Closed Charging Cordon Design Problem

Summer project funded by the EPSRC under grand number GR/S86266/01. The co-operation of the UK Department for Transport is gratefully acknowledged

Edward E. Hult
University of Cambridge
Department of Pure Mathematics and Mathematical Statistics
Judge Business School
September 2006

Abstract

The UK Department for Transport (DfT) is currently looking into different methods of designing closed charging cordons around smaller cities. The DfT's current scope of interest is to find an algorithm or method which uses SATURN, a traffic modelling software package, to help find an optimal or near optimal closed charging cordon for any given network. They are currently using a genetic algorithm to design and optimize cordons together with SATURN which evaluates each cordon separately. This method seems to work for small road networks but takes too much computational time for cities with networks of over a few hundred nodes. This paper suggests some possible ways in which to approach the closed charging cordon design problem to get faster computational results while still staying within the DfT's current scope of interest.

Some parts of the paper discuss ideas that can quickly and easily be implemented to help decrease the computational time by a small degree. These ideas are "quick fixes" to help get faster results straight away without investing in extra resources and/or lots of time. One set of ideas are improvements and suggested changes to the DfT's current genetic algorithm to help decrease the number of iterations needed before finding a satisfactory closed charging cordon design. The other set of ideas comprise of changes to the set up of SATURN to help speed up the evaluation process of each cordon.

Other parts of the paper discuss future research areas that are strongly suggested to be pursued. These ideas are more time consuming and need to be researched before they will be able to be implemented but will likely lead to favourable results. The first is to create a new heuristic algorithm that uses more local then random searching compared to a GA. The second is to design and implement a network aggregation algorithm to let SATURN evaluate cordons on a much smaller but well represented network compared to the original network. Beyond these issues is the question, for areas like the West Midlands in which there are several cities in close proximity, of how to model and optimize road charging schemes where a single cordon may or may not be appropriate.

Contents

1. Introduction

The UK Department for Transport (DfT) is currently researching methods on designing congestion charging schemes for smaller cities/towns that are located within the UK. The objective of their research is to find or construct an optimization method that will find a combination of tolled links which will form an optimal or near optimal closed charging cordon. *"A closed charging cordon is a set of tolled links surrounding a designated area so that all travelers entering or passing the area will be tolled. A definition of a closed cordon in the context of graph theory is that all paths from all zones outside the cordon passing through the nodes inside the cordon must be tolled at least once on a link related to those paths"* (Sumalee, 2004b). In this case the objective of the optimal cordon is to maximize the net social welfare function, see Sumalee (2004a) for a detailed formulation. The purpose of this paper is to suggest some different alternatives and ideas, within the DfT's scope of interest, on how to possibly go about to solve the closed charging cordon design problem. The DfT's scope of interest is to use some type of optimization algorithm together with a traffic modelling software called SATURN to find an optimal or near optimal closed charging cordon. The paper will give some concrete suggestions and also talk about ideas that could be pursued in future research projects in trying to find better and faster solution.

Currently the DfT is trying to find a near optimal cordon design using a genetic algorithm (GA) together with the traffic modelling software SATURN. A similar method, using a genetic algorithm together with SATURN, was also suggested by Sumalee (2004a). By entering a certain cordon design and its accompanying charge, for a certain city, SATURN calculates the net social welfare function[1]. This calculation takes anywhere from five minutes up to several hours depending on the size and number of users of the city studied and its road network. Since a

[1] SATURN does not actually calculate the net social welfare function, this is calculated separately using the outputs from SATURN. Throughout the paper it will however be stated that SATURN calculates the net social welfare function since this calculation by itself is very simple compared to the needed outputs that SATURN produces.

GA in general needs to make thousands of goal function calculations before a significant improvement can be noticed, the GA together with SATURN as the net social welfare function calculator is not a sufficient method.

Shrewsbury is one of the smaller cities in the UK that could eventually be considered to have a congestion charging scheme put in place. It is therefore an excellent 'testing ground' to be used for experimentation work in trying different methods for finding near optimal solutions. To run Shrewsbury's road network through SATURN to get the net social welfare function calculated takes approximately seven minutes with the current setup of SATURN. The current method will therefore be used on Shrewsbury's road network to find some results that can later be compared to the results from other new methods.

The paper suggests some ideas for short term "quick fixes" and some ideas for long term real solutions. The quick fixes can quickly and easily be implemented to help decrease the computational time by a small degree without investing in extra resources and/or lots of time. They consist of ideas to improve the DfT's current genetic algorithm to help decrease the number of iterations needed before finding a satisfactory closed charging cordon design. They also consist of changes to the set up of SATURN to help speed up the evaluation process of each cordon.

The long term real solutions are ideas that first need to be researched in order to be implemented. It is strongly suggest that the DfT invest in these or similar research ideas in able to find real and actual solutions to the closed charging cordon design problem for any given network size. A particularly promising idea is to create a new heuristic algorithm that uses more local then random searching compared to a GA. Designing and implementing a network aggregation algorithm to let SATURN evaluate cordons on a much smaller but well represented network compared to the original network. There is also the open question of how to model and optimize road charging schemes in areas like the West Midlands

where a cordon may not be optimal.

The paper is setup as follows: Chapter two defines the closed charging cordon design problem and summarizes a few different suggestions on how to approach the problem. Chapter three explains how the GA used by the Department for Transport currently works and how it is setup. The results from a run using the GA and SATURN with the Shrewsbury network are shortly summarized in chapter four. Chapter five talks about the fist “quick fix” ideas concerning suggested changes to the GA. Chapter six presents the fist long term approach solution regarding the creation and testing of a new heuristic algorithm. Chapter seven and most of eight present different suggestions on ways in which to relax and reduce the data needed for SATURN. Chapter seven, which is the other set of the “quick fix” ideas, talks about different options in SATURN that can be turned on or off to decrease the computation time weighed against the cost of receiving less accurate results. Chapter eight discusses the other long term solution approaches to be pursued by further research. Section 8.1 talks about network aggregation algorithms and how they could be used to more quickly find a near optimal closed charging cordon. Section 8.2 talks about a fairly different topic concerning areas like the West Midlands in which there are several cities in close proximity where the question is how to model and optimize road charging schemes when a single cordon may or may not be appropriate. Finally, in chapter nine, the paper finishes off by recommending future steps to be taken which may be of great interest.

2. Defining the Problem

2.1. Formulation

A given road network can be represented by a graph G = (V, E) where the nodes V represent the intersections and the links E represent the roads in the road network. The number of nodes and links in graph G are given by |V| = n and |E| = m respectively.

The problem is to find a closed charging cordon and its accompanying charge in a road network that will maximize the net social welfare function:
f($w_1, w_2, \ldots, w_n, C$).
Where:
w_i = 0 if node v_i is outside the cordon and
w_i = 1 if node v_i is inside the cordon
for $\forall v_i \in V, \qquad i = 1,2,...,n$
and C is the amount charged for entering the charging cordon. C is a positive real number.

Now consider the graph $G_c = (V_c, E_c)$ where V_c is a subset of V representing all nodes that are placed inside the cordon in the road network and E_c is a subset of E representing all the links that are part of or inside the cordon. Given the road network represented by the graph G = (V, E), the problem can then be stated as to find a graph $G_c = (V_c, E_c)$ that is connected and not 'donut' shaped with a charge C that will maximize the function f($w_1, w_2, \ldots, w_n, C$).

2.2. Solving methods

A city with n nodes, where each node can be considered to be inside or outside of the cordon, has 2^n different possible cordon designs, not including the different

charging choices. Listing all possible cordons and choosing the one with the highest net social welfare function is unrealistic. The closed charging cordon design problem appears, like MPEC (Mathematical Program with Equilibrium Constraints) problems, to belong to the class of NP-Hard problems, meaning that there are no know methods or algorithms to finding a guaranteed optimal solution within polynomial time. The closed charging cordon design problem can be formulated as an MPEC problem but would most likely not be of any help, see Sumalee (2004a) for more details. Therefore an optimization algorithm or some other method can be used at best to find a near optimal solution.

Some possible approaches:

Approximation Algorithms

An approximation algorithm will find a near optimal solution to a hard problem in polynomial time with a certain guarantee. The guarantee could be for example that the solution found is within 5% of the optimal solution. However because of the complexity of the cordon design problem it may be very difficult to construct an algorithm that can give any such guarantees.

Heuristic Algorithms

Heuristic algorithms find solutions to hard problems but without any guarantees on the quality of the solution or its run time. They have however been shown to give good results with many practical problems. The main drawback with heuristic algorithms is that they usually require fast goal function calculations (in this case the net social welfare function) to be able to produce any reasonable results (Carson and Maria, 1997). The net social welfare function takes a very long time to calculate and this will be a major challenge to get around.

