Master's degree thesis

INF950

Solving combined transportation and production problems using metaheuristic parallelization

Oskar B. Bævre

Børge Gjengstø

Number of pages, including the first page: 75

Molde, 2009

Molde University College

Publication agreement

Title: Solving combined transportation and production problems using metaheuristic parallelization

Author(s): Oskar B. Bævre, Børge Gjengstø

Subject code: INF950

ECTS credits: 30

Year: 2009

Supervisor: Arne Løkketangen

I/we hereby give Molde University College the right to, free of charge, make the thesis available for electronic publication:	[✓ **yes**] **no**
Is there an agreement of confidentiality? (A supplementary confidentiality agreement must be filled in)	**yes** [✓ **no**]
- If yes: **Can the thesis be online published when the period of confidentiality is expired?**	**yes** **no**

Date: 25.05.2009

Abstract

This master thesis addresses both a combined production and distribution problem, The Integrated Production-Distribution Problem(IPDP), and modernization of meta heuristic algorithms by introducing parallelization.

The IPDP consists of two NP-hard problem types; Vehicle Routing Problems(VRP) and Lot Sizing Problems(LSP). When these are combined the resulting problem gets even harder to solve. The IPDP minimizes the total cost of both production and trasportation in order to globally streamline the supply chain.

In the recent years there has been a generation change in the processor architecture on personal computers. The clock frequency is not increasing much anymore, instead the increased computational power is achieved by increasing the number of separate cpu-cores inside the processor. To make applications able to unlock the increased computational power these multicore processors offer, the applications must be multithreaded. This means that applications must be developed with parallelization in mind. This thesis presents a scaleable multithreaded tabu-search algorithm for solving the IPDP, which can fully utilize all available cpu-cores in modern multicore processors. In this process several different and interesting strategies for meta heuristic parallelization are discussed. The results obtained and presented clearly shows that parallelization of meta heuristics is well worth the extra effort.

Acknowledgements

We would like to thank our supervisor, Professor Arne Løkketangen, for being a great source of inspiration, motivation and information. Without his good expertise in the field of meta heuristics, his ideas, constructive criticism and the discussions with him, this master thesis would not been at the level it is today. His efforts are highly appreciated.

Table Of Contents

Abstract 3

Acknowledgements 4

1 Introduction 7

1.1 Motivation and background . . . 7

1.2 Tools used . . . 8

1.2.1 Applications and libraries . . . 8

1.2.2 Compilers . . . 8

1.3 Outline . . . 8

2 Integrated Production-Distribution Problem 9

2.1 Inventory management strategies . . . 10

2.1.1 VMIR-OU (Order-up-to level) . . . 10

2.1.2 VMIR-FFD (Fill-Fill-Dump) . . . 10

2.1.3 VMIR -ML (Maximum level) . . . 11

2.1.4 VMIR . . . 11

2.2 VRP - Vehicle Routing Problem . . . 11

2.2.1 Extensions . . . 12

2.3 LSP - Lot Sizing Problem . . . 13

2.3.1 Extensions . . . 13

3 Methods for solving IPDP, VRP, LSP 14

3.1 Exact methods . . . 14

3.2 Heuristics . . . 14

3.2.1 VRP Construction heuristics . . . 14

3.2.2 VRP Improvement heuristics . . . 16

3.2.3 LSP Construction heuristics . . . 16

3.2.4 IPDP heuristics . . . 16

3.3 Meta heuristics . . . 16

4 Parallelization 20

4.1 Parallelization strategies . . . 21

4.2 More advanced parallelization schemes . . . 22

4.3 Meta heuristic parallelization 22

5 Mathematical model **25**

6 Implementation **28**

6.1 Tabu search 28

6.1.1 Initial solution 28

6.1.2 Objective function 29

6.1.3 Neighborhood 29

6.1.4 Move 29

6.1.5 Move evaluation 30

6.2 Parallelization scheme 32

6.2.1 Conceptual model for the parallelized IPDP solver 32

7 Computational experiments **35**

7.1 Test Instances 35

7.2 Parameter testing 35

7.3 Results 41

7.3.1 Scalability 41

7.3.2 Solution quality 43

8 Conclusion and future work **46**

A Appendix **50**

A.1 Visualization 50

A.1.1 SVG 50

A.2 TBB implementation 52

A.2.1 Parallel_reduce (used in VRP) 52

A.2.2 Task 53

A.2.3 Parallel_scan 53

A.3 Compiler differences 54

A.4 Computational results 54

1 Introduction

1.1 Motivation and background

During the spring 2008 we attended the course *heuristic optimization methods* lectured by professor Arne Løkketangen at Molde University College. This course was our introduction to the world of combinatorial optimization and heuristics, and it sparked our interest in this reaserch field. One part of the heuristics course was to apply a meta heuristic for solving the 0/1 Multiple Knapsack Problem. This work should result in a scientific article. It was during the work with our article *Solving large Multiple Knapsack Problems using tabu search* (2008), we decided that we wanted a master thesis in the field of combinatorial optimization using meta heuristics.

The choice of supervisor was then quite simple, professor Arne Løkketangen is one of the leading reaserchers in combinatorial optimization and was therefore our main choice. Arne Løkketangen had many proposals for a master thesis, and in cooperation with him we finally chose the Integrated Production and Distribution Problem (IPDP) and solver parallelization.

This thesis will continue the work presented in the Doctoral Thesis of Dr. Andre Shiguemoto (2009) (at UNICAMP, Campinas, Brazil). In his thesis he proposes an integrated model for the IPDP problem. This problem consists of the subproblems production and distribution, and when combined the new problem has increased complexity.

Over the past couple of decades there has been a substantial increase in computing power in normal personal computers, ranging from a few mega hertz to giga hertz, and the computational power of modern PC's can be multiplied with thousands over computers from the early 90's. A couple of years ago there was a generation change in the CPU architecture which introduced the multi-core processors. As a result of this the clock-frequency on new processors are not increasing very much, instead the number of CPU-cores increases. These new multi-core processors are especially good on multi-tasking. Since all CPU-cores operate in parallel, several tasks can be run at the same time. In theory a dual-core CPU is twice as fast as a single-core CPU running at the same frequency. In practice this is true for the whole system with all its running processes, but for running a traditional single-threaded program, the performance gain a modern multi-core processor can give is very small compared to a single-core processor running at the same clock-frequency. This is because single-threaded applications only can utilize one CPU-core, which means that for a quad-core processor, only 1/4 of the processing power will be utilized.

In this thesis the focus will be on looking into different parallelization schemes for modernizing meta heuristics to better utilize the increased computational power available in new multi-core processors. We will develop a scaleable parallel tabu search for the IPDP by splitting work into separate parallel tasks and solve the IPDP problem by cooperation between these tasks.

1.2 Tools used

1.2.1 Applications and libraries

- Xcode 3.1
- Visual studio 2008
- Intel Threading Building Blocks libraries(TBB)

1.2.2 Compilers

- Gcc 4.0.1
- Gcc 4.3.3
- Intel c++ 11.0.072
- Microsoft Visual C++ 15.0.30729.1

1.3 Outline

The thesis is structured in the following way: In chapter 2 the Integrated Production Distribution Problem, and its sub problems Vehicle Routing Problem and the Lot Sizing Problem is described along with different inventory management policies. Chapter 2 also contains a brief summary of previous work on the IPDP. Chapter 3 discusses various methods used for solving the IPDP, from exact methods to modern meta heuristics. Chapter 4 adresses parallelization and its limitations and strong sides. Different strategies for solver-parallelization is also discussed in this chapter. Chapter 5 contains the mathematical formulation for the IPDP, followed by the implementation details and method description of the presented tabu search algorithm in chapter 6. In chapter 7 the computational results obtained and test instances used are presented and described. Chapter 8 contains conclusions and future work.

2 Integrated Production-Distribution Problem

In the last couple of decades there has been an increased focus on optimizing and streamlining the supply chain in order to reduce costs. The advance in technology along with new models and algorithms have given better information and communication systems that has made coordination of the whole supply chain more manageable. Larger parts of the supply chain is now optimized globally in one step. In the case of the IPDP this means that both the manufacturing, inventory at the plant, the transportation and the inventory monitoring at the retailers are considered at the same time. This is in contrast to the more traditional decoupled two-step approach, where the production plan is produced first, and then a distribution plan is made based on the production plan. In most cases such global integrated optimizing of the supply chain gives cost reductions far beyond traditional optimization approaches.

As briefly mentioned, the Integrated Production-Distribution Problem considers both the manufacturing of products at the plant, the distribution to the retailers and managing their inventory levels. The IPDP can therefore be broken down to 1) Capacitated Lot Sizing Problem (CLSP) with multiple products and periods, and 2) a multi period split delivery CVRP. Each of these two sub-problems are considered to be computational hard to solve, since both problems belong to the complexity class NP-Hard.

In the last couple of years there has been published some interesting articles regarding solving the IPDP. Boudia et al. (2007) developed a reactive GRASP method for a combined production-distribution problem where the goal is to minimize the sum of three costs, namely: production, transportation and inventory costs. Later M. Boudia, C. Prins (2009) developed and applied a *Memetic Algorithm with Population Management (MA|PM)* to the exact same test-instances. These instances concern a single product and 50, 100 and 200 customers in a horizon of 20 time-periods. The results show a saving of 23 % or more, over classical decoupled optimization methods.

Archetti et al. (2007) had another approach, they used an exact method, *branch and cut*, for solving the problem to optimality while following and comparing three vendor inventory management strategies. The first strategy is the order-up-to-level (VMIR-OU), the second strategy allows for any delivery quantity as long as it is not violating the minimum or maximum inventory levels at the vendor (VMIR-ML). The third strategy is the same as the second strategy but disregards the vendors maximum inventory level (VMIR).

The work resulted in a comparizon of these inventory management strategies which shows the effect relaxation of the shipment quantities has on the total cost. There was large savings when the less constrained strategies were applied compared to the VMIR-OU.

Shiguemoto used a tabu search procedure to 1) compare the VMIR-OU and VMIR-ML strategies, and 2) compare the integrated VMIR-ML strategy against the decoupled VMIR-ML strategy. The results presented shows that the VMIR-ML yields significant reduction in the cost compared to the more restrained VMIR-OU, and that the integrated VMIR-ML approach is far superior to the decoupled approach.

The results and conclusions of the articles are all similar. They all conclude in that an integrated approach between production and distribution planning in a supply chain can give a substantial cost saving over traditional decoupled approaches.

2.1 Inventory management strategies

There are many different inventory management strategies, still all these strategies can be divided into two main classes. The first class is the **Retailer Managed Inventory (RMI)**, which is often considered the traditional strategy where the retailers manages their own inventory. This strategy is still common practice for many companies, depending on their size and business segment. The second class is the **Vendor Managed Inventory (VMI)**, where the vendor is responsible for managing the retailers inventory. There are many variants of VMI that are applied in different supply chains, and the most common is listed below.

2.1.1 VMIR-OU (Order-up-to level)

The vendor must deliver a quantity that brings the inventory for each retailer after each delivery, up to a predetermined level. This is a relatively constrained strategy due to the predetermined level the inventory must have after each delivery. This gives less possible solutions, and therefore this is depending on the inventory and vehicle capacity. This is often an expensive policy since one can expect that shorter but more routes are needed to satisfy the retailers delivery demand.

2.1.2 VMIR-FFD (Fill-Fill-Dump)

In this VMI strategy the quantities delivered to the retailers must bring the inventory level up to a predetermined level, as in the VMI-OU, except for the last retailer at each trip.

The quantity delivered to the last retailer on the route can be any quantity, typically the remaining capacity of the vehicle.

2.1.3 VMIR -ML (Maximum level)

When this strategy is enforced it allows for any quantity to be delivererd as long as the inventory bounds at the retailers are not violated after the delivery. The inventory must also stay between these bounds until the next delivery. Compared to the VMI-OU, and VMI-FFD strategies the VMI-ML is less constrained and more possible solutions exists. This means that the cost will be less than with the VMI-OU and VMI-FFD strategies or in worst case the same. This usually means that the optimal solution is harder to find, but good solutions are easier found.

2.1.4 VMIR

A VMI strategy where the maximum inventory level is disregarded. The only constraint is that the lower bound on the inventory levels must not be violated. This strategy is often impractical because it requires unlimited storage capacity at the retailers. In most real world situations this is not realistic since most retailers have a capacity limit on the inventory.

2.2 VRP - Vehicle Routing Problem

The Vehicle Routing Problem was first formulated and described by Dantzig and Ramser (1959). Since then the Vehicle Routing Problem has been a popular research topic in the field of combinatorial optimization. The VRP is defined on a complete graph $G = (N, A)$ where $N = \{0, ..., n\}$ is the set of Nodes, and $A = \{(i, j) : i, j \in N\}$ is the arc set. Node 0 represents the depot and the rest of the nodes represent the retailers. The retailers have a certain demand, d_i that must be fulfilled and the cost of travelling between node i and j is defined by a cost c_{ij}.
The classical variant of the VRP is referred to as the capacitated VRP (CVRP). In most VRP variants, the vehicle fleet is uniform, meaning that each vehicle is identical and have a capacity q.

The objective of the classical VRP is to minimize the overall cost, usually the travel distance, for the whole set of routes. All vehicles are required to start and end at the depot, and the total demand for each route can not exceed the capacity q of the vehicle. Each retailer must be visited once by only one vehicle, and get their demand fulfilled.

Solving large VRP problems are considered to be a computational hard optimization problem, because of the exponentional growth of the problem size. The VRP problem belongs to the class of NP-Hard problems, which means that for larger problem instances, heuristic methods usually must be applied as solver, as complete methods in general takes too long time, or runs out of memory.

When heuristics are applied this usually means that the goal is to find good solutions fast, rather than finding the global optimal solution.

To make the VRP models fit to real world applications better, the VRP has a variety of extensions. These extensions adds complexity to the basic model which in general makes it harder to solve to optimality.

2.2.1 Extensions

- **Multiple periods**. If the Planning period is longer than one day, the problem is called a *Periodic VRP*. In a periodic vehicle routing problem the goal is to find the optimal routes for a *T-day* period.
- **Time Windows**. If the retailers have certain time windows when they must be served, constraints for this must be added. Such constraints can be either hard or soft, depending on the real world problem. When time windows are implemented as hard constraints, no violation of time windows are allowed in a feasible solution. Soft constraints usually adds a penalty to the solution if there is a constraint violation, meaning that the solution still is feasible.
- **Distance or duration constraints**. Typically maximum distance per route, or maximum time consumption per route. This will of course have an influence on the number of routes and the total cost for the whole set of routes.
- **Heterogeneous vehicle fleet**. In many real world situations, a company have many types of vehicles. These vehicles can range from small to large, have different purposes and varying cost efficiency.

- **Split delivery** When split delivery is allowed, a customer can be visited by more than one vehicle each period. The customer's demand then gets fulfilled by one or more deliveries. In real life this can often occur because of capacity limits on the vehicles.
- **Multiple depot**. There might be more than one depot, which introduces the choice of which depot that should supply which customer.
- **Retailer preferences**. Certain customers may want to be served from a specific type of vehicle or a certain driver. In the real world, customer preferences can be important for the supplier - customer relationship.

2.3 LSP - Lot Sizing Problem

The objective of the Lot Sizing Problem is to find the minimal cost for producing products such that the demand for each period is met, and considers the setup costs, inventory holding cost, production capacity and other costs that may apply to the production. This problem is NP-hard if there are multiple periods and products. In order to apply to real world situations the LSP has a number of extensions. These extensions constrains the original problem. As a result the complexity increases and the new problem is harder to solve.

