Distributed Disaggregate Simplicial Decomposition — A Parallel Algorithm for Traffic Assignment *

Olof Damberg[1] and Athanasios Migdalas[2]

[1] oldam@math.liu.se
Division of Optimization, Department of Mathematics,
Linköping Institute of Technology, S - 581 83 Linköping, Sweden
[2] samig@math.liu.se
Division of Optimization, Department of Mathematics,
Linköping Institute of Technology, S - 581 83 Linköping, Sweden

Abstract. In this work we present a parallel algorithm for the user equilibrium traffic assignment problem. The algorithm is based on the concepts of simplicial decomposition, regularization and partial linearization. The parallel implementation is a synchronous, single program multiple data, algorithm using local load balancing. PVM is used for communication purposes. We present numerical results for two parallel platforms: a DEC Alpha cluster and a Parsytec GC/PowerPlus and for three real world traffic networks. The results show moderate speed-up with increasing efficiency for increasing OD-pair to link ratio.

Keywords: Transportation Network Equilibrium, Traffic Planning, Column Generation, Simplicial Decomposition, Regularized Frank-Wolfe Algorithm, Parallel Algorithms, Parallel Computers

1 Introduction

We consider in this paper the solution of nonlinear, convex, uncapacitated multicommodity network flow problems. Such problems arise in a number of important applications, e.g., in transportation [11, 24] and telecommunication [3] network analysis and design. Our discussion here is restricted to the former case.

The network equilibrium or traffic assignment problem arises in connection with a variety of activities, including planning and design, operation and management, supervision, control and route guidance in transportation networks. Not surprisingly, it has attracted considerable theoretical, methodological and computational developments during the last two decades that have resulted in quite efficient algorithms and increasingly realistic problem formulations and modeling [11, 24, 20, 4, 21]. However, as the realism of modeling increases and the application area becomes wider, several limitations appear. Network supervision and route guidance put requirements on real-time response which cannot

* Work supported by the EC-Esprit III/Europort2/LINPARC-TRAFFIC project.

be met. Also traffic planning, design and control are characteristic examples of a hierarchical decision process, in which the public sector at one level makes decisions seeking to improve the performance of the network, while network users at another level make choices with regard to route, travel mode, origin and destination of their travel, etc. For example, society selects the link to build (or close) for capacity improvements but the users choose the routes they perceive to be best. Other examples include pricing of freight transportation, traffic signal setting and origin-destination (OD) matrix estimation based on traffic counts. The resulting models are formulated as bilevel programming problems [24], with the traffic assignment problem at the second level. Hence, there is no hope to solve other traffic problems in realistic time unless a very fast, possibly real-time algorithm for the (standard) network equilibrium problem is at hand. The currently available (sequential) algorithms are characterized by limitations that affect (see also [22]):

- *The size of the (multimodal) networks, either at urban or national level, that can be addressed.* The number of arcs and origin-destination pairs (corresponding to commodities in traffic networks) is huge. Computational times higher than an hour are reported for the national Swedish network [18].
- *The level of detail in the system representation.* Much effort is invested in aggregating the physical network and thereby achieve computational tractability. The result is reduced accuracy and policy sensitivity. For instance, different levels of aggregation are needed in the definition of OD-pairs in connection with route guidance, depending on where the request for a route originates and for what destination [10].
- *The realism of the assumptions in the problem formulation.* Multiple classes of users, asymmetric user interactions and dynamic characteristics result in models that are far more computationally demanding than the standard problem and often require the repeated solution of (problems similar to) the standard traffic assignment problem [4, 15].
- *The accuracy of the (OD) demand matrices.* Often a few, inaccurate, alternative OD matrices are stored and used heuristically at different times to meet changing traffic conditions [10].
- *The solution of the hierarchical problems.* Network design, signal setting etc., are difficult bilevel problems that belong to the class of NP-hard problems. Currently, the best available algorithms are heuristics that require repeated solution of (standard) traffic assignment problems, however, not even the local optimality of the obtained solution is guaranteed [24].

The advent of parallel computer architectures, the recent developments in computer hardware and software as well as the increased availability of such systems outside the academy in the form of local area networks or workstation clusters, has enhanced the opportunity to overcome the mentioned obstacles. The LINPARC-TRAFFIC subproject of the EC-ESPRIT III/Europort 2 project was defined to address several of the issues related to these problems and to promote the use of such architectures by the traffic engineers by demonstrating their usefulness. The project posed several limitations to the academy; it was a *porting*

not a research project, that is, it required explicitly the porting of an existing sequential code to parallel platforms, not the invention of a new algorithm and its implementation. Moreover, the code should be *in use* and should be selected by the end-users, i.e., the traffic engineers (in this case from the Traffic Department of the municipality of Linköping and the Swedish Road and Traffic Institute). The benchmarking should be proposed by the same engineers, and they should evaluate the results. The implementation languages were restricted to FORTRAN 77, C or C++ and the message passing library PVM [14] for portability purposes. The code should be portable and able to execute on at least two parallel platforms of a specified set, which included DEC Alpha 2000 clusters and the European machine Parsytec CG/Powerplus.

This paper reports on some of the results obtained in this project. The outline is as follows. In Section 2 we give the traffic assignment model and in Section 3 we describe the basis for our algorithm — simplicial decomposition. In Section 4 we give the parallel algorithm with the essential details on data structures, communication and implementation. We report on some computational tests for two quite different parallel platforms and three networks in Section 5. Finally, we give our conclusions and suggestions for further research.

2 Problem and Algorithms

The traffic assignment problem appears in a variety of versions that, for instance, incorporate elastic demands or asymmetric user interactions (i.e., nonintegrable travel cost functions), or in so-called combined models that include mode choice and trip distribution [11]. In this paper we consider the standard traffic assignment problem, which is limited to integrable travel cost functions without link interactions and fixed travel demands.

2.1 Notation and Problem Formulation

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a network where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs (links). Let $s_a(v_a)$, associated with the arcs $a \in \mathcal{A}$, be positive and strictly monotone travel time (or delay) functions of the arc flow v_a. Let $\mathbf{v}$ denote the vector $[v_a]$. Let $K \subset \mathcal{N} \times \mathcal{N}$ be the set of origin-destination (OD) pairs, and denote the OD-matrix with $\mathbf{R} = [r_k]$, where r_k is the travel demand between the kth OD-pair. With each OD-pair we associate a specific commodity and let v_a^k denote the amount of commodity flowing from the origin $o(k)$ to the destination $d(k)$ of the kth OD-pair on arc a. For each node $i \in \mathcal{N}$, let $\mathcal{S}(i)$ denote the set of arcs emanating from that node, and $\mathcal{T}(i)$ the set of arcs terminating at the node. The standard (user equilibrium) traffic assignment problem is stated as follows:

[TAP]

$$\min \quad \sum_{a \in \mathcal{A}} \int_0^{v_a} s_a(x)dx \tag{1}$$

s.t.

