Vrije Universiteit, Amsterdam
Faculteit der Exacte Wetenschappen

BMI Paper

An Auction Based Approach to Multicommodity Minimal Cost Flow Problems

by

Selmar Smit

Supervisor:
Sandjai Bhulai

Amsterdam, 2007

An old man on the point of death summoned his sons around him to give them some parting advice. He ordered them to bring in a bundle of sticks, and said to his eldest son: "Break it"

The son strained and strained, but with all his efforts was unable to break the bundle. The other sons also tried, but none succeeded.

"Untie the bundle," said the father, "and each of you take a stick." When they had done so, he told them: "Now, break," and each stick was easily broken.

Aesop, 620 - 560 BC

Contents

Preface

One of the last courses in the study Business Mathematics and Informatics is the BMI paper. This paper should be a research oriented paper that reflects the knowledge that BMI students gathered during their study period at the VU University of Amsterdam. The topic of the paper should cover the three main directions of BMI.

This paper is not focussing on a societal problem specifically, but introduces an algorithm that is able to solve a type of problem to which a whole range of problems is related. Furthermore, with this paper we hope to open the eyes of the many researchers in the field of Econometrics; No one can break a bundle of sticks, but untie them and everybody can break them.

Finally, I would like to thank dr. Sandjai Bhulai, not only for his support during this course, but also for our discussions about Operational Research and Computational Intelligence during my study period.

Abstracts

English

In this paper, we introduce a new algorithm for solving Multicommodity Minimal Cost Flow problems; the Auction Algorithm (AA). This algorithm is special: It is not efficient on its own, but it uses the strength of the many. The Auction Algorithm can be run fully distributed.

The main idea of the algorithm is that commodities bid on the arcs they want to use. A second algorithm determines which commodity is the ‘winner’ and which commodities have to re-route. After several rounds of bidding and re-bidding, the algorithm terminates with a feasible or optimal solution, depending on the settings.

In this paper we compare the results of the Auction Algorithm with the Price-Directive Decomposition (PDD) and initialization as proposed by Bobonneau [1]. From the results we conclude that the Auction Algorithm manage to solve the generated problems within a shorter time period than the implemented PDD algorithm, although it is very sensitive to the ‘hardness’ of the problem. Furthermore, we found a range of problems which cannot be solved by Price-Directive Decomposition, because the algorithm ran out of memory finding a base. The Auction Algorithm, however, eventually found the optimal solution.

Although the framework as proposed in this paper is generic, we focus on the Undirected Disjoint Arc variant.

Dutch

In deze paper introduceren we een nieuw algoritme voor het oplossen van Multicommodity Minimal Cost Flow problemen: het Auction Algoritme (AA). Dit algoritme is speciaal: Alleen is het algoritme niet krachtig, maar het gebruikt de kracht van het collectief. Het Auction Algoritme kan volledig gedistribueerd uitgevoerd worden.

In dit algoritme bieden de verschillende commodities op de arcs die ze willen gebruiken. Een tweede algoritme bepaald dan welke commodity de 'winnaar' is en welke commodities hun route moeten aanpassen. Na meerdere ronden van biedingen en herbiedingen is het algoritme afgelopen en is een toegestane of optimale oplossing gevonden hetgeen afhankelijk is van de instellingen.

In deze paper vergelijken we de resultaten van het Auction Algoritme met de resultaten van het Price-Directive Decomposition (PDD) algoritme dat beschreven is door Bobonneau [1]. Uit de resultaten kunnen we concluderen dat het Auction Algoritme de gegenereerde problemen eerder heeft opgelost dan het PDD algoritme, hoewel het AA zeer gevoelig is voor de 'moeilijkheid' van het probleem. Tevens hebben we een groep problemen geïdentificeerd waarbij de initialisatie van de basis door PDD faalt wegens een geheugentekort. Het Auction Algoritme vindt daarentegen uiteindelijk het optimum.

Hoewel in deze paper de focus ligt op de Undirected Disjoint Arc variant van het probleem, is de omgeving ook te gebruiken in aanverwante problemen.

Chapter 1

Introduction

In the last couple of years, researchers are focusing on solving well-known network problems like the Travelling Salesman, Vehicle Routing and Maximal -and Minimal Cost Flow. The big interest in these kinds of problems is not strange, because many of the network problems that are found within companies and research agencies are related to them. Problems like vehicle routing, resource planning, and/or scheduling can be represented as huge network problems consisting of thousands of nodes and arcs.

These problems are like the bundle of sticks from Aesop's fable, it is impossible to break it with brute-force. These problems are huge, therefore, linear programming is much to slow and memory expensive. To deal with this problem, researchers started to develop new algorithms. These algorithms are often very efficient and can solve problem instances very fast. Although they work fine for some of the commercial problems, they fail one the biggest ones due to the limited amount of computer-memory available.

In this paper we are presenting a method to solve these huge problems in limited time. We have chosen to develop an algorithm for one of these typical problems, namely the Multicommodity Minimal Cost Flow Problem. The algorithm proposed in this paper is not very efficient on its own, but can use the strength of many. The proposed algorithm is implemented within a software model that is using a massive amount of processor-time, simultaneously. Instead of one processor that takes care of the whole algorithm, hundreds of processors can work on solving their piece of the problem at the same time. Although there are more 'distributed' algorithms, these algorithms are not fully distributed and can only divide the workload instead of the problem [2] or can only handle binary problems [3]. These algorithms do not solve the problems related to size, they only speed up the algorithm and therefore, their performance is directly related to the performance of the overall algorithm, it is like the sons are trying to break the bundle of sticks simultaneously. Although this is more effective than trying it alone, it would no break the bundle.

In the next chapter an explanation of the Multicommodity Minimal Cost Flow Problem, the possible applications of the problem, and the currently used algorithms will be presented. Furthermore, some variations of the problem are described including the Multicommodity Minimal Cost Flow Problem with disjoint arcs. This is the variant that will be used in the paper to describe the algorithm and present its performance.

The chapter ‘Auction Algorithm’ gives an overview of the algorithm itself and proofs the optimality of it.

The chapters four, five, and six will present an implementation of the algorithms and will present the results of the Action Algorithm in comparison with standard linear programming.

At the end of this paper conclusions will be presented and several possibilities of future research are stated.

Chapter 2

Multicommodity Minimal Cost Flow Problem

2.1 Description

The Multicommodity Minimal Cost Flow Problem is a type of problem that is related to capacitated networks. This capacitated network consists of various nodes and arcs, which can be seen as villages and roads connecting them. Each of these arcs has got costs and capacity. On this network, items(commodities) are carried from one node to another. The problem is to find a path (or various paths) which can be used to transfer these items without violating the maximum capacity on each arc. This path not only has to be capacitated enough to transfer the items, but also has to be as cheap as possible. This problem is called the Minimal Cost Flow Problem. If this problem is extended with multiple types of items that are sharing the same arcs and capacities then this is called the Multicommodity Minimal Cost Flow Problem. An example of such an problem is displayed below:

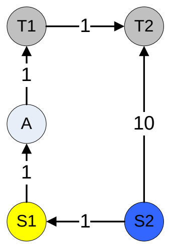


Figure 2.1: An Example with corresponding costs

In this example there are two commodities, each with their own source (s_1 and s_2) and their own sinks (t_1 and t_2). They are both of the same size (1) and both are sending 1 item from their source to their sink. All of the arcs have a capacity of 1 and the costs of each arc are 1 or 10. The obvious solution to this example is that commodity 1 goes from s_1 via A to t_1, and s_2 goes directly to t_2. Although commodity 2 prefers going via s_1, A, t_1 to t_2, this violates the capacity constraints on these arcs.

