Genetic Algorithm-Based Combinatorial Parametric Optimization
for the Calibration of Traffic Microscopic Simulation Models

by

Tao Ma

A thesis submitted in conformity with the requirements for
the degree of Master of Applied Science in Transportation Engineering
Graduate Department of Civil Engineering
University of Toronto

© Copyright by Tao Ma April 2001
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-58769-X
Abstract

This thesis outlines an implementation of Genetic Algorithms to traffic simulation optimization and development of a program called GENOSIM, a Genetic-based Optimizer for Traffic Microscopic simulation Models. GENOSIM is developed as a pilot software that employs the state of the art in combinatorial parametric optimization to automate the tedious task of calibrating traffic simulation models. The employed global search technique, Genetic Algorithms, is integrated with a dynamic traffic microscopic simulation modeler, Paramics, and experimented with Toronto network, Canada. The output of GENOSIM is the near-optimal values of its car-following, lane changing and dynamic routing parameters. Obtained results are promising.

Paramics consists of high performance cross-linked traffic models having multiple user-adjustable parameters. Genetic Algorithms in GENOSIM will manipulate the values of control parameters and search an optimal set of values as starting configuration for these parameters by matching model outcome with observed data. The most of C++ codes shown here have been simplified for clarity.
Dedicated to

my beloved wife, WENLI GAO

and

my lovely daughter, MINGJIE MA
Acknowledgments

I would like to express my sincere gratitude to my supervisor Professor Baher Abdulhai for providing invaluable guidance and constant inspiration, which made me come to be fascinated with Genetic Algorithms and have developed the combinatorial parametric optimization software, GENOSIM.

I also want to express my appreciation to Professor Eric Miller who reviewed the thesis and gave valuable comments, which is significant to future research.

Thanks to OGSST for funding my graduate study and research.

I also extend many thanks to my parents and brother Dr. Ying Ma and those who ever offered me any help and support during the course of this research.
# Table of Contents

1 Introduction ........................................................................................................ 1

1.1 Problem Description ....................................................................................... 1

1.1.1 Paramics .................................................................................................. 2

1.1.2 ITS and Microscopic Simulation ............................................................ 3

1.2 Model Calibration or Combinatorial Parametric Optimization ................. 4

1.2.1 Conception ............................................................................................ 4

1.2.2 Significance ........................................................................................... 5

1.3 Sensitivity to Parameter Selection .......................................................... 5

1.4 The Proposed Approach vs. Literature Review .................................. 11

1.5 Motivation of GENOSIM Development ............................................. 14

2 Genetic Algorithms ....................................................................................... 16

2.1 Introduction ................................................................................................. 16

2.2 Advantage in Combinatorial Parametric Optimization .......................... 17

2.3 Conception and Principles ........................................................................ 18

2.3.1 Representations .................................................................................... 18

2.3.2 Algorithms ............................................................................................ 20

2.3.3 Selection Schema ................................................................................. 21

2.3.4 Operators ............................................................................................... 22

2.3.5 Control Parameters ............................................................................... 24

3 GENOSIM Design and Implementation .................................................. 25

3.1 Internal Structure and Interface Design ............................................... 25
List of Figures

1.1 Sensitivity of Parameter (Mean Headway) to Simulation Output ................ 8
1.2 Sensitivity of Parameter (Mean Reaction Time) to Simulation Output .... 9
1.3 Sensitivity of Parameter (Feedback) to Simulation Output .................. 9
1.4 Sensitivity of Parameter (Familiarity) to Simulation Output ............... 10
1.5 Sensitivity of Parameter (Perturbation) to Simulation Output ............. 10
1.6 GUI for Link and Lane Parameter Calibration .................................. 12
1.7 Logic Flow of Model Parameter Calibration ..................................... 14
2.1 Any Point Crossover .......................................................................... 23
2.2 Mutation at Random Bits ................................................................... 24
3.1 Hierarchy Structure of GAs’ Components ......................................... 26
3.2 GENOSIM Graphic User Interface ...................................................... 43
3.3 Flow of Integration of GAs and Paramics .......................................... 44
3.4 Testbed Network for GENOSIM Design ............................................ 45
3.5 Link Count Comparison Using Point Mean
    Absolute Error (PMAE) Objective Function ........................................ 47
3.6 Turn Count Comparison Using Point Mean
    Absolute Error (PMAE) Objective Function ........................................ 47
3.7 Link Count Comparison Using Global Relative
    Error (GRE) Objective Function .......................................................... 48
3.8 Turn Count Comparison Using Global Relative Error (GRE) Objective Function ................................................... 48

3.9 Link Count Comparison Using Point Mean

Relative Error (PMRE) Objective Function ........................................ 49

3.10 Turn Count Comparison Using Point Mean

Relative Error (PMRE) Objective Function ........................................ 49

3.11 Link Count Comparison Using Theil’s Inequality

Coefficient (TIC) Objective Function .............................................. 50

3.12 Turn Count Comparison Using Theil’s Inequality

Coefficient (TIC) Objective Function .............................................. 50

3.13 Convergence through Generations (Simple GA) ...................................... 53

3.14 Convergence through Generations (Steady-State GA) ...................................... 53

3.15 Convergence through Generations (Crowding GA) ...................................... 54

3.16 Convergence through Generations (Incremental GA) ...................................... 54

4.1 Port Area in Toronto .............................................................. 58

4.2 Random Seed Selection .............................................................. 59

4.3 Convergence through Generations (SSGA) .............................................. 61

4.4 Convergence through Generations (Crowding GA) ...................................... 61

4.5 Comparison of Link Count Using Steady-State GA ...................................... 63

4.6 Comparison of Turn Counts Using Steady-State GA ...................................... 64

4.7 Comparison of Link Counts Using Crowding GA ...................................... 64

4.8 Comparison of Turn Counts Using Crowding GA ...................................... 65
List of Tables

1.1 Sensitivity of Parameters to Model Output .............................................. 8
3.1 GA Configuration for testing different objective functions ......................... 46
3.2 Experimental Comparison of Four Candidate Functions ............................ 51
3.3 Configuration for Different GAs Comparison ........................................... 52
3.4 Optimization Result Comparison Using Different GAs .............................. 54
3.5 GA Configuration for Different Representation Comparison ....................... 55
3.6 Comparison Result of Two Representations in Two Different GAs ............ 56
4.1 GRE Fluctuation by Using Different Seed Value ....................................... 59
4.2 GA Run-Time Configuration in Real Case Study ....................................... 60
4.3 Results of GA Evolution ........................................................................... 62
4.4 Optimal Value of Calibrated Parameters .................................................. 62
4.5 GA Performance Comparison .................................................................... 63
Chapter 1 Introduction

1.1 Problem Description

Numerous problems in engineering require either maximizing a system’s performance or minimizing a “misfit” function. “Misfit” is used here to denote the error between the model output and observations for the real systems [1]. An overall system model consists of a set of cross-linked sub-models that depend on a large number of parameters. Optimization of model performance involves the selection of the ‘best’ set of values for the parameters. Best values (loosely interpreted as optimal) can be obtained using genetic algorithms, achieving a combinatorial optimization of parameters of the target system (microsimulation) through minimizing a misfit function. Problems such as these pose a great challenge due to the large parametric space to be searched. The search space is usually a multidimensional one where the values of the parameters can be conceived as coordinates, and the ‘fitness’ representing goodness of fit as a hilly surface. The process of seeking an optimum point, either a global or the best attainable local optimum point, involves some systematic search methods.

A very common optimization challenge is how to thoroughly traverse the whole search space to get to a global peak in case of unevenly distributed, non-uniform, multiple-peak space. To solve such problems, one probably resorts to either traditional analytical gradients or numerical search methods. Due to various complex reasons, the traditional methods may fail to achieve good results. In the recent years, Genetic Algorithms have
gained popularity as a generic, systems independent optimization tool [2] [3] and have shown to do quite well.

1.1.1 Paramics [4]

Paramics is a suite of high performance cross-linked traffic models that interact to replicate the urban and highway road network and simulate movement and behavior of individual vehicle in the virtual replica of a physical roadway network. Paramics models a traffic network microscopically at the individual driver/vehicle level utilizing car-following, lane changing and gap acceptance etc models. These models have multiple user-adjustable parameters by which the modeler may adjust the simulated driving behavior and evaluate the performance of the network under different scenarios. Any change in one control parameter may cause network-wide cross-effects through the linkage chains and can result in different model output. Paramics takes as input the virtual physical network and decisions parameter values and outputs the corresponding network-wide vehicular movements. To accomplish realistic traffic simulation, parameters that govern vehicle movement and behavior in the simulation models need to be calibrated using a systematic approach.

