An efficient randomized algorithm for truck scheduling

by

James Adrian Strickland

B.Sc., University of Toronto, 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the School

of

Computing Science

SIMON FRASER UNIVERSITY

March 1994

APPROVAL

Name: James Adrian Strickland

Degree: Master of Science

Title of thesis: An efficient randomized algorithm for truck scheduling

Examining Committee: Dr. Jim Delgrande
Chair

Dr. Arvind Gupta
Senior Supervisor

Dr. Ramesh Krishnamurti
Supervisor

Dr. David Fracchia
External Examiner

Date Approved: March 21, 1994

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

Title of Thesis/Project/Extended Essay

An Efficient Randomized Algorithm for Truck Scheduling.

Author:

(signature)

James Strickland

(name)

March 28, 1994

(date)

Abstract

Many truck scheduling problems are computationally intensive. Furthermore, the behaviour of sites to be serviced by the truck is often unpredictable, making it impossible to define an "exact" solution to the problem.

This thesis examines one such practical scheduling problem. The problem is shown to be difficult to solve. Possible approaches to solving the problem are discussed, followed by the description of a randomized heuristic algorithm which was developed to efficiently solve the problem. The implementation of the algorithm in a simulation program is described, followed by the simulation results which support the claims made about the algorithm.

A powerful "Algorithm Performance Visualization Tool" is then described. This tool was constructed to allow the algorithm development to proceed more quickly. With minor modifications it could be used to aid in the development of algorithms for other vehicle scheduling problems.

Dedication

To Heidi and Buddy.

Contents

List of Figures

Chapter 1

Introduction

1.1 Schedule optimization problems

A *schedule* is a plan which indicates a sequence of events, usually in such a manner as to satisfy a set of constraints (call such a schedule a *feasible* schedule). Often the constraints are such that very few schedules satisfy them, and often finding these schedules is straightforward. For example, any sequence of events which depend on each other in a linear way (*i.e.* A must be done before B, B must be done before C, etcetera) have only one feasible schedule — do A, then B, then C, and so on.

If the constraints are less stringent then many feasible schedules may exist, and thus the issue of the relative merit of a feasible schedule must be addressed. Suppose, for example, that you need to go to the bank (B), the grocery store (G) and the library (L), starting and ending at home (see Figure 1.1 for a diagram showing these locations), with the only constraint being that you need money from the bank before going to the grocery store. Your objective is to minimize travel time.

There are 3! possible orderings of B, G, L, but half of these put G before B, so there are only three feasible schedules: BGL, BLG, LBG. Clearly LBG will take longer than the other two feasible schedules, and thus it is not an *optimal* schedule. Both BGL and BLG are optimal schedules, however, since there is no feasible schedule with a shorter travel time.

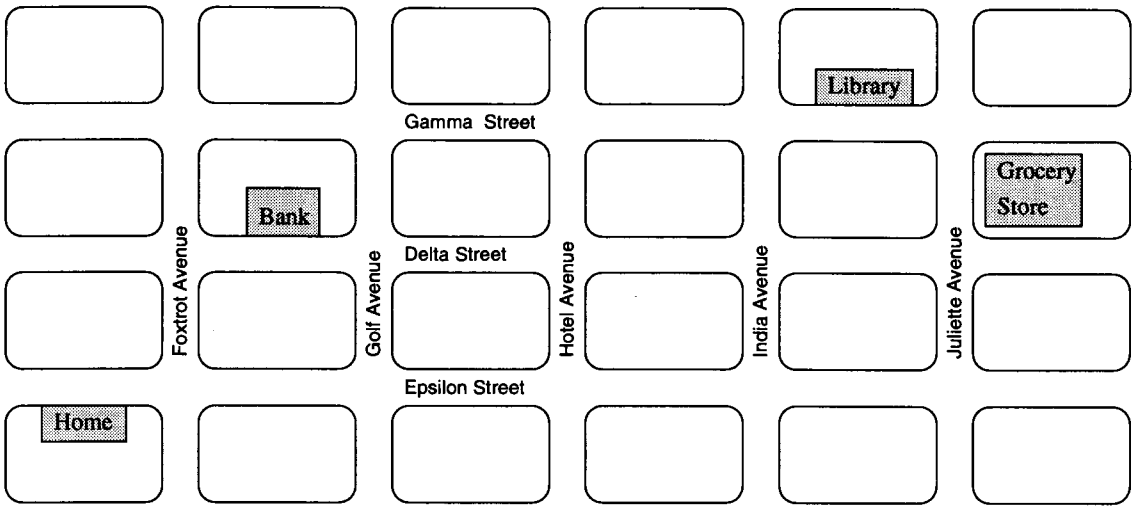


Figure 1.1: The neighbourhood

How do we determine an optimal feasible schedule for a given problem? We could use the brute force method of determining the feasibility and "goodness" of every possible schedule, then pick the best feasible schedule. This *exhaustive search* method only works for small problems, however. As the problem gets larger, it becomes necessary to "prune away" obviously bad schedules quickly.

In the example above, half of the possibilities could be ruled out immediately since they were not feasible. Such "intelligent" observations may make it possible to determine an optimal schedule in a reasonable time for some problems (*i.e.* ad hoc is often a good technique!).

For many problems, however, finding an optimal solution is not practical, so instead we have to try to find a good solution — one which is likely to be optimal or close to optimal. Algorithms for finding such solutions are called *heuristic* algorithms.

1.2 A sample problem

Suppose you are the coordinator of the *42nd Annual Convention of The Society for Having Fun Watching Movies* which is being held at your university. You have access to a large number

of identical rooms which are suitable for movie viewing, but the university administration would like you to minimize how many rooms you reserve so that they can also accommodate classes and the competing *Convention of The Association for Serious Film Studies*. Your convention lasts one day (8 hours, say) and hopes to show n movies, of varying lengths. How do you schedule the movies so as to minimize the number of rooms required?

1.3 Complexity of algorithms which compute exact solutions

The sample problem seems simple enough (*i.e.* it does not seem to be a particularly difficult schedule optimization problem), but it belongs to a class of problems called *NP-complete* problems. It is not currently known if there are any polynomial time algorithms for NP-complete problems. Many people believe that no such algorithms exist, in which case the movie allocation problem would be called an *intractable* problem.

A *polynomial time algorithm* is one which takes a length of time which is a polynomial function of the input length of the problem. Polynomial time algorithms are generally regarded as "reasonable" algorithms (and thus the problems they solve are termed *tractable*), whereas algorithms which do not run in polynomial time (*exponential time algorithms*) are quite "unreasonable". Thinking of the difference between the value of n^2 and 2^n as n grows larger should give a hint as to the validity of this distinction between "reasonable" and "unreasonable" algorithms. See Chapter 1 of [6] for a full discussion of the above concepts.

Readers familiar with issues of computational complexity will undoubtedly recognize the sample problem as the optimization version of the classic *bin packing problem*, which has been proven to be NP-complete. Section 2.3 will show that the sample problem above is no more difficult to solve than the problem which is central to this thesis (described in Chapter 2) by reducing the former problem to the latter.

1.4 Heuristics

Although the sample problem is NP-complete (and thus no polynomial time algorithm is currently known for it), a schedule for the movies is still required. The natural alternative to constructing an algorithm to solve the problem exactly is to try an approach which seems intuitive. The result may not be the optimal solution, but it might be good enough.

In fact, there is an intuitive algorithm for the bin packing problem which comes remarkably close to generating an optimal solution. The algorithm is as follows: sort the movies so that they are in decreasing order, then assign each movie (in order) to the first room which has enough time left unassigned to show that movie. It has been proven (see [6], page 126) that the resulting solution is guaranteed never to be more than about 22% worse than optimal (*e.g.* if an optimal solution uses 50 rooms, then the solution generated by this approximation algorithm will use somewhere between 50 and 61 rooms).

1.5 Randomization

The problem to be solved in this thesis is more complicated than the bin packing problem. Thus, we must use a heuristic, perhaps one similar to that for the bin packing problem. One problem with a deterministic algorithm is that of pathological[1] cases which can make the algorithm perform poorly. It may be possible to alleviate this by the use of a randomized algorithm.

An example of this behaviour can be seen by examining the standard deterministic Quicksort algorithm (see [3], page 161), which has an average case time complexity of $\theta(n \log n)$ but a worst case time complexity of $\theta(n^2)$ (where n is the number of elements to sort). Inputs which result in the worst case time performance tend to be those with structure; in this case, partially sorted lists. Unfortunately, these tend to be the inputs which arise in practice.

[1]The word pathological, when used in a mathematical context, has a meaning similar to that of the word "deviant".

These pathological inputs make up only an exponentially small portion[2] of all possible inputs, however. By mapping an input to an equivalent randomly chosen input (*i.e.* by permuting the elements in the original input) *before* running the Quicksort algorithm, the expected running time is reduced to $\theta(n \log n)$ time.[3] This algorithm is called randomized Quicksort.

In effect, the algorithm is protecting itself against pathological inputs which may occur. This is important for our problem because the input seems likely to be structured (cities are not randomly organized), and thus may cause worst case performance. An example which shows how our randomized algorithm handles one such pathological input is given in Section 3.2.3.

Few randomized algorithms have been formally analyzed because of the difficulty of such analysis. Randomized Quicksort has been analyzed, however; see Chapter 1 of [15].

1.6 Related work

1.6.1 The Vehicle Routing Problem

Both the movie scheduling problem and the problem which is central to this thesis are instances of a general problem called the *vehicle routing problem* (VRP). The general vehicle routing problem involves a set of vehicles, a set of customers, a depot, and a set of delivery and/or pickup requirements. An optimal solution to the vehicle routing problem is a set of routes which is feasible (satisfies all constraints) and which minimizes an objective function, such as travel time or number of vehicles required.

The movie scheduling problem, although not phrased in terms of vehicles, can be seen to be a vehicle routing problem if a correspondence is made between vehicles and rooms, with rooms "picking up" movies until they are "full" (all the time that room is available has been allocated). The objective is to minimize the number of vehicles (rooms) required.

[2] In other words, if there are n possible inputs, only $O(\log n)$ of these would be pathological.

[3] There is a *deterministic* sorting algorithm whose worst case time complexity is $\theta(n \log n)$, but in practice the algorithm is slow, since the "hidden constant" is large.

Considerable research has been done on algorithms to solve the vehicle routing problem, since the problem is commonplace, and improvements to ad hoc solutions can provide significant environmental and economic advantages.[4]

A more formal description of the vehicle routing problem can be found in [2]. Examples of many different types of vehicle routing problems are given in [5], as part of an explanation of the authors' proposal of a classification scheme for vehicle routing problems.

A very detailed survey of the vehicle routing problem and other related problems, such as crew scheduling, is given in [1]. The authors discuss theoretical problems and results, as well as current implementations of systems designed to solve practical problems. A very large bibliography (699 references) is given.

1.6.2 Methods for solving the VRP

The vehicle routing problem is a generalization of the *travelling salesperson problem* (TSP). This problem is to find the minimum cost tour of a set of points. See [10] for a detailed survey of work on the TSP.

The TSP is known to be a very difficult problem to solve (it is an NP-complete problem), and thus the VRP is also a very difficult problem to solve. In practice, "VRPs are in general much more difficult to solve than TSPs of the same size" ([9]).

One approach to solving the VRP is to use an algorithm which is guaranteed to provide an optimal solution. Such an algorithm is called an *exact* algorithm.

The other approach is to use an algorithm which provides *approximate* solutions; ones which are not guaranteed to be optimal. At present this is the only feasible option for problems of considerable size, such as the one which will be presented in this thesis.

A recent survey of both exact and approximate algorithms for the VRP can be found in [8].

[4]These advantages are due to reducing energy use (and hence pollution produced) by using vehicles less frequently, and/or having to own and maintain fewer vehicles.

Exact algorithms

A survey of exact algorithms for the VRP is given in [9]. The authors categorize exact algorithms for the vehicle routing problem as being one of:

- direct tree search methods
- dynamic programming
- integer linear programming

and conclude that "exact methods can only handle problems of relatively modest dimensions".

The current (as of 1992) meaning of "relatively modest dimensions" can be seen in [4], in which the authors state that their algorithm (for a related problem) "was capable of optimally solving 100 customer problems. This problem size is six times larger than any reported to date by other published research."

Approximate algorithms

Our algorithm uses the idea of sequentially building a route. An early example of this type of algorithm is described in [11]. A more recent example of a route building algorithm (extended to run in parallel) can be found in [14]. The problem to be solved by that algorithm (the VRP with time windows) is not quite the same problem that we are solving, however. See [19] for a survey of algorithms for the VRP with time windows.

A description of the implementation of an algorithm for a problem which is similar to ours (refuse collection) can be found in [13].

No mention of randomized algorithms for the vehicle routing problem was found in the literature, and indeed it seems that the use of randomization has not been pursued thus far ([12], [16]).

1.7 Organization of this thesis

A description of the particular scheduling problem considered in this thesis is given in Chapter 2. Possible approaches to solving the problem are explored in Chapter 3. An algorithm which efficiently solves the problem is given in Chapter 4. The implementation of this algorithm in a simulation program is given in Chapter 5, with results of running the simulation program given in Chapter 6.

Chapter 7 describes an "Algorithm Performance Visualization Tool" which allowed for rapid development of the algorithm presented. Conclusions are found in Chapter 8.

Chapter 2

The Problem

2.1 Description of the problem

The schedule optimization problem to be solved can be stated in terms of an unending producer-consumer process. There is one consumer (we will call the consumer's location the *depot*) and many producers distributed over southwestern British Columbia. The exact number of producers n varies from day to day.

Each producer has a standard capacity bin to store the commodity it produces. The producer at location l fills its bin on average every $p(l)$ days; the rate at which commodity is produced varies from day to day and is assumed to be approximately normally distributed.[1] Producers notify (complain to) the consumer when their bin has become full. Commodity produced after the bin is full (after an *overflow* occurs) is discarded.

The consumer removes the commodity by truck (we will call this a *pickup*) and processes it. At the time of pickup, all commodity that has accumulated in a bin must be removed.[2] The time taken to empty a bin is independent of its fullness. The truck has a fixed capacity C and the driver has a fixed maximum working time T. It is assumed that only one truck

[1] By "approximately" we mean that the distribution is assumed to be symmetric about a mean value in a manner similar to that of a normal distribution except that there are no "tails" - there cannot be a negative amount produced, for example.

[2] This is an important consideration; most vehicle routing problems do not have this constraint.

is available.

A schedule, or *route*, for a given day is an ordered list of locations $< l_1, l_2, \ldots, l_m >$ (where $m \leq n$) whose bins are to be emptied. A truck following such a schedule begins and ends at the depot (by convention we will call this l_0). A schedule is feasible if it is possible to fit all the commodity from the specified locations into the truck (the *space constraint*) and it is possible for the driver to take the truck from the depot to the locations and perform the pickups in the time available (the *time constraint*).

The space constraint is satisfied if the sum of the amounts in the bins at the locations is less than the capacity (C) of the truck. Commodity amounts will be expressed in units of "bins", since this is a more directly useful measure than mass or volume. Thus, C is also an upper bound on the number of bins the truck can empty in one day.

The time constraint is satisfied if the time after the last pickup (represented by the t_{ap} function) plus the travel time to the depot (given by the t_t function) is less than the maximum working time (T). Of course, the time after the last pickup is simply the time before the last pickup (represented by the t_{bp} function) plus the time required to perform the pickup (represented by the t_p function). This time is equal to the time after the second last pickup, plus the travel time from that location to the location of the last pickup. This recursive definition continues until the starting point, which occurs at time 0 at the depot, is reached. The reason for this being stated in what seems to be a complicated manner is that the time taken to perform a pickup or to travel between two places depends in general on the time of day. This is a complication which most city dwellers are familiar with!

More formally, a schedule is feasible if it satisfies the space constraint

$$\sum_{i=1}^{m} a(t_{bp}(i), l_i) \leq C$$

and the time constraint

$$t_{ap}(m) + t_t(t_{ap}(m), l_m, l_0) \leq T$$

where

$a(t, l)$ is the amount of commodity in the bin at location l

$t_p(t, l)$ is the time it takes to pick up the contents of location l's bin at time t

$t_t(t, a, b)$ is the travel time from location a to location b starting at time t

$t_{bp}(i) = \begin{cases} t_t(0, l_0, l_1) & \text{for } i = 1 \\ t_{ap}(i-1) + t_t(t_{ap}(i-1), l_{i-1}, l_i) & \text{for } 1 < i \leq m \end{cases}$ is the time before pickup i.

$t_{ap}(i) = t_{bp}(i) + t_p(t_{bp}(i), l_i)$ is the time after pickup i.

The consumer can only estimate the amount in a bin unless it has been told the bin is full. In other words, it is necessary to be at location l at time t to know $a(t, l)$ unless $a(t, l) = 1$. Thus, an estimate $g(t, l)$ of $a(t, l)$ must be made given the period $p(l)$ and the assumed commodity production rate distribution.

In constructing daily schedules, or *routes*, the main objective is to minimize the number of overflows which occur. Other objectives include minimizing the overall time and the number of days the truck is in use, and minimizing the distance travelled.

Minimizing a quantity over one day and minimizing that quantity over all future days is not the same thing! Changing a route one day can have far reaching effects. For example, removing one producer from a route will improve the route with respect to the "minimize time the truck is in use" objective *on that day*, but it may in fact cause an overall increase in the time the truck is in use if that same producer needs to be visited on a later day when it is less convenient to do so.

2.2 Simplifying assumptions

In solving the problem described above, we made certain simplifying assumptions:

- The time taken to pick up a bin does *not* depend on the location. This assumption is only necessary because the data regarding how pickup time varies by location was not available; the change required to handle this properly is straightforward.
- The time taken to pick up a bin does *not* depend on the time of day. In fact, it does depend quite significantly on the time of day, but in a manner which is roughly

uniform across all locations. Thus, there will be a bad time to visit any location which will happen every day — there is not much that can be done about this except to try to account for it by using an appropriately conservative value for the mean time taken to pick up a bin.

- Travel time does *not* depend on the roads used, except for the fact that barriers (*e.g.* mountains, the ocean, rivers, lakes, bogs) may only be crossed (if at all) in designated places (*e.g.* bridges or tunnels). This could be remedied by the use of a more accurate model of the road system which would take into account the actual geographical structure and the actual speeds expected on each road. Such a model is very expensive to produce, but could easily be incorporated.

