View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by NSU Works

Nova Southeastern University
NSUWorks

CEC Theses and Dissertations | College of Engineering and Computing

2015

A Heuristic Evolutionary Method for the Complementary Cell Suppression Problem

Hira B. Herrington
Nova Southeastern University, hiraherr@nova.edu

This document is a product of extensive research conducted at the Nova Southeastern University College of Engineering and Computing. For more information on research and degree programs at the NSU College of Engineering and Computing, please click here.

Follow this and additional works at: http://nsuworks.nova.edu/gscis_etd

Part of the Artificial Intelligence and Robotics Commons, and the Theory and Algorithms Commons

Share Feedback About This Item

NSUWorks Citation

Hira B. Herrington. 2015. *A Heuristic Evolutionary Method for the Complementary Cell Suppression Problem.* Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, Graduate School of Computer and Information Sciences. (28) http://nsuworks.nova.edu/gscis_etd/28.

This Dissertation is brought to you by the College of Engineering and Computing at NSUWorks. It has been accepted for inclusion in CEC Theses and Dissertations by an authorized administrator of NSUWorks. For more information, please contact nsuworks@nova.edu.

A Heuristic Evolutionary Method for the Complementary Cell Suppression Problem

by

Hira B. Herrington (hiraherr@nova.edu)

A dissertation report submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Information Systems

Graduate School of Computer and Information Sciences
Nova Southeastern University

2014

We hereby certify that this dissertation, submitted by Hira Herrington, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Sumitra Mukherjee, Ph.D. Chairperson of Dissertation Committee	Date
Michael J. Laszlo, Ph.D. Dissertation Committee Member	Date
Francisco J. Mitropoulos, Ph.D. Dissertation Committee Member	Date

Approved:

Eric S. Ackerman, Ph.D. Dean, Graduate School of Computer and Information Sciences	Date

Graduate School of Computer and Information Sciences
Nova Southeastern University

2015

SIGNED ORIGINAL ON FILE AT NSU

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirement for the Degree of Doctor of Philosophy

A Heuristic Evolutionary Method for the Complementary Cell Suppression Problem

by
Hira B. Herrington
December 2014

Cell suppression is a common method for disclosure avoidance used to protect sensitive information in two-dimensional tables where row and column totals are published along with non-sensitive data. In tables with only positive cell values, cell suppression has been demonstrated to be non-deterministic NP-hard. Therefore, finding more efficient methods for producing low-cost solutions is an area of active research.

Genetic algorithms (GA) have shown to be effective in finding good solutions to the cell suppression problem. However, these methods have the shortcoming that they tend to produce a large proportion of infeasible solutions. The primary goal of this research was to develop a GA that produced low-cost solutions with fewer infeasible solutions created at each generation than previous methods without introducing excessive CPU runtime costs.

This research involved developing a GA that produces low-cost solutions with fewer infeasible solutions produced at each generation; and implementing selection and replacement operations that maintained genetic diversity during the evolution process. The GA's performance was tested using tables containing 10,000 and 100,000 cells. The primary criterion for the evaluation of effectiveness of the GA was total cost of the complementary suppressions and the CPU runtime.

Experimental results indicate that the GA-based method developed in this dissertation produced better quality solutions than those produced by extant heuristics. Because existing heuristics are very effective, this GA-based method was able to surpass them only modestly.

Existing evolutionary methods have also been used to improve upon the quality of solutions produced by heuristics. Experimental results show that the GA-based method developed in this dissertation is computationally more efficient than GA-based methods proposed in the literature. This is attributed to the fact that the specialized genetic operators designed in this study produce fewer infeasible solutions.

The results of these experiments suggest the need for continued research into non-probabilistic methods to seed the initial populations, selection and replacement strategies that factor in genetic diversity on the level of the circuits protecting sensitive cells; solution-preserving crossover and mutation operators; and the use of cost benefit ratios to determine program termination.

Acknowledgements

I would like to thank my dissertation committee for their time, comments and support. I would especially like to thank Dr. Mukherjee for agreeing to be my advisor, his timely advice, and his limitless patience and understanding. Additionally, I would also like to thank Dr. Mitropoulos and Dr. Laszlo for serving on the committee and taking the time to review my work and provide their insight.

I would like to thank my wife Cathy for her understanding and patience during this process. Her commitment to research and teaching reminds me why I started down this road to begin with. Lastly, I would like to thank my parents for instilling in me the desire to continue learning and challenging myself.

Table of Contents

Abstract ... iii

List of Tables ... vii

List of Figures ... viii

1. Introduction ... 10
- Introduction ... 10
- Problem Statement ... 12
- Dissertation Goals ... 13
- Relevance and Significance ... 14
- Barriers and Issues ... 15
- Elements, Hypothesis, and Research Questions ... 16
- Limitations and Delimitations ... 16
- Definition of Terms ... 17
- Summary ... 19

2. Review of Literature ... 20
- Introduction ... 20
- Network Flow Approaches ... 21
- Heuristics Approaches ... 24
- Genetic Algorithm ... 26
- Selection and Replacement Strategies ... 28
- Portfolio of Selection and Replace Policies ... 29
- Evaluation ... 31
- Summary of Research ... 32
- Research Contributions ... 34

3. Methodology ... 35
- Introduction ... 35
- Genetic Algorithm ... 36
- Chromosomal Representation ... 37
- Solution Checking Functions ... 37
- Initial Population Generation ... 38
- Crossover ... 40
- Mutation ... 42
- Selection and Replacement ... 43
- Termination ... 46
- Evaluation of the Results ... 46
- Format of Results ... 48
- Resources ... 48
- Preliminary Testing ... 48
- Preliminary Tests Conclusions ... 53

4. Results ... 54
- Introduction ... 54
- Discussion of Test Results for Tables with 10,000 Cells ... 55
- Test Results for Tables with 10,000 Cells and 0.5% Sensitive Cells ... 55

Test Results for Tables with 10,000 Cells and 1% Sensitive Cells....57
Test Results for Tables with 10,000 Cells and 3% Sensitive Cells....60
Discussion of Test Results for Tables with 100,000 Cells....62
Test Results for Tables with 100,000 Cells and 0.25% Sensitive Cells....62
Test Results for Tables with 100,000 Cells and 0.5% Sensitive Cells....65
Test Results for Tables with 100,000 Cells and 1% Sensitive Cells....67
Test Results for Tables with 100,000 Cells and 3% Sensitive Cells....70
Summary....72

5. Conclusions, Implications, Recommendations and Summary....74
Conclusions....74
Implications....78
Recommendations....82
Summary....83
About Appendix....88

Appendix A: Sample Output....89

Reference List....90

List of Tables

Table 1: DataSets 47
Table 2: Preliminary Testing of Selection / Replacement Strategies for with 10,000 Cells with 0.5% Sensitive Cells 49
Table 3: Preliminary Testing of Selection / Replacement Strategies for 10,000 Cells with 1% Sensitive Cells 51
Table 4: Preliminary Testing of Selection / Replacement Strategies for 10,000 Cells with 3% Sensitive Cells 52
Table 5: Comparison of Average Solution Costs with 10,000 Cells with 0.5% Sensitive Cells 55
Table 6: Comparison of Average CPU Run Times (in seconds) with 10,000 Cells with 0.5% Sensitive Cells 56
Table 7: Comparison of Average Solution Costs with 10,000 Cells with 1% Sensitive Cells 58
Table 8: Comparison of Average CPU Run Times (in seconds) with 10,000 Cells with 1% Sensitive Cells 59
Table 9: Comparison of Average Solution Costs with 10,000 Cells with 3% Sensitive Cells 60
Table 10: Comparison of Average CPU Run Times (in seconds) with 10,000 Cells with 3% Sensitive Cells 61
Table 11: Comparison of Average Solution Costs with 100,000 Cells with 0.25% Sensitive Cells........ 63
Table 12: Comparison of Average CPU Times with 100,000 Cells with 0.25% Sensitive Cells 64
Table 13: Comparison of Average Solution Costs with 100,000 Cells with 0.5% Sensitive Cells 65
Table 14: Comparison of Average CPU Times (in seconds) with 100,000 Cells with 0.5% Sensitive Cells........ 66
Table 15: Comparison of Average Solution Costs with 100,000 Cells with 1% Sensitive Cells 68
Table 16: Comparison of Average CPU Times (in seconds) with 100,000 Cells with 1% Sensitive Cells........ 69
Table 17: Comparison of Average Solution Costs with 100,000 Cells with 3% Sensitive Cells 70
Table 18: Comparison of Average CPU Times (in seconds) with 100,000 Cells with 3% Sensitive Cells........ 71
Table 19: Comparison of Improvement Ratios in Tables of 10,000 Cells 73
Table 20: Comparison of Improvement Ratios in Tables of 100,000 Cells 73

List of Figures

Figure 1: Genetic Algorithm Overview 36
Figure 2: Shortest Path Heuristic for creating Rectangle of Suppressed Cells 38
Figure 3: Procedure to Generate Chromosomes 39
Figure 4: Procedure crossoverCircuits 41
Figure 5: Procedure mutateOffspring 43
Figure 6: Selection and Replacement Process 44
Figure 7: Comparison of Average Costs per Strategy for each Dataset with 10,000 Cells with 0.5% Sensitive Cells 50
Figure 8: Comparison of Average Costs per Strategy for each Dataset with 10,000 Cells with 1% Sensitive Cells 51
Figure 9: Comparison of Average Costs per Strategy for each Dataset with 10,000 Cells with 3% Sensitive Cells 52
Figure 10: Comparison of Average Solution Costs (Y-Axis) with 10,000 Cells with 0.5% Sensitive Cells 56
Figure 11: Comparison of Average CPU Run Times (Y-Axis in seconds) with 10,000 Cells with 0.5% Sensitive Cells 57
Figure 12: Comparison of Average Solution Costs (Y-Axis) with 10,000 Cells with 1% Sensitive Cells 58
Figure 13: Comparison of Average CPU Run Times (Y-Axis in seconds) with 10,000 Cells with 1% Sensitive Cells 59
Figure 14: Comparison of Average Solution Costs (Y-Axis) with 10,000 Cells with 3% Sensitive Cells 61
Figure 15: Comparison of Average CPU Run Times (Y-Axis in seconds) with 10,000 Cells with 3% Sensitive Cells 62
Figure 16: Comparison of Average Solution Costs (Y-Axis) with 100,000 Cells with 0.25% Sensitive Cells 63
Figure 17: Comparison of Average CPU Times (Y-Axis in seconds) with 100,000 Cells with 0.25% Sensitive Cells 64
Figure 18: Comparison of Average Solution Costs (Y-Axis) with 100,000 Cells with 0.5% Sensitive Cells 66
Figure 19: Comparison of Average CPU Times (Y-Axis in seconds), Using a Single Dataset, with 100,000 Cells with 0.5% Sensitive Cells 67
Figure 20: Comparison of Average Solution Costs (Y-Axis) with 100,000 Cells with 1% Sensitive Cells 68
Figure 21: Comparison of Average CPU Times (Y-Axis in seconds), Using a Single Dataset, with 100,000 Cells with 1% Sensitive Cells 69
Figure 22: Comparison of Average Solution Costs (Y-Axis) with 100,000 Cells with 3% Sensitive Cells 71

Figure 23: Comparison of Average CPU Times (Y-Axis in seconds) with 100,000 Cells with 3% Sensitive Cells 72
Figure 24: Comparison of Improvement Ratios (Y-Axis) at 100,000 Cells at Different Sensitive Cell Percentages (X-Axis)........ 87
Figure 25: Sample HeurGene Summary Output........ 89
Figure 26: Sample HeurGene Summary Data Output 89

Chapter 1

Introduction

Introduction

Cell suppression can be defined as a method of Statistical Disclosure Control in which the data in a two-dimensional statistical table considered sensitive are blocked from publication by suppressing their value. Cell suppression is typically accomplished by setting the value of the sensitive cell to null to conceal its information before the table is released (Fischetti & Salazar, 1998). However, suppressing the sensitive cells alone is not sufficient as their values can be derived from the remaining values due to marginal row and column totals present in the table. It is therefore necessary to suppress additional non-sensitive cells, called complementary suppressions, to guarantee that the values of the sensitive cells cannot be calculated within a predetermined disclosure interval. The goal is to minimize the information lost by suppressing non-sensitive cells while protecting all sensitive cells (de Carvalho, Dellaert, & de Sanches Osorio, 1994; Fischetti & Salazar, 1998).