Given the time it takes for SATURN to calculate the goal function, the complexity of the problem and that most existing optimization algorithms require hundreds or even thousands of goal function calculations before showing any promise, makes

this problem very difficult to solve within a reasonable amount of time. There are a few different ways to approaching this problem to keep runtimes down.

1: Net Social Welfare Function Manipulation

One approach is to find a function that closely represents the net social welfare function calculated by SATURN, or any other traffic modelling software, but that can be calculated very quickly for any given road network. This would then be used as the goal function in, for example, the genetic algorithm suggested by Sumalee (2004a) for finding a near optimal closed charging cordon designs. It would then be possible to generate the necessary thousands of generations in the GA in a reasonable amount of time. The challenge with this approach is in constructing a function that can quickly be calculated and that realistically represents reality, almost as well as SATURN does, with what happens when introducing different cordon designs in a road network. If this new representation differs to greatly from reality then the solution found by the GA will most likely be poor. In this approach a fine balance must be found between calculation time and accuracy. This approach will not be discussed further in this paper but would be a good option for further research.

2: Genetic Algorithm Customization

A second approach is based on customizing a Genetic Algorithm so that a lot fewer iterations would be needed to find the same quality of solution that a 'regular' GA would find. Making modifications in how the initial population is produced and how the GA performs reproduction in later iterations will aid in this. The approach will be discussed in further detail in chapter 5.

3: A New Heuristic Algorithm

A third approach is to designing a new and very specific heuristic algorithm that would require relatively few calls, perhaps only in the hundreds, to SATURN (or any other traffic modelling software) to find a near optimal solution to the closed charging cordon design problem. The idea with this approach is to construct a

method that conducts a set of calculations so it can predict a good cordon design using information learnt from previous evaluations. Every new predicted cordon design would be sent into SATURN to be analysed. The method would then update itself by adding the new evaluation to its “knowledge”. The idea is that after a few hundred iterations a suitable and near optimal closed cordon design would emerge. Ideas and thoughts about how this could possibly be done are further examined in chapter 6.

4: SATURN Options

SATURN has a lot of functions that can be switched on and off to get more or less accurate results in the net social welfare calculation. One approach is to turn off some of the default settings to see how the runtime and results differ. This would need to be conducted on a small network so adequate testing can be done. See chapter 7 for further details on how this can be performed.

5: Network Aggregation

A final idea that will be discussed is network aggregation. The time it takes for SATURN to calculate the net social welfare function for a network increases four times when doubling the number of nodes in the network while using the same parameter setup (number of user classes, simulation on or off, ect.). The idea with this approach is to aggregate and disaggregate different parts of the network to get quicker SATURN calculations.

None of the above mentioned approaches will likely lead to an optimal solution. However all of them, if solved and performed correctly, have the potential of giving good or even near optimal solutions to the closed charging cordon design problem in less time then the method currently used by the Department for Transport.

3. Genetic Algorithm currently used by the DfT

3.1. Introduction

A genetic algorithm is a search and optimization technique used to identify good solutions to hard optimisation problems. It is an optimization method that has been inspired by the genetic evolution process. A GA is a structured random search that combines promising candidates to produce new candidates for evaluation. The technique has been proven efficient in large solution spaces where the combination process is effective at moving through the solution space. This chapter describes the way that the UK Department for Transport has currently setup their GA to find a closed charging cordon design.

The GA needs a genetic representation of the solution and a fitness function which in this case is the closed cordon and the net social welfare function calculated by SATURN. The genetic representation of the cordon is an array of 191 zeros and ones which represent each of the nodes in the Shrewsbury network. Each 'zero' represents a node outside the cordon and each 'one' represents a node inside the cordon. The different fitness functions for the different cordons are then directly compared to each other and the higher the value the stronger the solution. The fitness function can also take on a negative value if the net social welfare is less with a specific cordon then without any cordon at all.

3.2. Initialising the Population

The GA needs an initial population of cordons which is randomly generated. To generate these random cordons the study area of nodes has been mapped and divided into a Delaunay triangulation. A Delaunay triangulation is a set of triangles connecting a set of points in the plane where the circumcircle of each

triangle does not contain any of the points. The reason for mapping the road network into a Delaunay triangulation is because it is an easy way to randomly select closed cordons.

The GA randomly selects one of these triangles as a starting point in the study area. It will then undertake a random walk to surrounding triangles, with a 60% chance of walking out from each side. Each of the nodes on each of the triangles visited will become included in the cordon. If an outer edge triangle is walked into, the two outer edge nodes will not be included in the cordon. A check is undertaken to ensure the cordon is a solid area without holes and is valid (i.e. there are no nodes within the cordon boundary which can not be reached by all other nodes using the road network within the cordon). A check is also taken to make sure that the cordon produced contains at least five nodes. These checks are also done on the children, see below.

The size of the population has been set to 200 cordons.

3.3. Selection

When the initial population has been created and the fitness functions calculated by SATURN the GA can begin to generate new solutions. Two parents are chosen at random from the population. The probability of each cordon being selected as a parent can vary and there are many methods for assigning these probabilities. The one used by the DfT is weighted according to rank.

Weighted according to rank

If n is the total number of cordons and C_i is cordon i for $i = 1,2,\ldots,n$ then $F(C_i)$ is the fitness function of cordon i. $P(F(C_i))$ is the probability of choosing cordon i as a parent where:

$$P(F(C_i)) > P(F(C_m)) \quad and \quad F(C_i) > F(C_m) \quad for \quad m = i+1, i+2, ..., n$$

and

$$P(F(C_1)) = n / \sum_i i, \quad P(F(C_2)) = (n-1) / \sum_i i, \quad ..., \quad P(F(C_n)) = 1 / \sum_i i$$

The two selected parents are then combined to generate a child, a new cordon, by a set of defined reproduction rules.

3.4. Reproduction

There are two main methods that the GA goes about generating new cordons from the two parent cordons. They are intersection and union.

Intersection:
With intersection a new cordon is made by using only the nodes which are in both of the two parent cordons as the new cordon. If the two parents do not have any similar nodes inside their cordons then no new child is made.

Union:
With union a new cordon is made by using all the nodes in both the parent cordons to create the new cordon.

Intersection and union are chosen randomly with a 50% chance each of being picked. Before scoring the child there is an X% chance that the child undergoes mutation. Mutation is applied to either increase or decrease the boundary of the cordon by randomly taking away or adding on extra nodes. The reason for mutation is to make sure that the GA does not get stuck in a local optimum. After this the GA checks that the resulting child is a valid cordon and that it is not exactly the same as an earlier cordon. If the child is not feasible or is exactly the same as an old cordon then the GA picks two new parents and tries to make a child from them. The GA keeps trying to make a new child until it has failed $n^2/2$

times in a row, where n is number of the population, before it gives up and aborts the program. Once found, the child cordon is scored using Saturn. If the new child cordon has a higher fitness function than the weakest cordon in the population it will replace the weak cordon. Otherwise the new cordon is discarded from the population.

Mutation rates can be set at run time, but should remain low, approximately 1%. If a cordon is mutated there are further mutation rates that are applied to determine exactly how the cordon is changed. For each pair of nodes, where one is inside the cordon and the other is outside, the GA will decide whether to increase, decrease or not change the cordon area. Note that each node can be in more than one pair. These rates are initially set to 15%, 15% and 70% respectively for each pair. To increase the cordon area the external node is included within the cordon, whereas to decrease the cordon area the internal node is removed from the cordon.

3.5. Termination

The process of generating children is repeated until a predefined stopping condition is met. The condition can be any of the following: no more time to generate solutions, the diversity of the population is so low that the GA can not generate any new cordons within the allowed maximum number of tries, a pre-defined number of generations have been generated.

After the program has terminated then all that is left is to sort the remaining population by their fitness functions and plot the top cordons to see their designs. Hopefully these designs will be similar and have similar fitness function values.

3.6. Cordon Charge

Two different methods are currently used for choosing the charge for entering the

cordon. The first method is simply predefining the charge before the GA is run and finding a near optimal cordon for that charge. The second method lets the GA choose the charge by randomly assigning one of the following charges to the initial population: .50, 1.00, 1.50, 2.00, 2.50, 3.00, 3.50, 4.00, 4.50 and 5.00 pounds. During reproduction the child is run through SATRUN twice if the two parents have different charges, once with each of their charges. This gives two new children with the same cordon structure but different charges.

