February 2007

Progress in solving large scale multi-depot multi-vehicle-type bus scheduling problems with integer programming

Uwe H. Suhl
Freie Universität Berlin, suhl@wiwiss.fu-berlin.de

Swantje Friedrich
Freie Universität Berlin

Veronika Waue
Freie Universität Berlin

In: Oberweis, Andreas, u.a. (Hg.) 2007. *eOrganisation: Service-, Prozess-, Market-Engineering*; 8. Internationale Tagung Wirtschaftsinformatik 2007. Karlsruhe: Universitätsverlag Karlsruhe

ISBN: 978-3-86644-094-4 (Band 1)
ISBN: 978-3-86644-095-1 (Band 2)
ISBN: 978-3-86644-093-7 (set)

Progress in solving large scale multi-depot multi-vehicle-type bus scheduling problems with integer programming

Uwe H. Suhl, Swantje Friedrich and Veronika Waue

Institut für Produktion, Wirtschaftsinformatik und Operations Research

Freie Universität Berlin
D-14195 Berlin

suhl@wiwiss.fu-berlin.de

Abstract

This paper discusses *solution methods* of multi-depot, multi-vehicle-type bus scheduling problems (MDVSP), involving multiple depots for vehicles and different vehicle types for timetabled trips. All models are from real-life applications. Key elements of the application are a mixed-integer model based on time-space-based networks for MDVSP modeling and a customized version of the mathematical optimization system MOPS to solve the very large mixed integer optimization models. The modeling approach was already described in several other publications. This paper focuses on the solution methods critical to solve the very large integer optimization models. We discuss aspects to solve the initial LP and the selection of the starting heuristic. Real life applications with over one million integer variables and about 160000 constraints were solved by the optimizer MOPS. A key role plays also the architecture of new Windows workstations with Intel 64 bit processors. We present numerical results on some large scale models and a brief comparison to another state-of-the-art mathematical programming system. The presented research results have been developed in cooperation with the Decision Support & OR Lab of the university Paderborn which was responsible for the development of the MDVSP-model and the application software.

1 Introduction

Public transportation companies are under high cost pressures, because market prices for suburban transportation are not cost covering and traditional subsidies from municipalities are decreasing. The companies must therefore focus on efficient use of resources, especially vehicles and drivers.

We consider the scheduling of vehicles under constraints and objectives arising in urban and suburban public transport. Each timetabled trip can be served by a vehicle belonging to a given set of vehicle types. Each vehicle has to start and end its work day in one of the given depots. After serving one timetabled (loaded) trip, each bus can serve one of the trips starting later from the station where the vehicle is standing, or it can change its location by moving unloaded to an another station (deadhead trip) in order to serve the next loaded trip starting there.

The cost components include fixed costs for required vehicles as well as variable operational costs. The variable costs consist of distance-dependent travel costs and time-dependent costs for time spent outside the depot – the case where a driver is obliged to stay with the bus. All cost components depend on vehicle type. Since the fixed vehicle cost components are usually orders of magnitude higher than the operational costs, the optimal solution always involves the minimal number of vehicles.

The combinatorial complexity of the multi-depot bus scheduling problem (MDVSP) is determined by numerous possibilities to assign vehicle type to each trip, to build sequences of trips for particular buses, and to assign buses to certain depots. To represent these sequences of trips, exact modeling approaches known in the literature consider explicitly all possible connections - pairs of trips that can be served successively.

It is well known that the general MDVSP-model is NP-hard [BeCG87]. The practical complexity of instances of the MDVSP depends on factors such as:

- the number of timetabled trips,
- the number of depots, or more precisely, the average number of depot-vehicle type combinations per timetabled trip,

- the number of possible unloaded trips, which can vary depending on the completeness of the distance matrix for stop points.

