CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Metaheuristic for Stochastic Service Network Design

Arild Hoff
Arnt-Gunnar Lium
Arne Løkketangen
Teodor Gabriel Crainic

December 2007

CIRRELT-2007-62

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Metaheuristic for Stochastic Service Network Design

Arild Hoff[1], Arnt-Gunnar Lium[2], Arne Løkketangen[1], Teodor Gabriel Crainic[3,*]

[1] Molde University College, P.O. Box 2110, NO-6402 Molde, Norway

[2] SINTEF Technology and Society, Applied Economics and Operations, NO-7465 Trondheim, Norway

[3] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), and Department of Management and Technology, Université du Québec à Montréal, C.P. 8888, succursale Centre-ville, Montréal, Canada H3C 3P8

Abstract. This paper considers the time-dependent service network design problem with stochastic demand represented by scenarios. To our knowledge, this is the first attempt to address real life-size instances of this problem. The model integrates the balancing of empty vehicles, the cost of handling freight in intermediate terminals, the costs associated with moving freight using the selected services, and the penalty costs of not being able to deliver freight. A metaheuristic method is presented and computational results are reported on a set of large new problem instances.

Keywords. Stochastic programming, scheduled service network design, scenarios, metaheuristics.

Acknowledgements. The authors thank Øyvind Halskau for pointing out errors and helping us improve the paper. While working on this project, Arild Hoff and Arnt-Gunnar Lium were employed as Research Fellows at Molde University College (Norway). Dr. Teodor Gabriel Crainic was Adjunct Professor at Molde University College and the department of Computer Science and Operations Research of the Université de Montréal (Canada). Partial funding for this project was provided by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

---

* Corresponding author: Teodor-Gabriel.Crainic@cirrelt.ca

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2007

Introduction

Planning operations with the aid of decision support systems (DSS) can help in making better decisions. In industries ranging from telecommunications to various modes of transportation of freight and passengers, many DSS are based on service network design formulations. In the freight transportation industry, service network design methods provide help with deciding between which terminals a company should offer services, at which frequency and according to what schedules service should be offered, and how to deal with the empty vehicles. Simultaneously with the construction of the underlying service network, policies regarding operations at terminals have to be addressed, as well as the movement of freight from origins to destinations. This is usually done so that costs are minimized or profit maximized. The output of the process is usually denoted "transportation plan". See Crainic (2000, 2003) for an introduction to service network design in freight transport.

The last two decades have brought significant improvements in the modeling of these problems as well as in algorithmic and computational efficiency. Most of this work has been done for the deterministic case, however; the interested reader is referred to Crainic (2003) for an overview. Studies by Lium, Crainic, and Wallace (2007a, 2007b) indicate that cost reductions can be achieved when building plans based on the explicit consideration of stochastic demand compared to plans based on the assumption of deterministic demand. Due to combinatorial challenges, only small instances could be solved by the authors. This paper is based on their formulations and looks into how to obtain good solutions for large instances of the time-dependent service network design problem with stochastic demand. We also extend the formulation proposed by Lium, Crainic, and Wallace (2007a, 2007b) to incorporate the cost of handling freight at intermediate terminals as well as the cost associated to movement/storage of the commodities (for example inventory cost of storing freight at intermediary terminals). In previous research, Ghamlouche, Crainic, and Gendreau (2003, 2004) used linear programming to optimize the commodity flow associated to a given service design. We propose instead a local search-based greedy heuristic able to obtain a more than tenfold increase in speed compared to the leading commercial solver, usually giving solutions close to optimality.

The goal of this paper is to solve large time-dependent service network design problems with stochastic demand. These problems are seen as intractable and currently impossible to solve using exact methods. Our contribution is to provide the first attempt to solve such problems using metaheuristics. The paper presents computational results for 55 new test instances and shows that it is possible to solve such real life size problems within reasonable time. We also look at how the representation chosen for demand stochasticity can affect the solutions obtained.

The paper is organized as follows. Section 1 provides the assumptions for the model, the mathematical formulation, and the representation of the demand stochasticity. Section 2 describes the main ideas and elements of the solution approach, which are then computationally evaluated in Section 3. As a result of this evaluation, the final metaheuristic design of the procedure we proposed is presented in the last sub-section of Section 3. Computational results with this metaheuristic are reported in Section 4 and we conclude in Section 5.

1. The stochastic service network design model

Deciding how much capacity to offer and where and when to offer it so that it matches the demand, is one of the most important decisions being made in the transportation industry. The problem is that most cases, one does not know what the actual demand will be at the time when the plan is executed. Consequently, ad-hoc changes have to be made to the plans during operations to better match the offered capacity to the observed demand. Such ad-hoc changes usually come at a much higher cost compared to the capacity allocated in the original plan. Actually, the knowledge of

future demand is out of reach for most practical applications, but the decision maker might be able to describe uncertainty with statistical distributions based, for example, on historical data. The best we then can do is to find a solution or a plan that minimizes the cost of offering services and the expected cost of possible ad-hoc changes required for executing the plan.

We propose a model inspired by the time-dependent stochastic service network design formulation by Lium, Crainic, and Wallace (2007a, 2007b). Their model assumes the use of a homogeneous fleet of vehicles to move commodities. The commodities are moved from their origins, either in the period they become available or in later periods, and they are moved to their destinations within the delivery times, either directly or via one or more intermediary terminals. This formulation assumes that the handling of freight in terminals happens instantaneously (within the time period) and that there are no capacity limitations at terminals. It provides the user with decision support on how many vehicles to use and how to operate them. The planned capacity is supplemented by a high-cost ad-hoc "capacity increase", representing the recourse to additional vehicles, outsourcing or demand rejection.

The goal is to select and schedule services provided by a fleet of homogeneous vehicles, such that the expected total system cost is minimized. The expected total system cost consists of three components: 1) the cost of operating the vehicles; 2) the cost of handling and storing freight in intermediary terminals plus the cost of transporting freight using the vehicles; and 3) the cost of ad-hoc capacity increase. The schedules are to be repeated periodically. The stochasticity of demand is explicitly represented through scenarios. There are no restrictions on how many vehicles can be used. We assume that all demand must be met either using the company's own vehicles or by ad-hoc capacity increases. Traveling time from one terminal to another can take one or more periods and no delays are assumed while traveling. Handling of freight happens instantaneously (within the period) and without delays. The capacity of the vehicles cannot be exceeded.

We create a space-time network by repeating the set of nodes (freight terminals) N in each of the periods $t = 0,\ldots,T-1$. Because schedules must be periodic, we create repetitive schedules in a circular fashion and, thus, the issue of the end-of-horizon effects is not relevant. Assuming a T-period planning horizon, a circular notation means that the period following period t is $(t + 1)$ *mod* T. Each arc (i, j, t) represents either a service, if $i \neq j$, or a holding activity if $i = j$, in period t. We assume that all nodes for period t can be connected to any other node in period $((t + v)$ *mod* $T)$, where v is the number of periods required to go from one node to another.

The stochastic demand is represented by a set of scenarios $s \in S$. To each scenario is attached a probability $p^s \geq 0$, with $\sum p^s = 1$. The demand for each commodity $k \in K$ for every scenario s is defined as $\lambda(k,s)$. A scenario is thus $|K|$-dimensional, as it contains one demand realization for each commodity $k \in K$. We also define for each commodity its origin $o(k)$, destination $d(k)$, and the periods $\sigma(k)$ and $\tau(k)$ when it becomes available at its origin and must be delivered (at the latest) at its destination, respectively. The vehicle capacity is denoted Θ and is given in the same units as the demand.