The two-dimensional table needing protection can be represented as $A = [a_{ij}]$, where A is defined as a $(m + 1) \times (n + 1)$ matrix of real numbers a_{ij}. The values in the m rows and n columns of the table are known as internal cells, while the values in the $(m + 1)$ row are the column subtotals $a_{m+1,j} = \sum_{i=1}^{m} a_{ij}, j = 1, \dots, n$, and values in the $(n + 1)$ column are the row subtotals $a_{i,n+1} = \sum_{j=1}^{n} a_{ij}, i = 1, \dots, m$. The value at $a_{m+1,n+1}$ is the grand total $a_{m+1,n+1} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}$ (Almeida, Schütz, & Carvalho, 2006; Kelly,

Golden, & Assad, 1992). A cell in a table is denoted by (i, j) where i is the row location and j is the column location in table T such that $T = \{(i, j) \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ (de Carvalho et al., 1994). A primary suppression is a sensitive cell suppressed from publication. The set of primary suppressions $(i, j) \in S_1$ is a subset $S_1 \subseteq T$. S_1 is protected by lower and upper bounds l_{ij} and u_{ij} respectively, with a protection interval defined as $P_{ij} = [a_{ij} - l_{ij}, a_{ij} + u_{ij}]$ (Fischetti & Salazar, 1998; Almeida & Carvalho, 2005). The set of complementary suppressions is denoted by $S_2 = \{(i, j) \in A\}$ (de Carvalho et al., 1994).

The upper level protection for each cell in S_1 is defined as u_{ij}, with $u_{ij} \geq 0$. The lower level protection for each cell in S_1, is denoted by l_{ij}, where $l_{ij} > 0$. A confidential cell is right-protected if the smallest range an intruder is able to compute for a_{ij} contains $a_{ij} + u_{ij}$. The cell is left-protected if the computable range for a_{ij} contains $a_{ij} - l_{ij}$. A sensitive cell is considered protected if it is both left and right protected according to a range defined by the cell's protection interval.

A table is considered safe if each sensitive cell in S_1 is both right and left protected. S_2 is considered feasible if all cells in S_1 get protected when the values $S_1 \cup S_2$ are omitted from the table or set to null. Each cell in S_1 is assigned a weight of zero and each cell in S_2 is given a non-negative weight $w_{ij} = |a_{ij}|$ reflecting the loss of information due to suppression of non-sensitive cells. The cost of the complementary suppressions can be expressed as: $\sum_{(i,j) \in S_2} w_{ij}$ (Almeida et al., 2006). The *Cell Suppression Problem* (CSP) can now be defined as:

> Given a set S_1 of sensitive cells, with protection interval P, the CSP searches for the lowest cost set S_2, that minimizes information loss, where all cells in S_1 are considered safe (Kelly et al., 1992).

Kelly et al. (1992) demonstrated that the CSP is NP-hard, giving rise to the

development of a number of heuristic solutions that provide for a low-cost set S_2. A general table containing both postitive and negative entries yields a CSP that corresponds to an undirected bipartite network $G = (V, E)$ where $V = R \cup C$ is the set of nodes formed by the union of set R of $m + 1$ nodes, which represent the table's rows, and the set C of $n + 1$ nodes representing the table's columns. E is a set of edges representing a table's cell $(i, j) \in E$ corresponding to a subset of the set T, each with weight w_{ij}. Given set S_1 of primary suppressions and set S_2 of complementary suppressions, $G_{S_1 \cup S_2} = (V, E_{S_1 \cup S_2})$ represents the subgraphs of suppressed cells. A general table A with S_1, $S_2 \subset T$ is safe in the corresponding subgraph $G_{S_1 \cup S_2}$ if every suppressed cell belongs to a circuit (de Carvalho et al., 1994). With respect to positive tables, the solution of the CSP is additionally dependent on the cell values in A and the protection interval P_{ij} on sensitive cells.

The goal of the CSP can be expressed as finding a lowest cost set for S_2 where all cells in S_1 are protected. Genetic algorithms (GA) have been shown to be useful in finding low-cost solutions, but have the tendency to produce large numbers of infeasible solutions during the evolutionary process (Ditrich, 2010). This is due to the random nature of the GA's crossover operation, which combines selected pairs of existing solutions and the mutation operation, which disturbs existing solutions (Almeida et al., 2006). The application of a function to direct the crossover and mutation operations would allow existing circuits of suppression not involved in the operation to be preserved.

Problem Statement

Almeida et al., (2006) developed a hybrid heuristic and GA approach to the CSP

that demonstrated the feasibility of the approach, but were hampered by the random nature of the GA's mutation and crossover operators, which tend to produce infeasible solutions requiring repair or replacement at each generation. Ditrich (2010) improved upon this process and developed repair and replacement operations to help compensate for infeasible solutions, but at high computational cost. Therefore, the development of genetic operators that are able to produce feasible solutions requiring little or no repair or replacement at each generation is an area that warrants further investigation.

This research developed and evaluated new crossover and mutation operations for a GA in order to reduce the need for repair and replacement of infeasible solutions and improve the quality of the final solution. Additionally, a portfolio selection of chromosomal selection and replacement strategies was examined in order to provide good genetic diversity at each generation and mitigate premature convergence of the GA. The algorithm presented in the research will be referred to as *HeurGene* for *Heuristic-Genetic* algorithm.

Dissertation Goals

The primary goal of the proposed research was to develop an improved GA for the CSP that generated low-cost solutions without introducing excessive additional CPU overhead. To achieve this objective, the following primary goals needed to be realized:

1. the development of crossover and mutation operators that improved upon existing methods, and
2. the development of selection and replacement strategies that provided sufficient chromosomal diversity at each generation to avoid premature convergence.

In support of the primary goals, the following secondary goal needed to be

realized:

3. the development of a chromosomal representation to facilitate objective 1.

The GA was evaluated using techniques similar to those used by Almeida et al. (2006) and Ditrich (2010). The primary goals were realized when the following requirements were met:

1. the production of fewer infeasible solutions at each generation than previous methods,
2. the production of lower-cost solutions than previous methods, and
3. the accomplishment of goals 1 and 2 without introducing significantly increased runtime costs.

Relevance and Significance

Balancing the requirement of privacy and the need to release data in two-way tables for legitimate analysis often requires that sensitive data be suppressed. Unfortunately, in tables that contain totals in marginal rows and columns, it is possible to estimate the values of the suppressed data using linear programming. Disclosure has been compromised when the sensitive cell's estimated maximum or minimum values fall inside a given range as determined by the entity releasing the tables for analysis. This requires that additional data be suppressed to prevent the calculation of the sensitive data (Salazar-González, 2008; de Carvalho et al., 1994). In response, organizations such as the U.S. Department of Commerce have set requirements for unauthorized disclosures of sensitive data (Lu & Li, 2008). International organizations such as the United Nations Economic Commission for Europe and Eurostat have hosted special sessions to address the issue and agencies such as European Union and US National Science Foundation have supported research projects in the area (Salazar-González, 2008).

The use of cell suppression for statistical disclosure of sensitive data is one of the

most commonly used methods (Almeida & Carvalho, 2005; Fischetti & Salazar, 1998). It has been shown that GAs can quickly produce low-cost solutions to the CSP (Kratica, Čangalović, & Kovačević-Vijčić, 2009; Ditrich, 2010). Unless carefully designed, the application of GAs to the CSP is problematic in that existing cycles of suppressions are often lost during the mutation and crossover operations, requiring repair or replacement of offspring (Krasnogor & Smith, 2005). As a result, these operators are of particular interest to researchers working with GAs as applied to the CSP.

This research exploited the specific attributes of the CSP in order to improve upon crossover and mutation operators used in previous research. These operators made use of existing circuits of suppressions to direct their processes. A chromosomal representation and supporting external data structures were developed to facilitate the process. A secondary goal of the research was an exploration of selection and replacement strategies to improve upon genetic diversity. The outcome of this research was a more effective method for securing sensitive data while maintaining a high ratio of solution improvement over CPU time.

Barriers and Issues

Given the use of a solution-preserving crossover operation, along with a heuristic to direct the crossover and mutation operations, it was expected that the GA would quickly converge on a good quality solution with fewer infeasible solutions produced at each generation. However, this assumption was wrong for two reasons. First, the use of a heuristic on the crossover operations disturbed the random nature of the process, which limited the GA’s ability to fully explore the solution space and resulted in the GA prematurely converging regardless of the selection and replacement policies. This was

because the heuristic forced the chromosomal representation into a fixed number of configurations in which diversity was limited, causing localized convergence. Second, removing the randomness from the mutation process necessarily required that the chromosomal sequences needed for a near optimal solution already be present in the current population. This is a result of the inability to randomly create novel sequences through the mutation process.

Elements, Hypothesis, and Research Questions

The research presented in this report hypothesized that a heuristically-directed genetic algorithm could be developed that would find good solutions without the need for repair or replacement of infeasible solutions at each generation. Specific questions that this research answered included:

1. Can crossover and mutation operations be designed that produce few or no infeasible solutions?
2. Will this method provide for improvement in the cost of the solutions?
3. Will a portfolio of deterministic and probabilistic selection and replacement rules maintain sufficient genetic diversity to avoid premature convergence?
4. Does the computational overhead associated with the genetic algorithm negate its benefits?

Limitations and Delimitations

The Microsoft Visual C++ compiler, in order to handle datasets larger than 100,000 values, needed to be set to LARGEADDRESSAWARE, which had a significant negative impact on run time and memory usage.

Synthetic datasets tend to have cell values and sensitive cell locations evenly

distributed. Real world datasets may have data clustered around certain values and locations within the table. This may negatively skew the runtimes as clustering may alter the operational characteristics of the heuristics for crossover and mutation.