Since real life MDVSP-models result in a very large number of integer variables and constraints there have been numerous modeling and optimization approaches to solve such models. There are three different approaches among the existing modeling techniques:

- Path-oriented - leading to set partitioning formulation [RiSo94],
- Arc-oriented - leading to multi-commodity flow formulation [FoHW94],
- Combinations of these two approaches [CDFT89].

For an overview of the various modeling and optimization techniques see [KlMS06].

In all these models the possible trip connections are considered explicitly and the number of such connections, corresponding to the number of integer variables, grows quadratic as a function of the number of loaded trips. Therefore, models with several thousand scheduled trips become too large to be solved in a reasonable amount of time by standard integer programming software. Various techniques to reduce the number of possible connections have been proposed in the literature. Some approaches discard arcs with too long waiting times; others generate arcs applying the column generation idea to the network flow representation.

The model used in this paper is based on a time-space network based modeling approach described in [MelKl02, GiKS05, KlMS06]. For completeness we will describe in the following section the underlying mathematical model in its simplest form.

2 The multi-depot multi-vehicle type scheduling problem (MDVSP)

We define the vehicle scheduling problem (VSP), arising in public bus transportation, as the task of building an optimal set of rotations (vehicle schedule), such that each trip of a given timetable is covered by exactly one rotation. For each trip the timetable specifies a departure time and an arrival time with start and end stations respectively.

Within a bus tour consisting of several (loaded) service trips chained with each other, the use of deadhead trips (unloaded trips between two end stations) often provides an improvement in

order to serve all trips of a given timetable by a minimum number of buses. Thus a work day for a given bus is defined as a sequence of trips, deadheads, waiting times at stations (parking stops) and pull-out/pull-in trips from/to the assigned depot. Since deadhead trips mean an additional cost factor, minimization of this cost and minimization of waiting time cost are important optimization goals.

There are several variations of the bus scheduling problem involving different side constraints or numbers of depots and / or of bus types. The constraints and optimization criteria may differ from one problem setting to another. The presented model can be modified such that several practical side constraints such as outsourcing of the parts of timetables to private bus companies or return trips to different depots can be handled.

The multi-depot vehicle scheduling problem involves several depots, so that a vehicle has to return in the evening to the same depot from which it started in the morning.

A multi-vehicle-type VSP copes with a heterogeneous fleet of vehicles. For a given trip we define a group of vehicle types this trip can be served by. In a feasible solution each rotation is assigned to exactly one depot and to one vehicle type. Furthermore, it is possible to state capacity constraints for depots to consider the number of parking slots for buses. Other kinds of capacity constrains set a limit for the number of available vehicles of certain bus types and for the number of certain type vehicles in a given depot.

Time-space network (TSN) models have been proposed for routing problems in airline scheduling [HBJM95], because they are advantageous in modeling possible connections between arriving and departing flights. In a time-space network, connections within a location are realized by using a time line that connects all possible landing and takeoff events within the location. Thus, there is no need to explicitly model connections for each feasible pair of events within a location. Time-space network models were not used for bus scheduling problems until now, because, compared to airline scheduling where deadheading is generally not allowed, bus scheduling permits unrestricted deadheading. Thus the advantages given by TSN remained negligible, because of too many deadhead arcs.

However, as was shown in [MelKl02, GiKS05, KlMS06] a new modeling technique exploits the advantages of TSN models for bus scheduling problems. A crucial modeling technique is aggregation of possible trip matches, which allows a drastic reduction in model size. This

modeling technique allows the solution of large practical MDVSP models with exact solution algorithms such as integer programming.

