

Självständigt arbete på avancerad nivå

Independent degree project – second cycle

M. Sc. Thesis
Computer Engineering

Scaling an Optimization Engine for a Decision Support System

Joel Emil Sandvik



Mittuniversitetet
MID SWEDEN UNIVERSITY

Mid Sweden University

The Department of Information Technology and Media (ITM)

Author: Joel Emil Sandvik

E-mail address: joelsandvik@gmail.com

Study programme: Master of Science in Engineering - Computer Engineering

Examiner: Tingting Zhang, Ph.D., tingting.zhang@miun.se

Scope: 9600 words exclusive of appendices

Date: 2013-09-16

M.Sc. Thesis within
Computer Engineering D

Scaling an Optimization Engine for a Decision Support System

Joel Emil Sandvik

Abstract

The objective of this survey has been to determine whether an existing linear programming model could be used to implement a system for the optimization with regards to the matching of transports and requests. Real data from Härnösands harbour was enhanced and used to simulate scenarios that would be used and applied within the system. The LINGO-solver was integrated within a system that was developed in .Net to address issues around data in- and output, data persistence, and stability. Performance tests revealed that the system and model performed inadequately on larger datasets. A simpler model was developed that handle a subset of the types of datasets handled by the first model, a subset which includes the data collected from Härnösand harbour. The simpler model was implemented with the AMPL/CPLEX-solver software, and further tests showed that this provided a much improved performance to the system. These results indicate that further investigation might be required regarding the differences in performance between combinations of different solvers and models, and that a complete and production ready system is likely to benefit from incorporation of various models that can be applied to various characteristics of the input data.

Contents

1	Introduction	5
1.1	Background and Problem Motivation	5
1.2	Overall Aim	5
1.3	Scope	6
1.4	Concrete and Verifiable Goals	6
1.5	Outline	7
1.6	Contributions	7
2	Theory	8
2.1	Linear programming	8
2.1.1	the Simplex Algorithm	9
2.1.2	Branch-and-bound	10
2.1.3	Barrier	11
2.2	Decomposition	11
2.3	LINGO	12
2.3.1	LINGO models	12
2.4	AMPL	14
2.5	ILOG CPLEX	14
3	Method	15
3.1	Data	17
3.2	Analysis	17
3.3	A Simpler Model	22
3.4	Model Definition	23
3.4.1	Sets	23
3.4.2	Parameters and Variables	24
3.4.3	Decision Variables and Scenarios	27
3.4.4	Objective Function	28
4	Design	29
4.1	C#/.Net	29
4.2	System Design	31
4.3	Data Transfer	32
4.3.1	Dynamic-link Libraries	32
4.3.2	Unsafe Buffers	32
4.3.3	HTTP	33
4.4	Database Connectivity	33

4.4.1	External Processes	34
5	Results	35
6	Discussion	37
7	Appendices	42
A	LINGO Model	42
B	Lingo Performance	52
C	AMPL Model and Data	55
D	LINGO and AMPL performance	62

1 Introduction

The wider context of this survey is a collaboration between researchers at Mid Sweden University and the ports of Söråker and Härnösand, to develop a transport matching system that can be used in the daily operations at logistical centres in order to optimize the transport of goods in a corridor from the coast of Norway to Finland. The solutions and conclusions presented in this case must be aligned with the work that has been, and which continues to be conducted by others within the scope of this collaboration, and take into consideration the long-term goals of the project.

The survey will involve two parts: A written report and an implementation of the ideas and findings therein. The written report will contain the research behind the system, outline the rationale behind the decisions made before and during the implementation phase, and present and analyse tests performed on the implemented system.

1.1 Background and Problem Motivation

The cost of empty transports is a significant issue both in Sweden and internationally. A more effective use of transport, minimizing the number of empty and partially-filled transport, would have both desirable financial and environmental consequences[1]. This can be achieved by treating transport and routes - in this case from the coast of Norway to Finland, through Sweden - as an optimization problem.

1.2 Overall Aim

Leif Olsson and Aron Larsson have developed a calculation engine that uses network optimization and a mixed integer linear model to provide clients with alternative transport routes[1]. The model has the ability to handle conflicts between cost, time and emissions, and has been implemented using proprietary optimization software. The goal of this survey is to assist in relation to the integration of the current implementation of the model in a system that is practically usable and extendable. The main obstacle is performance, since the performance of the standalone calculation engine is not sufficient as of now. The focus will be on investigating how both the input and output are provided, stored, possibly cached etc. Static statistical sample data has been used for research and development, but, one of the objectives of this survey is to derive ways to enable the

system to handle real world data in real-time.

1.3 Scope

The correctness and implementation of the model developed by Leif Olsson and Aron Larsson, and further developed by Maria Kalinina[2], are not within the scope of this study. If modifications can be made with regards to how or when the calculation engine reads or writes data, such modifications might be suggested, if they are beneficial to calculation runtimes, or if they are necessary to enable performance gains through other means, or if they are necessary for, or facilitate the integration of the calculation engine within the wider system context.

1.4 Concrete and Verifiable Goals

After completion of this survey, the system should work effectively with real-time data. By that we mean that the system should be stable, and always return a result within a reasonable timeframe. For this to be possible, no unnecessary calculations must take place. Input, updates and calculation results must be processed and handled intelligently and efficiently. This behaviour should be confirmed through tests with real-time data.

Some of the unknowns and features that the study will attempt to investigate and implement are:

- **Consistent Data:** It must be possible to dynamically ensure that the state of the data store is updated to reflect bookings that occur as a result of optimizations based on the current state, whether this happens synchronously or asynchronously.
- **Minimized Redundancy:** More than one optimization with the same result should be avoidable by comparing past and present input and evaluating the possible impact of any changes.
- **Relationship between Data and Performance:** What is the relation between the number of transportations, number of customer demands and the time it takes to optimize the distribution of the latter over the former? Is it possible to predict how long the optimization step will take beforehand, with knowledge of the sizes of the inputs sets?
- **Preoptimization:** It is possible that steps can be performed on the input data to filter out irrelevant input before the optimization engine is called thus, a decreased dataset could lead to performance gains.

1.5 Outline

Chapter 2 contains a brief summary of the theories and concepts that the work presented here relies on. Chapter 3 outlines the approach and procedure and the models that are used. Chapter 4 describes the design of the implemented system, the technologies used and the motivations behind those choices. Chapter 5 describes the results that were achieved with the implementation, and the conclusions drawn from those results. Chapter 6 highlights the areas that were insufficiently addressed in this survey, lists problems and open questions that were encountered, and offers ideas for future work.

1.6 Contributions

The work presented here centres around the model developed by Leif Olsson *et al.*. Leif Olsson has also been actively involved with guidance and suggestions based on his great knowledge of the area. The project would also not have been possible without Magnus Eriksson, who suggested the collaboration. Much of the data that was used to develop and test the models and the system was collected by Per Engström in Härnösands harbour. An introduction to integrating the LINGO-solver in a .Net environment was given by Elias Vesterlund. Finally, I would like to thank my partner Charlotte for her patience during this period.

2 Theory

An introduction to the theory of linear programming, and an overview of some of the technologies available for solving linear programming problems is provided in this chapter.

2.1 Linear programming

Linear programming, a form of constrained optimization, attempts to determine the variable assignment that maximizes an objective function while satisfying a set of inequalities. Much research has been performed in this area, because of its economic importance, but the freely available algorithms and their implementations cannot, for the same reason, compare to the commercially available alternatives. A well-known quality of linear programming problems is that they are *NP-complete*. An in-depth discussion of this fundamental concept, or of computational complexity in general, is not within the scope of this report. Briefly, NP, is the set of problems whose solutions cannot be verified in polynomial time, while P is the set of problems that can be solved in polynomial time. This means that P is a subset of NP, but if it is a true subset or not - whether $P = NP$ - is an open problem, in spite of its fame. Understanding of this area is not directly required in this case, but, it is advantageous to realise that, very generally speaking, there are no shortcuts to solving NP-complete problems [3, p. 342]

The problems tackled with mathematical programming have the general form of an objective function, which is subject to constraints on its variables. The subset of problems where the objective is a linear function and the constraints are linear equations or inequalities are *linear programs*. They are models where, broadly speaking, the output is proportional to the input, and differ from *nonlinear programs*, where the objective or constraints use other smooth functions, and from *integer programs*, where integral values are required for some variables. Common abbreviations for the different, not mutually exclusive problem types are:

- **LP**: Linear programming problems;
- **MIPs** or **MILPS**: Mixed integer linear programming problems, sometimes divided into binary MILPS, where variables can only take the values of 0 or 1, and general, where any integer variables are possible;

- **QP/QCP:** Problems with quadratic objective and/or constraints;
- **MIQP/MIQCP:** Mixed integer QP/QCPs.

The model used in this project is an MILP. For MILP:s, a subset of variables will have integer values, and all expressions are linear. Intuitively, a linear function is one that, when graphed, will form a straight line, as opposed to a quadratic or cubic function ([4]).

Algebraic modelling languages are one way around the difficulty of allowing modellers to express a problem in a readable form that can also (after translation) be understood by computers [5, p. xvi].

2.1.1 the Simplex Algorithm

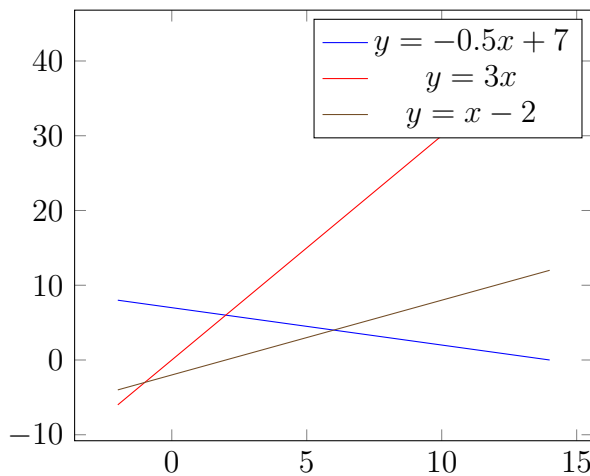
The Simplex algorithm has enjoyed wide adoption since its development in the 1940s [6], and is often the method used for linear optimization problems in textbooks, sometimes even with pen and paper. It is considered one of the most important algorithms of the 20th century [7, p. 22-23]. Consider the simple problem of finding the maximum and minimum values of

$$z = 4x + 5y \tag{2.1}$$

subject to

$$\left\{ \begin{array}{l} x + 2y \leq 14 \\ 3x - y \geq 0 \\ x - y \leq 2 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} y \leq -1/2x + 7 \\ y \leq 3x \\ y \geq x - 2 \end{array} \right\} \tag{2.2}$$

which can be plotted as

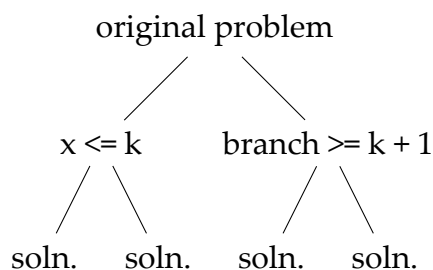


The area of the graph within the three functions is the feasible solution space. Each constraint generates (at most) one edge, and the number of

solutions within the feasible solutions space is infinite. A simple problem, such as this example, could be solved graphically by merely considering the corners of the feasible regions and picking the optimal solution from the maximum value of the equation with the values of x and y inserted into these corners. Intuitively, this must be true, since for any point on the vertices other than the corners, any point on the vertex closer to the corner must always result in a higher objective value. Algorithmically, the Simplex method starts at the origin and moves along an edge that increases the objective value, or does not change it, until it reaches another point - another corner of the feasible solution space -, where it continues with the next iteration. If the edge continues infinitely then the linear program is unbounded. The stopping condition of the algorithm is that no neighbouring point gives a higher value of the objective function than the current point. This procedure is normally conducted in steps using *slack variables* and a *tableau* [8, p. 3-12, 59-70].

2.1.2 Branch-and-bound

The branch-and-bound-method, applied to an MILP, starts with *relaxation* of the original problem, by removing all integrality restrictions. If the solution to the resulting LP satisfies the constraints on the original MILP, that will also now have a solution. If not, a variable is chosen, that should be an integer but is fractional in the found solution, and constrained with an upper and a lower limit, so that, if the fraction is x , $x > k$, $x < k + 1$, where k is an integer, x is constrained to be $x \leq k + 1$, $x \geq k$. This creates two subproblems - both MILPs - that are solved using the same method. This clearly leads to a binary search tree, where the nodes are the MILPs created by applying the steps just described to the parent node, the root is the original problem, and the leaves are either nodes that have yet to be branched, or solutions to the original problem [9]:



Branch-and-cut is another method that combines branch-and-bound with cutting planes.

2.1.3 Barrier

The *barrier method* or *interior point method*, is more complex than the Simplex method, but is worthy of a mention since it is polynomial in time as opposed to the Simplex algorithm, which is non-polynomial. Very broadly speaking, barrier methods (the terminology is not clear here, because of some controversy around Karmarkars projective LP-algorithm and its equivalence with the projected Newton method) apply steps to transform the corners of the solution space of the (linear or nonlinear) optimization problem, or the centres of the those corners, to geometrical shapes that can be optimized more easily ([10], [11]).

2.2 Decomposition

Decomposition is a term for an approach to problem solving that has some overlap with the, perhaps, more familiar term *divide and conquer*. In brief, it means tackling a problem by dividing it into subproblems that are more manageable and can be processed independently, and combining the solutions to a solution to the original problem. *Trivially parallelizable* problems in this context are those where, for example, every constraint involves only one subvector of a variable, so that each subproblem involving one of these subvectors can be solved separately. Where there is coupling between the subvectors, more interesting techniques are necessary.

