100	300	23000	1	10000	2000000	50	100	0	1	2000	4000

Table 8: NETGEN input parameters for the Klingman et al. networks.

No.	MCF default	CS2-3.4	RELAX-IV default	RELAX-IV auction	CPLEX NETOPT
1	0.01	0.08	0.03	0.03	0.04
2	0.02	0.12	0.03	0.03	0.06
3	0.02	0.12	0.05	0.04	0.06
4	0.02	0.13	0.04	0.06	0.08
5	0.02	0.12	0.05	0.08	0.08
6	0.04	0.23	0.08	0.08	0.15
7	0.05	0.25	0.10	0.09	0.18
8	0.07	0.30	0.10	0.11	0.20
9	0.07	0.37	0.12	0.16	0.24
10	0.09	0.35	0.16	0.12	0.25
11	0.04	0.07	0.02	0.03	0.08
12	0.03	0.13	0.05	0.06	0.15
13	0.05	0.13	0.04	0.06	0.16
14	0.06	0.13	0.02	0.07	0.20
15	0.06	0.17	0.13	0.10	0.23
16	0.02	0.10	0.03	0.03	0.06
17	0.03	0.13	0.03	0.03	0.09
18	0.03	0.08	0.04	0.04	0.05
19	0.02	0.12	0.06	0.05	0.10
20	0.02	0.10	0.04	0.04	0.07
21	0.03	0.13	0.07	0.09	0.12
22	0.03	0.10	0.03	0.03	0.06
23	0.03	0.13	0.06	0.04	0.09
28	0.06	0.23	0.11	0.10	0.15
29	0.07	0.23	0.13	0.15	0.21
30	0.08	0.30	0.08	0.24	0.27
31	0.09	0.32	0.11	0.17	0.22
32	0.12	0.35	0.17	0.14	0.29
33	0.12	0.37	0.38	0.23	0.36
34	0.13	0.37	0.14	0.20	0.38
35	0.14	0.42	0.18	0.44	0.48
36	2.16	4.43	2.94	4.05	5.84
37	1.75	4.08	3.05	2.52	5.03
38	2.13	3.92	1.52	2.00	4.69
39	1.45	2.73	2.90	1.77	3.21
40	0.98	2.18	2.23	1.65	2.66
$\sum$	10.22	23.88	15.45	15.23	26.91

Table 9: Cpu times (sec) for the Klingman et al. networks.

Appendix C Capacitated NETGEN Networks

No.	$\|V\|$	Sources	Sinks	$\|A\|$	Cost Range		Total Supply	Transshipments		Percent of High Cost	Percent of Cap. Arcs	Upper Bound Range	
					Min	Max		Sources	Sinks			Min	Max
1	1000	15	15	10000	1	10000	500000	0	0	30	80	20000	120000
2	1000	15	15	30000	1	10000	500000	0	0	30	80	20000	1200000
3	1000	15	15	40000	1	10000	500000	0	0	30	80	200000	1200000
4	5000	40	60	30000	1	10000	600000	0	0	40	90	30000	150000
5	5000	40	60	40000	1	10000	600000	0	0	40	90	30000	150000
6	5000	40	60	50000	1	10000	600000	0	0	40	90	30000	150000
7	5000	40	60	60000	1	10000	600000	0	0	40	90	30000	150000
8	10000	100	100	40000	1	10000	1000000	0	0	30	80	30000	150000
9	10000	100	100	50000	1	10000	1000000	0	0	30	80	30000	150000
10	10000	100	100	70000	1	10000	1000000	0	0	30	80	30000	150000
11	10000	100	100	80000	1	10000	1000000	0	0	30	80	30000	150000
12	10000	100	100	90000	1	10000	1000000	0	0	30	80	30000	150000
13	1000	15	15	10000	1	10000	500000	0	0	30	80	200	12000
14	1000	15	15	30000	1	10000	500000	0	0	30	80	200	12000
15	1000	15	15	40000	1	10000	500000	0	0	30	80	200	12000
16	5000	40	60	30000	1	10000	600000	0	0	40	90	300	15000
17	5000	40	60	40000	1	10000	600000	0	0	40	90	300	15000
18	5000	40	60	50000	1	10000	600000	0	0	40	90	300	15000
19	5000	40	60	60000	1	10000	600000	0	0	40	90	300	15000
20	10000	100	100	40000	1	10000	1000000	0	0	30	80	300	5000
21	10000	100	100	50000	1	10000	1000000	0	0	30	80	300	15000
22	10000	100	100	60000	1	10000	1000000	0	0	30	80	300	15000
23	10000	100	100	70000	1	10000	1000000	0	0	30	80	300	15000
24	10000	100	100	80000	1	10000	1000000	0	0	30	80	300	15000
25	10000	100	100	90000	1	10000	1000000	0	0	30	80	300	15000
26	1000	15	15	10000	1	10000	20000	0	0	30	80	20	1200
27	1000	15	15	30000	1	10000	20000	0	0	30	80	20	1200
28	1000	15	15	20000	1	10000	20000	0	0	30	80	20	1200
29	5000	40	60	30000	1	10000	20000	0	0	40	90	30	1500
30	5000	40	60	40000	1	10000	20000	0	0	40	90	30	1500
31	5000	40	60	50000	1	10000	20000	0	0	40	90	30	1500
32	5000	40	60	60000	1	10000	20000	0	0	40	90	30	1500
33	10000	100	100	40000	1	10000	30000	0	0	30	80	30	1500
34	10000	100	100	50000	1	10000	30000	0	0	30	80	30	1300
35	10000	100	100	60000	1	10000	30000	0	0	30	80	30	1300
36	10000	100	100	70000	1	10000	30000	0	0	30	80	30	1500
37	10000	100	100	80000	1	10000	30000	0	0	30	80	30	1500
38	10000	100	100	90000	1	10000	30000	0	0	30	80	30	1500
39	10000	200	200	100000	1	10000	70000	0	0	30	80	500	10000
40	10000	200	200	120000	1	10000	70000	0	0	30	80	500	10000
41	10000	200	200	140000	1	10000	70000	0	0	30	80	500	10000

Table 10: NETGEN input parameters for the capacitated networks.