2.3.1 Extensions

- **Multiple products**. More than one product must be produced. Different setup costs, production costs and inventory holding costs may apply.
- **Time windows**. Certain products could only be produced in given time windows. Any production of these products outside the time window is not allowed.
- **Lead time**. Different products can have different lead time, this can reduce the possibilities for when to start production in order to meet the demand.
- **Back orders**. Back orders can be allowed for all or some of the products.

3 Methods for solving IPDP, VRP, LSP

There are several different approaches for solving the IPDP or its sub problems, from exact methods to modern meta heuristics. Exact methods are usually preferred if the problem size is manageable because exact methods do guarantee to find the optimal solution. However in real world situations the problem size is often very large and solving them with exact methods will be impractical as they either take too long time or the method will run out of memory. In such cases it is common to use heuristic methods to find good solutions. Most heuristic methods have no guarantee for the solution quality, but in practice good meta heuristic methods usually finds good or near optimal solutions.

3.1 Exact methods

Exact methods examines every possible solution, but can use cut-off mechanisms to remove uninteresting parts of the search space. Branch-and-bound and branch-and-cut are examples of this. Memory problems and time consumption are the main issues for exact methods. As a result exact methods are not suited to solve large instances of NP-hard problems, but should be used over heuristic methods if the problem size is manageable.

3.2 Heuristics

Heuristics are methods that are used to find solutions to spesific problems. The different heuristic methods use different techniques for reaching a solution for the same problem. Still all heuristics use some knowledge of the problem to guide the search to a solution. Heuristics can be divided into two groups, construction heuristics and improvement heuristics. Construction heuristics create (feasible) solutions from the data input, and improvement heuristics try to improve solutions that already exists.

3.2.1 VRP Construction heuristics

Typical construction heuristics for CVRP are algorithms like the sweep algorithm by Gillet and Miller (1974) and the savings algorithm by Clarke and Wright (1964).

Sweep heuristic

The sweep heuristic is a construction heuristic that makes feasible VRP routes. Feasibility is judged by given constraints that must be satisfied, like travel time, distance, capacity, or other constraints. A route is constructed by drawing a straight line away from the depot or plant, then moving the line in one direction (left or right) while feasibility is still true (figure 1).

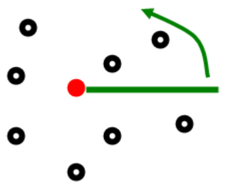


Figure 1: Sweep algorithm

Savings heuristic

The savings heuristic is also a construction heuristic that makes feasible VRP routes. A savings matrix is calculated before the algorithm is run. The matrix contains the distances between all the customers and depots. The problem size is n^2 if asymmetric and $\frac{n^2}{2}$ if symmetric. Initial routes are created for every customer, from depot to customer and back to the depot again. If N denotes the number of customers, there will be N routes. After this the algorithm checks if there will be a saving if two routes are merged. The saving is computed from the formula $S_{ij} = C_{dj} + C_{id} - C_{ij}$. If the cost of the merged routes will be less or equal than the two separate routes, they are merged. The highest saving will be chosen first. When routes are merged, the total route cost will decrease and this reflects back to the name, Savings heuristic.

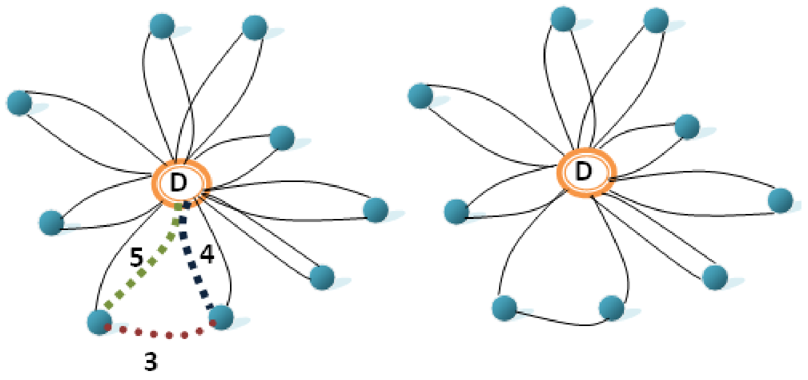


Figure 2: Savings algorithm

$C_{dj} = 4, C_{id} = 5, C_{ij} = 3$. The total cost of the two separate routes are $2*4+2*5 = 18$. The saving of merging these routes are $S_{ij} = 4+5-3 = 6$. If there are no route combinations that give higher saving than 6, these two routes will be merged first, as seen in figure2.

3.2.2 VRP Improvement heuristics

Improvement heuristics like 2-OPT, 3-OPT and K-OPT can be applied after a VRP-solution has been created, in order to improve the solution by reducing the travel distance for the routes.

These improvement heuristics can be implemented as both intra-route and inter-route optimization. With Intra-route optimization, one single route is optimized by changing the order in which the customers are visited. The inter-route optimization goes beyond a single route, including 2 or more routes and works by exchanging customers between the routes to improve the total cost of the routes.

3.2.3 LSP Construction heuristics

In lot sizing problems, the most known construction heuristics is the orginal Wagner and Whitin (1958) algorithm, and later the Evans (1985) heuristic which is an efficient reimplementation of the Wagner and Whitin algorithm.

Evans heuristic

The Evans heuristic is an efficient implementation of the Wagner-Whitin algorithm. The heuristic calculates the cost of producing products for the next period(s) vs the cost of producing it the period it is demanded. It does so in a recursive manner. Plant production capacity and inventory capacity is not taken into consideration, and can create infeasible solutions. This is a dynamic programming implementation that requires little memory and is very fast.

3.2.4 IPDP heuristics

There are no known IPDP construction heuristics known to the authors of this article. However since the IPDP is a combined production and transportation problem, an IPDP solution can be generated by combining construction heuristics for LSP and VRP.

3.3 Meta heuristics

Classical construction heuristics can generate a solution, but usually get stuck in local optima because it always choose a move that makes the objective value better than the previous. When no improving move is available the search stops. Meta-heuristics can perform moves that will worsen the objective value, in order to get out of the local optima.

A meta-heuristic method is a strategy used to guide heuristics so that the heuristic can continue past the local optima. Meta-heuristics have been developed and widely applied to combinatorial optimization problems in the last couple of decades. Still most meta-heuristics do not guarantee to find the global optimum solution, but good solutions can often be obtained fast.

Meta-heuristics can be divided into two groups, **local search based** and **population based** meta-heuristics.

Local search

There are a number of available local search versions, from purely deterministic to stocastic moves. Examples of such versions are steepest descent and random walk. Steepest descent evaluates all neighbors in each iteration and select the neighbor leading to the best solution. Random walk selects its neighbor after a probability p. Depending on the probability for the neighbor, the chosen move can be either greedy like the steepest descent or purely random.

In order to perform a local search, four components must be defined.
1) A defined search neighborhood, 2) A defined move, which describes how to get from the current solution to a neighboring solution, 3) A move evaluation function to evaluate the possible moves, and finally choose the next move and 4) A stopping criterion which defines when the search should terminate.

Tabu Search

The problem in regular heuristics is that you always use the same move-criteria, ie. the best move. Tabu search was created by Glover (1986) as a means of getting out of local optima.

The idea in tabu search is to set some of the attributes of the moves made in a tabu status for a number of iterations, called tabu tenure. If a move is considered tabu, but will give a new best solution, the move is chosen anyway. This is called the aspiration criteria.

Other strategies and penalties can be added to guide the search in other directions. Two strategies are called intensification and diversification. The intensification strategy uses recency memory. A number of solutions are kept in memory, and a counting is performed on each value in them. The variable that has been in the solutions the longest will be held in place, while a search is performed in the remaining part of the solution. The diversification strategy forces the search into other areas of the search space that is not yet explored. This can be done by keeping a long term memory in form of a frequency table. The table keeps track of the number of times an attribute has been in a solution. This frequency can be added to the objective value, to make other attributes more attractive. In meta-heuristics the solutions does not have to be feasible. Moving between feasible and infeasible solutions can be done by using a process called strategic oscillation. Weights are added to the infeasibility component of the objective value in a strategic manner. If the solutions have been infeasible for a few iterations, the weight is increased. For feasible solutions, the weight decreases to encourage the search to go infeasible.

Simulated Annealing

Simulated annealing (SA) is inspired by the cooling of metal (Kirkpatrick, Gelatt, and Vecchi 1983). The algorithm is local-search based and allows non-improving moves to avoid getting stuck in a local optima. The search chooses a random neighbor and if it gives a new best solution $f(\overline{s}) < f(s)$, the move is taken unconditionally. If the move would give a solution $f(\overline{s}) > f(s)$, a probability $P(s, \overline{s}, T)$ is calculated based on the differences in the objective value for the new solution and the incumbent solution, and the temperature T. This probability is given to the solution $\overline{s}$ and is the probability for the move to be chosen. The solution $\overline{s}$ with the highest probability is the move that eventually will be chosen. The probability function makes SA able to escape local optima. If the temperature T is infinite the algorithm is poorly stochastic and all moves are chosen randomly. However if $T = 0$ the algorithm becomes a greedy algorithm that can not escape local optima. Initially the temperature T is often set high and a user specified cooling schedule is applied to gradually reduce the temperature at a given distribution that ends with $T = 0$. It is proven that SA will find the global optimum, but the run time needed could be infinite.

Population based

The most known population based meta heuristic is the **Genetic algorithm** introduced by Holland (1975) and is based on analogies in biology. The idea behind is the survival of the fittest, and terms from biology are used to describe the process of the algorithm. Initially a population(set) of solutions is created. The initial population can be created randomly or by a construction heuristic. An iteration consist of making a new generation. The search is done by crossover or mutation to create new individuals. The crossover takes two individuals and switches some of the genes at a crossover-point. Some of the individuals will be mutated. This means that a random gene will change value. The individuals are encoded with binary values, 0 or 1. A mutation is done by flipping a value 0 to 1 or 1 to 0. There is usually a small probability that mutation will occur, and is used as a diversification strategy. If there is a high diversification rate, many mutations will occur. The offspring of the crossovers and mutations are evaluated by a fitness computation, and the fittest individuals are chosen to be inserted into the new population.

4 Parallelization

Today the frequency(Hz) of new processors are not increasing very much. The reason for that is called electromigration. Electromigration is considered to be the result of momentum transfer from the electrons in the electric current. For semiconducting chips the problem arises when the circuit size decreases and frequency increases over a certain point. The result of electromigration is dramatically reduced life time or in worst case, a damaged chip. To avoid this phenomena, new processors have an increased number of CPU-cores placed inside the chip, rather than increasing the processor's frequency. This way new processors get increased computational power and still avoids electromigration.

Still, there is a drawback with this approach, namely that to make use of the increased computational power several tasks must be run at the same time. Traditional single threaded applications can only utilize one CPU-core, since one thread cannot be split over several cores. This means that there will be next to no gain in computational power for single threaded applications if run on a new 3 Ghz quad core CPU instead of an older 3 Ghz single core CPU. To be able to unlock the increased computational power these new multi-core processors have, the applications must be multi-threaded and developed with parallelization in mind.

Developing for parallelization generates some questions. E.g. what part of the algorithm should be run in parallel? Will parallelization give the wanted result? These questions are not always trivial to answer, but is important to think through. Since work has to be done simultaneously, data concurrency issues are due to arrive and the developer must apply techniques to prevent data corruption. Splitting up work into several threads and join the results generates administration work, called overhead. If the amount of overhead gets to large compared to the actual task that should be performed, a parallel implementation can actually be slower than a pure sequential implementation. Therefore the parts of an algorithm that is the most beneficial to parallelize is the parts that contains the largest work load.

Parallelization has been a popular research topic in computer science for many decades, which has resulted in many observations and conclusions regarding the applications for parallelization. According to Amdahl's law (1967), a program where every component is running with two parallel threads is twice as fast as a serial version of the program. One could imagine that if only half of the program were speeded up by 2x, the total speed up would be 1.5 over the serial version. However, this is not the case. A program where half of the components are speeded up 2x, will run only 1.33 times faster than the serial version.

Even if the parallel half of the program could utilize an infinite amount of processors, the program would not run faster than the remaining serial part of the program. In a way, Amdahls law can be used as an argument against taking the effort of parallelizing programs and applications, but there is another way to look at this, namely Gustafson's observations.

Gustafson (1988) stated that parallelization is more useful when the workloads are higher. He also noticed that as computers got more powerful, the applications became more complex and gave computers a higher workload. The most interesting part of Gustafson's observations, is that as the workload or problem size grows, the part that can be parallelized is the part that grows the most. Based on Gustavson's observations, meta heuristics should be able to get good profit from parallelization, since metaheuristics usually have no problem with small workloads due to the combinatorial nature of the problems applied. In fact a too large workload is usually the reason for why metaheuristics must be applied. Still, there is reason to believe that the difference between a good parallel metaheuristic and a serial version will grow with the problem size. This means that the usefulness of parallelism increases with the problem size.

(Le Bouthilliera and Crainic 2005) uses a multi-search approach for the vehicle routing problem with time windows. It is based on a solution warehouse strategy where the best solutions are kept. This multi-search approach is an attempt to solve problems more generally than other parallelized metaheuristics. This is done mainly by running several different methods in parallel, and applying some post optimization steps on each solution before considering if the solution should be added to the solution warehouse. Doing this will give a better chance of improving the solution. The results were quite good, and suggests that cooperation between solvers should be further experimented with.

4.1 Parallelization strategies

There are many strategies for implementing parallel solvers. Earlier the effort was focused around parallelization of the most computationally intensive step of the algorithm, using traditional master-slave schemes. In meta-heuristics, the most computationally intensive step usually is the neighborhood exploration. As the field of parallel metaheuristics evolved, it became clear that it was possible to go much further if the low level parallelization schemes, like master-slave, was abandoned. Schemes like master-slave give less CPU-time than pure sequential implementations, but they have the same overall behavior e.g. give the same search trajectories. It is clear that the sequential nature of these low level parallelization schemes is a performance limitation, but the lost opportunity to get more of the search space explored at the same time is possibly the weakest point for these schemes.

4.2 More advanced parallelization schemes

Most of the research in the field of parallel metaheuristics in the last decade has been on more high level parallelization approaches that give a new algorithmic behavior by letting several parallel components explore different areas of the search space simultaneously. When this type of search is done in a coordinated fashion, this approach becomes superior to the traditional parallelization schemes.

(Crainic, Toulouse, and Gendreau 1997) introduced a taxonomy of parallel Tabu Search approaches to classify known parallelization schemes and give researchers a better understanding of parallel tabu search. In addition the taxonomy should help point out interesting research areas, that would help finding new and interesting search strategies. This taxonomy is based on a three dimensional classification of the algorithmic features in the different parallel search schemes. Namely **Control cardinality**, **Control and communication type**, and **Search differentiation**.

The control cardinality defines whether one process control the search, called 1-control or in cooperation with other processes, called p-control. The control and communication type dimension, defines the level of communication and information sharing between processes. Levels ranging from **Rigid Synchronization(RS)** to **Knowledge Collegial(KC)**, which in other words mean; from the traditional master-slave scheme to a scheme that have asynchronous knowledge sharing between the processes in such fashion that processes in a way help guide each other toward better solutions. The last dimension in this taxonomy, search differentiation, addresses the following issues; Do the threads have the same start solution, do they use the same search strategies etc.

4.3 Meta heuristic parallelization

There are several parallelization strategies that can be applied for development of a parallelized meta heuristic. Some interesting strategies are listed and briefly described below ordered by implementation difficulty.