The Multicommodity Minimal Cost Flow Problems are, due to their specific network structure, easy to write as a linear programming model^{T1} :
The following model assumes equally-sized commodities with equal costs in a directed

network.
Consider a network with a finite set V of nodes $1, \cdots, N$, and a finite set E of pairs of nodes $e_m = (i, j)$, called arcs. Arc e_m has capacity b_m and costs c_m with $m = 1, \cdots, M$. Let there be K commodities and let x_m^k be the flow of commodity k on arc m. Commodity k has got a source called s_k and a sink called t_k each with a supply and a demand of S_k. Let $A_n \subset E$ denote the arcs that originate at node n and let $B_n \subset E$ denote the nodes that terminate at node n.
The problem may be stated (in node-arc formulation) as:

$$min \sum_{k,m} c_m \cdot x_m^k \tag{2.1}$$

$$\sum_{e_m \in A_n} x_m^k - \sum_{e_m \epsilon B_n} x_m^k = \begin{cases} S_k, & \texttt{if } n = s_k \\ -S_k, & \texttt{if } n = t_k \\ 0, & \texttt{otherwise} \end{cases} \quad \texttt{all } n, k \tag{2.2}$$

$$\sum_k x_m^k \leq b_m \texttt{, all } m \tag{2.3}$$

$$x_m^k \geq 0 \tag{2.4}$$

We see directly from this model that the number of variables is equal to the number of arcs multiplied with the number of commodities. Therefore the size of the problem is getting huge when there are a lot of communities and arcs. Furthermore, this problem is proven to be NP-complete$^{\text{T2}}$ [4] and, therefore, probably not solvable in polynomial time. This means that if the problems gets bigger, the time needed to solve it gets even bigger.

2.2 Applications

There are many commercial problems that are closely related to the Multicommodity Minimal Cost Flow. Bellmore, Bennington and Lubore [5] used it for solving scheduling and rerouting problems. Bradley [6], Carter and Stowers [7] and Charnes and Cooper [8] used the problem to model urban traffic control systems. The problem is also used to assign students to schools to achieve a desired ethnic decomposition [9]. Wrathall used it to model a railroad traffic scheduling system [10]. Geoffrion and Graves used the problem to solve a large multicommodity warehouse location problem [11]. In this problem there are multiple plants that are producing various commodities. Furthermore, there are some distribution centers and several customer zones with a certain demand for a commodity. The number of items that is transported from a plant to a distribution center is bounded and costs a certain amount of money. This model can be changed into a Multicommodity Minimal Cost Flow problem by defining plants as sources, the customer zones as sinks and define possible flows from and too the distribution centers as arcs in the network. The distribution centers itself are the nodes between sources and sinks. This results in a Multiple sinks, multiple sources variant of the Multicommodity Minimal Cost Flow Problem, which will be discussed in Section 2.3.
In more recent applications the problem is used to solve container shipment problems by Krile[12], where Yahaya and Sunda [13],[14] used it to model internet traffic routing.

The Multicommodity Minimal Cost Flow Problem can therefore be seen as widely used, very generic and still very popular when defining network and transshipment problems.

2.3 Problem Variants

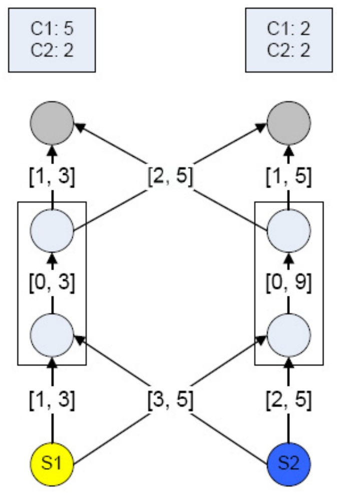
The Multicommodity Minimal Cost Flow Problem is widely used and therefore many researchers studied the possible applications. During these studies many variants to the problem arose. Some of them simplified the problem, where others added extra constraints that made the problem even more complex. In the next section the most famous variants are discussed.

2.3.1 Integer

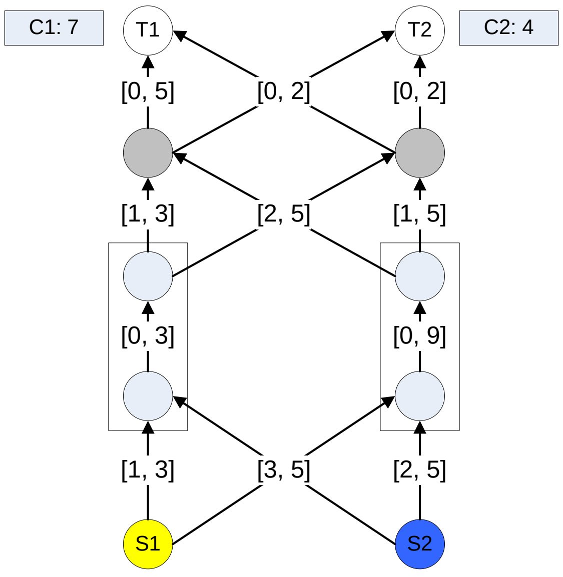
The Integer Multicommodity Minimal Cost Flow Problem is the most famous variant to the original problem. In this variant a new constraint is added: The flow through one or more of the arcs has to be integer. While in the One-Commodity Minimal Cost Flow Problem the integer constraint is not needed if supply, demand, and capacities are integer, these restrictions are not enough to guarantee that a Multicommodity Minimal Cost Flow Problem respects the integer constraints. Therefore, these constraints really increase the difficulty of the problem. Only if the problem is also a Unsplittable Flow Problem (2.3.4) or Disjoint Arcs (2.3.3) Problem, then these additional constraints can be left out.

2.3.2 Multiple sinks and/or sources

Many of the problems that can be found consist of multiple sources and/or multiple sinks. Transshipment problems often deal with the fact that commodities have to be shipped from a certain production plant to multiple shops, distribution centers or consumer zones. Although this problem looks very different from the ordinary Multicommodity Minimal Cost Flow Problem, it is very easy to transform it into a single source, single sink Multicommodity Minimal Cost Flow Problem. In the following problem there are two commodities that are produced in two different plants. They are both equal in size and transportation costs. There are two shops (T_1, T_2) that have to be supplied and each of the shops is having a certain demand for a certain commodity. They can be supplied via two different distribution centers, each with its own capacity to store the commodities. Distribution centers are modeled by two nodes, one representing the incoming commodities, and one representing the outgoing. The line between them are the storage costs per unit (zero in this case) and the maximum storage size. The other lines represent possible flows, with transport costs (first number) and capacity (second number). The demand for a certain commodity is stated above the shop.