$$\sum_{a \in \mathcal{S}(i)} v_a^k - \sum_{a \in \mathcal{T}(i)} v_a^k = \left\{ \begin{array}{rl} r_k & \text{if } o(k) = i \\ -r_k & \text{if } d(k) = i \\ 0 & \text{otherwise} \end{array} \right\} \forall i \in \mathcal{N}, \forall k \in \mathcal{K} \qquad (2)$$

$$v_a = \sum_{k \in \mathcal{K}} v_a^k, \forall a \in \mathcal{A} \qquad (3)$$

$$v_a^k \geq 0, \forall a \in \mathcal{A}, \forall k \in \mathcal{K} \qquad (4)$$

TAP can be restated in an alternative form. This is possible by reformulating it in terms of paths (routes) between origin-destination pairs. Let $\mathcal{P}_k$ denote the set of all (simple) paths from $o(k)$ to $d(k)$ and for every path $p \in P_k$, let h_p be its flow. Define the arc-path incidence matrix $\Delta = [\delta_{ap}]$ for $\mathcal{G}$ according to

$$\delta_{ap} = \left\{ \begin{array}{l} 1, \text{if arc } a \in p, p \in \mathcal{P}_k. \\ 0, \text{otherwise} \end{array} \right.$$

Then,

$$v_a = \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k} \delta_{ap} h_p, \qquad (5)$$

and **TAP** is restated as follows:

[TAP$_h$]

$$\min \qquad f(\mathbf{v}) = \sum_{a \in \mathcal{A}} \int_0^{v_a} s_a(x) dx \qquad (6)$$

s.t.

$$\sum_{p \in \mathcal{P}_k} h_p = r_k, \forall k \in \mathcal{K}$$

$$h_p \geq 0, \ \forall p \in \mathcal{P}_k, \ \forall k \in \mathcal{K}$$

TAP and **TAP$_h$** are equivalent in the sense that each optimal solution to the second problem corresponds to the unique solution of the first problem. However, the unique optimal arc flow of the first problem can usually be decomposed to more than one optimal path flows for the second problem. Thus, the strict convexity of the objective function in **TAP** is replaced by convexity. **TAP$_h$** is the arc-path, while **TAP** is the node-arc formulation of the traffic assignment problem.

2.2 Algorithms

Several sequential algorithms have been developed during the past two decades for the solution of traffic assignment problems. Their derivation and implementation are based either on the **TAP** or the **TAP$_h$** model. It is not our intention to give a detailed review on these results, see instead [11, 21, 19, 20, 23] and the references therein.

Among the most efficient algorithms, certain column generation schemes, known under the generic name *simplicial decomposition*, have attracted considerable theoretical as well as practical attention due to their efficiency, accuracy, generalized applicability, and re-optimization capabilities [15, 16, 21, 4, 3, 11, 21, 20].

The sequential implementation of such a scheme given in [20] seems to be, to this point, the most efficient and accurate algorithm for the traffic assignment problem. It has been used in practical traffic planning situations by e.g., the Swedish Road and Traffic Institute and the Traffic Department of the Linköping municipality.

Despite of considerable advances in the development of parallel optimization algorithms [5, 26], we do not know of any parallel algorithm developed with the traffic assignment problem in mind. In particular, no parallel implementation of the simplicial decomposition schemes for the traffic assignment problem have previously been reported. On the other hand, related activities have been documented. For instance, [22] *vectorize* the linearization algorithm of Frank-Wolfe on a CRAY supercomputer for the traffic assignment problem and report computational advantages for randomly generated networks.

Concerning *coarse-grained* or *MIMD parallelism*, [31, 23, 19, 9] develop parallelizable decomposition algorithms for block-constrained problems similar to **TAP**, however, they do not report on any implementation or computational results. An approach similar to [23] is utilized in [7] together with orthogonal projections in order to develop and implement a parallel algorithm for single-commodity, convex cost, bipartite transportation problems. They report computations and comparisons for an implementation on a 16 processor T800 Transputer system. [9] report results for the quadratic cost, bipartite, transportation problem on a Connection Machine CM-5 platform using an alternating direction method of multipliers. For a review of other algorithms for the single-commodity, convex cost, flow problems see the mentioned paper [7], the references therein as well as [5].

On the other hand, *fine-grained* or *SIMD parallelism* has attracted considerable attention. The work of Zenios and his associates [28, 30, 27, 29] has brought parallel implementations of simplicial decomposition schemes, in connection with the quadratic penalty approach, for the solution of capacitated network flow problems. They report on computational results for large-scale linear problems on the CM-2 platform. Although their approach would be able, in principal, to attack the traffic assignment problem, they do not present any code specialization to the case neither do they report on computations for such a problem or for networks of the size that usually appear in connection with the traffic assignment problem.

3 Simplicial Decomposition

The simplicial decomposition approach to traffic assignment problem can be viewed as a *column generation* approach to solving $\mathbf{TAP}_h$. That is, starting

with a subset Π_k of paths in $\mathcal{P}_k$, for all $k \in \mathcal{K}$, the *master problem* **MP** below is solved.

[MP]

$$\min \quad \sum_{a \in \mathcal{A}} \int_0^{v_a} s_a(x)dx$$

s.t.

$$\sum_{p \in \Pi_k} h_p = r_k, \forall k \in \mathcal{K}$$

$$v_a = \sum_{k \in \mathcal{K}} \sum_{p \in \Pi_k} \delta_{ap} h_p, \forall a \in \mathcal{A}$$

$$h_p \geq 0, \forall p \in \Pi_k, \ \forall k \in \mathcal{K}.$$

If the optimal path flows $\bar{h}_p$ in **MP** are also optimal in $\mathbf{TAP}_h$, then the traffic assignment has been solved. Otherwise, the optimal path flows $\bar{h}_p$ in **MP** are used in order to evaluate the arc flows in (5) and the gradient of the objective function $f(\mathbf{v})$ in (6). The gradient is used in *linearizing* the objective function (6) at the current solution point $(\bar{v}_a)$. The following *linearized subproblem*, where $c_a = s_a(\bar{v}_a)$ denotes the gradient component associated with arc $a \in A$, is thus obtained:

[LP]

$$\min \quad \sum_{k \in \mathcal{K}} \sum_{a \in \mathcal{A}} \sum_{p \in \mathcal{P}_k} c_a h_p$$

s.t.

$$\sum_{p \in \mathcal{P}_k} h_p = r_k, \forall k \in \mathcal{K}$$

$$h_p \geq 0, \ \forall p \in \mathcal{P}_k, \ \forall k \in \mathcal{K}.$$

This is an *all-or-nothing assignment* problem that separates with k into $|\mathcal{K}|$ subproblems:

[$\mathbf{LP}_k$]

$$\min \quad \sum_{a \in \mathcal{A}} \sum_{p \in \mathcal{P}_k} c_a h_p$$

s.t.

$$\sum_{p \in \mathcal{P}_k} h_p = r_k$$

$$h_p \geq 0, \ \forall p \in \mathcal{P}_k.$$

The optimal solution to each $\mathbf{LP}_k$ furnishes a new path to the corresponding working set Π_k, and the new enlarged master problem **MP** is solved again.