By simulating the movement of all individual vehicles through the network, a second-by-second image of the state of the network is produced. This provides engineers and planners with detailed information on the average, range and time-variation of traffic conditions. In addition, the vehicle carries a set of parameters (currently about 75) that define the physical and behavioral characteristics of the driver-vehicle unit (DVU). This detailed level of parameterization allows the complexity of a real traffic system to be
modeled far more accurately. Paramics includes a sophisticated microscopic car-following, gap acceptance and lane-changing model. Dynamic and intelligent routing are also modeled. Paramics development cycle is summarized in the following seven steps:

1. Create a new Paramics network and embed a template road geometry file
2. Build a road network by adding nodes, links and zones and coding detailed lane and junction descriptions.
3. Build demand matrices from origin/destination data and including fixed demand data such as bus routes.
4. Assign traffic using an appropriate assignment technique.
5. Collect and analyze model results.
6. Calibrate base conditions by comparing model results to observed data.
7. Validate the base model against independent data.

As with all traffic models each of the above stages requires checking and validation. Usually checking consists of comparisons of model results to observed data e.g. link flows, queue lengths, journey times etc. However, Paramics also allows the user to visualize the movement of individual vehicles through the network and this can be compared to videos from site or to local knowledge of how the network operates. This gives a mixture of quantitative and qualitative analysis.

1.1.2 ITS and Microscopic simulation

ITS or Intelligent Transportation Systems is the study of how to explore and exploit the opportunities where the advanced communication and transportation technology could be potentially and possibly integrated. Thus it is possible to improve the performance of
traffic flow on congested network, provide travelers with efficient traffic real-time information, and coordinate the provision of alternative modes of transport.

Intelligent Transportation Systems and modern urban road network design require more insight and knowledge of the constitutive properties of traffic systems. Such knowledge is needed to predict, through numerical simulations, the behavior of the vehicles under physical conditions. Reliable predictions allow for meeting the engineering requirements at a lower cost. Therefore, various traffic models have been developed to describe the driving behavior as much detailed as possible. However, Paramics, a comprehensive visualization and microscopic traffic simulation model, displayed distinguished features and filled the gap as a key component of the solution and is getting more and more popular nowadays. Through microscopic simulation, individual vehicle-driving behavior could be observed and analyzed in the network-oriented context. In order to replicate real world as accurately as possible, the grouping parameter values need to be determined so that the simulation output is valid, reasonable and reliable. The identification of combinatorial parameters for the simulator's internal models has come to be a major issue.

1.2 Model Calibration vs. Combinatorial Parametric Optimization

1.2.1 Conception
The overall traffic simulation model is a microscopic description of underlying driving behavior, via a set of internal models with a number of parameters. By changing parameter values, the simulation outcome will be different. Generally, the quality of
simulation and parameter specification can be evaluated by comparing, under given experimental conditions, observed data from the real network (field counts) with simulation results. The simulation is said to be accurate if the error between the simulation results and the observed data is small enough. It is robust if a slight change in the experimental condition result in minimal oscillations in the simulated result [5].

Therefore, the definition of parameter calibration or optimization here refers to minimizing the “misfit” by fine-tuning parameter values, thus adjusting simulation results. By iterative search using genetic algorithms, combinatorial parameter values could be eventually optimized so that discrepancy between the real system and its virtual replica is minimized.

1.2.2 Significance

Once parameters are found that adequately fit modeled to real systems’ output, then the models can be applied in several ways. It could be used as a summary way of describing reality, to make reliable predictions about further yet unobserved data, and perhaps even to give explicative power to the model to formulate efficient policy, evaluate different scenarios and aid real-time traffic control and operation.

1.3 Sensitivity to Parameter Selection

Two elementary factors are taken into account for parameter selection. One is to select those that have significant effect on model output. The other is to consider those that
closely relate to and prominently affect vehicle-driving behavior and dynamic route choice models.

In micro-simulation models in general, and in Paramics parameter pool, the magnitude of effect on driving behavior and network performance arising from different parameters are significantly different. Therefore, appropriate or inappropriate choice of decision parameters can vastly affect the final network performance, as measured by a relevant objective or fitness function.

On the other hand, one chromosome in GA represents a candidate solution, e.g. a set of parameters in our case, and is encoded in a binary bit string format. The length of the binary string is proportional to the number of parameters encoded in one chromosome. Too large length of binary string will heavily slow down the GA iterative search speed and increase computational cost. For these reasons, attention should be focused on those parameters to which the fitness function is most sensitive. Therefore, the GA optimization could be carried out with reasonable population size and number of generations. Also, since we select parameters with relatively higher impact on overall performance, good improvements of the fitness function could be realized within fewer generations, fewer function evaluations and fewer runs of simulation.

In this research, five top sensitive parameters are selected for calibration: mean headway, mean reaction time, perturbation, feedback and familiarity. The first two parameters influence network wide vehicle-driving behavior, the other three, perturbation, feedback and familiarity, are key parameters that enable routes to be updated dynamically in response to ITS or network conditions. The five parameters are defined as follows:
(1) Mean headway is the average time between leading edges of successive vehicles.

(2) Mean reaction time is the time taken by driver to implement any action through the simulation period. (e.g. car following, gap acceptance ... etc.)

(3) Perturbation is a factor used to randomize the route-cost perception to effect stochastic route choice.

(4) Feedback is a loop mechanism used to update travel time costs for equipped drivers throughout a simulation period in order to influence route choice.

(5) Familiarity is also a factor describing the composition of the driver population with respect to their different levels of knowledge of the network with network details and rat runs, ultimately affecting route choice.

In Paramics, the default value of parameters are mean headway = 1.0 sec, mean reaction time = 1.0 sec, perturbation = 5%, feedback = 300 sec and familiarity = 85%. Each parameter has been individually tested for its sensitivity to the fluctuation on model output. In each experiment, only the value of one of five parameters has been changed so that it is different from its default value and the other parameters are kept on default values. The table 1.1 summarizes the experimental results, which indicate the sensitivity of these parameters to model output.
Table 1.1 Sensitivity of Parameters to Model Output

The figure 1.1 ~ 1.5 show the changes in link count by adjusting parameter value. The experimental data is explained in 3.3 Chapter 3.
Figure 1.2

Figure 1.3
Sensitivity of Parameter (Familiarity) to Simulation Output

Figure 1.4

Sensitivity of Parameter (Perturbation) to Simulation Output

Figure 1.5
1.4 The Proposed Approach vs. Literature Review

In Paramics documents, it was suggested that calibration is an essential stage of the traffic modeling process. Usually this procedure requires comparing model results to observed data and ensuring that these comparisons fall within acceptable guidelines.

As part of model calibration the visualization of traffic moving through the road network may be compared to junction video information or local knowledge of traffic operation. In addition, the output statistics such as turning count, link count, journey times and queue length etc. can be collected and compared to data observed on site.

The calibration procedure for Paramics involves standard checks that are carried out to verify vehicle routing, traffic demand matrices, road layout details etc. After completing these checks and verifying they are acceptable, the user must output model statistics for comparison to observed data. Comparisons between modeled and observed data should include some of the following assessments: traffic flows, queue lengths and stop time delays, traffic speeds, traffic density, and journey times. Modeled routing compare to an assessment of actual routing. If the observed to modeled comparisons are within recommended guidelines and the visualization of the vehicles is realistic, then the model is deemed to be calibrated. However, if the comparisons are unacceptable then the user should carry out further analysis to

- Assess traffic assignment
- Check road geometry (visualization)
- Check routes
- Check observed data
Check traffic demand

The checking procedure should identify any coding errors that can be quickly rectified using the editor function. The updated model is then re-run and results are collected and analyzed. The process is repeated until good calibration is achieved.

If the model still gives inaccurate simulation then the user may wish to change some default parameters coded in Paramics to model the behavior of individual driver vehicle units (DVU). Usually, the major adjustments to a model include moving kerb and stop-line control points, and coding forced lane changes to override default Paramics lane changing. The main link, lane, and junction calibration parameters include (refer to figure 1.6)

- Changing the hazard warning distance
- Including link gradients
- Coding junction visibility
- Changing link headway factor
- Coding link end speeds
- Coding lane endstop time
- Forced lane changes
- Forced merges
- Stay in lane
- Lane and turn restrictions

Figure 1.6 GUI for Link and Lane Parameter Calibration
Overall, above Paramics documents do point out how to consider or carry out the calibration, either by correcting the network-input error or tuning some parameters. However, it didn’t propose any efficient systematic method for calibration. All the ways mentioned above have to be done manually by graphic user interface. Actually, in real projects, it’s impractical to do calibration manually. Therefore, an efficient and realistic solution has to be formulated so that calibration could be implemented automatically instead of manually.