When the percentage of sensitive cells present in a table becomes large, it becomes more probable that the suppression of sensitive data alone will provide sufficient protection for a large number of the sensitive cells. This has the effect of skewing results as the overall cost for protecting a small number of sensitive suppressions is necessarily low. However, it has the opposite effect on CPU time owing to the heuristic's attempt to locate a suitable circuit of protection for crossover.

Definition of Terms

The following is a listing of definitions for key terms used in this research report.

Term	Definition
Cell Suppression	Setting the value of the cell to null to conceal its information.
Chromosomal Representation	Set of structures containing data about the encoding of cells. It also serves to represent the individual parents and offspring.
Cell Suppression Problem (CSP)	Method of protecting sensitive information from disclosure in statistical tables that minimizes information loss without altering the values of non-sensitive cells through the use of complementary suppressions.
Circuit of Protection	A set of suppressions forming a closed circuit protecting a sensitive cell.
Complementary Suppressions	Suppressed, non-sensitive cells that guarantee that the values of the sensitive cells cannot be calculated within a predetermined disclosure interval.
Cost	The sum of the values of the complementary suppressions.
Crossover	The method by which two individuals in the parent population exchange the genetic information present in their chromosomal representation.
Feasible Solution	A set of complementary suppressions that protects sensitive cells such that they are considered safe.
Genetic Algorithm (GA)	A heuristic search algorithm based on the biological process of reproduction.
Hypercube	A fast heuristic to find sets of complementary suppressions

	to protect confidential data. In two-dimensional tables, for each confidential cell, a set of three cells forming the corners of a rectangle is required.
Left Protection	When the lower bound for a sensitive cell is less than its lower protection level.
Linear Programming	A mathematical model for the optimization of an outcome based on a set of constraints.
Lower Bound	The smallest value that can be calculated for a suppressed sensitive cell.
Lower Protection Level	The value subtracted from a sensitive cell's value to establish the largest acceptable lower bound that can be calculated from the non-suppressed cells.
Mating Pair	Two individuals from the parent population selected for crossover and mutation operations.
Marginal Cell	A cell containing the sum total for a row, column or the grand total for the table.
Mutation	The method by which changes are made to the chromosomal representation of a single individual.
Offspring	The product of two parents that have undergone crossover and mutation.
Oversuppression	A suboptimal pattern of complementary suppressions used to protect sensitive cells.
Parent Population	The current population exclusive of offspring.
Population	Set of all individuals being acted upon by the genetic algorithm.
Primary Suppression	A sensitive cell whose value has been set to null to protect it from disclosure.
Protection Interval	The range of values lying in the interval defined by the lower and upper protection levels.
Replacement	The process of selecting offspring to succeed members of the parent population.
Right Protection	When the upper bound for a sensitive cell is greater than its upper protection level.
Selection	The process of choosing individuals from the parent population for crossover and mutation.
Sensitive Cell	Cells singled out as containing information that is inappropriate or too revealing for publication.
Sliding Protection	The distance between the upper and lower protection levels.
Statistical Disclosure Control	The process by which an entity provides protection to sensitive cells in statistical table.
Sub-Network	A sub-table consisting of a primary suppression and complementary suppressions forming a cycle that protects the primary suppression from disclosure.
Upper Bound	The largest value that can be calculated for a suppressed sensitive cell.

Upper Protection Level: The value added from a sensitive cell's value to establish the smallest acceptable upper bound that can be calculated from the non-suppressed cells.

Summary

Complementary cell suppression is a proven method for statistical disclosure control that maximizes the quality of released statistical tables. Genetic algorithms have proven to provide good quality solutions but are hampered by crossover and mutation operations that generate infeasible solutions, requiring repair or replacement. This research was premised on the theory that crossover and mutation operations could be developed that preserved feasible solutions between generations. This was achieved in part by heuristic algorithms that act on the crossover and mutation operations to direct the evolutionary process, along with a list of suppressions that allowed those suppressions shared between different circuits of protection to be factored into the operations. In addition, a portfolio of selection and replacement rules was developed to help maintain genetic diversity to avoid premature convergence at local optima.

Chapter 2

Review of Literature

Introduction

Cell suppression is a common method of statistical disclosure control used to protect sensitive information in a statistical table where non-sensitive data is released along with row and column totals (Fischetti & Salazar, 1998). Various other techniques have been applied to the CSP, including those based on network flow and various heuristic approaches. The different approaches typically offer a tradeoff between the quality of the solution and speed of execution. As problem sets become increasingly large the issue of finding a quality solution with minimal computational overhead becomes more acute. First developed by John Holland and based on processes of natural evolution, genetic algorithms have proven to be useful for quickly finding good but suboptimal solutions to large instances of optimization problems (Goldberg, 1989).

Genetic algorithms are typified by an initial parent population composed of chromosomal representations of a solution space and ranked by a fitness function, which allows for selection of most fit pairs for mating. Offspring are created through a process of crossover and mutation with the more fit individuals replacing the less fit members of the parent population according to the fitness function (Russell & Norvig, 2010). The process is repeated until a stopping condition is met. The evolutionary process takes advantage of the fitter individuals produced by the genetic operators and increases their relative frequency in the population such that they are more likely to reproduce, producing fitter offspring (Smith, 2007).

The initial parent population may be created randomly or by any method that provides good genetic diversity. The chromosomal representation reflects the solution space being explored with the most typical representation consisting of a binary array with one element for each value under evaluation. A fitness function is used to rank the population in order to insure that positive genetic characteristics are passed to future generations. Crossover is used to mate selected members of the parent population, producing offspring with potentially superior chromosomal makeups. The mutation operation ensures genetic diversity in the parent population to help prevent premature convergence at a suboptimal solution. A stopping condition terminates the process, usually when a specified number of generations have passed.

Network Flow Approaches

Network flow approaches, such as those described in Cox (1980) and Carvalho et al. (1994), were some of the earliest solutions to the CSP. The need for them arose out of the requirement to keep sensitive information from being estimated to within a value that is too close to the actual value as determined by the entity releasing the data. Cox suggests using algorithms that find a minimal set of suppressions based on cost. He discusses a number of methods for measuring the cost of suppression: (1) the evaluation of the sensitivity of the published aggregations not expressible as a function of other published aggregations, (2) the total value of suppressed cells, (3) the number of respondents in the suppression pattern, and (4) the total number of suppressed cells. Cox goes on to suggest that the total number of suppressed cells provides the best measure for cost and provides the greatest degree of process control for minimizing oversuppression.

Kelly et al. (1992) differentiate themselves from Cox by suggesting that the CSP

should be optimized as the sum of the suppressed cells rather than the count. They also point out that optimization based on the number of suppressions does not take into account cells with large values that may represent important data. They propose that a solution with a low sum can provide more usable data. They also introduce the concept of a sliding protection interval as a possible method to further reduce oversuppression. In a sliding protection interval, the width of the protection interval is what needs to be considered and not the upper and lower bounds protecting a sensitive cell. Their overall method examines one sensitive cell at a time to determine which complementary suppressions will be required. In order to compensate for oversuppression, Kelly et al.'s (1992) method additionally involves removing one complementary cell at a time and retesting the solution for feasibility. This process is repeated for each of the complementary suppressions. Smith, Clark, & Staggemeier (2009) note that ordering of the cells in Kelly et al.'s (1992) method has an effect on the cost of the suppressions and suggest using a GA to optimally order the cells prior to adding complementary suppressions.

Cox (1995), building on the work of Kelly et al. (1992) and de Carvalho et al. (1994), proposes using a mathematical network utilizing alternating cycles of arcs between rows and columns in the graph to form circuits. Circuits with only suppressed cells are considered safe from disclosure. Fischetti and Salazar (1999) further refine the process by defining properties for an optimal solution. One of the most basic is that any optimal solution will have no row or column with just one suppressed entry. Other properties include the bridge, comb, and cover inequalities. The bridge inequality provides that optimal solutions are bridgeless, where a network with a node of degree one

is not present in the solution. Likewise, comb inequalities will not be present in an optimal solution. Solutions that violate the knapsack constraint of cover inequalities are also precluded from being optimal.

De Carvalho, Dellaert, and Osorio's (1994) method utilizes sliding protection and minimizes the sum of suppressions as an objective function as proposed by Kelly et al. (1992), but differ in that they suggest the final solution should never include marginal totals. They use a network flow approach based on the theory that rows and columns of a two dimensional table can be modeled into a bipartite graph where the edges form a circuit protecting each primary suppression. However, the process of forming circuits often leads to oversuppression, requiring additional processing to reduce solution cost. De Carvalho et al. (1994) use Dijkstra's shortest path (SP) algorithm to reduce the cost by finding the shortest distance between two nodes and removing high cost edges. However, the runtime cost was high for medium to large graphs.

Another method of reducing computational cost is to reduce the size of the solution search space. Cox (1980) describes a combinatorial procedure for finding optimal solutions to the CSP by searching for intersecting rows and columns of the primary suppressions for candidates for complementary suppressions. Cox notes that these intersections represent the best locations for complementary suppressions due to their lower-cost, which results from being shared by two or more cycles protecting a primary suppression. Complementary suppressions located at other cells represent poor candidates and will be less likely to lead to an optimal solution. Once a partial solution is found, it may be used to identify other candidates for complementary suppression (Cox, 1995).

Heuristics Approaches

To reduce runtime costs, shortest path (SP) and Castro heuristics are commonly applied to the CSP. With the SP method, the confidential cells are protected one at a time without regard for order. A sequence of minimal cost paths for the sensitive cell's right and left-protection levels is built and the nonconfidential cells added to the solution. This process is repeated for all the sensitive cells with the cost of the solution equal to the sum of the values of the nonconfidential cells suppressed. Since the bounds are calculated only when all primary suppressions are protected, complementary suppressions added early in the process are not reevaluated in light of complementary suppressions made later in the process, leading to higher cost solutions (Kelly et al., 1992; Almeida et al., 2006).

In an effort to further reduce solution cost, Castro's method uses the SP algorithm, but recalculates the bounds between suppressions. The reevaluation process allows additional suppressions to be avoided based on their effect on the upper and lower bounds. In a method similar to Castro's, Kelly et al., (1992) use their SP heuristic to generate an initial solution. Once the initial cycles of suppressions is established by the SP heuristic, a cleanup phase iterates through each of the complementary suppressions, determining whether the table remains secure if a complementary suppression is removed.

A parallel bound and path heuristic was designed by Almeida et al. (2006) in order to quickly seed their GA with a diversified set of feasible solutions. Their approach uses a two phase scheme that delivers two solutions. The first phase generates a set of complementary suppressions based on the suppression needs in the rows and then on the

suppression needs in the columns. This two-step approach allows the suppressions added in the second step to take into account the complementary suppressions added in the first. Finally, a cleanup phase attempts to reduce oversuppression by removing complementary suppressions in rows or columns without confidential cells. Phase two applies a SP heuristic to the solutions from phase one to verify their feasibility and add any needed additional suppressions.