The simplicial decomposition approach is thus, an iterative approach in which linear path generating subproblems ($\mathbf{LP}_k$) and a nonlinear master problem (**MP**) of increasing dimensionality are solved alternately.

A second view on the simplicial decomposition approach is based on Caratheodory's theorem. The algorithmic development is based on **TAP**. For given feasible arc flows $\bar{v}_a$, the objective function in (1) is linearized, and the *all-or-nothing assignment* subproblem below is obtained:

[ANP]

$$\min \quad \sum_{a \in \mathcal{A}} c_a v_a$$

s.t.

$$\sum_{a \in \mathcal{S}(i)} v_a^k - \sum_{a \in \mathcal{T}(i)} v_a^k = \left\{ \begin{array}{rl} r_k & \text{if } o(k) = i \\ -r_k & \text{if } d(k) = i \\ 0 & \text{otherwise} \end{array} \right\} \forall i \in \mathcal{N}, \forall k \in \mathcal{K}$$

$$v_a = \sum_{k \in \mathcal{K}} v_a^k, \forall a \in \mathcal{A}$$

$$v_a^k \geq 0, \forall a \in \mathcal{A}, \forall k \in \mathcal{K}$$

The optimal solution to this problem is an extreme point of the feasible region (2)-(4) in the original problem **TAP**. Since by the Caratheodory theorem, any feasible point of the bounded polytope (2)-(4) can be expressed as a convex combination of its extreme points, the solution to **TAP** can be found by searching for those combiners in the expression that result in the minimization of (1). Let $\mathcal{L}$ enumerate (a subset of) the set of extreme points. This results in a *master problem* of the following type:

[MP']

$$\min \quad \sum_{a \in \mathcal{A}} \int_0^{v_a} s_a(x) dx$$

s.t.

$$\sum_{j \in \mathcal{L}} \lambda_j = 1$$

$$\lambda_j \geq 0, \ \forall j \in \mathcal{L}$$

$$v_a = \sum_{j \in \mathcal{L}} \lambda_j v_a^j,$$

where v_a^j is the component of the jth extreme flow point $\mathbf{v}^j$ that associates to arc a. Thus, in this case, the simplicial decomposition approach iterates alternating between an extreme point generation phase, where **ANP** is solved, and a master solving phase, in which a master problem **MP'** is solved over an enlarged $\mathcal{L}$. This is the approach traditionally taken for the traffic assignment problem and its variants [15, 16, 21].

However, since the feasible region in **TAP**, as well as in **ANP**, is a Cartesian product of polytopes, it is possible to consider the extreme points of each polytope separately. This leads to a *disaggregate master problem* [20], where a convex combination constraint for each polytope is included:

[MP"]

$$\min \quad \sum_{a \in \mathcal{A}} \int_0^{v_a} s_a(x)dx$$

s.t.

$$\sum_{p \in \mathcal{L}_k} \lambda_p = 1, \ \forall k \in \mathcal{K}$$

$$\lambda_p \geq 0, \ \forall p \in \mathcal{L}_k \ \forall k \in \mathcal{K}$$

$$v_a = \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{L}_k} \lambda_p v_a^p,$$

where $\mathcal{L}_k$ enumerates (a subset of) the set of extreme points of the kth polytope, which corresponds to the constraints of the kth OD-pair, and v_a^p is the component of the extreme arc flow $\mathbf{v}^p$, $p \in \mathcal{L}_k$, $k \in \mathcal{K}$, that corresponds to arc a.

It is also useful to notice that the linear problem **ANP** separates by k into $|\mathcal{K}|$ *shortest path* subproblems of the following type:

[SPP$_k$]

$$\min \quad \widehat{f}_k(\mathbf{v}) = \sum_{a \in \mathcal{A}} c_a v_a^k$$

s.t.

$$\sum_{a \in \mathcal{S}(i)} v_a^k - \sum_{a \in \mathcal{T}(i)} v_a^k = \left\{ \begin{array}{rl} r_k & \text{if } o(k) = i \\ -r_k & \text{if } d(k) = i \\ 0 & \text{otherwise} \end{array} \right\} \forall i \in \mathcal{N}$$

$$v_a^k \geq 0, \forall a \in \mathcal{A}.$$

Clearly, **[SPP$_k$]** is equivalent to **[LP$_k$]**. Moreover, utilizing the concept of *fractions of flow* [3],

$$\lambda_p \ = \ \frac{h_p}{r_k} \ \forall p \in \mathcal{P}_k,$$

and that each extreme arc flow in **[SPP$_k$]** uniquely corresponds to an extreme path flow in **[LP$_k$]** (and conversely), the master problem **MP** is equivalent to the disaggregate master **MP"**. Indeed,

$$v_a \ = \ \sum_{k \in \mathcal{K}} r_k \sum_{p \in \mathcal{P}_k} \delta_{ap} \lambda_p, \ \forall a \in \mathcal{A}$$

and

$$h_p \ = \ \lambda_p r_k, \ \forall p \in \mathcal{P}_k, \ \forall k \in \mathcal{K}.$$

The two approaches are therefore algorithmically equivalent.

3.1 Solving the Master Problem

In all versions of the simplicial decomposition, the theoretical difficulty of the master problem approaches the difficulty of the original traffic assignment problem. However, its much lower dimensionality, its special structure and re-optimization capabilities of specialized algorithms result in tractable approaches.

For specializations of the gradient projection method, scaled gradient projection, and the projected Newton method to the master problem consult [4, 16, 5, 3] and the references therein. These techniques are based on the elimination of the convexity constraints by substituting one of the variables in each constraint. Thus, the master problem is transformed into an equivalent problem with only simple bounds on the variables. A similar approach is described in [28, 29]. In [30] the well-known code MINOS is applied to a more complicated master problem. Typically, the overall approach is applied in a cyclic manner over the OD-pairs, i.e., as a Gauss-Seidel type of decomposition [3, 5, 11]. However, [28, 29, 30] do not follow this approach. Instead, all commodities are treated in each overall iteration.

The disaggregate simplicial decomposition in [20] employs the reduced gradient approach for the master problem. Moreover, the overall approach is not cyclic, that is, in each iteration, the shortest paths are calculated for all OD-pairs.