Let $w \in W$ represent all arcs (i, j, t). Thus, the arc index w is dependent on the values of $i, j,$ and t, and is calculated as $w = t|N|^2 + (i-1)|N| + j$, where N is the set of nodes. A cost c_{ij} is associated with each arc (i, j, t), equal to the cost of operating a vehicle from terminal i to j if $i \neq j$, or to the cost of holding a vehicle at the terminal i if $i = j$. These arcs w can be combined to form paths $l \in L$. To indicate whether an arc belongs to a path we introduce the indicator function $\delta_{lw}(k)$,

$$\delta_{lw}(k) = \begin{cases} 1 \text{ if arc } w \text{ belongs to path } l \in L^k \text{ for commodity } k \\ 0 \quad \text{otherwise} \end{cases}$$

Each commodity k can have several possible paths $l \in L^k$ from its origin $o(k)$ to its destination $d(k)$. Paths are commodity specific, but the capacity of the vehicles must be shared by the commodities when they have one or more arcs (services) in common.

The use of these paths comes at a unit cost a_l which is made up of the following components: 1) the cost associated with transporting a unit of freight (for example incurred by increase in the vehicle fuel consumption); 2) the cost associated with storing freight in a vehicle or at an intermediate terminal (there are no costs of storing freight at origin or destination); and, 3) the cost of moving freight from one vehicle to another at intermediate terminals when the path l for commodity k is served by two or more consecutive services (and, thus, vehicles). The latter cost component can also represent real-world costs associated with having freight being delayed at intermediate terminals. The cost of ad-hoc capacity increase of one unit is represented by b. The cost components c_{ij}, a_l, and b are not commodity specific. u_w is the capacity offered on arc w that equals the total capacity of the vehicles traveling on the arc.

The decision variables are:

x_{ij}^t: Number of vehicles traveling from node i in period t to node j in period t + ν, where ν is an integer number greater than or equal to 1 representing the "traveling" time from i to j. This applies for all i, j and t.

$f_l(k,s)$: Flow of commodity k on path l in scenario s.

$z(k,s)$: Amount of commodity k sent to its destination using the extra, ad-hoc capacity, in scenario s.

The model then becomes:

$$\min \left[\sum_{i\in N} \sum_{j\in N} \sum_{t=0}^{T-1} c_{ij} x_{ij}^t \right. \tag{1a}$$

$$+ \sum_{s\in S} p^s \sum_{k\in K} \sum_{l\in L^k} a_l f_l(k,s) \tag{1b}$$

$$\left. + b \sum_{s\in S} p^s \sum_{k\in K} z(k,s) \right] \tag{1c}$$

$$\sum_{i\in N} x_{ij}^t = \sum_{i\in N} x_{ji}^{(t+\upsilon \bmod T)}, \; t = 0,\ldots,T-1, \forall j \in N \tag{2}$$

$$x_{ij}^t \geq 0 \text{ and integer}, \quad t = 0,\ldots,T-1, \forall i, j \in N \tag{3}$$

$$\sum_{k\in K} \delta_{lw}(k) f_l(k,s) \leq x_{ij}^t \Theta, \; \forall i, j \in N, \forall s \in S, t = 0,1,\ldots,T-1, \forall l \in L^k, \quad \forall w : w = (t|N|^2 + (i-1)|N| + j) \in W \tag{4}$$

$$\sum_{l\in L^k} f_l(k,s) + z(k,s) = \lambda(k,s), \quad \forall s \in S, \forall k \in K \tag{5}$$

$$f_l(k,s) \le \Theta \min_{\delta_{hw}(k)=1} \left\{x_{ij}^t\right\}, \ \forall i,j \in N, \forall l \in L^k, \forall k \in K, \forall s \in S, \\ t = 0,\ldots,T-1, \forall w : w = (t|N|^2 + (i-1)|N| + j) \in W \tag{6}$$

$$f_l(k,s), z(k,s), x_{ij}^t \ge 0, \quad \forall k \in K, \forall l \in L^k, \forall s \in S, \forall i,j \in N, \forall t \in T \tag{7}$$

The objective function is given by the sum of (1a), (1b), and (1c) and minimizes the total expected system cost, where (1a) associates a cost with the vehicles moving between terminals plus the cost of holding them at terminals, (1b) denotes the expected cost of moving all commodities *k* on the paths *l* over all the scenarios, and the expected cost of adding ad-hoc capacity is given by (1c). Equation (2) is the conservation of flow constraint for vehicles, while (3) is a non-negativity and integrality constraint for vehicles. Relation (4) ensures that the flow of all the commodities *k* on an arc do not exceed its capacity. Equation (5) ensures that the demand of commodity *k* in scenario *s* is satisfied by using the vehicles of the company and, eventually, additional ad-hoc capacity. It can be noted that nothing restricts the flow splitting of a commodity, which can therefore use multiple paths from origin to destination. Relations (6) enforce that the flow of a commodity *k* on path $l \in L^k$ must be equal to or lower than the minimum capacity of the arcs making up the path *l*. Relation (7) is a non-negativity constraint.

The model makes some tacit assumptions regarding the construction of the paths for the commodities. One assumption is that any terminal can be used as an intermediate terminal. Another is that freight cannot be shipped out of such a terminal before it has arrived. We also make the assumption that there is a cost of having a commodity "on its way" to discourage excessive transportation and storage at intermediate terminals. There is no cost associated with storing freight at its respective origin and destination.

The quality of our solutions can only be as good as the input used. Ideally, the distributions describing stochasticity in demand should be used directly. However, the use of scenarios generated by sampling the scenario distributions allows us to compare our solutions to solutions obtained using exact methods (that cannot be obtained using continuous distributions). Unfortunately, the change from continuous to discrete distributions means losing of some of the information inherent in the underlying problem. The information loss happens during the scenario generation process, where the underlying distribution is being represented by relatively few scenarios. Scenarios are created similar to Lium, Crainic, and Wallace (2007a, 2007b) who based their scenario generation on triangular distributions for the demands. The use of a similar approach to test our algorithms is motivated by the opportunity to address real-life size instances of comparable but somewhat "richer" problems.

When using scenarios in an optimization problem, one has to decide how many scenarios to use. There is no clear answer to this, but most researchers seem to agree that larger problems require a higher number of scenarios, and that the more scenarios used, the better the representation of the underlying distributions. Scenarios that represent the underlying distributions in a good way should also give consistent results. This means that, creating multiple sets of scenarios from the same distribution should produce the same objective function value for all the sets. This is refereed to as in-sample stability by Kaut and Wallace (2007). For us, the consequence of having in-sample stability is that the objective function value will not depend on which scenario tree we choose, but on the underlying stochasticity of the demand and our ability to solve the problem. Experience has

shown that in-sample stability improves with the use of more scenarios and is dependent on the number of commodities ($|K|$) considered. Previous work by Lium, Crainic, and Wallace (2007a, 2007b) shows that in-sample stability is achieved using approximately $|K| + 10$ scenarios for similar problems. Though addressing quite similar problems they did not integrate issues such as costs of handling and cost storing of goods. Unfortunately, increasing the number of scenarios implies an increase in the size of the problem. For problems that are difficult to solve this could mean having to choose either a poor solution to a good description of the problem, or a good solution to a poor description of the problem. We address this issue by letting each problem be represented by several sets of scenarios with different cardinalities. This allows us to compare how the number of scenarios affects the search time as well as the resulting objective function values. Ideally, this should yield good solutions either because the use of few scenarios provides us with a smaller problem or because the use of many scenarios gives a better description of the problem. Computational results are shown in Sections 3 and 4.

2. Proposed method

The last two decades have seen a great increase in computation power. Nonetheless, several combinatorial problems are NP-hard and thus intractable for large instances using exact methods. For this reason, alternate techniques such as metaheuristics have received considerable attention from researchers as well as practitioners. Problems that earlier were considered impossible to solve can now be solved to near optimality within minutes or hours.

We have developed a method for finding good solutions to the Stochastic Service Network Design Problem by combining exact and heuristic methods. The heuristic is based on Local Search and changes of neighborhood in different phases of the search. A solution is defined as a set of active arcs in the network which describes the possible routes of the vehicles over the time horizon. The search space will then be all combinations of arcs that describe a solution which can be served by the vehicles. The neighborhood of a solution will be all solutions that can be reached by performing particular transformations to the current solution (the so-called "move"). When evaluating a solution, one has to consider two different parts in calculating the cost. First, the cost of operating the vehicles according to the active arcs in the network, and second, the cost of delivering the commodities by moving the freight along paths, including the cost of outsourcing the part of the commodities which cannot be delivered by regular services.