This is clearly a Two-commodity Single Source, Multiple Sinks Minimal Cost Flow Problem. To change this into a Two-commodity Single Source Single Sink Minimal Cost Flow Problem, two more nodes are added, one for each commodity. These two nodes will function as a super-sink, collecting the flows from all sinks. Each super-sink is connected to a sink with an arc of costs zero and with a capacity equal to the demand of the sink. The demand of the super-sink is equal to the sum of the demands of the connected sinks.



In this way the multiple-sink problem is transformed into a single-sink problem, which can be solved by many different algorithms.

2.3.3 Disjoint Arcs

The disjoint arc problem rises when there are different types of commodities that cannot be transported over the same arc. With this problem, some (or all) of the arcs have an additional constraint that enforces that only one commodity can use it. This problem is often combined with the Unsplittable Problem (2.3.4) with demands, supplies, and capacities of 1. In this paper a generic Disjoint Arcs Multicommodity Minimal Cost Flow Problem is used to describe the Auction Algorithm and to determine the performance. Commodities can be split into several paths, but only one type of commodity can use an arc. The Disjoint Arc problem can be solved with multiple types of algorithms and can be very efficiently implemented with the Auction Algorithm.

2.3.4 Unsplittable and K-Splittable Flows

The Unsplittable Flow Problem is a variant that adds a constraint to the problem which is simplifying the problem. This constraint enforces that the total flow of the commodity can go through only one path of arcs. It forbids splitting of the flow and is therefore not possible to combine with the transformation of the multiple source/sink problem. To deal with this problem there is the K-Splittable problem which forbids the commodity to take more than k paths. This type of problem is introduced by Baier, Köhler, and Skutella [15] and is also used in transportation problems with a bounded number of vehicles [16]. It is very hard to write this problem as a linear programming model and therefore most algorithms are very specialized and can only calculate good approximations, rather than exact solutions. The Auction Algorithm can very efficiently solve the unsplittable problem because it is very similar to the Shortest-Path Problem$^{\mathrm{T3}}$. The K-splittable problem in contrary is still very hard and depends on the available One-Commodity K-Splittable Minimal Cost Flow Problem.

2.3.5 Non-linear costs

The last famous variant that will be discussed in this paper is the Non-linear Cost Problem. In most problems the costs of travelling through the network only depends on the costs of using an arc multiplied with the usage of the arc. There are some problems in which these costs are different. Some of these problems have additional fixed costs for using a certain arc or fixed costs for using a certain node. Other problems have costs that are not linear but piecewise linear or depend on a special cost-function. Non-linear cost problems are often very hard and in some cases impossible to write as a linear programming model. Therefore, there are some specialized algorithms to solve this kind of problems. The Auction Algorithm is able to solve such problems exactly as long as the costs depend on the sum of the individual costs.

Chapter 3

Current Algorithms

Multicommodity Minimal Cost Flow Problems are linear programs, and can therefore be solved with the simplex method. Commercial problems are however often too huge to solve within reasonable time or even impossible to solve because of a lack of memory to store all needed variables. Therefore multiple algorithms were created to deal with these problems using specialized techniques. Most of these algorithms are extensions or variations of the standard Price-Directive Decomposition.

3.1 Price-Directive Decomposition

The Price-Directive Decomposition approach uses a master program and several subprograms in order to find an optimal solution. The master program consists of a basic feasible solution to which, on every iteration, a new path is added. This new path is chosen in such a way that it contributes to a better solution. The algorithm stops if there is no new profitable path possible. At that moment the optimal solution can be found by solving the master program. Subprograms are used to suggest which paths have to be added to the main program in order to obtain the optimal solution. To determine which paths are possible candidates, the subprogram uses the dual variables from the main program. Dual variables can be calculated by solving the dual version of the main program with revised simplex. These values represent the costs that can reduced by adding a new path to the main program that uses certain arcs.

This way of adding variables from subprograms into the master program is called column-generation and was first proposed by Ford and Fulkerson [17] to solve Multicommodity network flow problems. The method of finding a solution by solving multiple subproblems based on the dual variables of a main problem is called the Dantzig-Wolfe decomposition [18]

The algorithms operates in the following way:

1. *Initialization*: Find a feasible basis for the main program and calculate the corresponding dual variables.

2. *Pricing*: Calculate for each commodity if there is a path to which the sum of the dual variables on the used arcs is higher than the costs of the path. Calculating this can be done by solving an one-commodity minimal cost flow, which can be done very efficiently ([19], [20], [21], a.o.).

   If such a path exists, than this path is a candidate to enter the base of the main program. If no such path exists for all of the commodities optimality is reached.

3. *Solving the Master*: Select the best path of the candidates and add it to the base. Solve the Master program and update the dual variables. Return to step 2.

Because each of the subproblems can be solved for each commodity independently, it can be solved distributed too. Although this algorithm is very efficient when solving on a single processor, it is not build to be run distributed, because of its dependence of solving the main program. Therefore it is not guaranteed that it will be as efficient as on a single machine. The biggest problem with this approach is to find an initial solution. There are various ways to find such a solution, but most of them require solving a Linear Programming Model with $3 \cdot m \cdot k$ continuous and $m \cdot k$ integer variables .

Chapter 4

Auction Algorithm

4.1 Description

The base idea of the Auction Algorithm is new [*]. Other algorithms first try to find a feasible solution and then iterate till it reaches optimality. The Auction Algorithm, in contrary, first creates a base that is optimal for each individual commodity and then iterates till the general solution is feasible.

1. *Initialization*: First, every commodity calculates which paths to take in order to get an individual optimal path. This can be done by solving a one-commodity minimal cost flow problem. After that the main program counts which of the arcs are used by two or more commodities. On each of these arcs an auction is started.

2. *Auctions*:

   a) *Start*: Each of the commodities that is present on the arc has to place a bid on the usage of the specific arc, therefore, each of the commodities calculates the costs of avoiding the arc. This can be done by solving an one-commodity minimal cost flow problem and then subtract the current optimal value.

   b) *Finish*: The main program determines which of the bids is the highest. This commodity can use the specific arc, the others change the maximal capacity on that arc to zero.

   c) *Reopen Auctions*: Some of the commodities that 'lost' the auction now have to change their routes and can therefore reconsider bids they did on previous auctions. Because their costs and capacities are changed they can reopen a previous auction by changing the capacity on that previous arc back to the original value.