4. One Set of Results from the Current GA

Originally there were a lot of numerical results from different runs of the current setup that the DfT uses. However these results were later found to be unreliable since a programming error was found in the setup. A recommendation is for DfT to go back and fill in this chapter after more new runs have been made and analysed. For now there is only one run that has been completed.

At the DfT several computers are available to make parallel calculations of different cordons simultaneously in SATURN. Using multiple computers and the Shrewsbury network a run was made pre set to stop after 2000 cordons had been calculated. Using the GA and SATURN to solve for the closed charging cordon design problem took approximately two and half days. The GA was set to randomly place a charge, see chapter 3.6., for each initial cordon in the population. After the run was completed the results were analyzed.

The best few cordons designed by the GA were all very similar. They were all placed in and around the same area with only a few nodes differing from being inside or outside the cordon. The best cordon resulted in a net social welfare score of approximately 841 pounds and was the 1724^{th} cordon designed by the GA. The charge was set to 2.50 pounds. The 10^{th} best cordon got a score of approximately 640 pounds giving a difference of 201 pounds between the top ten

scores. This difference is quite significant which means that perhaps the GA should have been let to run a bit longer. In total 235 cordons got a positive net social welfare score but with many of the scores being very small, the rest were negative.

After the run was completed the top cordon was tested with some different charges to see what would happen. Using higher charges resulted in lower scores. When the cordon was tested with lower charges the score improved for the first couple of steps before it started getting worse again. This suggests that either the GA should have been left to make a few more runs as commented on above or that the GA is not finding very good charges for its cordons which in effect may be damaging the entire search. Looking at the way the GA assigns charges it is most likely that the GA just needs to be let to run longer, however a few suggestions on possible improvement to the charging method are given in the next chapter.

5. Genetic Algorithm modifications

Solving the closed cordon design problem with shorter and/or fewer calculations becomes very important with larger road networks. Using the setup discussed in chapter three it would take over a year before any reasonable solutions would be found with larger road networks. In this chapter some modifications to the current GA used by the DfT are suggested to try to minimize the number of iterations needed by the GA to find a good solution.

According to the No Free Lunch Theorem all search algorithms over all problems will perform on average the same (Wolpert and Macready, 1997). With this theorem in mind, simply using another search algorithm in place of the genetic algorithm will most likely not lead to any significant improvements. Instead to get any significant improvements it is important to customize an algorithm as much

as possible to fit the problem at hand.

5.1. Changes to the Genetic Algorithm

One criterion when using a Genetic Algorithm is that the fitness function can quickly be calculated for each child. This is necessary because it usually takes several generations before any significant improvements in the fitness function can be found (Carson and Maria, 1997). In the case of optimal cordon design it takes a very long time to calculate the fitness functions for each cordon, up to several hours. It is therefore important that the GA makes as few fitness function calculations as possible which mean that changes need to be made in the GA to make this happen.

Sumalee (2004a) also suggests the use of a genetic algorithm together with a traffic modelling program to find the optimal cordon design. The genetic algorithm he suggests uses a Branch-Tree Framework with a small fixed starting cordon that will be included in every produced cordon. During the reproduction process his GA uses crossover and mutation. With crossover the process starts by finding the identical nodes in each parent. Then they randomly exchange a non identical set of nodes that get added to the identical set creating two new children. Note that this is only a rough description of how the crossover process works with the Branch-Tree Framework, see Sumalee (2004a) for a thorough and in-depth presentation of this process. One of the advantages with his method is that each new cordon that is produced maintains the closed cordon design and therefore no checks for feasibility need to be made.

5.1.1. Initial Population

In the GA that Sumalee (2004a) suggests there is a small chosen starting cordon that all the cordons made by the GA build from. This is a risk since a poorly chosen starting cordon will most likely not lead to any optimal or near optimal final solutions. On the other hand in the GA currently used by the DfT a lot of

computing time is being wasted on undesirable cordons along the outskirts of the city. Therefore a combination of the two methods may be more desirable.

Idea 1: Partly fixed initial population

Manually choose 5 nodes with the rule in mind that every initial cordon made by the GA has to randomly pick, with a 20% chance, one of these nodes to start its random walk from. By adding this rule it is most likely that the following benefits will be found.

- There will be fewer undesirable cordons to be evaluated since all initial cordons will be within or partly within the critical area.
- It will be easier for the GA to generate new cordons using intersection and union since there will be more cordons that have identical or neighbouring nodes.
- Smaller initial populations can be used because many undesirable cordons are no longer initially produced.
- Poorly chosen initial nodes can be overcome with reproduction and/or mutation since only the initial population is required to have them within their cordons.

5.1.2. Choice of Parents

Parents are chosen for reproduction with a certain probability depending on their rank. What makes one parent better then the other? In the current GA it is solely based on the fitness function value from Saturn but is this really accurate?

Possible things to consider:

- What percent of the worst congested areas are within the parent cordon?
- What links that lead to the worst congested areas are within the parent cordon?

If these can be identified then perhaps the parents containing more of the two points above should get an additional few percent added to their probability of being chosen as a parent cordon.

5.1.3. Reproduction

After the parents have been chosen a reproduction method is randomly selected with a 50% chance of being either intersection or union. Both intersection and union will produce a very different child cordon if the two parents have very few identical nodes inside their cordons. This can both be good and bad. It is more likely, but not at all definitely, bad if one of the parents is very strong. This may be bad because the strong cordon may loose all of its beneficial nodes.

Idea 2: Priority Reproduction

Lower the chance for the GA to choose intersection or union as the reproduction methods to 35% each. Add a third reproduction method with a 30% chance of being chosen. The new reproduction method should let the child inherit more of the stronger parents' traits. The new cordon should closely represent the stronger parent with only a few minor changes inherited from the weaker parent. Start off by using intersection but then make a random walk, as described in chapter three, from the intersected edge out adding each step taken into the child cordon but staying within the stronger parents cordon boundary. The probability to take each step should be P_s. Then do the same random walk from the edge of the intersection but only allowing steps to be taken within the weaker parents cordon boundary with a probability of P_w. P_s should equal the rank of the stronger parent divided by the sum of the rank of the stronger and weaker parent.

$P_s = Rank_{StrongParent} / (Rank_{StongParent} + Rank_{WeakParent})$

$P_w = 1 - P_s$

Idea 3: Strong Mutation

A fourth reproduction method that is very simple but can be very beneficial is Strong Mutation. Strong Mutation is performed simply by selecting the stronger of the two parents and conducting a random walk in any direction starting from a random point along the edge of the cordon. If the walk goes into the cordon then each node reached should be taken out of the cordon. If the walk goes out of the

cordon then every node visited should be added to the cordon. The random walk should be performed in the same manner as described in chapter 3 but should have a 50% chance of taking a step in any direction. Nothing is done with the weaker parent. This method should have a 10% chance of being chosen and so Union and Intersection need to be lowered to a 30% chance each of being chosen as the reproduction method and keeping the Priority Reproduction method at 30%.

5.1.4. Offspring

Sending every feasible offspring through SATURN is not advisable since each calculation takes a long time. Instead every new child should quickly be analysed to see if the new cordon has even the smallest possibility of being or leading to a desirable cordon. If so then it should be kept otherwise it should be discarded.

Idea 4: Pre-checking the offspring

Define what nodes in the network are within the critical area, meaning: what nodes are in the more congested parts of the road network or have critical links leading into these parts. If the offspring produced has none of these nodes within its cordon then the child can be discarded without being analysed further.

5.1.5. Charge

Currently the charge is randomly chosen for each cordon in the initial population and then the parents transfer over their charges to the child resulting in two new children, with the same cordon shape, if the charges differ. There are a few things that could be added to the charge method and also a worst case scenario that needs to be solved. The worst case scenario with this method could lead to the cancellation of one or more important charges leading to a local optimum trap.

Idea 5: Charge Mutation

After the children have gotten their charges from their parents, there should be a

30% chance that the charge inherited from the weaker parent should randomly be changed with an equal probability of receiving any of the other charges not possessed by either of the two parents.

Idea 6: Initial Charge Rank

Give each available charge an initial rank that seems suitable for the given network. That is, what charges will most likely turn out to be good according to a judgemental approach?

Ex) If charge 2.00, 2.50, 3.00, 3.50 and 4.00 pounds seem like they might realistically be better then give them a higher probability to be chosen then the others during the charge assignment for the initial population.

Idea 7: Charge change for area difference

If the area within the cordon changes dramatically, as will happen with union and intersection if the parents differ greatly, decrease or increase the charge in part of the percent increase or decrease of the child cordon respectively. This is because a larger cordon will 'consume' more travellers and will therefore most probably not need an as high charge.