In [8], the partial linearization [19] and the regularized Frank-Wolfe approach [23] are adapted to solve the master problem. Although other approximations are possible, an additive and separable by OD-pair, strongly convex function is used to approximate the objective of the master problem. This results into convex, quadratic knapsack problems (one for each OD-pair or convexity constraint) that are solvable efficiently in linear time; see e.g., [25, 17, 6]. By using the second order Taylor expansion of the objective we obtain the following problem for OD-pair k:

[QPP$_k$]

$$\begin{aligned} \min \quad & \sum_{p \in \Pi_k} \{\frac{1}{2} d_p (h_p - \bar{h}_p)^2 + l_p (h_p - \bar{h}_p)\} \\ \text{s.t.} \quad & \\ & \sum_{p \in \Pi_k} h_p = r_k \\ & h_p \geq 0, \ \forall p \in \Pi_k, \end{aligned}$$

where

$$d_p = \sum_{a \in \mathcal{A}} \delta_{ap} s'_a(\bar{v}_a), \quad l_p = \sum_{a \in \mathcal{A}} \delta_{ap} s_a(\bar{v}_a).$$

3.2 The Line Search Obstacle

The feasible direction methods adapted to solve the master problem, all, require line search. In effect, the line search is the most demanding part of the algorithms for the master problem. Indeed, in the disaggregate simplicial code of [20], the line search module accounts for about 60% of the overall CPU time in the cases of the Linköping and Barcelona networks. Moreover, line searches are inherently non-parallel. Bertsekas [3] suggest that constant step length values close to 1 work typically quite well in practice and are particularly well-suited for distributed implementation. However, such a scheme cannot be shown to be theoretically convergent in general. In [8], the objective function of the master problem is locally approximated by its second order Taylor expansion, and the step length is calculated from this quadratic approximation.

3.3 Solving the Linear Subproblems

In all versions of the simplicial decomposition, the linearized subproblems are all-or-nothing problems that separate by OD-pair. For each such OD-pair, the corresponding subproblem is solved in two steps; by first identifying the shortest path from $o(k)$ to $d(k)$, and then assigning the entire demand r_k to the arcs of that path. These steps can be performed quite efficiently and in polynomial time. The Dijkstra shortest path algorithm or any other efficiently implemented shortest path algorithm can be utilized, see e.g.,[13, 1].

4 The Parallel algorithm

We decided to opt for a synchronous single program multiple data (SPMD) algorithm. There are three main reasons for this choice: i) the convergence properties are exactly the same as for the sequential algorithm, ii) it is relatively easy to implement since the compute nodes (processors) are doing (virtually) the same thing, and iii) the $\mathbf{TAP}_h$ problem structure suits this parallelization model nicely. In the following we describe the data structures and distribution of data, communication (message passing) patterns and give a pseudo-code for the SPMD algorithm.

4.1 Data structures and distribution

By using the $\mathbf{TAP}_h$ model, we can readily see that there are two main data structures:

1. **Network**. The network is defined by the number of nodes, links (with corresponding delay functions) and the underlying graph. The graph is stored as a sparse adjacency list of size $|\mathcal{N}| + |\mathcal{A}|$ and the link data (flow, three delay function terms, and two for temporary calculations) are stored in vectors of size $|\mathcal{A}|$. All compute nodes hold the entire network data so (given that the compute nodes know the present link flow) the link delay can be computed

i parallel and the shortest path problems can be solved in parallel without any communication.

2. **OD-pairs**. An OD-pair is defined by its origin, destination and the demand of flow that is to carried between the two. All processors hold this (constant) information (of size $O(|\mathcal{K}|)$ to reduce the amount of data to be communicated if load balancing (see Section 4.2) is used. Furthermore, in the disaggregated simplicial decomposition case, there are also routes associated with the OD-pairs. For each OD-pair there is a structure holding a linked list of the routes (a dynamically allocated vector of network link indices) which has been generated, i.e., the set Π_k (or $\mathcal{L}_k$) from the subproblem phase $\mathbf{LP}_k$ (or $\mathbf{SPP}_k$). The same structure also holds the present route flow. The total size of these structures can be estimated with $O(|\mathcal{K}||\mathcal{A}|)$, since, in general, there are only a handful of routes generated per OD-pair in an user equilibrium solution. This is the (possibly huge set of) data which is to be divided among the compute nodes.
   By distributing the OD-pairs over the processors we obtain a data distribution which allows for the communication-less solution of the route generating shortest path problems ($\mathbf{SPP}_k$) in parallel. Furthermore, within the master problem **MP"** we can solve the quadratic knapsack problems (see Section 3.1) in parallel without communication.

By summation over all routes in Π_k and subsequently over $\mathcal{K}$, we obtain the total flow on every network link and, thus, we can compute the objective [cf. (6)] and the link gradients (i.e., the link cost or delay). This is the obvious drawback of the data distribution chosen since we need to collect the link flow from the compute nodes every time we evaluate the objective and the link delay. This procedure is performed once per iteration in the master problem solver and once per main iteration (see Section 4.3). Furthermore, the amount to be communicated will be linearly dependent on the number of parallel processors used, since they hold their share of the link flow on *all* links. It is virtually impossible to arrange the OD-pair distribution so that we can guarantee that the routes generated will only use a certain set of the links, hence, all compute nodes must in all practical cases have all link information.

4.2 Communication (message passing)

Only one type (essentially) of communication is used in the proposed algorithm. Denote it `reduce-add_and_multicast` for further reference.

It works as follows: all nodes compute their share of the data in question. A 'reduction with add' operation is then performed to gather and compute (summation) the total result, which in its turn is sent (multicasted) back to all compute nodes. Optimal algorithms and their time complexity for these operations can be found in, e.g., [5, Section 1.3].

Load balancing. In order to fully utilize the processing power it is essential that the compute nodes use equal amount of time to process their share. This

is especially true for our synchronous algorithm. Since the workload changes dynamically (and we can not easily know how much beforehand) iteration by iteration, there is a strong possibility that the computing time for each processor will differ significantly after a few iterations if no load balancing is performed. A balancing scheme was therefore incorporated into the code. It performs local load balancing between pairs of processors by comparing their compute time and transferring OD-pairs, i.e., the structure holding the route data (see Section 4.1, item 2) in order to attempt to equalize the compute time (see Figure 1). Denote this procedure `balance-load` for further reference. Computational tests

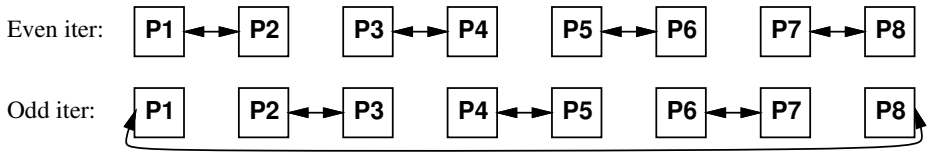

Fig. 1. Local load balancing scheme.

has proven this to be efficient, especially if the algorithm is used for solving several snapshots of the network with varying OD-pair demand for each snapshot. One snapshot is in general solved with only a handful major iterations, so there is little to gain in this case.

Synchronization. The `reduce-add_and_multicast` and `balance-load` routines are implemented so that the processors wait for the incoming data before continuing (i.e., blocking receives). This forces the processors to operate synchronously. Hence, at all times they will know the same information as a single processor would (running the sequential version of the algorithm). This ensures that the parallel algorithm has exactly the same convergence properties as the sequential [8] algorithm.