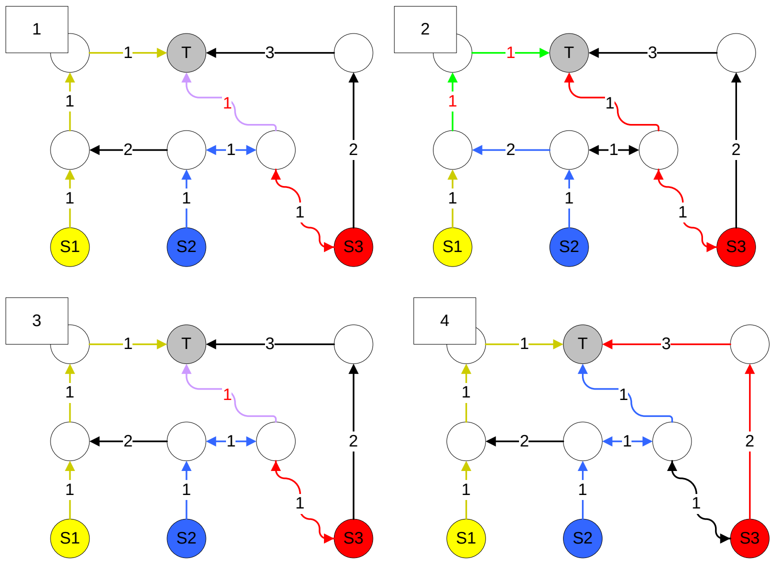
3. *Updating*: After all capacities are updated, each of the commodities solves their (adapted) one-commodity minimal cost flow. If there is any arc with more than one commodity present, return to step 2. Otherwise stop.

[*]During our research we got attended to the ideas of Maria Gini [22] about task allocation on robots using auctions. Both algorithms work similarly, but on a different problem-space. A description of her algorithm is out of the scope of this thesis.

4.1.1 Example

The following example illustrates the algorithm:

There are three different commodities (yellow, blue, and red), they all have the same sink and size. With the use of three auctions finally a feasible and optimal solution is reached:



In the first graph each commodity calculated the shortest route. This results in one auction (marked red). Commodity 2 bids 2 because its current path has costs 3 and losing the auction results in a path with costs 5. Commodity 3 bids 3 (5 minus 2) and wins the auction.

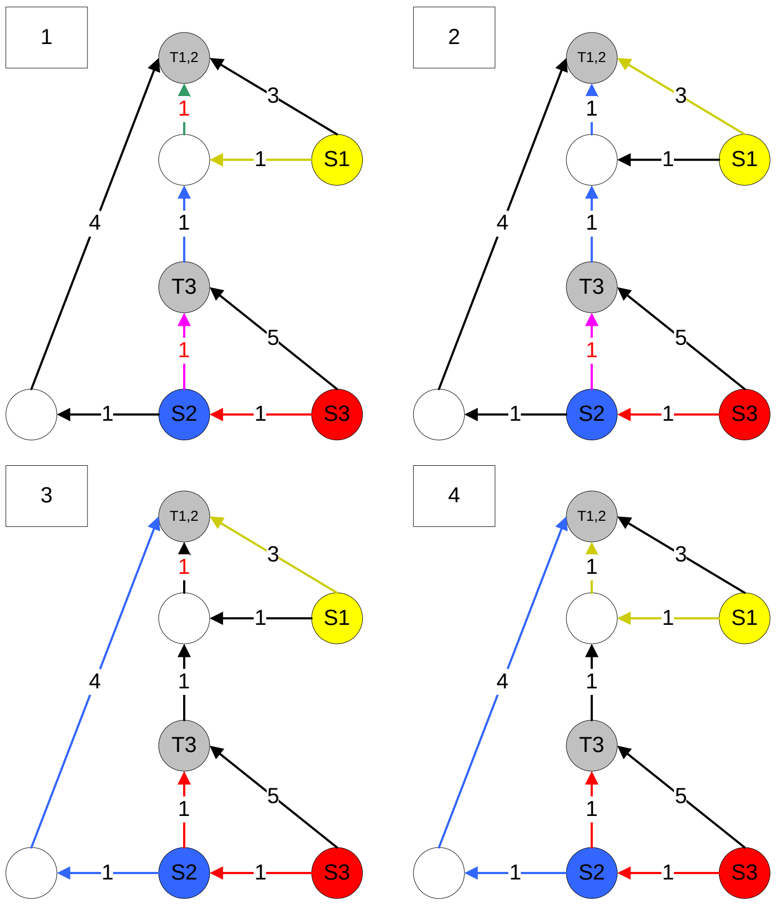
In diagram 2, Commodity 2 calculates a new path and a new auction is started. In this case commodity 2 bids 3 again because its costs will increase from 5 to 8. Commodity 1 needs the arc in auction to reach the sink (because of the directed path) and bids infinity. Commodity 2 loses the auction and has to reroute.
Commodity 2 lost the first auction, because commodity 3 bid 3. The value of its path at that point was 3. The value at that moment plus the maximum bid at the auction is equal to 6, and therefore less than the current costs of 8. So the auction is reopened (Diagram 3). Commodity 3 still bids 3, but because the capacities for commodity 2 were changed, the bid of commodity 2 is now 5 (8-3).

In the last diagram the final solution is reached, commodity 3 rerouted its path and there are no running auctions anymore.

Although in the previous example optimality is reached, the rules as defined in the previous section are not enough to guarantee an optimal solution. These rules however, are enough to enforce a feasible solution. For some commercial problems it is enough to find a feasible solution, others need a good or optimal one. To enforce an optimal solution we can use other algorithms, or extend the Auction Algorithm with one more rule:

If the last occupying commodity changes its usage of a certain arc in the current solution, and there were auctions on that arc in the past, then all commodities that bid on that auctions have to change their maximum capacity on that arc to the original value and revise their bids on the other arcs.

The following case is an example in which the rule above is needed:



Now, two conflicts arise at the same time. One between commodity 1 and 2, and one with commodity 2 and three. First the auction for commodity 1 and 2 is started. Commodity 1 bids 1 (3-2) and commodity 2 bids 2 (5-3). Therefore, commodity 2 wins the auction and commodity 1 calculates a new path. Now the second auction takes place, again commodity 2 bids 2, but now commodity 3 bids 3. Commodity 2 reroutes its path, but now the arc of the first conflict is free again. Although there are no conflicts, this solution is not optimal. Therefore the final rule triggers, because there is an arc on which an auction took place which is unoccupied, every commodity resets its capacity on that arc. Now commodity 1 is able to reroute again and now the optimal solution is found.