The Hypercube method, designed for *k*-dimensional general tables, developed at Landesamt für Datenverarbeitung und Statistik in Nordrhein-Westfalen, Germany, involves dividing the table into a set of sub-tables. The sub-tables are then protected in an iterative procedure that starts from the highest level. For each sensitive cell in the current sub-table, all possible cubes with suppressed cells at each of the corners are formed with the sensitive cell at one of the corners. A lower bound is then calculated for the width of the sensitive cell's suppression interval for each cube. If the calculated bound is sufficiently large, the cube is considered a feasible solution. The cube with the minimum information loss due to suppression is selected. Once all sub-tables have been protected, the process is repeated with the complementary cells belonging to more than one sub-table treated like sensitive cells. Since the hypercube method ensures that each suppressed cell must be part of a cycle protecting the sensitive cell, it has been shown to provide better solutions than other methods. Although this method provides a satisfactory protection pattern, it does necessarily provide the only possible pattern as it fails to reevaluate existing patterns as new patterns are added to the solution set. For this reason, the hypercube tends toward oversuppression as it fails to find patterns with the minimal overall amount of information loss (Giessing & Repsilber, 2002).

Genetic Algorithm

GA-based approaches have proven to provide low-cost suboptimal solutions with relatively low computational cost compared to SP based approaches. Almeida et al. (2006) were the first to apply a GA, called GenSup, to the CSP. They use two heuristic functions, shortest path and parallel bound and path, for the initial population. Their GA uses a crossover method involving the mating of an elite (25%) population with less fit individuals (75%) using a 1-point crossover. Each mating pair produces four new members during the process. This method uses a random position within the chromosome as a crossover point, which results in damage to existing sub-networks for cell protection. This requires the offspring undergo a repair process, which adds additional complementary cells as necessary to produce a feasible solution. A mutation operator is used that considers one cell at a time in the best parent and offspring and checks the resulting set with a SP heuristic that introduces added runtime costs. Chromosomes with the same cost value are disallowed to help maintain genetic diversity. The two most fit offspring then replace the two least from the parent population. This process repeats for a fixed number of total generations or until a fixed number of generations passes without improvement. Their method provides lower-cost solutions than the SP method and proves the validity of GAs application to the CSP.

Improvements to Almeida et al. (2006) GA were made by Smith et al. (2009) by utilizing tournament selection with Davis' permutation-specific recombination to help preserve information present in the parents. Ditrich (2010) also improved upon past solutions by implementing a single point crossover operator that uses the primary suppression set associated with other suppressed cells in the crossover to avoid treating

each cell independently, thereby preserving feasible solutions. Additionally, Ditrich (2010) implemented a mutation scheme that reduces solution cost by identifying oversuppressions and targeting them for mutation. This method has a large effect on the cost of solutions, especially in cases where fewer than four cells were required to provide protection.

Almeida et al. (2006) utilize a repair routine that is triggered when a possible solution is checked with a SP heuristic and returns other than a zero cost. GenSup, like SP and hypercube methods, does not consider sets of cells. GenSup's mutation operator could remove a cell from the solution with regard to its participation across multiple cycles protecting a sensitive cell. After each mutation, the SP heuristic checks the solution and rejects infeasible solutions. This results in a large computational cost as possible solutions are repaired or rejected after being checked and rechecked with the SP heuristic.

Ditrich's (2010) Genetic Algorithm - Protection Network (GAPN) method builds on Almeida et al. (2006) with several differences. The crossover operation randomly selects a primary suppression and attempts to identify the complementary suppressions in a simple rectangle that forms the protection network. More complex protection networks such as those found in higher quality solutions are not identified due to the interdependency of the cells. The cells identified during the process are placed in sets based on the primary cell they protected. If a set contains at least one complementary suppression, it is available for crossover. After crossover, the child solutions are validated and infeasible solutions rejected. The mutation operation is then performed on the best offspring. The mutation operator searches the child's chromosomal representation for

complementary suppressions that could be removed and still maintain a feasible solution. Population size is constant with the parent and child populations combined and lowest quality solutions removed.

Selection and Replacement Strategies

GAs can employ a variety of selection and replacement strategies in an effort to balance selective pressure with genetic diversity. Often GAs employ deterministic selection strategies that consist of elitist selection, where only the fittest members of the population are selected for mating, complemented by an elitist replacement strategy where the least fit members of the current population are replaced by more fit offspring (Mashohor, Evans, & Arslan, 2005; Smith, 2007).

Selection and replacement strategies can be closely coupled as with Modified Random Tournament Selection and Replacement. Here, selection and replacement are combined into a single strategy where n individuals from the current population are selected at random for both mating and replacement. The offspring then compete with these individuals for replacement based on fitness ranking (Smith et al., 2009; Razali & Geraghty, 2011). This method has been shown to maintain good diversity, but at the cost of increased convergence time (Razali & Geraghty, 2011).

Other selection and replacement strategies are independent from each other. The Fitness Proportional Roulette Wheel selection strategy involves individuals being selected for mating with a probability proportional to their fitness. The probability of any one individual being selected for mating is defined as $p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$ where f_i is the fitness values of individual i and p_i is the probability of individual i being selected. The advantage of this method is that all individuals in the current population have a chance of

being selected (Razali & Geraghty, 2011).

Similar to the Fitness Proportional Roulette Wheel strategy, Rank-Based Roulette Wheel Selection uses an individual's rank in the population rather than its fitness. Rank is defined as $p_i = \frac{r_i}{\sum_{j=1}^{n} r_j}$, where r_i is the rank of individual i in the current generation and p_i is the probability of individual i being selected. The advantage of this method is that it tends to avoid premature convergence and eliminates the need to scale fitness values, but it can be computationally expensive owing to the need to resort the parent population at each generation (Razali & Geraghty, 2011).

The Elitist replacement strategy requires that the least fit individuals in a population be replaced by more fit offspring (Smith, 2007). This strategy provides for a high level of selective pressure, resulting in an increased level of convergence (Lozano, Herrera, & Cano, 2005; Vavak & Fogarty, 1996).

The Replace Random strategy functions by randomly replacing a member of the current population at each generation. This method typically produces poor results due to the variance it introduces into the population (De Jong & Sarma, 1993).

The Kill Tournament strategy involves selecting members from the current population at random to compete with the offspring for a place in the next generation. Replacement is based on fitness with the number of parents selected for the tournament being subject to variability. This allows for a range of selective pressure. This strategy has the advantage of not requiring resorting of the population after replacement (Smith, 2007).

Portfolio of Selection and Replace Policies

One of the problems that arise when applying GAs to complex problems is that of

premature convergence to local optima. This can often be related to loss of genetic diversity in the evolving population. However, too much diversity can hurt the runtime performance of the GA (Galan & Mengshoel, 2010). The use of portfolios with GAs applied to the traveling salesperson problem has demonstrated that improved performance is possible as compared to methods that use a single approach (Fukunaga, 2000).

Developed in the discipline of economics, portfolio theory attempts to answer the question of how financial assets should be allocated in order to maximize expected returns and minimize risks. Huberman, Lukose and Hog (1997) were the first to suggest the use of what they call "computational portfolios" to solve hard computational problems. They demonstrate that an algorithmic portfolio can outperform the individual algorithms used in the portfolio (Fukunaga, 2000).

GAs often use deterministic crowding to identify the fittest members of the population for selection and replacement. Probabilistic crowding uses a probabilistic formula, which also uses fitness for selection and replacement. Deterministic crowding is an elitist replacement strategy that has a tendency toward premature convergence. Probabilistic crowding allows for exploration of the solution space for less fit individuals in order to improve genetic diversity. When both strategies are used, a balance must be achieved to provide the selective pressure needed for the GA to function efficiently (Galan & Mengshoel, 2010). While it may not be possible to develop a set of parameters to balance deterministic and probabilistic approaches for a general case, it should be possible to develop a set of parameters that can be tuned for a specific application of the GA (Fukunaga, 2000).

Mengshoel and Goldberg (2008) developed a portfolio of replacement rules where the rules are chosen based on an associated probability. The portfolio$\mathbb{R}$ is represented by a set *n* of 2-tuples: $\mathbb{R} = \{(p_1, R_1), \ldots, (p_n, R_n)\}$ where p_i is the probability associated with a given rule R_i being selected. In order to overcome the possibility of weak selection pressure related to using only a portfolio of probabilistic rules and too strong selection pressure associated with purely deterministic rules, deterministic (R_D) and probabilistic (R_P) approaches are combined into a portfolio $\mathbb{R} = \{(p_D, R_D), (p_P, R_P)\}$ where p is the probability of selecting either a deterministic or probabilistic approach. In this context, p_D and p_P can be used to control the GA's performance (Mengshoel & Goldberg, 2008). Using the sample means and covariance for the different strategies taken from successive executions of the GA, an object function can be used to tune the probabilities that drive the portfolio (Ewald, Schulz, & Uhrmacher, 2010).

Evaluation

Almeida et al. (2006) performed testing on two datasets: (1) Class I, which had tables with dimensions up to 100 x 100 and internal cells having random integer values in the range of [0, 499] with values in the range [1, 4] being confidential and (2) Class II, which had dimensions up to 300 x 300 with internal cell values ranging from [0, 1000] in value. Upper and lower protection levels were generated following the rules used in Fischetti & Salazar (1999).

Ditrich (2010) chose to use datasets containing 1000, 5000 and 10,000internal cells. As with Almeida et al. (2006), the values for internal cells were randomly assigned between 0 and 1000. Upper and lower protection levels were set equal to 15% of a given confidential cell's value rounded up to the nearest integer (Ditrich, 2010).

Assessment of the quality of the solutions produced by Almeida et al. (2006) was made using the following formula $gap = \frac{UB - LB}{LB} \times 100$, where LB is a lower bound value as computed by a heuristic solution and UB is the upper bound based on a GENSUP solution. The gap value represents the percentage difference between the two solutions. Their research on the 10,000 cells case had runtimes of two minutes or less and bridged over 70% of the optimality gap of the constructive heuristic solutions. However, GENSUP did not solve to proven optimality in 45 of 550 cases. For tables with greater than 20,000 cells, the percent of the optimality gap bridged increased to 85% (Almeida et al., 2006).

Evaluation of the performance requires assessment of the cost of the solution versus computational resources used. The cost can be based on the sum of the values of the complementary suppression or, as Cox (1980) suggested, by counting the number of complementary suppressions. The computational resources have two components: (1) is the computational time to reach a solution and (2) the memory required by the algorithm. Because data can be time-sensitive, an algorithm that takes too long to reach a solution runs the risk of protecting a table that has diminished value. If the memory requirements are excessive, a publishing entity may be required to make expenditures in hardware or choose to not use an otherwise useful method.

Summary of Research

Statistical disclosure limitation continues to be an area of active research in order to prevent personal data from accidental disclosure. Suppressions of sensitive data combined with complementary cell suppression represent one of the best methods to protect information while maintaining the quality of the data published. As technology

allows the quantity of data available for analysis to grow, the need to release larger statistical tables for publication drives the search for new methods to quickly and efficiently provide protection.

Early methods for finding a minimal set of suppressions use network-flow approaches to evaluate the conditions under which a sensitive cell is considered protected. These methods define the constraints necessary for a sensitive cell to be protected along with measurements of the cost and quality of cell suppression. These methods typically examine one sensitive cell at a time and build a system of sub-networks to protect each of the cells. However, these methods often produce low quality solutions due to oversuppression. The need to minimize oversuppression leads to the computationally costly post processing of solutions. Shortest path algorithms have been used to find low-cost solutions, but come with a high runtime costs, especially for large graphs.