Imagine a simple scenario where the goal is to minimize the sum of two functions. If no variables are shared between the two functions, they can be treated individually - and, importantly from a practical point of view, in parallel. If that is not the case, the variable or variables shared between the two functions are called *complicating*, *public*, or *boundary* between the subproblems, while the other variables are *private* to either of the two functions. Since the problem would be *separable* if the complicating variable was fixed, it is equivalent to the problem involved in determining the optimal value of the complicating variable. For most models, including the one being worked on in this case, there is more than one coupling constraint between more than two subproblems. In *primal decomposition* every subproblem is optimized separately, using global shared variables, and each subproblem is optimized to meet the fixed boundaries, which are iteratively minimized. If *dual decomposition* is used, each subproblem uses a different (probably) value for the boundary variables. To reflect these differences, extra cost is added, and this cost is updated iteratively in order to make the total solution consistent ([12]).

2.3 LINGO

Work on the LINGO algebraic modelling system began in 1987, when standard PC:s, thanks to the adoption of 32-bit architecture, had become sufficiently fast, and had sufficient memory, to solve real world optimization problems. One of the design principals behind it was that it should be relatively easy to be used even by those who lacked the knowledge of the fundamentals of math programming. Therefore, the LINGO system solvers execute as part of the algebraic modelling language, so that an appropriate solver is chosen through analysis and reduction of the model, without any direction from the user. The solver used is the one that is most suitable for the class of the model [13].

LINGO can handle the problem classes mentioned above (2.1), and a number of additional classes [14, p. 11]. It uses a variant of the Simplex method, or a barrier solver, for linear problems, and a branch-and-bound method for integer problems [14, p. 660], and decomposition of suitable problems can be applied [14, p. 340].

2.3.1 LINGO models

LINGO optimization models are defined through one *objective function*, that expresses what should be optimized, *variables* and *decision variables*, in which the aim of the optimization is to discover the optimal values for, and *constraints* that defines the limits for the possible values of the variables.

Functions: The model uses three submodels to minimize cost, time and emissions. The function in listing 2.1 minimizes the cost.

Variables: The model uses a number of variables to define transports, customer demands on the routes of the network, the fulfilment and distribution of customer demands in relation to the available transports etc., e.g. *MaxTrp* - maximum allowed volume on a route with a certain transporter; *StartTimeTrp* - when a certain customer will be ready to send a commodity on a certain route; *Shortage* - the quantity in volume of a customer's demand that could not be met (after optimization).

Constraints: The model defines a variety of constraints on the outputs and the relationships between the inputs. The first statement in listing 2.2 declares that for no route can the quantity transported exceed the available capacity on the route. The second statement sets the total cost regarding the sum of the costs for the transport on the different routes.

The sets used in the definitions above are an integral part of the LINGO modelling language. With the concept of sets, the matrices of relationships between the variables in a model, which could be described in large equations, can be expressed in something similar to mathematical nota-

Listing 2.1: Example of LINGO calculation.

```
CALC:
@FOR (POINTS( b):
@SOLVE(MIN_COST);
Costlimit=TotalCost;
Shortcostlimit=Shortcost;
TimeLimit=TotalTime;
EmissionLimit=MaxEmissions;
TC(b)=TotalCost;
SC(b)=Shortcost;
TT(b)=TotalTime;
ME(b)=MaxEmissions;
@FOR(Transport(i,j,p,q,v):
Freight1(i,j,p,q,v,b)=Freight(i,j,p,q,v));
@FOR(Pointscust(v,l,k,b):
Shortage1(v,l,k,b)=Shortage(v,l,k));
@FOR(Pointscust(v,l,k,b):
Sendtocust1(v,l,k,b)=Sendtocust(v,l,k));
@FOR(Volpoints(i,j,p,q,v,l,k,b):
Vol1(i,j,p,q,v,l,k,b)=Vol(i,j,p,q,v,l,k));
...

```

Listing 2.2: Example of LINGO definitions.

```
@For(Transport(i,j,p,q,v):@Sum(Distrib(i,j,p,q,v,l,k)
:Vol(i,j,p,q,v,l,k))<=MaxTrp(i,j,p,q,v));
TTotalCost=@sum(Transport(i,j,p,q,v):
TrpCost(i,j,p,q,v)*Freight(i,j,p,q,v));

```

tion. Sets are groups of related entities that all have differing values for the same set of attributes. The set *Transport* contains all the transports in the input, and the variables i, j, p, q, v are what defines them. $T_{TotalCost}$ is the sum of the trip cost $TrpCost$ multiplied by the freight for all transports. A *derived set* is defined through other sets that can be primitive or also derived. The set *ComCustSup* in the LINGO model is derived from the parent sets *ComType* and *Nodes*, and contains combinations of commodity types between a start and an end node. Refer to appendix A for the full LINGO implementation of the model.

2.4 AMPL

The AMPL ("A Mathematical Programming Language") modelling language is similar to the LINGO modelling language in syntax. Models are defined using parameters, variables, sets, an objective function, and constraints. It was developed by Robert Fourer, David Gay and Brian Kernighan at the Bell Laboratories ([15]). AMPL, the modelling language, is implemented by the AMPL software, which parses models and attempts to simplify them before they are passed on to a user-specified solver. There are a number of different solvers available for AMPL, all with different strengths and weaknesses, some of which are commercial, with some being free to use, extend and modify. As opposed to LINGO, there is no API available to facilitate communication between a CLR-process (see section 4.1) and the AMPL-parser. AMPL-models are declared with a similar syntax to that used by LINGO. For more details, see chapter 3.

2.5 ILOG CPLEX

One of the solvers that can be used with AMPL is the CPLEX-optimizer, originally developed by Robert E. Bixby and made available in 1988 by CPLEX Optimization Inc., acquired by ILOG in 1997, and - through the acquisition of ILOG - by IBM in 2009[16]. It can solve linearly or quadratically constrained problems for which the input can be continuous functions or integer values. When there are no gaps between the values in the objective functions, CPLEX applies Simplex algorithms as well as barrier algorithms ([17, p. 28-29], [18, p. 37]), and it uses a branch-and-bound-method for integer problems ([18, p. 55]). It can also handle quadratic objective functions and certain kinds of quadratically constrained problems, and mixed integer linear programs. By default, CPLEX automatically determines which algorithm to apply to a problem ([17, p. 321]).

3 Method

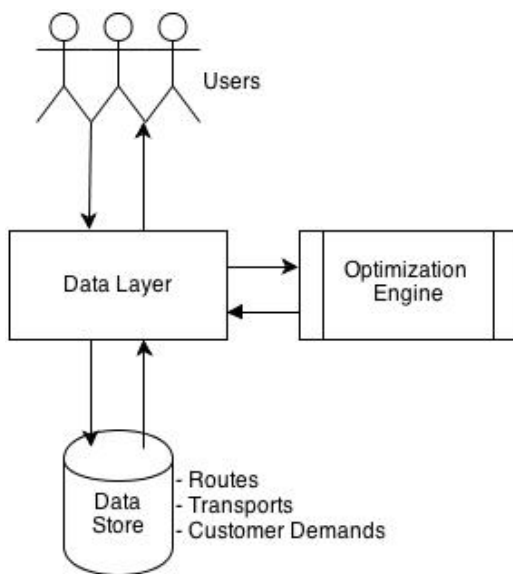
Fourer *et al.*, present this generic approach to linear programming ([5, p. xv]):

1. Formulate a model, the abstract system of variables, objectives, and constraints that represent the general form of the problem to be solved.
2. Collect data that define a specific problem instance.
3. Generate a specific objective function and constraint equations from the model and data.
4. Solve the problem instance by running a program, or solver, to apply an algorithm that finds optimal values of the variables.
5. Analyse the results.
6. Refine the model and data as necessary, and repeat.

The scenario for this project is different in that a significant amount of effort has already been involved in developing a model that is used as a starting point. In addition, there is a dataset collected for the problem instance. Based on the approach outlined above, that leaves two main directions for the project to take: A) Steps 1 - 2 above have been performed, and steps 3 - 5 remain to be addressed here, or B) all the steps above have been performed, and our focus must be on making the results of that process useful in practice, by identifying and focusing on the technical issues surrounding the perceived usecase (see page 6). Since the practical usecase has been defined, it should be used to determine the direction which should be taken. The steps for this particular project must be:

1. Design and implement a framework that makes the model practically usable.
2. Test and profile the model using the framework, improving the framework (with regards to the parameters previously defined).
3. Determine from the test results, whether if A or B is the present scenario.
4. If A) perform steps 3 - 5, if B) project completed.

As mentioned in chapter 2, the field of linear programming and constrained optimization is one of great economic significance, and has been heavily researched. The advantages of choosing the commercial LINGO solver over other alternatives is not under investigation here, but it is safe to state that it would require resources on a different scale to develop and implement a comparable alternative. The goal of this survey is to determine how the optimization model and its execution can be integrated into a practically usable system. In defining the goals of this project, emphasis was placed on effectiveness and stability in a real world scenario. The focus must therefore be on the input and output of data. A system is imagined where the input is a changing set of available transports over a set of routes, and a set of customer demands on the same routes, and the output is the optimal distribution of these demands in relation to the available transport. In figure 3, the first focus is the "Data Layer", and, as a



dependency, the "Data Store". In order to simulate a fully implemented system, a data store is required. A first step in creating our environment is to define a schema that covers our data, and to produce and/or parse an existing dataset and transfer it to the data store. Data must then need to be read from the data store and fed to the optimization engine. Filtering and modification of the data could be advantages during this stage. The engine need to be integrated with the data layer being implemented, and its execution controlled by it. Output from the optimization should be used to update the state of the data store, and, ultimately, this will be displayed to the user.

3.1 Data

A larger dataset is required in order to simulate real world usage than which can be feasibly produced manually. One such dataset was collected in Härnösand harbour during the summer of 2012, logging the transport of pellets during that period. This data meets the requirements as being sufficiently similar to real world input, but only lists the transport routes and capacities. The costs involved regarding the transporting can also be derived, which means that duration and emissions remain. The method used to estimate the first of these variables is to establish the coordinates of the postcodes given for the locations, or nodes, in the given dataset, and then estimate the duration between them for every transport. Google offers an API that provides limited access to its vast geographical information resources, and enabled the retrieval of the distance and the approximate duration of a trip between two geographical locations provided as coordinate pairs [19]. The interface is a URL that accepts GET-requests containing the required parameters, returning an XML-formatted response. With the duration variable set for every transport, the distance variable can be used to approximate emission levels. Previous analysis and enrichment of a subset of the data contains the emission levels for some of the transportations, gathered from an API provided by Green Cargo ([20]). Continuing this work for the remaining majority of the transport is not practically feasible, so instead *linear regression* is used to roughly estimate the emissions [21]. This is performed using a simple algorithm that takes an ordered - ascending - dictionary of distance to emission key-value-pairs and outputs the slope of the trend line, a , and intercept of the trend line, b , in the formula $y = ax + b$. This then amounts to a sufficiently large and sufficiently realistic data set to set up a test environment. Data is parsed, enriched through a series of filters, performing the operations described above, and loaded into a normalized relational database, from which it can be read both quickly and easily.

3.2 Analysis

With the focus on the flow of data to and from the optimization engine, the system must be implemented before it is possible to observe and draw conclusions regarding how the LINGO-solver can be fitted into the wider system. There are no real changes to the optimization model at this stage, the only confirmation being that it behaves as expected. It is also important that interoperability between the optimization engine and the surrounding system is seamless and that the different parts of the system are appropriately decoupled etc.. Behaviour can be specified and measured

Listing 3.1: Linear regression.

```
double xAvg = 0;
double yAvg = 0;

foreach(int dist in distances)
{
    xAvg += dist;
    yAvg += distanceEmissions[dist];
}

xAvg = xAvg / distances.Count;
yAvg = yAvg / distances.Count;

double v1 = 0;
double v2 = 0;

foreach(int dist in distances)
{
    v1 += (dist - xAvg) * (distanceEmissions[dist] -
        yAvg);
    v2 += Math.Pow(dist - xAvg, 2);
}

_a = v1 / v2;
_b = yAvg - _a * xAvg
```

with tests. Satisfactory design is more difficult to assess, but some source code analysis tools do exist for that purpose. With a black box component at the centre of the system, feeding off and processing our data, it is difficult to envision any applicable caching functionality, but the optimization step can and should be bypassed when nothing has been added or updated in the relevant subset of the input data. There are also pre-optimizations that can be performed on the input parameters, to limit the amount of data that the optimization engine is required to handle. Without the capacity to change the core behaviour of the optimization model, these steps must be stateless and one-directional, and the only way to determine how useful they are is through repeated testing and timing of performance. The results of such testing are displayed in appendix B. The table contains three columns of average times; one where all requests were passed to the solver, even if some of them were not relevant for the transport requests (they were transports on other routes), one where irrelevant transport were filtered out in a pre-processing stage so that they never reached the solver, and one where an additional preprocessing step was applied as follows:

- For all transport requests, one at the time
 - Pick the request with the highest volume
 - From the available transport that meets the requirements (considering volume and times), pick the one that has the lowest duration, cost and emissions
 - If it is found:
 - Use its free capacity (that has not been used for another request) to meet the volume demand and mark the amount of its load capacity that is necessary to be used
 - If it is not found:
 - While the total capacity of the remaining transport that meet the requirements is greater than the total volume:
 - Pick the transport that has the highest duration, cost and emissions
 - If it is found:
 - Remove the transport, and remove its capacity from the total capacity
 - If it is not found:
 - Break

Note that this algorithm only removes transports that cannot possibly be optimal, since it requires all three factors - cost, duration and emissions

- to be more favourable for all transports in the selected set than for any transport of those that are filtered out.

Simple filtering regarding the irrelevant transportation for the problem at hand, because they are for routes that are not present in the set of transport requests, can also be conducted within the model itself by defining a subset of the set of available transport, and using that everywhere that the original set would otherwise have been used:

Listing 3.2: LINGO transport filtering.

```
RelevantRoute (ComType , Nodes , Nodes) | @in (ComCustSup ,  
    &1 , &2 , &3) ;  
RelevantTransport (Transport) | @in (RelevantRoute , &5 ,  
    &1 , &2) ;
```

This was also tested for performance, and no obvious advantage could be detected.