4.2 Pseudocode

The algorithm as described in the previous sections can be written with the following pseudocode:

1.

```
FOREACH c:commodity {
   c.value = c.solve_local_problem()
}
```

2.

```
DO
   FOREACH a:arc {
      finished = true
      IF (a.commodities > 1) {
         finished = false
         a.previous_usage = -1
         FOREACH c2:commodity ON a {
            c.setCapacity(a, 0)
            c.auctionValue = c.solve_local_problem()
            c.bid = c.auctionValue - c.value
         }
```

3.

```
         minBid = getMinimalBid(a)
         FOREACH c2:commodity ON a {
            IF (c.bid < minBid) {
               c.storeMinimalAuctionValue(a, c.value + minBid)
               FOREACH a2:arc {
                  IF (c.getMinimalAuctionValue(a2) < c.auctionValue) {
                     c.setCapacity(a, a.originalCapacity)
                  }
               }
            } ELSE {
               c.setCapacity(a, a.originalCapacity)
            }
         }
      }
   }
```

4. ______

```
  FOREACH a:arc {
     IF (a.commodities==0 && a.previous_usage > 0 ||
         a.commodities==1 && a.previous_usage > a.usage) {
            FOREACH c:commodity {
              IF (c.getCapacity(a)==0) {
                finished = false;
                c.setCapacity(a, a.originalCapacity)
 c.reviseBids();
              }
            }
     }
  }
```

5. ______

```
LOOP UNTIL (finished)
```

To explain the pseudocode, it is split into five parts that represent the different stages in the algorithm. First every commodity is initializing their optimal path. Section two describes the creation and bidding. Section three is about updating the capacities of winning and losing commodities and describes how auctions can be reopened. Part four contains the extra rule such that free arcs can be reused. By removing the `c.reviseBids();` line, we create an algorithm for finding a feasible solution instead of the optimal one. The last part of the code is only one rule that tells the algorithm to loop until no conflicts are there and no wrongly used arcs are left.

4.3 Proofs

With a logical explanation the fact that this algorithm always returns an optimal solution, if there are any feasible solutions, can be proven Therefore, the proof is split into two different proofs that together imply optimality. First, a proof will be given that shows that if there are any feasible solutions, the algorithm will return a feasible solution. In the second section, a proof will be given that if the algorithm returns a solution, then this solution is optimal.

4.3.1 Termination

At first it is necessary to see that the algorithm never terminates if the solution is not feasible. The basic rule of the algorithm is to keep iterating till there are no conflicts anymore. If there are no conflicts, then the solution is feasible, so the only thing that have to be proven is that the algorithm eventually terminates.

The easiest way to show that the algorithm terminates is to look at the number of conflicts. If the maximal number of conflicts is bounded, then the algorithm will eventually terminate. It is easy to see that the total number of conflicts is at max the maximal number of conflicts per arc multiplied with the number of arcs. Furthermore, the maximal number of conflicts per arc is less than or equal to the maximum number of auctions that a certain commodity can start per arc, multiplied with the number of commodities. So how many times can a commodity start an auction? The only way that a commodity can start a new auction is: if it lost an auction on a certain arc and as a result migrates to an arc on which already another commodity is present. Furthermore, a commodity can only reopen auctions if it increases it bidding strength, which can only be the result of a lost auction. So the maximum number of auctions that one commodity can start on a certain arc is equal to the number of arcs. Therefore, the maximum number of auctions is equal to $M^2 \cdot K$ with M as the number of arcs, and K the number of commodities.

Unless the number of arcs or the number of commodities is infinite, this number is bounded. Furthermore, we can observe that, even if we have an infinite number of processors, the maximum amount of time needed for solving an instance of the Multicommidity Minimal Cost Flow is still polynomial. Although this algorithm can solve these kind of NP-hard problems relatively fast, it does not proof $P = NP$.

4.3.2 Optimal Solution

To prove that the returned solution is optimal a logical model is created that uses the divide and conquer technique. It splits the proof into several subproofs which will be refined further till the small proofs can be given. If all subproblems can be proved then the original statement is proved too. In this case, the assumption is made that the solution is not optimal which will be rejected because all of the possible causes are rejected too.

If the solution is not optimal, then this has to be caused because one or more commodities are not route optimal. This can be caused by two errors.
The first error could be: There is an arc that is free and has to be used by the commodity. If we assume that the One-commodity Minimal Cost Flow solver(MCF-Solver) works correctly then this cannot be caused by consciously ignoring that arc. So the only other possibility is that it is not used because the capacity is changed into zero. But it was assumed that the arc was free, so the additional rule changed the capacity back to the original value. So this part of the tree is rejected.

The second error could be that the commodity has to use an arc that is wrongly occupied by another commodity. If we again assume a working MCF-Solver, then the only cause can be that the commodity tried to occupy the arc, but lost an earlier auction. The loss of an auction can only be caused by too little bidding-strength or a too high bidding-strength of the opposing commodity.

If the other strength is too high, this can only be caused by two different things. First, this can be caused by the fact that it ignores a free arc, which is already rejected or it can be caused by the loss of an even earlier auction. We can reject this part of the tree because the loss is caused by an earlier loss which is caused by an earlier loss, etc., and if there is only a finite number of auctions, then there is an auction (the first) without an earlier auction.

So, the only other possibility is that the strength is too small. This can only be caused because there is another path that is incorrectly free. This is the case if another commodity did not occupy the path that is optimal. Because, if it did, it results in an auction which leads to an increase in strength. So, this other commodity is occupying an arc that is not optimal. This can only happen if it had an auction with a third commodity that ended incorrectly. If we assume that the number of commodities is limited then we can conclude that there is a commodity that cannot win or lose an auction incorrectly from a commodity that is not used before. Therefore this part of the tree can be rejected too.

All possible causes of a non-optimal routed commodity are rejected, so all commodities have to be routed optimally. Therefore the overall solution is optimal too.

4.4 Advantages and Disadvantages

The Auction Algorithm has many advantages compared with the Price-Directive Decomposition, but there are some disadvantages too Advantages can be found in the field of problem size and performance. The disadvantages are related to problems where there is knowledge about the problem or exact solutions are not needed.

One of the biggest improvements of the Auction Algorithm is the initialization phase. This phase is the most inefficient part of the pricing-directive algorithm. Finding a feasible base solution is very time consuming since there is no current algorithm available that can calculate it fully distributed or efficiently. The most widely used method is to add additional variables to the problem which makes a new problem with even more variables and solve it with a 'distributed' branch-and-bound method. This method divides the work over multiple processors, but this does not decrease the size of the problem. It is very likely that some of the huge problems are too big to be solved with these kind of methods. The lack of computer memory makes it impossible to find a base solution. Furthermore, even if the memory is sufficient, it takes a lot of time to solve it. The Auction Algorithm is not dependent on any initialization and problems are only as big as the number of arcs. The number of commodities only determines the performance and the needed number of processors. They do not influence the problem size that can be handled. In Chapter 7 the performance of both methods is measured on very huge problems, furthermore, some settings are given on which the Price-Directive fails to initialize due to memory exhaustion.

The second major advantage is its generic approach and use of knowledge about the problem. If the problem is a Unsplittable problem, then this knowledge can be used. Instead of Minimal Cost Flow problems, Shortest-Path algorithms can be used to solve the subproblems. The algorithm itself can be extended in many ways it consist of independent parts that can easily be substituted if needed.