- Travel time does *not* depend on the time of day. Modelling travel times that vary depending on the time of day is extremely difficult and not likely to be accurate in any event.

- Travel time is symmetric (*i.e.* the travel time from location A to location B is always the same as the travel time from location B to location A). This follows from the previous two assumptions.

- One day's commodity accumulation is generated all at once before the truck starts its route. This introduces a small error in the amount of commodity at a location l: $\frac{1}{p(l)}$ in the worst case. This error is overwhelmed by the next assumption:

- The commodity production rate distribution at a location remains constant. There are, in fact, seasonal variations, but for the purpose of evaluating possible algorithms this is not a complication that needs to be considered.

- The variance of the commodity production rate at each location is known. It is assumed to be non-zero, and is selected to be large enough to make it impossible for an algorithm to predict commodity levels extremely accurately, but small enough to

make scheduling feasible.[3]

- The number of producers, n, is fixed throughout the simulation run. Once again, it is much easier to see what is going on if this assumption is made.

As a result of the above, a and t_t functions "lose" their time argument[4] and t_p becomes a constant. The constraints are now much simpler; the space constraint is

$$\sum_{i=1}^{m} a(l_i) \leq C$$

and the time constraint is

$$m \cdot t_p + (\sum_{i=0}^{m} t_t(l_i, l_{i+1})) \leq T$$

where $l_{m+1} = l_0$. Recall that l_0 refers to the depot.

2.3 Complexity of related problems

(Note: this section may be skipped without affecting understanding of the rest of the thesis.)

The problem, with simplifying assumptions given in Section 2.2, is an "on-line" problem; it continues indefinitely. As such, it cannot be stated in terms of being a problem in NP, although two simpler variants of the problem can be.

The first variant is as follows: Given a list of locations to pick up on a given day, what is the minimum time required to visit the locations?

This variant is simply the travelling salesperson problem, which has been proven to be NP-complete (see [6] or [10]).

The second variant (call it "CLEANUP") is as follows: Given an initial amount of commodity a_l at each location l and a commodity production rate of zero at each location,

[3]The ability to predict the amount of commodity at a location decreases as the variance of the production rate increases. With a large enough variance all algorithms are effectively working "in the dark".

[4]Actually, a is independent of the time of day, but it is still a function of the day. For correctness, assume that a refers to a particular function only on a certain day; the next day a refers to a different function.

what is the minimum number of days required to pick up the commodity at each location (with the constraints as given above)?

Theorem 1 *CLEANUP is NP-complete.*

Proof: Consider the *decision problem* version of CLEANUP, namely instead of asking for the minimum number of days simply ask whether all locations can be picked up in no more than k days. Stated formally, the problem is as follows: Given a set of amounts $\{a_1, a_2, \ldots, a_n\}$, a function $t_t : \{0, \ldots, n\} \times \{0, \ldots, n\} \rightarrow \mathbb{R}^+$ satisfying the triangle inequality[5] (t_t is meant to be the travel time between any two locations, with location 0 being the depot) and constants $t_p, C, T \in \mathbb{R}$, is there a partition of $\{1, \ldots, n\}$ into disjoint subsets $\{S_1, S_2, \ldots, S_k\}$ and a set of k permutations $l_i : \{1, \ldots, |S_i|\} \rightarrow S_i$ such that

$$\sum_{j \in S_i} a_j \leq C$$

and

$$|S_i| \cdot t_p + t_t(0, l_i(1)) + \left(\sum_{q=1}^{|S_i|-1} t_t(l_i(q), l_i(q+1)) \right) + t_t(l_i(|S_i|), 0) \leq T$$

for each day i in the range 1 to k.

This problem is clearly in NP, since a nondeterministic algorithm can guess a k-day schedule and check in polynomial time whether

- the schedule is feasible; *i.e.* for each day verify that
  - the sum of the amounts picked up is not more than C
  - the sum of the travel and pickup times is not more than T
- the schedule will involve visiting all n locations

We will transform the (decision version of the) bin packing problem to the decision version of CLEANUP. The bin packing problem, formally stated, is as follows: Given a

[5]Namely, $t_t(a, b) + t_t(b, c) \geq t_t(a, c)$ for any three locations a, b, and c.

finite set $U = \{u_1, u_2, \ldots, u_n\}$ of items and a size $s(u) \in Z^+$ for each item $u \in U$, a positive integer bin capacity B, and a positive integer K, is there a partition of U into disjoint subsets $U_1, U_2, \ldots, U_k$ such that the sum of the sizes of the items in each U_i is B or less?

Each item u_l in U will correspond to a location, with $a_l = s(u_l)$. The bin capacity B will correspond to the truck capacity C. The partition of U into disjoint subsets will correspond to the assignment of each location to the unique day that the location is picked up. The travel and pickup time will be assumed to be zero. This transformation is basically a change of notation, and thus can clearly be done in polynomial time.

It then follows that there is a k-day schedule for $\{a_1, a_2, \ldots, a_n\}$ if and only if there is a satisfying partition of U. Showing this formally is an exercise in symbol manipulation and would not make anything clearer; the correspondence between bin packing and this problem should be intuitively clear.

Since the decision problem version of CLEANUP is NP-complete, it follows that CLEANUP itself is NP-complete — see [6], page 19.

2.4 The problem is difficult to solve exactly

From the previous section it seems clear that the problem at hand is very difficult to solve. The "CLEANUP" problem is very similar but is simplified in that no commodity is being added to the system. Adding this complexity does not result in an easier problem.

The fact that the amount of commodity being produced is not known precisely does not seem to be a major factor in making the problem hard. It seems clear from the above arguments that even if the commodity production rate were a constant, thus making it possible for an algorithm to have an exact knowledge of the future, the problem would still be difficult to solve.

Chapter 3

Possible approaches to solving the problem

3.1 Fixed schedules

One possible solution to the truck scheduling problem described in Chapter 2 is to determine a set of schedules for the next k days such that all locations are visited at least once and each location l is visited every $v(l)$ days, where $v(l) < p(l)$. Then on day i simply perform the schedule for day i mod k.

Schedules resulting from this approach are often of a form that "sweeps" around the service area. One way to construct such a schedule is to "slice" the "pie" defined by the depot at the centre and the outermost producer on the outside in such a manner as to give equal length routes in each slice, while somehow taking the commodity production rates at the locations within that slice into account. See [17] for the description of an algorithm of this type.

There are three problems with this approach. First of all, it may not be possible to satisfy the restriction $v(l) < p(l)$ since there are just too many locations to satisfy with geographically pleasing routes. In fact, if it is possible to satisfy this requirement then it is likely that the variance in production rates is small (making it a simpler problem) or there

is more truck capacity than required.

The second problem is that the production of commodity over the next k days may turn out to be very different than predicted, due to the rate at which new producers come into existence and old producers disappear.[1]

The third problem is that total production rates within geographic areas change over time, necessitating constant reshuffling of the routes.

3.2 Greedy heuristics

Instead of determining a sequence of schedules for k days, we could simply determine a schedule each day, constructed by starting with an empty schedule and adding locations which are chosen in some manner.

One manner in which to choose locations is to use a heuristic which estimates the benefit of visiting a particular location at a particular time. This heuristic is given as a *weighting function* which determines the *weighting* for each location.[2] It is a simple manner to then choose the location which seems best at the time. This is termed a *greedy* approach, since we are always taking the best thing we can get, perhaps at the expense of overall benefit.

The issue, then, is determining an appropriate weighting function.

3.2.1 Simple greedy heuristic

Two simple greedy heuristic algorithms are obvious: one is greedy with respect to space (*i.e.* the weighting assigned to a location is proportional to the amount of commodity at that location), the other is greedy with respect to time (*i.e.* the weighting assigned to a location is inversely proportional to the travel time required to reach that location).

One way of implementing a greedy algorithm with respect to space would be to sort the locations in descending order of $g(l)$ (or, alternatively, in ascending order of the estimated

[1] This is not allowed under the simplifying assumptions presented, but it does occur in the actual problem. The algorithm presented later handles this without difficulty.

[2] The weighting is a non-negative real number which indicates the benefit of adding that location to the schedule.

time before an overflow is expected), then visit locations as they appear on the list until running out of space or time. The problem, of course, is that the locations which make it to the head of the list may be geographically scattered, resulting in the truck spending an inordinate amount of time travelling. Even if the order of the locations which can be visited is optimized (or at least made close to optimal) as locations are added to the list, the truck may still travel excessively due to the distance between the furthest locations. The result is that the truck will usually return with a load much smaller than is possible.

A possible greedy algorithm with respect to time would simply choose the closest location l with $g(l) > x$ for some threshold value x, then add locations which minimize the added time required to visit them. The problem with this approach is that time will be wasted visiting locations which do not have much commodity. The end result is similar to that for the greedy algorithm with respect to space.

The actual performance of these greedy algorithms is shown in Section 6.2.1.

3.2.2 Sophisticated greedy heuristic

It is clear that being greedy with respect to either space or time alone will not work because both considerations are important.

If we could find a function which weighs these two factors appropriately we could hopefully avoid the problems inherent in ignoring space or ignoring time.

But how do we balance the relative importance of space and time? Intuitively, it seems that the desirability of visiting location l is directly proportional to $g(l)$ and inversely proportional to how long it takes to get to location l. But how long it takes to get to location l depends on the current candidate locations to visit.

The idea, then, is to somehow select a location as a "seed", then add locations until a maximal route is achieved; *i.e.* until it is no longer possible to add a location without making the resulting route infeasible. The weighting function will have to be recomputed at each step in order to take into consideration the locations which have been chosen previously.

There is now an issue of how the depot is treated: is it always implicitly on the list, or is it not on the list? If the former, then the weighting function may be biased towards choosing locations close to the depot.

The issue of whether the depot is "on the route" also determines the method by which the "seed" location is selected. If the depot is implicitly on the list, then the same weighting function can be used to select the seed as is used in later steps of the algorithm. This will provide a severe bias towards the depot, however. The alternative is to use a different weighting function for selecting the seed, namely one which is independent of the travel time from the depot. This alternative was chosen.

3.2.3 Randomized sophisticated greedy heuristic

Instead of always picking the location with the highest weighting, we could randomly select a location based on the computed weightings. In other words, if the weighting for location l is $weight(l)$, choose it with probability

$$\frac{weight(l)}{\sum_{i=1}^{n} weight(i)}$$

But what possible advantage is there in allowing the algorithm to choose a location which does not have the highest weighting? In general, the advantage is that the location with the highest weighting may not be the most desirable, no matter how good the weighting function is, due to the existence of a group of locations which together are more desirable. The following example will illustrate this concept.

Suppose there are 7 locations, labelled A through G, distributed as shown in Figure 3.1. For ease of exposition, suppose that travel can only be accomplished along the lines indicated, and that travel times are as shown. The fullness of the bin at each location is given in Figure 3.2 and is indicated in Figure 3.1 by a rectangular box beside each location label.

Suppose that the truck has a capacity of 2 bins, it takes 6 minutes (0.1 hours) to pick

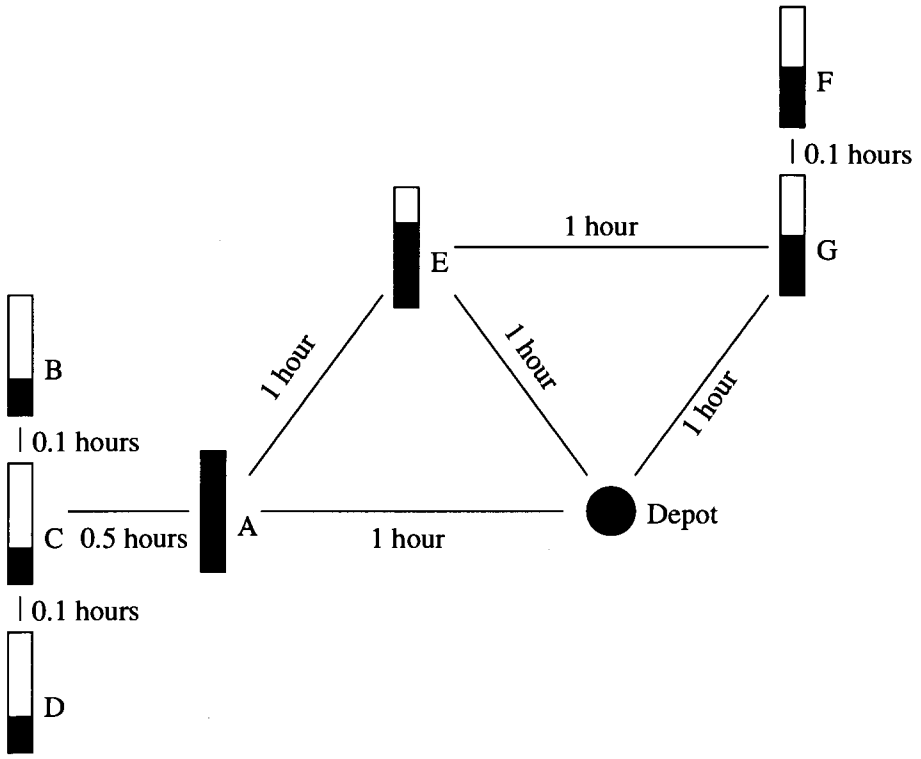


Figure 3.1: Sample world

Location identifier	Amount of commodity
A	1.0
B	0.3
C	0.3
D	0.3
E	0.7
F	0.6
G	0.6

Figure 3.2: Bin fullness

Location identifier	Amount of commodity	Weighting
A	0.0	0 (already visited)
B	0.3	$0.3/0.6 = 0.5$
C	0.3	$0.3/0.5 = 0.6$
D	0.3	$0.3/0.6 = 0.5$
E	0.7	$0.7/1.0 = 0.7$
F	0.6	0 (cannot reach in time remaining)
G	0.6	0 (cannot reach in time remaining)

Figure 3.3: Weightings after truck picks up bin at location A

up a bin, and the driver only works 4 hours per day.[3] Suppose further that the weighting function to be used is $a(l)/t(l)$, where $a(l)$ is the the amount of commodity in location l's bin, and $t(l)$ is the time to location l from the current truck location.

Consider what will happen if the deterministic algorithm is used. The truck will travel to A first, since it is full. Now the weightings are as given in Figure 3.3, and the truck is committed to working 2.1 hours (1 hour to A, 1 hour to return to the depot, plus 0.1 hours to pick up the bin at A). At this point it is not feasible to reach location F or location G since that would require at least a 4 hour round trip plus some time to pick up bins.

The location with the highest weighting is location E, and thus the truck now goes to location E. At this point, the route involves picking up 1.7 bins with 3.2 hours of work. Since the closest location to A or E is at least half an hour away (necessitating a 1 hour round trip), it is now impossible to add any other locations.

The next day the truck will go to location G. Once there, it can only reach location F in the time remaining, and hence will service location F. The result is 2.4 hours of work to pick up 1.2 bins.

On the third day the truck will service locations B, C, and D, taking 3.7 hours to pick up 0.9 bins.

The randomized algorithm is likely to do a better job on this example. It starts out servicing location A since it has overflowed. Now it chooses a location according to the weightings given in Figure 3.3. Thus, for example, location E is chosen with probability

[3]These constants are unrealistically small, but this is necessary in order to avoid an unwieldy example.

$0.7/2.3 \approx 0.3$, since the sum of the weightings is 2.3.

The decision to service E next was the downfall of the deterministic algorithm, since it then committed itself to spending two more days to service the remaining locations. It chose to go to a location with a large amount of commodity, but in so doing it left behind a *group* of locations (*i.e.* locations B, C, and D) with an even larger amount of commodity.

The randomized algorithm will choose one of B, C, or D with probability 0.7 (approximately). If it does choose one of these locations, then it will be in a position where the only remaining locations it can reach are in the group B, C, D. In this case it will end up servicing locations A, B, C, and D, taking 3.8 hours to pick up 1.9 bins of commodity.

The next day the truck will be able to service the remaining locations (E, F, and G), taking 3.5 hours to pick up 1.9 bins of commodity.

Thus the randomized version manages to service all locations in 2 days with probability 70% (it will take 3 days with probability 30%), whereas the deterministic version always requires 3 days. We have therefore shown that there are cases in which the randomized algorithm will do no worse than the deterministic algorithm, but in fact can do better.

Performance of the actual randomized and deterministic algorithms is discussed in Section 6.4.4.

Chapter 4

The Algorithm

4.1 Overview

In order to solve the problem outlined in Chapter 2 we must construct a *route selection algorithm* which will provide a route to perform on a given day when all relevant information *which is known by the consumer at that time* is provided as input. This "relevant information" is given in Figure 4.1.

Some of the information used as input to the algorithm could be inaccurate. Thus, we cannot expect the algorithm to produce a route which, in practice, will be seen to have been the best possible choice. The route may not even turn out to be feasible; for example,

Quantity	Description
n	number of producers
T	maximum working time
$t_t(a, b)$	travel time between any two locations a and b
t_p	time to pick up a bin at a location
C	truck capacity
$g(l)$	estimated amount in bin at location l
$p(l)$	period with which bin at location l fills up
f	fullness threshold below which truck should not be sent out
F	set of locations whose bins are full

Figure 4.1: Input to route selection algorithm

if more commodity existed at the locations chosen than was estimated, it is possible that the total amount of commodity to be picked up will exceed the truck capacity.

More importantly, however, the information used only provides an estimate of the true *system state*. The system state is a snapshot of the state of all entities in the "system", and in this case consists solely of the fullness of the bin at each producer's location.

What is needed, therefore, is a *simulation algorithm* which will maintain the true system state and the information required by the route selection algorithm. The simulation algorithm uses the route selection algorithm to select a route, then "performs" the route by modifying the state of the locations on the route appropriately, while enforcing the constraints of the problem. The simulation algorithm then adds commodity to each producer's bin according to that producer's production rate. This process repeats indefinitely.

4.2 Description of route selection algorithm

A high level description of the selection algorithm which was implemented is shown in Figure 4.2. The algorithm requires knowledge of the quantities given in Figure 4.1. The output of the algorithm is an ordered list of locations $R =< l_1, l_2, \ldots, l_m >$. The order given in this list is not guaranteed to provide the minimum travel time since this would necessitate solving the TSP. In reality, the actual order with which the locations are processed is decided upon by the driver.

The algorithm begins by selecting a "seed" location if there are no overflows, or else creating a route out of the locations which have overflows. The notation $approx\_tsp(F)$ is meant to refer to a list which is the result of applying a TSP approximation algorithm to a list containing the elements of F.