We see in appendix B that the solution runtime quickly becomes very long even with a relatively modest number of requests and transports. Other factors, such as reading and writing of output, have been benchmarked and deemed irrelevant for the total runtime. With primitive pre-processing of the input, the runtime improves, but the performance is still not what would be required from a practically usable system. The fact that such simple methods can have such a noticeable effect also raises questions about the suitability of the model and/or the choice of solver for the dataset at hand. As has been established previously, the model used for benchmarking is not under investigation, nor is there any reason to suspect that it does not perform as expected. That leaves two possible areas of improvement: the solver, and the fit between the model and data. LINGO being propriety, closed-source software makes it difficult to gain an understanding of how it works, or to determine if the benchmarked performance is abnormal or to be expected. Fully understanding the intricacies of the software is also not within the scope of the project. There are, however, a number of flags and options that can be passed to LINGO simply by adding them to the command used to invoke the process ([14]). For this project, the focus is on those that, as they are described in the documentation, appear likely to have an impact on the runtimes in this scenario:

- MXMEMB - Setting an upper limit on the memory allocated for the model generator: Increasing the memory has no noticeable effect on the runtimes.
- HURDLE - Setting a tolerance on the objective value of the solution:

Could greatly reduce solution time, but is not practically useful with dynamic data.

- REDUCE - Telling LINGO to remove constraints and variables that will not have an effect on the solution, before solving: Can be turned on or off, or left pending for the solver to decide upon analysis. **Enabling this option does appear to lead to a better performance, but the gains are not comparable to those achieved through pre-optimization.**
- BRANPR - Forcing the solver to give priority to binary variables during the branch-and-bound procedure: The documentation mentions that the default where LINGO determines the next variable for branching usually gives the best results, and no improvement is gained by forcing binary priority.
- DECOMP - Allowing LINGO to solve the independent problems sequentially if the constraint matrix is decomposable into a series of block structures: This option cannot be leveraged in a framework that relies on reading input from memory buffers, see chapter 6 for further discussion.
- LCORES - Allowing LINGO to run N different solvers on N different cores - if possible on the machine at hand: The fact that this option has no noticeable effect on a multi-core system might be an indication that the solver, used when only one core/solver is used for the solution, is optimal.
- LOOPOP - Enable loop optimization, removing unnecessary inner loops, improving inefficient loops: That enabling this option has no effect on the solution times might again be an indication of the soundness of the model.

With no progress made from attempting to tune the LINGO software, we turn to the possible mismatch between model and data. Consider the last three of the items listed in the beginning of this chapter:

- Solve the problem instance by running a program, or solver, to apply an algorithm that finds optimal values of the variables.
- Analyse the results.
- Refine the model and data as necessary, and repeat.

Looking at the model at hand, a considerable amount of its logic concerns situations where a trip spans multiple legs, with the same request - or parts of it - being transported with one transport from node A to node B, and then further from node B to node C with a different transport. However, the dataset at hand, actual data gathered at Hännösand harbour, is not of that nature. Here, all the transports have the same origin, only the destination varies. A simpler model suffice for this particular dataset. The model also performs the calculation in three sequential steps; a single calculation step would, intuitively, be faster.

3.3 A Simpler Model

Instead of removing parts from the existing model, a new model has been developed from scratch, see appendix C. It is implemented in AMPL, and the logic is kept to a bare minimum for the data that it is applied to.

Writing the model in AMPL allows us to test another solver than that used by LINGO. Since there are over 40 different solvers available for AMPL (many of them constrained by pricing and licenses) ([22]), we focus on the CPLEX solver (see page 14). The objective is not an exhaustive evaluation of solvers, but to find one possible solution to the problem at hand, and to indicate possible areas of future work.

The model created for AMPL/CPLEX is simpler than the original model for LINGO, because of the restrictions it places on the input data. It follows the general structure of AMPL-models with variable declaration, an objective function, and constraints:

Listing 3.3: AMPL variable declaration.

```
set Transport within {Nodes,Nodes, Leg_Id,  
    LoadCarrier, ComType};  
param MinTrp {Transport};  
param MaxTrp {Transport};  
...
```

Listing 3.4: AMPL objective function.

```
minimize optimum: total_cost * c_factor +  
    total_emissions * e_factor + total_time *  
    t_factor;
```

Listing 3.5: AMPL constraints.

```
zero_volume {(m_m,i_i,j_j,i,j,k,l,m) in Distrib}:  
    DistribDecide[m_m,i_i,j_j,i,j,k,l,m] = 0 ==>  
    Vol[m_m,i_i,j_j,i,j,k,l,m] = 0;  
transport_use{(m_m,i_i,j_j,i,j,k,l,m) in Distrib}:  
    DistribDecide[m_m,i_i,j_j,i,j,k,l,m] = 1 ==>  
    TransportDecide[i,j,k,l,m] = 1;  
...
```

3.4 Model Definition

3.4.1 Sets

The set of all transport, where i is the origin, j is the destination, k is the identity, l is the load carrier, and m is the commodity:

$$\text{Tr}\{i, j, k, l, m\}$$

The set of all customer demands, where o is the origin, p is the destination, and n is the commodity:

$$\text{Dem}\{n, o, p\}$$

The set of all potential uses of transports for demands, where k is the transport identity, l is the transport load carrier, m is the transport commodity, i is the transport origin, j is the transport destination, n is the cus-

tomer demand commodity, o is the customer demand origin, p is the customer demand destination:

$$\begin{aligned}
 D_{n,o,p,i,j,k,l,m} = & \\
 & i, j, k, l, m \in \text{Tr} \wedge \\
 & n, o, p \in \text{Dem} \wedge \\
 & m = n \wedge i = o \wedge j = p \wedge \\
 & t_{n,o,p}^{\text{sdem}} \leq t_{i,j,k,l,m}^{\text{strp}} \wedge \\
 t_{n,o,p}^{\text{mdem}} \geq & t_{i,j,k,l,m}^{\text{strp}} + t_{i,j,k,l,m}^{\text{trp}}
 \end{aligned}$$

3.4.2 Parameters and Variables

3.4.2.1 Cost Related

Factor used to scale the cost:

$$C_{\text{factor}}$$

Potential cost of a transport with identity k , load carrier l , commodity m , going between i and j :

$$C_{i,j,k,l,m}^{\text{trp}}$$

Actual cost of a transport with identity k , load carrier l , commodity m , going between i and j :

$$C_{i,j,k,l,m}$$

The total cost of transportation:

$$C_{\text{total}} = \sum_{n,o,p,i,j,k,l,m \in D} V_{n,o,p,i,j,k,l,m} * C_{i,j,k,l,m}$$

or

$$C_{\text{total}} = \sum_{n,o,p,i,j,k,l,m \in D} C_{i,j,k,l,m}$$

depending on whether the cost given is per volume unit or per transport.

3.4.2.2 Emissions Related

Factor used to scale the emissions:

$$E_{\text{factor}}$$

Potential emissions from a transport with identity k , loadcarrier l , commodity m , going between i and j :

$$e_{i,j,k,l,m}^{\text{trp}}$$

Actual emissions from a transport with identity k , loadcarrier l , commodity m , going between i and j :

$$e_{i,j,k,l,m}$$

The total emissions from all of the transport:

$$E_{\text{total}} = \sum_{i,j,k,l,m \in \text{Tr}} e_{i,j,k,l,m}$$

3.4.2.3 Time Related

Factor used to scale the duration:

$$t_{\text{factor}}$$

Potential duration of a transport with identity k , loadcarrier l , commodity m , going between i and j :

$$t_{i,j,k,l,m}^{\text{trp}}$$

Actual duration of a transport with identity k , loadcarrier l , commodity m , going between i and j :

$$t_{i,j,k,l,m}$$

The total duration of all transports:

$$T_{\text{total}} = \sum_{i,j,k,l,m \in \text{Tr}} t_{i,j,k,l,m}$$

Maximum transport time (latest arrival) of a customer demand between o and p for commodity type n :

$$t_{n,o,p}^{\text{mdem}}$$

Start time for a customer demand between o and p for commodity type n :

$$t_{n,o,p}^{\text{sdem}}$$

Start time of a transport with identity k , loadcarrier l , commodity m , going between i and j :

$$t_{i,j,k,l,m}^{\text{sttrp}}$$

3.4.2.4 Volumes

The volume of the customer demand between o and p of commodity n :

$$cd_{n,o,p}$$

Minimum volume that can be transported on a transport with identity k , load carrier l , commodity m , going between i and j :

$$V_{i,j,k,l,m}^{\text{mintrp}}$$

Maximum volume that can be transported on a transport with identity k , load carrier l , commodity m , going between i and j :

$$V_{i,j,k,l,m}^{\text{maxtrp}}$$

The volume transported on a transport with identity k , load carrier l , commodity m , going between i and j to meet a customer demand of commodity n , between o and p :

$$V_{n,o,p,i,j,k,l,m}$$

The total volume requested to be delivered by customers:

$$Re = \sum_{n,o,p \in Dem} cd_{n,o,p}$$

The total volume delivered by all transports, for all customer demands:

$$De = \sum_{n,o,p,i,j,k,l,m \in D} V_{n,o,p,i,j,k,l,m}$$

The total volume delivered should match the total volume requested:

$$De = Re$$

Maximum volume constraints must be upheld for all transports:

$$\forall \{i, j, k, l, m\} \in Tr : \\ \sum_{n,o,p,i,j,k,l,m \in D} V_{n,o,p,i,j,k,l,m} \leq V_{i,j,k,l,m}^{\text{maxtrp}}$$

Minimum volume constraints must be upheld for all transports:

$$\forall \{i, j, k, l, m\} \in Tr : \\ \sum_{n,o,p,i,j,k,l,m \in D} V_{n,o,p,i,j,k,l,m} \geq V_{i,j,k,l,m}^{\text{mintrp}}$$

The volume transported on a route cannot be greater than that requested on that route:

$$\forall \{n, o, p\} \in Dem : \\ \sum_{n,o,p,i,j,k,l,m \in D} V_{n,o,p,i,j,k,l,m} \leq cd_{n,o,p}$$

3.4.3 Decision Variables and Scenarios

The decision variable is 1 if the transport with identity k , loadcarrier l , commodity m , going between i and j is used for the demand of commodity n , between o and p :

$$decd_{n,o,p,i,j,k,l,m}$$

The decision variable is 1 if the transport with identity k , loadcarrier l , commodity m , going between i and j is used:

$$dect_{i,j,k,l,m}$$

If a transport is not used, the actual cost, emissions and duration for that transport are zero:

$$\begin{aligned} \forall \{i, j, k, l, m\} \in D : dect_{i,j,k,l,m} = 0 \Rightarrow \\ c_{i,j,k,l,m} = 0 \\ \wedge \\ e_{i,j,k,l,m} = 0 \\ \wedge \\ t_{i,j,k,l,m} = 0 \end{aligned}$$

If a transport is used, the actual cost, emissions and duration for that transport are equal to the potential values:

$$\begin{aligned} \forall \{i, j, k, l, m\} \in D : dect_{i,j,k,l,m} = 1 \Rightarrow \\ c_{i,j,k,l,m} = c_{i,j,k,l,m}^{trp} \\ \wedge \\ e_{i,j,k,l,m} = e_{i,j,k,l,m}^{trp} \\ \wedge \\ t_{i,j,k,l,m} = t_{i,j,k,l,m}^{trp} \end{aligned}$$

If a transport is not used to meet a demand, the volume for that transport/demand combination is zero:

$$\begin{aligned} \forall \{n, o, p, i, j, k, l, m\} \in D : decd_{n,o,p,i,j,k,l,m} = 0 \Rightarrow \\ v_{n,o,p,i,j,k,l,m} = 0 \end{aligned}$$

If a transport is used for any customer demand, that transport is (obviously) used:

$$\begin{aligned} \exists \{n, o, p, i, j, k, l, m\} \in D : decd_{n,o,p,i,j,k,l,m} = 1 \Rightarrow \\ dect_{i,j,k,l,m} = 1 \end{aligned}$$

If a transport is not used for any demands, it is not used at all:

$$\begin{aligned} \nexists \{n, o, p, i, j, k, l, m\} \in D : decd_{n,o,p,i,j,k,l,m} = 1 \Rightarrow \\ dect_{i,j,k,l,m} = 0 \end{aligned}$$

3.4.4 Objective Function

Mimimize the sum of the scaled total cost, emissions and duration:

$$\mathbf{min} \mathbf{z} = C_{total} * c_{factor} + E_{total} * e_{factor} + T_{total} * t_{factor}$$

4 Design

The implemented system has four main areas of functionality:

- Parse and persist input;
- Read persisted data;
- Run the optimization;
- Output and persist the results of the optimization.

Of these, the first is specific to this project, and would not - in its current form - be included in a production environment. Still, the same technology - .Net and C# 4.5 - has been used for the parsing and persisting of test data. In this manner the introduction of additional technologies is avoided, and additionally, C# 4.5 has convenient support for the reading and writing text, made easier through its integration of functional concepts into what was initially a procedural object oriented language.