When using the AA for finding the optimal solution, the major disadvantage of the algorithm is its way of finding the optimal solution. If the algorithm is stopped prematurely, the current solution is not of any use. The solution at that moment is in most cases not feasible. Furthermore, there is no indication of the needed running time. The number of auctions per iteration is no indicator of the time that is needed to find a feasible solution. If the number of auctions at time step t is equal to 1, then the number of auctions in the next timestep can easily be 1000 again. Algorithms like the Price-Directive approach always have feasible solutions and are increasing in quality during the run. After a while the algorithm can be stopped if the solution is good enough.

The second major disadvantage is the knowledge that can be put into the algorithm. If a certain non-optimal but feasible solution is known, then this cannot be used in the Auction Algorithm as base.

These disadvantages can be overcome by combining the strength of AA; finding a base solution and PDD; getting to the optimum. Investigation this combination is left for future research. In this paper, we will only address the full Auction algorithm.

It is always necessary to think about which algorithm to take when solving large problems. Although there are some disadvantages, the Auction Algorithm is very useful when the problems are huge and consist of many commodities. Sometimes this approach is even the only way to find a solution, because it can handle much bigger problems when no solution is known a priori.

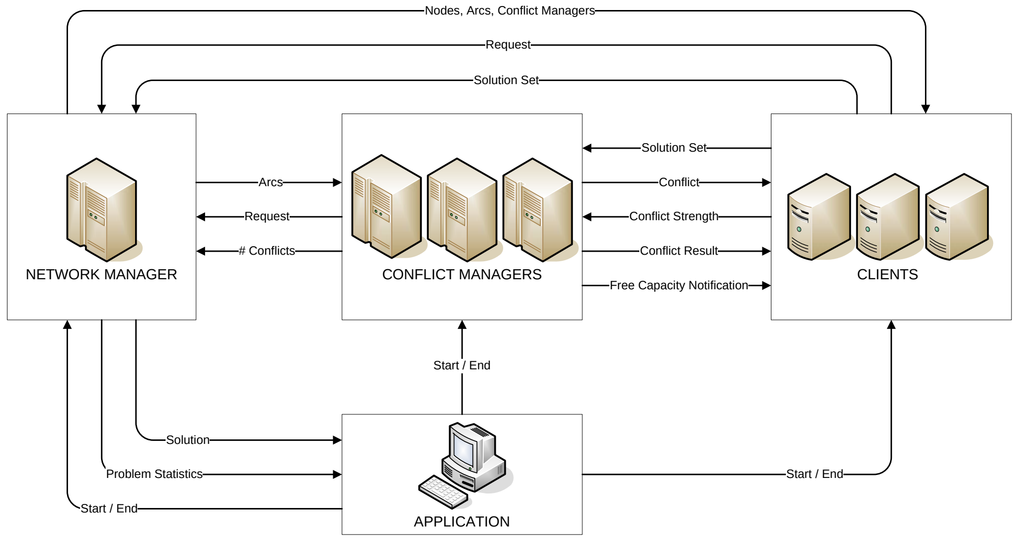
Chapter 5

Implementation

The implementation of the algorithm is the core feature. The Software Model is created in such a way that it is generic and can be used to solve all kinds of Multicommodity Minimal Cost Flow Problems without changing the architecture. It is possible to split the problem in very tiny bits and create specialized components for every task. Furthermore it is able to run multi threaded and distributed over a network.

5.1 Software Model

The software model consist of several components that do a specific task. Most of them are used multiple times in order to limit the number of calculations that a single component has to perform. There are four types of components: Application, Network Manager, Auction Manager, and Client. They communicate with each other the following information.



The only thing that the application has to do, is controlling the algorithm and saving the found solution. The application is communicating with all other components, sending them messages to shutdown if desired. Furthermore, it gets information from the Network Manager about the problem and the solution. At the moment the Network Manager notices that a solution is found, it will send this solution to the Application which has to store it.

The Network Manager stores the base of the program: the problem. Its function is to communicate the problem to the Clients and Auction Managers if they request for it. Furthermore, it collects information about the state of the solution. If all Auction Managers are sending the finished-message, then it will collect the solutions from the Clients and send them to the Application. Although it is possible to split the tasks into several pieces, it is probably enough to have only one instance of the Network Manager.

The Auction Manager is responsible for the handling of an auction. All Clients are sending the arcs they are using to the different Managers, which will notice any conflict and inform the clients about the auction. At the moment they receive the results they will determine the winners and losers of the auction and send them the new capacities. Furthermore, the Auction Manager has to keep track of all arcs that had an auction, but are unoccupied now. Because Auction Managers have many tasks it is probably needed to add many of them to the model. They are, together with the Clients, responsible for the distributed part of the algorithm and are the core feature with respect to solving the Multicommodity Minimal Cost Flow Problem in a distribute manner.

Clients represent the commodities in this model. They have to constantly calculate solutions to the problem with changing capacities. Clients have to be run on fast machines with a lot of memory, because they are the only part that really solve problems.

This model makes sure that every part of the process runs in a specific environment with enough memory and speed to finish its tasks. Every part knows only that part of the problem that is needed to solve its own sub-problem. It is even possible to distribute the problem over the internet creating Peer-to-Peer[T4] opportunities and enables projects like SETI@Home.

5.2 Software and Hardware

Because of the limited scope of this paper it was chosen to use only very basic algorithms within the Clients. Various specialized algorithms could increase the performance of the algorithm further. Clients are solving the Minimal Cost Flow Problems with an ordinary simplex solver called LPsolve [23]. LPsolve is a Mixed Integer Linear Programming solver, distributed under Lesser General Public License. Each of the problems can be solved without integer constraints because the setting of the problem ensures integer values. The software model is implemented in Java and uses the Socket object for communication between the components. These components are running on two machines. Machine one is running the Application, Network Manager, Auction Manager, and 1/3 of the clients. The other machine is running the rest of the clients. Both machines have an Intel Core 2 Duo processor with 1 GB of memory

For solving the problem with the Price-Directive Decomposition the algorithm as stated in the thesis of Bobonneau [1] is implemented. The initialization phase is the most memory and time expensive, therefore only this part of the algorithm is measured. LPsolve is used to calculate the base solution on an Intel Core 2 Duo processor with 1 GB of memory. In this phase the following Linear Programming Model is solved:

$$min \sum_{k,m} s^{-} \tag{5.1}$$

$$\sum_{e_m \epsilon A_n} x_m^k - \sum_{e_m \epsilon B_n} x_m^k = \begin{cases} S_k, & \texttt{if } n = s_k \\ -S_k, & \texttt{if } n = t_k \\ 0, & \texttt{otherwise} \end{cases} \quad \texttt{all } n, k \tag{5.2}$$

$$\sum_{k} x_m^k - s^{-} \leq b_m\texttt{, all } m \tag{5.3}$$

$$M \cdot Z_m^k \geq x_m^k\texttt{, all } k, m \tag{5.4}$$

$$\sum_{k} Z_m^k \leq 1\texttt{, all } m \tag{5.5}$$

$$x_m^k \geq 0 \tag{5.6}$$

$$s^{-} \geq 0 \tag{5.7}$$

$$0 \leq Z_m^k \leq 1\texttt{, integer} \tag{5.8}$$

With M a number bigger than the maximum capacity. s^{-} is a vector representing the overcapacity on each certain arc. Equations (5.4), (5.5) and (5.8) are used to enforce the disjoint constraint. Every feasible solution to the original model will have costs of 0 in this model.