The trial and error method was applied to this same problem in 1998 [6] in Irvine USA. In 2000, the use of the Genetic Algorithm for Paramics parameter calibration was proposed and explored on a short freeway link, within Paramics’ Application Programming Interface (API) [7]. In that case, only two driving parameters were calibrated without inclusion of any routing parameters and without full automation. An elaborate application of genetic algorithms and full automatic mechanism are still absent however.

It has been proven by many decades of experience and practice of optimization research that Genetic Algorithms is an extensively used search technique that borrows ideas from natural evolution to effectively find good solutions for combinatorial parametric optimization problems. Therefore, GENOSIM is created as an independent optimization tool that externally links to any simulation environment, thus having a potential of interacting with simulation environments other than Paramics.

As described in Figure 1.7, the misfit is evaluated through the difference between the observed data and simulation outcome. The term ‘optimization’ is loosely used here to
indicate finding parameter values for the simulation models that minimize the misfit function. However, genetic algorithms do not necessarily guarantee global optimality in the mathematical sense.

Figure 1.7 Logic Flow of Model Parameter Calibration

1.5 Motivation of GENOSIM Development

Our interest in using Genetic Algorithms to tackle combinatorial parametric optimization problem is motivated by the need for a systematic approach to calibrate the increasingly popular traffic micro-simulation models for Intelligent Transportation Systems (ITS) applications, thus offering an automated optimization tool for microscopic simulation parameters. However, the process suggested in Paramics documents seems to be impractical, because users have to keep on fine tuning parameter values manually to fit the simulation output with observed data. In order to implement an idea that automates model calibration, GENOSIM, a Windows-based program, has been developed in C++.
By using it, the user can easily and automatically optimize the parameter values for the simulation model.

In this thesis we introduce and describe GENOSIM: a Genetic Algorithms-based simulation-optimization system. The objective of GENOSIM is to search for optimal configuration for microscopic simulation parameters in order to minimize discrepancy between simulation output and real field conditions.
Chapter 2 Genetic Algorithms

2.1 Introduction

"...the metaphor underlying genetic algorithms is that of natural evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to a complicated and changing environment. The ‘knowledge’ that each species has gained is embodied in the makeup of the chromosomes of its members.” [15]

During past three decades, the grown demand on combinatorial optimization problems put genetic algorithms in a significant position. Genetic Algorithms have been quite widely and successfully applied to numerous optimization problems like wire routing, scheduling, adaptive control, game playing, cognitive modeling, transportation problems, traveling salesman problems, optimal control problems, database query optimization, etc.

Genetic Algorithms (GAs) are a stochastic search method based on the principles and mechanisms of natural selection and ‘survival of the fittest’ from natural evolution. GAs has come to be a popular optimization method since introduced in 1970s by Holland’s study of adaptation in artificial and natural systems [8]. By simulating natural evolutionary processes, a GA can effectively search the problem domain thoroughly on population-based solutions rather than a single solution, and employ heuristics to evolve better solution. The facility of restarting the iterative search from a wide variety of starting points provides some safeguard against entrapment on a local optima, thus making GAs prevail over conventional search methods.
A GA performs a multi-directional search by maintaining a population of potential solutions and encourages information formation and exchange between these directions. The population undergoes a simulated evolution: at each generation the relatively “good” solutions reproduce, while the relatively “bad” solutions die. To distinguish between different solutions, an objective function is used for evaluation that plays the role of environment.

2.2 GAs Advantages in Combinatorial Parametric Optimization

Various conventional optimization techniques have been developed and implemented in practical applications, such as analytical methods (least mean-squares or maximum likelihood estimates), various types of hill climbing, randomized search and trial and error.

However, many models and their misfit functions cannot be expressed either in an analytically soluble form or with differentiable error function suitable for gradient-guided search techniques. Conventional approaches can therefore easily fail in obtaining the global optimum in complex search space situations in practice. For many such cases that cannot be optimized analytically, various “hill-climbing” or “valley-descending” techniques have been used to search toward an optimum in iterative loop. But, For problems which have multiple local optima, both the iterative incremental step and steepest descent methods lead to a danger of entrapment on local optima and saddle points. [1] In addition, some conventional optimization methods suffer from lack of a
priori information on the system parameters or cannot easily be applied to nonlinear systems.

Genetic Algorithms have been found to be particularly effective and powerful in exploring and exploiting poorly understood or non-differentiable spaces for optimization and machine learning. It has also been successfully applied to systems identification and parameter estimation [9][10]. For this type of problem, genetic algorithms have the advantage that all parts of the feasible space are potentially available for exploration and exploitation. So the global optima stand better chance of being attained. Overall, Genetic Algorithms have unique advantages and charisma in solving the issues of combinatorial parametric optimization.

2.3 Conception and Principles

The following is a brief description of key components of GA:

2.3.1 Representation

Chromosome: one distinct element in the GA is the chromosome that is encoded as a single solution. A single solution here means one set of values of combinatorial parameters for the simulation model. One chromosome is subdivided into genes. Associated with each chromosome is a misfit value, which determines its chance to survive and produce offspring.

Genes: a gene, a bit string, is a binary representation of a single parameter value, which must have an upper and lower bound declared. The length of the bit string is of paramount importance. It determines how precise a point the GA could reach in the
search space. The longer the binary bit string is, the better but at the expense of high computation cost. In this research, a 16 or 8 bit is used for each gene to attempt to make every possible point in the search space reachable from the initial population through genetic operators.

Real-Valued and Binary: The standard GAs are characterized by the use of the binary coding for each string. However, in this paper, both real (decimal) and binary formats are offered because, as pointed out by Wright (1991), real-valued genes offer a number of advantages in numerical optimization:

1. Efficiency of the GA is increased, as there is no need to convert chromosomes between binary and decimal before each function evaluation
2. Less memory is required
3. There is no loss in precision
4. There is greater freedom to use different genetic operators
5. Real-valued GA can take advantage of higher mutation rate than binary coding, thereby increasing the level of possible exploration of the search space.[11] [12]

The binary representation traditionally used in genetic algorithms has some drawbacks when applied to multidimensional, high-precision numerical problems. For example, for 100 variables with domains in the range [-500, 500] where a precision of six digits after the decimal point is required, the length of binary solution vector is 3000. This, in turn, generates a search space of about $10^{1000}$. For such problems genetic algorithms perform poorly [14]. In particular, for parameter optimization problem with variables over continuous domains, real-number representation performs better than binary code. In
GENOSIM design, two types of representation have been implemented and examined for their difference.

2.3.2 Algorithms

Although, there are many versions of genetic algorithms, all of them have two basic steps: selection and replacement. The selection step determines how the parents are selected for producing offspring. The replacement step determines how offsprings will be inserted into the population and how old members are to be eliminated.

If the problem requires iteration on more than two parameters, then it may be very difficult to check for local optima. In order to avoid a danger of entrapment on local optima or saddle points and to examine what type of GA is more efficient for combinatorial parameter optimization, four types of GAs are implemented in GENOSIM: simple GA, steady-state GA, crowding GA and incremental GA.

Simple GA (Non-Overlapping Populations)

The simple genetic algorithm is a very common implementation. It uses non-overlapping populations. In each generation, the entire population is replaced with new individuals. If the elitism mechanism is specified, the best individual will be carried over from one generation to the next to increase converge speed. The best individual is more likely to be selected for mating [13].

Steady-state GA (Overlapping Populations)

The steady-state genetic algorithm uses overlapping populations. In each generation, a portion of the population is replaced by the newly generated individuals. The steady-state
algorithm is another standard genetic algorithm. If only one or two individuals may be replaced in each generation, it is so-called Incremental GA. At the other extreme, the steady-state algorithm becomes a simple genetic algorithm when the entire population is replaced [13].

Crowding-Based GA

Crowding is a generalization of pre-selection. In crowding GA, selection and reproduction are the same as in the SGA; but replacement is a distinct feature. Before replacement, the new offspring will execute a comparison with individuals of the population using a distance function as a measure of similarity. The population member that is most similar to the offspring is replaced by the offspring. This procedure is repeated. This strategy maintains the diversity in the population and slows down premature convergence of the traditional GA, thus making crowding GA prevail over the others. In most cases, It can find the global optimum in a multi-dimensional search space [13].