Let N = {1,2,…,n} be the set of trips, and let D be the set of depots. The depot is here defined as a combination of a depot and a vehicle type. The vehicle scheduling network $G^d = (V^d, A^d)$ corresponding to depot d is defined as an acyclic directed graph with nodes V^d and arcs A^d. Let c_{ij}^d be the vehicle cost of arc $(i,j) \in A^d$, which can be a function of travel and idle time. The vehicle cost of arcs representing idle time activity in the depot is 0. Furthermore, a fixed cost for using a vehicle is set on the circulation arc. Let $N^d(t) \in A^d$ be the arc corresponding to trip t in the vehicle scheduling network G^d. Decision variable x_{ij}^d indicates whether an arc (i,j) is used and assigned to the depot d or not. An upper bound u_{ij}^d is defined for each decision variable as follows:

$$u_{ij}^d = \begin{cases} 1, \text{ if } x_{ij}^d \text{ corresponds to a timetable trip} \\ u^d, \text{ if } x_{ij}^d \text{ corresponds to a circulation arc where } u^d \text{ is the capacity for depot d} \\ M, \text{ otherwise, where M is the maximum number of available vehicles.} \end{cases}$$

The MDVSP model can now be formulated as

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

$$\sum_{\{j:(i,j) \in A^d\}} x_{ij}^d - \sum_{\{j:(j,i) \in A^d\}} x_{ji}^d = 0, \quad \forall i \in V^d, \forall d \in D \tag{2}$$

$$\sum_{d \in D, (i,j) \in N^d(n)} x_{ij}^d = 1, \qquad \forall n \in N \tag{3}$$

$$0 \leq x_{ij}^d \leq u_{ij}^d, \qquad \forall (i,j) \in A^d, \forall d \in D \tag{4}$$

$$x_{ij}^d \text{ integer}, \qquad \forall (i,j) \in A^d, \forall d \in D \tag{5}$$

The objective (1) is to minimize the sum of total vehicle costs. Constraints (2) are flow conservation constraints, indicating that the flow into each node equals the flow out of each

node, while constraints (3) assure that each trip must be covered by exactly one vehicle. This is a time-space network model based on a multi-commodity flow formulation. It should be mentioned that some integer variables in the MDVSP model can be declared as continuous variables. They obtain automatically integer values in a feasible integer solution.

3 Solving the Linear Programming relaxation of MDVSP-models

Real life MDVSP-models are usually very large. The following table represents real life models of the PTV AG and the Decision Support & OR Lab of the university Paderborn. The number of depots is actually depot-vehicle-type combinations, as mentioned above, and thus corresponds to the terminology in [Löbe99].

MDVSP model instances					**Integer model sizes**			
City	#trips	#stations	#v. types	#depots	rows	variables	Int. vars.	nonzeros
Halle1	2047	21	3	2	14997	53249	40387	113069
Halle2	2047	21	3	3	21939	87128	67367	184111
Halle3	2047	21	3	4	29031	118768	91957	250675
Mun1	1808	76	1	19	52303	478823	429088	981163
Mun2	3054	49	1	9	61254	573300	515530	1174095
Mun3	11062	161	12	19	163142	1479833	1330580	3031285

Table 1: MDVSP model instances and corresponding integer model sizes

All successful solution methods to solve general integer optimization methods are based on a branch-and-bound / cut approach. A key role plays the solution of the LP-relaxations: the initial LP and the LPs corresponding to subproblems (nodes) in the branch-and-bound / cut tree.

There are three competitive solution algorithms to solve general LP-models [Bixb02]:

- primal simplex method, the oldest simplex solution algorithm [Dant63]
- dual simplex method, which has become a strong contender over the years [Lemk54, Bixb02, Kobe05, KoSu07]
- Interior point algorithms, also called barrier algorithms [Karm84, Mehr92, Mész96].

It is well known, that there are problem classes where each of those algorithms works best. Furthermore each method has fundamental advantages and disadvantages:

A fundamental advantage of the simplex methods is that an optimal LP solution is a basic solution. This means that only basic variables may have values between their bounds. Nonbasic variables (except free variables) have values at their lower or upper bound. Most lower bounds of practical models are zero. Therefore, an optimal basic solution has typically a smaller number of nonzero activities than an optimal solution produced by an interior point method. A basic solution exploits furthermore a tight linear programming relaxation of an integer model. The simplex method has in addition very good warm start capabilities if an LP-model is slightly modified and a nearly optimal basic solution is available. The simplex method is therefore the method of choice for solving LPs during branch and bound / cut algorithms. State-of-the-art dual simplex codes are in general superior to primal simplex codes. However both codes are dependant on each other that is the dual requires frequently the primal to remove a cost perturbation and the primal requires frequently the dual simplex code to remove a bound perturbation [KoSu07].