4.1 C#/.Net

The .Net platform was introduced by Microsoft in 2002. C#, the platform's primary language, was viewed as a Windows-specific alternative to Java at this time, and even though both languages are part of the C family of languages, C# has evolved more rapidly into the state where it contains several advanced features that its counterpart lacks. Unlike C and C++, and in a similar manner to that of Java, C# is not compiled to machine code, but is executed in a *runtime layer*, or *virtual machine*, called a *Common Language Runtime* (CLR). Important (not unique) features of the language are: automatic memory management, commonly called *garbage collection*; generic types and members (since version 2.0, circa 2005); extension methods, which provide the ability to extend the functionality of a type without subclassing it (since version 3.5, circa 2008); high-level constructs for simplifying multi-threaded code (version 4.5 in particular). Overall, the strength of C# is probably its balance between the ease and speed of development on the one hand, and its robust, performant code on the other. The biggest downside - not of C# as a language, but of the .Net platform that is its primary environment - is that it is tightly coupled with the Windows platform and other Microsoft technologies [26, p. 3-6]

Listing 4.1: Using Linq to parse a text file.

```
private List<Transport> ReadTransports(string
    filePath)
{
    string[] allLines = File.ReadAllLines(filePath);
    var query = from line in allLines
                let data = line.Split(',')
                select new
                    Transport(_nodes[data[0]],
    ...
    return query.ToList();
}
```

Listing 4.2: Using LINQ to aggregate numeric properties.

```
double aggregatedCost = transports.Sum(x =>
    x.LoadCost);
double aggregatedEmission = transports.Sum(x =>
    x.Emissions);
double aggregatedTime = transports.Sum(x =>
    x.Duration);
```

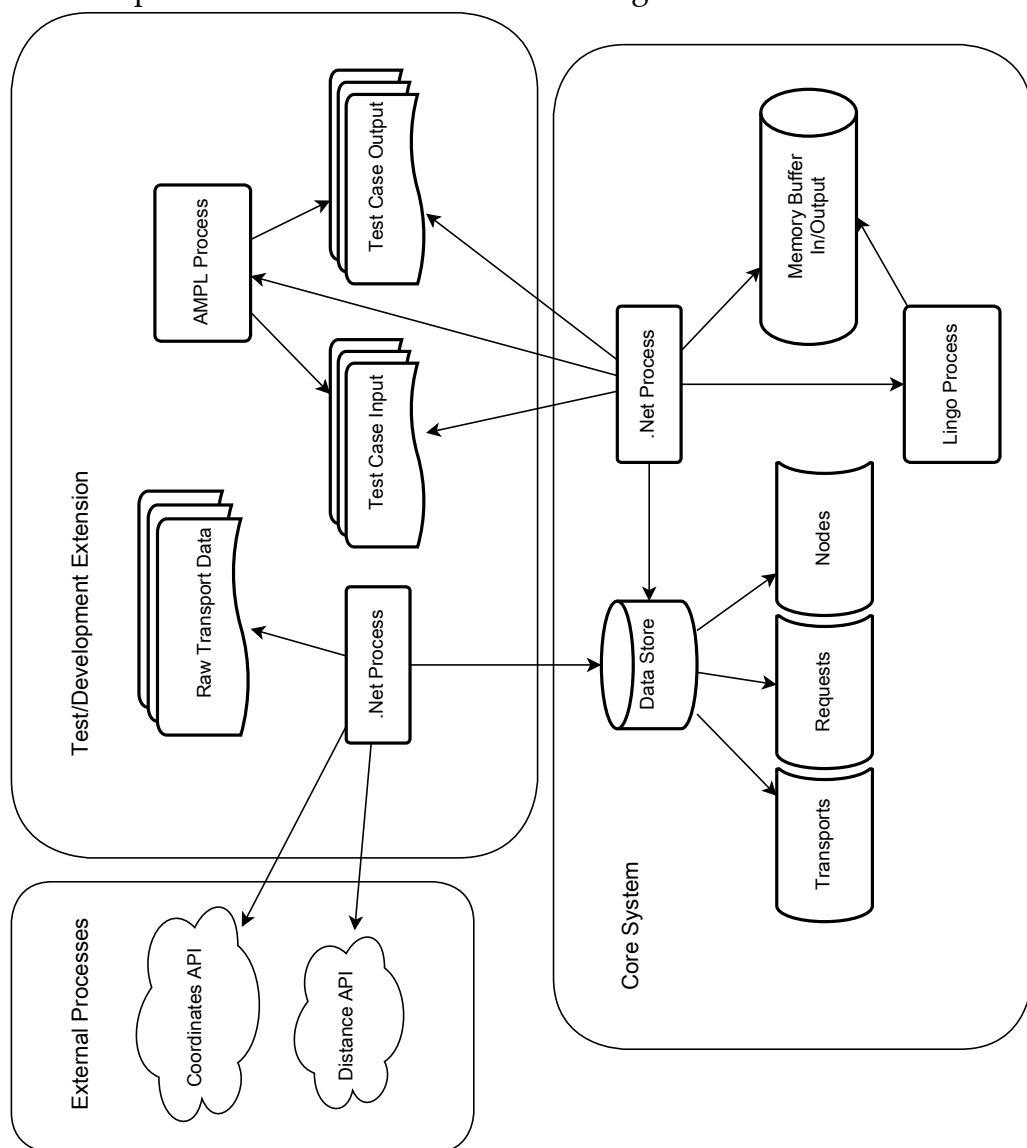
LINQ and *lambda statements* ([27], [28]) are two latter-day features that were leveraged in this project. They deserve to be highlighted as examples of high-level aspects of the language that increase the development speed, and the fact that they are available together with low-level concepts such as unsafe memory buffers which causes the language to stand out. The snippet from a parser class in 4.1 shows how succinctly LINQ allows an application programmer to parse lines from a comma-separated input file into data object instances.

The snippet in 4.2 is a very simple example of how lambda expressions are used to aggregate certain properties on the *Transport* data objects.

These expressions are more concise than their procedural for-loop-equivalents, and by limiting the scope of the variable keeping state during the iterative totalling of the properties, the risk of side effects or inconsistency is reduced.

4.2 System Design

The complete system can be divided into the core part and a pluggable test- or development part. The core part is the system that a production system would be based on, hopefully without any major modifications. The development part contains, primarily, the functionality required for processing and reading test data, but, the AMPL-implementation and the classes used to communicate with and control the AMPL/CPLEX-process also falls within this. A full overview of the system must also reference the external processes that are used for enriching the test data.



4.3 Data Transfer

4.3.1 Dynamic-link Libraries

LINGO comes with a *Dynamic-link library* (DLL), which provides some interoperability with the .Net platform and is one reason why Microsoft's platform was chosen for this project. It is used in this case for communication between the main process and the LINGO process. DLLs are essentially shared libraries that can be mapped to the address space of the executing process. From an application programmer's perspective, they often provide a set of methods that enables the executing process to invoke and control an external process [29, p. 342].

4.3.2 Unsafe Buffers

The LINGO DLL does not provide methods for passing data to, or retrieving data from, LINGO directly. Instead, pointers to *unsafe buffers* are passed to LINGO via the DLL, so that data can subsequently be written and read to and from these. C# on Microsoft's .Net platform promotes statically verifiable memory safety. Access to arbitrary memory addresses is not possible, nor is it possible to obtain the exact address of an object. This can be circumvented with *unsafe code*. Pointers are allowed inside a code block marked with the keyword *unsafe*, which signals caution to the programmer and readers of the source code. The normal runtime checks do not apply to operations that rely on pointer manipulation ([23]). The, perhaps, most immediate danger of this is the possibility of *buffer overflow*, where memory adjacent to the intended buffer is overwritten [24]. Common reasons for accepting the risks that are associated with access to pointers is performance, or the need for direct access to the operating system or memory mapped devices.

An alternative to unsafe buffers for returning optimization results from LINGO, is to use the report functions that the solver provides. Instructions that decide if, how and where output from optimizations are printed to the file system can be combined with print statements (*@Text*) to output values and variables to a file in a format that is easy to parse by another process ([14]). The advantages of this method are the value that it provides as automatic auditing, and that it reduces the complexity of the system by maintaining the number of memory buffers to a minimum. A possible additional advantage of the last feature, is that it could lead to a better performance. This might not be obvious, since the unmanaged code should have less overhead than code that is managed by the runtime environment. However, unmanaged memory is allocated on the same heap as

managed memory and if the buffers are sufficiently large, if they appear in large numbers, or if they have a longer lifespan, they might end up causing a negative impact on the performance of the garbage collection. The buffers must be fixed on the heap (with the *fixed* keyword), so that their memory addresses are guaranteed not to change during the phase of the garbage collection, where allocated memory is moved around to minimize fragmentation. Having to work around several larger areas of the heap can increase the memory and CPU usage by the garbage collector (p. 235-239 [25]).

4.3.3 HTTP

The development section of the system displayed in 4.2 clearly has dependencies on external systems. All communication with those use the *Hypertext Transfer Protocol* (HTTP), which the reader is probably familiar with. Since all calls are one-sided retrievals - nothing is updated in-, deleted from-, or added to the external systems -, they all use the GET-request methods declared in the APIs. The parameters of the calls are provided as part of the request string used to call the endpoint URLs. No status codes were handled specifically other than the standard 200 (OK). This is acceptable since the external dependencies are localized to the development section of the system.

4.4 Database Connectivity

Open Database Connectivity (ODBC) is a specification for a database API. Most developers come into contact with it in the form of *drivers* for different *DBMSs*, or databases. These act as middleware between a calling client application and the database, and allows the client call functions on the ODBC-driver, using external libraries or, support that exists in the execution environment, instead of communication directly with the database with the DBMS-dependent protocol ([30]). The ODBC-driver used for database connectivity for this project is the *SQLite ODBC Driver*, which can be found at <http://www.ch-werner.de/sqliteodbc/>.

One common reason for using ODBC-drivers that is of particular relevance for this project is data store transparency: one ODBC-driver can often be easily replaced by another, and, with it, the data store itself. *SQLite* is a lightweight database often used for test and development purposes, but seldom for production systems (the most common exception being embedded systems). While it is a good choice for the development phase, it is most probably that it would be replaced with a more robust and performant relational database before the system could be considered production-

Listing 4.3: Usage of the process class.

```
Process theProc = new Process();
theProc.StartInfo.FileName =
    @"c:\master\AMPL\amplcml2\ampl.exe";
theProc.StartInfo.Arguments =
    @"c:\master\AMPL\simple.run";
theProc.StartInfo.UseShellExecute = false;
//theProc.StartInfo.RedirectStandardOutput = true;
theProc.StartInfo.CreateNoWindow = true;
//theProc.OutputDataReceived += (sender, args) =>
    Console.WriteLine(args.Data);
theProc.Start();
theProc.WaitForExit();
```

ready and, because of ODBC this would be easy.

4.4.1 External Processes

Communication between the .Net process and AMPL is not as straightforward as it is between .Net and LINGO, since there is no DLL with regards to this purpose. There are some tools available for Oracles' Java platform, but the overhead of introducing a new technology is not justifiable when the more primitive method of using the *Process* class in the C# API works sufficiently well in this case. A process that calls the AMPL-parser with the correct arguments is created in the sourcecode as listed in 4.3. *simple.run* contains the input to the AMPL-solver, such as the paths to the files containing the model and the input data.

The downsides to this approach, compared to using a DLL, are that it provides fewer options for monitoring and controlling the external process, which makes it less safe, since error states are more difficult to detect and address. During the course of this project, there was an occurrence during which the AMPL-parser failed to parse a malformed model- and/or input data file(s) without this causing any disruption to the execution of the main process. The same argument might be raised against the use of DLLs, but since DLLs are designed as single-purpose APIs, they can generally be expected to contain this type of functionality.

5 Results

A system has been successfully implemented that could be used in a production environment with only a few changes. It consists of a persistent data store and a .Net application that handles the data transfer between the calculation engine and the data store, and the processing of in- and output data. A system extension is also in place that enables us test the core system to be tested. A dataset was created that could be used for testing, and, although no problems were detected in the system design, the optimizations took longer than expected, and, longer than would be feasible for a usable application. An adequate performance was the most important goals in relation to the project, and when the performance was not sufficiently acceptable for a usable application, this issue was given precedence over areas that otherwise would have been the focus of further investigation (see section 6 and chapter 6 for an overview of some of these). To establish whether a solution exists, and indicate which direction future work could take, a simplified model was created in the AMPL-language. Optimizations using this model and the CPLEX-solver provides a considerable performance gain compared to the LINGO-model and solver (see appendix D). It is impossible to state how much of this gain should be attributed to the use of a simpler model, and how much to the use of different solver, but it is clear that it is possible to improve the initial poor performance of the system. The AMPL-model and CPLEX-solver often find solutions that are worse than the solutions found by the original LINGO-model and solver with regards to cost, but the differences are minor in the majority of cases, and, in many cases the AMPL-solutions are better with regards to one or more of the objectives, namely total time and total emissions. Since the objective function of the AMPL-model attempts to minimize the sum of the objectives, and the values assumed by the objectives have different numerical ranges, they are scaled in the objective function, to give them equal weight. Applying a greater weight to the cost objective relative to the other two objectives would lead to results that are even more in line with the LINGO-results. The original model uses a multi-step solution procedure, in which a solution is firstly calculated without constraints on total emissions and total time and which is only minimizing the total cost. The subsequent calculations then use the total emissions and total time from the previous solution as the maximum constraints on the total emissions and total time, but, which still minimize only the cost. The same procedure can also be performed with the focus

on either the total emissions or total time, using constraints as described for the remaining two variables. The solution procedure of the simplified model on the other hand, uses a single calculation step, where all three variables - the sum of their weighted totals - are minimized. This means that the simplified model can never find a solution with a more optimal value for the total cost, or a solution with the same value for the total cost and more optimal values for the total emissions and total time, but it does have the ability to discover solutions where the sum of the weighted totals is lower than the same sum, using the same weights, for the solution found by the original model. To illustrate this, imagine in-data containing two transports: **T1**, with cost **C1**, emissions **E1** and duration **D1**, and **T2**, with cost **C2**, emissions **E2** and duration **D2**, where $C1 < C2$, $E1 > E2$ and $D1 > D2$. The first iteration of the solution procedure of the original model will prefer **T1** to **T2** (if possible depending on other factors such as hard time- and volume constraints). The second iteration will look for a transport with lower total emissions and total time, so it will not pick **T2**. The single step of the simplified model may prefer **T2** over **T1**, depending on how large the differences are between **C1** and **C2**, **E1** and **E2**, **D1** and **D2**, and the weights that have been given to the variables. The greater the weight of the total cost is relative to the other weights, the more likely it is that the solution found by the simpler model is the same as the solution found by the original model.

If further investigation should show that all or the greater part of the performance gain should be attributed to the use of a different solver, there might be reason to port the entire model to a different language (like AMPL), and replace the LINGO-solver, or use multiple solvers side by side. If the simpler model is the main cause of the performance gain, it could be ported to LINGO, and different models used side by side. As has been mentioned previously, the simpler model can only be applied to a subset of the data that a production system would have to handle. It is possible that it would be beneficial to incorporate several different models, all of them customized for different characteristics of the input, and either decide which to apply pending analysis of the input data, or apply all of them in parallel and use the output from whichever the model/solver that first completes the optimization.