2.3.3 Selection Schema

Selection is the process of choosing two parents from the population for exchanging genetic materials. Four types of selection schemas have been implemented into GENOSIM: roulette wheel, tournament, rank and uniform.

Roulette Wheel: Roulette selection is one of the traditional GA selection techniques. This selection method picks an individual based on the magnitude of the fitness relative to the rest of the population. A target value is set, which is a random proportion of the sum of
the fitness in the population. The population is stepped through until the target value is reached. This is only a moderately strong selection technique, since fit individuals are not guaranteed to be selected for, but the higher the fitness, the more likely an individual will be selected. Any individual has a probability $p$ of being chosen where $p$ is equal to the fitness of the individual divided by the sum of the fitness of each individual in the population. It is essential that the population not be sorted by fitness, since this would dramatically bias the selection.

Tournament: The tournament selection uses the roulette wheel method to select two individuals then picks the one with the higher fitness. The tournament selection typically chooses higher valued individuals more often than the Roulette Wheel.

Rank: The rank selection picks the best member of the population every time.

Uniform: The stochastic uniform selection picks randomly from the population. Any individual in the population has a probability $p$ of being chosen where $p$ is equal to $1$ divided by the population size.

2.3.4 Operators

During the reproduction phase of genetic algorithm, there are two classical genetic operators: crossover and mutation.

Crossover: This operator creates new offspring from the original parents. In this implementation the parent chromosomes are grouped into pairs through selection schemata and each of these pairs creates two child chromosomes by a process of
combination. The following two chromosomes have been selected to exchange genetic material and reproduce two new child chromosomes. (refer to figure 2.1)

![Figure 2.1 Any Point Crossover](image)

Mutation: This is the process of randomly disturbing genetic information. The mutation probability determines the rate at which a gene of chromosome is altered. Mutation add new information in a random way to genetic search process, and ultimately help to avoid GAs from getting stuck at local optimums. Mutation operates at the bit copy level; when the bits are being copied from the parent to the child, there is a probability that each bit may become mutated. The probability is usually set to a quite small value. A coin toss mechanism is employed; if a random number between 0 and 1 is less than mutation probability, the bit is inverted, e.g. zeros become ones and ones become zeros. This random scattering might find better optima or even modify a part of a genetic material that will be beneficial in a later crossing. On the other hand it might produce a weak individual that will never be selected for crossing. (refer to figure 2.2)
2.3.5 Control Parameters

The population size, generations, probability of crossover, probability of mutation are the control parameters for GAs run. User configures the run time setting for genetic algorithm through specifying these control parameters. The search process usually stops at the specified number of generations.
Chapter 3 GENOSIM Design and Implementation

3.1 Internal Structure and Interface Design

There are many decisions to be made when designing a problem-specific GAs. To some extent, configuring a GA suitable to a particular problem is an art.

3.1.1 Structure of Components

Currently, various Genetic Algorithms and its components have been developed and applied to different specific issues. Genetic Algorithms are regarded as a problem-specific optimization method. In GENOSIM design for combinatorial parametric optimization, various scenarios have been implemented in chromosome representation, algorithms, selection schemas and operators. Different options are offered and experimental comparisons have been made for different option combination. Eventually, the suitable combinatorial option is recommended as a highly efficient and efficacy configuration for the real case study. Meanwhile, four objective functions have been tested for accuracy and efficiency and the global relative error function is employed in the design as objective function. Figure 3.1 shows hierarchy structure of genetic algorithms designed in GENOSIM.

Representation (Real-Valued and Binary)

The two distinct elements in the GA are individuals and population. An individual is a single solution. The population is the set of individuals currently involved in the search process. A gene, a calibrated parameter, is a component of an individual and can be represented either in a binary code or real number. The binary representation is a bit
string of arbitrary length that indicates the number of intervals between a lower bound and upper bound of the parameter. The range declared for a parameter is divided into the number of intervals that is a function of the length of gene’s bit string. A bit string of length n can represent NI intervals by following expression (equation 3.1):

\[ NI = 2^n - 1 \]

(Hierarchy Structure of GA Components in GENOSIM)

- Representation
  - Binary
  - Real Number

- Algorithms
  - Simple GA
  - Steady-State GA
  - Crowding GA
  - Incremental GA

- Selection Schema
  - Roulettewheel
  - Tournament
  - Rank
  - Uniform

- Operators
  - Crossover
    - Uniform (Binary/Real)
    - One-Point (Binary/Real)
    - Two-Point (Binary/Real)
    - Blending (Real)
    - Averaging (Real)
  - Mutation
    - Bit-Flip (Binary)
    - Uniform (Real)
    - Gaussian (Real)
    - Boundary (Real)

- Control Parameters
  - Population Size
  - Generations
  - Probability of Crossover
  - Probability of Mutation

Figure 3.1 Hierarchy Structure of GAs’ Components
The size of the interval would be SI by the following expression: (equation 3.2)

\[
SI = \frac{B_{\text{upper}} - B_{\text{lower}}}{NI}
\] (3.2)

For example, Parameter A has a range of 0.5 ~ 2.0 and the bit string is eight bits in length. This gives 255 intervals and an internal size of 0.00588. The bit string ‘00000000’ represents zero intervals, so the gene’s value is equal to the lower bound.

\[
0.5 + (0 * 0.00588) = 0.5
\]

The bit string ‘11111111’ represents 255 intervals, so the gene’s value is equal to the upper bound.

\[
0.5 + (255 * 0.00588) \approx 2.0
\]

The bit string ‘01101101’ represents 109 intervals, so the gene’s value is equal to 1.14092.

\[
0.5 + (109 * 0.00588) \approx 1.14092
\]

The size of the bit string is of paramount importance. If the interval is too coarse, the GA may never be able to find an optima simply because the genes can’t express a value close enough to the optima. Too fine a resolution may result in excessive hair splitting in the search.
Selection Schema

/**
 * Roulette Wheel Selector
 *
 * This selection routine screens the members of the population through a weighted roulette wheel. Likelihood of selection is proportional to the fitness score. It assumes that the individuals are in order from best (0th) to worst (n-1). When the summation of fitness before ith individual reaches the random target value, the (i-1)th individual will be selected.
 */

RouletteWheel() {
    float Target;
    int i, upper, lower;
    Target = Population.SumFitness*Random(0,1);
    lower = 1;
    upper = Population.size();
    while(upper >= lower) {
        i = lower + (upper-lower)/2;
        if(SumFit[i] > Target)
            upper = i-1;
        else
            lower = i+1;
    }
    lower = Min(Population.size(), lower);
    lower = Max(1, lower);
    return Population.individual(lower);
}
Tournament Selector

Tournament method picks up two individuals from the population using the Roulette Wheel selection method. Then return the better of the two individuals.

Tournament()
{
    int pick1, pick2;
    float Target;
    int i, upper, lower;

    Target = Population.SumFitness*Random(0,1);
    lower = 1;
    upper = Population.size();
    while(upper >= lower) {
        i = lower + (upper-lower)/2;
        if(SumFit[i] > Target)
            upper = i-1;
        else
            lower = i+1;
    }
    lower = Min(Population.size(), lower);
    lower = Max(1, lower);
    pick1 = lower;

    Target = Population.SumFitness*Random(0,1);
    lower = 1;
    upper = Population.size();
    while(upper >= lower) {
        i = lower + (upper-lower)/2;
        if(SumFit[i] > Target)
            upper = i-1;
        else
            lower = i+1;
    }
    lower = Min(Population.size(), lower);
    lower = Max(1, lower);
pick2 = lower;

if (Population.individual(pick1).score() > Population.individual(pick2).score())
    X = Population.individual(pick1);
else
    X = Population.individual(pick2);
return X;

/* ---------------------------------------------------------------

Rank Selector
This routine always returns the best individual from the population. Any population may contain more than one individual with the same score. This method must be able to return any one of those 'best' individuals.

--------------------------------------------------------------- */

Rank() {
    int i, count=1;
    float Best[Population.size()];
    Population.SortFitness();
    Best(1) = Population.individual(1);
    for(i=2; i<Population.size(); i++){
        if(Population.individual(i).fitness() == Population.individual(1).fitness()){
            count++;
            Best(count) = Population.individual(i);
        }
    }
    return Population.Best(RandomInteger(1,count));
}
/* Uniform Selector

Randomly select an individual from the population. */

Uniform() {
    float Z;
    int I;
    I = RandomInteger(1, Population.size());
    Z = Population.individual(I);
    return Z;
}

Operators

/* Uniform Crossover

Randomly take bits from each parent. Flip a coin for each bit to see if that bit should come from the mother or the father. Assume strings have the same length. */

UniformCrossover() {
    int i;
    for(i=dauhter/son/mom/dad.length(); i>=1; i--){
        if(Random(0 or 1)){
            daughter.gene(i, mom.gene(i));
            son.gene(i, dad.gene(i));
        }
        else{
            daughter.gene(i, dad.gene(i));
            son.gene(i, mom.gene(i));
        }
    }
}
One-Point Crossover

Pick a single point then copy genetic material from each parent.