4.3 The distributed disaggregate simplicial decomposition algorithm

In this section we provide a pseudo code for the proposed parallel algorithm.

See Sections 2.1 and 3 for the notation introduced there. Let $\mathcal{C}$ denote the set of compute nodes and let $\mathcal{K}^c$ be the set of OD-pairs that compute node $c \in \mathcal{C}$ deals with. Clearly $\cup_{c\in\mathcal{C}}\mathcal{K}^c = \mathcal{K}, \mathcal{K}^i \cap \mathcal{K}^j = \emptyset, \forall i,j \in \mathcal{C}, i \neq j$. Let $\mathbf{v}^c$ denote the vector of link flows $[v_a^c]$, where $v_a^c = \sum_{k\in\mathcal{K}^c}\sum_{p\in\Pi_k} h_p\delta_{ap}$, i.e., the part of the link flow that the generated routes held by compute node c carry. Let $\widehat{f}^c(\mathbf{v}) = \sum_{k\in\mathcal{K}^c}\widehat{f}_k(\mathbf{v})$ be the total shortest path cost for problems $\mathbf{SPP}_k$ which are solved by processor c. Finally, define UBD as the upper bound on the objective and by LBD the lower bound.

Note that all processors run the same program, but each have different data sets (which correspond to $\mathcal{K}^c$), i.e., it is a SPMD algorithm.

Algorithm `SPMD-DDSD`.

Initialization. *Data input and initial distribution of OD-pairs. The $|\mathcal{K}|$ OD-pairs are evenly distributed over the $|\mathcal{C}|$ compute nodes.*

Heuristic. *Generate the first route for each OD-pair at zero link flow, and assign the full OD demand to the route — an all-or-nothing assignment.*

1. $\mathbf{v} \longleftarrow 0, \quad c_a \longleftarrow s_a(\mathbf{v}), \quad \Pi_k \longleftarrow \emptyset.$
2. $\widehat{p} \Longleftarrow$ Solve $\mathbf{SPP}_k(c_a, \ldots), \quad h_{\widehat{p}} \longleftarrow r_k, \quad \Pi_k \longleftarrow \Pi_k \cup \widehat{p}.$
3. $\mathbf{v} \Longleftarrow$ `reduce-add_and_multicast(`$\mathbf{v}^c$`)`
4. UBD $\longleftarrow f(\mathbf{v}), \quad$ LBD $\longleftarrow -\infty$

Main solver. *Generate routes based on the current link delays. Augment the set of generated routes. Solve the master problem over the restricted set of generated routes.*

Subproblem solver. *Shortest path based on current link delays. Augment the set of generated routes if not previously included. Compute a lower bound on the objective.*

5. $c_a \longleftarrow s_a(\mathbf{v}).$
6. $\{\widehat{f}_k(\mathbf{v}), \quad \widehat{p}\} \Longleftarrow$ Solve $\mathbf{SPP}_k(c_a, \ldots)$
   If $\widehat{p} \notin \Pi_k$ then $\Pi_k \longleftarrow \Pi_k \cup \widehat{p}, \quad h_{\widehat{p}} \longleftarrow 0.$
7. $\widehat{f}(\mathbf{v}) \Longleftarrow$ `reduce-add_and_multicast(`$\widehat{f}^c(\mathbf{v})$`)`
8. LBD $\longleftarrow \max\{\text{LBD}, \text{UBD} + \widehat{f}(\mathbf{v}) - \sum_{a \in \mathcal{A}} c_a v_a\}$

Convergence test. *Terminate algorithm if the relative objective error is below some a priori set constant.*

9. If $(\text{UBD} - \text{LBD})/\text{LBD} \leq \varepsilon$ then Terminate!

Restricted master solver. *Solve the equilibrium problem over the restricted set of generated routes. Each iteration the objective is approximated with a separable (over the OD-pairs) quadratic function; see Section 3.1.*

10. $d_p \longleftarrow \sum_{a \in \mathcal{A}} \delta_{ap} s'_a(v_a), \quad l_p = \sum_{a \in \mathcal{A}} \delta_{ap} s_a(v_a).$
11. $h_p^{\text{old}} \longleftarrow h_p, \quad v_a^{\text{old}} \longleftarrow v_a$
12. $h_p \Longleftarrow$ Solve $\mathbf{QPP}_k(h_p^{\text{old}}, \ldots)$
13. $\mathbf{v} \Longleftarrow$ `reduce-add_and_multicast(`$\mathbf{v}^c$`)`
14. $h_p^{\text{dir}} \longleftarrow h_p - h_p^{\text{old}}, \quad v_a^{\text{dir}} \longleftarrow v_a - v_a^{\text{old}}$
15. step $\longleftarrow \min\{1, -\sum_{a \in \mathcal{A}} s_a(v_a^{\text{old}}) v_a^{\text{dir}} / \sum_{a \in \mathcal{A}} s'_a(v_a^{\text{old}})(v_a^{\text{dir}})^2\}$
16. $h_p \longleftarrow h_p^{\text{old}} + \text{step} * h_p^{\text{dir}}, \quad v_a \longleftarrow v_a^{\text{old}} + \text{step} * v_a^{\text{dir}}$
17. UBD $\longleftarrow f(\mathbf{v})$
18. Terminate master after (a priori set) number of iterations. Return new equilibrium flow.
19. Goto 10.

Load equalization. *Re-distribute OD-pair data to obtain equal running times for the compute nodes.*

20. `balance-load`
21. Goto 5.

Note also that a bounding procedure can be used to terminate the master problem. A lower bound can easily be calculated by adding the cost of the least cost route for each OD-pair (see [8]). However, this cost will unfortunately have to be collected by a `reduce-add_and_multicast` call in order to be made accessible to all compute nodes. Experiments indicate that this will not decrease the running time of the algorithm.

4.4 Implementation

As mentioned in the introduction, one of the criteria for this project was that an existing code should be parallelized — in our case the state-of-the-art DSD Fortran 77 code of Larsson and Patriksson [20]. However, after studying the code we found that the structure of the code and data made it virtually impossible to do a straightforward parallelization. Furthermore, a parallel version of their master solver (cf., Section 3.1) with Armijo-type [2] line searches would have resulted in excessive amount of communication. We opted therefore to write the code from scratch and incorporate a new master solver (see [8]) which parallelizes more efficiently. The two main embedded routines needed are a shortest path and a quadratic knapsack solver. As a shortest path solver we implemented the `L-TRESHOLD` code in [13]. The quadratic knapsack solver is an implementation of the $O(n \log n)$ time code in [17].