The heuristic we propose starts at an initial solution and moves from one solution to another using an initial neighborhood. Then, after a while, the search stagnates and does not improve for a number of iterations. The neighborhood is then changed and the search continues with the new neighborhood. The search may then improve for some iterations, but after a while it may stagnate again, and the search will swap back to the original neighborhood and continue with it. The search being in a different position of the search space, the initial neighborhood may yield possible improvements. The swapping of neighborhoods is done continuously during the search until a stopping criteria is fulfilled. The method utilizes the best from each neighborhood, focusing the search in some promising areas with one neighborhood and exploring these new areas with the other neighborhood. The idea of changing neighborhoods is inspired by the Variable Neighborhood Search introduced by Mladenović and Hansen (1997).

The heuristic considers possible new solutions by changing the active arcs in the network, and the flow of the freight has to be calculated along the possible paths for each commodity in that network. The optimization of the flow distribution could be done by exact methods, but this is very

time consuming. In fact, the need for exact optimization is not very important during the search as most solutions are intermediary steps in the search for better solutions. To speed up computations, a greedy method is thus proposed to find indications of good moves, exact optimization being limited to solutions that are candidates for the overall best solution found during the search.

2.1 Initial solutions

Most metaheuristics require a starting solution that they can use as a basis for further improvement. In this paper we consider three different initial-solution procedures, all with a fixed number of vehicles.

2.1.1 All vehicles parked

Having decided on the number of vehicles to use, we start with the vehicles parked for the entire time horizon, evenly distributed among the terminals. In such a solution no commodities will be delivered and all transportation has to be outsourced. When the search starts, the solution will build itself from scratch with no predefined structure.

2.1.2 Randomized initial solutions

The second procedure lets vehicles follow randomly selected paths. Each vehicle starts at a randomly selected terminal, and then goes to another randomly selected terminal, where it arrives v periods later, and so on. At the end of the planning horizon, each vehicle is repositioned to its starting point.

2.1.3 Demand-driven initial solutions

More sophisticated initial solutions can be created by letting the underlying demand be taken into account in the process. We define the *commodity flow factor* as the average demand for each commodity for all the scenarios divided by the number of periods between $\sigma(k)$ and $\tau(k)$. A commodity with a large average demand to be delivered within a few time periods will have a large flow factor, while a commodity with a lesser demand and more periods available will have a smaller flow factor. Sorting the commodities in descending order according to this factor will prioritize them accordingly.

We also define a *period flow factor* for each period to be the sum of the commodity flow factors for the commodities it is possible to send in the period. Then the periods can be sorted ascending according to this period flow factor to decide which periods are least demanded by the commodities.

The idea behind the demand-driven initial solution is to find the most dominant commodities, and construct a solution that transports those commodities during the possible time periods that are least demanded by the other commodities. An initial solution can then be built by introducing direct service arcs in the network for one commodity at a time for the least demanded time period available.

The demand-driven initial solution will be constructed using the following algorithm:

1. *For each commodity k, find the commodity flow factor.*
2. *Sort the commodities in descending order of their commodity flow factors.*
3. *For each period, find the period flow factor.*
4. *Sort the periods in ascending order of their period flow factors.*
5. *Initialize all vehicles by marking them available to use in all the periods.*

6. *Construct the paths for the vehicles.*
   a. *Loop through all commodities from the highest commodity flow factor to the lowest.*
   b. *Loop through all periods from the lowest period flow factor to the highest.*
   c. *If it is possible to send the current commodity in the current period, loop through all the vehicles.*
   d. *Add the direct connection from origin to destination for the current commodity to a possible vehicle. Connections which can be linked with other existing connections are prioritized. Once a connection is added, return to 6a for the next commodity.*
7. *Go through the routes of all the vehicles and check if any of them is still not used in any period. If true, change the position to the same node as in an adjacent period.*

2.2 Neighborhood structure and moves

A solution is defined as a set of active arcs in the network and the associated flow distribution. We represent the first part by an integer vector describing the position of each vehicle, i.e., the index of the terminal where the vehicle is, at each time period. Once the positions of the vehicles are known, the active arcs are the services between the corresponding terminals. The search space is then defined as all vectors of vehicle positions corresponding to feasible solutions given the fleet of vehicles.

Neighborhoods condition how the search progresses and performs. Moving from one solution to a neighboring one is performed by changing the value of one or more variables in the vehicle-position vector thereby obtaining a different solution by shifting capacity from some arcs (services) to others. The neighborhoods used are described in the following, while the evaluation procedures are detailed in Section 2.3.

2.2.1 Random-period neighborhood

One way to define a move in a service network design problem is simply to change the position of one of the vehicles in one time period. We have named this type of neighborhood a *Two-Terminal Swap Neighborhood* (*TTS*). An example of this neighborhood can be seen in Figure 1 where arcs represent the integer decision variables (the movement of a vehicle) for three different solutions. The solid arcs represent the services offered in the current solution, while the dashed and the dotted arcs represent the corresponding services in two neighboring solutions. The solid, the dashed, and the dotted arcs form direct paths starting in the first period in terminal 2 and ending at the same terminal in the last time period. By enforcing that these paths must end up in the same terminal they started from, we are sure that the schedule for the vehicle is feasible in all moves and that the constraint sets (2) and (3) are not violated. A move in this setting implies a shift of Θ units of capacity from the path in the current solution to a neighboring solution with the result of possibly closing one or more arcs if their capacity is equal to Θ.

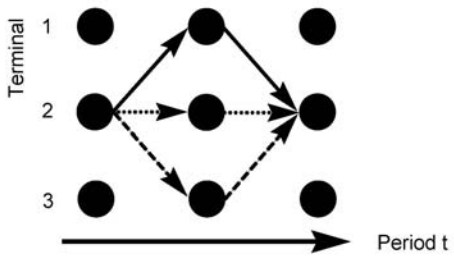

Figure 1. The current solution and two of its neighboring solutions

The TTS neighborhood could be very large for large instances and a full evaluation of all possible moves would be very time consuming. Restricting the neighborhood to a subset of moves only would speed up the search, which we prefer even though some promising moves might be overlooked.

One approach to restrict this neighborhood is to focus on one randomly selected time period and limit the moves to this period. Thus, every vehicle can be moved to all nodes other than the current one in the selected time period and the number of neighboring solutions to evaluate is ($|N|$-1) x *Number of Vehicles*. The type of move illustrated in Figure 1, where relatively short paths are considered, makes the neighborhood quite small, thereby making it relatively fast to evaluate all possible moves.

The variant of the TTS-neighborhood which evaluates the possible moves in one randomly selected period only is called *Random-Period Neighborhood* (*RPN*). An overview of a search with RPN is as follows

1. *Chose one time period randomly.*
2. *Loop through all vehicles.*
    a. *Try to change the position of the current vehicle in the chosen time period to all possible positions;*
    b. *Evaluate the neighbor solutions found in 2a;*
3. *Select the best neighbor found in 2.*

A special case in the Random-Period Neighborhood is when the last time period is chosen. The mathematical model allows vehicles to end up at another customer than it started from as long as the total number of vehicles at each node is the same in the last period as it was in the first. In these cases the Random-period Neighborhood will have an extra control if the solution is feasible in the case where the last time period is chosen and only the feasible solutions will be evaluated.

Notice that we did not include tabu-like mechanisms to prevent the search from cycling (Glover and Laguna 1997). Indeed, preliminary experimentations indicated that the high degree of randomness inherent to these neighborhoods made them superfluous.

2.2.2 Demand-based neighborhood

While the neighborhood structure described in Section 2.2.1 has the advantage of providing a somewhat limited neighborhood that can be evaluated quickly, it also has the drawback of being myopic, bearing the risk of focusing too strongly on parts of the network. We therefore explore an alternate neighborhood structure based on paths covering more time periods to try to overcome these myopic tendencies. An example of this neighborhood can be seen in Figure 2, where the solid arcs constitute the current solution while the dashed arcs represent one of its potentially many neighbors.