---

OnePointCrossover() {
    unsigned int cutsite;
    cutsite = RandomInteger(0, mom or dad.length());
    momsection = mom.length() - cutsite;
    dadsection = dad.length() - cutsite;
    daughter.copy(mom, 0, 0, cutsite);
    daughter.copy(dad, cutsite, cutsite, dadsection);
    son.copy(dad, 0, 0, cutsite);
    son.copy(mom, cutsite, cutsite, momsection);
}

---

Two-Point Crossover

Two-point crossover is similar to the single point crossover, but randomly pick two points then exchange the sections between those two points.

---

TwoPointCrossover() {
    unsigned int cutsite1, cutsite2, tmp;
    unsigned int midsection, lastsection;
    cutsite1 = RandomInteger(0, mom or dad.length());
    cutsite2 = RandomInteger(0, mom or dad.length());

    if(cutsitel > cutsite2) {
        tmp = cutsitel;
        cutsitel = cutsite2;
        cutsite2 = tmp;
    }

    midsection = cutsite2 - cutsitel;
lastsection = mom.length() - cutsite2;

daughter.copy(mom, 0, 0, cutsite1);
daughter.copy(dad, cutsite1, cutsite1, midsection);
daughter.copy(mom, cutsite2, cutsite2, lastsection);
son.copy(dad, 0, 0, cutsite1);
son.copy(mom, cutsite1, cutsite1, midsection);
son.copy(dad, cutsite2, cutsite2, lastsection);

/*
Blending Crossover
Recalculate the boundary based on two parents' distance. Then generate new children
within new boundary.
*/

BlendCrossover() {
    for(int i=0; i<parent.length(); i++) {
        float low = Min(mom.gene(i), dad.gene(i));
        float hig = Max(mom.gene(i), dad.gene(i));
        float distance = (float) (hig - low)/2.0;
        low -= distance;
        hig += distance;
        low = Max(low, lowerbound);
        hig = Min(hig, upperbound);
        child.gene(i, RandomFloat(low, hig));
    }
}
Averaging Crossover (Real)

Take the average of two parents' genetic material as children genetic material.

```c
AveragingCrossover() {
    float average;
    for(int i=0; i<parent.length(); i++) {
        average = (mom.gene(i) + dad.gene(i))/2.0;
        average = Max(average, lowerbound);
        average = Min(average, upperbound);
        child.gene(i, average);
    }
}
```

Bit-Flip Mutation

If the mutation probability is small, then we toss coin for each bit in the string see if that bit mutate. Otherwise, simply mutate a known number of bits based on the mutation rate and length of bit string.

```c
BitFlipMutation() {
    float nMut = pmut * child.length();
    if(nMut < 1.0){
        for(i=child.length()-1; i>=0; i--){
            if(TossCoin(pmut)){
                child.gene(i, ((child.gene(i) == 0) ? 1 : 0));
            }
        }
    }
}
```
else{
    for(n=0; n<nMut; n++){
        i = RandomInteger(0, child.length()-1);
        child.gene(i, ((child.gene(i) == 0) ? 1 : 0));
    }
}

/**************************************************************************

Gaussian Mutation

Genetic material of an individual is replaced with a random number from Gaussian
distribution defined by a mean and standard deviation.

***************************************************************************/

GaussianMutation() {
    for(int i=0; i<individual.length(); i++) {
        if(TossCoin(pmut)) {
            float new;
            do {
                new = Gaussian(individual.gene(i));
            } while(new < individual.gene(i).lower() || new > 
                individual.gene(i).upper());
            individual.gene(i, new);  
        }
    }
}
Uniform Mutation
Genetic material of an individual is replaced by a random number within declared boundary.

```java
UniformMutation() {
    for (int i=0; i<individual.length(); i++) {
        if (TossCoin(pmut)) {
            float new, low, hig;
            low = individual.gene(i).lower();
            hig = individual.gene(i).upper();
            new = RandomGenerator(low, hig);
            individual.gene(i, new);
        }
    }
}
```

Boundary Mutation
The genetic material is replaced either by lower bound or upper bound.

```java
BoundaryMutation() {
    for (int i=0; i<individual.length(); i++) {
        if (TossCoin(pmut)) {
            float low, hig;
            if (Random(0, 1)) {
                low = individual.gene(i).lower();
                individual.gene(i, low);
            } else {
                hig = individual.gene(i).upper();
                individual.gene(i, hig);
            }
        }
    }
}
```
Simple GA
Evolve a new generation. When this routine starts, population contains the current generation. When finishes, population contains the new generation and old Population contains the current generation. The previous old generation is replaced. The best individual in old Population will migrate into new generation. If supposed to be elitist, migrate the best individual from the old population into the new population.

SimpleGA()

```cpp
SimpleGA()
{
    tmppopulation = oldPopulation;
    oldPopulation = population;
    population = tmppopulation;

    for(i=0; i<pop->size()-1; i+=2) {
        mom = oldPopulation.select();
        dad = oldPopulation.select();
        if(TossCoin(pCrossover())) {
            GACrossOver(mom, dad, population.individual(i),
                        population.individual(i+1));
        }
        else {
            population.individual( i ).copy(mom);
            population.individual(i+1).copy(dad);
        }
        population.individual( i ).mutate(pMutation());
        population.individual(i+1).mutate(pMutation());
    }
    if(population.size() % 2 != 0) {
```
mom = oldPopulation.select();
dad = oldPopulation.select();

if(TossCoin(pCrossover())) {
    GACrossover(mom, dad, population.individual(i), 0);
} else {
    if(Randomflip())
        population.individual(i).copy(mom);
    else
        population.individual(i).copy(dad);
}
population.individual(i).mutate(pMutation());

population.evaluate();

if(oldPopulation.best().score() > population.best().score())
    population.replace(oldPopulation.best());
}

/* ---------------------------------------------------------------

Steady-State GA
A steady-state GA has no 'old' and 'new' populations. Pick parents from the current population and save new ones in temporary population. Migrate all new ones into current population and replace the worst members of the proceeding population. Keep original population size.
--------------------------------------------------------------- */

SteadyStateGA() {
    for(i=0; i<tmpPop.size()-1; i+=2) {
        mom = pop.select();
        dad = pop.select();
        if(TossCoin(pCrossover())) {

GACrossover(mom, dad, tmpPop.individual(i),
    tmpPop.individual(i+1));
}
else{
    tmpPop.individual( i ).copy(mom);
    tmpPop.individual(i+1).copy(dad);
}

if(tmpPop.size() % 2 != 0) {
    mom = pop.select();
    dad = pop.select();
    if(TossCoin(pCrossover())) {
        GACrossover(mom, dad, tmpPop.individual(i), 0);
    }
    else{
        if(Randomflip())
            tmpPop.individual( i ).copy(mom);
        else
            tmpPop.individual( i ).copy(dad);
    }
    tmpPop.individual( i ).mutate(pMutation());
}
for(i=0; i<tmpPop.size(); i++) {
    pop.add(tmpPop.individual(i));
}
pop.evaluate();
pop.remove(WORST, population.size());
}
Crowding GA

Clone current members into mating pool, pick parents from mating pool and remove them from the pool. Create new children and compare with the parents. Then return higher fitness members to population.

CrowdingGA() {
    for (int i=0; i<pop.size(); i++) {
        matingpool.clone(i);
    }
    do {
        mom = pop.Randomindividual();
        matingpool.remove(mom);

        dad = pop.Randomindividual();
        matingpool.remove(dad);

        GACrossover(mom, dad, child, 0);
        Child.mutate(pMutation());

        float d1 = child.compare(mom);
        float d2 = child.compare(dad);
        FitnessStrategy == minimize;
        if (d1 < d2 &&) {
            if (child.score() < mom.score()) {
                mom.copy(child);
            }
        } else {
            if (child.score() < dad.score()) {
                dad.copy(child);
            }
        }
    } while (matingpool.size() > 1);
Incremental GA

Similar to a steady-state GA. Pick parents from the current population and replace its members with the new ones created. Only generate two children in each generation. The replacement strategy is set by the GA (either replace their parent or the worst member).