The application of genetic algorithms improves upon existing methods by providing lower-cost solutions with relatively low computational cost. However, the nature of a GA's crossover and mutation operations tends to disturb existing solutions, requiring that offspring undergo repair or replacement, increasing runtime costs. Methods such as GenSup lack the ability to operate on sets of cells, which prevents locating lower-cost solutions. GAPN utilizes a crossover operation that examines sets of cells selecting a best candidate for crossover. The mutation operation tests and removes complementary suppressions and searches for lower-cost solutions in the current population. However, after both crossover and mutation, the solutions have to be checked for feasibility and rejected or repaired if infeasible. This is due to the complexity of the sub-network of cells

produced as a function of the GA's evolutionary process.

Research Contributions

This research attempted to provide an improved method for statistical disclosure control using cell suppression by improving on past methods and introducing new elements to allow GAs to work more efficiently. Building on GenSup and GAPN, this research improved upon past implementations by developing a heuristically controlled multi-region crossover that preserved feasible solutions during its operation. Additionally, a mutation operation was developed to remove redundant suppressions created by the crossover operation and combine the resultant circuits of protection to form larger more complex circuits. A portfolio of selection and replacement policies was developed to improve population diversity and selective pressure. The completion of this research demonstrated the feasibility of these approaches and may contribute to future research in the field.

Chapter 3

Methodology

Introduction

This chapter describes the approach to designing the GA and evaluating its performance. The goals of this research were to (1) develop procedures that use heuristics to improve the probability of lower-cost solutions resulting from crossover and mutation, (2) develop feasible solutions preserving crossover and mutation operations, and (3) explore the use of a portfolio of selection and replacement policies to mitigate premature convergence.

Given a table T, set S_1 of sensitive cells, and set S_2 of complementary suppressions, the following definitions apply:

$S = S_1 \cup S_2$,
$n_{i+}(S)$: Number of suppressed cells in row i,
$n_{+j}(S)$: Number of suppressed cells in column j,
$a_{i+}(S)$: Sum of the values of suppressed cells in row j,
$a_{+j}(S)$: Sum of the values of suppressed cells in column j,
$a_{rc}^{max}(S)$: Maximum value that a sensitive cell (r, c) can assume,
$a_{rc}^{min}(S)$: Minimum value that a sensitive cell (r, c) can assume

For a sensitive cell (r, c) in a table with a set of suppressions S, $a_{rc}^{max}(S)$ and $a_{rc}^{min}(S)$ can be defined as follows:

$a_{rc}^{max}(S) = \max x_{rc}$ such that:
$\sum_{j|(i,j)\in S} x_{ij} = a_{i+}(S)$ for $i = 1,2, \dots, m$, and
$\sum_{i|(i,j)\in S} x_{ij} = a_{+j}(S)$ for $j = 1,2, \dots, n$

and:

$a_{rc}^{min}(S) = \min x_{rc}$ such that:
$\sum_{j|(i,j)\in S} x_{ij} = a_{i+}(S)$ for $i = 1,2, \dots, m$, and
$\sum_{i|(i,j)\in S} x_{ij} = a_{+j}(S)$ for $j = 1,2, \dots, n$

A sensitive cell (r, c) is *unsafe* with respect to a set of suppressions S if any of the following four conditions hold true, and is deemed *safe* otherwise:

1) $n_{r+}(S) = 1$,
2) $n_{+c}(S) = 1$,
3) $a_{rc}^{max}(S) < (a_{rc} + u_{rc})$,
4) $a_{rc}^{min}(S) > (a_{rc} - l_{rc})$.

Table T is considered *safe* with respect to a set of suppressions S if every sensitive cell $(r, c) \in S_1$ is *safe* with respect to S. Each sensitive cell (r, c), is protected by a *protection circuit* $C(r, c)$ composed of a set of suppressed cells forming a circuit that renders (r, c) *safe*.

Genetic Algorithm

This research involved the development of a GA that uses solution-preserving crossover and mutation operations, a heuristic to modify the behavior of the crossover operation and a portfolio of selection and replacement rules to balance selective pressure with genetic diversity. Software used in this research was written in Microsoft C++. Development and testing was carried out using a cyclic, iterative and incremental development model until the research was completed. Additional detail is provided in the sections that follow. Figure 1 shows an overview of the genetic algorithm.

```
Load Statistical Table and Initialize GA Parameters
Create initialPopulation of size n as currentGeneration
while Not Termination Condition
    set nextGeneration to null
    for i = 1 to n/2
        Select a pair of parent chromosomes from current generation
        Apply crossover to pairs of parents to generate a pair of offspring
        Apply mutation to each offspring and add to nextGeneration
    Replace currentGeneration with nextGeneration
```

Figure 1: Genetic Algorithm Overview

The initial population consisted of ten individuals created as described in the section on *Initial Population Generation*. Once created, the initial population was assigned to the current generation. Each generation was created by first selecting parents from the current population according to the current selection rule. See the section on *Selection and Replacement* for more information on the parental selection process. A crossover operation was applied to the parents to create two offspring. Each of the offspring then underwent mutation and was assigned to the next generation. See the sections on *Crossover* and *Mutation* for details of those operations. Members of the next generation replaced members of the current population according to the replacement rules. See the section on *Selection and Replacement* for more information. Program termination took place after 1,000 total generations or 100 generations had passed without an improvement in solution cost. See the section on *Program Termination* for more information.

Chromosomal Representation

The chromosomal representation was composed of the set of complementary suppressions S_2 that represents a feasible solution to a cell suppression problem (i.e, table T is *safe* with respect to $S = S_1 \cup S_2$). To ensure the chromosomal representation provided a *safe* solution with respect to *T*, two checking functions, *isSafe* and *isSafeTable* were developed.

Solution Checking Functions

The function *isSafe*(r, c, T, S_1, S_2) determined whether a single primary suppression (*r*, *c*) was *safe* with respect to *S*. The function used the suppressed cells present in *C*(*r*, *c*) to calculate the maximum and minimum values that *cell* could assume

and compare them to the *cell's* upper and lower protection limits. If either maximum or minimum calculated values for *cell* violated the requirements for protecting the sensitive cell, the function returned *false*; otherwise, it returned *true*.

A second function *isSafeTable* was developed to check the entire table *T* to determine whether it was *safe* by using the function *isSafe*. *isSafeTable* iterated through each of cells in S_1 and called the function *isSafe* to determine whether they were protected. If *isSafe* returned *false* during any iteration, *isSafeTable* returned *false*.

Initial Population Generation

A population is a collection of chromosomes. The initial population of parent chromosomes was created using a hypercube-based method on a two-dimensional table that forms a circuit of suppressions in the form of a rectangle (Giessing & Repsilber, 2002). The hypercube method was implemented in the procedure *generateRectangle.*

The function *generateRectangle* $(s_i, T, S_1, S_2, lpr, upr)$ was based on a shortest path (SP) heuristic, which finds a protection pattern of three cells (*r*, *j*) - (*i*, *j*) - (*i*, *c*) that form a rectangle protecting a sensitive cell (*r*, *c*) (Castro, 2012). Figure 2 outlines the main steps of the function.

```
generateRectangle(s_i, T, S_1, S_2, lpr, upr)
    cS = ∅                    // suppressions protecting s_i
    S = ∅                     // rectangle protecting s_i
    upl = w_i + w_i × upr     // sensitive cell's upper protection requirement
    lpl = w_i − w_i × lpr     // sensitive cell's lower protection requirement
    for *pl ∈ {upl, lpl} do   // for protection levels
        Find SP for *pl
        If not SP found
            Find lowest cost path and assign to SP
        cS = {cells forming SP or lowest cost path}
        S = S ∪ cS
    Return S
```

Figure 2: Shortest Path Heuristic for creating Rectangle of Suppressed Cells

The function started by searching the sensitive cell's rows and columns for suppressions to cover the cell's upper protection limit. The function then searched for suppressions to cover the sensitive cell's lower protection level. It used the function *getCost* to return a cell's cost depending upon its inclusion in either S_1 or S_2. If sufficient cover was not found given the constraint of a circuit with four cells, the function found the lowest cost circuit forming a rectangle and returned it in S_2.

Due to the cube's cardinality constraint, this method had a tendency to over-suppress. Additionally, the cube method could not guarantee that a feasible set of suppressions would be found (Almeida et al., 2006).

The function *getCost*$(cell, T, S_1, S_2)$ found the cost of the cell under consideration by checking for the condition $cell \in S_1 \cup S_2$. If the cell was present in S_1 or S_2, a cost of zero was returned. Otherwise, *getCost* returned a cost equal to the cell's weight in *T*.

The function *removeRedundant*(T, S_1, S_2) searched S_2 for complementary suppressions in rows and columns where $n_{r+}(S) > 2$ and $n_{+c}(S) > 2$. The complementary suppression located at the row or column was removed from S_2 and the sensitive cells present in the circuits that included the removed cell were tested using *isSafe*. If *isSafe* returned *true* for each affected sensitive cell, the modification to S_2 was accepted. If *isSafe* returned *false*, S_2 was restored to its original state.

The process used to create the initial population is outlined in Figure 3.

$S_2 = \emptyset$
$L = [c_1, c_2, \dots, c_{|S_1|}]$ // random shuffled list of sensitive cells in S_1
For $k = 1,2, \dots, |S_1|$ // iteratively protect sensitive cells
While *not isSafe*$(L[k], T, S_1, S_2)$
$S_2 = S_2 \cup$ *generateRectangle*$(L[k], T, S_1, S_2, LPR, UPR)$
removeRedundant(T, S_1, S_2) // remove redundant suppressions.

Figure 3: Procedure to Generate Chromosomes

Chromosomes were created by first randomly ordering the sensitive cells in S_1. This had the effect of changing the order of the existing complementary suppressions in S_2 used to form new circuits of protection for the cells in S_1, which in turn ensured the overall genetic makeup of each individual was unique. Next, each of the sensitive cells was tested for protection from existing suppressions using the function *isSafe*. If a sensitive cell proved to be *unsafe*, the function *generateRectangle* was called to add rectangles (circuits of protection) as needed until *isSafe* returned true. After each execution of *generateRectangle*, the additional complementary suppressions were added to S_2.

After all of the sensitive cells were protected, the resulting set of complementary suppressions S_2 was be processed by the procedure *removeRedundant*, which systematically checked for complementary suppressions in S_2 that could be removed while leaving table *T safe*. After the initial parent population was created, the GA entered a bounded loop where the chromosomes underwent successive generations of crossover and mutation operations in an attempt to create offspring that provided lower-cost solutions.

Crossover

The goal of the crossover function was to increase the quality of solutions by exploiting genetic diversity in the current population through the exchange of complementary suppressions between selected circuits of protection in the offspring. This research explored the effect of using a heuristic to select the circuit of protection to be crossed over in order to improve the probability of producing a lower solution cost at each generation. The entire set of complementary suppressions protecting a sensitive cell

was swapped between parents, ensuring feasible solutions in the offspring. A list of complementary suppressions protecting each sensitive cell in S_1 was maintained to allow for a crossover operation that checked the participation of complementary suppressions across multiple circuits of protection. Complementary suppressions used to protect sensitive cells not involved in crossover were preserved in S_2. The complementary suppressions list was set to a maximum of 32 cells per circuit to allow for complex circuits greater than four cells while minimizing memory and computational overhead.