The algorithm was written in C++. For message passing purposes we used the Parallel Virtual Machine (PVM) library [14] on both platforms. We used the GNU C++ (g++) compiler on the Parsytec and Digital's own C++ (cxx) compiler on the Alpha. We were forced to write our own reduction code (see Section 4.2) since the the message passing library (PVM) reduction call was not available on the Parsytec platform due to an old release of the PVM library. It would have been preferable to use a reduction call optimized specifically for the respective platforms, but for portability reasons we had to implement the reduction code ourselves. Unfortunately, a slight performance penalty can be expected from this.

Finally, we mention that all memory is allocated on demand, hence, the code will use the minimal amount of memory necessary to hold the data. No static (over-)allocation of memory is necessary (as in Fortran-77 codes) in order to solve problems of different sizes.

5 Computational Results

To evaluate the implementation of the proposed algorithm we ran some numerical tests on two parallel platforms and three relatively large scale real world networks.

5.1 Parallel platforms

Parsytec GC/Powerplus. The machine we have used is installed at the University of Linköping. It is a MIMD computer with distributed memory and 128 processors. Each node consists of two PowerPC-601, four T805 transputers, and a memory capacity of 32 Mbyte. The peak performance per node is 160 Mflops (64-bit). The nodes are connected in a bidirectional 2D fat-grid. The maximum communication rate has been empirically measured to 3.2 Mbyte/s between neighboring processors in the network and the bandwidth does not drop significantly for communication between processors several hops away. The corresponding minimum network latency has been measured to 141 microseconds between nearest neighbors. The latency, however, increases slightly for increasing hops. For example, for five hops the latency is 200 microseconds.

The system provides a number of tools for parallel program development, for instance, optimizing compilers for C, C++ and Fortran 77, debuggers, performance analysis, etc. It supports the PVM, PARMACS and MPI communication packages.

Digital Equipment Corporation (DEC) Alpha cluster. The DEC Alpha cluster, installed at Para//ab, University of Bergen, Norway, is a multicomputer which consists of one DEC Alpha 2000 at 233 MHz with 512 MB memory, three DEC Alpha 2000 at 233 MHz with 128 MB memory and four DEC Alpha 1000 at 233 MHz and 128 MB memory. The machines are connected by a DEC GIGAswitch/FDDI and a high performance DEC Memory Channel. Latency is below 10 microseconds and transfer rate around 150 Mbyte/second.

The operating system is a Digital UNIX. It supports several message passing packages, such as PVM (versions 3.2.6 and 3.3.5), PARMACS, and MPI. The compilers supported are C, C++, Fortran 77 and Fortran 90.

5.2 Network descriptions

We have tested the proposed algorithm on three relatively large scale real world traffic networks.

The Barcelona network. Barcelona is the second largest city in Spain and capitol of the Catalonian Region. It is the center of a greater metropolitan area that extends over 585 km^2 and has approximately a total of 4 million people or about 68% of the population in Catalunia. The Barcelona network consist of 1020 nodes, 2522 links and 7922 OD-pairs. The link delays are expressed as highly nonlinear functions of the link flow and have previously demonstrated numerical instability on sequential platforms.

The Linköping network. This test case is essentially similar to the previous one. However, Linköping, which is Swedens fifth largest city is much smaller. In the metropolitan area of Linköping live about 128 000 people. The road

network of Linköping consists of 335 nodes, 882 links and 12372 OD-pairs. It is interesting to note that, although the two networks show considerable difference in the number of nodes and links, the number of OD-pairs do not show the same difference. Thus, even for small cities, the number of OD-pairs can exceed by thousand the configuration of any currently available MIMD machine. We, thus, think that the approach to parallelization of the DSD, based on decomposition of the network model by OD-pairs, is promising from a scalability point of view.

The Winnipeg network. This network of the Canadian city Winnipeg is often used in benchmarking tests (see [12] for a description). It has 1052 nodes, 2836 links and 4344 OD-pairs.

5.3 Comparison of platforms and networks

The most significant difference (disregarding the number of available processors) between the two parallel machines is the communication performance. The DEC Alpha cluster has orders of magnitude better latency and transfer rate. We believe this will be beneficial for our algorithm, since the communication requirements are rather large. This is also confirmed in Section 5.4 where we can observe a better speed-up for the Alpha cluster.

A summary of the network data is given in Table 1. The table also specifies the OD-pair to link ratio which is an important factor for the efficiency of our algorithm. In Section 5.4 we can readily see that the higher ratio the higher efficiency of the parallel algorithm. This implies that the need for high aggregation of OD-pair data is not necessary from the perspective of parallelization, i.e., more detailed networks can be solved without penalty.

Table 1. Summary of network data

Network	# nodes	# links	# OD-pairs	OD-pair to link ratio
Barcelona	1020	2522	7922	3.1
Linköping	335	882	12372	14.0
Winnipeg	1052	2836	4344	1.5

5.4 Numerical experiments

Each experiment was run five times and the solution times reported in the tables below is the average of the five runs. We performed two kinds of tests: i) *one snapshot*, i.e., finding the user equilibrium (UE) solution for the network once and ii) *100 snapshots*, i.e., finding the network's UE solution 100 times, where the demand is changing from run to run.

One snapshot. The computational results for one snapshot for the Barcelona, Linköping and Winnipeg network are depicted in Table 2. The speed-ups for the two parallel platforms are given in Figure 3. In Figure 2 we provide a GIS map depicting the flow on the Linköping network according to the one snapshot user equilibrium solution.

Table 2. Wall clock time (not including I/O to disk) in seconds for one snapshot the networks. Requested relative objective error: 0.1 %.

		Number of processors				
Network	Platform	1	2	4	8	16
Barcelona	Parsytec GC/Powerplus	44.0	27.3	19.5	13.1	12.9
	DEC Alpha cluster	30.5	14.6	11.0	7.6	—
Linköping	Parsytec GC/Powerplus	49.8	35.2	22.0	12.0	9.0
	DEC Alpha cluster	36.6	17.5	12.3	6.99	—
Winnipeg	Parsytec GC/Powerplus	60.1	33.4	22.7	18.8	18.5
	DEC Alpha cluster	39.9	20.1	15.2	12.6	—

Fig. 2. GIS map of the flow on the Linköping network.