6 Discussion

One option for the LINGO-solver that could not be taken advantage of for this project, is the enabling of decomposition ([14]). Decomposition of the problem could lead to improved solution times, but enabling the option on the solver, using the setup described here, causes an *access violation exception*. An access violation occurs in unmanaged or unsafe code when the program attempts to read or write to memory that has not been allocated, or to which it does not have access. This usually occurs because a pointer has a bad value. Not all reads or writes through bad pointers lead to access violations, so an access violation usually indicates that several reads or writes have occurred through bad pointers, and that memory might be corrupted. Thus, access violations almost always indicate serious programming errors. In the .NET Framework version 2.0 and later versions, an *AccessViolationException* clearly identifies these serious errors ([31]. Why these exception occur has not been determined in detail in this case, primarily because it is impossible to debug, but it is clear that is involved with the the input data from memory buffers. It would appear that the solver attempted to modify the contents of the buffers themselves, and thereby causing the .NET-virtual machine to detect corrupted memory - that the content of the buffers have been modified by a different process. One way of circumventing this phenomenon could, in theory, be to use direct database access from LINGO [14]. To achieve this, and to not relocate control of process from the application layer, temporary tables could be built from the rows of the underlying, original tables. The lingo solver could then be pointed to these, and gather data from them through the ODBC-driver. Whichever operation causes the access violation exceptions when direct memory access is used to transport data between the application layer and the solver, can no longer occur when data is read directly from the database. Output would still be routed through memory buffers, so that it can be audited, displayed, etc., and so that the state of the datatore can be updated intelligently. If needs be, the output could be written to intermediate tables in the database, in a similar manner to how data is read, and then, from there, parsed and processed in relation to these purposes. Direct database access is also possible from CPLEX ([5]), and it would be worth investigating how this approach would affect performance for both LINGO and CPLEX.

ILOG Concert Technology can provide an application written in .NET #C (or any of a number of other programming languages) access to the CPLEX

solver ([17]). This would replace the use of an external process, and could possibly result in performance enhancements, in addition to the advantages it could provide in terms of monitoring, control and safety.

Further investigation into other solvers that accept AMPL-models could also point to possible performance enhancements, both for the original model - if it was ported to AMPL - and the simpler model.

Finally, further investigation is required regarding concurrency in the system. What is the performance impact of running multiple optimization simultaneously on the same machine? What are the implications for safety and consistency of allowing simultaneous optimizations in a live system, and how would they be addressed in an implementation?

Bibliography

- [1] Leif Olsson, Aron Larsson, Matching of Intermodal Freight Transports Using Optimization in a Decision Support System.
- [2] Maria Kalinina, Aron Larsson, Leif Olsson, Generating and Ordering of Transport Alternatives in Inter-Modal Logistics in the Presence of Cost, Time, and Emission Conflicts.
- [3] Steven S. Skiena, the Algorithm Design Manual, Spring-Verlag London Limited, 2nd Edition, 2008.
- [4] "Linear Programming", Wikipedia: The Free Encyclopedia, Wikimedia Foundation, Inc., http://en.wikipedia.org/wiki/Linear_programming. Retrieved 2013-03-15.
- [5] Rober Fourer, David M. Gay, and Brian W. Kernighan, *the AMPL Book, AMPL: A modelling Language for Mathematical Programming*, Duxbury Press / Brooks/Cole Publishing Company, 2nd edition, 2002.
- [6] Robert E. Bixby, *Progress in Linear Programming*, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, 1993.
- [7] *Computing in Science and Engineering*, Vol. 2, No. 1, 2000.
- [8] Abraham M. Glicksman, *An Introduction to Linear Programming and the Theory of Games*, Courier Dover Publications, 2001.
- [9] Branch-And-Bound Methods: General Formulation and Properties, L. G. Mitten, *Operations Research*, Vol. 18, No. 1, (Jan. - Feb., 1970), pp. 24-34.
- [10] Philip E. Gill, Walter Murray, Michael A. Saunders, J.A. Tomlin, Marharet H. Wright, On Projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarkar's Projective Method, *Mathematical Programming* 36, 183-209, 1986.
- [11] Interior Point Methods 25 Years Later, Jacek Gondzio, *European Journal of Operational Research* 218, pp. 587-601, 2012.

- [12] Notes on Decomposition Methods Stephen Boyd, Lin Xiao, Almir Mutapcic, Jacob Mattingley, Notes for EE364, Stanford University, 2008.
- [13] Kevin Cunningham, Linus Schrage, The Lingo Algebraic Modeling Language, Modeling Languages in Mathematical Optimization, Springer, 2004.
- [14] Lingo User's Guide, Lindo Systems Inc., 2012.
- [15] "AMPL", Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc., <http://en.wikipedia.org/wiki/AMPL>. Retrieved 2013-05-17.
- [16] "CPLEX", Wikipedia: The Free Encyclopedia., Wikimedia Foundation, Inc., <http://en.wikipedia.org/wiki/CPLEX>. Retrieved 2013-06-02.
- [17] IBM ILOG CPLEX Version 12.1 User's Manual for CPLEX, 2009.
- [18] IBM ILOG AMPL Version 12.1 User's Guide, June 2009.
- [19] Google, "The Google Distance Matrix API", <https://developers.google.com/maps/documentation/distancematrix/>. Published 2013-06-05. Retrieved 2013-06-20.
- [20] Green Cargo, "EcoTransIT Miljökalkyl", <http://world.ecotransit.org/greencargo/>. Retrieved 2013-06-20.
- [21] Linear Regression", Wikipedia: The Free Encyclopedia, Wikimedia Foundation, Inc., http://en.wikipedia.org/wiki/Linear_regression. Retrieved 2013-06-02.
- [22] "Solvers that Work with AMPL", AMPL, <http://www.ampl.com/solvers.html>. Retrieved 2013-06-25.
- [23] Pietro Ferrar, Francesco Logozzo, Manuel Fähndrich, Safer Unsafe Code for .NET, Proceedings of the 23rd ACM Conference on Object-Oriented Programming, 2008.
- [24] "Buffer Overflow", Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc., http://en.wikipedia.org/wiki/Buffer_overflow. Retrieved 2013-04-12.
- [25] Sasha Goldshtein, Dima Zurbalev, Ido Flatow, Pro .NET Performance: Optimize Your C# Applications, Apress, 2012.
- [26] Pro C# 5.0 and the .Net 4.5 Framework, Andrew Troelsen, Apress, 2012.

- [27] Don Box, Anders Hejlsberg, *LINQ: .NET Language-Integrated Query*, Microsoft Corporation, 2008.
- [28] Anson Horton, The Evolution Of LINQ And Its Impact On The Design Of C#, *MSDN Magazine*, 2007.
- [29] Charles Petzold, *Programming Windows*, Chapter 21, 5th edition, O'Reilly Media, 2010.
- [30] Microsoft, "What is ODBC?", [http://msdn.microsoft.com/en-us/library/windows/desktop/ms714591\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms714591(v=vs.85).aspx). Retrieved 2013-05-15.
- [31] Microsoft, "AccessViolationException Class", [http://msdn.microsoft.com/en-us/library/vstudio/system.accessviolationexception\(v=vs.90\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/system.accessviolationexception(v=vs.90).aspx). Retrieved 2013-05-12.

7 Appendices

A LINGO Model

Listing 1: The original model with three calculation steps.

```
Model :

Sets :

Nodes ;
Leg_Id ;
ComType ;
LoadCarrier ;
Points /1..3/ :
TC, SC, TT, ME, ETC, ESC, ETT, EME,
TTC, TSC, TTT, TME ;

ComCustSup (ComType, Nodes, Nodes) :
TotTimeTrp, StartTimeTrp, CustDem, SendToCust, Shortage ;

NCS (Nodes, ComCustSup) ;
Pointscust (ComCustSup, points) : shortage1,
    shortage2, shortage3, sendtocust1, sendtocust2,
    sendtocust3 ;

Transport (Nodes, Nodes, Leg_Id, LoadCarrier, ComType) :
MinTrp, MaxTrp, TrpCost, TrpTime, StartTime, Emissions,
    Freight ;

Distrib (Transport, Nodes, Nodes) |
@in (ComCustSup, &5, &6, &7) :
Vol, Trp ;

Transportpoints (Transport, Points) :
Freight1, Freight2, Freight3 ;
Volpoints (Distrib, Points) : Vol1, Vol2, Vol3 ;

Endsets
```

```

Data:

Nodes = @POINTER(1);
Leg_Id = @POINTER(2);
LoadCarrier = @POINTER(3);
ComType = @POINTER(4);
Transport = @POINTER(5);
ComCustSup = @POINTER(6);
StartTime = @POINTER(7);
StartTimeTrp = @POINTER(8);
Emissions = @POINTER(9);
MinTrp = @POINTER(10);
MaxTrp = @POINTER(11);
TrpCost = @POINTER(12);
TrpTime = @POINTER(13);
CustDem = @POINTER(14);
TotTimeTrp = @POINTER(15);

Costlimit=100000;Shortcostlimit=100000;
TimeLimit=100000;EmissionLimit=100000;
ETotalCostlimit=100000; EShortcostlimit=100000;
Etimelimit=100000;EEmissionlimit=100000;
TCostlimit=100000; TShortcostlimit=100000;
TTimeLimit=100000; TEmissionLimit=100000;

Enddata

!-----;

SUBMODEL MIN_COST:

ShortCost=@Sum(Distrib(i,j,p,q,v,l,k):
TrpCost(i,j,p,q,v)*MaxTrp(i,j,p,q,v)
*Shortage(v,l,k));

TotalCost= @sum(Transport(i,j,p,q,v):
TrpCost(i,j,p,q,v)*Freight(i,j,p,q,v));

TotalTime= @sum(Distrib(i,j,p,q,v,l,k)|j#eq#k:
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v))*Trp(i,j,p,q,v,l,k));

```

```

min= ShortCost+ TotalCost;

@For(ComCustSup(v,l,k):SendToCust(v,l,k)=
    CustDem(v,l,k)-Shortage(v,l,k));

@For(NCS(j,v,l,k)|j#eq#k:
@Sum(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k))=
    SendToCust(v,l,k));

@For(NCS(j,v,l,k)|j#eq#k:
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=0);

@For(NCS(j,v,l,k)|j#eq#1:
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=
    SendToCust(v,l,k));

@For(NCS(j,v,l,k)|j#eq#1:
@Sum(Distrib(m,j,p,q,v,l,k):Vol(m,j,p,q,v,l,k))=0);

@For(NCS(j,v,l,k)|(j#ne#1)#and#(j#ne#k):
@Sum(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k))-
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=0);

@For(Transport(i,j,p,q,v):
@Sum(Distrib(i,j,p,q,v,l,k):
Vol(i,j,p,q,v,l,k))<=MaxTrp(i,j,p,q,v));

@For(Distrib(i,j,p,q,v,l,k):
Vol(i,j,p,q,v,l,k)<=
MaxTrp(i,j,p,q,v)*Trp(i,j,p,q,v,l,k));

@For(Distrib(i,j,p,q,v,l,k):
Vol(i,j,p,q,v,l,k)>=
MinTrp(i,j,p,q,v)*Trp(i,j,p,q,v,l,k));

@For(Distrib(i,j,p,q,v,l,k)|(i#eq#1):
(StartTimeTrp(v,l,k)-StartTime(i,j,p,q,v))<=
(StartTimeTrp(v,l,k)+1)*(1-Trp(i,j,p,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k)|j#ne#k:
@For(Distrib(j,m,r,q,v,l,k)|(m#ne#i)#and#(r#ne#p):

```

```

StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v)-StartTime(j,m,r,q,v)<=
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v))*
(2-Trp(i,j,p,q,v,l,k)-Trp(j,m,r,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k)|(j#eq#k):
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v)-TotTimeTrp(v,l,k))<=
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v))*(1-Trp(i,j,p,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k):
@Bin(Trp(i,j,p,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k): Trp(i,j,p,q,v,l,k) <=
Freight(i,j,p,q,v));

@For(Transport(i,j,p,q,v):Freight(i,j,p,q,v) <=
@Sum(Distrib(i,j,p,q,v,l,k):Trp(i,j,p,q,v,l,k)));

@For(Transport(i,j,p,q,v):Freight(i,j,p,q,v) <=1);

MaxEmissions = @Sum(Transport(i,j,p,q,v):
1000*Emissions(i,j,p,q,v)*Freight(i,j,p,q,v));

TotalTime < Timelimit-0.001;

MaxEmissions < Emissionlimit-0.001;

!@For(Nodes(i):@For(ComCustSup(v,l,k):
@Sum(Distrib(i,j,p,q,v,l,k):Trp(i,j,p,q,v,l,k))<=1));

ENDSUBMODEL

!-----;
SUBMODEL MIN_EMISSION:

EMaxEmissions = @Sum(Transport(i,j,p,q,v):
1000*Emissions(i,j,p,q,v)*Freight(i,j,p,q,v));

EShortCost=@Sum(Distrib(i,j,p,q,v,l,k):

```

```

TrpCost(i,j,p,q,v)
*MaxTrp(i,j,p,q,v)*Shortage(v,l,k));

min = EMaxEmissions+EShortCost;

@For(ComCustSup(v,l,k):SendToCust(v,l,k)
= CustDem(v,l,k)-Shortage(v,l,k));

@For(NCS(j,v,l,k)|j#eq#k:
@Sum(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k))=
SendToCust(v,l,k));

@For(NCS(j,v,l,k)|j#eq#k:
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=0);

@For(NCS(j,v,l,k)|j#eq#1:
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=
SendToCust(v,l,k));

@For(NCS(j,v,l,k)|j#eq#1:
@Sum(Distrib(m,j,p,q,v,l,k):Vol(m,j,p,q,v,l,k))=0);

@For(NCS(j,v,l,k)|(j#ne#1)#and#(j#ne#k):
@Sum(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k))-
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=0);

@For(Transport(i,j,p,q,v):
@Sum(Distrib(i,j,p,q,v,l,k):
Vol(i,j,p,q,v,l,k))<=MaxTrp(i,j,p,q,v));

@For(Distrib(i,j,p,q,v,l,k):
Vol(i,j,p,q,v,l,k)
<=MaxTrp(i,j,p,q,v)*Trp(i,j,p,q,v,l,k));

@For(Distrib(i,j,p,q,v,l,k):
Vol(i,j,p,q,v,l,k)
>=MinTrp(i,j,p,q,v)*Trp(i,j,p,q,v,l,k));

@For(Distrib(i,j,p,q,v,l,k)|(i#eq#1):
(StartTimeTrp(v,l,k)-StartTime(i,j,p,q,v))
<=(StartTimeTrp(v,l,k)+1)*(1-Trp(i,j,p,q,v,l,k)));