Recent interior point technology is based on primal-dual methods [Mehr92, Mész96]. There are large LP-models which can be solved much faster than with the best simplex codes. The number of iterations for an interior point method is typically relatively small (20-80) and independent of the size of the problem. The main work of an iteration is the solution of a symmetric, positive definite system of linear equations. A symbolic Cholesky factorization can be computed once by using an ordering algorithm [GeLi81]. The number of nonzeros in the Cholesky factorization is a key factor for the performance of the interior point method and is strongly influenced by the ordering algorithm used to compute the pivot sequence. It is therefore important, in particular for very large LP-models, to experiment with the different ordering heuristics for a given model class. The Cholesky factorization has in general much more nonzeros than LU-factorizations in a simplex type algorithm. The use of an interior point method therefore requires a much larger amount of main memory than the simplex method. This behavior rules out the use of interior point methods for very large models on some system platforms such as a classical Windows XP-system with 2 GB address limit (see Table 2).

Another disadvantage is a very limited warm start capability which is required during branch-and-bound / cut because the similarity of LP subproblems can be exploited by the simplex method but not by the interior point method.

Interior point methods do generally not produce a basic solution. In many situations (integer programming, tight linear programming relaxations, save / restore of basis) basic solutions are necessary. There are “cross over” algorithms, also called (optimal) basis identification which can be used to produce an optimal basic solution from an optimal interior point solution [Ande99]. These algorithms are specialized simplex algorithms. The crossover method used in MOPS uses the numerical kernels of the simplex method.

The optimization system MOPS [MOPS06, Suhl94] contains three state-of-the-art engines which were tested on the MDVSP models. The interior point method in MOPS is based on the work of C. Mészáros [Mész98]. The dual simplex method was recently completely new designed and implemented [Kobe05, KoSu07] and is one of the best implementation according to our benchmarks. A key question was initially which engine is best suited to solve the initial LP. It was clear from the beginning that the primal simplex method will probably not be competitive to the other methods on the MDVSP problems. The numerical experiments with the smallest model Halle1 in Table 1 shows the expected results (see Table 2). We ran a comparison against Cplex 9.1 [ILOG06] and it shows the same behavior. As a consequence the larger models were only tested with dual simplex and barrier with crossover (x-over). The smallest models Halle1 and Halle2 were solved faster with the dual simplex. When model sizes get larger the barrier code outperforms the dual simplex. This observation is in line with the fundamental advantage in computational complexity of the barrier code compared to a simplex code. The following numerical results are based on a typical Windows XP Workstation with a 32-Bit Pentium Processor. This machine has a maximum virtual address space limit of 2 Gigabytes (GB). Under certain conditions an address limit of 3 GB is possible. This type of workstation is frequently used in practice for running such applications. As can be seen in Table 2 model Mun3 cannot be solved with the barrier code on such a machine, because the required virtual memory exceeds 3 GB. We were also not able to solve that model with the 32-Bit barrier code of Cplex 9.1 on this machine.

Name	MOPS times (secs) on PIV (3,4 GHz, WinXP) to solve the initial LP		
	Primal simplex	Dual simplex	Barrier + x-over
Halle1	281.69	21.28	33.19
Halle2	Nt	61.38	73.06
Halle3	Nt	176.05	108.81
Mun1	Nt	3051.58	1519.33
Mun2	Nt	4266.36	798.24
Mun3	Nt	15012.08	nem

nt: not tested, nem: not enough (virtual) memory, i.e. > 2 GB

Table 2: LP Solution times on a 32 Bit Windows workstation with MOPS (32)

Model Mun3 was also solved by Cplex 9.1 on the same 32 bit workstation. The barrier code of Cplex 9.1 was also not able to solve this model due to insufficient memory. The dual simplex engine of Cplex 9.1 solved the initial LP of Mun3 in 14832.17 secs. The purpose of this paper is not to make a comparison between Cplex and MOPS. The test just shows that the current state-of-the-art system Cplex required also several hours computing to solve this model.