Although it would be fair to test both algorithms only for the time that is needed to find a feasible solution, we have chosen to enforce an optimal solution of the AA algorithm. If the AA algorithm can beat the initialization phase, which is not a part of the PDD algorithm, then it would beat the whole algorithm, as described by Bobonneau, too.

Chapter 6

Problem setting

The problems that are used to compare both algorithms are generated by a problem generator that is specifically written to generate Multicommodity Undirected Disjoint Arc Minimal Cost Flow Problems. It takes as input the number of nodes (n), the number of arcs (a) and the number of commodities (k). First n nodes are created, after that a MCF generator is used to generate k different Shortest Path Problems. Now each of the nodes in these problems is mapped to one of the created nodes, and its arcs are mapped to the new problem too. After that a number of arcs are added till the problem reaches the desired number of a arcs. The capacity of each arcs is drawn from the collection of capacities of the arcs connected to both nodes. The last step is to draw random costs for every arc. With this generator the 10 instances of the following problems are generated and solved by both algorithms.

Due to the limited amount of time, we have chosen to test only two types of setups. The first one is rather small, but the second one is two times as big. These two setups are used for testing the scalability. Both setups have the same ratio of arcs, nodes, and commodities.

Arcs	**Nodes**	**Commodities**
24	12	6
36	18	9

Table 6.1: Problem settings

Chapter 7

Results

In the following table, the performance of the Auction Algorithm(AA) and the Price-Directive Decomposition(PDD) are displayed. In experiments with only a small number of arcs and commodities, the PDD performs excellent (except for one run), but when the problems get bigger, AA outperforms PDD. This is due to the fact that the profit of using AA is higher than the costs of the communication between the components. These costs caused the performance loss of AA on the small experiments.

Furthermore, we see that the total processor time used by AA is 35% higher than the total time used by PDD. This means that if we had only used one machine, executing AA would take much more time than executing PDD. The runtime of PDD is highly affected by the scale of the problem, while AA can still separate the work and only faces an increased number of conflicts. This results into a lower amount of time needed to finish the problem, although it needs more processor time.

Arcs	**Nodes**	**Commodities**	AA		PDD	
			avg	sd	avg	sd
24	12	6	16.2	329.2	22.7	135
36	18	9	49105	26825	71472	31377

Table 7.1: Performance of the algorithms in minutes

Figure 7.1 and Figure 7.2 illustrate the performance of the two algorithms in both setups. We can conclude that the time AA needs to find the optimum depends much more on the problem, 'easy' problems are solved much faster than 'hard' problems with the same number of commodities, arcs, and nodes. This is mainly caused by the number of conflicting paths. For example, if the shortest routes of all commodities are disjunct, then the optimum will be found within a few seconds, while problems with only one feasible solution and a lot of conflicting paths result into a very long runtime. Finally we did one last test with 400 arcs, 200 nodes and 100 commodities, which resulted into an Out-of-Memory error when calculating the base solution of PDD, while the combined strength of six machines running AA was enough to find the optimum within 7 days.

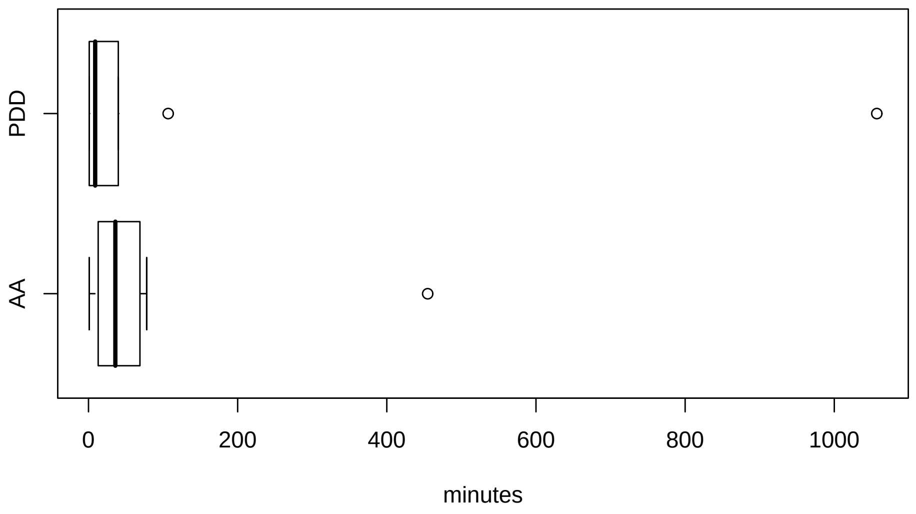


Figure 7.1: Performance on the small-sized experiments

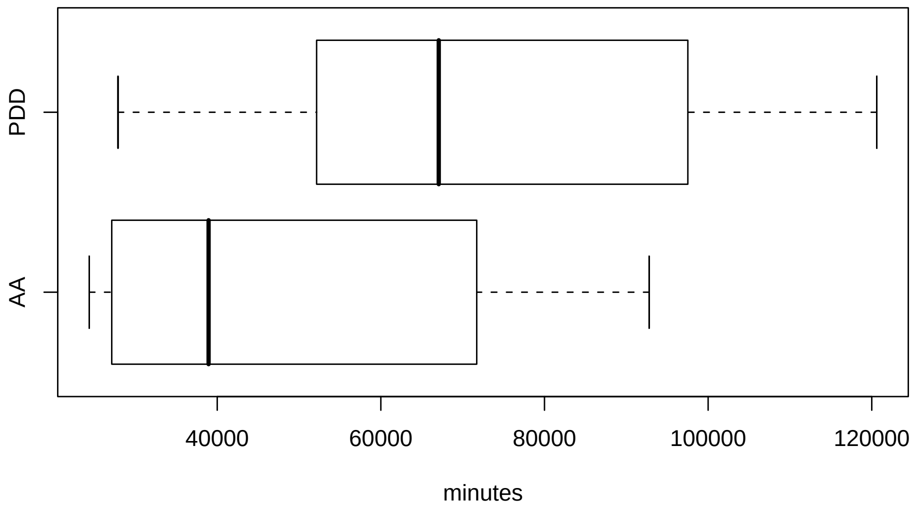


Figure 7.2: Performance on the large-sized experiments

Chapter 8

Conclusion

We can conclude that the Auction Algorithm outperforms Bobonneau's implementation of PDD on big Multicommodity Undirected Disjoint Arc Minimal Cost Flow Problems without a base solution. Although the total processor-time on AA experiments is at least 35% higher than PDD experiments, these runs are finished much earlier, due to the distributed computations. Therefore, future research should investigate the effects of increasing the number of used processors, with respect to the runtime. We believe, it will outperform other 'distributed' algorithms too, if there are enough processors available.