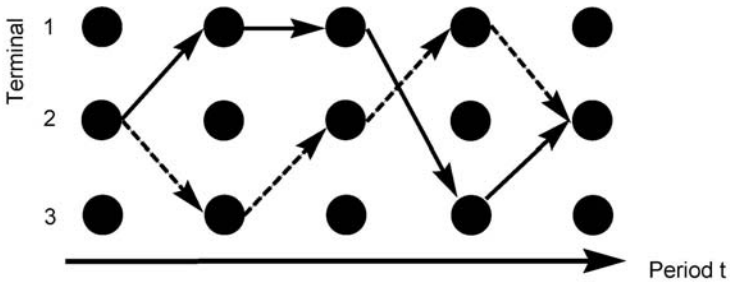

Figure 2. Neighborhood consisting of longer paths

Enumerating and evaluating all possible neighbors is usually a daunting task due to the size of the neighborhood, which grows very rapidly with the number of periods and nodes. We thus evaluate only the most promising neighbors.

To use this approach efficiently, we need to identify certain characteristics of neighboring solutions that are especially promising. Thus, adding capacity to a solution where a number of commodities rely heavily on "expensive" ad-hoc capacity increase would probably result in a better solution. One may therefore identify the terminals between which the largest number of commodities make use of ad-hoc capacity and insert a service (vehicle) instead. Of course, vehicle flow-conservation constraints must continue to be enforced. This is illustrated in Figures 3 to 5. Figure 3 shows the case of ad-hoc capacity use between terminals 1 and 2 at period 2, while a vehicle is held at terminal 3. In Figure 4, the vehicle is moved to serve the commodities between terminals 1 and 2, removing the need for ad-hoc capacity. This move makes for an infeasible schedule for the vehicle. Feasibility is recovered by repositioning the vehicle in periods 1 and 3 as illustrated in Figure 5.

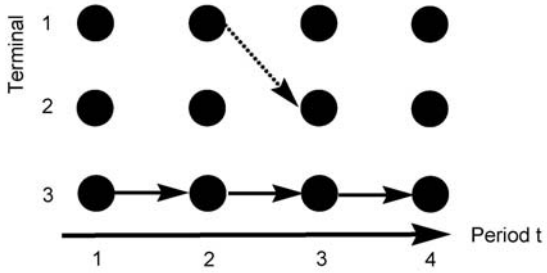

Figure 3. Ad-hoc capacity use from terminal 1 to terminal 2 (dotted arc), while a vehicle is held at terminal 3 (solid arcs)

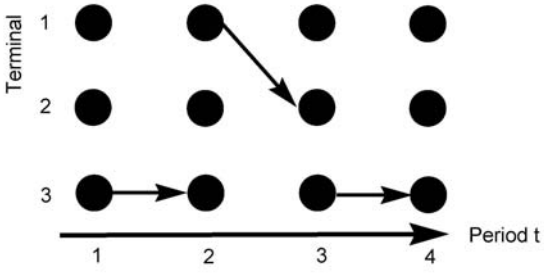

Figure 4. Move of vehicle to serve from terminal 1 to terminal 2 and elimination of ad-hoc capacity use

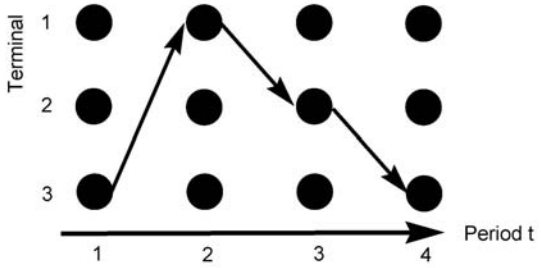

Figure 5. Feasibility recovering by repositioning the vehicle in periods 1 and 3.

Selecting the neighbors that are to be evaluated is done by identifying the commodity with the highest expected ad-hoc capacity utilization cost in the current solution. Then, the demand-based neighborhood consists of the new direct arcs that can be inserted into the solution and service that commodity. Thus, the possible moves are found by inserting an arc from the origin to the destination of that commodity for each of the periods the commodity is available and for each of the vehicles available as shown in Figures 3, 4, and 5. Figure 6 illustrates the search based on this neighborhood.

```
Initialization;
iteration = 0;
while iteration < max iteration do
    Find the commodity k with the highest expected
    level of utilization of ad-hoc capacity in the current solution;
    if no ad-hoc capacity used then
        set k = random commodity;
    end
    for first vehicle to last vehicle do
        startTime = σ(k);
        notDeliv = τ(k);
        if notDeliv < 0 then
            notDeliv = notDeliv + T;
        end
        while startTime ≠ notDeliv do
            Create candidate by:
              - inserting an arc from the origin to destination at startTime;
              - recovering feasibility regarding vehicle flows by connecting the
                inserted arc to the path of the vehicle;
            Evaluate the candidate;
            if cost of candidate < cost of best neighbor solution then
                best neighbor solution = candidate;
            end
            startTime ++;
        end
    end
    if cost of best neighbor solution < cost of best found solution then
        best found solution = best neighbor solution;
    end
    current solution = best neighbor solution;
    iteration ++;
end
```

Figure 6. Algorithm 1: A demand-based neighborhood search

2.3 Evaluating moves

Three factors have to be considered when evaluating moves: the cost of operating the resulting schedule (the cost of the design), the expected cost of moving freight using this schedule, and the expected cost of ad-hoc capacity utilization. The cost of operating the schedule is given by (1a) and

evaluating it is straightforward. All that has to be done is to calculate the cost of operating vehicles between terminals or holding them at terminals according to the given schedule. To calculate the cost of moving the freight in the network (term 1b), one has to solve the flow distribution problem. Once the resulting path flows are known, the cost of using ad-hoc capacity (term 1c) may be computed directly. Evaluating the cost of the flow distribution is the potentially most time-consuming part of this procedure.

The exact evaluation of the flow distribution can be done in polynomial time by formulating an LP with (1b) and (1c) as objective function and (3) - (9) as constraints (with fixed integer variables). This is, however, a computationally costly procedure: preliminary tests have shown it to be the most time-consuming component of the entire method.

We therefore developed a greedy heuristic that finds good new neighboring solutions substantially faster than the exact method. Preliminary tests indicate that the solutions of the greedy method are usually almost as good as those obtained using the exact method. The greedy method was on average 40 times faster than the exact optimization on our test instances.

The method proceeds as follows. For the move being evaluated, we generate a list of paths for each commodity from its origin to its destination. We sort the paths in increasing order of their cost. In addition, the commodities are sorted by dominance, i.e., the average demand from the scenarios divided by the number of time periods between when the commodity is available and when it should be delivered. The most dominant commodity is then assigned to the cheapest path that can be used to move it. This is continued until all demand in all scenarios is satisfied using this same path or the path is saturated. If demand exceeds the capacity of the path in one or more scenarios the exceeding freight is shifted to the second cheapest path for this commodity (and so on). The subsequent commodities are treated similarly until all commodities have been examined. Commodities that were not assigned to a path use ad-hoc capacity. The cost associated to the paths and the ad-hoc capacity is then computed.

3. Evaluating the algorithmic components

Several algorithmic components have been introduced in the previous section. The worth of each and of the possible combinations thereof must be evaluated to yield the final metaheuristic design for the class of problems we contemplate. This is the scope of this section, which also introduces the problem instances used throughout the experimental phases.

3.1 The test cases

Service network design research has, with few exceptions, focused on deterministic problem variants. Consequently, test instances that integrate stochasticity are to the best of our knowledge not publicly available. We therefore present 55 new test instances that take demand stochasticity into account. The instances are available online at: http://www.himolde.no/OR-problems.

3.1.1 Nodes, planning horizon, O-D pairs, and demand

We created instances as diverse as possible. Using from 6 to 30 terminals, we generated problem instances from small enough to be solved using exact methods to problems equivalent in dimensions to those faced by medium-sized European less-than truckload (LTL) carriers.