5.2. Manual Design Together with the GA

A lot of computational time is spent evaluating the initial population in the GA. The initial population is more or less randomly produced by the GA and the purpose with the GA is to keep upgrading the population so only stronger and stronger individuals survive. Starting with a random population makes this process very time consuming. If the GA instead could skip the first X% of fitness function evaluations and generations then the time for the process of finding a near optimal solution would greatly decrease. This could be done by having a well trained and experienced person manually design the initial population for the GA.

5.2.1. Implementation

Thoroughly design ten to fifteen different but well placed closed cordons for the given road network. Choose three different tolls that seem most plausible to be used in the final cordon. Assign one of the tolls to each of the cordons designed but make sure that each of the different toll charges are used by at least two of the cordons. Use these cordons with their associated charges as the initial population in the GA. Run the GA using the reproduction rules discussed in chapter 3 and 5.1. Let the charge associated with the stronger parent go to the child with a 60% chance and with the other two charges available the child should get one of these with a 20% chance for each. Run the GA for 150 fitness function calculations or more for the smaller networks if desired.

6. Specific Heuristic Algorithm

A GA can be seen as a guided randomization search for finding optimal or near optimal solutions. If this randomization could instead be turned into a more specific local optimum search a lot of time could be saved. One idea in trying to solve the cordon design problem is to construct a very specific heuristic algorithm that can make good predictions how a cordon should be setup for any given network. The idea is that for every produced cordon on average to be sent through SATURN should be a better prediction of how the cordon should look than what a GA produces by itself during reproduction. A correctly designed algorithm will require a greatly decreased number of calls, compared to a GA, to SATURN, or any other traffic modelling software, and a lot of time can be saved. This is assuming that the extra work done by the algorithm takes in total less time then the time saved from the decreased number of calls to SATURN. The goal is to construct the algorithm is such a way that only approximately 150 SATURN calculations need to be performed. Below presents a ‘crazy’ idea of how something like this could possibly be setup. It also serves as a template for how

other heuristic algorithms can be set up and tested. More research is advised and needed in this area for it to be able to mature.

6.1. Initialization

Start off by producing around 15 random cordons using the process described in section 5.1.1. assigning a few different charges to the different cordons. Run all the initial cordons through SATURN to be evaluated. The purpose of the first runs is to gather information that can be used to make the first prediction of how the cordon should look.

For simplicity's sake an example will be used from here on out to help describe the methodology of this idea.

Take the following small network with 9 nodes and 13 links.

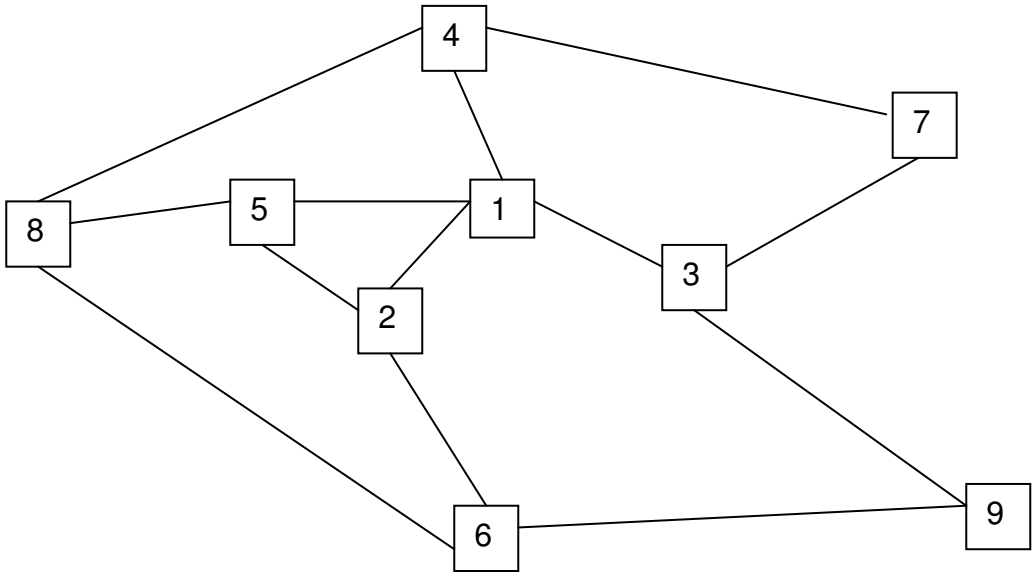

Figure 1: Example Road Network

Assume that the five following cordons represent the total initial set of randomly produced cordons. (In the example only five cordons are used in the initial set

because of the extremely small size of the network)

1. Node 1 is in the cordon and a charge of 2 pounds is set on links 5-1, 2-1, 4-1 and 3-1 (where 5-1 represents the link going from node 5 to node 1).
2. Nodes 1, 4 and 5 are in the cordon and a charge of 1 pound is set on links 8-4, 7-4, 8-5, 2-5, 2-1 and 3-1.
3. Nodes 1, 2 and 5 are in the cordon and a charge of 3 pounds is set on links 8-5, 4-1, 3-1 and 6-2.
4. Nodes 1, 2, 4, 5 and 8 are in the cordon and a charge of 1 pound is set on links 6-8, 6-2, 7-4 and 3-1.
5. Nodes 1, 2, 3 and 5 are in the cordon and a charge of 2 pounds is set on links 8-5, 4-1, 6-2, 7-3 and 8-3.

Assume that the following matrix represents the traffic behaviour, in a specific unit (traffic flow, travel time per vehicle, ect.), on each link for the network in figure 1 before any cordon has been placed. The rows represent the origin nodes and columns represent the destination nodes.

	1	2	3	4	5	6	7	8	9
1	-	20	6	8	15	-	-	-	-
2	30	-	-	-	25	5	-	-	-
3	15	-	-	-	-	-	5	-	2
4	20	-	-	-	-	-	3	4	-
5	18	15	-	-	-	-	-	5	-
6	-	25	-	-	-	-	-	8	7
7	-	-	15	4	-	-	-	-	-
8	-	-	-	5	20	3	-	-	-
9	-	-	20	-	-	10	-	-	-

Table 1: Traffic behaviour for each link before implementation of a cordon

Now assume that the following matrix represent the traffic behaviour calculated

by SATURN, or any other traffic modelling software, with cordon 1 in place. The bold numbers represent the charged links and the red and blue numbers show where there have been increases and decreases in the link usage respectively.

	1	2	3	4	5	6	7	8	9
1	-	20	6	8	15	-	-	-	-
2	**25**	-	-	-	25	5	-	-	-
3	**14**	-	-	-	-	-	5	-	2
4	**10**	-	-	-	-	-	6	9	-
5	**16**	15	-	-	-	-	-	5	-
6	-	28	-	-	-	-	-	8	7
7	-	-	18	4	-	-	-	-	-
8	-	-	-	5	23	5	-	-	-
9	-	-	20	-	-	10	-	-	-

Table 2: Traffic behaviour for each link with cordon 1 in place

As can be seen, only the tolled links, in this example, have had a decrease in traffic behaviour and an increase on some of the other links has occurred.

6.2. Information Gathering

When all the initial cordons have been sent through SATURN some information, other then the net social welfare function result, must be gathered so it can later be used.

6.2.1. A Vectors

Define the column vector $A1(x)_k$ that represents the difference in the traffic behaviour on each link *k* in the network between having cordon 1 in place and having no cordon in place. In this example $A1(x)_k$ will be written as $\mathbf{A}1(1)_k$. The A in $\mathbf{A}1(1)_k$ represents that it is an A vector which is a family of vectors that stand

for the difference in traffic behaviour between a certain cordon and no cordon. The 1, A**1**$(1)_k$, represents when the cordon was analyzed in relation to the rest of the A vectors (1 means it was the first cordon to be analyzed, 23 means it was the 23rd cordon to be analyzed). The numbers in the parentheses, A1($\mathbf{1}$)$_k$, represents what nodes are inside the cordon.

Example)

Link 1-2:	20-20=0 gives:	$A1(1)_1 = 0$
Link 1-3:	6-6=0 gives:	$A1(1)_2 = 0$
Link 1-4:	8-8=0 gives:	$A1(1)_3 = 0$
Link 1-5:	15-15=0 gives:	$A1(1)_4 = 0$
Link 2-1:	25-30=-5 gives:	$A1(1)_5 = -5$
…	…	…
Link 9-6:	10-10=0 gives:	$A1(1)_{26} = 0$,

$$A1(1)_k = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -5 \\ \vdots \\ 0 \end{pmatrix}$$

Produce the A vectors A2(1,4,5)$_k$, A3(1,2,5)$_k$, A4(1,2,4,5,8)$_k$, and A5(1,2,3,5)$_k$ that represent the other initial cordons and calculate their values.