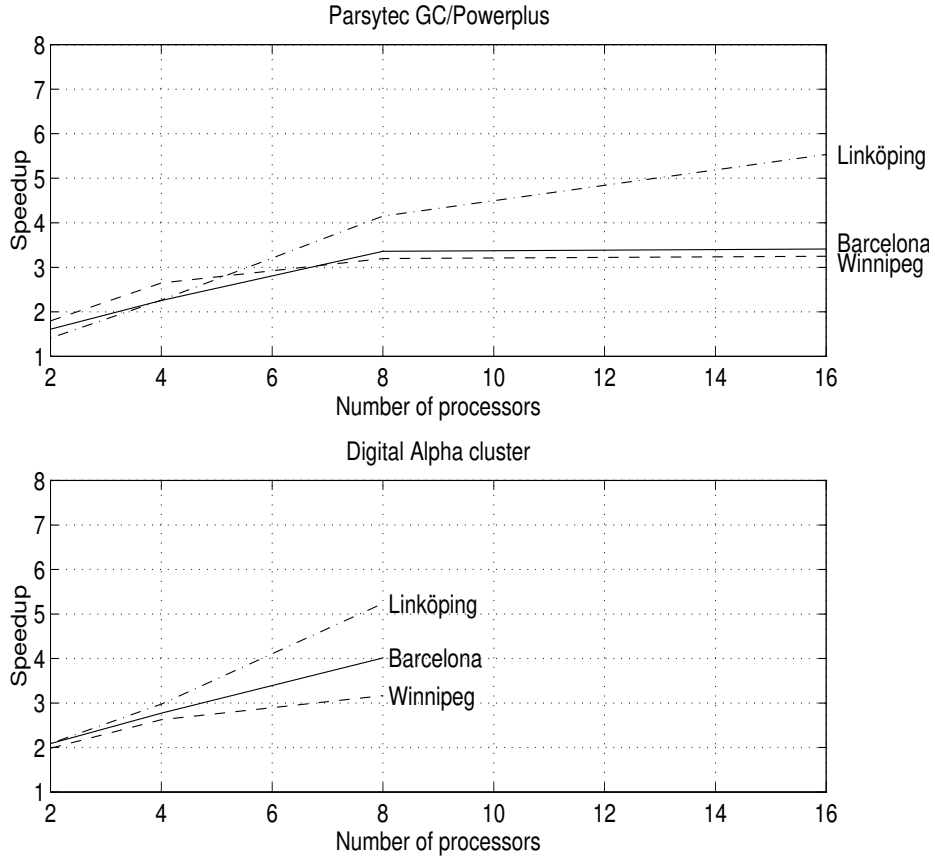

Fig. 3. Speed-up for one snapshot of all three networks (data from Table 2).

100 snapshots. To simulate the variation in traffic load during a workday (12 hours, with traffic load peaks in the morning, at lunch and in the evening) we implemented a network generator, from which we obtain the OD-pair demand at a given time. The demand is scaled according to Figure 4. The solver is restarted from the previous optimal equilibrium flow for each snapshot.

This test may also be seen as an indicator on the possibility of computing solutions in real-time for large networks as well as in using the code for the solution of more complicated models such as the hierarchical problems mentioned in Section 1.

The computational results for 100 snapshots for the Barcelona, Linköping and Winnipeg network are depicted in Table 3. The speed-ups for the two parallel platforms are given in Figure 5.

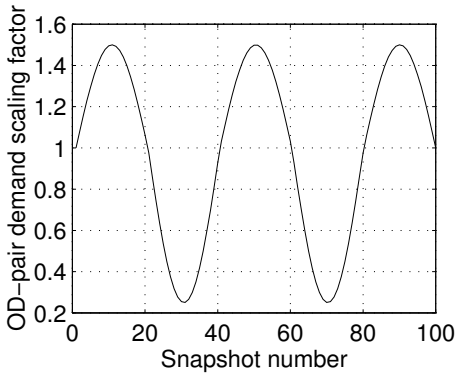

Fig. 4. OD-pair demand scaling of original demand for each snapshot.

Table 3. Wall clock time (not including I/O to disk) in seconds for 100 snapshots of the networks. Requested relative objective error for each snapshot: 0.5%.

		Number of processors				
Network	Platform	1	2	4	8	16
Barcelona	Parsytec GC/Powerplus	2889	1520	822	576	561
	DEC Alpha cluster	1970	987	526	356	—
Linköping	Parsytec GC/Powerplus	7426	4014	2196	1329	1045
	DEC Alpha cluster	3898	1954	1035	675	—
Winnipeg	Parsytec GC/Powerplus	3294	1734	925	708	689
	DEC Alpha cluster	2281	1150	633	450	—

5.5 Discussion

It is interesting to notice that although the Linköping network is smaller in size than the other two, better speed-up is obtained for the former. This is due to the higher OD-pair to link ratio of the Linköping network (see Table 1) which clearly affects the result. Indeed, it can be predicted from Table 1 that a higher speed-up should be attainable for the Barcelona network than for the Winnipeg network. Figure 3 confirms this prediction. The explanation for this behavior is that there is a better computation to communication ratio, since the work load depends in large on the number of OD-pairs and the communication load depends on the number in links.

We can conclude from Table 3, that solutions times between five and ten seconds for each snapshot on all tested networks and platforms are achievable.

From Figure 5 we can observe that the speed-up achieved for the 100 snapshot case is better than for the one snapshot case. This is mainly due to the load balancing which has better effect. Moreover, we see that the speed-up is better

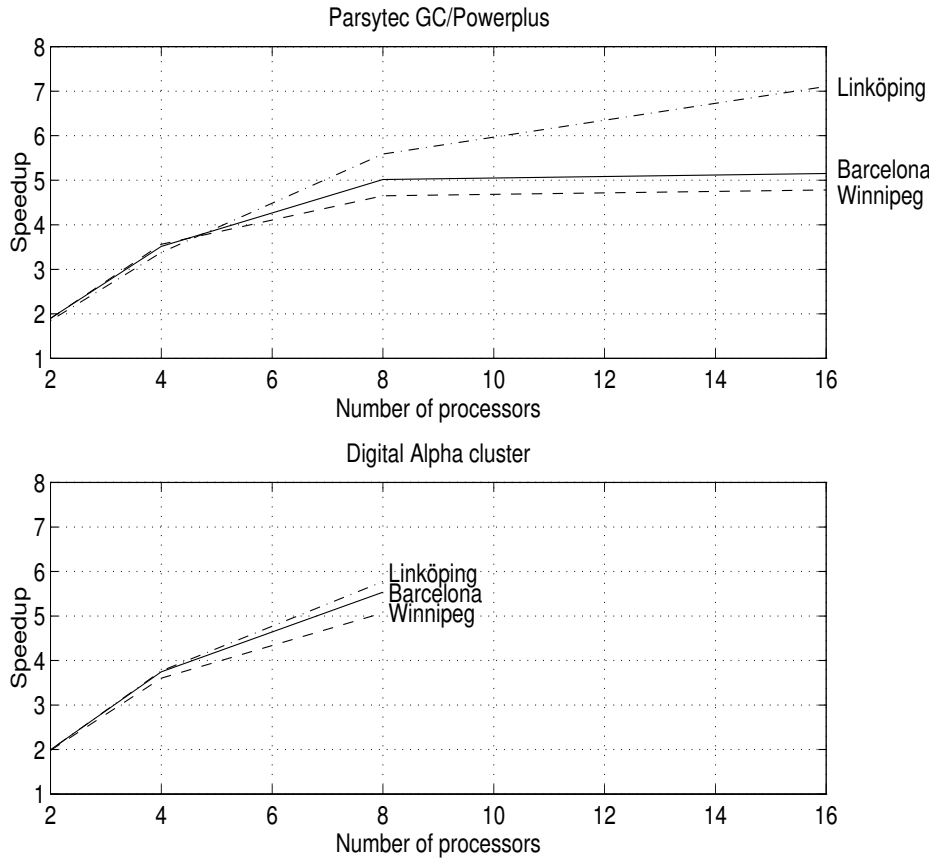

Fig. 5. Speed-up for 100 snapshots of all three networks (data from Table 3)

for the Alpha platform. This is expected since the communication characteristics are superior for this platform.