The results of the crossover operation were passed out through the parameters S_{2p1} and S_{2p2}. A high-level overview of the crossover process follows in Figure 4:

crossoverCircuits$(T, S_1, S_{2p1}, S_{2p2})$
$tC_1 \leftarrow \emptyset$ // temporary location for circuit being crossed from S_{2p1}
$tC_2 \leftarrow \emptyset$ // temporary location for circuit being crossed from S_{2p2}
$tC_{s1} \leftarrow \emptyset$ // shared complementary suppressions being crossed from S_{2p1}
$tC_{s2} \leftarrow \emptyset$ // shared complementary suppressions being crossed from S_{2p2}
// random search for overprotected sensitive cells
$L = [c_1, c_2, \dots, c_{|S_1|}]$ // random shuffled list of sensitive cells in S_1
For $k = 1,2, \dots, |L|$ // iteratively search for lower-cost offspring
If lower-cost offspring found break // exit for loop
If ***not*** found *return* ***false***
// if ***true***, perform crossover using circuits returned in tC_1, tC_2 and complementary suppressions returned in tC_{s1}, tC_{s2}.
$S_{2p1} = \left(S_{2p1} - (tC_1 - tC_{s1})\right) \cup tC_2$
$S_{2p2} = \left(S_{2p2} - (tC_2 - tC_{s2})\right) \cup tC_1$
return ***true***

Figure 4: Procedure crossoverCircuits

A sensitive cell in $S_1[k]$, where *k* is a randomly selected index between zero and $|S_1|$, was selected and its protection circuit crossed between the selected parents. A heuristic was applied that worked on the level of the circuits, requiring that the crossover

operation produce a lower-cost offspring than the lowest-cost parent based on $f(n) = g(n) + h(n)$, such that $h(n) < C(n)$ where (1) $g(n)$ is cost of the circuit being crossed; (2) $h(n)$ is the estimated lowest-cost solution which provides cover for all sensitive cells not including the sensitive cells being crossed; and (3) $C(n)$ is the current solution cost not including the circuit being crossed. If either of the offspring's solution cost made the function *true*, the crossover was accepted. Otherwise, the crossover was rejected and a new sensitive circuit was selected for crossover and testing. This cycle was repeated until a lower-cost offspring was generated or 100 attempts passed without success. If the crossover was successful, the offspring were passed to the next phase, mutation. Protection circuits of the same cost were assumed to be identical and therefore rejected. When the operation failed, two new parents were selected for mating and the sequence was repeated. This process continued until a protection circuit was selected or a predetermined stopping condition was reached.

Complementary suppressions shared with other protection circuits in $S_1 \cup S_{2p*}$ were returned in tC_{s*} and not removed from S_{2p*} to safeguard the feasibility of the remaining protection circuits in $S_1 \cup S_{2p*}$. The feasibility of the offspring's solution was ensured by crossing over all complementary suppressions used by the circuits protecting the selected sensitive cell $S_1[k]$.

Mutation

The purpose of the mutation operation in this research was to improve solution cost by removing unneeded complementary suppressions. A sensitive cell was selected at random and the complementary suppressions protecting it checked for redundancy. Once a redundant complementary suppression was found, it was removed from S_{2p*} and all

affected circuits protecting S_1 were merged. Finally, the new chromosomal representation $S_1 \cup S_2$ was tested for feasibility. A high level overview of the mutate process follows in Figure 5:

mutateOffspring$(k, S_1 \cup S_{2p*})$
    while redundant suppression ***not*** found
    check S_{2p*} for redundant complementary suppression
    if redundant suppression found
        $S_{2p*} = S_{2p*} - tC_1$ // remove complementary suppression
        if *isOffspringTableSafeByCell*$(k, S_1 \cup S_{2p*})$
            JoinOffspringCircuits(k, S_1, S_{2p*})
        else
            $S_{2p*} = S_{2p*} + tC_1$ // restore complementary suppression

Figure 5: Procedure mutateOffspring

To ensure the feasibility of the solution, all protection circuits affected by the removal of a complementary suppression were tested using the function *isSafe*.

Selection and Replacement

A problem with GA's is their tendency to converge around a local optimum. This is the result of the selection and replacement operations that choose most fit members of the population for mating and replace less fit members of the population with their offspring. Through successive generations, the offspring of most fit members will tend to cluster around the genetic patterns of the parents, preventing a solution at the global optima. This problem may be mitigated through selection and replacement operations that help provide for genetic diversity between generations (Smith et al., 2009; Razali & Geraghty, 2011). Selection is the process whereby individuals from the current population are chosen for mating, which determines the search space available for the GA's crossover and mutation operations. Therefore, the strategy used contributes to the genetic diversity present at each generation and can be instrumental in determining the

rate of convergence of the GA (Razali & Geraghty, 2011). The replacement strategy determines which individuals in the current population will be replaced by the offspring at each generation. The goal of replacement is to increase the frequency of most fit genetic sequences present at each generation, allowing for convergence at or near an optimal sequence (Smith, 2007; Mengshoel & Goldberg, 2008).

A purely deterministic approach made up of elitist selection and replacement risks giving too strong a convergence at inferior local optima by forcing successive generations into the genetic patterns of the most fit members of the current population while eliminating genetic diversity present in the least fit members (Smith et al., 2009; Razali & Geraghty, 2011). It is possible this problem may be mitigated by introducing probabilistic strategies into the selection and replacement process.

This research evaluated different permutations of Roulette Wheel and Tournament Selection along with Similar and Kill Tournament replacement strategies in order to determine whether genetic diversity could be increased and premature convergence reduced as compared to elitist selection and replacement. Figure 6 gives a high level representation of the process:

Select deterministic or probabilistic selection and replacement for testing
if (*deterministic*)
Employ elitist selection and replacement
else // *probabilistic strategies*
    Select one of: // Selection strategies
        Tournament selection
        Roulette Wheel Selection
    Select one of: // Replacement strategies
        Similar
        Kill Tournament

Figure 6: Selection and Replacement Process

Elitist selection and replacement provides for a high level of selection pressure,

resulting in rapid convergence at the expense of genetic diversity (Lozano, Herrera, & Cano, 2005; Vavak & Fogarty, 1996). To counter this tendency, this research explored selection strategies that mated more fit individuals with less fit individuals, in concert with replacement strategies that maintained less fit individuals in the current population in order to moderate selective pressure and maintain genetic diversity.

Tournament and Roulette Wheel selection strategies were tested to determine whether they were capable of increasing the search space, allowing for genetic recombination that provided for lower solution cost. Tournament selection functioned by choosing several members from the populations at random to form a set from which the most fit members were selected for mating. This method had been shown to help maintain good diversity, but at the cost of increased convergence time (Smith et al., 2009; Razali & Geraghty, 2011). The Roulette Wheel selection strategy involved individuals being selected for mating with a probability proportional to their fitness. The probability of any one individual being selected for mating is defined as $p_{i=\frac{f_i}{\sum_{j=1}^{n} f_j}}$ where f_i is the fitness values of individual i and p_i is the probability of individual i being selected. The advantage of this method is that all individuals in the current population have a chance of being selected (Razali & Geraghty, 2011).

Kill Tournament and Similar replacement strategies were tested to determine whether genetic diversity could be sufficiently maintained to allow promising regions of the search space found in less fit individuals that would otherwise be lost to more fit offspring to be explored. The Kill Tournament replacement strategy involved selecting members from the current population at random to compete with the offspring for a place in the current generation. Replacement was based on fitness with the number of parents

selected for the tournament being subject to variability. This allowed for a range of selective pressure. This strategy has the advantage of not requiring the resorting of the population after replacement (Smith, 2007). The Similar replacement strategy functioned by randomly selecting several members from the current population and replacing those that were *most similar* to the offspring. *Most similar* was defined by the chromosomal makeup (Gupta & Ghafir, 2012). This strategy has the advantage of maintaining genetic diversity by ensuring genetically dissimilar individuals are present in the current population. The disadvantage is that less fit offspring have the possibility of replacing more fit individuals in the current generation.

Both Tournament and Roulette Wheel selection strategies were paired with Kill Tournament and Similar replacement strategies to determine if a selection/replacement strategy pairing could be found that consistently provided for lower-cost solutions compared to Elitist selection and replacement.

Termination

Termination took place when 1,000 generations passed or 100 generations took place without an improvement in the solution cost. The number of generations used to determine program termination was based on solution cost and execution time. Experiments that exceeded one hour for a single dataset were terminated due to excessive run time.

Evaluation of the Results

Solutions returned from Shortest Path, GenSup, HyperCube and HeurGene algorithms were compared. Comparisons were made for both solution cost and execution time. The goal of this research was to achieve lower-or equal-cost solutions at lower

computational expense as compared to previous methods.

The data used consisted of seven synthetic sets of two-way tables (see table 1 for a summary), with cell counts of 10,000, and 100,000 each with an internal value distribution of pseudo randomly-selected real numbers between 0 and 1,000. Each of the three sets had a randomly-selected subset of sensitive cells consisting of 0.5%, 1%, and 3% of the total internal cells (Ditrich, 2010). Additionally, there was one set labeled with an RC that contained sensitive cells equal to the number of rows or columns in the set. Ten different instances of each of the datasets were tested to minimize the possibility that the results were specific to a particular instance of a dataset. The upper and lower protection levels were set to ±10% of the sensitive cell's value for all datasets.

DataSet	Name	Rows × Cols	Sensitive Cells	Marginal Cells
1	10000Cells0.5	100 × 100	50	201
2	10000Cells1	100 × 100	100	201
3	10000Cells3	100 × 100	300	201
4	100000Cells0.5	400 × 250	500	1,101
5	100000Cells1	400 × 250	1,000	1,101
6	100000Cells3	400 × 250	3,000	1,101
7	100000CellsRC	400 × 250	250	1,101

Table 1: DataSets

Performance of the algorithm was measured as a function of the amount of CPU time required for program completion and the cost of the solutions as determined by averaging repeated runs of the GA. The performance of the HeurGene algorithm presented in this research was evaluated against HyperCube, Shortest Path and Almedia's GenSup Genetic Algorithm methods.

The evaluation investigated the merits of using:

1. a heuristic to select circuits in a chromosomal representation for crossover,
2. a feasible solution-preserving crossover,

3. a deterministic selection of locations in a chromosomal representation for mutation,
4. a portfolio of selection and replacement techniques to avoid premature convergence,

Format of Results

The results of the research are presented in the form of graphs in order to allow for a high-level graphical illustration of the results and textual tables that present more detailed information. In addition, written summaries of results accompany the graphs and tables, providing additional information and analysis.