The purpose of the A vectors is to use them as a start in trying to find some sort of cause and effect in the traffic behaviour when different nodes are placed inside or outside the cordon.

6.2.2. B Matrix

The next step is to produce a B matrix $B_{k,i}$ for all links *k* in the network and for all nodes *i* that have been inside a cordon. $B_{k,i}$ must be extendable to allow

additional columns to be added that represent other nodes that might get placed inside the cordon. Use the A vectors to calculate the values for in $B_{k,i}$. For all numbers that are ether all greater then zero or are all less then zero, that represent the same link and node in each of the A vectors, calculate the average and enter that number in the appropriate section in the $B_{k,i}$ matrix. If not all numbers are of the same sign or some are zero then the corresponding number in the $B_{k,i}$ matrix must be zero. The exception is if 95% or more of the numbers are of similar type (all greater then zero or all less then zero) then an average should also be calculated using all the values representing that link and node.

Example)

Assume that the following data is collected from the five cordons above and that each of the rows with actual values represents the same rows in each vector (for descriptions sake assume these rows represent links X, Y and Z).

$$A1(1)_k = \begin{pmatrix} \vdots \\ -5 \\ \vdots \\ 3 \\ \vdots \\ 0 \\ \vdots \end{pmatrix}, \qquad A2(1,4,5)_k = \begin{pmatrix} \vdots \\ -8 \\ \vdots \\ -3 \\ \vdots \\ -1 \\ \vdots \end{pmatrix}; \qquad A3(1,2,5)_k = \begin{pmatrix} \vdots \\ -6 \\ \vdots \\ 3 \\ \vdots \\ 0 \\ \vdots \end{pmatrix}$$

$$A4(1,2,4,5,8)_k = \begin{pmatrix} \vdots \\ -7 \\ \vdots \\ -1 \\ \vdots \\ 0 \\ \vdots \end{pmatrix}, \qquad A5(1,2,3,5)_k = \begin{pmatrix} \vdots \\ -5 \\ \vdots \\ 4 \\ \vdots \\ -3 \\ \vdots \end{pmatrix}$$

When filling in $B_{k,i}$ it is important not to get the columns mixed up. $B_{k,i}$ has 6 columns representing nodes 1, 2, 3, 4, 5 and 8 since these are the only nodes that have so far been inside an analyzed cordon.

$$B_{k,1} = \frac{\left(A1(1)_k + A2(1,4,5)_k + A3(1,2,5)_k + A4(1,2,4,5,8)_k + A5(1,2,3,5)_k\right)}{5}$$

$$B_{k,2} = \frac{\left(A3(1,2,5)_k + A4(1,2,4,5,8)_k + A5(1,2,3,5)_k\right)}{3}$$

$$B_{k,3} = \frac{\left(A5(1,2,3,5)_k\right)}{1}$$

$$B_{k,4} = \frac{\left(A2(1,4,5)_k + A4(1,2,4,5,8)_k\right)}{2}$$

$$B_{k,5} = \frac{\left(A2(1,4,5)_k + A3(1,2,5)_k + A4(1,2,4,5,8)_k + A5(1,2,3,5)_k\right)}{4}$$

$$B_{k,6} = \frac{\left(A4(1,2,4,5,8)_k\right)}{1}$$

Note that the above formulas are only correct for each link *k* if the values in at least 95% of the A vectors are all greater then zero or are all less then zero. If this is not the case then the value entered in the B matrix must be zero for that link and node. Using the values in the example above the B matrix becomes:

$$B_{k,i} = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -6.2 & -6 & -5 & -7.5 & -6.5 & -7 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 4 & -2 & 0 & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & -3 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

The idea behind the B matrix is to show how any one node that has been inside a cordon will affect the traffic behaviour on some link or links in a certain way until proven otherwise.

In the example it can be seen that by placing any of the nodes 1, 2, 3, 4, 5 or 8 in the cordon a decrease in the traffic behaviour will occur on link X. This is not however true on links Y and Z. Placing node 3 in the cordon will lead to an increase of traffic behaviour on link Y by 4 units. This is obviously not an accurate representation of what happens but the thought is that it will give a starting point in helping to predict what the next cordon should look like and it will also become

increasingly more accurate for each iteration to come.

6.3. New Cordon Prediction

When the B matrix has been calculated it is time to predict the next cordon design with its help. Assume that *C* represents the subset of nodes which have been in at least one analyzed cordon. Then define X_i to be a binary integer that equals one if node *i* is inside the cordon and zero otherwise. Now setup the following:

$$\min \sum_{\forall i \in C} \sum_{\forall k} \left(B_{k,i} \cdot X_i \right)$$
$$s.t. \quad X_i = \{0,1\} \qquad \forall i \in C$$
$$X_i = 0 \qquad \forall i \notin C$$

This could be solved by linear programming if it were not for the risk of losing the closed cordon design. Therefore it is easier to solve this approximately by using a genetic algorithm like the one described in chapter 3 and 5. The main difference from the GA in chapter 3 and 5 is that the above goal function would be used to calculate the fitness function instead of SATURN. It can also be desirable to allow one or two of the nodes not in *C* to be able to be placed inside the cordon if it is necessary to keep the closed cordon design. This can also lead to further benefits in that more nodes will have values representing them in the B matrix.

When the new cordon is found by the GA it needs to be checked to make sure that an exact same cordon has not already been produced and analyzed, if so then the next best cordon suggested by the GA should be checked for uniqueness and then the next until a unique cordon is found. When a unique cordon is found it should be sent through SATRUN to be analyzed. The results should be gathered and used to calculate a new A vector that must be produced, in this case $A6(x)_k$.

6.4. Avoid Local Traps

Getting stuck in a local optimum is very likely if not more different nodes can be brought into a cordon than those from the initial population. The one or two nodes that may be brought into the cordon, as described in section 6.3., that have not earlier been inside is not a sufficient enough method. To keep from getting stuck in a local optimum it is important to produce a completely new and random cordon every time one cordon has been produced with the help of the B matrix. This can be done by using the method described in chapter 3.2. The new random cordon also needs to be checked for uniqueness and should then be sent through SATURN if unique. If it is not unique a new random cordon should be made. The results should be used to calculate the values for a new corresponding A vector. Here it would be $A7(x)_k$.

6.5. Updating and New Predictions

The two new A vectors produced from 6.3. and 6.4. must now be taken into consideration for the next cordon prediction and so the B matrix must be updated. Recalculate all the values in the B matrix, as described in section 6.2. but using all of the 7 A vectors. Also add columns to the B matrix where needed to make sure all the nodes that have been inside a cordon are represented.

Example:
Say that the best result from the GA suggests the following unique cordon to be analyzed by SATURN is:

> Nodes 2 and 5 are inside the cordon with a 2 pound charge set on links 8-5, 1-5, 1-2 and 6-2.

Also say that the random cordon made is:

> Nodes 6 and 8 are inside the cordon with a 3 pound charge set on links 4-8, 5-8, 2-6 and 9-6.

Assume that the two new A vectors get the following data on links X, Y and Z as

used above.

$$A6(2,5)_k = \begin{pmatrix} \vdots \\ -4 \\ \vdots \\ 2 \\ \vdots \\ 0 \\ \vdots \end{pmatrix}, \qquad A7(6,8)_k = \begin{pmatrix} \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ -8 \\ \vdots \end{pmatrix}$$

Now recalculate the B matrix with the new data, find a new cordon with the GA as discussed in section 6.3 and make a new random cordon. Keep repeating these steps for approximately 60-80 times. By the end of all the iterations hopefully a descent cordon design has been found.

6.6. Needed Improvements

This process is far from complete and does not talk about how to update the charges. This would need to be added into the B matrix some how so it can be considered when predicting the new cordon design. The B matrix itself needs to be revised to better represent different nodes and also a faster method of updating the B matrix itself would be beneficial.

To get a better representation of how each node affects different links it might be a good idea to look at specific combinations of nodes. Very specific rules must be setup for this since all combinations of nodes cannot be looked at in polynomial time. Perhaps certain nodes can be aggregated, or some play off each other more than others and how can they be found? There is still much that can be done and tested with this method to see if proper predictions can be made with only one or two hundred calls to SATURN being made.

6.7. Testing Procedure

To test and implement this idea some basic steps can be followed. These steps can also serve as template for developing and testing other heuristic algorithms.