Master-slave with post-optimization

The VRP and LSP solver could run in a decoupled master - slave scheme, where one solver make a solution to use as input for the other solver. A separate task could be run in parallel to improve the solution(s), as shown in figure 3.

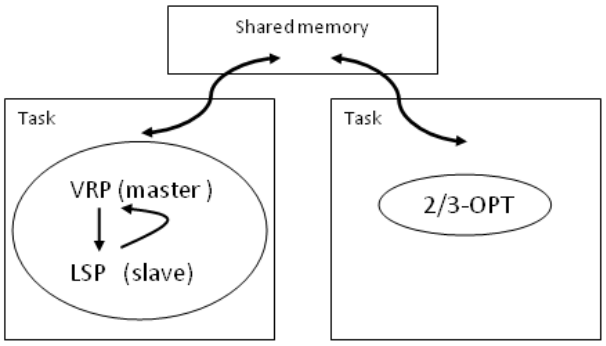


Figure 3: Master-slave PDP-solver with optimization task

IPDP with post-optimization

Instead of a decoupled master-slave approach, an integrated approach where both production and distribution is considered simultaneously can be considered. A separate asynchronous post optimization task can also be applied to improve the incumbent solutions in parallel with the main search.

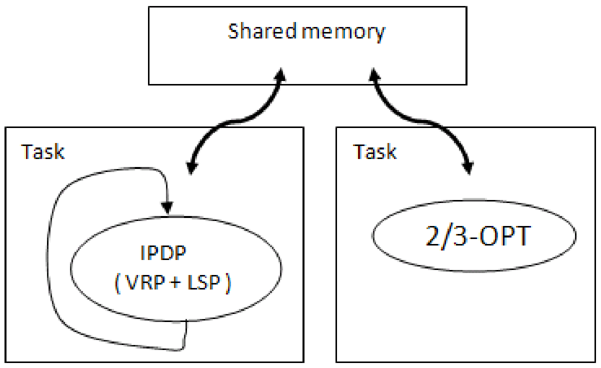


Figure 4: Integrated PDP-solver with optimization task

Split by problem size

The problem instances could be split in half by customer size, time periods or vehicles and run in parallel by a master-slave or integrated scheme. Partial solutions will have to be merged together in order to find the objective value for the instance as a whole. This is seen in figure 5. A problem that may arise when opting for this scheme is that when a problem instance is split each new part will be a new problem instance which does not have the full knowledge of the original problem instance. This may result in that even if each new part is solved to optimality, the merged result may not yield the optimal solution for the orginal instance.

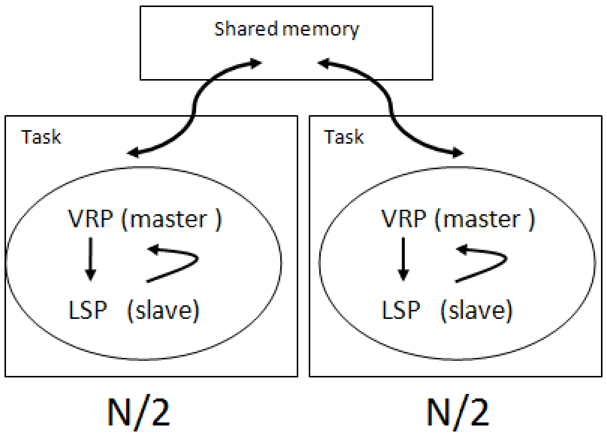


Figure 5: Split problem by customer

Asynchronous sub-problem solvers

One way of cooperation is to split the IPDP-solver into two solvers of each sub-problem LSP and VRP. These solvers should run asynchronously and cooperate to find the global IPDP optima. See figure 6. One issue with this scheme is the solver cooperation. Since both parts are dependent on the other for producing a solution for the IPDP, it can be a challenging task to get both solvers to run asynchronously. It is likely that an advanced communication process must be applied if the sub-problem should be able to guide each other towards the global IPDP optima in an asynchronous fashion.

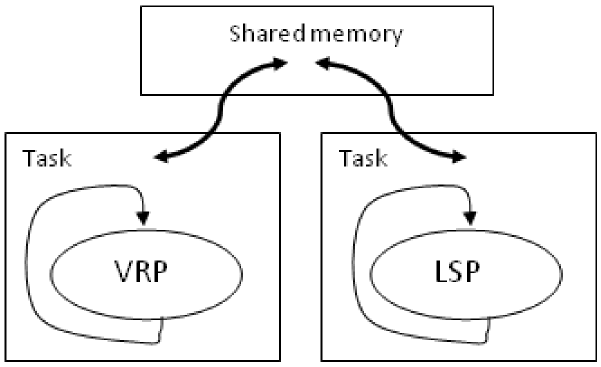


Figure 6: Running solvers in parallel

5 Mathematical model

This thesis uses Dr. Shiguemoto's mathematical model of the Integrated Production Distribution Problem since the tabu search parallization introduced here is an extension to his work.

The production-distribution problem (PDP) is defined on a complete graph $G = (W, A)$.

Sets:
W - set of nodes, $0, ..., N$
A - set of arcs, $(k, l) : k, l \in W, k \neq l$
J - set of products, $1, ..., J$
N - set of customers, $1, ..., N$
V - set of vehicles, $1, ..., V$
T - time periods, $1, ..., T$

Parameters:
p = capacity of the plant in time units
b_j = time required to produce one unit of product j
h_{j0} = unit inventory cost of product j at the plant (k = 0: plant)
h_{jk} = unit inventory cost for product j for customer $k, k \neq 0$
s_j = setup cost if product j is produced in period t
d_{jkt} = demand of product j for customer k in period t
l_{jk} = Lower bound on inventory for product j for customer k
u_{jk} = Upper bound on inventory for product j for customer k
f = fixed cost if vehicle v is used in period t

c_{jkl} = cost of transporting product j along arc (k,l)
m = large number, $\sum_{j=1}^{J}\sum_{t=1}^{T}\sum_{k=1}^{N} d_{jkt}$
c = capacity of each vehicle.

Variables:

P_{jt} = quantity of product j produced in period t;
I_{jkt} = inventory of product j of customer k at the end of period t;
$\delta_{jt} = \begin{cases} 1 & \text{if product } j \text{ is produced in period } t \\ 0 & \text{otherwise;} \end{cases}$
Q^v_{jkt} = quantity of product j delivered to customer k by vehicle v in period t;
X^v_{jklt} = quantity of product j transported on arc (k,l) by vehicle v in period t;
$\epsilon^v_{klt} = \begin{cases} 1 & \text{if vehicle } v \text{ travels along arc (k,l) in period } t \\ 0 & \text{otherwise;} \end{cases}$

Formulation:

$$min \sum_{t=1}^{T} \left\{ \sum_{j=1}^{J} \left[\sum_{k=0}^{N} h_{jk} I_{jkt} + s_y \delta_{jt} \right] + \sum_{v=1}^{V} \left[\sum_{l=1}^{N} f \epsilon^v_{0lt} + \sum_{l=0, k\neq l}^{N} c_{kl} \epsilon^v_{klt} \right] \right\} \tag{1}$$

subject to

$$P_{jt} + I_{j0,t-1} - I_{j0t} = \sum_{k=1}^{N}\sum_{v=1}^{V} Q^v_{jkt} \quad t = 1, ..., T; j = 1, ..., J \tag{2}$$

$$\sum_{v=1}^{V} Q^v_{jkt} + I_{jk,t-1} - I_{jkt} = d_{jkt} \quad t = 1, ..., T; j = 1, ..., J; k = 1, ..., N \tag{3}$$

$$\sum_{j=1}^{J} b_j P_{jt} \leq p \quad t = 1, ..., T \tag{4}$$

$$P_{jt} \leq m\delta_{jt} \qquad t = 1, ..., T; j = 1, ..., J \tag{5}$$

$$\sum_{\substack{i=0 \\ i\neq k}}^{N} X^v_{jikt} - \sum_{\substack{m=0 \\ m\neq k}}^{N} X^v_{jkmt} = Q^v_{jkt} \quad t = 1, ..., T; v = 1, ..., V; j = 1, ..., J; k = 1, ..., N \tag{6}$$

$$\sum_{i=1}^{N}\sum_{v=1}^{V} X^v_{ji0t} - \sum_{m=1}^{N}\sum_{v=1}^{V} X^v_{j0mt} = -\sum_{k=1}^{N}\sum_{v=1}^{V} Q^v_{jkt} \quad t = 1, ..., T; j = 1, ..., J \tag{7}$$

$$\sum_{j=1}^{J} X^v_{jklt} \leq c\epsilon^v_{klt} \quad t = 1, ..., T; v = 1, ..., V; k, l = 0, ..., N; k \neq l \tag{8}$$

$$\sum_{l=1}^{N} \epsilon^v_{0kt} \leq 1 \quad t = 1, ..., T; v = 1, ..., V \tag{9}$$

$$\sum_{\substack{i=0 \\ i \neq k}}^{N} \epsilon^{v}_{ikt} - \sum_{\substack{m=0 \\ m \neq k}}^{N} \epsilon^{v}_{kmt} = 0 \quad t = 1,...,T; v = 1,...,V; k = 1,...,N \tag{10}$$

$$l_{jk} \leq I_{jkt} \leq u_{jk}, P_{jt} \geq 0, Q^{v}_{jkt} \geq 0, X^{v}_{jklt} \geq 0, \delta_{jt} \in 0,1, \epsilon^{v}_{klt} \in 0,1, \forall j,k,l,t \tag{11}$$

The objective function (1) expresses the minimization of the setup costs, inventory costs at the plant and customers, and transportation costs. Constraint (2) represent the balance among production, inventory and deliveries at the plant, and constraints (3) ensures that the customer demand is fulfilled each period. Constraints (4) limit the production at the plant by the given capacity. Constraints (5) ensure that a setup cost is incurred only if there is production. Constraints (6) and (7) express the commodity conservation flow at the customers and at the plant. Constraints (8) represent the limited vehicle capacity. Constraints (9) impose that, in each period, at most one trip can be made by each vehicle. Constraints (10) ensure that each vehicle returns to the plant at the end of the route. Constraints (11) indicate the type of variables. Inventory levels have a lower and an upper bound.
Therefore, the PDP involves the following decisions: when to produce each product, when to visit the customers, how much of each product to deliver, and the determination of the vehicle routes.

6 Implementation

In this chapter a detailed description of the implemented parallelized tabu search procedure for the IPDP is given along with the chosen parallelization scheme and a conceptual model for the solver.

6.1 Tabu search

The implemented tabu search procedure is based on Dr. Shiguemoto's description of his tabu search for the IPDP, but some improvements in the search procedure have been implemented. The improvements and differences described in this thesis compared to Dr. Shiguemoto's description is stated in the method description. The tabu search procedure described in this thesis also features a parallelized neighborhood exploration and parallelized post-optimization procedures.

6.1.1 Initial solution

An initial feasible solution for the IPDP is created in the three following steps.

Step 1. Utilize custumers initial inventory

Use initial inventory at the customers to fulfill daily demand, and calculate inventory for each successive period that is served by the initial inventory. The first period any customer can not fulfill their demand from the initial inventory without violating the minimum inventory level, will be the first period with delivery.

Step 2. Creation of routes in delivery periods

The initial routes for the periods where there is need for delivery to customers are created by applying the savings construction heuristic. The created routes are feasible with respect to vehicle capacity, and the delivered amount is equal to $d_{jkt} - I_{jkt-1}$ if customer inventory in period $t-1$ is greater than the minimum inventory bound, d_{jkt} otherwise.

Step 3. Determine a production plan for the distribution plan

The production plan is determined by Evan's efficient implementation of the Wagner-Whitin heuristic. The Evan's heuristic is provided by the customers' total demand for each period with delivery and then determines the production periods, production quantities and the plant inventroy for all time periods.

6.1.2 Objective function

The objective function minimizes the total cost of production, transportation and holding cost at the customers and at the plant.

6.1.3 Neighborhood

The neighborhood is the different solutions from all combinations of customers, products, and time periods that can be reached in one move. The size of the neighborhood is stated by the equation $N * J * T^2$. This differs slightly to Dr. Shiguemoto's description where the neighborhood size is equal to $N * J * T * T - 1$, and results in a larger neighborhood where moves within the same time period is allowed.

6.1.4 Move

In similarity with the construction procedure a move consists of a 3 step procedure.

Step 1, determine the shift quantity $r_{jkt,t'}$

A move is defined as taking the maximum quantity $r_{jkt,t'}$ of a product j to be delivered to a customer k and move it from a period t to another period t'. This must be done without violating upper and lower bounds on the customer's inventory levels and the customer demand must be fulfilled. If the selected period t' is lower than period t, the move will result in a customer inventory increase by the quantity $r_{jkt,t'}$ for all periods t to t'. Similar if the period t' is a later period than period t, then there is a decrease of quantity $r_{jkt,t'}$ in the customer inventory for all periods t to t'.

Step 2, determine the cheapest insertion position

When a shift quantity $r_{jkt,t'}$ is determined, the cheapest possible insertion position in period t' must be determined. If there are no open routes in period t' a route to customer k is opened. In situations where there exists one or more routes in period t', the costs of the following possibilities must be calculated, and the one that yields the lowest cost is chosen.

1. If there exists one or more routes that are visiting customer k in period t', the cost of adding the shift quantity $r_{jkt,t'}$ to the vehicles is calculated.

2. If there are routes in period t' but customer k is not visited, the cost of insertion of customer k in every possible position in the existing routes are calculated.

3. The cost of opening a new route to customer k in period t'.

   In Dr. Shiguemoto's description possibility 1. is always chosen regardless of vehicle capacity and cost if there exists routes visiting customer k in period t'. This differs from the implementaion in this thesis, where all the possibilities above always are evaluated.

Step 3, determine a new production plan

When both the shift quantity $r_{jkt,t'}$ and an insertion position is determined, a new production plan must be determined due to the changes in the delivery quantities in periods t and t'. This production plan is determined by applying the Evan's heuristic in a similar fashion as in the construction procedure.
The combined move produced by the 3 step procedure above is calculated for all customers $k \in N$, all products $j \in J$ from all periods $t \in T$ to all periods $t' \in T$ and the move that produces the lowest cost is always chosen.

6.1.5 Move evaluation

The move that will be chosen is the move that gives the lowest total cost. The cost includes inventory holding cost at the plant and at the customers, setup cost at the plant, and travel cost for the vehicles. Let function $c(s)$ be the objective function (1). The function $f(s) = c(s) + \alpha g(s) + \beta(s)$ is the move evaluation function that incorporates the penalties of infeasible routes with respect to the vehicle capacity and a diversification penalty for moves with a high transition frequency. The transition frequency is the number of times a move has been performed.

Infeasibilities

The delivery quantity exceeding the vehicle maximum load capacity will be multiplied with a parameter α and added to the move evaluation function as a penalty. The function $g(s) = max(0, \sum_{v=1}^{V} \sum_{t=1}^{T} \sum_{k=1}^{N} \sum_{j=1}^{J} X_{jklt}^{v} - C\epsilon_{klt}^{v})$ denotes the total number of items exceeding vehicle capacity. When the solutions found in the last 5 iterations are infeasible, the α value is set to be $\alpha = \alpha * 3$, and similar if the solutions the last 5 iterations has been feasible the α value is set to $\alpha = \alpha/3$. This will encourage the search to oscillate between feasible and infeasible solutions. This is regarded to be beneficial since the best solutions are often found in the boundary between feasible and infeasible regions in the search space.