A recent development for Windows / Intel workstations are processor and memory architectures which allow 64 bit addressing and integer arithmetic. Microsoft provides the operating system WindowsXP (x64). Intel offers C++ and FORTRAN compilers which generate 64 bit code for such machines. This development is very important from a practical point of view because Intel / Windows system platforms are used predominantly in industry. Virtually all 32-bit software systems run unchanged on such 64 bit systems allowing the parallel use of 32 bit and 64 bit software systems.

It was a straightforward task to recompile MOPS using the Intel Compilers and generating a 64 bit library. The following numerical results with MOPS are based on a workstation with Intel Xeon processor (3.4 GHz) with Intel 64 bit memory technology 64MT, 4 GB of main memory. Both CPUs are Xeon Processors with a clock speed of 3.4 GHz. The internal data caches are identical with 16 KB. The 32 Bit CPU has an on board L2 cache with a size of 1 MB ECC whereas the 64 Bit CPU has an on board L2 cache with a size of 2 MB ECC resulting in a much higher memory bandwidth of the 64 Bit CPU. Furthermore the 64 Bit CPU has more registers and additional instructions. Despite of the same compiler releases one can expect some differences in the compiled code.

Name	MOPS (64) times (secs) on Xeon (3,4 GHz, Win x64 to solve initial LP	
	Dual simplex	Barrier + x-over
Mun1	2895.09	1496.81
Mun2	4106.36	778.24
Mun3	11336.45	3490.20

Table 3: LP Solution times on a 64 Bit Windows workstation with MOPS64

One surprise was the result with the dual on Mun3. The time was nearly 4000 secs faster than the result for the 32 bit version. It is not clear which of the possible influence factors (compiler, cache size and architecture) was responsible for this result.

A key influence on the running time of the interior point code has the ordering heuristic used for the Cholesky factorization. There are several well known ordering heuristics such as minimum degree, minimum local fill-in, and nested dissection [GeLi81] and more recent orderings such as multisection [AsLi98, Mész98] which are used in MOPS. In the default ordering we perform the minimum degree and the nested dissection ordering and compare the computed number of nonzeros; then we select the better ordering i.e. with the fewer number of nonzeros. However the best results for the MDVSP-models are based on the multisection ordering which was used throughout in the benchmarks. The following table contains a comparison of two ordering heuristics with respect to the number of nonzeros and solution times of three models of Table 1 on the Xeon (3.4) and Winx64. Note, that most other LP-models are solved faster with the default ordering.

Name	default ordering		multisection ordering	
	Nonzeros in Cholesky	Barrier time (sec)	Nonzeros in Cholesky	Barrier time (sec)
Mun1	63,648,566	3567.72	45,470,777	1496.81
Mun2	26,887,134	1206.81	22,688,525	778.24
Mun3	84,710,597	5689.22	71,388,781	3490.20

Table 4: LP Solution times and Cholesky nonzeros with the MOPS barrier code and two ordering heuristics

4 Solving the MDVSP-models

The time-space network based models presented above have automatically a very tight LP-relaxation. The relative gap between the value of the LP-relaxation after IP-Preprocessing and an optimal integer value is extremely small, sometimes zero. Almost all variables have integer values in the optimal basic solution of the LP-relaxation. Due to the aggregation of possible connections, the mathematical model tends to use one general integer variable instead of several

binary variables. The optimal vehicle schedule is computed in the post-processing phase from the optimal network flow via flow decomposition [KlMS06].