Before the steps are started make sure to define the main goal with the test and also some sub goals. The sub goals should be steps to be reached on the way to completing the main goal. For example if the main goal was to run a marathon some sub goals could be to run a 10 km race and a half marathon race prior the marathon and to run at least 15 miles a week. Some sub goals should be reached in sequence and some will run throughout the testing and implementation. In the example above the 10 km race will be run before the half marathon but the 15 miles minimum a week goal will be throughout. It is a good idea to write these sub goals differently to help keep the testing and implementation clear and on track. For example:
Sub goal 1:
Run 10 km race.
Sub goal 2:
Run half marathon race.
Sub goal A:
Run a minimum of 15 miles a week.

Writing them up this way helps keep it clear that sub goal 1 should be completed before pursuing sub goal 2 and that sub goal A (also sub goal B, C, … if there were any) should be under consideration at all times.
The main goal is usually quite obvious but can be worthwhile to define anyway. The sub goals are sometimes not as clear and are therefore more important to state and keep clear at hand.

Main Goal:
Construct a heuristic algorithm that can solve the closed charging cordon design

problem by requiring no more then N SATURN calls where N << M and M is the number of SATURN calls required by the GA. It is also important to make sure that the computation time spent outside of SATURN is insignificant in comparison to that of SATURN.

Sub goal 1:

Construct a heuristic algorithm that can solve the main goal but with a set charge and for a very small specific network.

Sub goal 2:

Test and modify the algorithm so it works the same as in sub goal 1 but with any random and slightly bigger (but still small) networks.

Sub goal 3:

Modify the algorithm so it can solve the main goal (without a set charge) for the same network used in sub goal 1.

Sub goal 4:

Test and modify the algorithm so it works the same as in sub goal 3 but using the same network types as in sub goal 2.

Sub goal A:

Get an idea of how much the quality of the solution is affected by varying N.

Sub goal 5:

Test and modify the algorithm so it works for big network and figure out, if it is possible, how big N needs to be in comparison to the size of the network.

When the main goal and sub goals are defined the following steps can be taken in testing and trying to construct the heuristic algorithm.

Step 1: Small network, GA with set charge

Construct a very small connected road network (approximately 15-25 nodes). Find a subset of global optimization results for a few different pre set charges. Get these by running the GA (described in chapter three with some of the modifications suggested in chapter five) together with SATURN and setting M (the number of SATURN calls used by the GA) to a few thousand. This

computation should not take very long because of the small size of the network.

Step 2: Small network, new algorithm with set charge.

Construct an algorithm as described in 6.1.-6.5. but use only a pre set charge. Run several tests using the different set charges that where used in step 1 and check what happens to the result when varying N (the number of SATURN calls made by the algorithm) for each charge. If the algorithm finds as good or better results then the GA in step 1 using N <= 200 go to step 3 (sub goal 1 completed). If the results are good but only with N > 200 go to step 9. If the results are worse then what the GA is producing then go to step 8.

Step 3:

The algorithm is so far finding good results using a fixed charge for a small network.

Step 3.1: Vary N

If not already done, see how low N can be and still get good results and also see how much better the results are, if any, with a much higher N. This can later be of good use to compare with other networks of similar size and with other networks of much greater size.

Step 3.2: Midsize network, GA with set charge

Randomly construct two or three slightly bigger connected road network (approximately 75-100 nodes). As in step 1 find a subset of global optimization results for a few different pre set charges for each network. Get these by running the GA together with SATURN and setting M = 5000-10000.

Step 4: Midsize network, new algorithm with set charge

Now test the new algorithm on each of the new networks. As in step 2 run several tests using the different set charges that where used in step 3 and check what happens to the result when varying N for each charge. If the algorithm finds as good or better results then the GA using N <= 200 go to step 5 (sub goal 2 completed). If the results are good but only with N > 200 go to step 9. If the

results are worse then what the GA is producing then go to step 8. If the results are as good or better the then the results from the GA using N <= 200 but there is a big improvement when N > 300 or so, then read ‘note 2’ at the end of this chapter before continuing.

Step 5: Small network, new algorithm with variable charge

Modify and test the new algorithm, using the small network constructed in step 1, so it can find a cordon with an accompanying charge without having a pre set charge given. This result should be as good or better then the best result found using different pre set charges by the GA in step 1. Again test this with different values of N. If the algorithm finds as good or better results with N <= 200 go to step 6 (sub goal 3 completed). If the results are good but only with N > 200 go to step 9. If the results are unsatisfactory go to step 8.

Step 6: Midsize network, new algorithm with variable charge

Test the modified algorithm on each of the slightly bigger networks constructed in step 3.2 without using a pre set charge and different values for N. Compare the results with the best results found in step 3.2. If the results are satisfactory for N <= 200 go to step 7 (sub goal 4 completed). If the results are good but only with N > 200 go to step 9. If they are worse go to step 8. Also if the results are good for N <= 200 but there is a big improvement when N > 300 or so, then read ‘note 2’ below before continuing.

Step 7: Test on smaller real road network with variable charge

The new algorithm is doing quite well and it is time to test it on a real road network. Find a near optimal closed charging cordon for, for example, the Shrewsbury network using the GA and setting M >= 8000 to make sure that a good result is found. Run the new algorithm with the same network using different values of N. Compare the results. If the results are highly unsatisfactory and major modifications and changes need to made, redo the entire testing process again from the start. If the results are good go to either step 9 or step 10

as appropriate.

Step 8: Non desirable results

The new algorithm is not producing an as good cordon as the GA. The algorithm needs to be modified since it is not producing any desirable results. Start off by looking into the couple of ideas given in the second half of section 6.6. and/or try to come up with other alternative modifications and/or additions to the algorithm. For each new idea found, make that change and start the testing process over at the appropriate stage. Add only one new change for each testing process unless that turns out to be unpractical.

Step 9: N is too high

The algorithm is producing good results but making to many SATURN calls. Try to figure out how to modify the algorithm so it “evolves” quicker. For each modification restart the testing procedure at the appropriate stage to see how much faster, if any, it has gotten.

Step 10: Test on large real road network with variable charge

The algorithm has so far passed all the tests and seems to working. The last and final test is to try it on a large network. At this time it would be good if there is an idea of how big N needs to be to give satisfactory result (sub goal A should now be completed but note that it has been under consideration throughout). Run the algorithm on a large real road network with the smallest N value that seems feasible to give good results. Make also a second run with a 20-50% increase of the N value. Compare the results with each other and also with the results found by using a judgmental approach. If the results are unsatisfactory try to figure out why the algorithm works for small networks but not for large ones. Does N need to be increased more? Is there some function in the algorithm that can’t handle large amounts of data? Make the necessary changes and start the testing process over from the beginning if need be. If the results are good, what can be said about varying N (sub goal 5, note that part of this sub goal goes hand in

hand with sub goal A)? Are there any modifications that can be taken to speed up the process even more or get even better result? Also check to make sure that the main goal is now fulfilled.

Note 1:
If the algorithm is finding a good solution and using few SATURN calls but still takes to long then modifications need to be made to speed up the calculations within the algorithm itself. For each modification restart the testing process to make sure the modifications are going in the right direction.

Note 2:
If good results, compared to the GA, are being found for N <= 200 but the results are much better for higher N values a couple of things should be looked at. Make sure that the GA is set up correctly and giving satisfactory and hopefully better results then results from a judgmental approach. If the GA is not giving good results then the whole testing process may be compromised. If the GA seems to be giving satisfactory result then it has now been shown that the new algorithm has the potential of finding very good results. See if it is possible to modify the algorithm so it "evolves" even faster so large network can benefit from the excellent results being produced when setting N > 200.

The testing process can be quite time consuming but is very important. It is important that the testing and modification process of any algorithm is done thoroughly to make sure that the algorithm is doing what it is thought to be doing. A poorly tested algorithm can lead to a lot of time wasted and can become extremely expensive if it is believed to be producing something it is not.

7. Reducing Data: SATURN Options

The vast amount of data and the precise as possible types of calculations being made for larger networks are what are causing much of the computational time in SATURN. The purpose with this and the next chapter is to look at some different ideas on how to reduce and relax the amount of data needed to find a near optimal closed charging cordon for larger networks.

Although the final result may differ slightly between using a “relaxed” data set up compared to the original set up does not mean that the results are incorrect. This is because, and is important to remember, that there are a lot of factors in the original set up that are not taken into consideration. Factors such as traffic accidents, weather changes, increased or decreased oil prices, the slow change towards free telecommunication, closure of a big factory resulting in major origin-destination changes, ect. which all individually affect traffic behaviour not accounted for. It may in fact be that crude, or relaxed, optimization modelling is more beneficial when looking at traffic charging schemes and cordon design.