Tabu status

In order to be able to avoid getting stuck in local optima, a short term memory structure is applied by stating that when a move is performed, any reversing move is considered tabu for a number of iterations. More precisely, when a move of a product j to a period t' is performed, any move containing the product j from period t' is considered tabu for `tabu tenure` iterations.

Diversification process

In order to diversify the search and avoid that the search starts looping over the same set of solutions, a long term memory is applied to the tabu search procedure. Each time a delivery of any quantity of product j is moved from a customer k in a period t to another period t', a move counter for customer k, product j and time period t' is incremented by one and stored in a vector. This move counter is called the transition frequency ω_{jkt}. A multiplicator variable λ is used to control the degree of diversification, and the incumbent solution $f(s)$ is used in the scaling factor in order to incorporate solution variation to the diversification process.
When a new solution $f(\overline{s})$ is considered, the solution is penalized by the formula $\beta(\overline{s}) = \lambda\omega_{jkt}\sqrt{NJTV}\,[f(\overline{s}) - f(s)]$ and the penalty is added to the objective value. If the solution $f(\overline{s})$ is a new best solution, then $\beta(\overline{s}) = 0$. The added penalty for solutions obtained by frequently performed moves, will be less attractive over time compared to moves performed less frequently. The scaling factor $\sqrt{NJTV}\,[f(\overline{s}) - f(s)]$ is adapted from Taillard (1993) from his paper about the vehicle routing problem, and ensures that the diversification factor is proportional to the problem size ($NJTV$).

Aspiration criterion

If a new best solution will be obtained by performing a move that is considered tabu, the tabu status will be disrespected and the move will be performed.

Stop criteria

The tabu search procedure terminates when it reaches the maximum iteration number Θ or when η iterations have elapsed without finding a new incumbent solution. Θ and η are parameters supplied to the search procedure and controls the search duration.

6.2 Parallelization scheme

This implementation of tabu search for the Integrated Distribution and Production Problem features a parallelized neighborhood exploration, which in each move considers both distribution and production at the same time. According to Dr. Shiguemoto an integrated approach is far superior over a decoupled approach where one plan is determined before another. The IPDP solver described in this paper also feature a post optimization procedure, which run in parallel with the main tabu search procedure. This post optimization procedure is a two-opt improvement heuristic for the VRP. This two-opt procedure collects the incumbent solution from the tabu search when it is updated, optimizes the routes, and returns the improved set of routes to the tabu search procedure.

6.2.1 Conceptual model for the parallelized IPDP solver

As seen from figure 7 the parallelized parts of the IPDP solver is chosen to be 1) The tabu search procedure for the IPDP with parallelized neighborhood exploration, and 2) A post-optimizer for the incumbent routes. A parallel implementation of the Evans heuristic, was also implemented, but was abandoned because it gave no speedup compared to the serial implementation. The post-optimizer is chosen to be a 2-OPT intra route improvement heuristic for the VRP. The post-optimizer task starts up as an asynchronous task along with the IPDP task. The post-optimizer blocks until a route has been pushed onto a queue. The blocking mode means that the task waits until there is something to optimize. It always runs in the background. When the task is in blocking mode it does not use any CPU.

There are 3 steps in the figure. The first step creates the objects that are used as shared memory, and starts the two tasks. The second step shows how the communication works. When the IPDP solver finds a feasible solution and it is set as the new best, the routes are added to a queue. When routes are added, the 2-OPT task will remove these routes from the queue and start to optimize them. After the optimization is done, the routes are added to another queue, and the task goes back to the waiting state. After the IPDP solver has added the routes to the queue, it continues the tabu search to find a new best move. After the new move is applied, the second queue is checked to see if it contains any routes. If there are any routes present, the IPDP task will take the routes out and apply them to the incumbent routes and the current routes. The search and optimization continues until the stopping criterion is met. Step 3 prints out the solution information. Since these two tasks run asynchronously they must be synchronized before any attempt to combine the results. This must be done to make sure that the tabu search procedure has not updated the incumbent solution while the post-optimizer ran. If the incumbent solution still remains the same, the improved routes from the post-optimizer is merged with the incumbent solution. The optimized routes are also added to the current solution, if the routes in the current solution have not been altered. If routes have been altered since optimization started, they could make the customers inventory and deliveries infeasible. The reason the solver only merges the optimized routes if the incumbent solution has not changed, is that there are many situations where it can go wrong. If a route has been added or removed since the optimizer started, the replacement of a route could be very bad, because it would be replacing the wrong route. This also applies if a route has been altered, a customer delivery could be removed from the route if replaced. It is important to notice that both the neighborhood exploration task and the post-optimizer task can utilize all available CPU-cores in the system. A task scheduler provided in Intel Threading Building Blocks is responsible for assigning CPU-cores to the tasks.

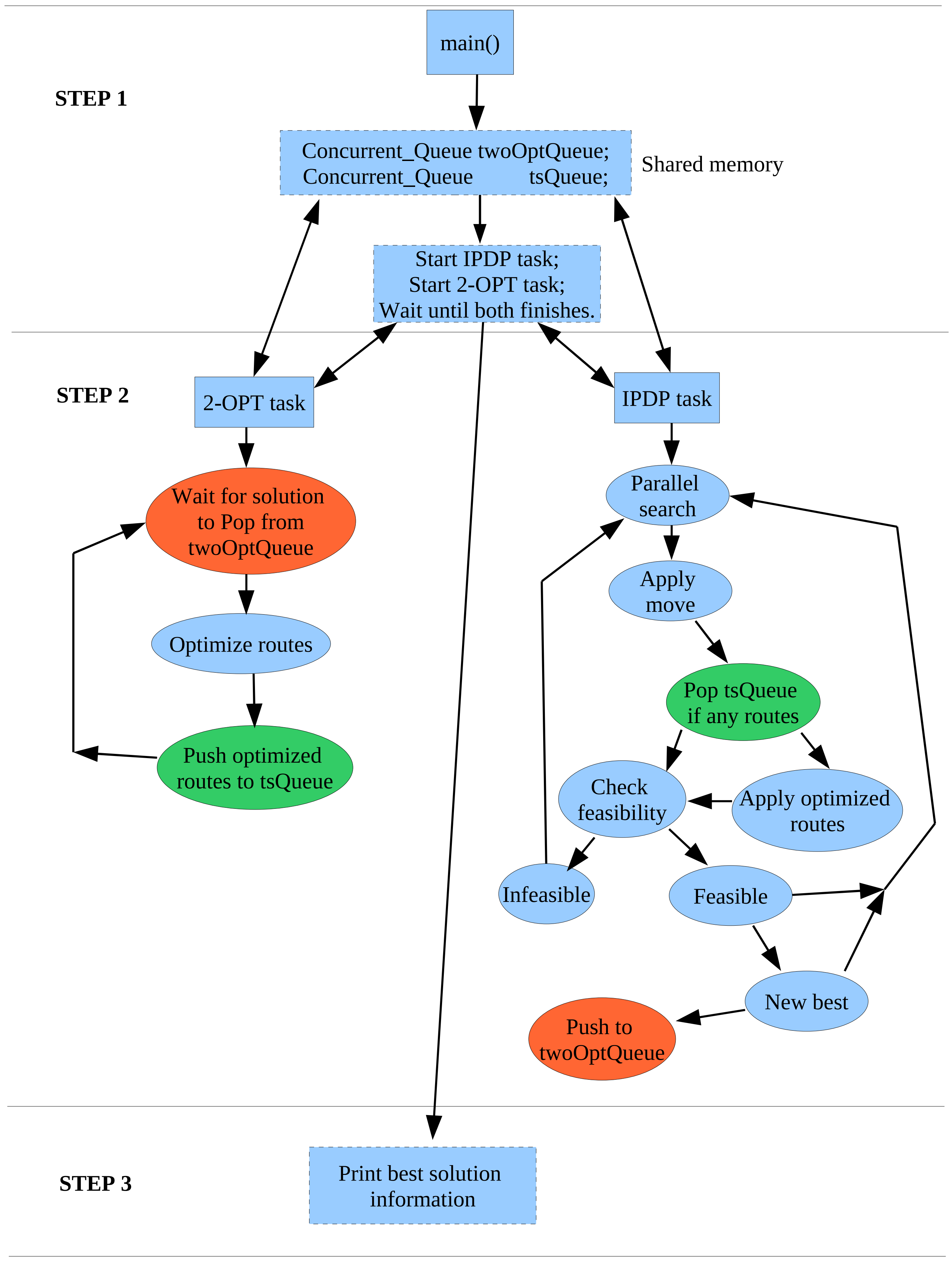


Figure 7: Current solver scheme

7 Computational experiments

7.1 Test Instances

Test instances used in this paper consists of two sets of single item instances generated by Bertazzi et al. (2005). Both sets contains 96 instances divided into four classes. The first set has $T = 6$ time periods, and $N = 50,75,100,125,150$ customers. The second set have $T = 30$ time periods and $N = 50$ customers. The different classes have different costs for the given parameters to emulate different situations in real world examples: unitary production cost c^p, fixed production cost per period f^p, inventory holding cost at customer k h^k and fixed transportation cost per period f^v. In class 1 these parameters are set to $h_0 = 3$, $c^p = 10h_0$, $f^p = 100c^p$ and $f^v = 32181$. For evaluation of situations where there are no fixed transportation costs specified, class 2 has fixed transportation cost $f^v = 0$. To see what impact on the solution there will be if the production cost is reduced compared to the inventory holding cost at the plant, the instances in class 3 have a unit production cost $c^p = h_0$. In the last class the customer inventory holding cost $h_k, k = 1, ..., N$ are set to zero. This will emulate situations where the plant does not have to pay for customer inventory holding costs.
These instances have been applied by both Bertazzi and Shiguemoto in their papers about the IPDP. Bertazzi used the more constrained VMI-OU and VMI-FFD strategies in his paper, while Shiguemoto used the less constrained VMI-ML strategy to measure the cost reduction this strategy gives over the results reported by Bertazzi. As mentioned earlier the tests performed in this thesis uses the VMI-ML strategy which give a god basis for result comparizon against results reported by Shiguemoto.

Computers used for testing

Intel Core 2 Duo , 3 Ghz
Intel Quad i7 920, 3.66 Ghz
MacBook Pro 2.2 Ghz Core 2 Duo

7.2 Parameter testing

Parameters for the solver presented in this thesis must be split in two parts, the first regarding parameters for the parallelization and the other regarding parameters for the tabu search procedure.

Parallelization parameters

The Intel TBB library supplies methods for automatic creation of threads that will utilize all the computational power available in multi-core or multi-processor system.
Parameters to these methods must be an object and the range of the object to divide into threads, usually start and end indices. The splitting into threads can occur over 1, 2 or 3 dimensions, corresponding to the number of dimensions in a matrix.
Splitting does not have to be based on the dimensions of the matrix that is passed to the function. A matrix with a single dimension can be passed to the function, and the second dimension can be a loop where some calculations are performed.
Splitting over one and two dimensions have been tested. The testing shows that splitting over more than one dimension increased the running time of the solver.
This indicates that the CPUs and cores are already fully utilized with splitting over a single dimension, and the overhead of splitting over more dimensions increases running time.

Grain size

The grain size parameter specify the amount of work that should be in each thread. If the iteration space has more work than the grain size, the work is split into several threads, each having a work load equal to the grain size.
Each thread requires a small portion of parallel scheduling work, known as parallel scheduling overhead. If the work load for each thread is too small, the amount of scheduling overhead can become large, and limit the performance of the algorithm. Similar if the grain size is set too high the work will be executed serially in one thread, and no parallelization will occur. A recommended rule of thumb is that the grain size should be set such that each thread should have at least 10 - 100,000 instructions to execute. If one should run through a loop 100 times, and in each round it executes 5,000 instructions, the total amount of work will be $100 * 5000 = 500,000$. This means that any grain size in the range $[2, 20]$ will follow this rule and give a work load in the range 10 - 100,000 instructions per thread. A grain size of 1 would give 100 threads with a work load of 5,000 instructions each, and a grain size of 100 would give one thread with a massive 500,000 instructions.
The Intel Threading Building Blocks library supply objects for automatic grain size detection, called the auto_ partitioner and simple_partitioner object. Intel strongly recommend that an object for automatic grain size detection is used, so that the

application has a better chance to deal with variable work loads and problem sizes. In order to test the efficiency of these auto partitioner objects, a range of different static grain sizes was tested against the two mentioned auto partitioner objects. The test was executed on test instance "instance1_50_30.dat" with a maximum iteration limit of 10 iterations. The parallel neighborhood exploration procedure in the tabu search algorithm has a one dimensional iteration space, with size equal to the number of VRP -routes in the current solution.
The routes in the iteration space are the routes that needs to be examined in order to determine a shift quantity to move to another route. For each route in the iteration space, these calculations must be done: 1) determine a shift quantity for each customer in the route for each time period. 2) For each shift quantity for each customer, all possible insertion possibilities must be determined for the chosen period t', including the possibility of creating a new route. 3) For each possible move a production plan must be determined.
It is therefore safe to say that there is a good portion of work to do for each iteration in the iteration space.
Threads created in the parallel neighborhood exploration procedure, must be merged together from right to left in order to find the best overall move with the lowest objective value.

Table 2: Grain size parameter

Grain size	No. threads	Time consumption (sec)
1	81	6.3
5	18	5.95
10	10	5.6
15	8	5.55
20	6	5.55
25 - 45	4	7.15
50	2	7.4
100	1	9.6
auto/simple	11	5.6

Table 2 shows the number of threads that is executed, and the runtime with respect to the given grain size. The number of routes in the current solution in the test was 81. The fastest execution of the predetermined 10 iterations was done with a grain size of 15 and 20, which meant that the search for the best move was split into 8 and 6 threads divided onto the 2 cpu-cores available on the test computer. With a smaller grain size of 5 and 1 the execution time went up, which means that the amount of parallel overhead is increasing in comparison to the original work load. In similar fashion when the grain size is larger than 20, the work is split in too few threads with too large work load, and the result is lost performance. The last row in table 2 is the result obtained when the grain size is determined with an auto partitioner object, and as seen, the performance from this auto partitioner object is quite good compared to the optimally tuned grain size for this particular problem size. The auto partitioner works quite well, and since it can determine a good value for the grain size for test instances with various problem sizes and computers with different processors, it seems that Intel's advice for using the auto partitioner objects is well worth to follow.

Tabu search parameters

Different values of the tabu tenure, penalty variables and diversification variables have been tested.

Tabu tenure

The range of tabu tenure values for the different test instances was obtained by the expression used in Shiguemoto's thesis: $\left[\lceil\sqrt{NTJV}/3\sqrt{N}\rceil, \lceil\sqrt{NTJV}/\sqrt{N}\rceil\right]$ This expression gives a range with uniform distribution that depends on the problem size. A tabu tenure range of 2,3 and 4 was tested. The average results of the testing is shown in table 3. These tests were run with GNU GCC in the Ubuntu linux distro. The columns that describes time usage is shown in seconds from wall clock time, and is based on the total running time of the solver. The test gave no evidence showing that one value is better than the others, because there was no clear connection between problem size and tabu tenure.

Table 3: Average results for each tabu tenure.

# Customers	TT	Avg Time	Avg Cost
50	2	13.2	252901
50	3	14	251408
50	4	14.9	**247597**
75	2	27.9	391424
75	3	27.9	**376354**

75	4	27.4	377587
100	2	40.6	536825
100	3	39.6	**524664**
100	4	40.9	525435
125	2	56.6	653458
125	3	60.1	651597
125	4	61.3	**628065**
150	2	84	**845080**
150	3	88.7	850166
150	4	85	845423

For the test instances with thirty time periods, no parameter tests weere performed due to time limitations. The time limitations was due to some last minute changes in the solver code. And since the tabu tenure parameter testing for the instances with six time periods were inconclusive the importance of performing such a parameter test for these instances seemed low. The tabu tenure value 10 was chosen from the uniform range of values given by the expression stated earlier in this chapter, and was applied for the instances with thirty time periods.