The normal IP-Preprocessing [SuSz94] to tighten the LP-relaxation does not produce any significant improvements. Neither lifted cover cuts [SuWa04] nor clique or implication cuts are violated in the LP-relaxation(s). Only Gomory mixed integer cuts are able to tighten the LP-relaxation on some MDVSP-models, reducing the fractionality of the LP-solution. However the Gomory cuts can be quite dense. The number of nonzeros depends on the density of row k of the inverse. Therefore inserting the cut may produce significant fill in the following LU-factorizations of the modified basis matrices reducing the iteration speed in the branch-and-bound / cut algorithm. Therefore the decision whether a cut is actually appended to the original model is crucial, in particular for very large models. This aspect is under further investigation and is not discussed here.

An initial heuristic is used to find good integer solutions quickly. MOPS contains different heuristics prior to the branch-and-cut algorithm. We use the relaxation-based search space (RSS) heuristic for solving the MDVSP-models which produces the overall best results.

The RSS heuristic distinguishes between basic and nonbasic integer variables of the current LP solution after the initial IP-Preprocessing. Let nb the number of nonbasic variables in the LP-solution and δ a parameter between 0 and 1 (default is 0.7). The δ*nb nonbasic variables with the largest magnitude of their reduced costs d_j are fixed to the corresponding lower or upper bound depending on the sign of d_j.

We define two rounding intervals [0,rl] and [ru,1] where $0 \leq rl < ru \leq 1$. The default values are rl = 0.1 and ru = 0.9. For a basic integer variable $j \in J_I$ with a value $\bar{x}_j = \lfloor \bar{x}_j \rfloor + f_j$, f_j specifies its fractional part, where $0 \leq f_j < 1$. Variable x_j is rounded to $\lfloor \bar{x}_j \rfloor$ if $f_j \in [0,rl]$ and to $\lceil \bar{x}_j \rceil$ if $f_j \in [ru,1]$. In other words “quasi integer” basic variables are rounded to the next integer value. The LP relaxation is solved after rounding all quasi integer variables. Several rounding iterations can be done as long as the LP solution is feasible, not integer and variables are rounded. In case of infeasibility the last rounding step is undone. The rounding intervals can be enlarged if no variable can be fixed in the first pass of rounding and reduced, if the LP-relaxation is infeasible in the first rounding pass or the LP-relaxation is integer.

Basic Algorithm of rounding process

```
1  For i=1 to number of rounding iterations do
2     For j=1 to number of all variable do
3        If variable is fixed or continuous cycle
4        f_j = x̄_j − ⌊x̄_j⌋
5        If f_j ≤ rl then
6           Fix variable to ⌊x̄_j⌋
7        Else if f_j ≥ ru then
8           Fix variable to ⌈x̄_j⌉
9        End if
10       Perform bound reduction on all variables
11       If problem is infeasible then
12          Clear settings of last rounding pass
13          If first rounding pass then reduce rounding intervals
14          Exit
15       End if
16    End for
17    If problem is infeasible then
18       If first rounding pass then
19         Cycle
20       Else
21         Exit
22       End if
23    End if
24    If no variables are rounded in this pass then
25       If first rounding pass then
26          Enlarge rounding interval
27          Cycle
28       Else
29          Exit
30       End if
31    End if
32    Solve LP
33    If problem is infeasible then
34      Clear settings of last rounding pass
35      If first rounding pass then
36         Reduce rounding intervals
37         Cycle
38       End if
39       Exit
40    Else if problem is integer then
41       Clear all settings
42       Reduce rounding intervals
43    End if
44 End for
```

The branch-and-bound engine of MOPS is called after rounding. If the search in the restricted search space is ended before one of the termination criteria (see below) is satisfied, the rounding intervals are reduced and the rounding procedure is repeated. The RSS heuristic is terminated if

- a given node limit is reached (default 50 nodes)
- the relative gap between the value of an integer solution found in the heuristic and the value of the LP relaxation after IP-Preprocessing is less than a threshold (default is 5%, i.e. 0.05)
- a time limit is reached (default is model size dependant).