An important aspect of any time-dependent problem is the planning horizon. The more periods in a planning horizon, the better the solutions that can be obtained. Unfortunately, this has the consequence of increasing the size of the problem. Our instances have a planning horizon of either

7 or 14 periods. The problem size also depends on v, which is the number of periods required to travel from one node to another. The model formulation in Section 1 allows for v taking any integer value $1 \leq v \leq T$. In our test instances, however, $v = 1$, which makes our problems consist of $|N|^2 \times T$ integer variables.

The time span between when a commodity becomes available ($\sigma(k)$) and when it has to be delivered ($\tau(k)$) at its destination typically affects the costs and the number of paths it can follow. This time span was randomly selected with an average of 3 or 5, according to the instance. Commodity demands were computed by discretizing triangular distributions. Each commodity has a demand lower than the vehicle capacity Θ. The test instances were constructed to avoid that few commodities dominate the solution through, for example, a significantly higher demand than the other commodities.

3.1.2 Cost structures

In the transportation industry, a large fraction of the costs is associated to operating a fleet of vehicles, trains, aircraft, ships, trucks, and can be divided into several components such as cost of capital, maintenance, fuel, depreciation, wage costs, wear and tear, etc. These costs are closely connected to the distance or travel time between the terminals. The cost (distance) for our test instances are given by symmetric matrices that are strongly inspired by the cost/distances found in the E-016-03 and E-031-09 vehicle routing test instances by Christofides, Mingozzi, and Toth (1981) and Hadjiconstantinou, Christofides, and Mingozzi (1995), respectively.

The instances also include costs for handling freight at intermediary terminals, holding vehicles at terminals, holding freight at intermediary terminals, moving freight with the particular vehicles considered, and adding ad-hoc capacity. These costs were estimated as proportions to the traveling costs of the particular instances.

3.1.3 Number of scenarios

Increasing the number of scenarios to describe the underlying distributions provides a better description of the actual problem, *ceteris paribus*. Unfortunately, it also increases the size of the instance being solved. Experience from solving similar problems using exact methods shows that the number of scenarios used has a significant impact on the solution time. We therefore balance the quality of the problem description with the problem size by using 3 different discretizations of the underlying distributions consisting of 20, 60, and 90 scenarios, respectively, for each problem instance. Our experiments have showed that, while the number of scenarios has an impact on the memory and the computation time required solving a problem instance, the objective function values of the instance obtained with different number of scenarios do not differ very much. We therefore do not generate a larger number of scenarios.

3.2 Calibration and variant evaluation

This section presents a comparative evaluation of the algorithmic variants introduced earlier on, the alternative initial solution procedures and neighborhood definitions, in particular. The resulting heuristic that we propose is shown in Section 3.2.5.

3.2.1 Comparison of initial solution procedures

Comparing the objective-function values of the initial solutions produced by the three procedures of Section 2.1 shows the demand-driven solution outperforming the others significantly. A parked initial solution implies that all the commodities use ad-hoc capacity and the costs of such solutions

are 44.79 times higher on average than the best solutions for the problem instances described in Section 3.1. A randomly-created initial solution is also having most commodities using ad-hoc capacity. On average for the test problem instances, random-initial solutions have a cost 32.27 times higher than the best solutions. Demand-driven initial solutions deliver many commodities using the existing fleet, their costs being on average only 3.08 times worse than the best solutions for the instances in our test set. The experiments show, however, that the results change as the search progresses and the number of iterations increases.

After 5000 iterations performed on the test instances, the results starting from the parked initial solutions were on average 2.86% worse than the best known solutions, compared to 3.53% and 5.90% worse starting from the random and demand-driven initial solutions, respectively. It thus seems that, the demand-driven strategy drives the search to local optima containing combinations of services from which the metaheuristic cannot escape easily. We conclude that the demand-driven initial solution should be chosen to obtain a "good" solution fast, but a parked initial solution is to be preferred for a longer search. We used parked initial solutions for our subsequent tests.

3.2.2 Limiting the search space

The model of Section 1 does not specify the route of each vehicle and yields the arcs (services) used and the loads on these arcs. It is thus possible that solutions exist where vehicles end up at nodes different from the ones they started, the only requirement being that the vehicle-balance conditions be verified at all nodes and time periods. This means that valid solutions may exist where vehicles swap routes from one week to another. Commodity path costs should be higher in this case. The evaluation of these costs would be problematic, however, without the knowledge of the vehicle routes.

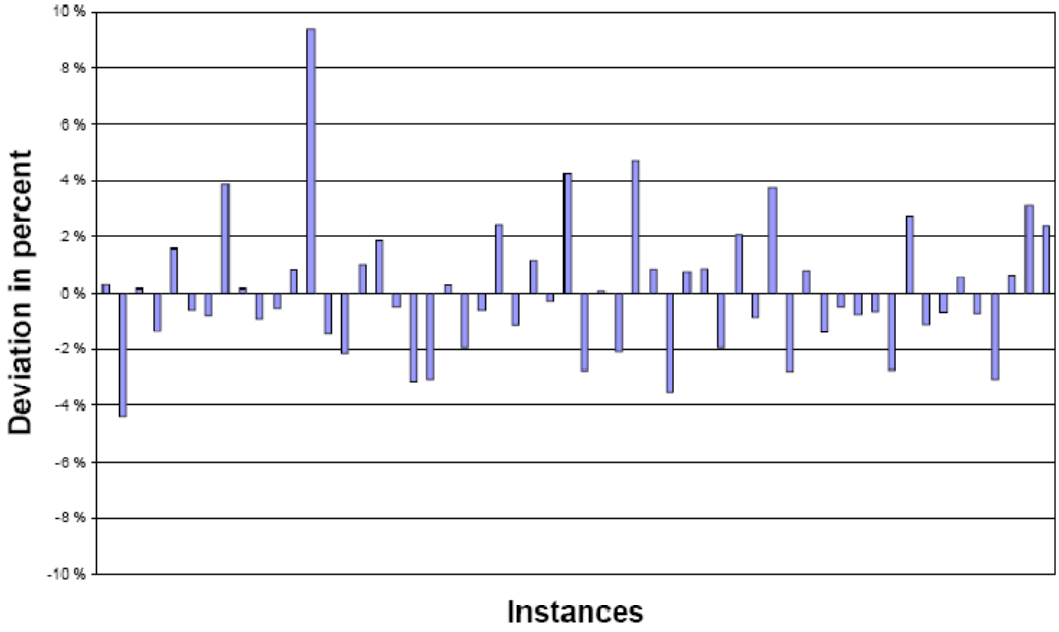

Figure 8. Impact on search performance of enforcing each vehicle to end up in the same terminal as it started (Λ)

These difficulties result in the described neighborhoods and moves not effectively addressing solutions where vehicles swap routes, generally leading to less good solutions than when vehicles are not permitted such swaps. We identify with the parameter Λ the limitation of the search space to the case where all vehicles have to end up at the same node they started from. Λ may be used in all neighborhoods and tests were run both with and without it.

Figure 7 displays the difference between the results of a search with and without Λ on all instances listed sequentially (detailed results may be found in Tables 2 and 3 in the Appendix). The objective-function value obtained by the search with Λ was divided by the corresponding value obtained without Λ, and 1 was subtracted from the result. Then, instances with a positive deviation found a better result using Λ, while those with a negative deviation found the best result not using Λ. As we can see in the figure, the variance can be significant for some instances. On average the solutions found with the search limitation Λ are 2.6% better than the solutions found without.

3.2.3 Number of vehicles

The model of Section 1 does not constrain the number of vehicles to be used. Solving the model using an exact method would yield the optimal required number of vehicles. A heuristic cannot provide such a figure. We can, however, use the heuristic imposing different fixed number of vehicles and, thus, approximating the minimum number of vehicles required by a given instance.

Figure 6 shows the results for the instance SND80C30N14D5T with 20, 60, and 90 scenarios and different numbers of vehicles. The tests show best results for this instance in the 8 to 11 vehicle range, the lowest values being obtained, by a small margin, for 10 or 11 vehicles. We can also clearly see from the large improvement obtained by adding the eighth vehicle that 7 vehicles are not enough. This instance is typical, test results displaying a similar behavior on al instances.