```

```

@For(Distrib(i,j,p,q,v,l,k)|j#ne#k:
@For(Distrib(j,m,r,q,v,l,k)|(m#ne#i)#and#(r#ne#p):
StartTime(i,j,p,q,v)+TrpTime(i,j,p,q,v)
-StartTime(j,m,r,q,v)<=
(StartTime(i,j,p,q,v)+TrpTime(i,j,p,q,v))
*(2-Trp(i,j,p,q,v,l,k)-Trp(j,m,r,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k)|(j#eq#k):
(StartTime(i,j,p,q,v)+TrpTime(i,j,p,q,v)
-TotTimeTrp(v,l,k))<=
(StartTime(i,j,p,q,v)+TrpTime(i,j,p,q,v))
*(1-Trp(i,j,p,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k):
@Bin(Trp(i,j,p,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k): Trp(i,j,p,q,v,l,k) <=
Freight(i,j,p,q,v));

@For(Transport(i,j,p,q,v):Freight(i,j,p,q,v) <=
@Sum(Distrib(i,j,p,q,v,l,k):Trp(i,j,p,q,v,l,k)));

@For(Transport(i,j,p,q,v):Freight(i,j,p,q,v) <=1);

ETotalCost= @sum(Transport(i,j,p,q,v):
TrpCost(i,j,p,q,v)*Freight(i,j,p,q,v));

ETotalTime= @sum(Distrib(i,j,p,q,v,l,k)|j#eq#k:
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v))*Trp(i,j,p,q,v,l,k));

ETotalTime < ETimelimit-0.001;

ETotalCost < ETotalcostlimit -0.001;

!@For(Nodes(i):@For(ComCustSup(v,l,k):
@Sum(Distrib(i,j,p,q,v,l,k):Trp(i,j,p,q,v,l,k))<=1));

ENDSUBMODEL

SUBMODEL MIN_TIME:

```

```

TTotalTime= @sum(Distrib(i,j,p,q,v,l,k)|j#eq#k:
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v))*Trp(i,j,p,q,v,l,k));

min = TTotalTime+TShortCost;

@For(ComCustSup(v,l,k):SendToCust(v,l,k)=
CustDem(v,l,k)-Shortage(v,l,k));

@For(NCS(j,v,l,k)|j#eq#k:
@Sum(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k))=
SendToCust(v,l,k));

@For(NCS(j,v,l,k)|j#eq#k:
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=0);

@For(NCS(j,v,l,k)|j#eq#1:
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=
SendToCust(v,l,k));

@For(NCS(j,v,l,k)|j#eq#1:
@Sum(Distrib(m,j,p,q,v,l,k):Vol(m,j,p,q,v,l,k))=0);

@For(NCS(j,v,l,k)|(j#ne#1)#and#(j#ne#k):
@Sum(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k))-
@Sum(Distrib(j,i,p,q,v,l,k):Vol(j,i,p,q,v,l,k))=0);

@For(Transport(i,j,p,q,v):
@Sum(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k))
<=MaxTrp(i,j,p,q,v));

@For(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k)
<=MaxTrp(i,j,p,q,v)*Trp(i,j,p,q,v,l,k));
@For(Distrib(i,j,p,q,v,l,k):Vol(i,j,p,q,v,l,k)
>=MinTrp(i,j,p,q,v)*Trp(i,j,p,q,v,l,k));

@For(Distrib(i,j,p,q,v,l,k)|(i#eq#1):
(StartTimeTrp(v,l,k)-StartTime(i,j,p,q,v))<=
(StartTimeTrp(v,l,k)+1)*(1-Trp(i,j,p,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k)|j#ne#k:
@For(Distrib(j,m,r,q,v,l,k)|(m#ne#i)#and#(r#ne#p):

```

```

StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v)-StartTime(j,m,r,q,v)<=
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v))*(2-Trp(i,j,p,q,v,l,k)
-Trp(j,m,r,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k)|(j#eq#k):
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v)-TotTimeTrp(v,l,k))<=
(StartTime(i,j,p,q,v)
+TrpTime(i,j,p,q,v))*(1-Trp(i,j,p,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k):
@Bin(Trp(i,j,p,q,v,l,k)));

@For(Distrib(i,j,p,q,v,l,k): Trp(i,j,p,q,v,l,k) <=
Freight(i,j,p,q,v));

@For(Transport(i,j,p,q,v):Freight(i,j,p,q,v) <=
@Sum(Distrib(i,j,p,q,v,l,k):Trp(i,j,p,q,v,l,k)));

@For(Transport(i,j,p,q,v):Freight(i,j,p,q,v) <=1);

TMaxEmissions = @Sum(Transport(i,j,p,q,v):
1000*Emissions(i,j,p,q,v)*Freight(i,j,p,q,v));

TShortCost=@Sum(Distrib(i,j,p,q,v,l,k):
TrpCost(i,j,p,q,v)*MaxTrp(i,j,p,q,v)
*Shortage(v,l,k));

TTotalCost=@sum(Transport(i,j,p,q,v):
TrpCost(i,j,p,q,v)*Freight(i,j,p,q,v));

TMaxEmissions < TEmissionLimit-0.001;

TTotalCost < TCostLimit -0.001;

!@For(Nodes(i):@For(ComCustSup(v,l,k):
@Sum(Distrib(i,j,p,q,v,l,k):Trp(i,j,p,q,v,l,k))<=1));

ENDSUBMODEL

```

```

CALC:
@FOR (POINTS( b):

@SOLVE(MIN_COST);
Costlimit=TotalCost; Shortcostlimit=Shortcost;
    TimeLimit=TotalTime; EmissionLimit=MaxEmissions;
TC(b)=TotalCost; SC(b)=Shortcost; TT(b)=TotalTime;
    ME(b)=MaxEmissions;
@FOR(Transport(i,j,p,q,v):
    Freight1(i,j,p,q,v,b)=Freight(i,j,p,q,v));
@FOR(Pointscust(v,l,k,b):
    Shortage1(v,l,k,b)=Shortage(v,l,k));
@FOR(Pointscust(v,l,k,b):
    Sendtocust1(v,l,k,b)=Sendtocust(v,l,k));
@FOR(Volpoints(i,j,p,q,v,l,k,b):
    Vol1(i,j,p,q,v,l,k,b)=Vol(i,j,p,q,v,l,k));

@SOLVE(MIN_EMISSION);
ETotalCostlimit=ETotalCost;
    EShortcostlimit=EShortcost;
    ETimeLimit=ETotalTime;
    EEmissionLimit=EMaxEmissions;
ETC(b)=ETotalCost; ESC(b)=EShortcost;
    ETT(b)=ETotalTime; EME(b)=EMaxEmissions;
@FOR(Transport(i,j,p,q,v):
    Freight2(i,j,p,q,v,b)=Freight(i,j,p,q,v));
@FOR(Pointscust(v,l,k,b):
    Shortage2(v,l,k,b)=Shortage(v,l,k));
@FOR(Pointscust(v,l,k,b):
    Sendtocust2(v,l,k,b)=Sendtocust(v,l,k));
@FOR(Volpoints(i,j,p,q,v,l,k,b):
    Vol2(i,j,p,q,v,l,k,b)=Vol(i,j,p,q,v,l,k));

@SOLVE(MIN_TIME);
TCostlimit=TTotalCost; TShortcostlimit=TShortcost;
    TTimeLimit=TTotalTime;
    TEmissionLimit=TMaxEmissions;
TTC(b)=TTotalCost; TSC(b)=TShortCost;
    TTT(b)=TTotalTime; TME(b)=TMaxEmissions;
@FOR(Transport(i,j,p,q,v):
    Freight3(i,j,p,q,v,b)=Freight(i,j,p,q,v));

```

```

@FOR(Pointscust(v,l,k,b):
    Shortage3(v,l,k,b)=Shortage(v,l,k));
@FOR(Pointscust(v,l,k,b):
    Sendtocust3(v,l,k,b)=Sendtocust(v,l,k));
@FOR(Volpoints(i,j,p,q,v,l,k,b):
    Vol3(i,j,p,q,v,l,k,b)=Vol(i,j,p,q,v,l,k));
);

ENDCALC

Data:

TC = @POINTER(16);
SC = @POINTER(17);
TT = @POINTER(18);
ME = @POINTER(19);
ETC = @POINTER(20);
ESC = @POINTER(21);
ETT = @POINTER(22);
EME = @POINTER(23);
TTC,TSC,TTT,TME,
Freight1,Freight2,Freight3,
Shortage1, Shortage2, Shortage3,
Sendtocust1,Sendtocust2, Sendtocust3;

Enddata

End

```

B Lingo Performance

This following table lists the average, *AvgDur*, and median, *MedDur*, durations of calculations in milliseconds for optimizations performed with the original model and the LINGO-solver with the REDUCE-option enabled, for various numbers of requests, *Reqs*, and transports, *Transps*.

Table 1: LINGO optimization results.

Reqs	Transps	AvgDur	MedDur
1	50	1,400	1,334
5	50	6,416	6,225
7	50	14,014	14,485
1	75	2,003	2,013
5	75	13,645	13,643
10	75	44,762	46,714
1	100	3,288	3,175
5	100	18,779	18,994
10	100	62,962	66,324
1	200	6,714	6,974
5	200	45,969	48,478
10	200	133,693	131,486
15	200	291,326	282,604
20	200	495,896	487,550
1	500	22,194	23,502
5	500	156,686	152,912
10	500	511,139	490,194
15	500	1,203,709	1,213,829
20	500	1,938,119	1,946,175
1	1,000	69,377	68,765
5	1,000	521,591	519,967
10	1,000	1,746,824	1,757,016

The following table lists the average optimization times in milliseconds for optimizing using different numbers of transports and transport requests using the LINGO-solver with all options left to their default values. For column *A* all transports were passed to the solver. For column *B*, transports on routes, where nothing is requested, are filtered out in a preprocessing step and not passed to the solver. For column *C*, an additional preprocessing step is applied (see section 3.2) to the input data, further reducing (in some cases) the number of transports passed to the solver. The column *Best Time* highlights the best average runtime that was achieved.

Table 2: LINGO optimization results with pre-optimization.

Context	A	B	C	Best Time
50 transports	-	-	-	-
1 request	2309	608	421	421
5 requests	11825	3245	3728	3245
10 requests	20280	2153	2574	2153
15 requests	20343	4025	4960	4025
20 requests	20155	2481	3978	2481
100 transports	-	-	-	-
1 request	4336	265	936	265
5 requests	25241	5523	4025	4025
10 requests	78781	19890	10436	10436
15 requests	78234	14180	10452	10452
20 requests	79779	17316	10967	10967
500 transports	-	-	-	-
1 request	32729	468	671	468
5 requests	214283	9719	5335	5335
10 requests	682442	25787	36785	25787
15 requests	1405491	136438	77330	77330
20 requests	2367955	321752	140667	140667
1000 transports	-	-	-	-
1 request	91386	1014	1716	1014
5 requests	691568	2559	9313	2559
10 requests	2094080	124536	207014	124536
15 requests	15066888	766308	459048	459048
20 requests	17743600	550387	1328895	550387
2000 transports	-	-	-	-
1 request	293314	2527	718	718
5 requests	13155829	7707	40981	7707
10 requests	5102247	722924	288197	288197

Table 2: LINGO optimization results with pre-optimization.

Context	A	B	C	Best Time
15 requests	12468692	1118418	1401013	1118418
20 requests	31528	1215263	1661504	31528

C AMPL Model and Data

Listing 2: The simpler model with a single calculation step.

```
set Nodes;
set Leg_Id;
set ComType;
set LoadCarrier;

param c_factor;
param e_factor;
param t_factor;

set ComCustSup within {ComType,Nodes,Nodes};
param TotTimeTrp {ComCustSup};
param StartTimeTrp {ComCustSup};
param CustDem {ComCustSup};

set Transport within {Nodes,Nodes, Leg_Id,
    LoadCarrier, ComType};
param MinTrp {Transport};
param MaxTrp {Transport};
param TrpCost {Transport};
param TrpTime {Transport};
param StartTime {Transport};
param Emissions {Transport};

set Distrib :=
setof{(i,j,k,l,m) in Transport,
(m_m, i_i, j_j) in ComCustSup:
m = m_m and i = i_i and j = j_j
and StartTimeTrp[m_m,i_i,j_j] <= StartTime[i,j,k,l,m]
and TotTimeTrp[m_m,i_i,j_j] >= StartTime[i,j,k,l,m]
+ TrpTime[i,j,k,l,m]
}
(m_m, i_i, j_j,i,j,k,l,m);

param total_requested = sum{(i,j,k) in
    ComCustSup}(CustDem[i,j,k]);

var DistribDecide{Distrib} binary;
var Vol{Distrib};
```

```

var Emi{Transport};
var Tim{Transport};
var Cos{Transport};
var TransportDecide{Transport} binary;

var total_delivered = sum{(m_m,i_i,j_j,i,j,k,l,m) in
  Distrib}(Vol[m_m,i_i,j_j,i,j,k,l,m]);

var total_cost = sum {(i,j,k,l,m) in Transport}
  (Cos[i,j,k,l,m]);
var total_emissions = sum {(i,j,k,l,m) in Transport}
  (Emi[i,j,k,l,m]);
var total_time = sum {(i,j,k,l,m) in Transport}
  (Tim[i,j,k,l,m]);

minimize optimum: total_cost * c_factor +
  total_emissions * e_factor + total_time *
  t_factor;

subject to

zero_volume {(m_m,i_i,j_j,i,j,k,l,m) in Distrib}:
  DistribDecide[m_m,i_i,j_j,i,j,k,l,m] = 0 ==>
  Vol[m_m,i_i,j_j,i,j,k,l,m] = 0;

transport_use{(m_m,i_i,j_j,i,j,k,l,m) in Distrib}:
  DistribDecide[m_m,i_i,j_j,i,j,k,l,m] = 1 ==>
  TransportDecide[i,j,k,l,m] = 1;

transp_cost{(i,j,k,l,m) in
  Transport}:TransportDecide[i,j,k,l,m] = 1 ==>
  Cos[i,j,k,l,m] = TrpCost[i,j,k,l,m];
no_transp_cost{(i,j,k,l,m) in
  Transport}:TransportDecide[i,j,k,l,m] = 0 ==>
  Cos[i,j,k,l,m] = 0;
transp_emission{(i,j,k,l,m) in
  Transport}:TransportDecide[i,j,k,l,m] = 1 ==>
  Emi[i,j,k,l,m] = Emissions[i,j,k,l,m];
no_transp_emission{(i,j,k,l,m) in
  Transport}:TransportDecide[i,j,k,l,m] = 0 ==>
  Emi[i,j,k,l,m] = 0;