This initial route is extended by adding locations which are selected by the function sl. These locations are (conceptually) added at an appropriate place[1] in the list by the add_loc function.

[1]Once again, it is not possible to say "in the optimal order" since that would involve solving the TSP.

if $F = \emptyset$ then	*If there are no overflows, then*
$R \leftarrow < sl(<>) >$	*pick a seed location.*
else	*If there are overflows, then*
$R \leftarrow approx\_tsp(F)$	*start with them.*
while sufficient space and time remain do	*While there are still possibilities,*
$R \leftarrow add\_loc(sl(R), R)\}$	*add a location*
if $F = \emptyset$ and amount to pick up $< f \times C$ then	*If the route isn't worthwhile doing, then*
$R \leftarrow <>$	*don't send the truck out.*

Figure 4.2: The route selection algorithm

The function sl works by computing a weighting for all locations, then choosing the location with the largest weighting (the *deterministic* version) or selecting a location randomly based on the weightings (the *randomized* version). The weighting given to a location l is likely to be[2] a function of many factors, including the additional travel time from R to l.

The definition of the travel time from a route (list of locations) to another location is discussed in Section 5.1.2. Intuitively, what is meant is the difference between the time required to process route R and the time required to process the shortest route containing R and the new location.

Locations already in R, or which cannot be reached in the time remaining, or which have more commodity than the space remaining in the truck, are assigned a zero weighting and thus cannot be chosen. This is in fact how the "sufficient space and time remaining" condition of the loop is implemented; it is true if and only if there is at least one location with a positive weighting.

Once the loop is completed, the algorithm decides whether the route it has selected is worthwhile. With an appropriate threshold value f, truck usage is minimized while providing adequate service. If the threshold is zero, then the truck will be sent out every

[2]Various weighting functions were tried. These will be described in more detail later in the thesis.

day. This is undesirable if the total system commodity production rate is significantly lower than the truck's capacity, since in that case no producers will ever come close to filling their bins. It would be better to visit locations less often, and thus a higher threshold should be chosen. If too high a threshold is chosen, however, the truck may be prevented from servicing locations until overflows occur. The threshold value chosen is determined experimentally.

4.3 Algorithm features

Here are some important things to note about the route selection algorithm:

- Overflows are always handled immediately after they occur.[3] This corresponds with the desired real world behaviour.
- The seed location l is chosen randomly on the basis of $g(l)$ and $p(l)$ only. (The quantity $g(l)$ is calculated from the current day, the last day the bin at location l was picked up, and $p(l)$.) This means that the algorithm is not inherently biased towards any geographic area; it decides to try an area based on the amount of commodity in bins at locations in that area.
- The calculation of the weightings can be done in an intelligent manner in order to avoid recalculations. See Chapter 5.
- The algorithm automatically adjusts to different total commodity production rates.

[3]Actually, to be more precise, overflows are handled as quickly as possible, since it is possible for more overflows to occur all at once than can be handled in one day. This usually only occurs when the production rate is higher than what the algorithm can handle. See Section 6.1.3 for more details.

4.4 Algorithm extensions

4.4.1 Avoiding overflows

With the algorithm presented in Figure 4.2, it is possible that an overflow will occur on a later day as a result of not sending the truck out. Clearly this is undesirable. One method to alleviate this is to make the decision of whether to go out dependent on a projection of what will happen in the future. If the algorithm could "look ahead" and see that not sending the truck out today would cause an overflow before the next day the truck is not to be sent out, then clearly it should send the truck out today.

Such an extension could be incorporated into the existing simulation algorithm. The new simulation algorithm would have a loop with the following body:

- Run the simulation until a day in which the truck is not to be sent out. Save the simulation state.
- Run the simulation until a day in which either
  - one or more overflows occur, or
  - the algorithm decides not to send the truck out
- If the first condition occurs (*i.e.* overflows occurred before the next day the truck was not to be sent out), then reset the simulation state to the saved state and force the truck to go out on that day.

4.4.2 Scheduling more than one truck

Another useful extension to the algorithm would be the ability to use more than one truck. This problem naturally decomposes into two subproblems: choosing the locations, and assigning the chosen locations to different trucks. The latter problem is a generalization of the travelling salesperson problem called the *multiple travelling salesperson problem.* An algorithm exists which can transform an instance of the multiple TSP to an instance of the TSP; see pages 23 – 25 of [10].

One possible way to choose locations is to simply run the basic algorithm i times, each time only considering the locations which have not yet been selected. Another possibility is to use the algorithm with a truck capacity equal to $i \times C$ and a maximum working time of $i \times T$. Note that it may not be possible to visit all locations which are chosen in this manner; some method is needed to allow the "dropping" of some locations so that i feasible routes can be generated.

In short, it seems as though the multi-truck problem adds significant complexity. The basic algorithm, however, would probably remain a useful part of a multi-truck algorithm.

Chapter 5

The Implementation

5.1 Implementation of the route selection algorithm

The route selection algorithm given in Section 4.2 was implemented in ANSI C on a Sun workstation. Several issues which were not addressed in the high level description of the algorithm given in Section 4.2 are addressed here.

5.1.1 User defined formula for the weighting function

The weighting function is specified at run time (and can in fact be changed at the start of any simulated day) by a formula given as a string, each character of which is either a number, the standard operators +, -, *, / with their standard meanings, the operator ^ for exponentiation, ~ for integer exponentiation (allowing quicker running time when raising to an integer power), # for duplicating the top of stack (the formula is evaluated using a stack), or one of the following letters: d, e, g, p, t. These letters, when the formula is being applied to compute the weighting for location l, refer to the following values:

- d — estimated number of days before an overflow will occur at l, or 1 if the estimated number of days is less than one. We shall refer to this value as $d(l)$.[1]

[1]This definition is due to a desire to avoid negative weightings and divide by zero errors; $d(l)$ usually appears in the denominator of weighting functions.

- e — time to empty a bin. This is the same as t_p.
- g — estimated amount in the bin at a location. This is the same as $g(l)$.
- p — period of filling (*i.e.* assumed number of days after pickup before location will fill its bin). This is the same as $p(l)$.
- t — additional time required to service a location. This is the same as $t(l)$, a variable whose value will be defined in Section 5.1.2.

Some letters are redundant; for example, the value of $g(l)$ is always the same as $\frac{p(l)-d(l)}{p(l)}$ for all locations l, and thus only two of the three letters d, g, and p are required. The added letters are there for convenience in formula entry.

This primitive input specification was adequate for the task at hand, and allowed for an extremely simple and quick "parser".

5.1.2 Definition of $t(l)$

Ideally, $t(l)$ should be defined to be t_p plus the difference between the total travel time for the current route and the total travel time for the shortest possible route after adding location l. This definition requires solving the travelling salesperson problem, however, and thus is not practical.

A good approximation to this definition is that $t(l)$ is the shortest additional time required to go to location l if it were inserted into the route.[2] Formally,

$$t(l) = t_p + \min_{i=0}^{m}(t_t(l_i, l) + t_t(l, l_{i+1}) - t_t(l_i, l_{i+1}))$$

(where $< l_1, l_2, \ldots, l_m >$ is the current list of locations on the route, and l_0 and l_{m+1} refer to the depot).

[2]The difference between this definition and the "ideal" one is that for the ideal, the ordering of the locations on the route can change arbitrarily when a new location is added.

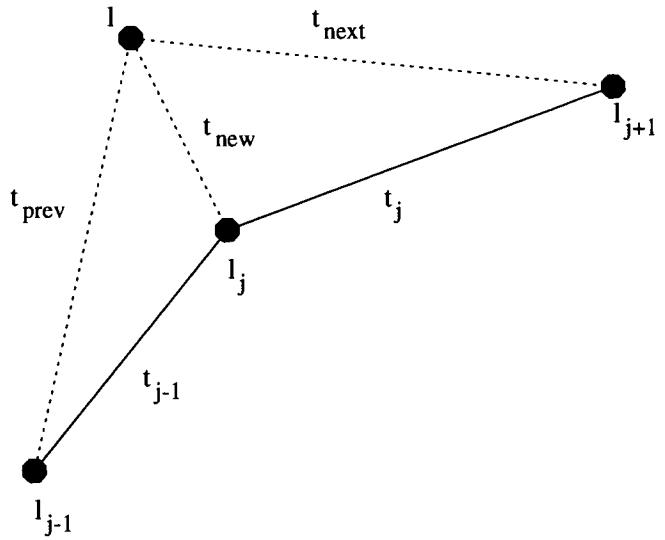


Figure 5.1: Time to the route

The following is a method for quickly recomputing $t(l)$ after a location l_j is added to the route (resulting in a route $< l_1, \ldots, l_{j-1}, l_j, l_{j+1}, \ldots, l_m >$):

- compute the travel times t_{prev}, t_{new}, t_{next}, t_{j-1} and t_j according to Figure 5.1.
- compute $t_{before} = t_{prev} + t_{new} - t_{j-1}$
- compare t_{before} with the current value of $t(l)$. If it is smaller, then set $t(l)$ equal to t_{before}, and set $insert\_after$ to $j - 1$.
- compute $t_{after} = t_{new} + t_{next} - t_j$
- compare t_{after} with the current value of $t(l)$. If it is smaller, then set $t(l)$ equal to t_{after}, and set $insert\_after$ to j.

As long as the order of the previously chosen locations is not changed,[3] this recomputation will ensure that the value stored as $t(l)$ is as defined above at all times.

Consider the following alternative definition of $t(l)$:

[3]The order is, in fact changed whenever the TSP approximation algorithm is used. This necessitates recomputing $t(l)$ for each location l.

$$t(l) = t_p + \min_{i=1}^{m} t_t(l, l_i)$$

This is the smallest time from location l to a location which is already on the route, plus the time to service location l. With this definition the value of $t(l)$ is independent of the order of the locations already chosen. Thus, after adding a location l_j, it is only necessary to compute $t_t(l, l_j)$ and set $t(l)$ to the minimum of this value and $t(l)$.

We chose this definition of $t(l)$ since it yields comparable results to those obtained using the previous definition, but has the advantage of being simpler, being independent of order, [4] and taking significantly less time to compute.

5.1.3 Maintaining an ordered list of locations

The method by which locations are added into the list depends on the definition of $t(l)$ chosen. With the definition which depends on the list order, it is necessary to maintain a variable $insert\_after(l)$ for each location l so that the algorithm will know where that location should be inserted into the list if it is chosen. After adding a number of locations in this manner, however, the route may not be a very efficient one.

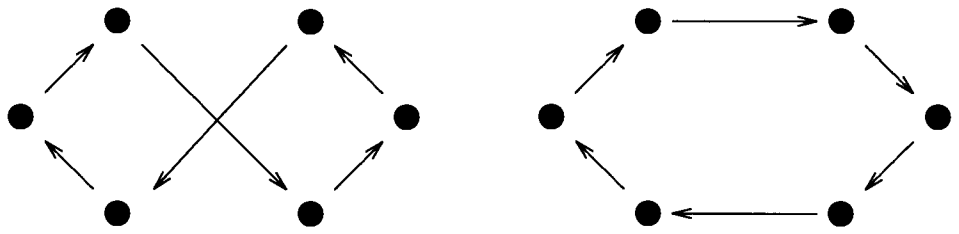
These complications are avoided by using the definition of $t(l)$ which does not depend on list order. This allows the algorithm to simply add new locations to the head of the list.

5.1.4 Computing a time estimate

There is one required quantity which does depend on list order: the estimate of the time required to complete the route. This estimate is not relevant, however, until the route is large enough that the estimate becomes close to T. This is because the only use of the estimate is in determining whether the time constraint is respected.

The definition of "close to T" is a bit vague; what is meant is that the time estimate is

[4] This is important because it means that times do not have to be recomputed after the TSP approximation algorithm is used.



Before edge exchange After edge exchange

Figure 5.2: Sample route improvement

large enough that it may become larger than T after another location is added. A sufficient condition to ensure that the time estimate is not "close to T" is for the estimate to be greater than $T - t_{longest}$, where $t_{longest}$ is the round trip travel time from the depot to the producer which it takes the longest time to reach.

Once the time estimate becomes relevant, an attempt is made to improve it by using an approximation algorithm for the travelling salesperson problem. The algorithm works by exchanging two edges of a route whenever doing so would improve the overall travel time. Figure 5.2 shows a sample route which is improved by swapping two edges. See [18] for a full description of the algorithm used.

5.1.5 Efficient computation of weightings

The efficient computation of weightings is a prime consideration in the implementation of the algorithm, since it is something which is done *very* frequently, and thus is the major determinant of the overall computation time required. Computing weightings inefficiently (*e.g.* by solving the travelling salesperson problem after each location is added to determine the optimal route, then recomputing the weighting for each location) would take far too much real time to be practical.

The value of $d(l)$ and $g(l)$ are fixed on any particular day while the route selection algorithm is executed.[5] The definition of $t(l)$, however, changes as a route is constructed.

[5]It is confusing to talk about a function value which changes. Of course, d, g and t are really functions of a location *and* a time argument. The time argument is not mentioned because it is always implicitly "now".

Thus the value of the weighting function at location l need only be recomputed when the value of $t(l)$ changes.

5.2 Computation of travel times

Travel times between locations are required so frequently in computations that it makes sense to pre-compute all travel times and store them in a matrix.

The program which does this computation was written on the assumption that rectilinear distance estimates[6] are sufficiently accurate. This is a reasonable assumption for much of the area in question, since there is a large road grid oriented north-south and east-west.

The only tricky part is that it is not always possible to get between two locations in the rectilinear distance, due to the existence of barriers such as the ocean, mountains, lakes, rivers, bogs, and large parks. These barriers were digitized by hand from maps, and were represented as a list of coordinates which implicitly defined a closed polygon. Roads which cross barriers (*i.e.* bridges and tunnels) were also digitized and represented as a list of coordinates which implicitly defined line segments.

The program begins by allocating enough memory for a bitmap (the "barrier" bitmap) to hold a description of the area of interest at the specified resolution. Initially all pixels in the bitmap are set to zero. A pixel with value zero (hereafter called a "go" pixel) defines an area it is possible to travel over. A pixel with value one (hereafter called a "no go" pixel) means there is a barrier in the square[7] defined by that pixel precluding travel through that area.

The program then reads in a description of the barriers and sets all pixels lying within the barrier polygons to have a value of one. The description of crossings is then read and lines of value zero are drawn. The result is a bitmap specifying "go" and "no go" pixels according to whether it is assumed there is a road network in the area corresponding to the pixel.

[6]The rectilinear distance between two points A at (x_a, y_a) and B at (x_b, y_b) is $|x_b - x_a| + |y_b - y_a|$.
[7]And occupying more than half the area of the square.

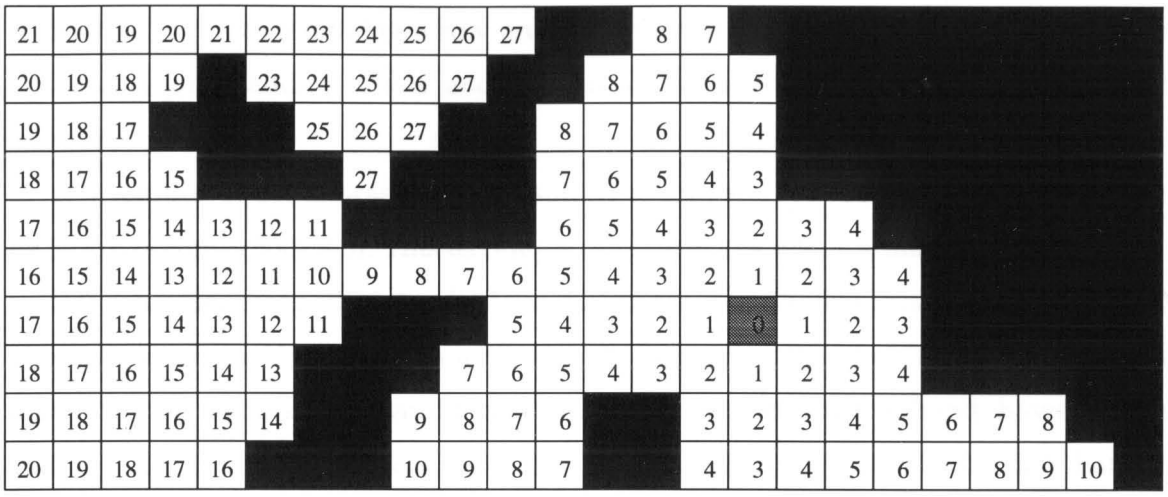


Figure 5.3: Illustration of distance calculation algorithm

To determine the distances from a producer location l to all other producer locations, the algorithm, in effect, labels the "go" pixels adjacent (*i.e.* to the left, right, up or down) to that in which location l lies as being reachable from the starting point in one distance unit. Each pixel which was reached is checked for the existence of a location on the list,[8] and if one is found the distance is entered in the matrix. This step is repeated recursively until all locations on the list have been found, or else the bitmap has been completely searched. The latter condition can only occur if there is at least one location which is unreachable from the starting point — this is likely an indication of an error in the input.

See Figure 5.3 for an illustration of the rectilinear distances computed from a given point (labelled zero and shaded grey) in the presence of barriers (solid black).

Once distance calculations have been done for all new locations, the resulting distance matrix is written out in a compressed format (2 bytes per matrix entry, with only the lower triangle[9] of the matrix stored since the matrix is symmetric about the main diagonal). A utility was written to print the compressed format in a human readable form.

The program also handled the addition and deletion of locations in an efficient manner.

[8]The locations are pre-sorted and entered on linked lists, one per y coordinate, for increased efficiency.

[9]To be more precise, all entries D_{ij} of the distance matrix D such that $i > j$. The main diagonal is composed entirely of zeroes, and thus is not stored.

A deletion involves removing a row and column from the distance matrix, whereas an addition requires adding a row and column and computing the distances to the new location.

A resolution of 100 m was chosen, resulting in bitmaps of about 250 000 pixels (50 km by 50 km becomes 500x500). This technique obviously has severe scaling problems!

For example, computing the distance matrix for roughly 500 locations on a 575x430 bitmap took about 20 minutes on a SPARC-10. Improving the resolution to 50 m would mean that the computation would take 80 minutes. For the task at hand, however, the memory and time requirements are reasonable, especially considering the fact that the initial distance matrix computation need only be done once.

Another problem with the algorithm is the possibility of an insufficiently fine resolution being specified, resulting in an impassable barrier in the bitmap where none exists in reality. This will occur if two barriers are within one half pixel width.