Penalty and diversification variables

The initial alfa value was set to 1.0 and for every 5 iterations the routes are infeasible, the α value will increase by a factor of 3. Similarly, for every 5 iterations the routes are feasible the α value will decrease by a factor of 3. This strategy seems to work very well and produce a good mix of feasible and infeasible solutions.
The λ parameter contributes to the decision of which penalty that should be the most decisive penalty. If λ is high, the penalty for infeasible routes easily gets overruled by the increasingly dominating long term penalty. This usually gives poor results since some routes are likely to be infeasible, and feasible solutions are harder to find. Another problem that can arise when λ is set high is that the penalty added to the objective value becomes larger than the double max value. This will if not handled in a proper way, cause the solver to crash. Similarly, if the λ value is too low, the effect of the diversification will be close to none. Based on these observations and small scale testing, the initial λ value of 0.15 was chosen. If there has been 10000 iterations without finding a

new best, the λ value is increased by 0.001. When a new best solution is found the lambda parameter is set to initial value.
To prevent that the long term diversification penalty getting too dominant, a strategy for resetting the diversification penalty was tried. The diversification matrix was reset when a number of iterations had passed without finding a new best solution. This was done for two reasons, 1) prevent the diversification penalty to become too large, and 2) give promising but highly penalized moves a new chance. Testing showed that this did not yield the wanted results and the diversification resetting was abandoned.

7.3 Results

The results from the parallelized IPDP-solver is divided into two separate parts. Results regarding task parallelization and CPU-scalability is discussed and interpreted in the first part. In part two, the results obtained by the solver is compared against the results presented by Dr. Shiguemoto in his article about the IPDP. The results were obtained by using the VMIR-ML strategy.

7.3.1 Scalability

Tests show that the parallel version of the IPDP solver is able to utilize several CPU-cores effectively. The gain in decreased wall-clock time is well worth the effort of implementing a parallel solver. Solutions can be found faster or more work can be done in the same amount of wall clock time compared to a serial version of the IPDP solver. Figures 8, 9 and 10 shows the scaling when more than one core is utilized in a processor. This chapter uses terms like *thread* and *hardware-thread.* They are not the same. Normally there is one hardware-thread per core or per processor. Each processor has a given amount of hardware-threads that are allowed to be run at the same time in a core. An operating system has hundreds of threads waiting to be executed. While one thread waits for data to be retrieved, another thread can use the processor. This can sometimes reduce the running time of an application. Tests were performed with the new Intel Core I7 processor. This processor has 4 cores with support for running 2 hardware-threads in each core, called Hyper-Threading(HT). Enabling or disabling these hardware-threads will show if the implementation of parallel-enabled code gives any improvement on the running time. Different problem sizes were used to see if the benefit of running parallel solvers grow with the problem size.
Figures 8, 9 and 10 shows time usage when hardware-threads from one through eight are enabled. If the number of running threads are less or equal than the number of cores or processors, there will be one thread executed in each core or processor. The operating system's scheduler handles this. When 4 hardware-threads are enabled, one thread is executed at each core (with this processor). The figures show wall clock time usage in percent to make it easier to compare the results.

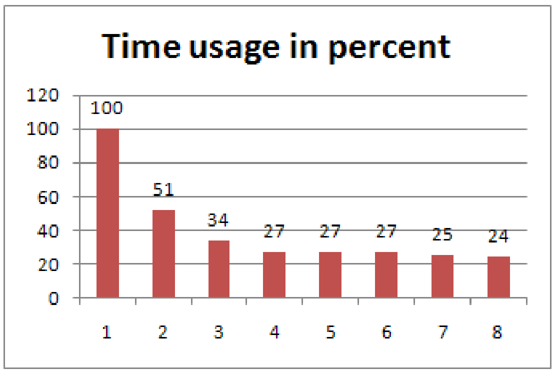


Figure 8: 50 customers, 6 time periods

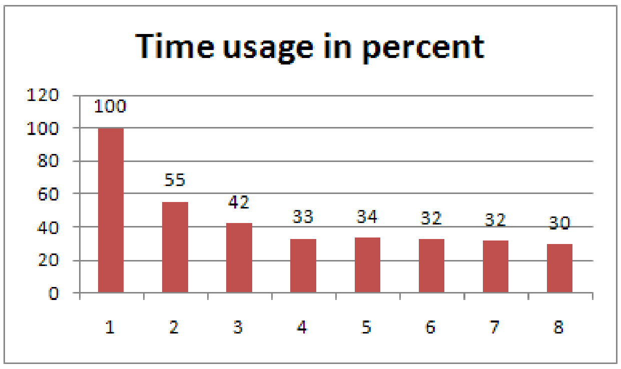


Figure 9: 150 customers, 6 time periods

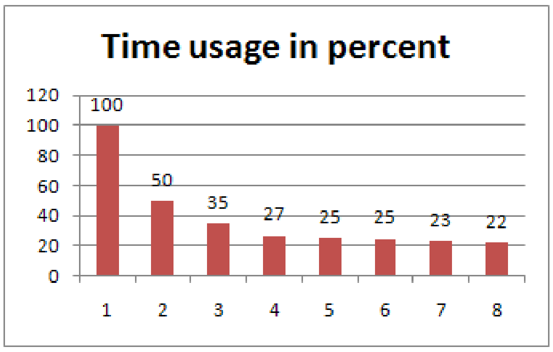


Figure 10: 50 customers, 30 time periods

The largest scaling improvement can be seen when the number of enabled hardware-threads are increased from 1 to 2, 3 or 4. The time usage is reduced nearly 50% on all three problem sizes when the hardware-threads are changed from 1 to 2, and another 50% when the number of hardware-threads are changed from 2 to 4. The results shows good scaling improvement when the number of hardware-threads are changed from 1 to 4, with a average reduction in wall clock runtime about 75% compared to the sequentional one thread execution. The test with 150 customers and 6 time periods show a slightly reduced scalability compared to the instances with less customers. One of the reasons for this is due to the increased amount of serial work, such as feasibility checks and inventory calculations, needed because of the increased number of customers. The test of the instances with thirty time periods shows the same amout of scalability as the instances with six time periods and 50 customers, which indicates that the amount of serial work compared to parallel work is only sensitive to the number of customers not number of time periods.
The test with 8 hardware-threads enabled showed a very small decrease in time usage, but since there is a decrease and not increase in time usage, it suggests that running with two hardware-threads in each core works better than using only one. This processor running with 8 threads can not be compared to an 8-core processor or dual quad-core processors. The gain with running 8 hardware-threads instead of 4 is about 5% which is not very much compared to running 8 cores with one hardware-thread enabled in each core.

7.3.2 Solution quality

The results obtained from the parallel IPDP solver presented in this thesis are compared to the results presented by Dr. Shiguemoto. For the instances with six time periods, the results obtained are quite good. Since Shiguemoto presented only average results based on problem size and class, the results presented in this section will also be based on average results in order to compare the results obtained. For more detailed results we refer to appendix A.4 which contains test results for each individual test instance. As seen from table 4 the reduction in the objective value obtained ranges from 13.3% to 36.9%, and interestingly the reduction in objective value increased with the increase in problem size. The average reduction in wall clock time spent to achieve these results ranging from 94% on the instances with 50 customers to 83.3% for the instances with 150 customers.

Table 4: Average solutions with 6 time periods

N	Time horizon	Cost, Shiguemoto	Cost, This thesis	% Cost Reduction	Time, Shiguemoto	Time, This thesis	% Time Reduction
50	6	288942.52	250635.3	13.3	235.39	14.03	94.0
75	6	483985.26	381788.3	21.1	267.69	27.70	89.6
100	6	670958.28	528974.7	21.2	302.76	40.36	86.6
125	6	905475.31	644373.3	28.8	396.52	59.3	85.0
150	6	1341683.49	846889.7	36.9	513.21	85.9	83.3

For the set of test instances with thirty time periods, the time available for testing was very limited. As a result of this only one run of each test instance with only one set of parameters were tested. Instance 1 through 62 is run with a maximum iteration count of 100000 as stopping criteria. Instance 63 through 96 is run with a maximum iteration count of 20000 as stopping criteria. The average results based on class are presented in table 5 along with comparison of the results obtained by Dr. Shiguemoto. The average objective value obtained increased in the range of 68 - 394%, which means that the solution quality obtained on this set of instances was quite the opposite to the solution quality obtained on the instances with six time periods. It must be stressed that these results were obtained in only one run for each test instance, and the possibility for obtaining better results with more testing time should be great.

Table 5: Average solutions with 30 time periods

Instance	Time horizon	Cost, Shiguemoto	Cost, This thesis	% Cost Increase	Time, Shiguemoto	Time, This thesis	% Time Increase
1-24	30	1005561.73	2719086.46	170	732.5	1348.08	84
25-48	30	853992.30	1433292.71	68	732.5	1285.10	76
49-72	30	430232.99	1694823.00	394	732.5	1182.98	62
73-96	30	871478.11	2677775.00	307	732.5	350.12	-52

Post-optimization

Table 6 shows a sample comparison of time, iteration count and solution quality when optimization is enabled and disabled. There is no clear evidence to suggest that the

2-OPT task gives better results than using no optimization. When the 2-OPT task is enabled the solution is sometimes worse than when there is no optimization enabled. This is because of the stopping criterion that makes the solver stop when there are no better solutions found after an amount of iterations. The search trajectory is also changed when the post-optimized routes are used, which can lead the search in to less promising parts of the search space.

Table 6: Solution comparisons

Instance	**2-OPT**	**Iteration**	**Time**	**no OPT**	**Iteration**	**Time**
instance49_50_6	154109	20095	62	160016	3960	41
instance49_75_6	263312	1517	87	227174	8095	95
instance4_100_6	695456	1117	132	629917	1139	124
instance4_125_6	766533	5556	210	834273	758	178
instance4_150_6	1069761	2351	314	1051424	12137	345

8 Conclusion and future work

The results obtained by the parallel IPDP-solver are quite interesting. Compared to the results reported by Dr. Shiguemoto the solver presented in this thesis found very good solutions for the instances with six time horizons. The solutions had an average of 13 - 36% decrease in objective value and the solver used about 80 - 95% less computational time to obtain these results.
The instances with thirty time periods are very computational expensive, and due to late changes in the code of the IPDP solver, there was only time for one run per test instance. This meant that only one value for each parameter could be tested. The results from this set of instances were not as good as the results reported by Dr. Shiguemoto. The average objective value obtained for each instance class increased in the range 68 - 394% compared to his results. It is hard to say what the reason for this is, but we believe that our IPDP solver could achieve far better results if more testing had been done. Another factor that might contribute to this result is that our method differs from his method by allowing for moves within the same period, and only feasible start solutions are allowed. Still we are reasonably satisfied with the results obtained.
The parallelization of the neighborhood procedure resulted in a large reduction of computational time compared to a serial implementation. The solver scales very well over the tested number of cpu-cores available on the test systems. On a dual-core system the parallel implementation was about twice as fast as the serial implementation. On a quad-core system the reduction in wall clock time was 75% compared to a sequential implementation, and almost twice as fast as a dual-core system. Based on these observations, the conclusion is that the parallelization was very successful and worth the extra effort.
The parallel post optimization did not yield the desired results. The results varied a lot, sometimes the solver gave better results with the post optimization enabled, sometimes it gave worse results. There were no indications that the results are better when the post optimization task is activated, than when it is deactivated. The post optimization task also have a negative effect on the computational time needed for the tabu search, since the post optimizer task uses computational power that otherwise could have been used for the tabu search.
Still we think that task parallelization might be a good strategy if used differently. A very interesting strategy is to have several tabu search tasks with different search strategies to be able to search in different areas of the search space in parallel.

Another interesting strategy is to utilize the large computational capacity of the modern graphics cards in modern computers. Nvidia CUDA(CUDA) is a framework for using the GPU of Nvidia graphics cards for general purpose computing. The computational power in new graphics cards are superior to ordinary CPUs in a computer, and can deliver up to 5 - 600 times speed up on certain algorithms. This makes CUDA a very interesting framework to use for the neighborhood exploration procedure in the tabu search.

References

Amdahl, G. M. (1967, April). Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIP Conference Proceedings*, Volume 30, Reston, VA, pp. 483–485. AFIPS Press.

Archetti, C., L. Bertazzi, G. Laporte, and M. G. Speranza (2007). A branch-and-cut algorithm for a vendor-managed inventory-routing problem. *Transportation Sience 41*, 382–391.

Armentano, V. and A. Shiguemoto (2009, Apr). A tabu search procedure for coordinating production, inventory and distribution routing problems.

Bertazzi, L., G. Paletta, and M. Speranza (2005). Minimizing the total cost in an integrated vendor-managed inventory system. *Journal of Heuristics 11*, 393–419.

Boudia, M., M. Louly, and C. Prins (2007). A reactive grasp and path relinking for a combined production-distribution problem. *Computers & Operations Research 34*, 3402–3419.

Boudia, M. and C. Prins (2009). A memetic algorithm with dynamic population management for an integrated production-distribution. *European Journal of Operational Research 195*, 703–715.

Bævre, O., B. Gjengstø, and A. Løkketangen (2008). Solving large multiple knapsack problems using tabu search. Available at: http://home.himolde.no/˜041072/NIK-2008-B-G-L.pdf.

Clarke, G. and J. W. Wright (1964, 8). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research 12*(4), 568–581.

Crainic, T. G., M. Toulouse, and M. Gendreau (1997). Towards a taxonomy of parallel tabu search algorithms. *INFORMS Journal on Computing 9(1)*, 61–72.

CUDA. Nvidia cuda. Available at: http://www.nvidia.com/object/cuda_home.html.

Dantzig, G. and J. Ramser (1959). The truck dispatching problem. *Management Science 6*, 80–91.

Evans, J. R. (1985). An efficient implementation of thw wagner-whitin algorithm for dynamic lot-sizing. *Journal of Operational Management 5*, 229–235.

Gillet, E. and L. R. Miller (1974, 3). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research 22*(2), 340–349.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research 13*, 533–549.

Gustafson, J. L. (1988). Reevaluating amdahl's law. *Communications of the ACM 31(5)*, 532–533.

Holland, J. H. (1975). *Adaption in natural and artificial systems*. The University of Michigan Press, Ann Harbor, MI.

HT. Intel hyper-threading technology. Available at: http://www.intel.com/technology/platform-technology/hyper-threading/.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983, May). Optimization by simulated annealing. *Science, New Series 220*, 671–680.

Le Bouthilliera, A. and T. Crainic (2005). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research 32*, 1685–1708.

Taillard, E. (1993). Parallel iterative search methods for vehicle routing problems. *Networks 23*, 661–673.

TBB. Intel threading building blocks. Available at: http://www.threadingbuildingblocks.org/.

Wagner, H. M. and T. M. Whitin (1958). Dynamic version of the economic lot size model. *Management Science 5*, 89–96.