Basic Algorithm of RSS heuristic prior to the branch-and-bound-algorithm

```
1  Solve LP after IP-Preprocessing
2  Fix the δ*nb variables with the maximum magnitude of reduced costs to the
   corresponding lower or upper bound
3  Do
4      Perform Basic Algorithm of rounding process
5      Use branch-and-bound algorithm until node limit, time limit or gap is reached
6      Clear settings
7      If termination criterion is reached Exit
8      Reduce size of rounding intervals
9  Enddo
```

5 Numerical results on real life models

Table 5 summarizes the computational results of the test problems presented in Table 1. Since the heuristic is also a specialized branch-and-bound-algorithm where the main work is to solve an LP at given node the nodes are not distinguished between heuristic and branch-and-bound algorithm. The heuristic is executed at most 50 nodes. The branch-and-bound algorithm is used thereafter to prove optimality.

Name	initial LP time (sec)	Nodes in heuristic + b&b	Total time (sec)
Halle1	33.19	0	33.77
Halle2	73.06	0	73.19
Halle3	108.81	0	115.81
Mun1	1527.81	0	1665.34
Mun2	798.24	10	879.13
Mun3	3490.20	0	3586.45

Table 5: Solution times on a 64 Bit Windows workstation with MOPS64

With the proposed modeling approach in [KlMS06] we were able to solve quite large problems in an acceptable amount of time. This required the selection of the: proper LP-engine, ordering heuristic for the Cholesky factorization, starting heuristic, branching and node selection strategies.

One remark on “acceptable” solution times is in order. Running times of a couple of hours do not seem ideal. However, the MDVSP-models are not solved on a daily basis. It is therefore acceptable to run such models over night.

6 Conclusion

MDVSP models from real life applications as modeled by time-space network flow models [MelKl02, GiKS05, KlMS06] can now be solved efficiently by a customized version of the optimizer MOPS. Customization requires only the setting of a few parameters. The progress in solution times is based on several improvements of the computational engines, an improved heuristic and the use of 64 bit platforms (Windows XP x64). Many of these instances were not solvable with the existing approaches or the running time was too long. It should be mentioned that the improvements in algorithms and implementation are also beneficial to many other applications based on linear mixed-integer programming models.

References

[Ande99] Andersen, E.D.: On exploiting problem structure in a basis identification procedure for linear programming, In: INFORMS Journal on Computing 11 (1999) 1, S. 95-103.

[AsLi98] Ashcraft, C. and Liu, J.W.: Robust Ordering of Sparse Matrices using Multisection, In: SIAM Journal on Matrix Analysis and Applications 19 (1998) 3, S. 816 – 832.

[BeCG87] Bertossi, I., Carraresi, P., Gallo, G.: On some matching problems arising in vehicle scheduling models, In: Networks 17 (1987), S. 271-281.

[Bixb02] Bixby, R. E.: Solving real-world linear programs: a decade and more of progress, In: Operations Research 50 (2002) 1, S. 3-15.

[CDFT89] Carpaneto, G., Dell'Amico, M., Fischetti, M., Toth, P.: A branch and bound algorithm for the multiple depot vehicle scheduling problem, In: Networks 19 (1989), S. 531-548.

[Dant63] Dantzig, G.: Linear Programming and Extensions, Princeton University Press, Princeton (1963).

[FoHW94] Forbes, M. A., Holt, J. N., Watts, A. M.: An exact algorithm for multi-depot bus scheduling, In: European Journal of Operational Research 72 (1994), S. 115-124.

[GeLi81] George, A. and Liu, J.W.: Computer Solution of Large Sparse Positive Definite Systems, Prentince-Hall Inc. (1981).

[GiKS05] Gintner V., Kliewer N. and Suhl L.: Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice, In: OR Spektrum 27 (2005), S. 507-523.