IncrementalGA() {
    mom = population.select();
    dad = population.select();

    if(TossCoin(pCrossover())) {
        GACrossover(mom, dad, child1, child2);
    } else{
        child1.copy(mom);
        child2.copy(dad);
    }
    child1.mutate(pMutation());
    child2.mutate(pMutation());

    child1 = population.replace(child1, replacementstrategy);
    child2 = population.replace(child2, replacementstrategy);

    population.evaluate();
}

3.1.2 Interface Design

GENOSIM, a standard Windows-based application, has graphic user interface (GUI). To execute the optimization process, the user needs to follow two steps. First step is to set up the run-time configuration for genetic algorithms. By pressing the "setting" button, six
sub-windows will pop up. There are multiple options in each sub-window. The user can easily specify the run-time option for representation, operator, selection schema, algorithm etc. and determine the control parameter values for genetic algorithm such as population size, generations, and crossover rate and mutation rate. After finish run-time setting, second step is to press "evolve" button to run GENOSIM. Figure 3.2 shows the main window of GENOSIM.

On the left side of main window, there are three graphs that describe the searching process and performance of genetic algorithms. The first graph describes the evolution convergence information. There are two major curves in the first graph. The upper curve represents the population average misfit, whereas the lower one describes the minimum misfit in each generation. When mean misfit curve (upper) goes down to meet minimum misfit curve (lower) as searching processes, then the convergence appears. The second graph shows evolutionary state of individual members in each generation. The third graph displays the change of misfit summation and worst individual of each generation. On the right side, there are three tables that show the dynamic numerical results of optimization process. The first table indicates the configuration of genetic algorithms during run time. The middle one shows the statistics of misfit of each generation such as mean, best, worst, standard deviation, variance and summation of misfit of each generation. The last table displays the dynamic optimal results of calibrated parameters as the optimization progresses. When the searching process terminates, the last table shows the final optimal values of calibrated parameters by one-time complete search.
Figure 3.2 GENOSIM Graphic User Interface

3.2 Integration of GAs with Paramics

To integrate GAs with the simulation environment, two aspects must be taken into account. The first is the combinatorial parameter configuration that the GA will manipulate. Each parameter must have its domain declared. This is the range of useful values that the parameter may take. The sets of parameter values shape up the solutions for the simulation models, and it is this solution set that the GA will try to optimize.
The second is the evaluation of the solution set. At the end of a simulation run, the model outcomes are compared against real target values. The closer to the target value, the better the solution set is. The deviations from the targets are aggregated to a total misfit value. A misfit of zero is a perfect solution.

Figure 3.3 illustrates how Genetic Algorithm produce a set of values for the combinatorial parameters of the simulation model, which are then passed to Paramics models via a text configuration file. A simulation run is then automatically triggered based on that configuration. The GA iterative loop halts until the simulation stops and results are generated. Then the simulation outcome is read into GA objective function from the output text files and converted to a misfit value. The misfit is calculated based on the difference between model output from various configurations and corresponding observed data.

Figure 3.3 Flow of Integration of GAs and Paramics
3.3 Experimental Comparison

3.3.1 Experimental Data

Figure 3.4 shows a small roadway network that was employed as a testbed for GENOSIM development. It is part of downtown area in Toronto, Canada. The east-west bounds are between Jarvis and Parliament Street, and north-south bounds are from Queen to Front Street. In this simulation network, there are 396 links, 53 junctions, and 149 nodes. The Origin-Destination (O/D) matrix is generated based on the Toronto Tomorrow Travel Survey in 1996, via traffic assignment using EMME/2. Approximately 7546 vehicles travel demand is released during morning peak hour from 8:00-9:00am. The survey vehicles consist of 85% cars and 15% light trucks. The one-hour turning counts at fifteen key signalized intersections are employed for fitness computation.

![Figure 3.4 Testbed Network for GENOSIM Design](image-url)
3.3.2 Different Scenarios

3.3.2.1 Objective Function Selection

A good objective function plays a critical role in obtaining good results. Four objective functions were experimented with respectively. The misfit function ultimately adopted is the "Global Relative Error". In this section, four objective functions will be individually discussed in details. In each experiment one of four candidate objective functions is coded into GENOSIM. Use the same GA setting (Table 2) to trigger optimizing process. After each run, the modeled link count and turning count are compared to observed data to evaluate the corresponding objective function. Table 3.1 indicates the run-time configuration of genetic algorithm for testing different objective functions. Figure 3.5 ~ 3.12 show the comparison curves of modeled link, turning count and on-site data when each objective function was applied to GENOSIM respectively.

<table>
<thead>
<tr>
<th>Run-Time Setting:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generations</td>
</tr>
<tr>
<td>Population Size</td>
</tr>
<tr>
<td>Probability of Crossover</td>
</tr>
<tr>
<td>Probability of Mutation</td>
</tr>
<tr>
<td>Representation</td>
</tr>
<tr>
<td>Crossover Option</td>
</tr>
<tr>
<td>Mutation Option</td>
</tr>
<tr>
<td>Algorithm Option</td>
</tr>
<tr>
<td>Parent Selection</td>
</tr>
</tbody>
</table>

Table 3.1 GA Configuration for testing different objective functions
Using Objective Function 1: “Point Mean Absolute Error (PMAE)”

\[
\text{Misfit} = \frac{1}{n} \sum_{i=1}^{n} | Q_{\text{real}} - Q_{\text{sim}} |
\]

**Link Count Comparison Using Point Mean Absolute Error (PMAE) Objective Function**

![Link Count Comparison Diagram](image1)

Figure 3.5

**Turn Count Comparison Using Point Mean Absolute Error (PMAE) Objective Function**

![Turn Count Comparison Diagram](image2)

Figure 3.6
Using Objective Function 2: “Global Relative Error (GRE)”

\[
\text{Misfit} = \frac{\sum_{i=1}^{n} | Q_{real}^{i} - Q_{sim}^{i} |}{\sum_{i=1}^{n} Q_{real}^{i}}
\]

**Figure 3.7**

**Figure 3.8**
Using Objective Function 3: "Point Mean Relative Error" (PMRE)

\[ \text{Misfit} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left( \frac{Q_{\text{real}} - Q_{\text{sim}}}{Q_{\text{real}}} \right)^2} \times 100 \%
\]

**Figure 3.9**

**Figure 3.10**
Objective Function 4: "Theil’s Inequality Coefficient" (Pindyck and Rubinfeld 1981)

\[
\text{Misfit} = \frac{\left[ \frac{1}{n} \sum_{i=1}^{n} \left( \frac{Q_{\text{act}} - Q_{\text{sim}}}{Q_{\text{act}}} \right) \right]^2}{\frac{1}{n} \sum_{i=1}^{n} Q_{\text{act}}^2 + \frac{1}{n} \sum_{i=1}^{n} Q_{\text{sim}}^2}
\]

Figure 3.11

Figure 3.12
In the experiment with objective functions, table 3.1 shows the run-time configuration of genetic algorithm. The same configuration was applied to different objective function experiment in order to have the test under the same experimental condition and make the results comparable.

From numerical results of the table 3.2 and figure 3.5 - 3.12, Point Mean Relative Error (PMRE) showed bad performance and is therefore ruled out. Even though Point Mean Absolute Error (PMAE) and Theil’s Inequality Coefficient (TIC) functions have acceptable performance, the objective function, Global Relative Error (GRE), obviously prevail over them in terms of its descriptive and representative attributes. Because the PMAE expresses error from a single point perspective instead of network-wide, it poorly describes the error for the whole network. The TIC is just a coefficient, so it also lacks good description for network-wide error. However, the GRE function is able to fully describe the network-wide error. So it has been employed in the following experiment and real case study.

<table>
<thead>
<tr>
<th>Objective Function</th>
<th>Mis-Fitness Value</th>
<th>Operation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-Converted</td>
<td>Converted to GRE</td>
</tr>
<tr>
<td></td>
<td>Turn Count Error</td>
<td>Turn Count Error</td>
</tr>
<tr>
<td>Point Mean Absolute Error (PMAE)</td>
<td>148.43</td>
<td>65.52%</td>
</tr>
<tr>
<td></td>
<td>2:17 ~ 7:40</td>
<td></td>
</tr>
<tr>
<td>Global Relative Error (GRE)</td>
<td>65.81%</td>
<td>65.81%</td>
</tr>
<tr>
<td></td>
<td>10:01 ~ 14:42</td>
<td></td>
</tr>
<tr>
<td>Point Mean Relative Error (PMRE)</td>
<td>895.86</td>
<td>86.67%</td>
</tr>
<tr>
<td></td>
<td>5:03 ~ 14:44</td>
<td></td>
</tr>
<tr>
<td>Theil’s Inequality Coefficient (TIC)</td>
<td>0.317</td>
<td>66.71%</td>
</tr>
<tr>
<td></td>
<td>14:39 ~ 19:17</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2 Experimental Comparison of Four Candidate Functions
3.3.2.2 Comparison of Four Types of GAs

The next four experiments will respectively examine four types of GAs that are coded in GENOSIM and see which GA is more robust, more efficient and more efficacy in solving the issue of combinatorial parametric optimization. Based on the performance of different GAs, one option will be applied to the real case study, e.g. Port Area, Toronto, traffic network-based model calibration.