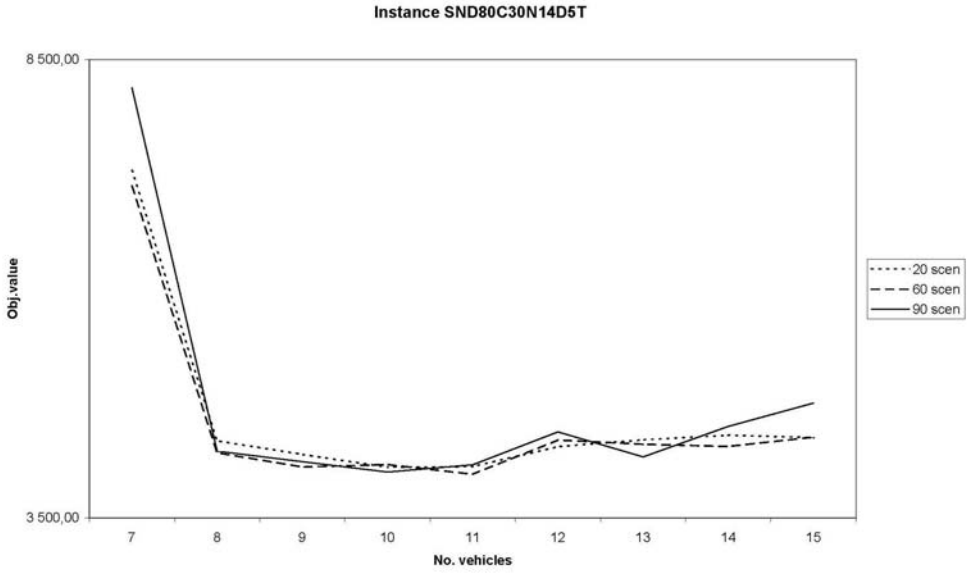

Figure 9. Impact of the numbers of vehicles and scenarios on the solution cost

The cost of ad-hoc capacity is relatively high compared to the cost of operating the vehicles. Hence, a solution where a significant part of the commodities requires ad-hoc capacity will be more expensive than a solution using mostly the existing vehicles. Typically, the number of vehicles where the graph flattens corresponds to the number needed to be able to deliver all commodities in a majority of scenarios. Ad-hoc capacity might still be needed in some scenarios, but better

solutions can then be created by adding more vehicles. Thus, for the number of vehicles used in the best found solutions the amount of ad-hoc capacity used is normally close to zero for all commodities.

A direct link from the origin to the destination terminal is the cheapest way to move the commodity freight. Hence, when the number of vehicles is sufficient, solutions are created with direct links (services) able to transport the commodities in most scenarios. Alternate paths have to be created, however, to cope with scenarios with large demand. When the number of vehicles is too small to create enough direct links, some commodities are transported using longer service paths. In most such situations, the vehicle routes are such that the commodities stay on the same vehicle for several periods before arriving at destination. Paths that include more than one vehicle are generally avoided because of the extra cost of transferring the goods, but for some commodities such paths are possible alternatives.

For most problem instances in our test set, the number of vehicles was sufficient to create direct links for a majority of the commodities and most of them were transported only one period. Alternative paths with up to three periods were however used in some of the scenarios. Paths spanning more than three periods and paths including more than one vehicle were used by only 1.2% of all the commodities and only in the scenarios with an amount much larger than the average.

3.2.4 Evaluation of moves

The greedy algorithm makes the search proceed much faster than when the exact optimization of the freight flow is used. Experimentally, for up to 100 iterations, no significant difference in the quality of the results may be observed between a search using the greedy and a search using the exact optimization. The results after 100 iterations are on average 3.65 times poorer than the best results found for the test problem instances. In some cases, the same neighbor was selected by both methods but even when the search trajectories differed, the greedy method produced solutions with an objective value that did not differ by more than 4.24% on average above those found with a search using exact optimization. Comparing the computation time required by the two methods indicates a significant difference, however. For the instances in our test set, a search with the exact optimization uses on average more than 40 times more time than a search with the greedy method.

Notice that, during the search, it is not vital to find the exact solution value for each temporary solution evaluated, but rather to find indications whether a move seems good or not and compare the alternative candidates. For this purpose, a greedy and fast algorithm for distributing the freight can be almost as useful as an exact optimization method which uses much more time.

The exact optimization method should then be limited to particular candidate solutions only. In our case, we limit the exact method to the solutions found by the greedy algorithm with an objective-function value within 5% of the value of the best known solution and after the search has progressed for more than 60% of the total number of iterations. This strategy increases the computational time by only some 6% compared to a pure greedy search. This is considered acceptable to provide optimal flows for the best solutions found.

3.2.5 Evaluation of neighborhoods and final heuristic

The preliminary experiments have shown that a local search using the neighborhoods described in Sections 2.2.1 and 2.2.2 performs very differently.

A search using the random-period neighborhood strategy for 5000 iterations is rather time consuming due to the size of the neighborhood being evaluated. However, it provides gradual

improvements during all phases of the search and produce results on average 13% above the best known solutions. A search conducted for the same duration with the demand-based neighborhood converges very early in the search and the results produced are on average 44% above the best-known solutions for our test instances. This strategy seems get often stuck in a local optimum and has problems escaping from it due to the limited number of possible neighbors.

By combining these two neighborhoods, it is however possible to utilize their best features. The demand-based neighborhood would identify which commodities use the most ad-hoc capacity and focus the search around these for a limited number of iterations. When the improvement stagnates, the search swaps to the random-period neighborhood. Then after a given number of iterations without improvement, the search swaps back to the demand-based strategy, and so on.

We tested this mixed-neighborhood strategy by swapping between the two neighborhood after 100 iterations without improvement for the RPN and 10 iterations without improvement for DBN. This strategy gave significantly better results than using only one of the neighborhoods. The global best solutions to our test instances were obtained by selecting the best out of a large number of solutions found with a number of runs with different parameter settings. The results found by the mixed-neighborhood strategy were on average less than 3% above these global best solutions. We therefore selected the mixed-neighborhood for the final form of the metaheuristic we propose and for all experiments. The search using the mixed-neighborhood and switching between the neighborhoods corresponds to the Algorithm 2 presented in Figure 10.

```
initialization;
RandomPeriod = true;
iteration = 0;
iteration RPN = 0;  // Counter for iterations with random-period Neighborhood
iteration DBN = 0;  // Counter for iterations with demand-based Neighborhood
while iteration < max iterations do
    if iteration RPN > φ then
        RandomPeriod = false;
        Iteration DBN = 0;
    end
    if iteration DBN > β then
        RandomPeriod = true;
        iteration RPN = 0;
    end
    if RandomPeriod = = true then
        search using random-period neighborhood search strategy;
        if cost of best neighbor solution < cost of best found solution then
            best found solution = best neighbor solution;
            iteration RPN = 0;
        else
            iteration RPN ++;
        end
    end
    if RandomPeriod = = false then
        search using demand-based neighborhood search strategy;
        if cost of best neighbor solution < cost of best found solution then
```

```
            iteration DBN = 0;
            best found solution = best neighbor solution;
        else
            iteration DBN ++;
        end
    end
    move to best neighbor solution;
end
```

Figure 10. Algorithm 2: The mixed-neighborhood search

The metaheuristic that we propose may then be summarized as follows:

1. *Create an initial solution where all vehicles are parked at a terminal.*
2. *Set φ = 100, β = 10, and the max number of iterations = 5000.*
3. *Perform a search with greedy optimization and variable neighborhoods as shown in algorithm 2.*
   a. *If iterations = ψ*
      i. *Perform an exact optimization of the flow of the best found solution so far.*
      ii. *Save the solution value in BestSolVal.*
   b. *If iterations > ψ and the greedy value of the best neighbor solution is within α% of BestSolVal*
      i. *Perform an exact optimization of the flow on the best neighbor solution and save the solution value in BestNeighVal.*
      ii. *If BestNeighVal < BestSolVal, set BestSolVal = BestNeighVal and save the solution as the best found so far.*
4. *Report best found solution and stop.*

A local search from a demand-driven initial solution with a full TTS neighborhood will normally find a local optimum within 30 iterations for the instances in our test set. With a parked initial solution and a mixed neighborhood, the local optimum is not that easy to identify because of the high degree of randomness in the search, but the search starts to level before 500 iterations even on the largest instances. Figure 11 illustrates the general observed behavior by displaying the improvement in the objective-function value for instance SND80C30N14D5T90S. The first 125 iterations are omitted for scaling reasons.