Algorithms exist which can compute shortest paths in the plane in the presence of barriers without discretizing the barriers. One such algorithm is given in [7]. It has a worst-case time complexity of $O(n \log^2 n)$ and a worst case space complexity of $O(n \log n)$ where n is the total number of vertices in the obstacle polygons.

While such an algorithm would likely have been preferable to the one which was implemented,[10] it is undesirable because it does not solve the problem that needs to be solved! What is really required is the *travel time* between any two locations, not the distance between any two locations. Computing the time from the distance necessitates assuming an average speed. It is possible to assume different average speeds depending on location, but such a scheme is not very accurate; it would seem better just to set a pessimistic average speed. The best solution is to model the actual road system as a graph (nodes of the graph would represent intersections, and edges would represent roads, with edge weights being the average speed along that edge), then compute shortest paths on this graph. A "time to get to the nearest intersection" could be computed based on a pessimistic average speed

[10] In terms of accuracy and speed of execution, but not necessarily in terms of speed of implementation!

and the geographic distance to the nearest node on the graph.

Such an algorithm would allow more accurate time estimates. But the algorithm which was implemented, together with a pessimistic average speed estimate, provides a sufficiently good time estimate in most cases, and overestimates the times required for longer distances and for trips which could in reality be done on "diagonal" roads (*i.e.* roads which are not oriented north-south, or east-west). Thus, improving the travel time estimates would only improve the performance of the algorithm of Figure 4.2. In any event, it must be recognized that the variance in travel times along roads is very large at times due to construction, collisions or queuing delays, and thus there is a limit to how well one can estimate travel times.

5.3 Implementation of the simulation algorithm

As explained in Chapter 4, the simulation algorithm has a simple structure. The outer structure is a loop, whose body is executed once per simulated day. The body of the loop adds commodity to each location's bin based on the assumed commodity production distribution,[11] then uses the route selection algorithm to determine a route which is processed in such a manner that the constraints imposed are satisfied. Enforcing the constraints of the problem is necessary because it is possible for the route selection algorithm to specify an infeasible route, due to its imperfect knowledge of the state of the system. Such infeasible routes should be extremely rare, however, if conservative assumptions are made.

Useful statistics such as the number of overflows, the amount of commodity at the producers, and the distance travelled, are kept. After a predetermined number of days (the *warmup period* — see Section 6.1.1), the statistics are "reset" (*i.e.* totals are set to zero, maximums are set to minimum possible values, etc) and the simulation continues for some

[11]The amount added is a random variable which is the result of summing a number of random variables which have a uniform distribution over a finite interval. Thus, due to the central limit theorem, the amount added is approximately normally distributed. The range of possible values, however, is finite, and only includes non-negative numbers.

```
SUMMARY

Number of locations: 452
Simulation over 1000 days
Truck operated on 982 days

Assumptions:
Truck capacity:           20.00 bins
Truck speed:              35.00 km/h
Pickup time:              12.00 minutes
Max allowed working time:  7.50 hours
Min fullness to pick up:   0.25 bins
Min fullness to go out:   70.00%
Prob truck is operable:    1.00
Work on: Monday Tuesday Wednesday Thursday Friday Saturday Sunday
Formula: gdt*/#*#*#*#*
```

Statistic	Total	Mean	Minimum	Maximum	Per?
New overflows	42	0.04	0	3	day
Overflows	42	0.04	0	3	day
Days of overflow	0	0.00	0	0	overflow
Pickups	23896	24.33	16	31	workday
Amount picked up	17973.6	18.30	13.57	20.00	workday
Hours working	7113.4	7.24	5.60	7.50	workday
Distance travelled (km)	81723.6	83.22	24.20	147.20	workday
Amount generated	17983.4	17.98	17.44	18.51	day
Amount lost	0.6	0.00	0.00	0.08	day
Fullness/bin/day	186850.5	0.41	0.38	0.48	day

Figure 5.4: Sample text-based simulation output

other number of days. At this time the value of the statistics is printed. These values can be compared with the values obtained by running the simulation with different parameters to determine relative performance.

A sample printout is shown in Figure 5.4. This example used data (consumer locations and rates of production) provided by the local company. With this data (*i.e.* with $n = 452$) the simulation program ran at the rate of roughly 0.3 seconds per simulated day on a lightly loaded SPARC-10 computer.

As can be seen by examining the sample printout, many simulation parameters (assumptions about the real world) need to be specified. These include the truck capacity and speed, the time it takes to empty a bin ("Pickup time"), the total time the driver can work ("Max allowed working time"), the minimum amount allowed to be picked up at any one location[12] ("Min fullness to pick up"), the minimum amount of commodity expected to be picked up before allowing the truck out ("Min fullness to go out"), the probability of the truck and driver being available on any given day ("Prob truck is operable"), and the days on which the truck is supposed to be available.

The truck goes out (*i.e.* the route selected is performed) if and only if

- The day of the week is a day on which the truck and driver are available.

  AND

- The truck is in operating condition. This is simulated by the condition $U > P$ where U is a uniformly distributed random variable on the interval [0,1) and P is the parameter "Prob truck is operable".

  AND

- The expected amount of commodity to be collected on the selected route is greater than that specified by "Min fullness to go out", OR, there are one or more locations which have overflowed.

[12]This parameter exists because the current practice is to ignore a bin which is mostly empty rather than take the time to empty it. This sort of situation is very unlikely with our algorithm.

Some of the statistics shown in the sample printout require further explanation. "New overflows" refers to the number of times a location changed from having a less than full bin to having a full bin (with excess commodity being "lost"). "Overflows" refers to the sum over all days of the number of locations with full bins. These two quantities will be the same if overflows are handled immediately (as they are with our algorithm). If a location has to wait k days on average after having filled its bin before being picked up, then one would expect the "Overflows" figure to be k times higher than the "New overflows" figure, and the "Days of overflow" mean would show as k.

As will be explained in Section 6.1.1, the simulation starts with all locations having empty bins. Thus, at any given time the total amount of commodity generated must equal the amount picked up plus the amount lost plus the amount currently in storage in bins. This is in fact the case, but it doesn't appear to be so when the statistics shown in the sample printout are examined. The discrepancy occurs because the values shown are for that portion of the simulation which occurred after the warmup period. Thus, the difference between the amount generated, and the sum of the amounts picked up and lost, is simply the amount which has been added or removed from the system from the day at which the simulation statistics were reset.

Chapter 6

Simulation results

6.1 Preliminary comments

Some issues need to be addressed before presenting the actual results.

6.1.1 Simulation "warmup"

The state of the system (*i.e.* the amount of commodity in each producer's bin) changes over time as commodity is created and removed. In time, however, the system state will become relatively stable;[1] the amount removed from bins over a period of time will be approximately the same as the amount that is added over the same period.[2] It is the behaviour of the algorithm once the system reaches this *steady state* that is of primary interest.

But the steady state for a given algorithm and parameter setting is not known in advance. How do we get the algorithm to a steady state so that we can begin "recording" its behaviour?

The only possible answer is to run the simulation to determine the steady state, then

[1] Assuming a reasonable algorithm!

[2] Note the use of "added to bins" as opposed to "produced". In the case of the system reaching a steady state of all bins full, the amount produced will likely be greater than the amount added to bins, with the excess being "lost".

begin "recording" (accumulating relevant statistics) once steady state is reached. The initial part of the simulation run is called the *warmup period* .

There may, however, be more than one possible steady state for the system, depending on the initial state. For example, if the simulation is started with all bins full, it is likely that the steady state will be "all bins full"! If the simulation is started with all bins empty, then hopefully the steady state which is reached will be one in which the bins are less than full.[3] It is the latter steady state which is of interest.

The initial system state which was chosen was for all the bins to be empty. This can be seen by observing the extreme left of the graph of "Mean Fullness/Bin" in Figure 7.4. It is also evident, on examining the first half of this graph,[4] that the mean fullness per bin rises quickly until it reaches a certain point, and shortly after this the system seems to be in a steady state.

This brings us to the final issue, which is how long the warmup period should be. This can be determined by looking at graphs as described above and determining a point at which the simulation seems to have reached a steady state. A margin of safety is then added, since it is always safe to err on the side of making the warmup period too long; the only penalty is increased computation time.

The length of the warmup period needs to be determined for each different algorithm and set of parameters. By examining graphs for many such circumstances, however, it became clear that steady state was always reached relatively quickly — within 200 days, certainly. A very conservative warmup period of 1000 days was selected, since the computation time involved (about five minutes) was relatively small.

6.1.2 Length of simulation run

Once the length of the warmup period has been decided, the length of the simulation run after warmup must be decided upon. Choosing too short a time may cause results which

[3]Recall that overflowing bins are to be avoided at all costs. Too many bins too close to being full results in overflows.

[4]The second half demonstrates behaviour when parameters are changed.

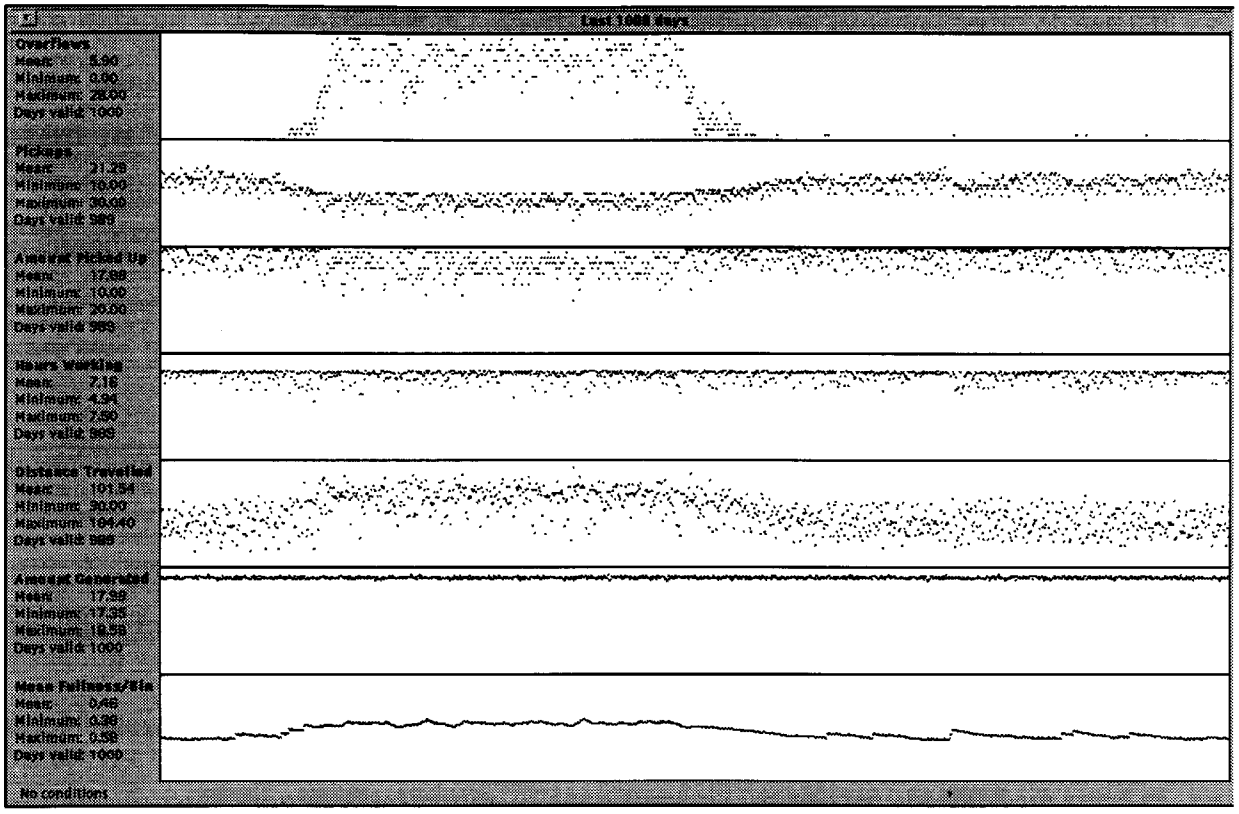


Figure 6.1: Sample overflow recovery

are not statistically significant. Choosing too large a time may be an unnecessary waste of computational effort.

Once again, by observing graphs of simulation statistics, it became clear that the algorithm does not have any extremely long term cycles,[5] but does, in fact, appear to have events of a period on the order of a year and a half (with the data used). A length of 1000 days was then chosen so as to ensure the inclusion of at least one of these events.

6.1.3 Instability of algorithms with respect to overflows

Any algorithm for the problem will be unstable as the production rate reaches a critical value. This is due to the fact that overflows are often costly to handle, since the truck may

[5]On one occasion the algorithm was run for 100 000 days (approximately 274 years).

have to travel a long distance if the overflowing locations are geographically dispersed. Thus, the existence of overflows causes less efficient servicing, which results in more overflows.[6]

It is possible, therefore, for circumstances to conspire to cause a sudden influx of commodity to the system which pushes the system "over the edge". An example of this occurring, and a miraculous recovery (due to circumstances conspiring to help the system back from the edge!) is shown in Figure 6.1.

The main reason for pointing this out is that the number of overflows per day is a very unstable value once it reaches a certain point with increasing commodity production. Any commodity production rate beyond this point should be considered impractical to handle with the given assumptions. Of course, in reality, if this situation arose then a decision would be made to buy an additional truck!

6.1.4 Comparability of results

In order to fairly compare the results of algorithms which use pseudo-random number generators, it is necessary to ensure that the same random number streams are used for different runs. It is also necessary to check that the results are not unduly dependent on the particular random number stream chosen. One important technique that is used is to use separate random number streams for separate purposes to ensure the two uses are independent. In our case, one stream is used for determining the actual commodity production, and one is used in generating routes.

It is also important that the data used (producer locations and production rate distributions) are the same. This is really another facet of the same problem — in short, in order to have comparable results from two algorithms, they must be given the same input!

All these issues were taken into consideration.

[6]This is reminiscent of a service guarantee which the Toronto-Dominion Bank currently makes: if a customer is delayed more than 5 minutes then she or he is entitled to $5. Of course, getting the $5 involves the teller getting a form which has to be filled out and which the customer has to sign, which delays customers currently in the line, which...

6.2 Results

Unless stated otherwise, simulation results presented below are the result of running for 1000 days after a 1000 day warmup period. The following parameter values were used:

- Truck capacity 20 bins
- Truck speed 35 km/h
- Pickup time 12 minutes
- Maximum allowed working time 7.5 hours
- Minimum bin fullness in order to pick up 0.25 bins
- Minimum expected fullness of the truck in order to go out 70% (14 bins)
- Probability truck is operable 1
- Work 7 days per week

The producer locations and production rates used were as given by the local company. There are 452 producers distributed unevenly through a region of approximately 2500 km^2. Approximately half (214) of these had filling periods (inverse of commodity production rate) of 35 days, one quarter (121) filled in 70 days, and one quarter (117) filled in 105 days. These production rates are uniformly adjusted to achieve the desired overall production rate. Without adjustment the overall production rate is 8.96 bins per day; thus, if the desired production rate is 18 bins per day, actual production rates would be double those specified, and thus filling periods would be 17.5, 35, and 52.5 days.

6.2.1 Simple greedy heuristic results

As (strongly) hinted at in Section 3.2.1, simple greedy approaches to the problem do not work well. Simulation results confirm this hypothesis. The results presented below are

based on averages of four simulation runs with different seeds for the random number generator which is used to generate the commodity. The algorithm itself made deterministic choices for these trials.

Amount in bin

Using the deterministic version of the algorithm (Figure 4.2) with formula $g(l)$ (amount in bin) gives the same result as would the greedy heuristic outlined in Section 3.2.1 with respect to the amount in a bin.

The results are shown in Figure 6.2. The bottom graph shows that there is more than one new overflow per day once the production rate reaches 40% of truck capacity. This is an unacceptably high rate.

Once the production rate surpasses 70% of truck capacity, there are so many overflows occurring that more than a quarter of the truck's schedule[7] is predetermined every day, since overflowing locations must be serviced immediately. Once the production rate surpasses 80% of truck capacity, the number of overflows levels off because the system has reached the saturation point — the truck spends every day picking up overflows and never catches up.

The other important thing to note in Figure 6.2 is that the top graph shows that the mean amount of commodity picked up per workday remains roughly constant no matter what the overall production rate is. This happens because of the fullness threshold f given by the "minimum expected fullness of the truck in order to go out" parameter. At low production rates the algorithm will fail to produce a route which is expected to result in the truck being more full[8] than f (set to 70% in this case). This results in the truck not being sent out on that day. The idea is that if the overall production rate is, say, 50% of the truck capacity, then the truck should only have to go out every other day,[9] instead of going out

[7] With the assumptions presented and the data used it is common for the truck to pick up approximately 20 locations per day.

[8] as a ratio of truck capacity

[9] At a minimum; if the truck only went out every other day then it would *always* have to return full.