Furthermore, we can conclude that there are problems that can not be solved with algorithms that only divide workload, because of a shortage in memory when searching for a base solution. The Auction Algorithm, however, is able to find the optimal solution for this problem.

There are some disadvantages of the Auction Algorithm. First of all, it is very sensitive to the 'difficulty' of the problem. If the problem is 'hard' for AA to solve, namely if there are a lot of conflicts, then PDD outperforms AA. In 40% of our experiments, this was the case, therefore it is not guaranteed that using AA is profitable.

8.1 Future Research

The results from Chapter 7 are very promising. Although more data is needed for validation, it is likely that the distributed calculation is possible and profitable. Further research is needed to determine if AA is only useful for finding a base solution or the AA also performs good on finding the optimum.

In this paper, we discussed only a very small part of the problems that can be solved with this algorithm. Many of the variants of the problem require a different approach and algorithm within Clients and Auction Managers. Creating sub-algorithms for these variants can lead to a whole new range of problems that can be solved.

Furthermore, there is enough to improve within this variant too. In order to simplify the model, only very simple algorithms are used to solve the MCF problems. It would be interesting to see the performance of this algorithm when using specialized techniques instead of simplex. To investigate the true effect of the distributed algorithm, it is needed to run some tests with more than two processors. This can give insight in the relation between performance, the number of commodities, and the number of clients.

And last but definitely not least, it would be interesting to create other algorithms that can be executed distributed and are able to finish their task even faster. In this age of internet, we can use the strength of many.

Chapter 9

Terminology

T1 **Linear Programming Model**
A linear programming model is a kind of mathematical problem which consists of a objective function and multiple constraints. The objective function and constraints have to be a linear combination of variables.

T2 **NP-Complete**
A NP-complete problem is a kind of problem that is seen as the most difficult type of problems within operational research. Problems of the type NP-complete are probably not possible to solve within polynomial time.

T3 **Shortest-Path Problem**
A Shortest-Path Problem is a problem that consists of a network of nodes and arcs with a certain length. The objective is to find the shortest route from one node s to another node t.

T4 **Peer-to-Peer**
Peer-to-peer is a communications model in which each party has the same capabilities and either party can initiate a communication session.

Appendix A

Detailed Results

A.1 Auction Algorithm

Arcs	Nodes	Comm	Minutes				
24	**12**	**6**	69	38	43	13	1
			78	455	34	7	28
36	**18**	**9**	92792	24357	34461	71719	46516
			28284	34454	37377	28540	23032

Table A.1: Detailed Performance of the Auction Algorithm in minutes

A.2 Price Directive Decomposition

Arcs	Nodes	Comm	Minutes				
24	**12**	**6**	1	40	107	1	5
			1	22	1057	1	13
36	**18**	**9**	97523	109110	27875	120616	30328
			43428	27109	32513	92491	25669

Table A.2: Detailed Performance of the Price-Directive Decomposition in minutes

Bibliography

[1] F. Bobonneau. *Solving the multicommodity flow problem with the analytic center cutting plane method*. PhD thesis, Universite de Geneve, 2006. [cited at p. 2, 3, 23]

[2] H. Hai, I. et, and M. A distributed processing algorithm for solving integer programs using a cluster of workstations, 1994. [cited at p. 4]

[3] S. Tschoke, R. Lubling, and B. Monien. Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network. *ipps*, 00:182, 1995. [cited at p. 4]

[4] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing 5*, 1976. [cited at p. 7]

[5] G. Bennington M. Bellmore and S. Lubore. A multi-vehicle tanker scheduling problem. *Trans. Sci.*, (5):36–47, 1971. [cited at p. 7]

[6] S.P. Bradley. Solution techniques for the traffic assingment problem. *ORC 65-35, Operational Research Center, University of California, Berkeley*, 1965. [cited at p. 7]

[7] E.C. Carter and J.R. Stowers. Models for fund allocation for urban highway systems capacity improvements. *Highway Res. Rec.*, 20:84–102, 1963. [cited at p. 7]

[8] A. Charnes and W.W. Cooper. *Multicopy Traffic Network Models*. Elsevier Publishing Co. Amsterdam, 1961. p.p. 85-96. [cited at p. 7]

[9] S. Clark and J. Surkis. An operations research approach to racial desegregation of school systems. *Socio-Econ. Plan Sci.*, (1):259–272, 1968. [cited at p. 7]

[10] W.W. White and E. Wrathall. A system for railroad traffic scheduling. Technical Report 320-2993, IBM Philidelphia Scientific Center, 1970. [cited at p. 7]

[11] A.M. Geoffrion and G.W. Graves. Multicommodity distribution system design by Bender decomposition. *Management Sci.*, (20):822–844, 1974. [cited at p. 7]

[12] S. Krile. Application of the minimum cost flow problem in container shipping. *Electronics in Marine, 2004. Proceedings Elmar 2004. 46th International Symposium*, pages 466–471, June 2004. [cited at p. 7]

[13] A. Yahaya and T. Sunda. IREX: Inter-domain QoS automation using economics. In *Proceedings of IEEE CCNC*, 2006. [cited at p. 7]

[14] A. Yahaya and T. Sunda. IREX: Inter-domain resource exchange architecture. In *Proceedings of IEEE INFOCOM*, 2006. [cited at p. 7]

[15] G. Baier, E. Kohler, and M. Skutella. On the k-splittable flow problem. *Algorithmica*, (42):231–248, 2005. [cited at p. 10]

[16] M. Martens and M. Skutella. Length-bounded and dynamic k-splittable flows. In *Proceedings of the International Scientific Annual Conference 'Operations Research 2005'*, 2005. [cited at p. 10]

[17] L.R. Ford and D.R. Fulkerson. A suggested computation for maximal multicommodity network flows. *Management Sci.*, (5):97–101, 1958. [cited at p. 11]

[18] G.B. Dantzig and P. Wolfe. The decomposition algorithm for linear programming. *Econometrica*, 29(4):767–778, 1961. [cited at p. 11]

[19] H.A. Aashtiani and T.L. Magnanti. Implementing primal-dual network flow algorithms. *Technical Report OR 055-76, Operational Research Center, M.I.T Cambridge*, 1976. [cited at p. 12]

[20] G.G. Brown G.H. Bradley and G.W. Graves. Design and implementation of large-scale primal transshipment algorithms. *Management Sci.*, 24(1):1–34, Sept 1977. [cited at p. 12]

[21] C.M. Shetty R.W. Langly, J.L. Kennington. Efficient computational devices for the capacitated trasnportation problem. *Naval Res. Logist. Quart.*, (21):637–647, 1974. [cited at p. 12]

[22] M. Nanjanath and M. Gini. Dynamic task allocation for robots via auctions. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, 2006. [cited at p. 13]

[23] M. Berkelaar, K. Eikland, and P. Notebaert. lp_solve (alternatively lpsolve). GNU LGPL (Lesser General Public Licence). Version 5.5.0.10. [cited at p. 22]