SATURN is the traffic modeling software used by the DfT to calculate the net social welfare function for any given network together with its accompanying charging cording. SATURN is set up to calculate as accurate a net social welfare function as possible. With larger networks this calculation takes a very long time. There is however some different options within SATURN that the user may turn on, off or change in order to increase the calculation speed of SATURN but to a cost of a less accurate result. One can however argue the accuracy of any calculation performed by any traffic modeling software, but this is a different question and will not be discussed here. For now assume that what SATURN is calculating with its current setup is accurate. This chapter serves as a guideline for how a few different setups of SATURN could be used to help increase the calculation speed.

7.1. Different choices

There are several different choices that a user can choose, change and/or modify in SATURN to get satisfactory results. Only some of all the available options will be discussed here. For a full description of all the possibilities and also how to implement them see SATURN 10.5 USER MANUAL by Dirck Van Vliet and Mike Hall (2004). The choices looked at here will be of interest concerning computational time.

There are two main setup options available in SATURN which are of interest for the closed charging cordon design problem. The first one is a pure assignment setup and the other is a combination of an assignment and simulation setup. The purpose with the simulation part is that it takes into consideration the time delays that occur for travelers at junctions which might in turn lead to different root choices.

Within the Assignment methods there are four main categories:

1. All-or-nothing assignment
2. Pure stochastic assignment with fixed costs
3. Wardrop's equilibrium assignment (UE)
4. Stochastic user equilibrium assignment (Burrell multiple route assignment) (SUE)

The first two options should, as a rule of thumb, only be used with networks that have very low trip volumes. Studies have shown that the difference between UE and SUE tend to be very small with high trip volumes but that SUE gives better results for intermediate trip volumes spread across the network (Van Vliet and Hall, 2004). There are also other assignment options within SATURN but these will not be looked at here as these options tend mostly to increase the computational time.

The simulation part can be as big or small as one wishes. It is optional within

SATURN to have the entire network or only part of the network subject to simulation during the assignment-simulation setup. Note, although not discussed further there are some different parameters that can be manipulated within the simulation part itself, for example: flows, signal settings, number of junctions simulated, number of maximum iterations allowed, ect. that might be of interest.

Within the assignment method itself there are many variables that can be manipulated, turned on or turned off. For example there is the number of different user classes, meaning the number of different subsets of travelers that perceive the cost of time differently. The maximum number of iterations made by an algorithm can be changed and so on. This will be discussed further in the sections to come.

SATURN also has some inbuilt functions that can be turned on or off that are specially made to help decrease the computation time on bigger networks. One such function is of special interest for the closed charging cordon design problem and is called the UPDATE function.

7.2. Current Setup

SATURN is currently being used with the assignment and simulation combination turned on for the entire network. The assignment method used is the default setting which is to find Wardrop's equilibrium using the Frank-Wolfe algorithm. There are four different User Classes being used and all other options are set on default. See SATURN 10.5 USER MANUAL by Dirck Van Vliet and Mike Hall (2004) for what all the default settings are.

The following changes in SATURN will be suggested in various ways.

1. Turn on the Update function.
2. Use the All-or Nothing assignment.
3. Try different number of User Classes.

4. Make changes in the Frank-Wolfe assignment method.
5. Use different simulation spaces.

7.3. Changes

Changes made in the setup of SATURN should be done following a set procedure with test runs and performed under a controlled environment. There are two main ways to approach the testing of how different changes in SATURN affect the computational time and accuracy of the results. Each change made should be tested by running the GA (for a set amount of iterations of a few thousand) with the new SATURN setup and compared to the result from using the same GA (with the same number of iteration) with the 'original' SATURN setup. One approach to the testing is to start by "stripping off" one function or change at a time leading to "one step" of decreased run time and checking that the results remain satisfactory. This test should be stopped first when the results become unsatisfactory and the time decreased for a run should be noted. If the decrease in amount of time is not satisfactory then a trade off of time decrease and resulting accuracy needs to be made. The second approach is the reverse, start by "stripping off" as many extra functions and changes as possible to get the shortest computational time as possible. Then add one function or change at a time leading to longer run times and more accurate results. When the results are satisfactorily note the time decreased and check if this is satisfactory. As in the first approach, if the time saved is not satisfactory then a weighing of time vs accuracy needs to be made.

Here the second of the two above approach will be taken. However before all this is begun a test should be made by activating the 'Update' function to see what time can be saved by adding this (see subsection 7.3.1). If the time saved is not significant enough then the Update function should be kept on since it should not interfere with the final results and the second approach of testing can begin.

7.3.1. Update Function

Use the update function in SATURN (see section 15.3 in SATURN 10.5 User Manual) to decrease the number of iterations for each cordon analysis after the first cordon has been analysed. Since only slight changes to the network are being made for each cordon to be tried it may be very beneficial to start each new cordon analysis by using the end results from the last run as the initial starting values. This can help to greatly decrease the number of iterations needed in finding the net social welfare function compared to starting each analysis from scratch. The update function in SATURN does this automatically if used.

A more advanced version of this function is to use the perturbation techniques that can be used with the path-based algorithm. See chapter 21 in SATURN 10.5 User Manual for more information about this method.

7.3.2. Simple Setup

Start the testing phase with the most simple and fastest assignment method available in SATURN. Turn off the simulation setup, use only one User Class and analyze each cordon with the All-or-Nothing assignment method. This method should be fairly quick but will most likely not lead to a great result. However it is still interesting to see what the GA can come up with using SATURN with this assignment method and how fast the results are found.

7.3.3. Max Iteration and Stopping Distance

The next step to try is to reinitialize the Frank-Wolfe algorithm. This algorithm is based on the All-or-Nothing assignment but keeps modifying it with each iteration. There are four stopping parameters for the Frank-Wolfe algorithm that can be set in SATURN where two are of particular interest. One is called NITA and is the maximum number of allowed iterations for the algorithm and other is called UNCRTS. UNCRTS is set to stop the algorithm when a set “distance” between the current and ultimate solution is found. The default values for NITA

and UNCRTS are 20 and 0.2% respectively. Start by setting the NITA = 3 and UNCRTS = 10%. This should still lead to fairly quick results. Then start testing a few different runs by decreasing UNCRTS. The lower UNCRTS gets the better the final solution should become. For the first few cordons the results will most likely be fairly poorly calculated but they should become better and better since the Update function is turned on. Then try a few runs by increasing NITA and also with a few different combinations of different appropriate values of NITA and UNCRTS. Note the time it takes for each run to complete and also the differences between the end results compared to each other and the original solution.

7.3.4. User Classes and Simulation

If the results are still poor then retry 7.3.2. and 7.3.3. with more User Classes. If this does not work go back to one User Class and retry 7.3.3. using the assignment-simulation combination. It can be worth setting the simulation so only critical junctions are simulated, that is, junctions that have, or possibly will get, high traffic flows. Then up the number of User Classes and the size of the simulation space until a satisfactory result is found.

7.4. Summary

Although only roughly explained, the testing process should be done carefully so appropriate steps are taken and good combinations of changes are not missed. The SUE assignment has not been a suggested change since this in general takes a lot more computational time. The steps above which should be followed until a satisfactory balance between time and accuracy has been found are summarized.

1. Turn the Update function on.
2. Use the All-or Nothing assignment with one User Class.
3. Use the Frank-Wolfe assignment setup with a decreased maximum number of allowed iterations and an increased stopping distance parameter.

4. Redo step 2 and 3 with more User Classes.
5. Redo step 3 with part of the network being simulated and with one User Class.
6. Same as step 5 but with more User Classes.
7. Redo step 5 and 6 but with an Increased simulation space.

Hopefully before the end of step 7 a satisfactory result has been found and some time has been decreased from the total calculation time. SATURN also has a function called Partan Assignment which is a variation on the Frank-Wolfe method. It is designed to decrease the computational time but still seems to be under development. There are several other options in SATURN that are unexplored and also ways in which SATURN could be further used. For example SATURN could most likely be used to analyse the marginal social cost per link with respect on cordon fees which could lead to some development of a modification of an algorithm. There is much that can be done by manipulating and using SATRUN to speed up the computational time and it could be worth spending some time experimenting with it.

8. Research Questions

8.1. Reducing Data: Network Aggregation

This is effectively an extension on chapter 7 in that it talks about ways of reducing the amount of data needed.