A Appendix

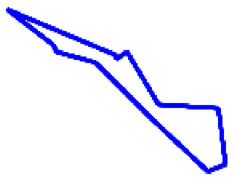
A.1 Visualization

A.1.1 SVG

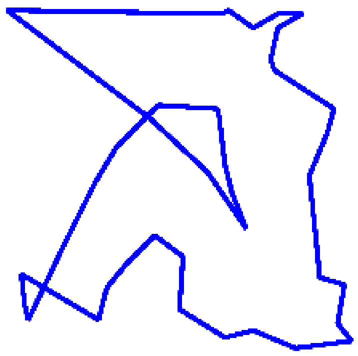
Scalable Vector Graphics (SVG) is a language for the creation of graphical objects. The objects are stored in XML format. SVG is used to represent the routes created from the VRP solver. The visualization process here will make it easier to see if the route created is a good route. It is easier to observe graphically rather than interpreting numbers in a sequence. A big advantage about these graphical objects, are that they can also be used by decision makers, not only for testing and debugging purposes.

Current SVG

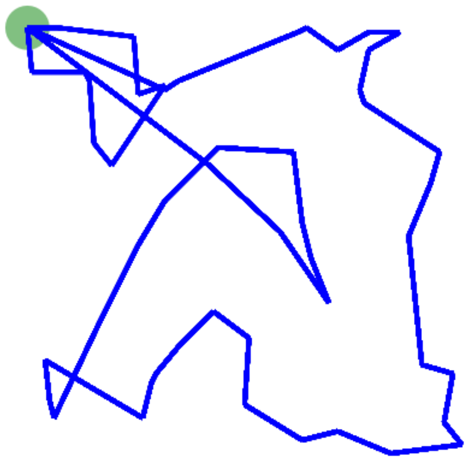
An SVG file is created for each route and each period and an HTML file is created with a select box (combobox/listbox) that contains the word "Route" and the number of the route (Route0, Route1). The user can select each item in the box in order to show the VRP route in a web browser. An example of how the routes looks is seen in figure 11 Figure 11(c) shows two routes from the same period. Here we can observe that the routes are crossing each other and themselves. Both intra-route and inter-route optimization could be applied here to make the solution better.



(a) Good VRP route



(b) Not so good VRP route



(c) two routes in one period. Overlapping.

Figure 11: VRP route examples

A.2 TBB implementation

From the TBB library, the tbb::parallel_reduce, tbb::parallel_scan, tbb::blocked_range and tbb::task classes have been used.

A.2.1 Parallel_reduce (used in VRP)

The theory in parallel_reduce is to split some work across several working threads. The variables passed to the constructor must also be passed to the split function (if needed), because the threads has its own memory space. When all the threads are finished, the results are joined together (compared), and the best result is returned. The splitting can be done on one, two or three-dimensional vectors/arrays. The current implementation uses only one-dimensional vectors, and the splitting is done on the routes. The customers must be examined individually, so it seemed like a natural place to split.
A new class is needed to create the parallel version. All the methods used in the serial version must be copied and adapted to the parallel loops within the new class. There is much to think about if the solver has to be able to reproduce the same results as the serial solver version. The threads that are created are not started and ended serially. The best solution found in a parallel_reduce loop therefore depends on the order in which the threads returns.
A simple `for` loop and a parallel version of the `for` loop is seen below. The `blocked_range` says where the for loop starts and ends.

```
for ( int i = 1; i < N; i++ ) {
    //
}
```

——————————-

```
void operator()( const blocked_range<int>& r)  {

    for ( int i = r.begin(); i < r.end(); i++ ) {
        //
    }
}
```

Things to beware of:

1.) When variables are passed as reference, do not alter any of its content, it reflects onto all the working threads. This is easy to see because you will get a different result each time the solver is run. The reason for this is that the threads created do not start or finish the same time each time you run the solver, and thus doesn't alter the reference variable the same place every time.
2.) Two parameters must be given when initializing the parallel_reduce function, start and end indices of the routes. If there are 50 routes, (1,50+1) must be given as parameters, because it uses an interval like this: [start,end).
3.) Because the threads does not end serially, care must be taken if the goal is to make reproducable results.

A.2.2 Task

To create a task you create a new class which inherits task. Existing classes does not need to be altered, they can be instantiated within the task. Tasks can be started different ways. A single task can be started and waited for until it finishes, or several tasks can be started at once and waited for until all tasks finishes before the solver can continue. Starting with the wait function is like calling a regular function, it does not return until the tasks are done running. There are also schemes where new tasks are spawned within existing tasks.
In the solvers created in this thesis, tasks were used the following way:
Two tasks are created as `root` tasks, the IPDP tabu search and the 2-OPT algorithm. A `task_list` is created, and the two tasks are inserted into the list. The list is then started with a waiting function, `spawn_root_and_wait( task_list_object );`. This function starts the tasks and wait until all of them are completed. Tasks complete when the tabu search has reached its stopping criteria. The two tasks were created as root tasks because only root tasks can be started simultaneously this way. The 2-OPT routine blocks until a route is sent to it. The information exchange is handled by sending two `concurrent_queue` objects as reference to each task; one that sends, and one that receives data. The messaging is simply done by popping and pushing objects into queues.

A.2.3 Parallel_scan

This class is used in the parallel implementation of the recursive Evans algorithm. Parallel scan computes a parallel prefix. This is an advanced concept in parallel computing, but must be applied if the task performed has inherently serial dependencies.

Examples of this are recursion and cumulative summation. The parallel prefix is an associative operation, for instance a sum operator. Parallel scan must use two passes through the whole loop in order to complete the scan, as a result of this it may do twice the amount of calculations compared to a serial implementation of the loop. This strongly limits the gain that is achievable by parallelizing this part of the algorithm, and parallel scan is therefore better suited for future systems with many cpu-cores.
The use of the parallel implementation of the Evans heuristic was therefore abandoned, since it provided no gain in comparison with the serial implementation.

A.3 Compiler differences

GNU gcc compiler runs 3.3 times faster than the Intel Compiler and 4.7 times faster than the VC++ compiler. Table 7 shows the actual wall clock time used. VC++ and Intel compilers were run on Windows Vista, and GCC was run with Ubuntu 9.04. GCC was not compiled on the Windows platform because Intel TBB only supports Visual Studio natively (yet).
The parameters for the test was 100000 iterations, tabu tenure 3, 8 enabled hardware-threads in the I7 processor. The instance used was "instance1_50_6".

Table 7: Time used with different compilers

Compiler	**Time used**
VC++	71
Intel	50
GCC	15

A.4 Computational results

The figures 12, 13 and 14 shows the real values (run-time) for the figures 8 , 9 and 10 that shows time decreases in percent. The values in the figures are seconds in wall clock time.

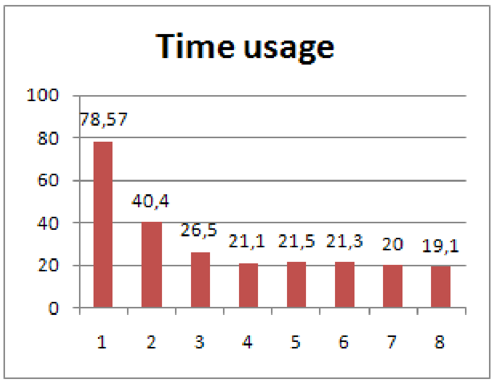


Figure 12: 50 customers, 6 time periods

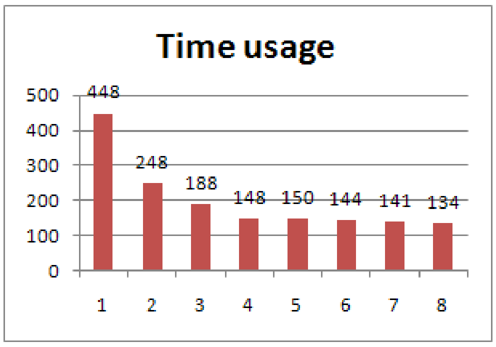


Figure 13: 150 customers, 6 time periods

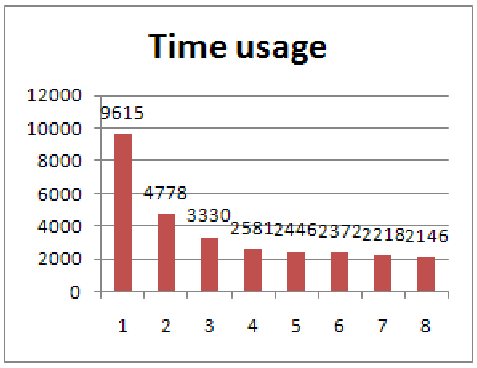


Figure 14: 50 customers, 30 time periods

Tables 8. 9. 10. 11 and 12 shows detailed information about the results for the 5 sets with time horizon 6. The first column describes the instance. the following columns describe time horizon. minimum. average and maximum solution cost and minimum and average time usage. A stopping criteria of 100000 iterations is given as a parameter to these tests.

Table 8: 50 customers, time horizon 6

Instance	**T**	**minCost**	**avgCost**	**maxCost**	**minTime**	**avgTime**
instance1_50_6	6	192599	208813	227849	12.7	16
instance2_50_6	6	227748	234504	243750	12.3	13.7
instance3_50_6	6	227443	231472	236358	15.37	16.4
instance4_50_6	6	314386	336558	348159	12.3	12.5
instance5_50_6	6	341297	342372	344178	10.6	11.0
instance6_50_6	6	343372	354483	376620	11.8	12.3
instance7_50_6	6	160907	182160	193907	10.9	11.1
instance8_50_6	6	160717	185343	198326	12.4	12.5
instance9_50_6	6	170468	190781	203676	12.3	12.6
instance10_50_6	6	278215	303402	317718	11.1	11.3
instance11_50_6	6	284805	284971	285279	11.1	11.2
instance12_50_6	6	286485	307646	320107	11.8	12.2
instance13_50_6	6	330066	349611	388072	11.6	14.5
instance14_50_6	6	276214	333367	393462	11.9	16.1
instance15_50_6	6	347375	348292	349130	13.8	16.7
instance16_50_6	6	384131	427114	451411	10.9	11.1

Table 8: 50 customers, time horizon 6

instance17_50_6	6	384971	434382	460888	11.1	11.1
instance18_50_6	6	465039	484168	520846	11.3	12.5
instance19_50_6	6	236011	302627	365492	10.4	10.4
instance20_50_6	6	237702	261237	302046	10.8	10.9
instance21_50_6	6	243492	289727	316568	11.2	11.9
instance22_50_6	6	358310	402673	426251	10.1	10.9
instance23_50_6	6	426990	427649	428387	10.3	10.5
instance24_50_6	6	362332	407875	432266	11.0	11.3
instance25_50_6	6	117970	117970	117970	11.2	11.4
instance26_50_6	6	119211	119211	119211	11.4	11.5
instance27_50_6	6	121151	121151	121151	12.2	12.4
instance28_50_6	6	240448	241775	242452	19.3	24.1
instance29_50_6	6	243707	244086	244378	13.9	21.0
instance30_50_6	6	244996	246058	247201	16.0	20.0
instance31_50_6	6	95826	96183	96565	13.0	16.3
instance32_50_6	6	97195	97370	97564	15.4	16.8
instance33_50_6	6	99678	99824	99994	13.8	17.1
instance34_50_6	6	213306	214004	214490	11.8	15.5
instance35_50_6	6	214566	215227	216122	10.2	14.4
instance36_50_6	6	219441	220253	221107	11.3	11.5
instance37_50_6	6	128578	128929	129406	10.9	12.6
instance38_50_6	6	131675	131992	132410	11.2	17.8
instance39_50_6	6	136972	137239	137520	11.6	15.2
instance40_50_6	6	248815	252193	254083	14.0	18.5
instance41_50_6	6	253693	255508	256885	12.7	17.8
instance42_50_6	6	260414	261112	261499	13.8	18.8
instance43_50_6	6	105271	105765	106076	11.2	14.6
instance44_50_6	6	107391	107890	108802	10.9	11.4
instance45_50_6	6	113485	114212	115081	16.9	17.6
instance46_50_6	6	223174	224329	226194	16.8	21.2
instance47_50_6	6	225162	226660	227547	11.4	16.2

Table 8: 50 customers, time horizon 6

instance48_50_6	6	232827	235552	240659	11.5	13.7
instance49_50_6	6	139917	154974	163047	11.7	16.0
instance50_50_6	6	135257	155911	169208	12.1	13.7
instance51_50_6	6	160933	173780	197878	11.9	13.4
instance52_50_6	6	153027	164924	171003	13.4	16.5
instance53_50_6	6	171853	181541	200501	11.7	12.9
instance54_50_6	6	145106	164359	174060	17.6	19.1
instance55_50_6	6	97552	120523	134157	11.7	14.3
instance56_50_6	6	134420	136033	137269	12.6	13.3
instance57_50_6	6	137271	146172	162388	12.3	12.6
instance58_50_6	6	139128	143352	147237	11.6	12.9
instance59_50_6	6	147879	149887	151851	11.9	12.2
instance60_50_6	6	149455	152227	154005	12.7	15.8
instance61_50_6	6	270062	271947	273467	12.6	13.6
instance62_50_6	6	273240	281695	294073	11.7	13.5
instance63_50_6	6	283398	286133	291438	13.6	15.4
instance64_50_6	6	283096	305033	346169	11.7	12.1
instance65_50_6	6	283249	305843	348646	13.2	13.8
instance66_50_6	6	239858	279096	300920	18.2	21.3
instance67_50_6	6	235152	236615	237630	12.3	12.5
instance68_50_6	6	183951	222535	246368	12.5	15.2
instance69_50_6	6	244020	252408	258177	14.2	15.9
instance70_50_6	6	247307	248975	249857	12.6	12.6
instance71_50_6	6	248834	270437	312410	11.8	12.2
instance72_50_6	6	257218	281139	328541	12.5	17.2
instance73_50_6	6	177039	177484	178144	11.4	11.6
instance74_50_6	6	179415	179789	180364	10.3	10.7
instance75_50_6	6	181104	182901	184718	11.1	11.2
instance76_50_6	6	294068	294997	295788	11.3	11.3
instance77_50_6	6	297615	300589	305564	11.6	11.6
instance78_50_6	6	299727	299881	300077	11.8	13.2

Table 8: 50 customers, time horizon 6

instance79_50_6	6	177039	177484	178144	12.2	12.5
instance80_50_6	6	179415	179789	180364	10.8	11.2
instance81_50_6	6	181104	182901	184718	11.6	11.8
instance82_50_6	6	294068	294997	295788	11.4	11.4
instance83_50_6	6	297615	300589	305564	11.2	11.2
instance84_50_6	6	299727	299881	300077	12.0	12.8
instance85_50_6	6	284482	285043	285406	11.7	12.1
instance86_50_6	6	287373	289371	291392	11.2	11.3
instance87_50_6	6	294513	317087	360849	12.0	15.8
instance88_50_6	6	402426	405088	406889	10.6	11.2
instance89_50_6	6	408064	411313	415972	11.0	11.3
instance90_50_6	6	411060	413281	414724	12.0	16.5
instance91_50_6	6	284482	285043	285406	11.0	11.7
instance92_50_6	6	287373	289371	291392	10.9	11.0
instance93_50_6	6	294513	317087	360849	11.9	12.0
instance94_50_6	6	402426	405088	406889	11.0	11.2
instance95_50_6	6	408064	411313	415972	10.8	11.2
instance96_50_6	6	411060	413281	414724	11.7	16.3

Table 9: 75 customers, time horizon 6

Instance	**T**	**minCost**	**avgCost**	**maxCost**	**minTime**	**avgTime**
instance1_75_6	6	318184	326927	333591	21.3	29.7
instance2_75_6	6	284483	318489	344805	22.7	29.6
instance3_75_6	6	328583	338028	352827	25.2	29.6
instance4_75_6	6	490621	491347	492078	19.5	20.1
instance5_75_6	6	509343	516542	522747	23.2	25.8
instance6_75_6	6	498685	526853	560820	24.8	25.5
instance7_75_6	6	279630	283286	290117	21.1	21.5
instance8_75_6	6	281325	304909	339990	23.2	24.5
instance9_75_6	6	289210	305558	333139	25.4	26.0
instance10_75_6	6	408503	424580	455701	20.0	20.4