Preliminary tests show that with the mixed neighborhood, most improvements will be achieved before the search reaches 4000 iterations. Even though for some instances later improvements will be seen, this number is considered sufficient to reach sufficiently good solutions. As indicated previously, we run the greedy search for 4000 iterations. The search is then be extended by 1000 iterations, for a total number of 5000. During these 1000 iterations, all solutions with a greedy objective-function value within 5% of the value of the best known solution will be optimized to see if they are better than the best solution found earlier. We denote ψ, the minimum number of iterations before the exact optimization kicks in and α, the maximum deviation from the objective-function value of best found solution used to decide if an exact optimization should take place.

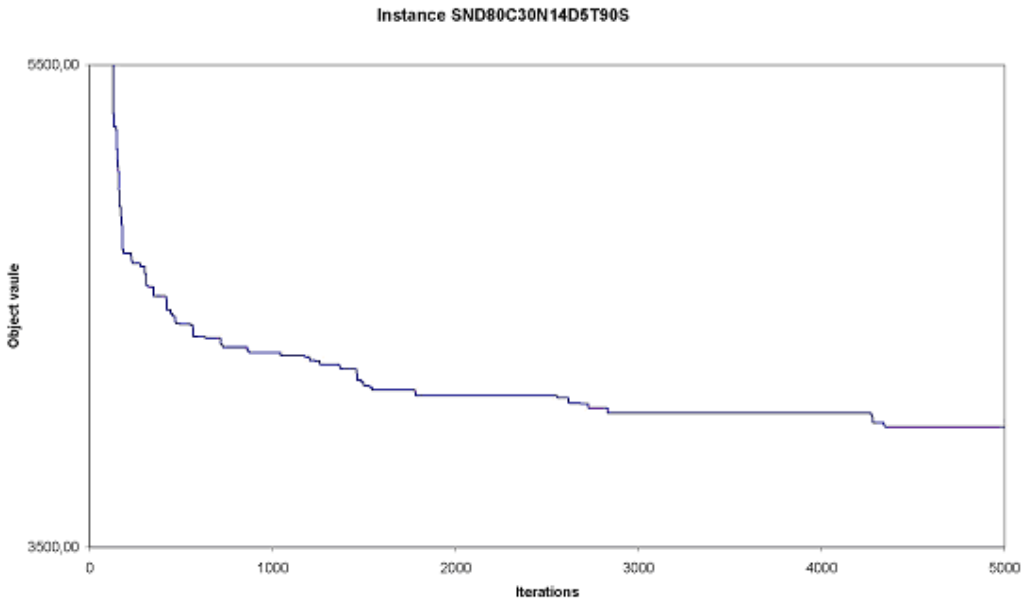

Figure 11. Example of improvement in the objective-function value during the search

4. Computational Experiments

The metaheuristic described in Section 3.2.5 was coded in C++ and run on a Pentium 4 2.40 GHz computer with 512 MB of RAM running Windows XP. The results are presented in Table 1 and in Tables 2 and 3 (in the Appendix).

Since we present the first experiments on these problem instances comparison with other methods is difficult. The smallest instance, SND14C6N7D3T20S, is however solved to optimality by exact methods using the model in Section 2. On this small problem, the solver CPLEX 9.0 using the network simplex algorithm and default options required about 9.5 hours to find the optimum, while our heuristic used 140 seconds to run for 5000 iterations. The heuristic yielded an objective value of 6057.50 after 5000 iterations, representing a deviation of 5.2% with respect to the optimal solution of CPLEX. Running the heuristic for 10000 iterations with the same parameters reduced the optimality gap to 3.3% only. Testing larger instances using CPLEX shows the limits of exact methods. The test on SND40C16N7D3T20Sa run for one week on a Pentium 4, 3.2GHz, 2GB RAM without finding the optimum. The best result found by CPLEX during this period, was 121740.3 which is more than 5 times higher than the result achieved by the metaheuristic in 4288 seconds for 5000 iterations.

Table 1 displays the results for the two instances. The table indicates the number of commodities ($|K|$), the number of terminals ($|N|$), the number of time periods in the planning horizon (T), the average available time for delivery of the commodities ($\Delta\bar{t}$), the number of scenarios ($|S|$), the minimum number of vehicles required to avoid ad-hoc capacity utilization (*Min. Veh.*), the number of vehicles used in the best found solution (*No. Veh.*) and their respective cost (*Obj. F. Meta.*), as well as the objective-function values for the metaheuristic (the value after 10000 iterations is indicated for the first instance) and CPLEX.

Table 1. Best found solutions by the metaheuristic and CPLEX

Instance	$\|K\|$	$\|N\|$	T	$\Delta\bar{t}$	$\|S\|$	Min. Veh.	No. Veh.	Obj. F. Meta.	Obj. F. CPLEX
SND14C6N7D3T20S	14	6	7	3	20	3	3	5948.5	5759.0
SND40C16N7D3T20Sa	40	16	7	3	20	8	8	24264.3	121740.3

We do not have a comparison point for the other problem instances and it is thus hard to evaluate how good the solutions found by our heuristic are. We can, however, compare them to each other and draw some conclusions from that. The set of test instances was actually created to make it possible to analyze each of the parameters individually. Thus, a number of instances differ in the number of scenarios only, with the same expected value of the demand for all scenarios.

When examining the instances with the same number of commodities, terminals, average time for delivery, and scenarios, but with different planning horizons, we find, as expected, that a seven-day planning horizon requires more vehicles than a fourteen-day one. On average, the minimum number of vehicles needed for the seven-day instances were 48% higher than for the corresponding fourteen-day instances. The differences in the length of the routes, however, made the solution value of the seven-day instances on average 88% of that the fourteen-day instances. The size of the neighborhood, which is the factor deciding most of the searching time per iteration, is not affected by the number of time periods. The CPU time used for the fourteen-day instances is, however, about 18% above that of the instances with a seven-day horizon.

From the point of view of the difference in time periods between the delivery deadline and the availability of the commodity, we find that instances with a longer difference generally need fewer vehicles than instances for which this difference is small. An average delivery time of five periods will, as a basic rule, save one vehicle and the solution value is on average 4% lower compared to those instances with an average of three periods delivery time.

When examining the impact of the number of scenarios, we find that using 20 or 90 scenarios yields approximately the same objective-function value. With 60 scenarios, we get an objective-function value 3.5% higher than the others. This is somewhat unexpected as previous experiences (Lium, 2006) show that an increase in the number of scenarios would result either in no change or an increase in the objective-function value.

It is hard to find patterns regarding the limitation of the search (Λ). The instances where this is profitable seem to be randomly distributed among instances of all sizes.

5. Conclusions

We proposed a new metaheuristic approach for time-dependent stochastic service network design problems. We believe this to be one of the first successful metaheuristics for this class of problems.

The metaheuristic is inspired by VNS principles and uses fast approximations to the move evaluations where appropriate. Computational experiments on a large set of problem instances indicate that the metaheuristic provides the means to address large instances of this complicated class of problems within reasonable time.