[HBJM95] Hane, C., Barnhart, C., Johnson, E.L., Marsten, R.E., Nemhauser, G.L., Sigismondi, G., The Fleet Assignment Problem: Solving A Large Integer Program, In: Mathematical Programming, 70 (1995) 2, S. 211-232.

[ILOG06] ILOG homepage: http://www.ilog.com/

[Karm84] Karmakar, N.: A new polynomial time algorithm for linear programming, In: Combinatorica 4 (1984), S. 373-395.

[Kliw05] Kliewer, N.: Optimierung des Fahrzeugeinsatzes im öffentlichen Personennahverkehr, PhD-Thesis at the University of Paderborn, II - Fakultät für Wirtschaftswissenschaften / Department Wirtschaftsinformatik, (2005)

[KlMS06] Kliewer, N.; Mellouli T. and Suhl L.: A time-space network based exact optimization model for multi-depot bus scheduling, In: European Journal of Operational Research 175 (2006), S. 1616-1627

[Kobe05] Koberstein, A.: The Dual Simplex Method: Techniques for a fast and stable implementation, PhD-Thesis at the University of Paderborn, II - Fakultät für Wirtschaftswissenschaften / Department Wirtschaftsinformatik, (2005)

[KoSu07] Koberstein, A. and Suhl U.H.: Progress in the Dual Simplex Algorithm for solving large scale LP problems: Practical dual phase 1 algorithms, to appear 2007 in Computational Optimization and Applications

[Lemk54] Lemke, C.E.: The Dual Method of Solving the Linear Programming Problem, In: Naval Research Logistics Quarterly 1 (1954), S. 36-47.

[Löbe99] Löbel, A.: Solving Large-Scale Multiple-Depot Vehicle Scheduling Problems, In: Computer-Aided Transit Schedulin 471 (1999), S. 193-220.

[Mehr92] Mehrotra, S.: On the implementation of a primal-dual interior point method, In: SIAM Journal on Optimization 2 (1992) 4, S. 575-601.

[MelKl02] Mellouli, T. und Kliewer, N.: Umlaufplanung im öffentlichen Verkehr mit mehreren Depots und Fahrzeugtypen: Neue Lösungsmodelle und praktische

Aspekte, In: Tagungsbericht der HEUREKA'02 (Optimierung in Verkehr und Transport), FGSV-Verlag, Köln (2002), S. 63-76.

[Mell03] Mellouli T.: Scheduling and Routing Systems in Public Transport Systems: Modeling, Optimization, and Decision Support, Habilitationsschrift, Universität Paderborn (2003).

[Mész96] Mészáros, Cs.: Fast Cholesky Factorization for Interior Point Methods of Linear Programming, In: Computers & Mathematics with Applications 31 (1996), S. 49-51.

[Mész98] Mészáros, Cs.: Ordering heuristics in interior point LP methods, In: New Trends in Mathematical Programming, (Eds.: F Gianessi, S. Komlósi und T. Rapcsák), Kluwer Academic Publishers (1998), S. 203-221.

[MOPS06] MOPS - Mathematical Optimization System homepage: http://www.mops-optimizer.com.

[RiSo94] Ribeiro, C., Soumis, F.: A column generation approach to the multiple-depot vehicle scheduling problem, In: Operations research 42 (1994) 1, S. 41-52.

[Suhl94] Suhl, U.H.: MOPS - Mathematical OPtimization System, In: European Journal of Operational Research 72 (1994), S. 312-322.

[SuSz94] Suhl, U.H. and Szymanski R.: Supernode Processing of Mixed-Integer Models, In: Computational Optimization and Applications 3 (1994), S. 317-331.

[SuWa04] Suhl, U.H. und Waue V.: Fortschritte bei der Lösung gemischt-ganzzahliger Optimierungsmodelle, In: Quantitative Methoden in ERP und SCM, (Eds.: L. Suhl und S. Voss), DSOR Beiträge zur Wirtschaftsinformatik (2004), S. 35-53.