Table 9: 75 customers, time horizon 6

instance11_75_6	6	409576	428654	466261	20.8	22.3
instance12_75_6	6	417858	448023	463557	22.2	22.6
instance13_75_6	6	475819	482490	487130	25.1	28.9
instance14_75_6	6	497761	525080	578093	28.6	29.6
instance15_75_6	6	426158	480273	513078	25.7	27.0
instance16_75_6	6	650721	684661	741017	19.8	22.0
instance17_75_6	6	593303	640499	676712	21.6	21.9
instance18_75_6	6	660841	696238	760371	22.0	22.7
instance19_75_6	6	356036	413710	451233	21.3	22.8
instance20_75_6	6	438390	479633	544309	21.5	21.6
instance21_75_6	6	448964	518932	559520	21.4	27.8
instance22_75_6	6	529100	588837	624958	18.5	20.8
instance23_75_6	6	526222	586671	618027	19.2	19.4
instance24_75_6	6	626769	692727	729476	19.9	20.2
instance25_75_6	6	166707	166707	166707	22.4	22.6
instance26_75_6	6	168864	169041	169131	22.4	22.8
instance27_75_6	6	173129	173129	173129	23.7	23.8
instance28_75_6	6	348791	351719	354357	20.6	40.5
instance29_75_6	6	350814	350975	351270	44.8	52.2
instance30_75_6	6	355910	358250	360839	22.1	36.3
instance31_75_6	6	139372	140031	140719	28.1	53.3
instance32_75_6	6	142231	142650	142942	30.1	44.7
instance33_75_6	6	147762	148181	148554	21.5	26.8
instance34_75_6	6	315812	316483	317165	34.3	38.3
instance35_75_6	6	320792	321392	322576	24.2	25.0
instance36_75_6	6	324746	325410	326145	42.8	55.8
instance37_75_6	6	182099	182384	182942	21.1	21.5
instance38_75_6	6	186546	186794	186974	22.2	22.3
instance39_75_6	6	196653	196653	196653	23.1	23.3
instance40_75_6	6	362323	365929	368458	28.6	36.1
instance41_75_6	6	372165	373111	374395	20.4	37.1

Table 9: 75 customers, time horizon 6

instance42_75_6	6	382124	383363	384363	22.5	42.1
instance43_75_6	6	153861	154533	155417	34.4	44.7
instance44_75_6	6	159946	160168	160585	20.2	20.3
instance45_75_6	6	168417	168876	169197	20.9	21.2
instance46_75_6	6	329464	330912	333452	42.0	48.3
instance47_75_6	6	334299	338144	340246	30.8	41.5
instance48_75_6	6	349071	350455	351518	28.8	31.6
instance49_75_6	6	227397	231700	233914	26.8	28.2
instance50_75_6	6	231512	245634	271598	21.2	27.1
instance51_75_6	6	229301	237314	246855	27.6	32.9
instance52_75_6	6	249923	253161	258071	23.6	24.9
instance53_75_6	6	243881	246223	248202	32.5	37.5
instance54_75_6	6	248938	257960	271192	23.4	26.5
instance55_75_6	6	193654	220743	235007	21.5	22.7
instance56_75_6	6	193262	194066	195250	33.6	35.7
instance57_75_6	6	200156	216074	243655	25.0	30.6
instance58_75_6	6	211876	214283	218622	23.0	29.9
instance59_75_6	6	213135	228544	256457	23.2	29.5
instance60_75_6	6	217530	218726	220827	30.4	34.3
instance61_75_6	6	382853	422455	480871	22.5	31.9
instance62_75_6	6	387023	397412	405188	26.9	27.6
instance63_75_6	6	332416	391650	427743	27.4	33.6
instance64_75_6	6	404259	439452	502842	24.3	28.1
instance65_75_6	6	421277	453254	505891	26.6	31.2
instance66_75_6	6	427178	479860	581603	26.4	27.9
instance67_75_6	6	435710	465422	523047	22.6	26.0
instance68_75_6	6	447381	500301	527126	22.2	26.5
instance69_75_6	6	538853	538853	538853	20.8	22.4
instance70_75_6	6	365491	460345	541907	21.5	28.2
instance71_75_6	6	373866	407517	469722	27.9	29.8
instance72_75_6	6	557713	557713	557713	23.3	23.5

Table 9: 75 customers, time horizon 6

instance73_75_6	6	261282	277204	307644	18.9	19.2
instance74_75_6	6	263677	265189	266171	19.6	20.3
instance75_75_6	6	316234	316951	317410	20.3	20.7
instance76_75_6	6	436758	437565	438702	19.8	20.9
instance77_75_6	6	441426	457382	489130	20.8	24.0
instance78_75_6	6	394659	430242	448683	22.0	23.8
instance79_75_6	6	261282	277204	307644	20.2	20.3
instance80_75_6	6	263677	265189	266171	20.7	21.2
instance81_75_6	6	316234	317208	318181	21.7	27.6
instance82_75_6	6	436758	437565	438702	20.5	21.5
instance83_75_6	6	441426	457382	489130	21.0	23.9
instance84_75_6	6	394659	430242	448683	22.1	23.9
instance85_75_6	6	415257	417098	418149	20.2	20.5
instance86_75_6	6	421864	454754	516858	20.3	24.9
instance87_75_6	6	435968	497752	529944	22.3	24.6
instance88_75_6	6	500349	594405	690256	20.0	20.2
instance89_75_6	6	595669	598966	601251	20.4	20.7
instance90_75_6	6	608612	609752	611175	22.1	27.5
instance91_75_6	6	415257	417098	418149	19.6	19.8
instance92_75_6	6	421864	454754	516858	20.0	24.2
instance93_75_6	6	435968	497752	529944	21.8	24.5
instance94_75_6	6	500349	594405	690256	20.1	20.6
instance95_75_6	6	595669	598966	601251	20.1	20.4
instance96_75_6	6	608612	609752	611175	22.8	28.1

Table 10: 100 customers, time horizon 6

Instance	**T**	**minCost**	**avgCost**	**maxCost**	**minTime**	**avgTime**
instance1_100_6	6	409520	411621	415735	35.7	40.0
instance2_100_6	6	431136	439401	453250	41.9	46.1
instance3_100_6	6	428191	457022	491292	46.8	60.7
instance4_100_6	6	639132	662577	692552	32.5	43.8

Table 10: 100 customers, time horizon 6

instance5_100_6	6	592893	628165	652825	35.0	44.5
instance6_100_6	6	647078	664856	673830	43.7	45.4
instance7_100_6	6	363832	388823	427149	31.8	34.5
instance8_100_6	6	377435	379467	381741	43.4	55.3
instance9_100_6	6	447397	484032	502475	38.1	39.7
instance10_100_6	6	585215	610018	653302	30.3	31.7
instance11_100_6	6	525840	572859	603448	32.9	38.9
instance12_100_6	6	596992	642030	665773	31.6	34.0
instance13_100_6	6	624127	634236	648258	39.8	45.3
instance14_100_6	6	629112	680073	759781	40.1	42.1
instance15_100_6	6	646116	800795	878135	38.3	41.5
instance16_100_6	6	857329	898648	966774	31.7	48.7
instance17_100_6	6	745612	898801	976322	31.5	33.8
instance18_100_6	6	760369	847564	893849	37.2	43.4
instance19_100_6	6	699641	708791	715886	32.6	35.4
instance20_100_6	6	468582	714647	837888	31.0	34.8
instance21_100_6	6	849171	849398	849620	33.5	33.9
instance22_100_6	6	668017	799031	926387	28.6	29.4
instance23_100_6	6	684586	810705	945047	30.1	30.9
instance24_100_6	6	709858	926198	1078791	25.4	28.7
instance25_100_6	6	204453	204480	204515	35.1	35.3
instance26_100_6	6	207833	207833	207833	35.0	35.8
instance27_100_6	6	212755	212755	212755	38.1	38.2
instance28_100_6	6	438256	438347	438526	33.4	33.6
instance29_100_6	6	441568	441692	441814	34.2	34.5
instance30_100_6	6	446970	446970	446970	35.9	36.5
instance31_100_6	6	174272	174678	175021	30.9	59.3
instance32_100_6	6	176980	177934	178472	32.7	49.6
instance33_100_6	6	184156	184226	184288	34.8	35.2
instance34_100_6	6	396837	397608	398792	65.0	81.7
instance35_100_6	6	402306	402755	403301	41.2	53.5

Table 10: 100 customers, time horizon 6

instance36_100_6	6	406666	409125	411125	54.4	78.3
instance37_100_6	6	224643	224709	224824	33.3	33.8
instance38_100_6	6	231811	231907	231965	34.4	34.6
instance39_100_6	6	241835	241835	241835	37.3	37.5
instance40_100_6	6	458132	458296	458397	32.1	32.6
instance41_100_6	6	465145	465398	465788	34.0	48.1
instance42_100_6	6	476000	476000	476000	36.5	36.7
instance43_100_6	6	193637	193866	194090	31.4	38.5
instance44_100_6	6	199833	200520	201436	31.7	32.0
instance45_100_6	6	212049	212861	213430	33.5	33.7
instance46_100_6	6	415867	420303	424413	29.8	62.0
instance47_100_6	6	425523	427947	431407	29.7	49.4
instance48_100_6	6	438292	440036	443068	48.1	56.1
instance49_100_6	6	307729	339927	361390	32.7	39.5
instance50_100_6	6	297580	328594	372057	46.3	61.0
instance51_100_6	6	310196	343128	403899	38.3	53.7
instance52_100_6	6	311880	346717	375758	41.3	51.6
instance53_100_6	6	318957	343011	378103	39.3	60.7
instance54_100_6	6	321333	345235	384629	49.6	54.7
instance55_100_6	6	366523	366714	366818	30.4	32.2
instance56_100_6	6	259822	333787	370837	34.8	39.1
instance57_100_6	6	375412	375786	375974	31.5	35.4
instance58_100_6	6	332412	353914	390198	33.6	35.8
instance59_100_6	6	275432	354577	394217	33.8	39.2
instance60_100_6	6	398792	399166	399354	31.4	35.4
instance61_100_6	6	503996	551112	623191	37.4	48.4
instance62_100_6	6	510665	664873	741977	34.6	42.3
instance63_100_6	6	751664	751664	751664	37.8	37.8
instance64_100_6	6	525076	604148	759938	34.7	47.4
instance65_100_6	6	550690	655937	765357	38.2	39.9
instance66_100_6	6	431602	660563	775044	39.6	43.0

Table 10: 100 customers, time horizon 6

instance67_100_6	6	703872	704179	704511	32.5	34.4
instance68_100_6	6	714340	714340	714340	36.2	36.3
instance69_100_6	6	723689	723689	723689	37.4	37.6
instance70_100_6	6	624924	693355	727891	31.8	40.2
instance71_100_6	6	737720	737720	737720	34.2	34.6
instance72_100_6	6	747069	747069	747069	37.2	37.5
instance73_100_6	6	406637	407224	407953	28.2	28.8
instance74_100_6	6	347144	369262	412485	29.8	33.0
instance75_100_6	6	356623	396751	417157	31.2	33.8
instance76_100_6	6	565652	588393	633424	28.6	29.8
instance77_100_6	6	566046	568989	570661	30.0	30.2
instance78_100_6	6	576915	599507	644210	34.1	34.6
instance79_100_6	6	406637	407224	407953	29.6	30.2
instance80_100_6	6	347144	369262	412485	30.6	33.6
instance81_100_6	6	356623	396751	417157	33.5	36.0
instance82_100_6	6	565652	588393	633424	31.6	33.4
instance83_100_6	6	566046	568989	570661	32.0	32.2
instance84_100_6	6	576915	599507	644210	34.3	35.4
instance85_100_6	6	552180	554140	555259	30.5	31.2
instance86_100_6	6	561629	604113	687870	31.1	31.5
instance87_100_6	6	705125	706738	707863	35.1	56.9
instance88_100_6	6	905073	906381	908543	29.5	30.3
instance89_100_6	6	781372	826112	913525	31.1	31.2
instance90_100_6	6	799807	843155	927964	33.9	34.3
instance91_100_6	6	552180	554140	555259	30.7	30.9
instance92_100_6	6	561629	604113	687870	30.3	30.8
instance93_100_6	6	704318	705768	707863	33.9	69.6
instance94_100_6	6	905073	906381	908543	29.2	30.0
instance95_100_6	6	781372	826112	913525	31.3	31.7
instance96_100_6	6	799807	843155	927964	33.5	33.9

Table 11: 125 customers, time horizon 6

Instance	T	minCost	avgCost	maxCost	minTime	avgTime
instance1_125_6	6	491591	537872	591077	55.2	67.3
instance2_125_6	6	512693	516134	519268	60.9	68.3
instance3_125_6	6	504652	544219	583757	59.2	68.0
instance4_125_6	6	782339	793043	805274	54.6	74.4
instance5_125_6	6	775861	791317	805665	67.0	77.7
instance6_125_6	6	807891	845162	871955	60.3	79.8
instance7_125_6	6	445894	447533	448377	56.2	71.7
instance8_125_6	6	453533	499785	591593	57.7	63.9
instance9_125_6	6	461561	507728	533326	57.0	74.9
instance10_125_6	6	710115	763154	792347	45.8	46.7
instance11_125_6	6	711077	741975	788630	46.7	48.5
instance12_125_6	6	796057	796837	797917	49.3	50.8
instance13_125_6	6	898630	948969	1029105	46.4	50.5
instance14_125_6	6	781263	905930	1034973	53.0	63.0
instance15_125_6	6	788795	883612	1044091	45.0	55.7
instance16_125_6	6	1031710	1118873	1164845	44.0	47.5
instance17_125_6	6	1163201	1181378	1194242	47.1	55.2
instance18_125_6	6	1081177	1171510	1218453	55.4	61.0
instance19_125_6	6	546723	799668	989335	41.0	65.1
instance20_125_6	6	707339	900801	997533	46.0	55.3
instance21_125_6	6	1006531	1006531	1006531	44.2	48.2
instance22_125_6	6	961460	1061597	1112736	42.0	50.1
instance23_125_6	6	1117844	1129606	1147977	41.6	46.0
instance24_125_6	6	986667	1139934	1280898	44.9	53.3
instance25_125_6	6	246064	246064	246064	50.2	50.7
instance26_125_6	6	248563	248563	248563	51.0	51.4
instance27_125_6	6	253202	253202	253202	54.6	55.4
instance28_125_6	6	527796	527923	528068	47.3	47.7
instance29_125_6	6	530354	530550	530696	49.4	49.9
instance30_125_6	6	534921	535113	535210	54.8	54.9