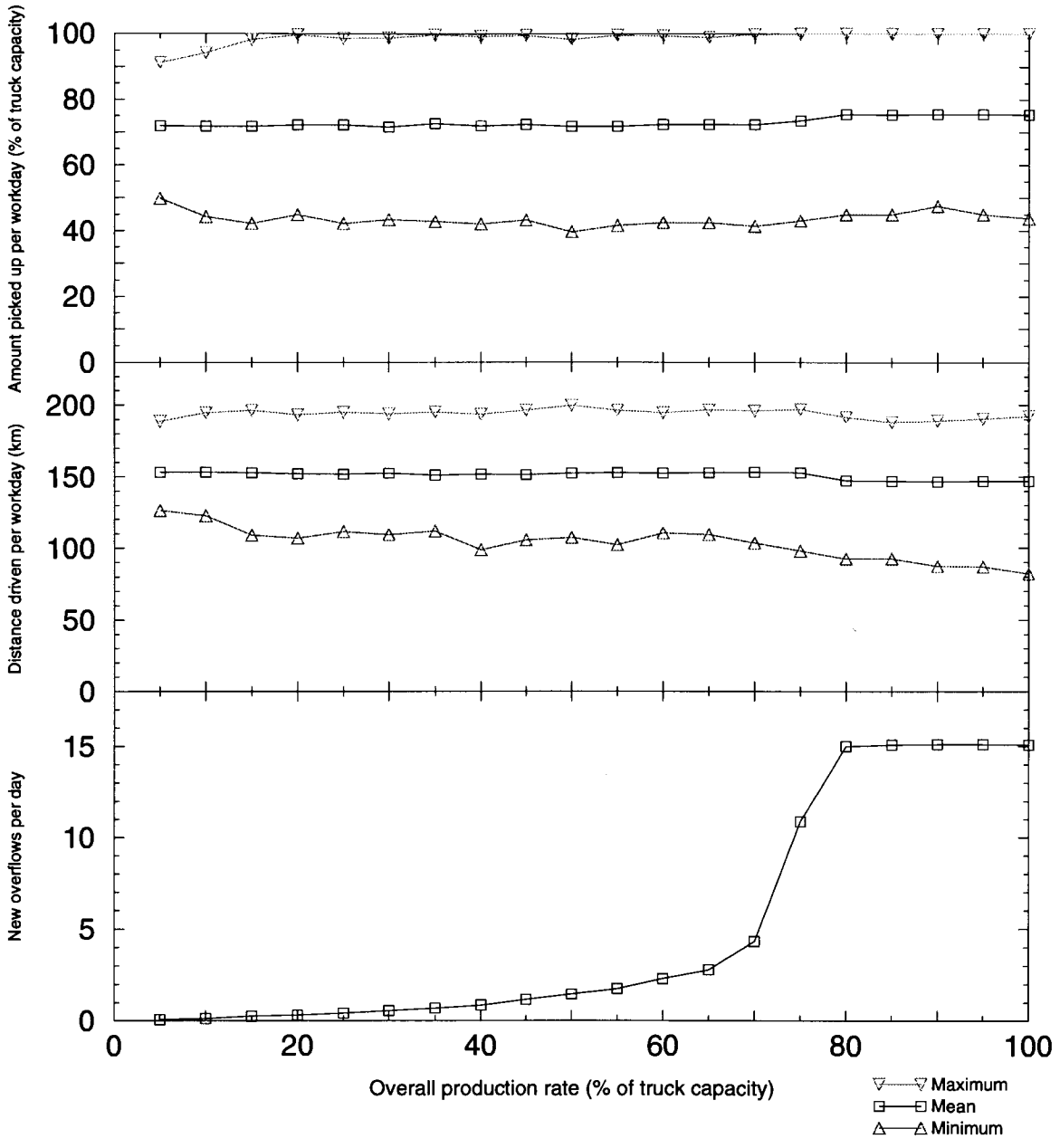


Figure 6.2: Results of simple greedy approach with formula $g(l)$

every day and returning half full.

The minimum amount picked up per workday can be less than the "minimum expected fullness of the truck in order to go out" since the truck is forced to do the route chosen on a day in which an overflow has occurred.

If the algorithm is allowed to send the truck out every day to pick up from any set of locations,[10] then it performs better at low production rates; it can handle a 20% production rate without incurring overflows. Once the production rate reaches 40%, however, the average number of overflows per day is greater than one. After this point the performance is essentially the same as with the parameter settings used above.

Days before overflow expected

Using the formula $\frac{1}{d(l)}$ (inverse of days to overflow) instead of $g(l)$ gives roughly the same results. Thus, the comments made in the previous section apply here as well.

Travel time

Using the deterministic version of the algorithm (Figure 4.2) with formula $\frac{1}{t(l)}$ gives roughly[11] the same result as would the greedy heuristic outlined in Section 3.2.1 with respect to time. See Figure 6.3 for the results.

As one might expect, the truck travels a smaller distance when the algorithm is run with formula $\frac{1}{t(l)}$ than when run with formula $g(l)$. This smaller distance results in more time being available to do pickups, resulting in a higher mean amount picked up per workday. This means the truck goes out on fewer days to do the same job.

The number of new overflows per day shown in Figure 6.3 is about the same as that shown in Figure 6.2. Both are unacceptably high.

Interestingly, allowing the algorithm to send the truck out every day (as mentioned above with respect to the formula $g(l)$) does not improve performance significantly. The

[10] *i.e.* if "Minimum expected fullness of the truck in order to go out" and "Minimum bin fullness in order to pick up" are both set to zero.

[11] The difference is in the method of choosing the seed location.

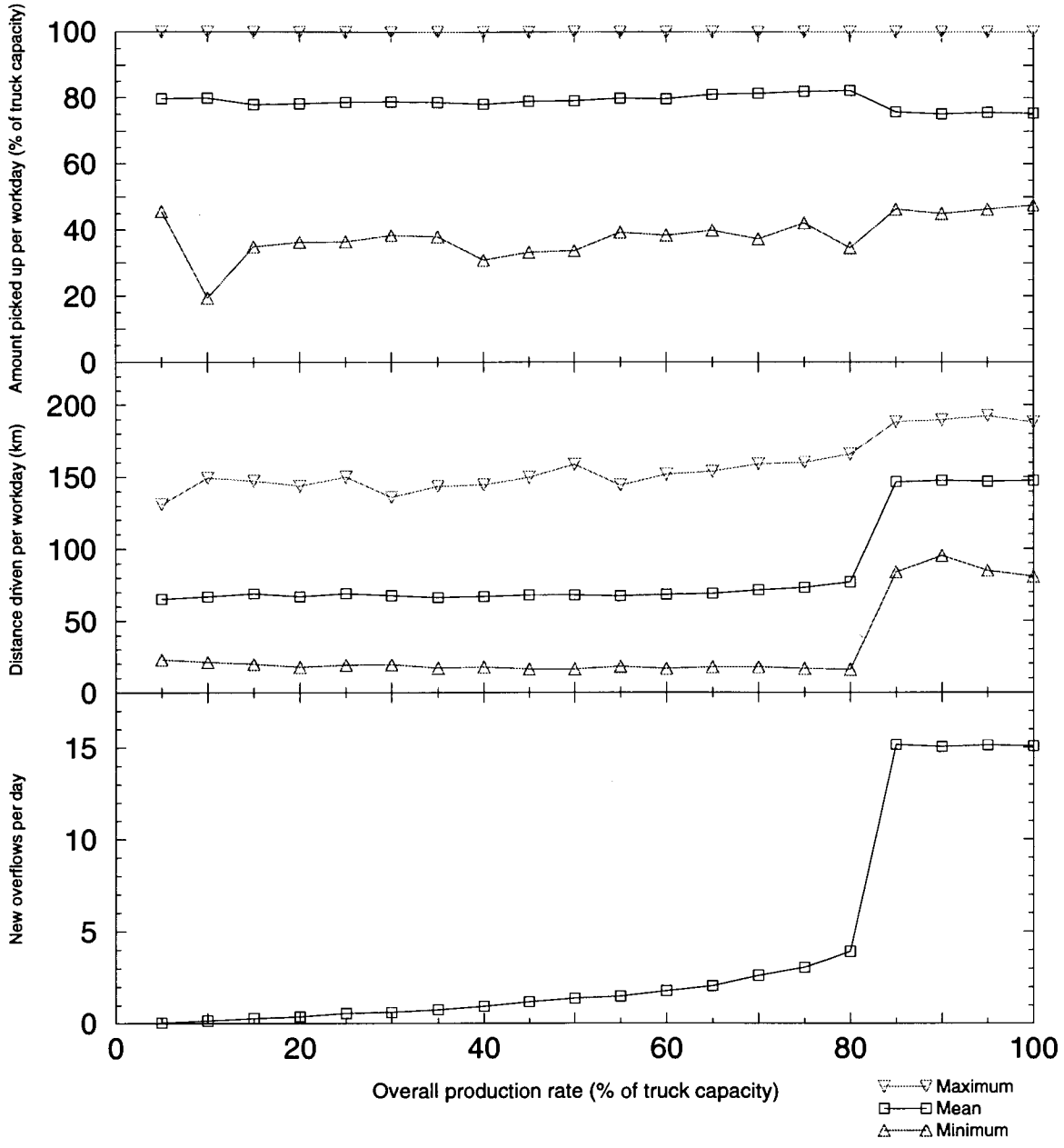


Figure 6.3: Results of simple greedy approach with formula $\frac{1}{t(l)}$

algorithm cannot even handle a production rate of 10% without incurring overflows. This clearly shows that ignoring the amount of commodity at locations is a bad idea!

6.2.2 Sophisticated greedy heuristic results

Motivation

The poor performance of the simple approaches imply that it is necessary to consider more than one of the 3 criteria simultaneously. The three criteria are: amount in bin, days before overflow, and travel time. It is clear that the probability of visiting a location should be proportional to the first criterion, and inversely proportional to the second and third. It is also clear, upon reflection, that all three criteria are required.

Travel time is definitely an important criterion because ignoring travel time will result in the truck travelling longer than necessary. The extra time spent travelling will result in less time being available to actually service the locations.

But why do we need to consider both the amount in a bin and the number of days before overflow? What is the difference between these two quantities, anyway?

As mentioned in Section 5.1.1, these two quantities are related by the period of filling for the location. Thus, for a location which fills its bin every 10 days, being 5 days away from overflow means the bin is half full. A location which fills its bin every 100 days and is 5 days away from overflow, however, is 95% full. Thus, for example, considering only the number of days before overflow would result in these two locations being treated equally, even though the latter location is better in terms of picking up a large amount of commodity.

Only considering the amount of commodity in a bin causes problems, however, since a location which fills in 10 days and which is 70% full (and thus will overflow in 3 days) is rather more important to pick up than one which fills in 100 days and which is 70% full (and thus will overflow in 30 days).

But how are they to be combined to produce one result? Two immediately obvious possibilities are to sum them together, or to multiply them together.

"Additive" formula

If the quantities are summed together, then there is a question of the relative importance of each quantity. The quantities must be scaled to reflect this.

In order to do this we must first know the range of the quantities to be considered. The quantity $t(l)$ was defined in such a way that the smallest value possible is t_p, corresponding to a location which is distance 0 away (*i.e.* right next door). The quantity $g(l)$ was defined to take values 0 (bin is empty) to 1 (bin is full). The quantity $d(l)$ was defined in such a way that the smallest value possible is 1 (any location which is one day away from overflowing is thus considered equivalent to a location which has overflowed).

Since the formula is to include terms which are inversely proportional to $t(l)$ and $d(l)$, and since we know the minimum values $t(l)$ and $d(l)$ may have, we know the maximum values these terms can have. The minimum value the terms can have is greater than zero, since $t(l)$ and $d(l)$ are both positive for all locations l at all times.

The formula includes a term proportional to $g(l)$, and we know the maximum and minimum value it can have. Thus, for example, we could throw the three simple terms together to get

$$\frac{t_p}{t(l)} + g(l) + \frac{1}{d(l)}$$

and know that each term is in the range 0 to 1. This would mean that the value of the formula for a location a whose bin is full ($g(a) = 1$) and which is expected to overflow ($d(a) = 1$) but which is a very long way away ($t(a)$ is very large) would have a value of 2. This would make it an equally likely location to visit as a location b which is right next door ($t(b) = t_p$), half full ($g(b) = 0.5$) and expected to overflow in two days ($d(b) = 2$).

Whether this relative weighting of the three terms is "good" is not immediately clear. There does not seem to be any objective measure of the "goodness" of any particular relative weighting of the terms aside from the effect the overall formula has within the algorithm. After much experimentation, the best formula determined was

$$\frac{t_p}{2t(l)} + g(l)^2 + \frac{1}{d(l)}$$

The results of running the algorithm with this formula are shown in Figure 6.4. Note the drastic improvement of the algorithm with this formula relative to the algorithm with the "simple" formulas. New overflows have been cut to zero up until the production rate reaches 65% of truck capacity.

"Multiplicative" formula

If the quantities are multiplied together, then very small values of one quantity have a very large effect on the overall value. This means that if any of the three quantities is extremely small (indicating that the location is undesirable from the point of view of that quantity), then the overall weighting will be small.

The simplest possible such formula, namely

$$\frac{g(l)}{d(l)t(l)}$$

works amazingly well. The results of running the algorithm with this formula are shown in Figure 6.5. Note that almost no overflows occur with the production rate equal to 90% of the truck capacity!

6.2.3 Randomized heuristic results

All the results presented thus far have used the deterministic version of the algorithm. Similar results are obtained using the randomized version.

Using the randomized version, however, brings up the issue of scaling of weightings. With the deterministic version the location with the largest weighting is chosen, and thus the difference between the largest weighting and other weightings is not relevant. This difference *is* relevant for the randomized version!

Suppose, for example, that there are 990 locations with weighting 1 and one location

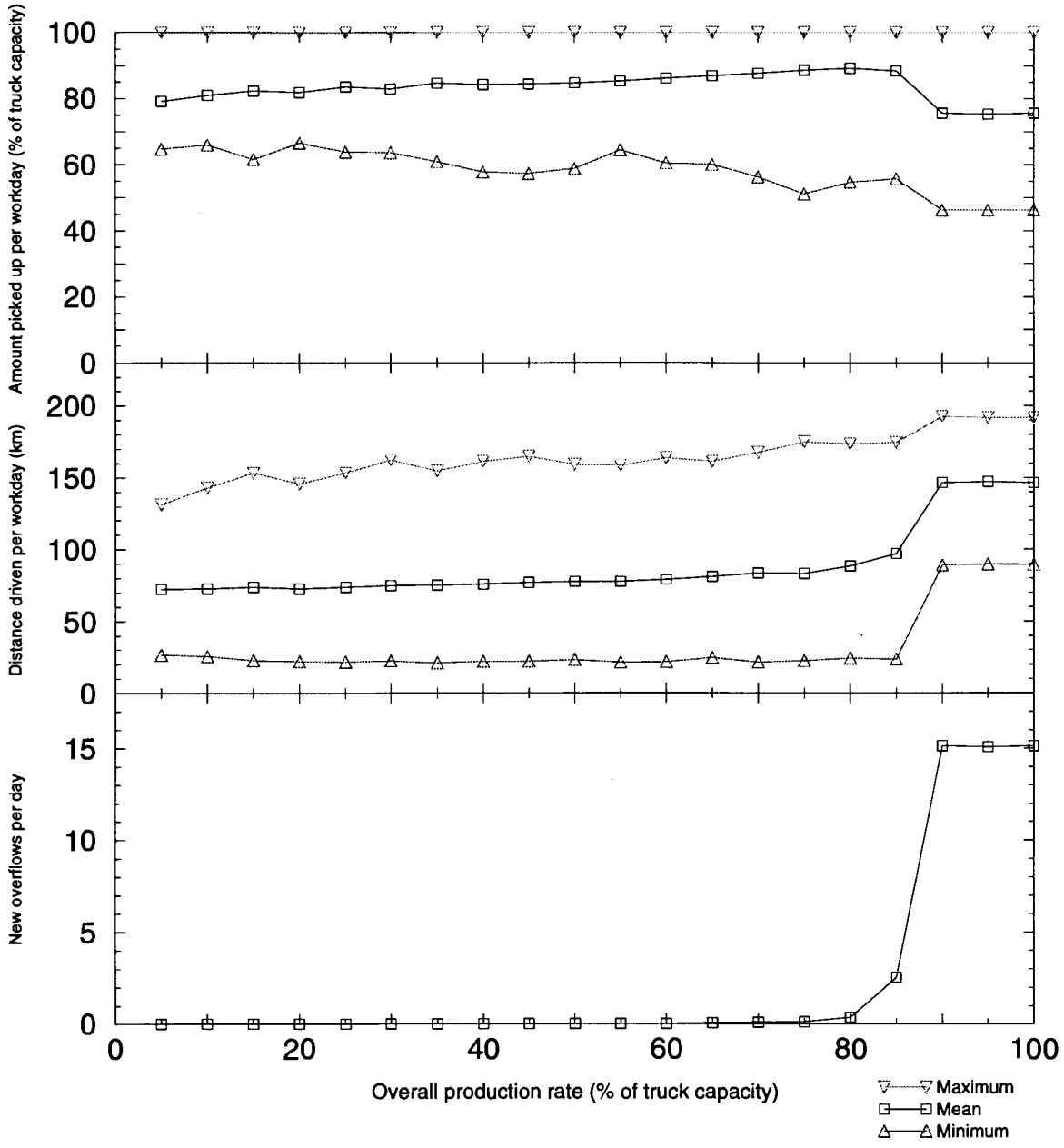


Figure 6.4: Results of greedy heuristic with "additive" formula

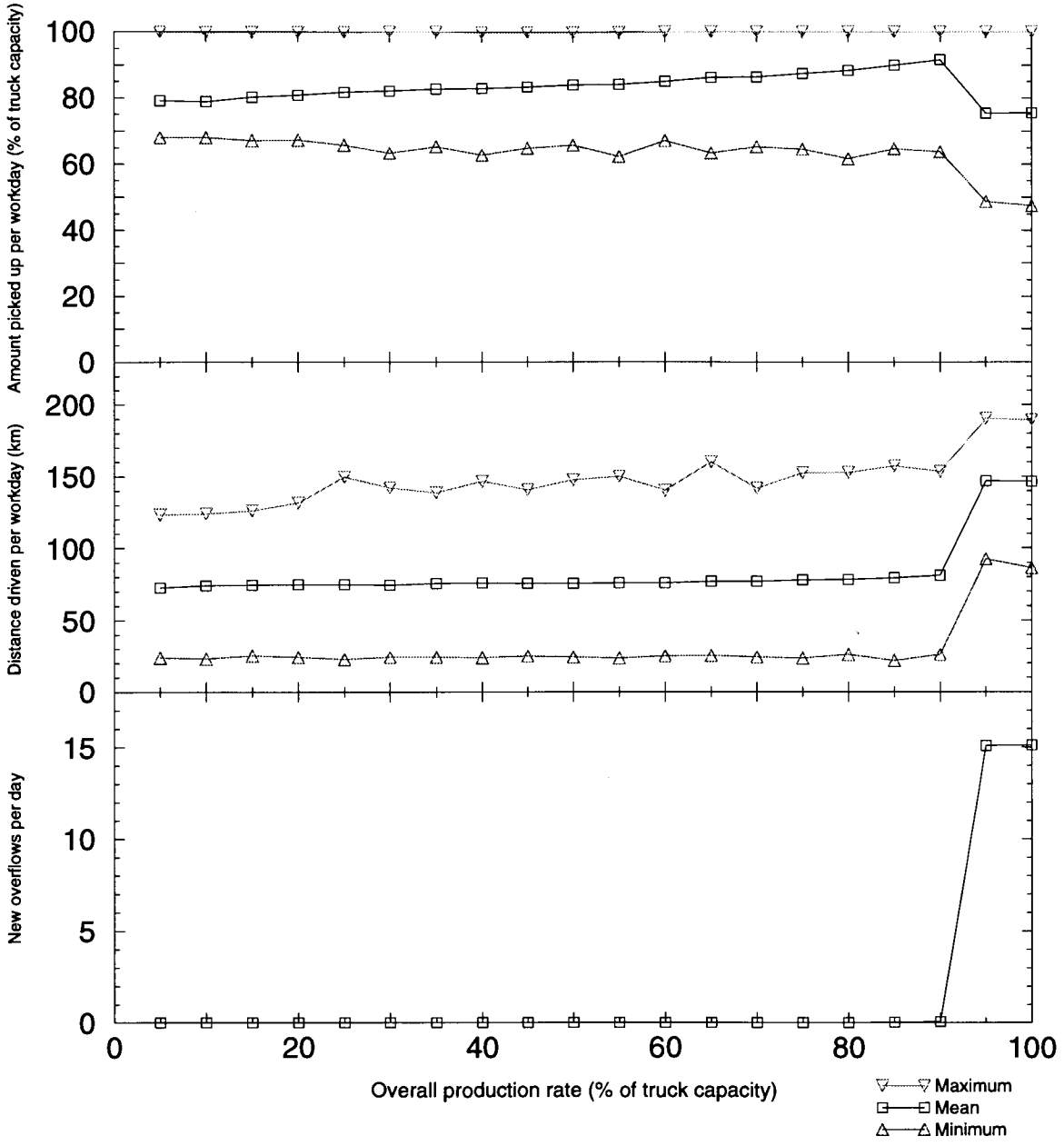


Figure 6.5: Results of greedy heuristic with "multiplicative" formula

with weighting 10. Then the location with weighting 10 has only a 1% ($\frac{10}{1000}$) chance of being chosen, despite the fact that it is obviously the superior choice.