```

```

transp_time{(i,j,k,l,m) in
  Transport}:TransportDecide[i,j,k,l,m] = 1 ==>
  Tim[i,j,k,l,m] = TrpTime[i,j,k,l,m];
no_transp_time{(i,j,k,l,m) in
  Transport}:TransportDecide[i,j,k,l,m] = 0 ==>
  Tim[i,j,k,l,m] = 0;
max_transport{(i,j,k,l,m) in Transport}:sum
  {(m_m,i_i,j_j,i,j,k,l,m) in Distrib}
  Vol[m_m,i_i,j_j,i,j,k,l,m] <= MaxTrp[i,j,k,l,m] *
  TransportDecide[i,j,k,l,m];
min_transport{(i,j,k,l,m) in Transport}:sum
  {(m_m,i_i,j_j,i,j,k,l,m) in Distrib}
  Vol[m_m,i_i,j_j,i,j,k,l,m] >= MinTrp[i,j,k,l,m] *
  TransportDecide[i,j,k,l,m];

max_transport_request{(m,i,j) in ComCustSup}:sum
  {(m_m,i_i,j_j,i,j,k,l,m) in Distrib}
  Vol[m_m,i_i,j_j,i,j,k,l,m] <= CustDem[m,i,j];

total_requested_volume:
  total_delivered = total_requested;

```

Listing 3: Example of sample in-data formatting for the AMPL/CPLEX-solver.

```
set ComType := C;

set Leg_Id:= 1 2 3 4 5 6 7 8 9 10;

set LoadCarrier := M;

set Nodes := ADAK ANGE ARE ARJEPLOG ARNAESVALL
  ARVIDSJAUR Alnoo As BERGVIK BISPGARDEN BJAESTA
  BLAVIKSJOON BOLIDEN BOLLNAES BOLLSTABRUK
  BONAESSUND BORGAFJAELL BREDBYN BUREA BURTRAESK
  BYGDEA BYSKE Boden Braecke DELSBO DIKANAES
  DOCKSTA DOROTEA ERSMARK Edsbyn FLARKEN FRAENSTA
  FROOSON FUNAESDALEN FOOLLINGE GAMMELSTAD
  GAEDDEDE GAELLO GAELLIVARE HALLEN HAMMERDAL
  HARADS HEDE HEMAVAN HUDIKSVALL Haparanda Holmsund
  Hortlax HAERNOSAND HOORNEFORS ILSBO JOKKMOKK JORN
  JAERVSO Jaerpen KALiX KILAFORS KLOVSJOO
  KOSKULLSKULLE KRAMFORS KVISSLEBY Kiruna KAELARNE
  Kaege LANGVIKSMON LJUNGDALEN LJUSDAL LJUSNE
  LOFSDALEN LOGDEA LOVANGER LOVIKKA LULEA Lit
  LILLPITE Lycksele MALA MALMBERGET MARSFJAELL
  MATIMAR NJURUNDA NORDMALING Norrfjaerden Norsjoo
  NAELDEN OFFERDAL OJEBYN ORNSKOLDSVIK OSSOLA
  OVIKEN PAJALA PILGRIMSTAO Pitea RAMSELE
  ROBERTSFORS ROKNAES ROSSON RUNDVIK ROOBAECK
  SAXNAES SIDENSJOO SKELLEFTEA SKELLEFTEHAMN
  SLAGNAES SODERHAMN SOLLEFTEA SORAKER SORBERGE
  SORSELE STORLIEN STORUMAN STROMSUND STOODE
  SUNDSBRUK SUNDSVALL SVAPPAVAARA SVEG SVENSTAVIK
  Stugun TALLASEN TAVELSJO TENNDALEN TIMRA
  TRANGSVIKEN TAEFTEA TAENNDALEN TAERENDOO TAERNABY
  ULLANGER UMEA UNDERSAKER URSVIKEN VALSJOOSYN
  VEMDALEN VILHELMINA VINDELN VITA VAENNAS AELVSBY
  AElandsbro OOSTERSUND OOVERKALIX OOVERTORNEA;

set Transport :=
AElandsbro, ADAK, 1, M, C
AElandsbro, Alnoo, 2, M, C
AElandsbro, Alnoo, 3, M, C
```

```

AElandsbro , Alnoo , 4 , M , C
AElandsbro , Alnoo , 5 , M , C
AElandsbro , Alnoo , 6 , M , C
AElandsbro , Alnoo , 7 , M , C
AElandsbro , Alnoo , 8 , M , C
AElandsbro , Alnoo , 9 , M , C
AElandsbro , Alnoo , 10 , M , C;

param MinTrp :=
AElandsbro , ADAK , 1 , M , C , 10
AElandsbro , Alnoo , 2 , M , C , 34
AElandsbro , Alnoo , 3 , M , C , 20
AElandsbro , Alnoo , 4 , M , C , 34
AElandsbro , Alnoo , 5 , M , C , 26
AElandsbro , Alnoo , 6 , M , C , 34
AElandsbro , Alnoo , 7 , M , C , 26
AElandsbro , Alnoo , 8 , M , C , 26
AElandsbro , Alnoo , 9 , M , C , 14
AElandsbro , Alnoo , 10 , M , C , 26;

param MaxTrp :=
AElandsbro , ADAK , 1 , M , C , 28
AElandsbro , Alnoo , 2 , M , C , 100
AElandsbro , Alnoo , 3 , M , C , 60
AElandsbro , Alnoo , 4 , M , C , 100
AElandsbro , Alnoo , 5 , M , C , 80
AElandsbro , Alnoo , 6 , M , C , 100
AElandsbro , Alnoo , 7 , M , C , 80
AElandsbro , Alnoo , 8 , M , C , 80
AElandsbro , Alnoo , 9 , M , C , 40
AElandsbro , Alnoo , 10 , M , C , 80;

param TrpCost :=
AElandsbro , ADAK , 1 , M , C , 1260
AElandsbro , Alnoo , 2 , M , C , 858
AElandsbro , Alnoo , 3 , M , C , 709
AElandsbro , Alnoo , 4 , M , C , 923
AElandsbro , Alnoo , 5 , M , C , 1399
AElandsbro , Alnoo , 6 , M , C , 1297
AElandsbro , Alnoo , 7 , M , C , 1108
AElandsbro , Alnoo , 8 , M , C , 1501
AElandsbro , Alnoo , 9 , M , C , 1254

```

```

AElandsbro , Alnoo , 10 , M , C , 1489;

param TrpTime :=
AElandsbro , ADAK , 1 , M , C , 86
AElandsbro , Alnoo , 2 , M , C , 69
AElandsbro , Alnoo , 3 , M , C , 70
AElandsbro , Alnoo , 4 , M , C , 69
AElandsbro , Alnoo , 5 , M , C , 70
AElandsbro , Alnoo , 6 , M , C , 70
AElandsbro , Alnoo , 7 , M , C , 70
AElandsbro , Alnoo , 8 , M , C , 69
AElandsbro , Alnoo , 9 , M , C , 68
AElandsbro , Alnoo , 10 , M , C , 69;

param StartTime :=
AElandsbro , ADAK , 1 , M , C , 89
AElandsbro , Alnoo , 2 , M , C , 79
AElandsbro , Alnoo , 3 , M , C , 104
AElandsbro , Alnoo , 4 , M , C , 13
AElandsbro , Alnoo , 5 , M , C , 38
AElandsbro , Alnoo , 6 , M , C , 74
AElandsbro , Alnoo , 7 , M , C , 119
AElandsbro , Alnoo , 8 , M , C , 73
AElandsbro , Alnoo , 9 , M , C , 133
AElandsbro , Alnoo , 10 , M , C , 3;

param Emissions :=
AElandsbro , ADAK , 1 , M , C , 0.0649
AElandsbro , Alnoo , 2 , M , C , 0.0201
AElandsbro , Alnoo , 3 , M , C , 0.0205
AElandsbro , Alnoo , 4 , M , C , 0.0199
AElandsbro , Alnoo , 5 , M , C , 0.0228
AElandsbro , Alnoo , 6 , M , C , 0.0217
AElandsbro , Alnoo , 7 , M , C , 0.0198
AElandsbro , Alnoo , 8 , M , C , 0.0206
AElandsbro , Alnoo , 9 , M , C , 0.021
AElandsbro , Alnoo , 10 , M , C , 0.0197;

set ComCustSup :=
C , AElandsbro , Alnoo
C , AElandsbro , ADAK;

```

```
param TotTimeTrp :=
C, AElandsbro, Alnoo, 147
C, AElandsbro, ADAK, 233;

param StartTimeTrp :=
C, AElandsbro, Alnoo, 0
C, AElandsbro, ADAK, 86;

param CustDem :=
C, AElandsbro, Alnoo, 53
C, AElandsbro, ADAK, 19;

param c_factor :=1;
param e_factor :=47003.984063745;
param t_factor :=16.6169014084507;
```

D LINGO and AMPL performance

This table lists the results of optimizations with the original model and the LINGO-solver, with the REDUCE-option enabled, compared to the results of optimizations with the simpler model and the AMPL/CPLEX-solver as duration in milliseconds for LINGO and AMPL/CPLEX, $DurL$ and $DurA$, cost for LINGO and AMPL/CPLEX, $CostL$ and $CostA$, time for LINGO and AMPL/CPLEX, $TimeL$ and $TimeA$, emissions for LINGO and AMPL/CPLEX, $EmissL$ and $EmissA$, for the same number of requests, $Reqs$, transports, $Transps$.

Table 3: Optimization times for LINGO and AMPL.

Reqs	Transps	DurL	DurA	CostL	CostA	TimeL	TimeA	EmissL	EmissA
1	50	1841	125	1112	1178	94	94	816	800
1	50	2012	156	574	574	89	89	778	778
1	50	1872	124	1019	1019	88	88	793	793
1	50	1810	172	760	763	70	68	197	203
1	50	1607	125	982	1301	93	93	812	815
1	50	1841	140	971	971	92	92	793	793
1	50	1529	109	1154	1166	70	69	233	214
1	50	1622	141	600	600	69	69	199	199
1	50	1560	125	1125	1125	69	69	200	200
1	50	1263	109	498	498	74	74	421	421
5	50	7769	187	4026	4026	395	395	2455	2455
5	50	8440	156	4287	4499	411	410	2871	2886
5	50	9157	156	4551	5384	398	395	2546	2505
5	50	9454	172	3998	4241	415	415	2849	2847
5	50	9267	188	4953	5804	399	398	2521	2517
5	50	9719	171	5461	5461	406	406	2798	2798
5	50	9594	140	5276	5323	411	409	2865	2842
5	50	8127	171	4812	4812	400	400	2518	2518
5	50	8689	125	3856	3905	408	409	2729	2745
5	50	9626	125	4178	4178	377	377	1927	1927
7	50	15631	125	7905	8137	554	554	3374	3398
7	50	16224	172	5675	5754	558	555	3414	3369
7	50	16723	187	6850	6850	554	554	3357	3357
7	50	16583	234	7220	7220	555	555	3414	3414
7	50	16770	141	6390	6390	553	553	3412	3412
7	50	16037	234	6594	7354	555	556	3397	3412
7	50	16147	140	6087	6087	552	552	3356	3356
7	50	16785	140	6359	6604	552	553	3385	3390
7	50	16754	156	7185	7185	556	556	3355	3355
7	50	16583	141	7602	7964	555	556	3389	3403
7	50	17036	172	7006	7272	554	556	3398	3364
7	50	16536	156	6245	6245	556	556	3359	3359
7	50	17035	156	7898	7898	552	552	3348	3348
7	50	16614	156	7163	7181	558	554	3433	3421
7	50	16427	171	6775	6974	558	555	3429	3401
7	50	16520	124	6601	6668	555	553	3431	3408
7	50	17004	140	7188	7218	554	552	3376	3356
7	50	16068	156	7851	8701	550	551	3301	3301
7	50	16411	187	6129	6205	551	552	3358	3348

Table 3: Optimization times for LINGO and AMPL.