No.	MCF	CS2-3.4	RELAX-IV		CPLEX
	default		default	auction	NETOPT
1	0.14	0.45	0.14	0.27	0.47
2	0.37	1.45	0.39	1.15	1.22
3	0.70	2.10	0.55	0.85	2.08
4	1.35	2.25	1.24	2.54	3.45
5	1.64	3.40	2.35	1.83	4.26
6	2.27	3.97	2.01	6.15	5.44
7	2.71	4.72	4.79	3.04	6.33
8	3.21	4.95	2.96	3.55	6.88
9	4.15	5.85	2.94	4.43	8.56
10	5.03	7.37	6.17	4.01	12.93
11	6.22	8.23	11.23	5.09	11.99
12	5.88	9.28	4.55	3.93	13.78
13	0.26	0.68	0.46	0.36	0.84
14	0.80	2.02	1.01	1.63	2.50
15	1.23	2.98	1.50	0.98	3.69
16	2.44	3.35	1.49	2.19	6.12
17	2.91	4.40	2.11	2.72	7.77
18	3.68	5.43	2.14	3.00	10.35
19	4.78	6.08	3.19	5.45	13.82
20	5.23	7.13	14.05	3.87	14.06
21	6.57	7.58	21.53	4.80	16.02
22	7.65	8.33	3.61	8.25	17.55
23	9.93	9.52	5.58	6.59	23.16
24	10.04	11.08	5.50	9.38	25.33
25	11.07	12.87	6.58	6.93	30.28
26	0.22	0.53	0.15	0.17	0.64
27	0.73	1.87	0.51	0.68	2.06
28	1.11	2.70	0.87	1.13	3.03
29	1.53	2.58	4.36	1.56	3.89
30	2.14	3.75	2.73	2.15	5.30
31	2.86	4.45	2.53	2.41	7.16
32	3.33	5.20	4.47	2.08	9.12
33	3.74	5.68	2.72	2.72	8.99
34	4.56	7.07	2.81	3.43	10.92
35	5.01	6.98	4.47	3.10	13.82
36	6.71	8.87	3.53	4.07	15.35
37	7.83	10.52	3.80	4.27	19.72
38	7.59	10.25	5.68	5.32	21.15
39	9.73	12.67	7.90	6.51	21.64
40	11.83	15.00	20.03	14.52	24.42
41	14.11	16.55	10.66	13.32	28.72
$\sum$	183.3	250.1	185.3	160.7	444.8

Table 11: Cpu times (sec) for the capacitated NETGEN networks.

Appendix D Transportation NETGEN Networks

No.	$\|V\|$	Sources	Sinks	$\|A\|$	Cost Range		Total Supply	Transshipments		Percent of High Cost	Percent of Cap. Arcs	Upper Bound Range	
					Min	Max		Sources	Sinks			Min	Max
1	800	400	400	10000	1	10000	200000	0	0	0	0	0	0
2	800	400	400	20000	1	10000	200000	0	0	0	0	0	0
3	800	400	400	30000	1	10000	200000	0	0	0	0	0	0
4	800	400	400	40000	1	10000	200000	0	0	0	0	0	0
5	1000	500	500	20000	1	10000	200000	0	0	0	0	0	0
6	1000	500	500	30000	1	10000	200000	0	0	0	0	0	0
7	1000	500	500	40000	1	10000	200000	0	0	0	0	0	0
8	1000	500	500	50000	1	10000	200000	0	0	0	0	0	0
9	400	200	200	10000	1	10000	200000	0	0	0	0	0	0
11	600	300	300	10000	1	10000	200000	0	0	0	0	0	0
12	600	300	300	20000	1	10000	200000	0	0	0	0	0	0
13	600	300	300	30000	1	10000	200000	0	0	0	0	0	0
14	600	300	300	30000	1	10000	200000	0	0	0	0	0	0

Table 12: NETGEN input parameters for the transportation networks.

No.	MCF default	CS2-3.4	RELAX-IV		CPLEX NETOPT
			default	auction	
1	0.32	0.75	0.45	0.37	0.77
2	0.59	1.50	0.48	0.86	1.54
3	0.69	2.47	1.39	1.23	2.62
4	0.90	3.72	1.52	1.68	3.10
5	0.68	1.52	0.77	0.70	1.82
6	1.11	2.90	1.40	1.39	2.87
7	1.43	3.77	1.70	1.86	3.64
8	1.71	5.00	2.00	2.10	4.69
9	0.15	0.63	0.22	0.22	0.41
11	0.27	0.72	0.34	0.36	0.63
12	0.44	1.27	0.46	0.68	1.20
13	0.67	2.52	0.89	0.84	1.84
14	1.06	3.10	0.87	1.98	2.58
$\sum$	10.0	29.9	12.5	14.3	27.7

Table 13: Cpu times (sec) for the transportation NETGEN networks.