One solution is to raise the original formula to some "appropriate" power, so that the difference between the good and the bad is greater. If, for example, the above weightings were squared, then the "good" location would have a weighting of 100, and would thus be chosen with probability 9% (actually $\frac{100}{1090}$). If the weightings were again squared, then the "good" location would now be chosen with probability 91% ($\frac{10000}{10990}$). Clearly this process can continue indefinitely, but in the limit specifying such a formula results in the randomized algorithm behaving in the same way as the deterministic one would (*i.e.* the location with the largest weighting is chosen with probability 1).

Thus, the exponent with which to raise the original formula must be chosen by trial and error to achieve a balance between the deterministic[12] version and a version with insufficient spread between "good" and "bad" weightings.

For the multiplicative formula with the data used (*e.g.* mean number of days to fill bin) the exponent 16 seemed to provide good results. See Figure 6.6 for the results of running the randomized algorithm with formula

$$\left(\frac{g(l)}{d(l)t(l)}\right)^{16}$$

Despite the fact that the bottom graph in Figure 6.6 shows a very large number of overflows at a production rate of 90%, in certain cases[13] the algorithm can handle a production rate of 90% without experiencing a large number of overflows. Figure 5.4 gives the textual output for one such execution of the algorithm, showing that only 42 new overflows occurred on 1000 days, or roughly one every 3 weeks on average. This rate is acceptable in practice, and is certainly better than what the company is experiencing at present with a production rate of roughly 50%, not 90% !

This example can also be used to illustrate just how well the algorithm can do relative

[12] Practically speaking, of course, there is a limit to how large the exponent can be before inaccuracies occur due to the finite representation of real numbers within a computer's memory.

[13] *i.e.* when the simulation runs with certain random number seeds

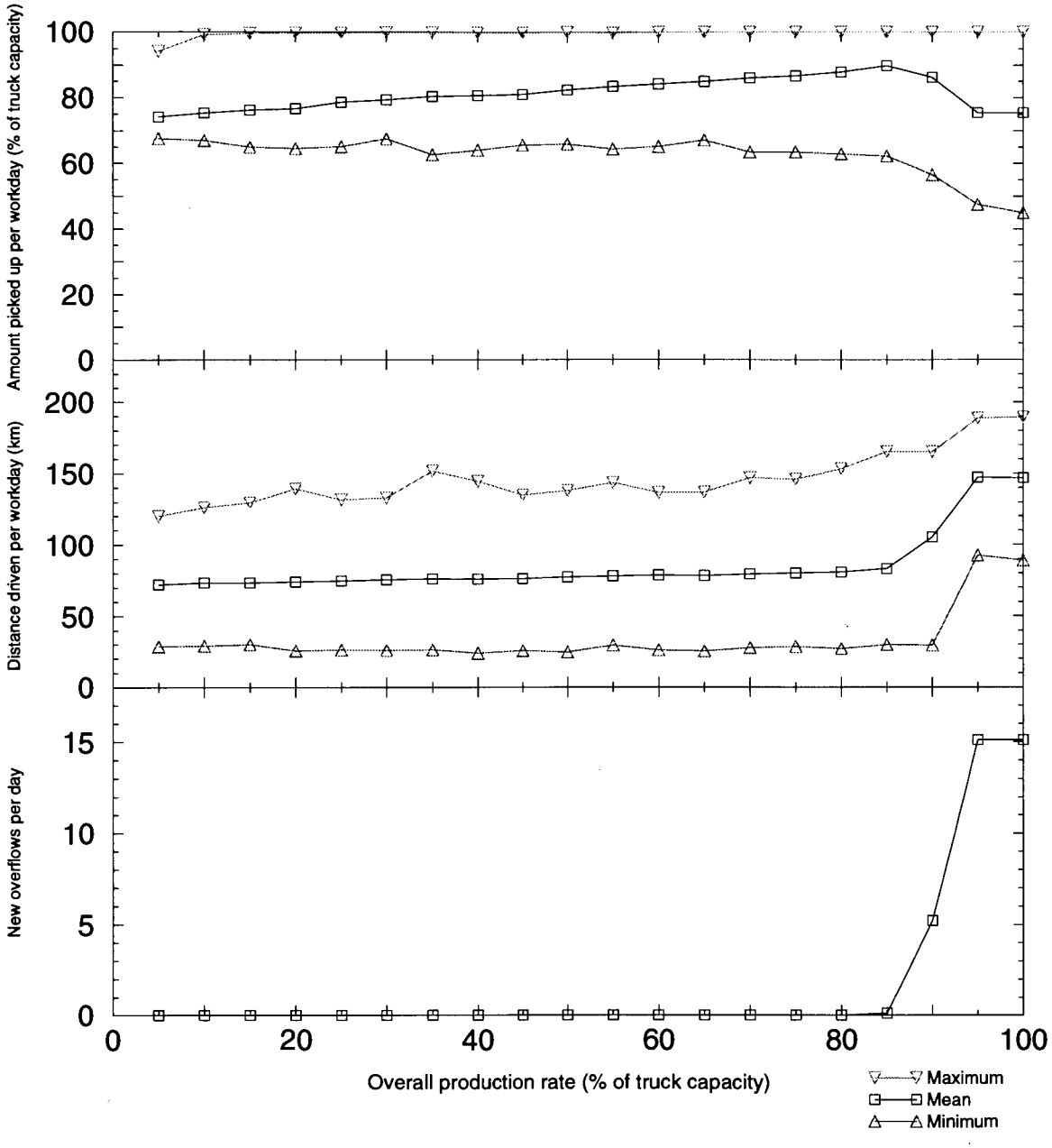


Figure 6.6: Results of randomized greedy heuristic with "multiplicative" formula

to the theoretical limit. In order to handle a 90% production rate with a truck capacity of 20 bins (in other words, a production rate of 19 bins/day) without overflows, a truck must visit, on average, at least 19 locations per day. At 12 minutes per pickup, this translates into 3 hours and 48 minutes spent picking up commodity, leaving 3 hours and 42 minutes for travel. At 35 km/h, this means the truck can travel a maximum distance of 129.5 km. It is difficult to determine what the smallest mean travel time achievable is, but it is important to note that there are producers which are situated more than 64.75 km away from the depot! Thus, clearly it is not possible to pick up 19 bins of commodity on days in which these distant locations must be visited, simply because there is not enough time to get there and back *and* pick up the commodity at 19 locations.

Even with a "perfect" algorithm, then, it may not be possible to prevent overflows occurring, since there must be days in which more commodity is produced than collected. This means that the flexibility required to select the "best" route may be lost due to an overflow occurring at an inopportune location.[14]

It must be admitted that the example chosen was one of the best possible for the algorithm. Due to the difficulty which any algorithm will encounter with recovering from overflows, (as explained in Section 6.1.3) it would be unwise to claim that the algorithm can handle a production rate close to the "critical" area. A reasonable claim, then, is that the randomized heuristic algorithm can handle a production rate of 80%. The probability of the algorithm entering an unrecoverable state at this production rate seems, from observation of many simulation runs, very low.

6.2.4 Comparison of results presented thus far

One important way to characterize the performance of the algorithm with different formulas is by the number of new overflows which occur per day. This appears as the bottom graph in Figures 6.2, 6.3, 6.4, 6.5, and 6.6.

In order to aid in making comparisons, these five curves are plotted together on the

[14]For example, a producer which is situated in an isolated area or in an area which has recently been serviced.

same graph. See Figure 6.7 for the result.

The graph clearly shows how superior the "sophisticated" greedy heuristics are relative to the "simple" greedy heuristics. It also shows that the sophisticated heuristics seem to each perform extremely well up until a certain point at which the performance becomes terrible. This is due to the instability described in Section 6.1.3. This "critical" production rate seems to be lower for the additive formula than for the multiplicative (and hence the multiplicative is better).

One important thing to note from the graph is that the difference in performance between the deterministic multiplicative heuristic and the randomized multiplicative heuristic may not be as clear cut as is shown. The simulation results used to determine the values in the graph showed that the deterministic version handled the 90% production rate well in all cases, but the randomized version only handled it well in half the cases. Perhaps with a larger sample size or a longer simulation length the deterministic version would have done poorly sufficiently many times that the graph values for the deterministic and randomized would be approximately the same. The result would be that the two graphs would then appear to be identical. Thus, the best conclusion that can be made is that the deterministic version does no worse than the random version in this case.

In any event, as stated earlier, not much confidence can be put in the ability of an algorithm to work without many overflows at a production rate close to the highest possible. To be safe, the algorithm should be used with a production rate slightly below this "critical value".

Actually, the "critical" production rate is ill defined; the ability of an algorithm to work well depends on the actual events which occur. Many sets of events which correspond to the same production rate will produce radically different results. For example, one producer located 50 km away from the depot suddenly producing 50% more commodity per day than usual will likely cause more problems than a producer located 10 km away doing the same thing.

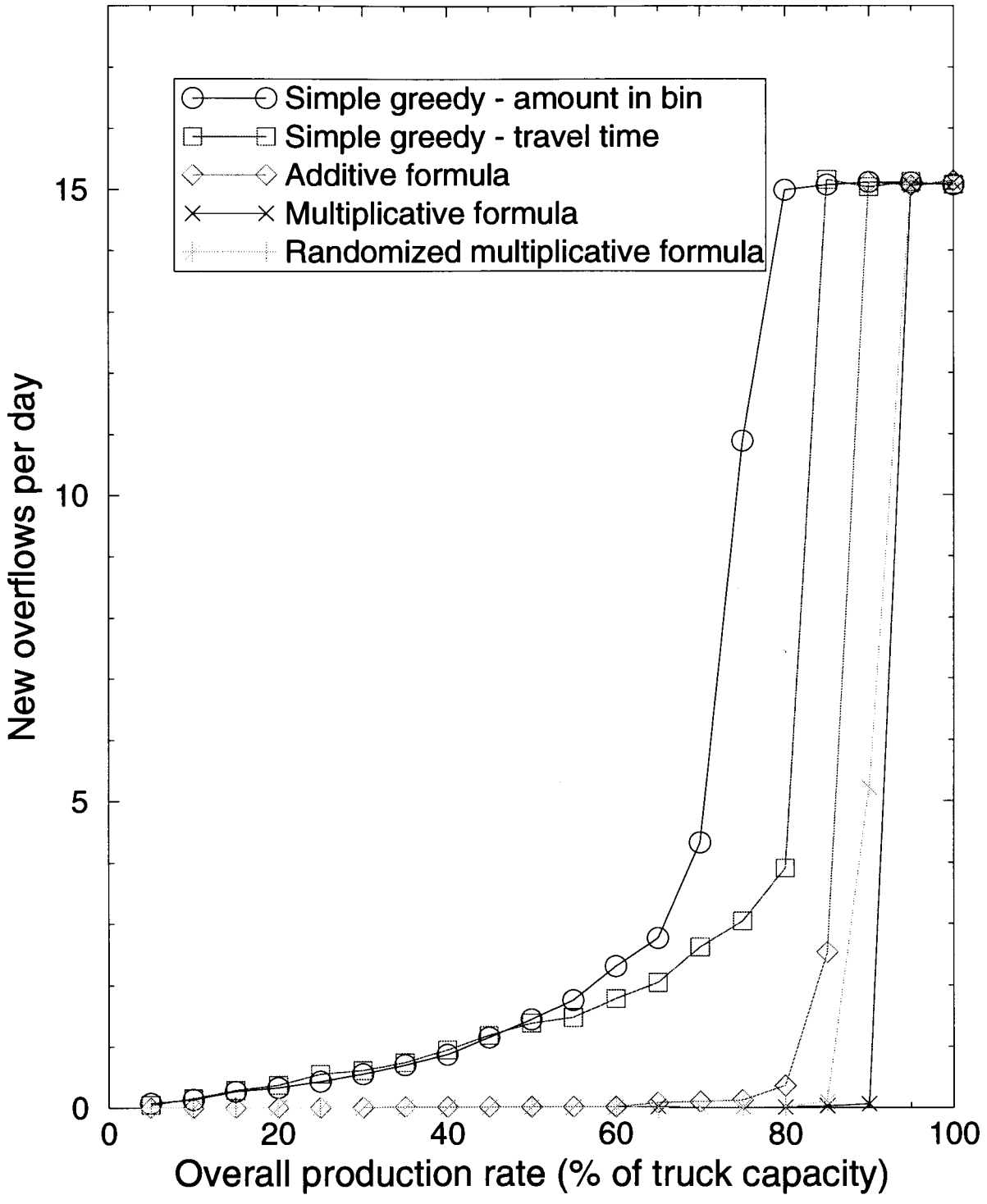


Figure 6.7: Comparison of results

6.3 Results with different assumptions

The previous section demonstrated how well the algorithm does with a particular set of assumptions. In order to demonstrate that this performance is not in some way related to the assumptions made, we must investigate the performance of the algorithm under other assumptions.

6.3.1 Assumptions about pickup time and truck speed

The assumptions made about pickup time and truck speed are critical to the feasibility of servicing the locations given. With the sample data the algorithm performs approximately 25 pickups per day (at 12 minutes per pickup this is a total of 5 hours), and travels about 80 km (about 2.5 hours). Thus, increasing pickup time by, say, 2 minutes, would result in an additional 50 minutes being required to do the same number of pickups. This additional time required simply is not available within the days the truck already goes out. Thus, the algorithm can only cope by sending the truck out on more days.

Similarly, decreasing truck speed from 35 km/h to 25 km/h requires an extra 55 minutes per day. Reducing to 30 km/h requires an extra 23 minutes.

Test runs were made with various values of truck speed and pickup time. These tests showed that the heuristics which performed best previously still perform best, but in many cases there simply is not enough "room to maneuver" to allow the algorithm to perform well.

In other words, if the assumptions about these critical quantities are invalid, then the algorithm will not perform well, but then neither will *any* algorithm.

Note that the assumed maximum working time is directly related to the assumptions about pickup time and truck speed; *i.e.* assuming a greater length of time is required to perform duties gives the same result as restricting the total length of time required to perform those duties.

6.3.2 Assumptions about truck availability

All results presented thus far presume that the truck is always available. In real life, of course, this is often not the case; five day work weeks are common, drivers are occasionally sick, and trucks occasionally break down. How well does the algorithm perform under these conditions?

Commodity which collects on days for which a truck is unavailable must be collected at a later date. Thus on those days that the truck does work, there must be more commodity picked up than is generated.

This behaviour can be seen by examining the "mean amount per bin" graph. The mean amount per bin increases when the truck does not go out, and (hopefully!) decreases when the truck does go out. The result is a sawtooth shaped curve.

The peak value of the "mean amount per bin" is important, because large values tend to be well correlated with overflows occurring. This is the problem with not sending the truck out every day; it is more likely that overflows will occur. If the days the truck does not go out are regular (*e.g.* the truck stays at the depot on weekends), then the problem could be avoided somewhat by using the "lookahead" extension proposed in Section 4.4.1. If the days the truck does not go out are random, there is not much any algorithm can do to "protect itself".

Running the simulation with various truck availabilities gave evidence that this factor does not affect algorithm performance significantly.

6.3.3 Rural setting

In order to show that the algorithm's performance was not in some way related to the geographic distribution of producers in the sample data provided by the local company, many tests were run on data which is of a more regular nature. This could be argued as being a test of the performance of the algorithm in a rural setting (one which is characterized by regularly spaced clusters of similar numbers of producers) rather than an urban one (one which is characterized by irregularly spaced clusters of varying numbers of producers).

Unfortunately, it is difficult to determine what data would result in a fair comparison. How can the travel time among two different data sets be compared? What about the geographical distribution of production rates?

One data set which was used was simply a set of 400 producers spaced at 1 km intervals on a 20 km by 20 km grid with the following coordinates:

$$\{(x,y)|x \in \{-10,-9,\ldots,-1,1,2,\ldots,9,10\}, y \in \{-10,-9,\ldots,-1,1,2,\ldots,9,10\}\}$$

The depot was given the coordinates (0,0), and all producers were given the same production rate. No barriers existed, and thus the distance to any given location (x,y) was simply $x+y$. With this data set the algorithm could handle very high (greater than 90%) production rates with no overflows.

This data set seemed to be a slightly unfair comparison in terms of travel time required, however. Thus, a data set with the coordinates multiplied by two (*i.e.* 400 producers on a 40 km by 40 km grid) was constructed. With this data set a production rate of 80% could be handled with very few overflows. This data set would seem to involve more travel time than the "urban" data set provided by the local company.

6.4 Other results

This section presents other significant findings which resulted from testing the algorithm.

6.4.1 Best performance possible

Figure 6.8 demonstrates how well the randomized algorithm can do. The assumptions made were the same as those listed in Section 6.2 except that the truck speed was increased from 35 km/h to 50 km/h, and the pickup time was decreased from 12 minutes to 9 minutes. Under these more favourable assumptions the algorithm (using the same multiplicative formula as used earlier) was able to service all locations for 1000 days without any overflows

occurring, with a commodity production rate of *98%* of the truck capacity.