Resources

This research utilized the resources as noted below:

- Computer, Dell Precision T5400 housing two quad core 2.5 GHz Xeon processors and 16 GB RAM running under Windows 7
- Microsoft C++ Compiler and Visual Studio 2010

Preliminary Testing

Preliminary tests were conducted with four variations of Fitness Proportional Roulette-Wheel and Random Tournament selection strategies paired with Kill Tournament and Similar replacement strategies. Additional tests were conducted using Elite selection and replacement and a probabilistic algorithm that used a pseudorandom number generator to select between each of the selection and replacement strategies. These tests were conducted to determine if any permutation of selection and replacement strategies provided for lower-cost solutions.

The tests were conducted on ten datasets. Each variation of strategies was run ten times for each dataset for a total of 100 executions each on tables of 10,000 cells with

0.5%, 1% and 3% sensitive cells. These datasets were selected due to the relatively low processing time required for a 10,000 cell table. The results of the tests are summarized in Tables 2, 3 and 4.

	Fitness Proportional Roulette-Wheel / Kill Tournament	**Fitness Proportion al Roulette-Wheel / Similar**	**Random Tournament / Kill Tournament**	**Random Tournament / Similar**	**Elite / Elite**	**Proba-bilistic**
Average	5760	5744	5674	5710	5711	***5623***
Median	5766	5709	***5638***	5757	5647	5640
Min	5193	***5135***	5224	5263	5229	5157
Max	6230	6393	6397	6112	6260	6155
Standard Deviation	309.64	382.08	385.80	295.25	363.02	355.09

Table 2: Preliminary Testing of Selection / Replacement Strategies for with 10,000 Cells with 0.5% Sensitive Cells

The tests conducted at the 0.5% level demonstrated that a Probabilistic strategy provided a slightly better average cost than other strategies while the Random Tournament / Kill Tournament strategy provided a better median cost. Summary results for the different strategies are presented in Table 2. A plot showing the average solution cost for each dataset is presented in Figure 7. On the figure, (1) FPRWS KTR stands for Fitness Proportional Roulette-Wheel selection paired with Kill Tournament selection; (2) FPRWS SR stands for Fitness Proportional Roulette-Wheel selection paired with Similar replacement; (3) Prob stands for Probabilistic selection and replacement where a random number generator selects the pairing of strategies; (4) RTS KTR stands for Random Tournament selection paired with Kill Tournament replacement; (5) RTS SR stands for Random Tournament selection paired with Similar replacement; (6) and Elite stands for Elite selection and replacement.

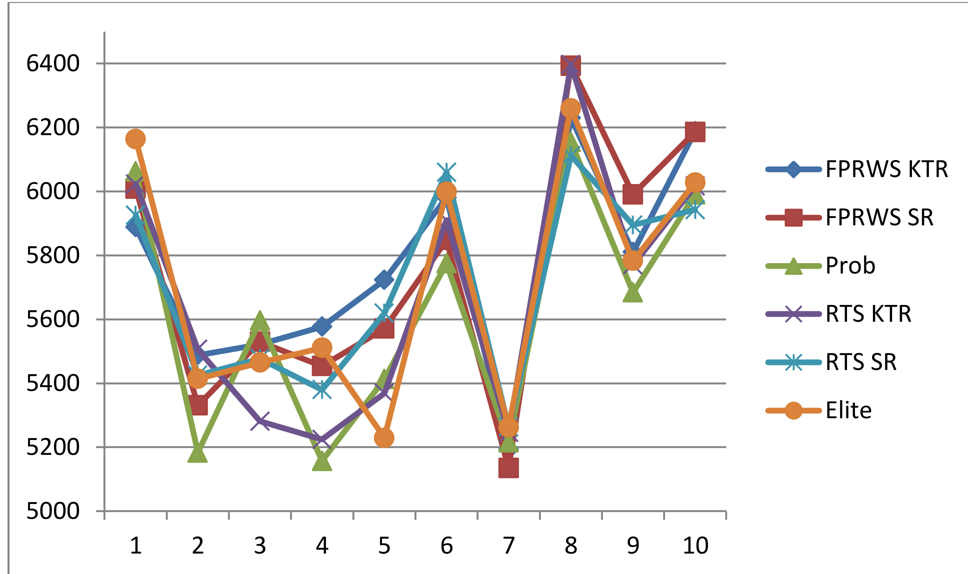

Figure 7: Comparison of Average Costs per Strategy for each Dataset with 10,000 Cells with 0.5% Sensitive Cells

The plot of the different strategies demonstrates that no one method produced significantly better quality solutions than any other. Overall, the Probabilistic algorithm produced the lowest average solution cost and also the lowest cost for four of the datasets and the highest cost for one. This suggested that the probabilistic method would make a satisfactory candidate for the purposes of this research.

The tests conducted at the 1% level demonstrated Random Tournament selection strategy paired with Kill Tournament replacement strategy provided a slightly better average cost than other strategies while the Probabilistic approach provided the lowest average median cost. Summary results for the different strategies are presented in Table 3.

	Fitness Proportional Roulette-Wheel / Kill Tournament	Fitness Proportional Roulette-Wheel / Similar	Random Tournament / Kill Tournament	Random Tournament / Similar	Elite / Elite	Proba-bilistic
Average	6847	6942	***6731***	6856	6830	6778
Median	6716	6799	6695	6797	6747	***6693***
Min	5964	6118	***5746***	6026	5885	5857
Max	7952	7832	7819	7908	8017	7951
Standard Deviation	615.94	619.80	534.20	631.75	635.70	625.45

Table 3: Preliminary Testing of Selection / Replacement Strategies for 10,000 Cells with 1% Sensitive Cells

Figure 8 shows the Random Tournament / Kill Tournament strategy produced the lowest cost on three of the datasets, while the Probabilistic algorithm generated the lowest cost on four. As with the table at 0.5% sensitive cells, this suggests the Probabilistic strategy has the potential for producing lower-cost solutions for these tables as compared to the other solutions.

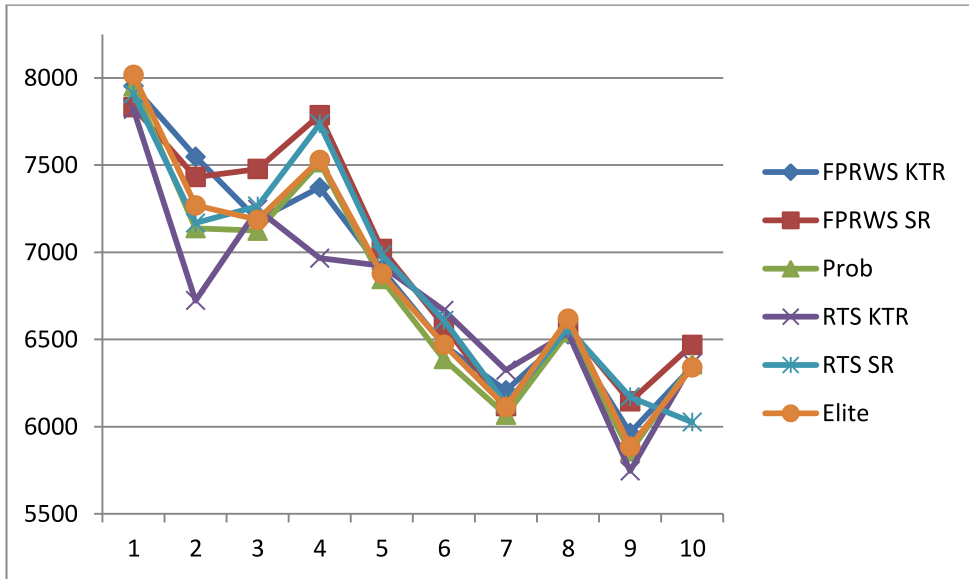

Figure 8: Comparison of Average Costs per Strategy for each Dataset with 10,000 Cells with 1% Sensitive Cells

The tests conducted at the 3% level demonstrated that the Random Tournament

selection strategy paired with the Kill Tournament replacement strategy provided a slightly better average cost than other strategies while the Random Tournament/Similar strategy yielded the lowest average median cost. Summary results for the different strategies are presented in Table 4.

	Fitness Proportional Roulette-Wheel / Kill Tournament	Fitness Proportional Roulette-Wheel / Similar	Random Tournament / Kill Tournament	Random Tournament / Similar	Elite / Elite	Proba-bilistic
Average	2877	2882	***2816***	2833	2874	2818
Median	2971	2917	2809	***2739***	2921	2845
Min	2137	2200	2164	***2171***	2236	2173
Max	3537	3519	3514	3520	3430	3309
Standard Deviation	382.03	346.1	353.24	339.27	324.15	362.12

Table 4: Preliminary Testing of Selection / Replacement Strategies for 10,000 Cells with 3% Sensitive Cells

An examination of Figure 9 suggests that at the 3% level of sensitive cells, none of the strategies produced lower-cost solutions than any other.

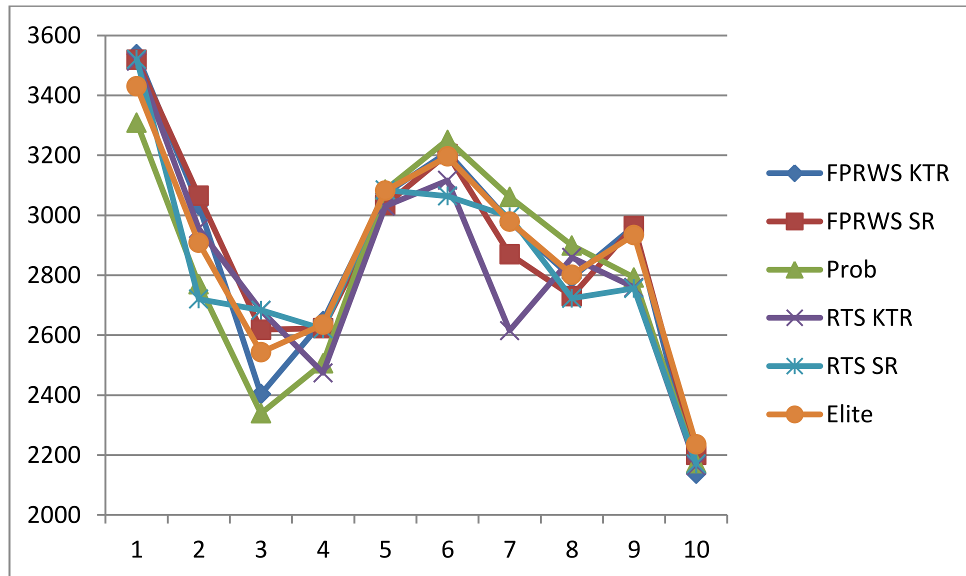

Figure 9: Comparison of Average Costs per Strategy for each Dataset with 10,000 Cells with 3% Sensitive Cells

Preliminary Tests Conclusions

The results of the preliminary testing were inconclusive, with no one strategy demonstrating a significant advantage of another. Even though the preliminary tests did not indicate that any one strategy provided for significantly lower solution costs at 0.5% and 1% sensitive cells, the Probabilistic strategy produced a greater number of lower-cost solutions than the other strategies. For these reasons, the Probabilistic strategy was selected for inclusion in this research.