Interesting research perspectives are now open in improving the methodology. One such avenue concerns the use of arc pseudo dual information to help decide where to increase/decrease capacity. The development of cooperative parallel mechanisms for this metaheuristic constitutes another way

to increase the efficiency and quality of the search. We are initiating research on some of these ideas and will report in the near future

Acknowledgements

The authors thank Øyvind Halskau for pointing out errors and helping us improve the paper. While working on this project, Arild Hoff and Arnt-Gunnar Lium were employed as Research Fellows at Molde University College (Norway). Dr. Teodor Gabriel Crainic was Adjunct Professor at Molde University College and the department of Computer Science and Operations Research of the Université de Montréal (Canada). Partial funding for this project was provided by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

Christofides, N., Mingozzi, A., and Toth, P. (1981). Exact Algorithms for the Vehicle Routing Problem Based on Spanning Tree and Shortest Path Relaxations. *Mathematical Programming* **20**, 255-282.

Crainic, T.G. (2003). Long Haul Freight Transportation. In *Handbook of Transportation Science* (Hall, R.W., ed.), pp. 451-516. Kluwer Academic Publishers, Norwell, MA, second edition.

Crainic, T.G. (2000). Network Design in Freight Transportation. *European Journal of Operations Research* **122(2),** 272-288.

Glover, F., and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

Ghamlouche, I, Crainic, T.G. and Gendreau, M. (2003). Cycle-based Neighbourhoods for Fixed-Charge Capacitated Multicommodity Network Design. *Operations Research* **51**, 655-667

Ghamlouche, I, Crainic, T.G. and Gendreau, M. (2004). Path Relinking, Cycle-Based Neighbourhoods and Capacitated Multicommodity Network Design. *Annals of Operations Research* **131**, 109-131

Hadjiconstantinou, E., Christofides, N., and Mingozzi, A. (1995). A New Exact Algorithm for the Vehicle Routing Problem Based on q-Path and k-Shortest Path Relaxations. *Annals of Operations Research* **61**, 21-43.

Kaut, M. and Wallace, S. W. (2007). Evaluation of Scenario-Generation Methods for Stochastic Programming. *Pacific Journal of Optimization* (forthcoming).

Lium, A.-G. (2006). *Stochastic Service Network Design*, PhD Thesis in Logistics. Molde University College.

Lium, A.-G., Crainic, T.G., and Wallace, S.W. (2007a). Correlations in Stochastic Programming: A Case from Stochastic Service Network Design. *Asia-Pacific Journal of Operational Research* **24(2)**, 161-179.

Lium, A.-G., Crainic, T. G., and Wallace, S. W. (2007b). A Study of Demand Stochasticity in Service Network Design. *Transportation Science* (forthcoming)

Mladenovic, N., and Hansen, P. (1997). Variable Neighborhood Search. *Computers and Operations Research* **24**, 1097-1100.

Appendix

The following two tables display detailed results by instances described by their name (column 1). For each instance, the following information is indicated: the number of commodities ($|K|$), the number of terminals ($|N|$), the number of time periods in the planning horizon (T), the average available time for delivery of the commodities ($\Delta\bar{t}$), the number of scenarios used ($|S|$), whether yes/no the vehicles must return to where they start (Λ), the minimum number of vehicles required to avoid ad-hoc capacity utilization (*Min. Veh.*), the number of vehicles used in the best found solution (*No. Veh.*) and their respective cost (*Obj. F. Meta.*).

Table 2. Best found solutions – 7 days planning period

Instance	$\|K\|$	$\|N\|$	T	$\Delta\bar{t}$	$\|S\|$	Λ	Min. Veh.	No. Veh.	Obj. F. Meta.
SND14C16N7D3T20S	14	16	7	3	20	Y	4	4	785.57
SND14C16N7D5T20S	14	16	7	5	20	N	3	3	715.42
SND14C30N7D3T20S	14	30	7	3	20	Y	4	4	780.62
SND14C30N7D5T20S	14	30	7	5	20	N	3	4	706.96
SND40C16N7D3T20S	40	16	7	3	20	N	8	8	2199.67
SND40C16N7D3T90S	40	16	7	3	90	Y	8	9	2176.45
SND40C16N7D5T20S	40	16	7	5	20	Y	7	7	2047.34
SND40C16N7D5T60S	40	16	7	5	60	N	7	8	2071.66
SND40C16N7D5T90S	40	16	7	5	90	N	7	8	2056.52
SND40C30N7D3T20S	40	30	7	3	20	Y	8	8	1744.14
SND40C30N7D3T60S	40	30	7	3	60	Y	8	8	1707.36
SND40C30N7D3T90S	40	30	7	3	90	N	8	9	1760.65
SND40C30N7D5T20S	40	30	7	5	20	N	7	8	1602.77
SND40C30N7D5T60S	40	30	7	5	60	Y	7	7	1611.62
SND40C30N7D5T90S	40	30	7	5	90	Y	7	8	1604.78
SND80C16N7D3T20S	80	16	7	3	20	N	14	14	4138.12
SND80C16N7D3T60S	80	16	7	3	60	N	14	14	4050.01
SND80C16N7D3T90S	80	16	7	3	90	N	14	14	4110.83
SND80C16N7D5T20S	80	16	7	5	20	Y	12	13	3694.77
SND80C16N7D5T60S	80	16	7	5	60	N	12	13	3699.18
SND80C16N7D5T90S	80	16	7	5	90	N	12	13	3724.42
SND80C30N7D3T20S	80	30	7	3	20	Y	16	16	3941.83
SND80C30N7D3T60S	80	30	7	3	60	N	16	16	4014.35
SND80C30N7D3T90S	80	30	7	3	90	Y	15	16	4003.77
SND80C30N7D5T20S	80	30	7	5	20	N	13	14	3414.57
SND80C30N7D5T60S	80	30	7	5	60	Y	13	14	3414.37
SND80C30N7D5T90S	80	30	7	5	90	N	13	14	3454.48

Table 3. Best found solutions – 14 days planning period

Instance	$\|K\|$	$\|N\|$	T	$\Delta\bar{t}$	$\|S\|$	Λ	Min. Veh.	No. Veh.	Obj. F. Meta.
SND14C16N14D3T20S	14	16	14	3	20	Y	3	3	924.72
SND14C16N14D5T20S	14	16	14	5	20	N	2	2	894.42
SND14C30N14D3T20S	14	30	14	3	20	Y	3	3	844.09
SND14C30N14D5T20S	14	30	14	5	20	Y	3	3	830.82
SND40C16N14D3T20S	40	16	14	3	20	N	6	6	2509.48
SND40C16N14D3T90S	40	16	14	3	90	Y	6	6	2570.39
SND40C16N14D5T20S	40	16	14	5	20	Y	4	5	2289.50
SND40C16N14D5T60S	40	16	14	5	60	N	4	5	2289.63
SND40C16N14D5T90S	40	16	14	5	90	Y	4	5	2279.95
SND40C30N14D3T20S	40	30	14	3	20	N	5	6	1993.47
SND40C30N14D3T60S	40	30	14	3	60	Y	5	6	1934.19
SND40C30N14D3T90S	40	30	14	3	90	N	5	6	1974.51
SND40C30N14D5T20S	40	30	14	5	20	Y	5	5	1823.63
SND40C30N14D5T60S	40	30	14	5	60	N	5	6	1850.93
SND40C30N14D5T90S	40	30	14	5	90	N	5	5	1811.24
SND80C16N14D3T20S	80	16	14	3	20	N	9	10	4524.15
SND80C16N14D3T60S	80	16	14	3	60	N	9	11	4585.39
SND80C16N14D3T90S	80	16	14	3	90	N	9	11	4503.08
SND80C16N14D5T20S	80	16	14	5	20	Y	8	8	4099.01
SND80C16N14D5T60S	80	16	14	5	60	N	8	9	4199.84
SND80C16N14D5T90S	80	16	14	5	90	N	8	9	4222.63
SND80C30N14D3T20S	80	30	14	3	20	Y	11	11	4432.54
SND80C30N14D3T60S	80	30	14	3	60	N	11	11	4534.27
SND80C30N14D3T90S	80	30	14	3	90	N	11	12	4351.18
SND80C30N14D5T20S	80	30	14	5	20	Y	9	10	4052.64
SND80C30N14D5T60S	80	30	14	5	60	Y	9	11	3975.22
SND80C30N14D5T90S	80	30	14	5	90	Y	9	10	3994.80