6.4.2 Adaptability

On many occasions while running the simulation, critical parameters such as the commodity production rate were changed by significant amounts. The result was usually a "spike" in the graph of the number of overflows, followed by a quick settling down into a stable pattern.

In real life such rapid change is very rare. If critical parameters were changed in stages — for example, raising the production rate from 60% to 70%, waiting 30 days, then raising it to 80% — rather than all at once, no overflow "spike" was observed.

The real world problem allows for the addition and deletion of producers. Given the number of producers the company deals with (roughly 500), it is unlikely that a 10% change in production rate will happen quickly, since this is likely to require 50 new producers.

Thus, it seems fair to claim that the algorithm is able to adapt to changing requirements quickly enough.

6.4.3 Algorithm running time

One important aspect of the algorithm which should be emphasized is its speed, which allows it to handle very large problem sizes.

To demonstrate this, a data set consisting of 2000 producers was constructed. With this many producers it took approximately 2 seconds (on a SPARC-10 computer) for the algorithm to construct a route. The running time of the algorithm seems to be linearly related to the number of producers.

The speed with which solutions can be generated allows our algorithm to be used in ways that slower algorithms cannot. For example, our algorithm can be used interactively, allowing the user to investigate how changing conditions will affect the performance of the system.

SUMMARY

Number of restaurants: 452
Simulation over 1000 days
Truck operated on 1000 days

Assumptions:

Truck capacity:	20.00 bins
Truck speed:	50.00 km/h
Pickup time:	9.00 minutes
Max allowed working time:	7.50 hours
Min fullness to pick up:	0.25 bins
Min fullness to go out:	70.00%
Prob truck is operable:	1.00

Work on: Monday Tuesday Wednesday Thursday Friday Saturday Sunday
Formula: gdt*/#*#*#*#*

Statistic	Total	Mean	Minimum	Maximum	Per?
New overflows	0	0.00	0	0	day
Overflows	0	0.00	0	0	day
Days of overflow: N/A					
Pickups	33069	33.07	27	42	workday
Amount picked up	19596.9	19.60	15.67	20.00	workday
Hours working	6751.4	6.75	5.55	7.50	workday
Distance travelled (km)	90067.6	90.07	36.60	148.20	workday
Amount generated	19590.8	19.59	18.96	20.42	day
Amount lost	0.0	0.00	0.00	0.00	day
Fullness/bin/day	154006.7	0.34	0.32	0.36	day

Figure 6.8: Results of simulation with more favourable parameters.

6.4.4 Performance of randomized versus deterministic algorithm

Results given earlier in this chapter indicate that the randomized version of the algorithm performs almost but not quite as well as the deterministic version. Why consider using the randomized version then?

This question was answered earlier in the thesis (see Section 1.5, for example) by stating that, theoretically, the randomized version had the advantage of robustness; it is less likely to encounter a "pathological" input which causes it problems.

To show this, a specific "pathological" input was constructed using the example given in Section 3.2.3 as a basis. The locations given in that example were used, with relative filling periods in the same relationship as shown in Figure 3.1. A set of locations which filled slightly faster than location A of Figure 3.1, and which were located just under two hours travel time away from the depot, were added. These locations are effectively isolated, in that it is not possible to service more than one location on any given day.[15] The idea is that the extra days available to the randomized algorithm due to its more efficient servicing of the locations given in Figure 3.1 could be put to use servicing these remote locations.

The results confirm the reasoning presented. For example, with a production rate of 80% the deterministic version encounters approximately 0.27 overflows per day (1.35% of all locations), whereas the randomized version only encounters 0.11 overflows per day (0.55% of all locations).

[15]Recall that the maximum working time was set to 4 hours for the example.

Chapter 7

Algorithm Performance Visualization Tool

7.1 The need for graphics

The main problem with a text based simulation program is that it is very difficult to see what is happening as a simulation progresses. This is due to the amount of information required in order to see the state of the system, and the difficulty in seeing trends within a list of numbers.

This problem can be solved by graphing the value of a statistic over simulated time. By comparing graphs of different statistics it becomes possible to see relationships among statistics and between statistics and simulation events. For example, Figure 6.1 clearly shows the impact overflows have on the number of pickups, the amount picked up, the distance travelled, and the mean fullness per bin.

Other uses for graphics may arise depending on the simulation model. Often it is useful to depict the state of the system graphically as the simulation progresses. Simulations of factory floor movements, for example, benefit from doing this. In our case it is very useful to see the route chosen as a series of line segments and dots on a map depicting the

geographical area.[1]

7.2 The need for an interactive tool

It is also possible to quickly see the effects simulation parameters have if parameters can be changed interactively and the effects shown graphically. This is especially important for our problem, because it allows for quick comparisons of weighting functions, and the selection of other experimentally determined values such as the threshold below which the truck should not be sent out. Development of a correct and effective algorithm would have been much more difficult without this ability.

Another justification for constructing an interactive graphical program is that it can be used to give demonstrations of the algorithm. Showing a printout or graph of the results of simulation runs is not as impressive or comprehensible as giving an interactive demonstration of the algorithm, with graphical results (such as the routes displayed on a map) which can be grasped immediately.

7.3 The design

For the reasons given above an "Algorithm Performance Visualization Tool" was constructed. It is important to note that this tool was created for a specific purpose, not as a polished, general purpose tool. Thus, there are some "rough edges" and some features which should be added.

The idea for the design of this tool is fundamentally simple: a panel with buttons must exist to allow control of the simulation (run, stop, single step, the setting of simulation parameters) and to allow viewing of simulation data in various ways.

A user should be able to create and destroy windows which provide these "views". There should not be any preset limit on the number of views that can exist at one time.

[1]The line segments only indicate an ordering of the locations; they do not indicate the actual roads to be used. This can be changed if a detailed description of the road network is obtained.

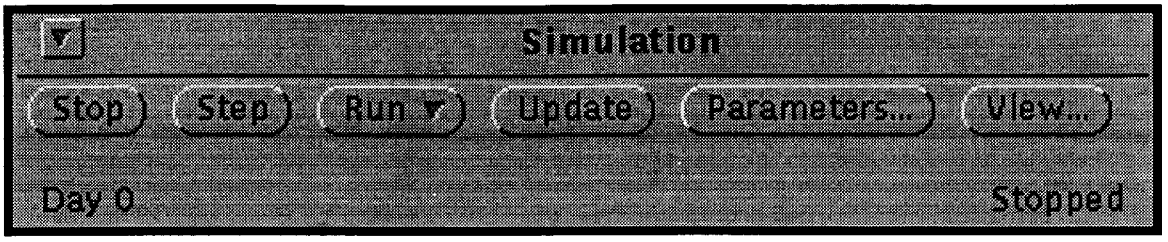


Figure 7.1: Algorithm Performance Visualization Tool main window

The interface between the visualization tool and the simulation program is simple and well defined. No change to the source code for the simulation program is required in order to compile it into either a text-based simulation or a graphical simulation program.

7.4 The implementation

The visualization tool was written in ANSI C using Sun's XView library. The tool has a main window (see Figure 7.1) with buttons allowing the simulation to be started, stopped, or single stepped through one day, a button labelled "Parameters..." and a button labelled "View...". Text indicating the number of days the simulation has been run, and whether the simulation is running or not, appears at the bottom of the main window. The button labeled "Update" causes the information in view windows (see below) to be updated.

Upon pressing the "Parameters..." button, a window appears showing the parameters of the simulation which the user can change. Figure 7.2 shows the parameters window with the user in the process of changing the mean amount of commodity generated per day.

Upon pressing the "View..." button, a window appears showing the options available for viewing data regarding the simulation (see Figure 7.3). There are three possible types of views:

- "Specified Statistics", which shows the mean, minimum, maximum, and days valid[2]

[2]The "days valid" idea is a method of cleanly dealing with statistics which are only really meaningful on certain days. For example, the number of pickups statistic is only valid on workdays.

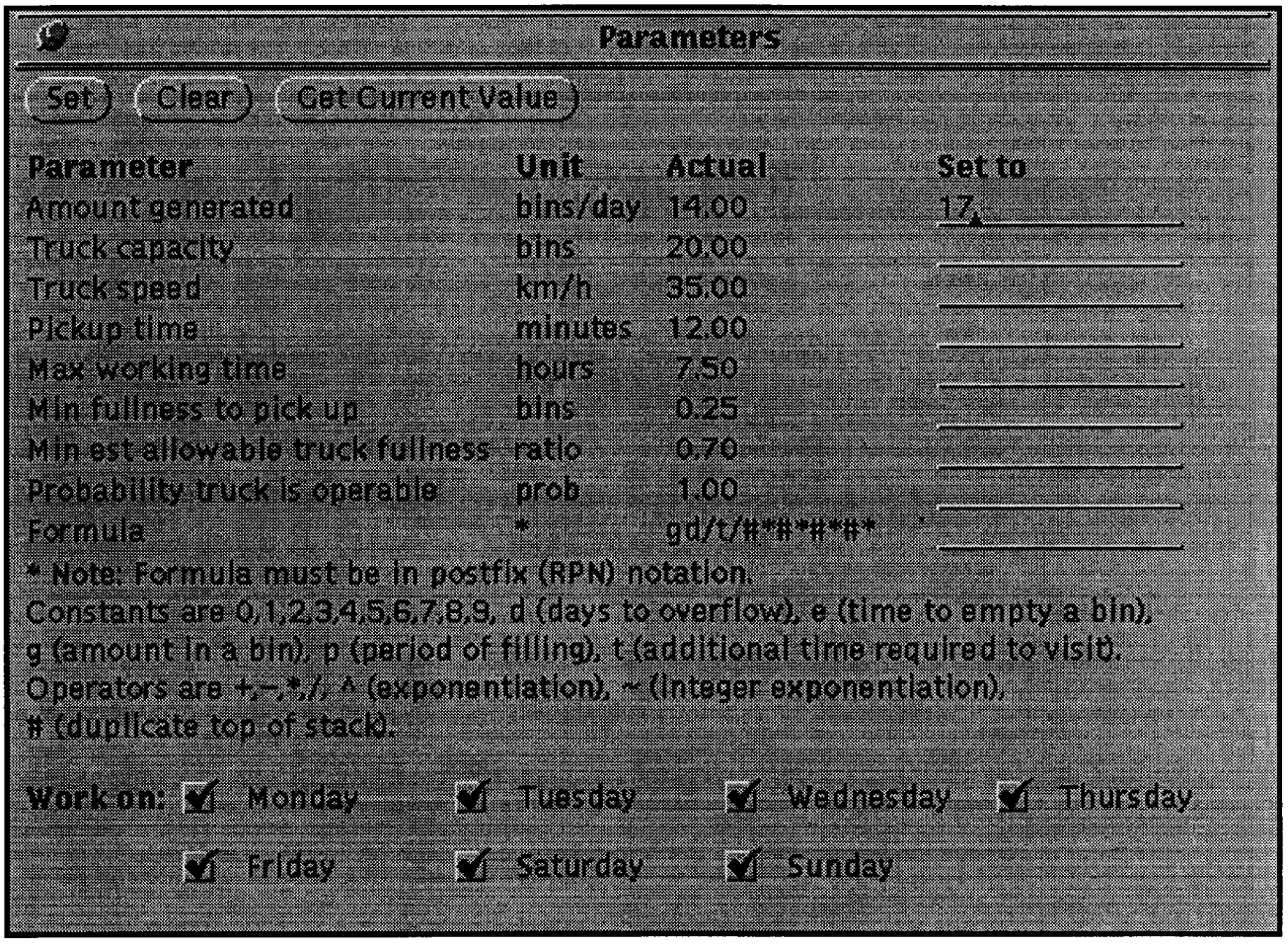


Figure 7.2: Parameters window

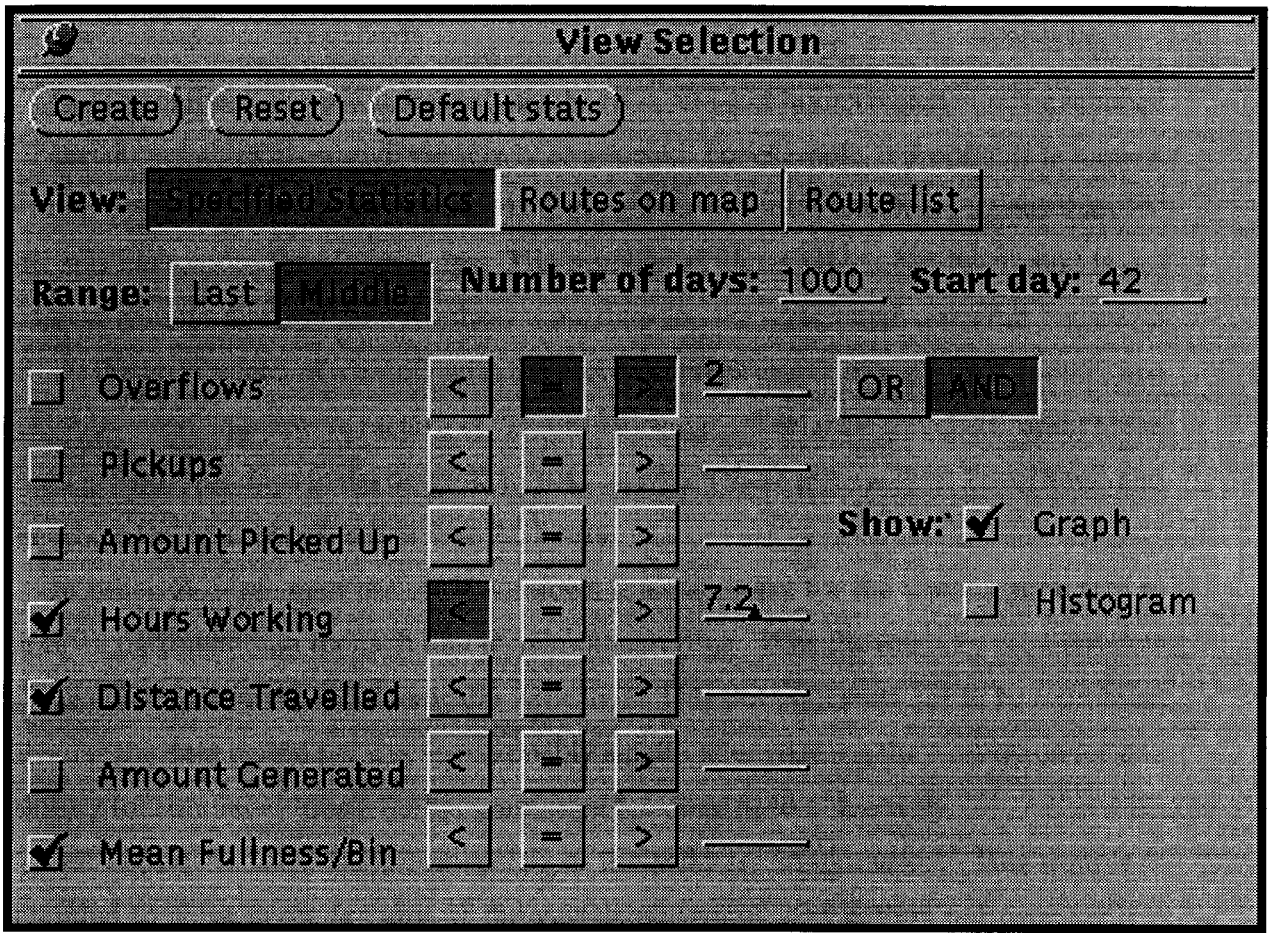


Figure 7.3: View window

for each variable selected, along with an optional graph and histogram (see Figure 7.4).

- "Routes on map", which shows routes as dots connected by lines on a map of the area[3] (see Figure 7.5).
- "Route list", which shows route data in a summary textual form.

For any of these views, the desired days to view, and conditions which must be satisfied for certain variables in order to view, may be selected. For example, it is possible to request a route list for the last 100 days which had overflows, or routes which are longer than 100 km and have more than 20 pickups. The number of views which can be created is not artificially limited; it is possible to show more maps on the screen than is useful! Figure 7.3 shows the view selection window with the user selecting a graph, along with summary text (*i.e.* minimum, maximum, mean), of "Hours Working", "Distance Travelled" and "Mean Fullness/Bin" for days in the range 42 to 1041 on which there were at least 2 overflows and for which the truck worked less than 7.2 hours.

In order to implement the view windows it is necessary for the visualization tool to store all pertinent data for each day the simulation runs. Thus the memory requirements of the simulation grow monotonically with time. This does not turn out to be a problem because the data stored per day is reasonably small - after running the simulation for one century (of simulated time!) the program used less than 8MB of memory.

See Figure 7.6 for a sample snapshot of the program in execution. This sample is intended to show some features of the visualization tool, and is thus not necessarily a useful setup. In the top left of the sample is a map which is zoomed in to show most of Burrard Inlet, with the routes selected for the last 5 days. To the right is a map of the whole region showing the route selected for the last day. To the right is the simulation main window, showing that it is currently running the simulation. Below that is a route

[3]Yes, this map was also digitized by hand. The 15 hour effort was definitely worthwhile, because demonstration programs with nice detailed maps get noticed!