6 Conclusions and further research

From the computations performed on the Parsytec GC/PowerPlus and the DEC Alpha cluster platforms, it is obvious that how well the algorithm performs on a given network is not based only on the network's size but also strongly on the ratio of the number of OD-pairs to the number of links. This is a consequence of the inherent modeling and algorithmic properties as well as of the way the parallelization is done. In section 5.4 we can see from the figures that the higher this ratio, the higher the efficiency of the parallel algorithm. This implies a valuable benefit with respect to actual network modeling; the need for high aggregation

of OD-pair data is not necessary from the perspective of parallelization, that is, more detailed networks can be solved without the computational time penalties that would have been encountered in the sequential case. The parallel code has also been shown able to reduce the time for each successive network snapshot to a few seconds in the range 5 to 10 for networks of realistic size, demonstrating strong reoptimization capabilities. This is a positive step towards bypassing the obstacles, mentioned in section 1, that are encountered in connection with planning, management and supervising of traffic networks on sequential platforms.

Subjects for further investigation include the application and validation of the code with respect to performance and scalability on extremely large networks, with high OD-pair to link ratio, as for instance the national Swedish road system, as well as its utilization in solving combined and hierarchical traffic models and extensions to the case of asymmetric traffic delays. From the implementation point of view, certain points that have not been examined thoroughly in this paper concern alternative and possibly more efficient data distribution, and elimination of the need for synchronization. With respect to the later, the theory of partial asynchronism introduced in [5] seems promising.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows — Theory, Algorithms, and Applications*. Prentice-Hall, 1993.
2. L. Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16:1–3, 1966.
3. D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, Englewood Cliffs, NJ, second edition, 1992.
4. D. P. Bertsekas and E. M. Gafni. Projection methods for variational inequalities with application to the traffic equilibrium problem. *Math. Programming Study*, 17:139–159, 1982.
5. D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation*. Prentice-Hall, Englewood Cliffs, NJ, 1989.
6. P. Brucker. An O(n) algorithm for quadratic knapsack problems. *Oper. Res. Lett.*, 3:163–166, 1984.
7. I. Chabini and M. Florian. Orthogonal projection based algorithms for convex cost transportation problems with serial and parallel implementations. *Computational Optim. Appl.*, 1996. To appear.
8. O. Damberg and A. Migdalas. Efficient minimization over products of simplices — application to traffic assignment. *Optimization*. Under revision.
9. J. Eckstein and M. Fukushima. Some reformulations and applications of the alternating direction method of multipliers. In W. W. Hager et al., editors, *Large Scale Optimization — State of the Art*, pages 115–132. Kluwer Academic Publishers, Dordrecht, 1994.
10. J. Ericsson. Private communication, 1995. Swedish Road and Traffic Research Institute (VTI).
11. M. Florian. Nonlinear cost network models in transportation analysis. *Math. Programming Study*, 26:167–196, 1986.
12. M. Florian and S. Nguyen. An application and validation of equilibrium trip assignment methods. *Transportation Sci.*, 10:374–390, 1976.

13. G. Gallo and S. Pallottino. Shortest path algorithms. *Ann. Oper. Res.*, 13:3–79, 1988.
14. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM — Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, 1994.
15. D. W. Hearn, S. Lawphongpanich, and J. A. Ventura. Finiteness in restricted simplicial decomposition. *Oper. Res. Lett.*, 4:125–130, 1985.
16. D. W. Hearn, S. Lawphongpanich, and J. A. Ventura. Restricted simplicial decomposition: computation and extensions. *Math. Programming Study*, 31:99–118, 1987.
17. K. Helgason, J. Kennington, and H. Lall. A polynomially bounded algorithm for a singly constrained quadratic program. *Math. Programming*, 18, 1980.
18. H. Jönsson. Private communication, 1995. Swedish Road and Traffic Research Institute (VTI).
19. T. Larsson and A. Migdalas. An algorithm for nonlinear programs over Cartesian product sets. *Optimization*, 21:535–542, 1990.
20. T. Larsson and M. Patriksson. Simplicial decomposition with disaggregated representation for the traffic assignment problem. *Transportation Sci.*, 26:4–17, 1992.
21. S. Lawphongpanich and D. W. Hearn. Simplicial decomposition of the asymmetric traffic assignment problem. *Transportation Res.*, 18B:123–133, 1984.
22. H. S. Mahmassani and K. C. Mouskos. Vectorization of transportation network equilibrium assignment codes. In R. Shandra et al., editors, *Impacts of Recent Computer Advances on Operations Research*, pages 70–81. North Holland, 1989.
23. A. Migdalas. A regularization of the Frank-Wolfe algorithm and unification of certain nonlinear programming methods. *Math. Programming*, 65:331–345, 1994.
24. A. Migdalas. Bilevel programming in traffic planning: Models, methods and challenge. *J. Global Optim.*, 7:381–405, 1995.
25. P. M. Pardalos and N. Kovoor. An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. *Math. Programming*, 46:321–328, 1990.
26. P. M. Pardalos, A. T. Phillips, and J. B. Rosen. *Topics in Parallel Computing in Mathematical Programming*, volume 2 of *Applied Discrete Mathematics and Theoretical Computer Science*. Science Press, New York, 1992.
27. C. Phillips and S. A. Zenios. Experiences with large scale network optimization on the connection machine. In R. Shandra et al., editors, *Impacts of Recent Computer Advances on Operations Research*, pages 169–178. North Holland, 1989.
28. M. Ç. Pinar and S. A. Zenios. Solving nonlinear programs with embedded network structures. In D.-Z. Du and P. M. Pardalos, editors, *Network Optimization Problems: Algorithms, Applications and Complexity*, pages 177–202, World Scientific, Singapore, 1993.
29. M. Ç. Pinar and S. A. Zenios. A data-level parallel linear-quadratic penalty algorithm for multicommodity network flows. *ACM Trans. Math. Software*, 20:531–552, 1994.
30. R. Qi and S. A. Zenios. Parallel decomposition of multicommodity flow problems using coercion methods. In O. Balci et al., editors, *Computer Science and Operations Research: New Developments in their Interfaces*, pages 307–318. Pergamon Press, Oxford, 1992.
31. B. Schieber and S. Moran. Parallel algorithms for maximum bipartite matchings and maximum 0-1 flows. *J. Parallel Distrib. Comput.*, 6:20–38, 1989.