In the experiments with different GAs, table 3.3 shows the run-time settings. In each experiment, the first 8 options in the table 3.3 were fixed on the same value in order to have the same experimental condition, each one of GAs is triggered to run one time.

<table>
<thead>
<tr>
<th>Run-Time Setting:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Generations</td>
<td>20</td>
</tr>
<tr>
<td>2. Population Size</td>
<td>30</td>
</tr>
<tr>
<td>3. Probability of Crossover</td>
<td>0.99</td>
</tr>
<tr>
<td>4. Probability of Mutation</td>
<td>0.05</td>
</tr>
<tr>
<td>5. Representation</td>
<td>Real Number</td>
</tr>
<tr>
<td>6. Crossover Option</td>
<td>Two-Point</td>
</tr>
<tr>
<td>7. Mutation Option</td>
<td>Gaussian</td>
</tr>
<tr>
<td>8. Parent Selection</td>
<td>Roulette Wheel</td>
</tr>
</tbody>
</table>

Table 3.3 Configuration for Different GAs Comparison

Figure 13 ~ 16 describe four types of GAs performance and convergence information. From these figures the Steady-State performance is better than the others. Table 3.4 shows the final optimized results using each one of four GAs. Table 3.4 also indicates that Steady-State GA is able to search better solution than other family members are. However, it is hard to say that one GA has absolute advantage over others. It is widely
accepted that GA is a problem-specific optimization technique and different GA shows different advantages in different searching environment. Based on these experimental results, Steady-State GA and Crowding GA are selected as two major algorithms to optimize model parameters in real case study.

1. Simple GA

![Figure 3.13 Convergence through Generations](image)

2. Steady State GA

![Figure 3.14 Convergence through Generations](image)
3. Crowding GA

![Figure 3.15 Convergence through Generations](image1)

4. Incremental GA

![Figure 3.16 Convergence through Generations](image2)

<table>
<thead>
<tr>
<th>Genetic Algorithms</th>
<th>Mis-Fitness (Using GRE Objective Function)</th>
<th>Operation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum</td>
<td>Mean</td>
</tr>
<tr>
<td>Simple GA</td>
<td>65.73%</td>
<td>67.86%</td>
</tr>
<tr>
<td>Steady-State GA</td>
<td>65.23%</td>
<td>66.08%</td>
</tr>
<tr>
<td>Crowding GA</td>
<td>65.44%</td>
<td>69.60%</td>
</tr>
<tr>
<td>Incremental GA</td>
<td>66.81%</td>
<td>68.58%</td>
</tr>
</tbody>
</table>

Table 3.4 Optimization Result Comparison Using Different GAs
3.3.2.3 Comparison of Two Representations

In GENOSIM design, two kinds of representation, real-valued and binary code, are implemented. In this section, we use similar methods mentioned above to experiment with two kinds of representations in two different GAs. See which representation could be recommended as a better option for the problem of combinatorial parameter optimization.

Table 3.5 shows the scenarios that were tested in two representations. Keep using the same configuration in first 7 items; switch representation option from real number to binary, from Steady-State GA to Crowding GA and have one run for each combinatorial option.

<table>
<thead>
<tr>
<th>Run-Time Setting:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Generations</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2. Population Size</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>3. Probability of Crossover</td>
<td>0.99</td>
<td></td>
</tr>
<tr>
<td>4. Probability of Mutation</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>5. Crossover Option</td>
<td>Two-Point</td>
<td></td>
</tr>
<tr>
<td>6. Mutation Option</td>
<td>Gaussian</td>
<td></td>
</tr>
<tr>
<td>7. Parent Selection</td>
<td>Roulette Wheel</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Representation</th>
<th>Mutation Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steady-State</td>
<td>Real Number</td>
<td>Gaussian</td>
</tr>
<tr>
<td></td>
<td>Binary 16 bits</td>
<td>Bit-Flip</td>
</tr>
<tr>
<td>Crowding</td>
<td>Real Number</td>
<td>Gaussian</td>
</tr>
<tr>
<td></td>
<td>Binary 16 bits</td>
<td>Bit-Flip</td>
</tr>
</tbody>
</table>

Table 3.5 GA Configuration for Different Representation Comparison

Table 3.6 shows that the real number option in Steady-State GA could be recommended as a better configuration for future use in terms of searching speed and optimized results.
<table>
<thead>
<tr>
<th>GAs</th>
<th>Representation</th>
<th>Mis-Fitness (Using GRE Objective Function)</th>
<th>Operation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Minimum</td>
<td>Mean</td>
</tr>
<tr>
<td>Steady-State</td>
<td>Real Number</td>
<td>65.23%</td>
<td>66.08%</td>
</tr>
<tr>
<td></td>
<td>Binary</td>
<td>65.91%</td>
<td>66.55%</td>
</tr>
<tr>
<td>Crowding</td>
<td>Real Number</td>
<td>65.44%</td>
<td>69.60%</td>
</tr>
<tr>
<td></td>
<td>Binary</td>
<td>66.37%</td>
<td>68.59%</td>
</tr>
</tbody>
</table>

Table 3.6 Comparison Result of Two Representations in Two Different GAs

3.3.3 Conclusion

In the following real case study, a good configuration for genetic algorithm application is recommended as shown in table 4.2 based on the previous experimental results. Actually, no configuration has absolute advantages. Different combinatorial options should be explored further. Table 4.2 configuration might be just suitable for this particular case.
Chapter 4 GENOSIM Application in Case Study

4.1 Network and Experimental Description

The road network employed for simulation parameter calibration using GENOSIM is called “Port Area” in downtown Toronto, Canada. The east-west bounds are between Jarvis and Woodbine Street, and north-south bounds are from Queen Street to Lake Shore Boulevard. This area also encompasses portions of the Gardiner Expressway and Don Valley Parkway. This road network has a significant and profound meaning for model calibration using GENOSIM, because it covers almost all-fundamental traffic elements from highway to surface road, from arterial to minors, from signalized intersection to uncontrolled intersection. The success of parameter optimization based on such complex network would illustrate the potential of GENOSIM as a scalable and generic optimization tool for microscopic simulation parameters.

Figure 4.1 shows the virtual replica of this road network coded in Paramics. In this simulation network, there are 1270 links, 109 junctions, and 470 nodes. The Origin-Destination (O/D) matrix is generated based on the Toronto Tomorrow Travel Survey in 1996, via traffic assignment using MME2. Approximately 15864 vehicles travel demand is released during morning peak hour from 8:00-9:00am. The survey vehicles consist of 85% cars and 15% light trucks. The one-hour turning counts at twenty key signalized intersections are employed for this design purpose.
4.2 Random Seed Selection

Paramics documents recommend users to run the models with different seed values to test the sensitivity of the model. We also attempt to determine a reasonable seed value as starting point for simulation. 10 Random numbers are picked up and put into model configuration file. Each random number has one simulation run.
Figure 4.2 shows that different seed value causes fluctuation of model output so that the Global Relative Error is different. Table 4.1 also shows the numerical changes in GRE by altering seed values. A good seed 580 is used in the real case study.

![Global Relative Error vs Random Seed](image)

**Figure 4.2 Random Seed Selection**

<table>
<thead>
<tr>
<th>Index</th>
<th>Random Seed</th>
<th>Global Relative Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5790768</td>
<td>73.21%</td>
</tr>
<tr>
<td>2</td>
<td>27662</td>
<td>70.65%</td>
</tr>
<tr>
<td>3</td>
<td>9638</td>
<td>69.99%</td>
</tr>
<tr>
<td>4</td>
<td>809760</td>
<td>68.94%</td>
</tr>
<tr>
<td>5</td>
<td>580</td>
<td>68.89%</td>
</tr>
<tr>
<td>6</td>
<td>37501</td>
<td>69.51%</td>
</tr>
<tr>
<td>7</td>
<td>62</td>
<td>70.12%</td>
</tr>
<tr>
<td>8</td>
<td>6116</td>
<td>70.92%</td>
</tr>
<tr>
<td>9</td>
<td>198</td>
<td>71.92%</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>73.37%</td>
</tr>
</tbody>
</table>

**Table 4.1 GRE Fluctuation by Using Different Seed Value**
4.3 GAs Configuration:

The control parameters for the GA, such as crossover rate, mutation rate, population size, number of generations, are critical to solution quality. Especially, sufficient but not excessive chromosomes should be initialized to fill the population. Genetic diversity in the GA process is particularly important when the solution space is topographically rugged or convoluted [1]. If the population is too small and sparsely spread, then the lack of genetic diversity may lead to quick convergence on local optima before the better optima may be visited. On the other hand, excessively large populations cause the GA to act like a random search algorithm. The search may flounder. Table 4.2 shows the recommended configuration for the case study.