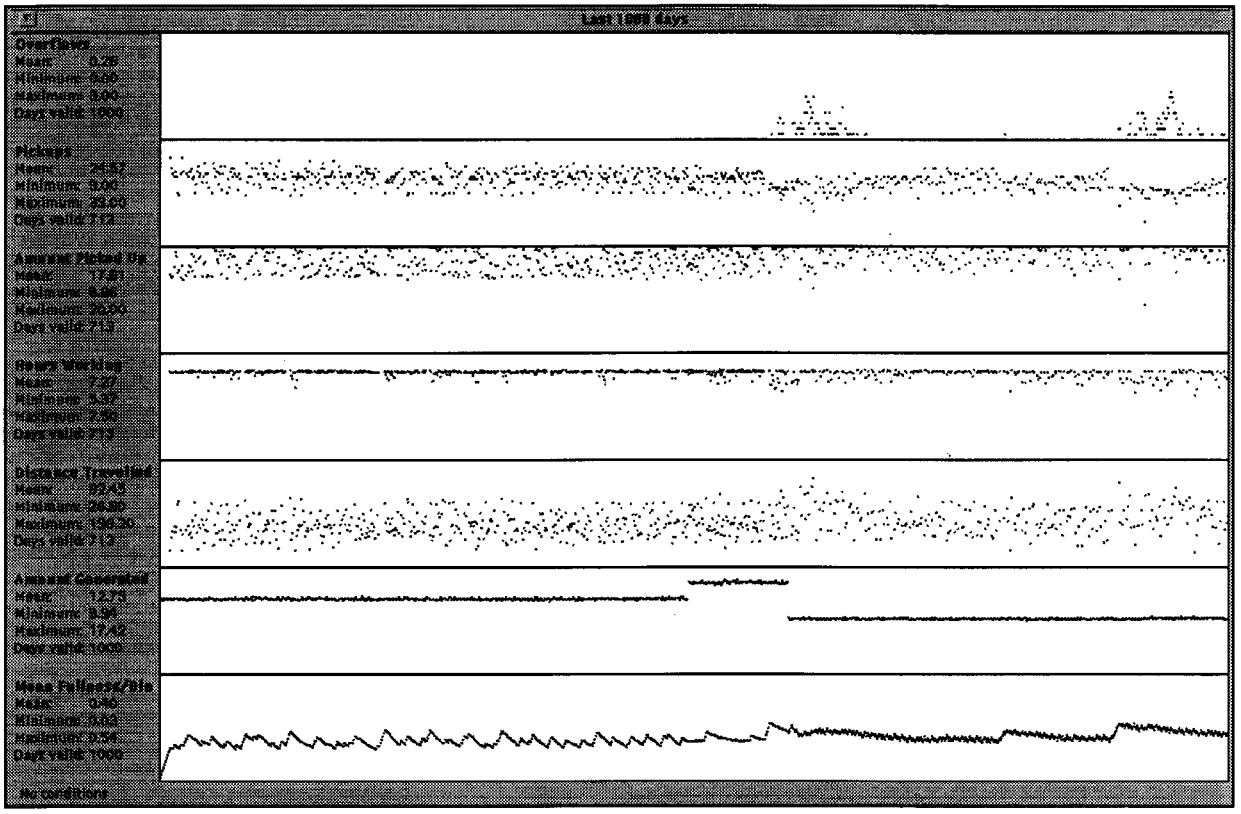


Figure 7.4: Sample graph

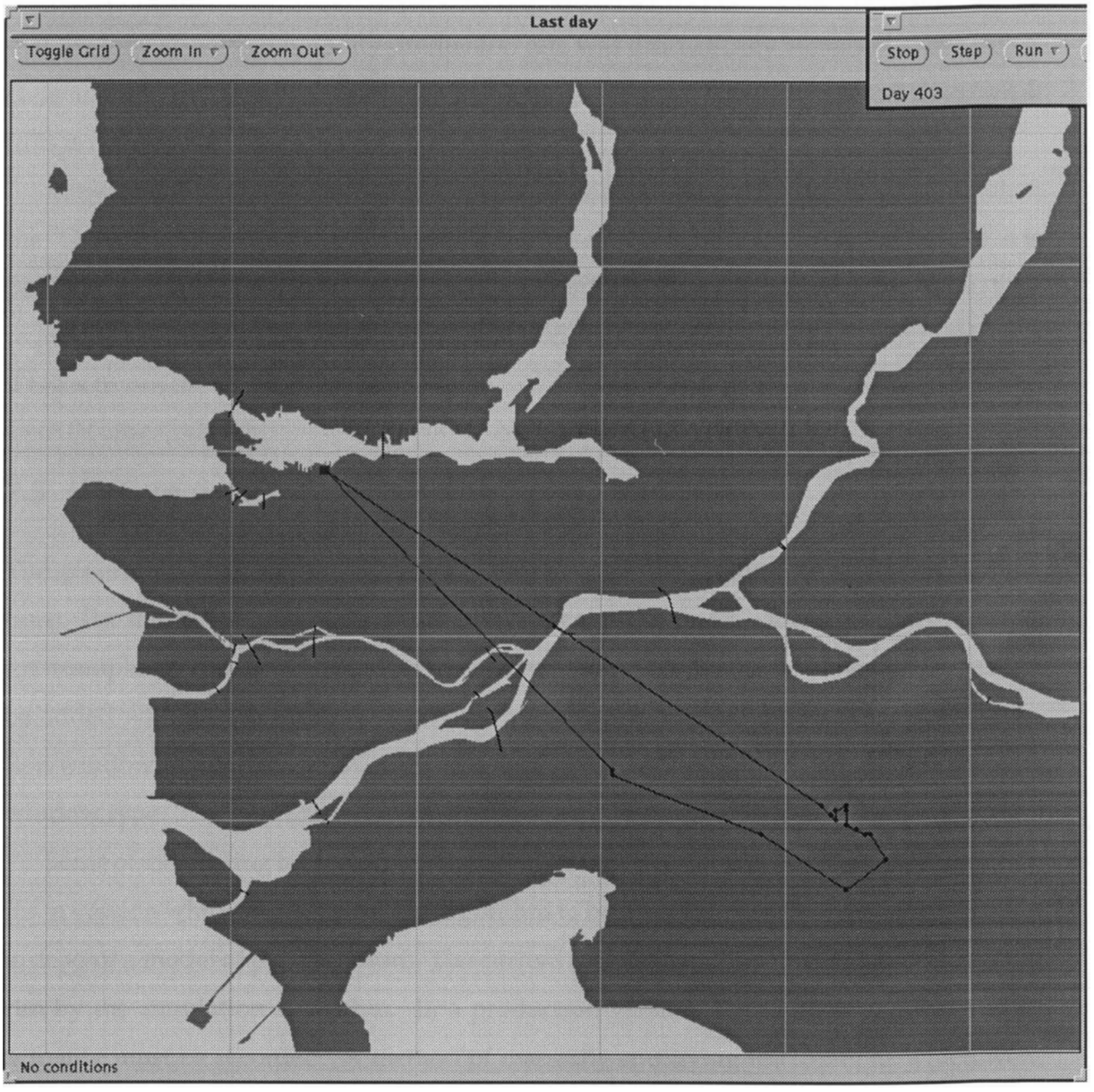


Figure 7.5: Sample route on map

list window. There is another route list window on the left. Below are two windows showing statistics; one is very large and partially covered, showing graphs, and the other shows just textual information about selected values. By inspecting the graph one can see that the total system commodity production rate was dramatically increased at one point, co-incident with the days of operation being increased from 4 days/week to 7 days/week (this is not shown).

View windows are only updated when the simulation stops at the day selected, or when the "Update" button on the main window is pressed. Thus, as is shown in the sample, it is possible for the view windows to not be synchronized. This was allowed due to the time required to update each window each day, which would typically be longer than the time it takes to run the simulation for one day!

7.5 Future uses

The general structure of the visualization tool matches the requirements of many simulations. The tool could be used to investigate other algorithms for vehicle routing problems, for example, by changing the list of simulation parameters available. New view types can be added without too much difficulty, as well — first, add an option to the view selection window to allow creation of the new type, then add the code to implement the new window type.

Some of the existing tool could be incorporated in a "production" version of the program (*i.e.* a version which would allow the algorithm to be used for the actual real life operation instead of a model of the operation). The current tool records the route which was actually run by the simulation algorithm. In a production version, the route which was run in actuality must be recorded. A method of manually specifying the locations which were visited would be required.

Once this is in place, however, the view windows (routes portrayed on a map, textual lists of routes, and graphical displays of route statistics) would all remain useful for displaying historical data. The parameters window would be simplified due to certain

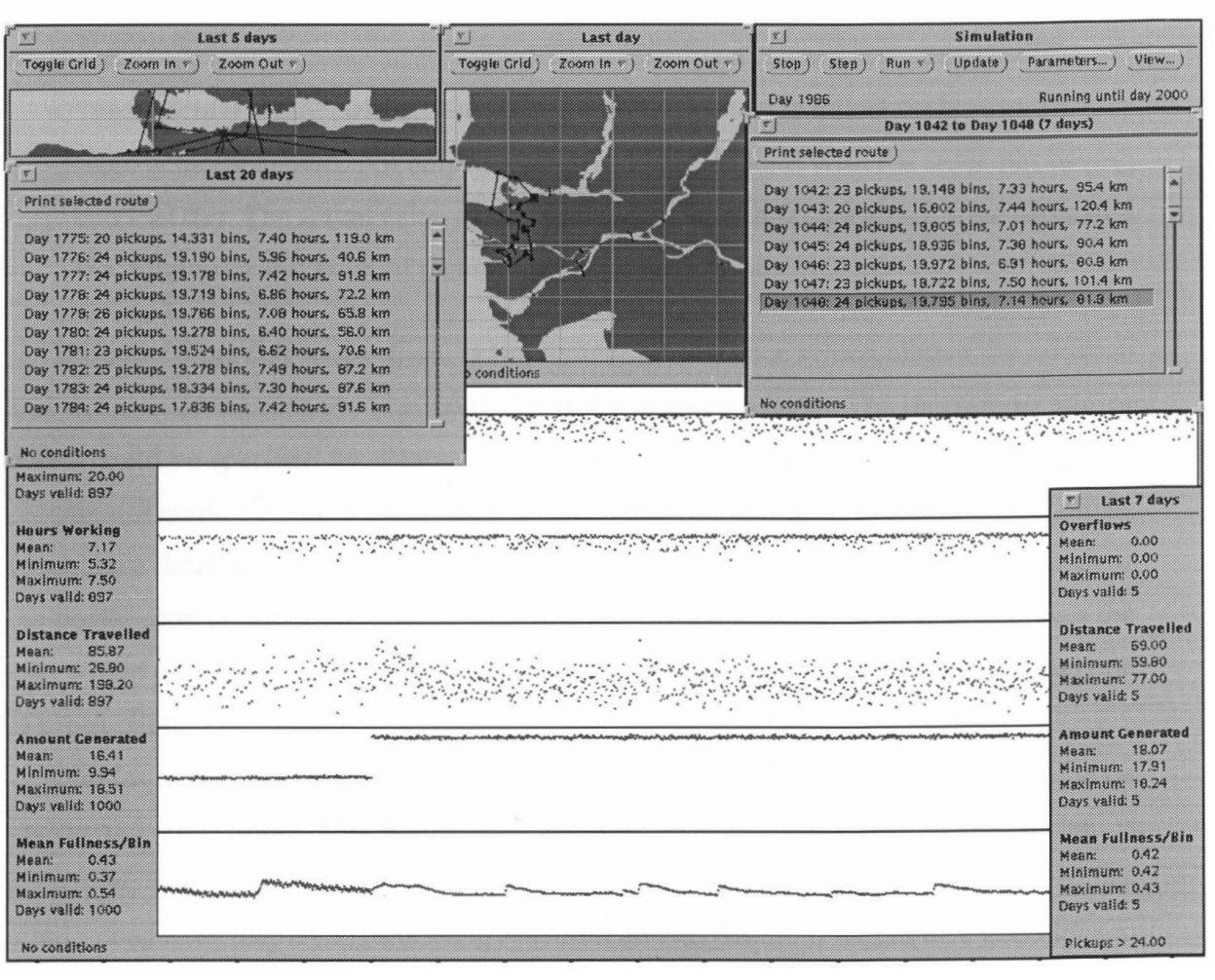


Figure 7.6: Sample simulation run

simulation parameters no longer being relevant (*e.g.* probability of the truck breaking down). The "simulation control" part of the tool would be changed to allow the following operations:

- "generate a route", which would suggest a route to perform based on the current system state.
- "input actual route", which would alter the system state to reflect the locations which were actually visited. An empty route would correspond with the truck not having run that day. The actual value of statistics concerning the route which were estimated (*i.e.* total time and amount picked up) could be entered.

The only thing left to do, then, is implement an interface which allows the addition and deletion of locations.[4] This could be done in many ways. The location to add or delete could be specified by using a pointer device on the map window. Or perhaps the coordinates could be entered manually. The best solution for adding locations, of course, would be a database which allows a translation from easily obtained information, such as the postal code or street address, to coordinates. If such a facility was available then the user would only have to specify the new location's name, address, and expected initial rate of production.

One important possible extension to the tool would be one which would support a multi-truck algorithm. This would mainly involve changing the way in which data is presented; instead of one route being displayed on a map for one day, multiple routes would need to be displayed indicating the route for each truck. It would be a good idea to display each truck's route in a colour which is unique to that truck.

A feature which could be useful would be one which allows the user to query the algorithm as to why it chose a particular location. In addition, a facility for allowing manual modification of a route could be useful in order to learn the tradeoffs inherent in route selection.

[4]This could either be a separate program or an extension of the current interface.

Chapter 8

Conclusions

8.1 Summary of thesis

This thesis presented one particular vehicle scheduling problem. The problem was shown to be one which is difficult to solve optimally. Some possible approaches to solving the problem approximately were given. A route selection algorithm which was developed to solve the problem was described, followed by a description of its implementation and a presentation of performance results obtained from a simulation program. Finally, a tool which was constructed to aid in the development of the algorithm was described.

This thesis makes three main contributions to the vehicle routing literature. First, it gives a simple, efficient algorithm which solves a practical problem much better than the method currently in use. Second, the idea of using a randomized algorithm is explored, something which apparently has not been pursued thus far. Third, a useful tool for development of vehicle routing algorithms was presented.

8.2 Discussion of algorithm strengths and weaknesses

The strengths of the route selection algorithm are as follows:

- It quickly provides excellent solutions.

- It is quite simple in structure. This makes comprehension of the algorithm easier, and allows for quick implementation and efficient operation.
- It is able to adapt to changing requirements.
- It is flexible enough to handle many different situations; *e.g.* both rural and urban settings.
- It is applicable, with minor changes, to a variety of similar vehicle scheduling problems.
- It is able to provide superior solutions to pathological examples due to its use of randomization.

The algorithm's main weakness is that it relies on a weighting function which must be carefully chosen in order to provide good results. Very little theoretical justification has been provided for the weighting function which was chosen, and no procedure is evident for the selection of a good weighting function aside from the "trial and error" method.

8.3 Usefulness of the algorithm performance visualization tool

The visualization tool allowed extensive experimentation with possible algorithms, and thus permitted development of a working algorithm in less time than would otherwise have been possible. The tool was also invaluable in allowing demonstrations of the problem and the algorithm constructed to solve the problem.

Despite being constructed for a single purpose, the tool is in fact generally useful. With modifications it could be made to work with a different scheduling algorithm or with different vehicle routing problems.

8.4 Practical results and future work

Plans are currently being made by a local company to make use of these results. It is likely that the algorithm will first be run "in parallel" with the current system, after which routes suggested by the algorithm will be run and evaluated. The company's current system achieves roughly 50% capacity (*i.e.* trucks come back, on average, half full).

The program implemented to test the algorithm is not directly usable. A method for adding and deleting locations from the database is required, as is a method of informing the algorithm of the locations actually visited by the truck. In other words, the simulated "real world" has to be kept synchronized with the real "real world"! This is a matter of replacing the simulation algorithm and altering the user interface.

We are hopeful that the assumptions made about real world parameters were conservative enough that the company will in fact find that the algorithm works extremely well. Hopefully, the look-ahead extension described in Section 4.4.1 can be implemented quickly, resulting in even more impressive performance. Other simple ideas, such as selecting the "best" among a set of routes generated using different random number seeds, could be explored. After achieving some success with the single truck version, an attempt should be made to implement an algorithm to solve the multi-truck version of the problem.

Other possible areas of investigation include determining a procedure for selecting a weighting function, investigating a theoretical basis for the performance of the algorithm (including perhaps determining a performance bound), improving the algorithm used to compute travel times, and investigating modifications of the algorithm which would allow it to solve related problems such as the vehicle routing problem with time windows.

Bibliography

[1] L. Bodin, B. Golden, A. Assad, M. Ball "Routing and Scheduling of Vehicles and Crews, The State of the Art", *Computers & Operations Research*, Volume 10, Number 2, pages 69 – 211, 1983.

[2] N. Christofides, A. Mingozzi, P. Toth, "The Vehicle Routing Problem", *Combinatorial Optimization: Annotated Bibliographies*, Chapter 11, pages 315 – 338, Wiley, 1979.

[3] T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[4] M. Desrochers, J. Desrosiers, M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows", *Operations Research*, Volume 40, Number 2, pages 342 – 354, 1992.

[5] M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh, "A classification scheme for vehicle routing and scheduling problems", *European Journal of Operations Research*, Volume 46, Number 3, page 322, 1990.

[6] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[7] J. Hershberger and S. Suri, "Efficient Computation of Euclidean Shortest Paths in the Plane", *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 508 – 517, 1993.

[8] G. Laporte, "The Vehicle Routing Problem: An overview of exact and approximate algorithms", *European Journal of Operations Research*, Volume 59, Number 3, pages 345 – 358, 1992.

[9] G. Laporte, Y. Nobert, "Exact Algorithms for the Vehicle Routing Problem", *Annals of Discrete Mathematics*, Volume 31, pages 147 – 184, 1987.

[10] E. Lawler, J. Lenstra, A. Rinnooy Kan, D. Shmoys, *The Traveling Salesman Problem*, Wiley, 1985.

[11] R. Mole, S. Jameson, "A Sequential Route-Building Algorithm Employing a Generalised Savings Criterion", *Operational Research Quarterly*, Volume 27, page 503, 1976.

[12] Hiroshi Nagamochi, personal communication.

[13] H. Ong, T. Goh, K. Poh, "A computerized vehicle routing system for refuse collection", *Advances in Engineering Software*, Volume 12, Number 2, page 54, 1990.

[14] Jean-Yves Potvin, Jean-Marc Rousseau, "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows", *European Journal of Operations Research*, Volume 66, Number 3, page 331, 1993.

[15] P. Raghavan, *Lecture notes on Randomized Algorithms*, Yale University, 1990.

[16] B.G. Raghevandra, personal communication.

[17] D.M. Ryan, C. Hjorring, F. Glover, "Extensions of the Petal Method for Vehicle Routeing", *or: the Journal of the Operational Research Society*, Volume 44, Number 3, page 289, 1993.

[18] M.W.P. Savelsbergh, "The Vehicle Routing Problem with Time Windows: Minimizing Route Duration", *ORSA Journal on Computing*, Volume 4, Number 2, pages 146 – 154, 1992

[19] M. Solomon, "Algorithms for the vehicle routing and scheduling problem with time window constraints", *Operations Research*, Volume 35 Number 2, pages 254 – 265, 1987.

Index