Table 11: 125 customers, time horizon 6

instance31_125_6	6	207912	208014	208110	45.7	47.0
instance32_125_6	6	210081	210172	210219	47.6	48.3
instance33_125_6	6	214356	214489	214569	51.9	52.4
instance34_125_6	6	478624	479003	479359	122.3	138.9
instance35_125_6	6	482176	482656	483335	111.7	120.4
instance36_125_6	6	488240	488391	488610	121.1	124.1
instance37_125_6	6	266491	266712	266840	48.4	48.8
instance38_125_6	6	272381	272524	272653	49.6	50.3
instance39_125_6	6	281791	281791	281791	54.3	54.7
instance40_125_6	6	548414	548427	548453	46.8	47.4
instance41_125_6	6	553850	554113	554439	49.9	50.4
instance42_125_6	6	562802	563010	563139	53.4	54.6
instance43_125_6	6	227035	227158	227366	44.5	45.0
instance44_125_6	6	231962	232358	232564	46.0	46.2
instance45_125_6	6	240900	241262	241973	50.5	50.7
instance46_125_6	6	496346	500949	505718	41.8	89.5
instance47_125_6	6	504742	505806	506875	56.2	89.0
instance48_125_6	6	522226	522361	522491	47.0	47.5
instance49_125_6	6	346558	378483	420921	53.1	64.6
instance50_125_6	6	349037	391790	476689	51.2	65.6
instance51_125_6	6	360656	422450	481457	55.1	88.3
instance52_125_6	6	372786	385991	398979	63.9	71.3
instance53_125_6	6	379129	388557	395698	70.5	79.7
instance54_125_6	6	408628	426403	459595	73.4	82.8
instance55_125_6	6	296350	345150	436126	48.3	70.7
instance56_125_6	6	301783	393393	439199	49.6	54.1
instance57_125_6	6	317444	401718	443855	48.5	56.2
instance58_125_6	6	328926	399401	464206	48.2	55.8
instance59_125_6	6	339641	424733	467279	49.7	64.6
instance60_125_6	6	413558	434082	471935	54.1	58.6
instance61_125_6	6	627763	757868	876785	48.3	74.0

Table 11: 125 customers, time horizon 6

instance62_125_6	6	615972	749936	882609	50.8	65.1
instance63_125_6	6	891784	891784	891784	54.6	54.9
instance64_125_6	6	635420	642143	646958	74.6	88.0
instance65_125_6	6	634412	774473	910689	51.8	62.8
instance66_125_6	6	919864	919864	919864	59.0	60.0
instance67_125_6	6	836418	836559	836682	49.3	50.5
instance68_125_6	6	845169	845169	845169	51.0	52.4
instance69_125_6	6	851969	853472	854224	52.3	53.6
instance70_125_6	6	864498	864639	864762	46.4	47.3
instance71_125_6	6	873249	873249	873249	49.8	50.2
instance72_125_6	6	880049	881552	882304	53.3	54.9
instance73_125_6	6	484886	485908	487409	40.0	40.7
instance74_125_6	6	409295	411926	413675	42.5	51.3
instance75_125_6	6	417951	444635	496540	46.1	47.3
instance76_125_6	6	680804	733804	761039	42.8	51.4
instance77_125_6	6	682464	709094	759810	43.2	43.6
instance78_125_6	6	686405	714329	767788	47.3	47.9
instance79_125_6	6	484886	485908	487409	43.6	44.1
instance80_125_6	6	409295	411926	413675	43.2	50.9
instance81_125_6	6	417951	444635	496540	50.2	51.8
instance82_125_6	6	680804	733804	761039	44.2	54.5
instance83_125_6	6	682464	709094	759810	43.8	45.3
instance84_125_6	6	686405	714329	767788	49.5	50.7
instance85_125_6	6	655441	863721	968279	42.0	46.5
instance86_125_6	6	817675	819103	820563	60.6	80.4
instance87_125_6	6	679178	778348	828224	48.1	49.4
instance88_125_6	6	925815	980007	1087439	42.9	49.0
instance89_125_6	6	931213	986394	1089904	36.6	42.2
instance90_125_6	6	1096772	1099806	1103140	46.9	47.8
instance91_125_6	6	655441	863721	968279	41.5	45.7
instance92_125_6	6	817675	819103	820563	58.7	78.1

Table 11: 125 customers, time horizon 6

instance93_125_6	6	679178	778348	828224	47.1	47.3
instance94_125_6	6	925815	980007	1087439	41.8	48.0
instance95_125_6	6	931213	935433	938066	45.0	47.7
instance96_125_6	6	1096772	1099806	1103140	47.6	48.0

Table 12: 150 customers, time horizon 6

Instance	**T**	**minCost**	**avgCost**	**maxCost**	**minTime**	**avgTime**
instance1_150_6	6	648553	687556	751588	110.0	110.9
instance2_150_6	6	694291	753139	818294	68.5	97.2
instance3_150_6	6	827689	827689	827689	78.8	80.6
instance4_150_6	6	1068043	1082040	1099223	63.6	75.7
instance5_150_6	6	996999	1064703	1109593	72.5	98.2
instance6_150_6	6	1016177	1083161	1117458	81.6	121.7
instance7_150_6	6	582172	612041	670961	94.3	101.2
instance8_150_6	6	578469	643214	769794	71.8	97.9
instance9_150_6	6	702868	728713	779082	82.4	93.3
instance10_150_6	6	912389	982491	1018480	66.0	67.4
instance11_150_6	6	820656	921588	1032270	70.8	79.9
instance12_150_6	6	937491	968034	1027982	75.0	78.3
instance13_150_6	6	972786	1107094	1345313	69.1	85.9
instance14_150_6	6	1216404	1264521	1352343	74.4	101.2
instance15_150_6	6	1196053	1312999	1371472	79.1	112.6
instance16_150_6	6	1320152	1402290	1545474	78.7	97.6
instance17_150_6	6	1327189	1343838	1362098	89.4	98.4
instance18_150_6	6	1387185	1448230	1549208	91.5	113.8
instance19_150_6	6	920406	1109836	1291165	63.2	105.8
instance20_150_6	6	923437	1175958	1302532	64.9	80.7
instance21_150_6	6	1322471	1322471	1322471	73.3	73.7
instance22_150_6	6	1041264	1244220	1440467	58.1	77.2
instance23_150_6	6	1026789	1311723	1455488	63.7	64.6
instance24_150_6	6	1473526	1608311	1675704	61.3	67.6

Table 12: 150 customers, time horizon 6

instance25_150_6	6	314270	314270	314270	68.8	69.3
instance26_150_6	6	317729	317729	317729	72.8	74.0
instance27_150_6	6	327124	327124	327124	80.0	80.6
instance28_150_6	6	667270	667270	667270	69.6	70.1
instance29_150_6	6	670729	670729	670729	74.8	74.9
instance30_150_6	6	680124	680124	680124	79.5	80.5
instance31_150_6	6	264899	264995	265072	65.1	66.5
instance32_150_6	6	268898	269014	269151	72.2	73.2
instance33_150_6	6	277926	278048	278260	75.8	76.4
instance34_150_6	6	606447	610782	614383	69.6	107.7
instance35_150_6	6	613270	615661	619575	77.0	166.1
instance36_150_6	6	628870	629622	630627	71.5	72.2
instance37_150_6	6	343428	343428	343428	67.5	68.1
instance38_150_6	6	350295	350447	350588	73.4	73.9
instance39_150_6	6	369008	369394	369587	79.3	80.0
instance40_150_6	6	696428	696428	696428	65.4	67.1
instance41_150_6	6	703431	703677	703801	71.5	72.3
instance42_150_6	6	721600	721915	722219	76.8	77.9
instance43_150_6	6	290442	290540	290734	61.7	63.1
instance44_150_6	6	299776	299899	300038	67.8	68.6
instance45_150_6	6	317936	318397	319145	73.9	74.3
instance46_150_6	6	642752	643232	643682	58.9	59.7
instance47_150_6	6	653116	653314	653476	64.9	66.1
instance48_150_6	6	671027	671459	672010	68.2	69.8
instance49_150_6	6	458672	498385	551034	84.4	89.8
instance50_150_6	6	495549	543366	627445	74.1	105.6
instance51_150_6	6	636836	636836	636836	79.2	79.8
instance52_150_6	6	490281	578225	659059	65.9	89.3
instance53_150_6	6	501549	564803	599943	117.6	143.0
instance54_150_6	6	539753	627818	671851	79.4	105.5
instance55_150_6	6	575361	575361	575361	65.4	66.2

Table 12: 150 customers, time horizon 6

instance56_150_6	6	495365	551035	578895	66.8	97.3
instance57_150_6	6	587463	587940	588179	78.0	78.5
instance58_150_6	6	436235	552329	610376	66.9	77.8
instance59_150_6	6	524483	584084	613910	65.1	84.8
instance60_150_6	6	622478	622955	623194	77.7	78.6
instance61_150_6	6	644798	873022	1154522	66.5	89.0
instance62_150_6	6	841197	898463	996354	94.0	95.2
instance63_150_6	6	1180275	1180275	1180275	78.9	79.6
instance64_150_6	6	834517	904549	1016595	81.2	97.8
instance65_150_6	6	839499	869183	897874	98.1	113.3
instance66_150_6	6	1215290	1215290	1215290	87.4	87.6
instance67_150_6	6	1105736	1105736	1105736	70.2	70.6
instance68_150_6	6	1110754	1112344	1113140	70.9	71.9
instance69_150_6	6	1131205	1131501	1131650	76.9	78.4
instance70_150_6	6	1140751	1140751	1140751	65.6	66.9
instance71_150_6	6	1145769	1147359	1148155	70.3	72.5
instance72_150_6	6	1166220	1166516	1166665	67.8	75.4
instance73_150_6	6	531256	600203	635095	55.9	56.5
instance74_150_6	6	536516	604967	639232	61.5	116.3
instance75_150_6	6	650125	650871	651717	99.0	103.9
instance76_150_6	6	974406	1015453	1093896	56.9	83.9
instance77_150_6	6	877398	986634	1097836	59.2	74.4
instance78_150_6	6	887696	922660	992363	68.4	69.2
instance79_150_6	6	531256	600203	635095	60.2	62.1
instance80_150_6	6	536516	604967	639232	62.1	122.1
instance81_150_6	6	650125	651061	652288	72.8	97.5
instance82_150_6	6	974406	1015453	1093896	59.9	90.6
instance83_150_6	6	877398	986634	1097836	60.3	78.8
instance84_150_6	6	887696	922660	992363	69.8	70.7
instance85_150_6	6	1061862	1062568	1063812	56.4	63.4
instance86_150_6	6	869121	1005183	1073520	66.2	115.1

Table 12: 150 customers, time horizon 6

instance87_150_6	6	629521	942110	1098760	84.2	90.4
instance88_150_6	6	1198798	1334249	1402522	57.8	57.9
instance89_150_6	6	1212647	1350197	1419004	59.8	96.9
instance90_150_6	6	1235375	1374318	1444626	66.4	101.1
instance91_150_6	6	1060916	1061603	1062031	57.7	77.7
instance92_150_6	6	869121	1005183	1073520	64.1	113.0
instance93_150_6	6	1098243	1163613	1293837	82.5	97.4
instance94_150_6	6	1198798	1334249	1402522	58.1	58.3
instance95_150_6	6	1212647	1350197	1419004	60.4	99.5
instance96_150_6	6	1235375	1374728	1444626	65.4	114.2

Table 13 shows results for the instances with 30 time periods. Instance 1 through 62 is run with a maximum iteration count of 100000 as stopping criteria. Instance 63 through 96 is run with a maximum iteration count of 20000 as stopping criteria.

Table 13: 50 customers, time horizon 30

Instance	**T**	**Cost**	**Time**
instance1_50_30	30	1722693	1771.0
instance2_50_30	30	1762331	1929.8
instance3_50_30	30	1759171	2487.3
instance4_50_30	30	2792085	1368.2
instance5_50_30	30	2772415	1795.2
instance6_50_30	30	2904943	943.9
instance7_50_30	30	1620674	672.4
instance8_50_30	30	1706871	733.4
instance9_50_30	30	1732038	860.8
instance10_50_30	30	2434295	1382.0
instance11_50_30	30	2607646	972.4
instance12_50_30	30	2677603	1248.5
instance13_50_30	30	2686181	2040.2
instance14_50_30	30	2772046	1552.3
instance15_50_30	30	2821388	1831.0

Table 13: 50 customers, time horizon 30

instance16_50_30	30	3850498	1516.1
instance17_50_30	30	3892899	1295.5
instance18_50_30	30	3935440	1504.1
instance19_50_30	30	2673088	1161.7
instance20_50_30	30	2637425	1019.9
instance21_50_30	30	2822677	1024.1
instance22_50_30	30	3402043	1246.5
instance23_50_30	30	3543942	902.2
instance24_50_30	30	3727683	1095.5
instance25_50_30	30	827511	1046.1
instance26_50_30	30	837984	1097.9
instance27_50_30	30	857908	1267.0
instance28_50_30	30	1960913	1014.0
instance29_50_30	30	1976010	1088.2
instance30_50_30	30	1997910	1227.0
instance31_50_30	30	782561	1431.0
instance32_50_30	30	798746	1239.0
instance33_50_30	30	822497	2077.7
instance34_50_30	30	1875627	1274.9
instance35_50_30	30	1890156	1794.1
instance36_50_30	30	1924000	1227.3
instance37_50_30	30	928402	1073.0
instance38_50_30	30	948544	1070.3
instance39_50_30	30	994962	1261.3
instance40_50_30	30	2042933	1959.3
instance41_50_30	30	2079371	1269.0
instance42_50_30	30	2133906	1224.3
instance43_50_30	30	864469	1585.7
instance44_50_30	30	894420	1034.9
instance45_50_30	30	951698	1030.4
instance46_50_30	30	1962075	1413.1

Table 13: 50 customers, time horizon 30

instance47_50_30	30	1993695	1058.3
instance48_50_30	30	2052727	1078.7
instance49_50_30	30	1143136	964.2
instance50_50_30	30	1154464	1017.0
instance51_50_30	30	1173909	1242.5
instance52_50_30	30	1259611	963.1
instance53_50_30	30	1241234	1820.6
instance54_50_30	30	1290384	1261.7
instance55_50_30	30	1083741	1950.9
instance56_50_30	30	1101545	1167.2
instance57_50_30	30	1147998	1187.6
instance58_50_30	30	1136145	2618.8
instance59_50_30	30	1216526	1196.6
instance60_50_30	30	1244256	1854.2
instance61_50_30	30	2079033	1790.8
instance62_50_30	30	2174297	1490.8
instance63_50_30	30	2218504	934.8
instance64_50_30	30	2260811	748.5
instance65_50_30	30	2280355	818.1
instance66_50_30	30	2273036	998.5
instance67_50_30	30	2066898	678.1
instance68_50_30	30	2162498	693.9
instance69_50_30	30	2208974	816.2
instance70_50_30	30	2190762	648.2
instance71_50_30	30	2278973	711.1
instance72_50_30	30	2288662	818.1
instance73_50_30	30	1628368	340.0
instance74_50_30	30	1643037	329.1
instance75_50_30	30	1666188	407.7
instance76_50_30	30	2691126	324.3
instance77_50_30	30	2737147	336.1

Table 13: 50 customers, time horizon 30

instance78_50_30	30	2722436	393.9
instance79_50_30	30	1628368	333.6
instance80_50_30	30	1643037	320.0
instance81_50_30	30	1666188	399.7
instance82_50_30	30	2691126	322.4
instance83_50_30	30	2737147	334.6
instance84_50_30	30	2722436	393.4
instance85_50_30	30	2569788	308.7
instance86_50_30	30	2632737	329.8
instance87_50_30	30	2664987	404.6
instance88_50_30	30	3637921	307.6
instance89_50_30	30	3740214	347.2
instance90_50_30	30	3799351	391.7
instance91_50_30	30	2569788	305.3
instance92_50_30	30	2632737	333.1
instance93_50_30	30	2664987	398.4
instance94_50_30	30	3637921	306.6
instance95_50_30	30	3740214	344.4
instance96_50_30	30	3799351	390.7