In this case study, the search space is five-dimensional. The searching range of each parameter is decided either by rules of thumb or from the highway capacity manual: mean headway: 0.5~1.5sec, mean reaction time: 0.4~1.6sec, feedback: 1~5min, perturbation: 1~100%, familiarity: 1~100%.

<table>
<thead>
<tr>
<th>Run-Time Setting:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generations</td>
</tr>
<tr>
<td>Population Size</td>
</tr>
<tr>
<td>Probability of Crossover</td>
</tr>
<tr>
<td>Probability of Mutation</td>
</tr>
<tr>
<td>Representation</td>
</tr>
<tr>
<td>Crossover Option</td>
</tr>
<tr>
<td>Mutation Option</td>
</tr>
<tr>
<td>Algorithm Option</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Parent Selection</td>
</tr>
</tbody>
</table>

Table 4.2 GA Run-Time Configuration in Real Case Study
4.4 Results

Figure 4.3 and Figure 4.4 show the process of convergence between the mean misfit and the best misfit (minimum) in each generation as the optimization progresses. The Steady-State GA still performed better than Crowding GA in this case.

4.4.1 Performance of Steady-State GA

![Graph showing convergence through generations for Steady-State GA](image1)

Figure 4.3. Convergence through Generations

4.4.2 Performance of Crowding GA

![Graph showing convergence through generations for Crowding GA](image2)

Figure 4.4. Convergence through Generations
Table 4.3 describes the final results of GA evolution in the 20th generation. The best score means the minimum absolute error between observed data and model output.

<table>
<thead>
<tr>
<th>Genetic Algorithms</th>
<th>Steady-State</th>
<th>Crowding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best (Minimum)</td>
<td>59.26</td>
<td>59.98</td>
</tr>
<tr>
<td>Worst (Maximum)</td>
<td>59.43</td>
<td>82.65</td>
</tr>
<tr>
<td>Mean</td>
<td>59.27</td>
<td>65.27</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.03</td>
<td>6.14</td>
</tr>
<tr>
<td>Variance</td>
<td>0.0009</td>
<td>37.72</td>
</tr>
<tr>
<td>Summation</td>
<td>1778.02</td>
<td>1957.98</td>
</tr>
</tbody>
</table>

Table 4.3 Results of GA Evolution

Table 4.4 shows the optimal values of calibrated parameters. While table 4.5 shows a performance comparison between steady-state GA and crowding GA in real case study. In table 4.4, there are two sets of calibrated parameter values, but it is hard to say which set is global optimum. However, both of them at least are local or near-global optimum. Even though one algorithm is used to run more than two times, the calibrated values are still slightly different from one another due to random attribute. It is believed that global optimum can be obtained when searching process is run long enough time and premature convergence is prevented.

<table>
<thead>
<tr>
<th>Genetic Algorithms</th>
<th>Steady-State</th>
<th>Crowding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Headway</td>
<td>1.22</td>
<td>0.83</td>
</tr>
<tr>
<td>Mean Reaction Time</td>
<td>1.16</td>
<td>0.66</td>
</tr>
<tr>
<td>Feedback</td>
<td>180</td>
<td>240</td>
</tr>
<tr>
<td>Familiarity</td>
<td>68.74</td>
<td>66.6</td>
</tr>
<tr>
<td>Perturbation</td>
<td>12.33</td>
<td>16.33</td>
</tr>
</tbody>
</table>

Table 4.4 Optimal Value of Calibrated Parameters
Figure 4.5 and 4.6 show the link count comparison and turning count comparison respectively. Using steady-state GA, we obtained one set of optimal values for model parameters. And use these optimal values as model starting configuration to run simulation. Then compare modeled link count, turning count with observed data by using GRE objective function.

Steady-State GA Output Comparison

<table>
<thead>
<tr>
<th>Genetic Algorithms</th>
<th>Turn Count Error</th>
<th>Link Count Error</th>
<th>Operation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steady-State</td>
<td>59.26%</td>
<td>46.89%</td>
<td>26.85 hrs</td>
</tr>
<tr>
<td>Crowding</td>
<td>59.98%</td>
<td>46.09%</td>
<td>36.5 hrs</td>
</tr>
<tr>
<td>Default Simulation</td>
<td>62.41%</td>
<td>51.04%</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5 GA Performance Comparison

Figure 4.5. Comparison of Link Count Using Steady-State GA
Figure 4.6 Comparison of Turn Counts Using Steady-State GA

Figure 4.7 and 4.8 are similar to Figure 4.5 and 4.6, which show the link count comparison and turning count comparison based on crowding GA optimization values.

Crowding GA Output Comparison

Figure 4.7 Comparison of Link Counts Using Crowding GA
Figure 4.8 Comparison of Turn Counts Using Crowding GA
Chapter 5 Conclusion

5.1 Summary of Research

GENOSIM is an attempt to apply state of the art combinatorial parametric optimization techniques to the tedious process of calibrating traffic micro-simulation models. GENOSIM's detailed design offers opportunity to examine a variety of GAs with varying efficiency and robustness. Future research will focus on applying GENOSIM to citywide simulation model.

5.2 GENOSIM Contribution and Characteristics

The GENOSIM contributes to the state of the art in combinatorial parametric optimization for traffic micro-simulation models in two aspects: integrating genetic algorithms with microscopic traffic simulation models; and automating model calibration instead of manual operation. GENOSIM has two distinguished characteristics. One is scalability. It can search the optimal value for traffic model parameters based on any size road network. The other one is that GENOSIM is potentially getting to be a generic optimizing tool, because it does not only externally link to traffic micro-simulation model-Paramics, but also could link to other models as long as the parameters of models are accessible, thereby calibrating or optimizing those parameters.
5.3 Future Research Direction

New genetic algorithms have been developed as time goes by. In GENOSIM there are four types of GAs which have been implemented. That is just a basic framework that needs to be expanded further by adding new algorithmic features. On the other hands more parameters need to be added into calibration pool to improve the simulation quality.
Reference


Parameter Optimization for Paramics Using a Genetic Algorithm, TRB 2001


Appendix A — Glossary

Chromosome Also called a string, a sequence of characters or bits that represents an individual in the GA population

Crossover A key operator used in the GA to create new individuals by combining portions of two parent strings

Crossover probability Probability of performing a crossover operation, i.e., the ratio of the number of offspring produced in each generation to the population size

Evaluation The computation performed to determine the quality of an individual in the GA population

Evaluation function Also called fitness function, the function used to compute a fitness value to assign to each member of the GA population

Exploitation The process of using information gathered from previously visited points in the search space to determine which places might be profitable to visit next

Exploration The process of visiting entirely new regions of a search space to determine if anything promising may be found; while exploitation concentrates on previously visited points to maximize the gain, exploration attempts to find new regions in the search space where the gain can be high

Fitness Number assigned to an individual in the GA population to represent the quality of the corresponding solution
Gene The smallest unit in a genome. In a binary string genome, the bits are genes. In an array of characters, each character in the array is a gene.

Generation The process of creating a new population from an existing population through reproduction

Genetic Algorithm An evolutionary algorithm in which a population of individuals is evolved using selection, crossover, and mutation. Originally devised as a model of evolutionary principles found in Nature, genetic algorithms have evolved into a stochastic, heuristic search method. A genetic algorithm may operate on any data type with operators specific to the data type.

Mutation A genetic operator that modifies the genetic material of an individual.

Mutation probability The probability of mutating each gene in a GA chromosome

Offspring An individual resulting from crossover of two parent individuals and possible mutation

Parent An individual selected from the GA population for reproduction in the next generation

Population A collection of character or bit strings (chromosomes) that are candidate solutions to a given problem

Population size The number of individuals in the GA population

Selection A genetic operator used to choose individuals for reproduction