A promising approach to solving the closed cordon design problem for larger networks within reasonable computational time could be by applying network aggregation. "*Broadly speaking, aggregation involves studying constituent micro processes of macro systems in order to represent the latter by a fraction of the*

complete information from the former with the greatest accuracy possible" (Uludag, Nahrstedt, Lui and Brewster, 2005). It is a topic that would require further research but seems likely lead to very favorable results. The idea would be to construct an algorithm that could automatically shrink a large network without losing vital data. Then we could solve the cordon design problem using a GA and SATURN on an aggregated network. Then we could use this solution to narrow the search area for the GA to get faster run times for the network when it is less aggregated. The idea would be to keep disaggregating the network in steps till the optimal cordon is found for the original non aggregated network.

8.1.1. Existing Literature

A lot of research has been conducted on network aggregation with heavy focus on two main areas. One area that is currently of special interest is within topological aggregation. Topological aggregation for quality of service routing is currently of great interest because of the strong growth in communication networks. Routing consists of gathering state information about each node and finding a new optimal path for a new connection. The state of a node is ideally broadcast to every other node in the network but becomes unrealistic with a strong growth of a network. Topological aggregation therefore plays an important roll in routing for communication networks to help with routing optimization algorithms. Another reason for topological aggregation of communication networks are for security reasons since it may not always be desirable to present the exact structure of a network itself.

Topological aggregation is usually done in a hierarchal fashion where a set of nodes and links get represented by a fewer number of nodes and links. The major challenge is to find the right balance between compaction and routing performance. Some basic aggregation techniques have been proposed for this which are the full-mesh, the star, the spanning tree and the Lee method but each have different short comings, see Yoo, Ahn and Kim (2006). Yoo, Ahn and Kim

(2006) propose a modified algorithm that maps nodes onto a Bruijn Graph which speeds up the computational time compared to the full-mesh approach. There are many different methods proposed to compensate the shortcomings of the full-mesh, the star and the spanning tree approach. See Uludag, Nahrstedt, Lui and Brewster (2005) for a good summary on different resent topological aggregation methods. Also Ansari (2003) conducts a clear explanation and looks at a few different existing methods. Kowalik and Collier (2004) use a game-theoretic framework to look at how three different aggregation techniques coexist in a single network. Also of interest is the paper by Sarangan, Ghosh and Acharya (2002) which propose a topological aggregation technique for state aggregation based on link flows for stochastic networks.

The other main area of research is within roadmap generalization. In roadmap generalization the main theme is to figure out which roads can be taken out in the aggregation without losing important points of interest and keep users from taking big detours when going from one point to the next. Important is also to keep the aggregated roadmap connected. Other themes consider the actual presentation of the roadmap itself by using different colors and going from double lines to single lines and so forth. Kreveld and Peschier (1998) suggest a three point method for roadmap generalization. They categorize each node and road by importance and delete unimportant roads weighed against conflicts. A conflict is when two roads lie to close together. They then delete more important roads to resolve the conflicts and finish off by adding less important roads to minimize possible detours that may have occurred. Peschier (1997) suggests a different method but with a similar ‘flavor’. One difference is that his third step involves adding roads to increase the general quality of the aggregated map for example keeping connectedness since his algorithm does not guarantee this in the first two steps.

8.1.2. Aggregation for the Closed Charging Cordon Design

An aggregation method for the closed charging cordon design problem would most likely need to be a combination of topological aggregation and roadmap generalization based on link flows. It is not however exactly clear how the existing literature could be used to help solve this problem. An idea would be to somehow keep all major and important roads disaggregated while performing a hierarchal aggregation method on smaller, less important roads. This aggregation would need to take into consideration the sum of flows going into and out of the aggregated areas.

Some main challenges would be to keep the user equilibrium accurately updated with the implementation of different cordon designs. To make sure that an optimally tolled link is not missed because it gets aggregated. To correctly represent the aggregated parts and to figure out how this should all be done automatically.

There are several possible ways to tackling this problem and a lot of information is available that handle similar situations. As mentioned it is not exactly clear though how this information might be utilized. If a specific algorithm can be constructed to handle the aggregation needed for the closed charging cordon design problem then considerable computational time could be saved.

8.2. West Midlands

In some geographical locations cities are located quite close to each other. They are located so near each other that any changes in the traffic behavior from one city will most certainly affect the traffic patterns in the other city or cities. One such area is the West Midlands containing three cities where traffic patterns interact.

There are several questions that need to be considered when trying to model traffic charging in places like these. For example how does a closed charging cordon around one city affect the traffic behaviour in the other cities? Is it even wise to set up closed charging cordons in any or all of the cities? How much traffic is drive through traffic and does not even start and end in any of the cities or their surrounding areas? How does this affect congestions within the cities themselves? What type of charging scheme is both good and realistic to implement?

A problem concerning traffic congestion modeling with places such as these is that it is not strait forward how to describe or model the space of admissible traffic charging schemes, let alone optimize over them.

9. Next Step Recommendations

This paper has presented some ideas and steps to be taken on how to possibly solve the closed charging cordon design problem within the realm of the DfT's interests. To get life into some of these ideas further work and steps are needed.

It is important to get the current GA used by the DfT up and running satisfactorily as this is currently not the case. Some of the suggestions in chapter five may be of good use for this. Since these suggestions may not be to difficult to try and may slightly speed up the process they should be considered and tried. It is also highly recommended that the different changes in SATURN be tried because they are trivial to implement.

Since it is still unclear how the closed charging cordon design problem can be solved and since the above recommendations are most likely not enough, further research is needed. A plan and budget must be set up for future research within this area to support DfT personnel, people in academia and/or consultants to

help further develop this field of knowledge.

A strongly recommended starting point is to invest in research regarding the ideas presented in chapter eight where investigation of optimal cordon design on aggregated networks seems very promising. A second topic is to figure out how to model and optimize road charging schemes in areas like the West Midlands where a cordon may not be optimal.

In parallel, a project that looks into ideas of designing a better search algorithm then the GA could be started, see chapter six. This together with an aggregation algorithm could lead to incredibly fast and reliable solutions.

There is nothing to say that different and further approaches should not be looked into and tried. It might be that faster and more suitable assignment algorithms can be found compared to those used by SATURN and that this in itself could solve the problem. Perhaps a single algorithm that optimizes the cordon design while it performs assignment iterations could be constructed using a more crude or relaxed model that still generates satisfactory result.

There is much still to be learnt and further research is of the essence. In my opinion however it would be advisable to start with the ideas presented in chapter eight and to immediately start planning and budgeting for the necessary resources needed to get this off the ground.

References

A. Ansari, “Topology Aggregation and Multi-Constrained Quality of Service Routing”, Department of Computer Science, Florida State University, 2003.

Y. Carson and A Maria, “Simulation Optimization: Method and Applications”, State University of New York at Binghamton, Department of Systems Science and Industrial Engineering, USA, 1997.

K. Kowalik and M. Collier, “Coexistence of various topology aggregation methods in a hierarchical network”, RINCE, Dublin City University, Ireland, 2004.

M. van Kreveld and J. Peshier, “On the Automated Generalization of Road Network Maps”, Department of Computer Science, Utrecht University and Compass Interactive, Netherlands, 1998.

J. Peschier, “Computer aided generalization of road network maps”, M. Sc. Thesis, Department of Computer Science, Utrecht University, Netherlands, 1997.

V. Sarangan, D. Ghosh and R. Acharya, “State Aggregation using Network Flows for Stochastic Networks”, Department of CSE, Penn State University, USA, 2002.

A. Sumalee, ”Optimal road pricing scheme design”, PhD thesis, Institute for Transport Studies, University of Leeds, UK, 2004a.

A. Sumalee, “Optimal Road User Charging Cordon Design: A Heuristic Optimization Approach”, Computer aided in Civil and Infrastructure Engineering, 19, p 377-392, 2004b.

Uludag, Nahrstedt, Lui and Brewster, “Comparative Analysis of Topology Aggregation Techniques and Approaches for the Scalability of QoS Routing”, DePaul University, University of Hong Kong, University of Illinois Urbana-Champaign, 2005.

D. Van Vliet and M. Hall, “SATURN 10.5 User Manual”, Institute for Transport Studies, University of Leeds and Atkins Planning Consultants Ltd. UK, 2004.

D. Wolpert and W. Macready, “No Free Luch Theorems for Optimization”, IEEE Transactions On Evolutionary Computation, Vol. 1, No. 1, 1997.

Yoo, Ahn and Kim, “Topology Aggregation Using a de Bruijn Graph in ATM Networks”, School of Computer Science and Engineering, Seoul National University, Department of Computer Science and Statistics, University of Seoul, Article was found by searching with Google on the internet 2006.