Reqs	Transps	DurL	DurA	CostL	CostA	TimeL	TimeA	EmissL	EmissA
7	50	16645	203	6516	6580	554	556	3477	3476
7	50	16677	172	6160	6449	553	554	3393	3421
7	50	17004	234	6837	6837	553	553	3357	3357
7	50	16927	140	7266	7505	557	556	3373	3372
7	50	16552	156	6458	6458	556	556	3469	3469
7	50	16864	124	6612	6612	554	554	3399	3399
7	50	16193	140	6229	6453	555	555	3415	3421
7	50	16458	140	7339	7339	554	554	3405	3405
7	50	16286	172	6377	6377	553	553	3419	3419
7	50	15147	156	7110	7110	553	553	3401	3401
7	50	15881	140	7083	7315	555	556	3471	3500
1	55	1607	109	543	543	89	89	792	792
1	55	1653	172	841	1020	92	93	818	800
1	55	1654	93	1158	1158	70	70	214	214
1	55	999	141	729	729	88	88	796	796
1	55	1076	94	513	513	89	89	791	791
1	55	1014	78	849	849	94	94	809	809
1	55	983	94	916	916	88	88	760	760
1	55	1045	94	823	823	90	90	797	797
1	55	1404	124	1186	1186	84	84	609	609
1	55	1498	124	1302	1302	88	88	756	756
5	55	9688	141	4272	4878	398	398	2500	2496
5	55	9298	172	3822	3822	392	392	2354	2354
5	55	9157	141	5068	5232	409	408	2740	2726
5	55	10250	188	4839	4851	394	394	2473	2488
5	55	8440	141	4764	4841	423	422	3032	3031
5	55	9189	250	3593	4462	402	400	2706	2639
5	55	10218	187	4494	4494	389	389	2277	2277
5	55	10857	156	4948	4963	405	402	2719	2638
5	55	10062	187	4812	4852	406	404	2747	2732
5	55	10498	156	3982	4568	391	392	2319	2332
9	55	27893	172	8365	8365	706	706	4294	4294
9	55	27955	187	8092	8590	708	704	4305	4240
9	55	29719	140	9452	9534	706	704	4300	4233
9	55	30155	172	7461	7524	712	711	4259	4286
9	55	29484	203	8073	8073	705	705	4231	4231
9	55	29048	172	7186	7186	706	706	4200	4200
9	55	28845	187	9179	9179	713	713	4267	4267
9	55	29750	156	7497	7772	706	705	4237	4232

Table 3: Optimization times for LINGO and AMPL.

Reqs	Transps	DurL	DurA	CostL	CostA	TimeL	TimeA	EmissL	EmissA
9	55	28049	156	8600	8744	710	711	4319	4284
9	55	29001	188	10002	10061	795	708	5076	4278
9	55	28814	141	7963	8688	712	710	4253	4270
9	55	29406	203	9022	9022	707	707	4280	4280
9	55	28767	156	9395	9395	709	709	4266	4266
9	55	27659	156	8980	9355	710	710	4272	4288
9	55	28923	156	8681	9279	706	705	4225	4227
9	55	28642	202	8581	8877	710	709	4277	4286
9	55	28626	234	8317	8441	705	706	4319	4310
9	55	29188	156	8020	8120	711	711	4343	4336
9	55	29048	187	9412	9412	711	711	4274	4274
9	55	28580	141	8099	8559	709	706	4273	4260
9	55	29640	141	8046	8343	707	707	4273	4251
9	55	27971	141	9336	9336	710	710	4294	4294
9	55	29234	141	9668	9668	707	707	4292	4292
9	55	28283	266	9596	9861	704	705	4188	4188
9	55	28938	156	10265	10766	707	708	4256	4266
9	55	28111	156	8221	8221	703	703	4298	4298
9	55	28361	141	9041	9041	706	706	4247	4247
9	55	27409	140	7207	7696	702	703	4126	4144
9	55	28860	156	9419	9419	711	711	4321	4321
1	60	1981	109	647	647	69	69	230	230
1	60	1576	109	605	605	86	86	675	675
1	60	1388	140	622	622	86	86	750	750
1	60	1217	94	680	680	69	69	215	215
1	60	1155	141	552	552	89	89	746	746
1	60	1092	78	699	699	88	88	723	723
1	60	1045	110	337	337	70	70	237	237
1	60	967	110	1021	1350	86	87	762	781
1	60	1841	172	489	489	74	74	400	400
1	60	1654	156	800	800	86	86	726	726
5	60	12527	187	4782	4782	426	426	3261	3261
5	60	11825	187	4538	4614	407	407	2781	2767
5	60	11388	141	4417	4750	408	408	2871	2843
5	60	10109	218	3966	3966	391	391	2320	2320
5	60	11201	171	3412	4096	390	389	2436	2416
5	60	11794	140	3587	4175	415	413	3005	2982
5	60	11591	156	3910	3910	384	384	2124	2124
5	60	11497	125	4449	4449	394	394	2471	2471

Table 3: Optimization times for LINGO and AMPL.

Reqs	Transps	DurL	DurA	CostL	CostA	TimeL	TimeA	EmissL	EmissA
5	60	11528	156	4262	4953	405	405	2783	2781
5	60	11856	140	3793	3963	410	410	2863	2857
10	60	38251	141	9624	9635	793	793	5009	4980
10	60	38673	172	8983	8983	793	793	5061	5061
10	60	39265	172	9035	9035	792	792	5053	5053
10	60	39344	140	9178	9178	796	796	5104	5104
10	60	38516	156	10504	10887	794	792	5011	5002
10	60	38704	140	8364	8676	794	794	5107	5117
10	60	33026	140	10844	11179	801	800	4985	4991
10	60	31325	125	8616	8622	792	793	5021	5014
10	60	37674	156	10309	10309	797	797	4925	4925
10	60	40342	141	9863	9912	792	792	5069	5027
10	60	36972	234	8397	8397	796	796	5097	5097
10	60	39452	296	9484	10245	793	795	5014	5017
10	60	40092	187	9242	9695	797	795	5011	4996
10	60	38252	124	9205	9806	792	792	5010	5020
10	60	38782	156	9298	9298	789	789	5048	5048
10	60	38532	171	8914	9843	794	794	5033	5043
10	60	39172	172	9655	9655	794	794	5049	5049
10	60	36426	125	10553	10553	798	798	5167	5167
10	60	38673	140	9968	9968	790	790	5090	5090
10	60	39141	203	8272	8359	800	800	5070	5074
10	60	39296	171	10693	12002	793	795	5056	5077
10	60	38064	203	10586	10586	792	792	5088	5088
10	60	38906	141	8745	8745	796	796	5089	5089
10	60	39687	187	9461	9461	790	790	4927	4927
10	60	38657	156	8719	9130	793	792	4995	4968
10	60	39218	156	10185	10678	795	792	5057	4997
10	60	38766	171	10720	10725	796	795	5088	5057
10	60	39390	374	8620	8969	795	794	5061	5094
1	65	2106	141	1057	1057	70	70	273	273
1	65	1981	172	670	670	72	72	294	294
1	65	1950	109	880	880	85	85	681	681
1	65	1903	125	547	547	69	69	238	238
1	65	1466	156	1276	1276	70	70	213	213
1	65	1326	109	1061	1061	70	70	241	241
1	65	1280	94	1555	1555	93	93	806	806
1	65	1747	140	541	541	87	87	731	731
1	65	1779	109	653	653	68	68	248	248

Table 3: Optimization times for LINGO and AMPL.

Reqs	Transps	DurL	DurA	CostL	CostA	TimeL	TimeA	EmissL	EmissA
1	65	2059	109	1234	1234	70	70	244	244
5	65	12183	141	4673	4673	402	402	2718	2718
5	65	11762	141	4523	4685	425	423	3161	3115
5	65	12433	172	4593	4675	410	410	2856	2835
5	65	12261	109	4586	4586	388	388	2269	2269
5	65	11606	156	4335	4505	408	408	2772	2764
5	65	12574	156	4426	5650	408	409	2817	2818
5	65	12434	187	5594	5594	415	415	2911	2911
5	65	11372	156	4465	5499	407	411	2808	2879
5	65	12214	156	4134	4134	394	394	2617	2617
5	65	12745	140	5260	5556	408	409	2883	2859
10	65	42074	125	8725	8816	798	797	5073	5079
10	65	43166	140	7907	8034	790	790	4984	4963
10	65	41013	140	8758	8904	791	792	5072	5090
10	65	41668	125	10360	10823	790	791	5016	4989
10	65	43119	202	8615	8615	793	793	5064	5064
10	65	42947	156	8711	8711	794	794	4993	4993
10	65	40560	140	9637	10387	798	799	5070	5115
10	65	43993	171	9686	9751	798	794	5062	5013
10	65	42916	156	10529	10911	793	794	5019	4999
10	65	41122	156	10464	10490	796	794	5078	5038
10	65	43119	406	9712	10395	795	797	5025	5007
10	65	41808	328	9114	9366	794	793	5122	5095
10	65	41637	499	9574	10576	796	794	5070	5060
10	65	42604	343	10571	10961	798	797	5041	5043
10	65	42947	265	8431	8459	792	791	5124	5112
10	65	42682	171	8786	9856	794	794	5077	5089
10	65	42791	140	9524	9550	792	790	4952	4951
10	65	39983	218	8730	9532	797	799	5131	5128
10	65	40638	140	9153	9498	793	794	5053	5029
10	65	36910	187	9079	9079	798	798	5029	5029
10	65	42900	140	9917	9917	792	792	4956	4956
10	65	41450	343	10434	10519	880	794	5805	5035
10	65	41637	140	8486	8990	787	790	4986	5013
10	65	43728	234	8921	9082	795	794	4950	4992
10	65	40310	187	8891	9681	791	788	5016	4872
10	65	41668	171	10561	10561	793	793	4999	4999
1	70	2106	125	1046	1046	85	85	660	660
1	70	2106	172	607	607	89	89	797	797

Table 3: Optimization times for LINGO and AMPL.

Reqs	Transps	DurL	DurA	CostL	CostA	TimeL	TimeA	EmissL	EmissA
1	70	1981	125	753	753	89	89	770	770
1	70	1732	109	1016	1016	69	69	209	209
1	70	2169	109	764	932	89	85	778	714
1	70	1716	140	677	677	76	76	396	396
1	70	2168	109	922	922	72	72	281	281
1	70	2403	141	810	810	89	89	791	791
1	70	2293	109	1398	1398	72	72	267	267
1	70	2075	156	978	978	86	86	745	745
5	70	13587	172	3996	3996	401	401	2670	2670
5	70	13323	140	3944	3944	390	390	2305	2305
5	70	13447	156	5029	5029	425	425	3254	3254
5	70	13058	203	4375	4723	393	393	2357	2374
5	70	13729	156	3531	3872	394	395	2301	2334
5	70	13774	156	4573	4573	409	409	2875	2875
5	70	13494	359	4612	5341	474	391	3119	2434
5	70	13182	125	5176	5176	403	403	2718	2718
5	70	13885	234	5216	5216	410	410	3008	3008
5	70	12808	171	4026	4026	384	384	2300	2300
10	70	42401	203	10235	10235	789	789	5018	5018
10	70	45865	234	9500	10157	795	797	5035	5027
10	70	41902	187	10536	10830	795	795	4932	4979
10	70	41636	156	8824	9560	797	797	5096	5023
10	70	43555	141	9072	9203	794	796	5068	5076
10	70	44647	140	9523	9734	795	794	4947	4942
10	70	44460	172	8911	8980	799	799	4981	5025
10	70	46707	187	9866	10180	796	796	4986	4971
10	70	44695	234	9353	9445	794	789	5065	5016
10	70	46629	156	9397	9453	794	793	5045	5004
10	70	46239	140	8493	8778	794	795	4950	4998
10	70	45069	234	9335	9463	797	797	5066	5075
10	70	45646	172	9213	9345	797	797	5151	5178
10	70	43789	156	9618	10054	793	794	4989	5001
10	70	46254	187	8998	10402	794	795	5048	5003
10	70	45505	188	10116	10133	798	797	5014	5022
10	70	44804	172	9009	9009	790	790	4945	4945
10	70	45209	188	9794	10487	794	793	5047	4977
10	70	45381	140	9116	9245	798	799	5013	5040
10	70	45895	219	10203	10203	796	796	4972	4972
10	70	46941	156	10022	10966	795	793	5040	5069

Table 3: Optimization times for LINGO and AMPL.

Reqs	Transps	DurL	DurA	CostL	CostA	TimeL	TimeA	EmissL	EmissA
10	70	47019	156	9969	10017	793	794	5039	5020
1	75	2199	110	1172	1172	93	93	790	790
1	75	2402	124	658	658	89	89	780	780
1	75	2387	171	585	585	69	69	212	212
1	75	2168	281	798	798	88	88	792	792
1	75	2168	172	834	834	88	88	791	791
1	75	1701	172	733	733	70	70	212	212
1	75	2184	124	982	982	93	93	833	833
1	75	2294	203	898	898	70	70	223	223
1	75	2387	141	1016	1329	87	87	795	781
1	75	2091	172	965	965	86	86	782	782
5	75	14727	156	4928	4928	406	406	2769	2769
5	75	14758	125	4776	5676	396	396	2514	2542
5	75	15413	234	4550	4893	499	411	3627	2816
5	75	14866	156	3528	3711	408	410	2940	2945
5	75	15070	187	3390	3390	410	410	2838	2838
5	75	14648	156	5038	5038	406	406	2751	2751
5	75	15273	156	4499	4613	404	404	2827	2823
5	75	14960	234	4166	4166	413	413	2835	2835
5	75	13884	156	4707	4707	389	389	2246	2246
10	75	49795	172	8352	8352	794	794	4952	4952
10	75	49358	172	8522	8547	800	795	5144	5103
10	75	49718	249	10176	10182	797	794	5015	4952
10	75	49218	234	9490	9490	796	796	4993	4993
10	75	50155	172	9263	9409	793	793	5044	5049
10	75	50934	265	7854	9429	794	797	5058	5060
10	75	48999	515	10373	10669	792	789	5034	5002
10	75	48111	156	7948	8505	798	797	5049	5006
10	75	48860	141	9162	9625	796	793	5028	5018
10	75	49608	219	8374	9113	795	797	5056	5096
10	75	49062	171	10100	10440	795	795	5063	5046
10	75	49421	156	10745	10745	792	792	4995	4995
10	75	50903	218	9135	9958	797	796	5077	5087
10	75	50045	156	9130	9130	792	792	4998	4998
10	75	50606	218	8119	8511	796	797	5062	5082
10	75	50358	312	9501	9557	795	790	4997	4956
10	75	49873	187	10435	10439	797	797	5136	5155
10	75	49327	156	9561	9561	798	798	4977	4977
10	75	51387	265	9203	9913	792	794	4908	4984

Table 3: Optimization times for LINGO and AMPL.

Reqs	Transps	DurL	DurA	CostL	CostA	TimeL	TimeA	EmissL	EmissA
10	75	49546	156	10015	10020	796	795	5082	5082
10	75	48625	140	10117	11510	796	793	5088	5118
10	75	41121	156	9238	9238	791	791	5056	5056
10	75	49858	203	9362	9362	795	795	5078	5078
10	75	47721	234	9377	9562	796	796	5021	5012