Chapter 4

Results

Introduction

Shortest Path and HyperCube heuristic methods as well as GenSup and HeurGene GAs were tested with table sizes of 10,000 and 100,000 cells with 0.5%, 1% and 3% sensitive cells. Additionally, datasets of 100,000 cells with 0.25% sensitive cells were included to provide tables with sensitive cells equal to the minimum number of rows or columns in the table. Each method was executed ten times against each of the datasets. A Unicode file was output at the termination of execution for each method for each dataset, giving a run time summary with costs and execution times. See Appendix A for an example of the data file output format.

The data from the experiments are summarized in the following tables and figures. The Shortest Path heuristic results were significantly inferior to the HyperCube heuristics, both in terms of solution quality and computational costs, and therefore not included in the results that follow. Results for solution and computational costs are presented in separate tables, which provide the average, median, highest, and lowest values produced along with the standard deviation. The best average and median values among the different methods are highlighted in boldface italics. The results tables allow for a tabular visual comparison of the summary data for each dataset.

The accompanying figures give a graphical representation of the aggregated data for each dataset. Results presented include solution costs and execution times. Gaps in

the graphs indicate that a solution was not found for a dataset for a given strategy. The results represent the average of ten executions for each method on ten datasets for each of the specified tables.

Discussion of Test Results for Tables with 10,000 Cells

The purpose of the datasets with 10,000 cells was to test the HeurGene's effectiveness on small tables and establish a baseline for comparison with tables with larger tables of 100,000 cells. The comparisons were made on the bases of solutions costs and CPU time. CPU time is presented in seconds. The CPU times for GenSup and HeurGene were measured exclusive of the time required to create the initial population using the HyperCube Method.

Test Results for Tables with 10,000 Cells and 0.5% Sensitive Cells

Table 5 shows that the HeurGene algorithm produced the lowest average overall solutions cost as compared to the other strategies. The GenSup algorithm demonstrated a ~10% improvement over the HyperCube solution cost while the HeurGene algorithm yielded ~11% improvement. Additionally, the HeurGene strategy produced the lowest median cost along with the smallest variance.

	HyperCube	**GenSup**	**HeurGene**
Average	5768	5530	***5445***
Median	5601	5421	***5351***
Highest	6371	7110	6013
Lowest	5297	4021	5131
Standard Deviation	324	973	315

Table 5: Comparison of Average Solution Costs with 10,000 Cells with 0.5% Sensitive Cells

Figure 10 gives a comparison of the average solution cost for each dataset. The graph shows that while the HeurGene did not create the lowest cost solution it

consistently produced average costs less than the average cost of the HyperCube algorithm.

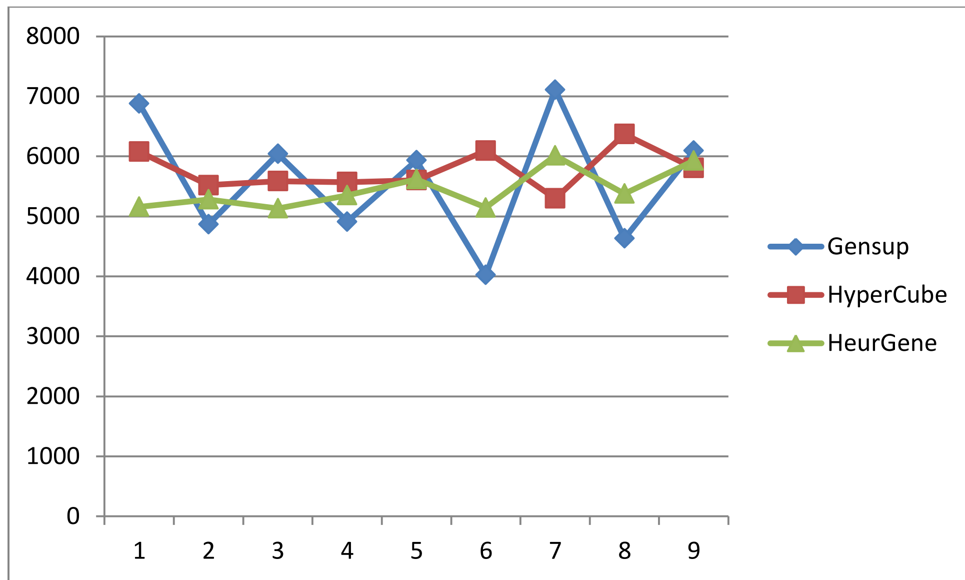

Figure 10: Comparison of Average Solution Costs (Y-Axis) with 10,000 Cells with 0.5% Sensitive Cells

A comparison of the run times presented in Table 6 shows that on average the HeurGene strategy was capable of producing solutions requiring CPU time an order of magnitude less than GenSup. CPU times for the Hypercube method averaged 0.0468 seconds with a median time of 0.0468 seconds and a standard deviation of 0.0002.

	GenSup	HeurGene
Average	220.2564	11.1194
Median	214.6291	10.9510
Highest	322.5163	13.6190
Lowest	154.477	8.8610
Standard Deviation	44.7983	1.5367

Table 6: Comparison of Average CPU Run Times (in seconds) with 10,000 Cells with 0.5% Sensitive Cells

Figure 11 gives a detailed summary of the runtimes for each of the datasets. The graph shows that HeurGene's CPU execution times were consistently low across each of

datasets. These results suggest that the feasible solution-preserving genetic operators allowed HeurGene to quickly converge on local optima. GenSup required longer execution times and showed a larger variance.

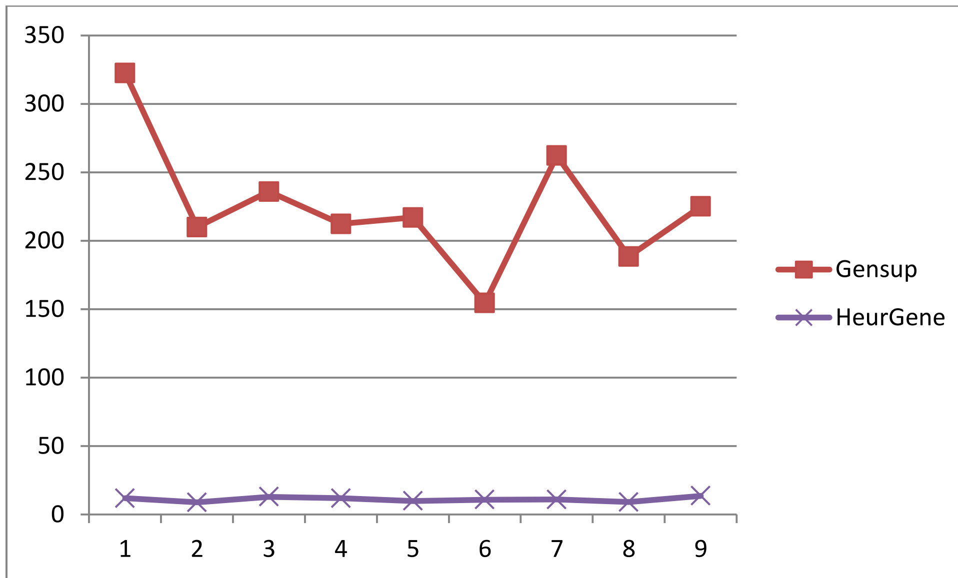

Figure 11: Comparison of Average CPU Run Times (Y-Axis in seconds) with 10,000 Cells with 0.5% Sensitive Cells

These results suggest that with tables of 10,000 cells and 0.5% sensitive cells, HeurGene is capable of producing higher quality solutions than the other algorithms, while requiring a fraction of the processing time as compared to GenSup.

Test Results for Tables with 10,000 Cells and 1% Sensitive Cells

Table 7 shows that the GenSup algorithm produced the lowest average overall solutions cost as compared to the other strategies. The GenSup algorithm demonstrated an ~21% improvement over the hypercube solution cost while the HeurGene algorithm yielded an ~11% improvement. GenSup also produced the smallest standard deviation and lowest median values.

	HyperCube	GenSup	HeurGene
Average	6885	***5441***	6066
Median	6782	***5429***	6099
Highest	7867	6366	6971
Lowest	6030	4885	5268
Standard Deviation	636	406	493

Table 7: Comparison of Average Solution Costs with 10,000 Cells with 1% Sensitive Cells

Figure 12 gives a comparison of the solution costs for the different methods. The graph demonstrates that both GenSup's and HeurGene's ability to improve the HyperCube's average solution cost and that on a dataset-by-dataset comparison, GenSup was able to consistently produce lower-costs than HeurGene.

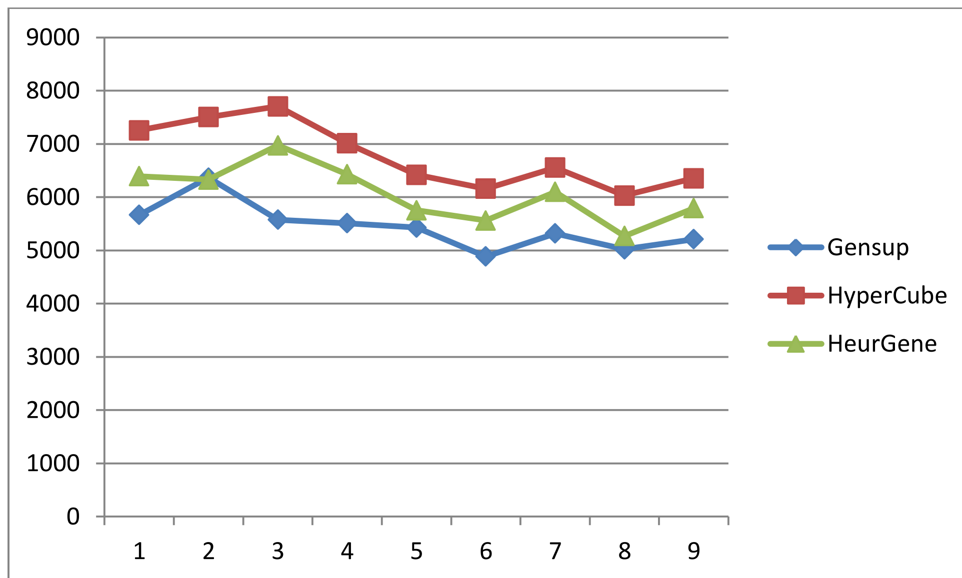

Figure 12: Comparison of Average Solution Costs (Y-Axis) with 10,000 Cells with 1% Sensitive Cells

A comparison of the run times presented in Table 8 shows that on average the HeurGene strategy was capable of producing solutions requiring about 5% the CPU time as compared to GenSup. CPU times for the HyperCube method averaged 0.0936 seconds

with a median time of 0.0936 seconds and a standard deviation of 0.0001.

	GenSup	HeurGene
Average	472.2239	20.1943
Median	474.4680	18.2830
Highest	480.7680	30.8880
Lowest	455.7240	15.9120
Standard Deviation	6.9158	4.5564

Table 8: Comparison of Average CPU Run Times (in seconds) with 10,000 Cells with 1% Sensitive Cells

Figure 13 gives a detailed summary of the CPU runtimes for each of the datasets. The graph shows how HeurGene's CPU execution times were consistently low across each of datasets as compared to GenSup. These results indicate that HeurGene converged more quickly on local optima than did GenSup.

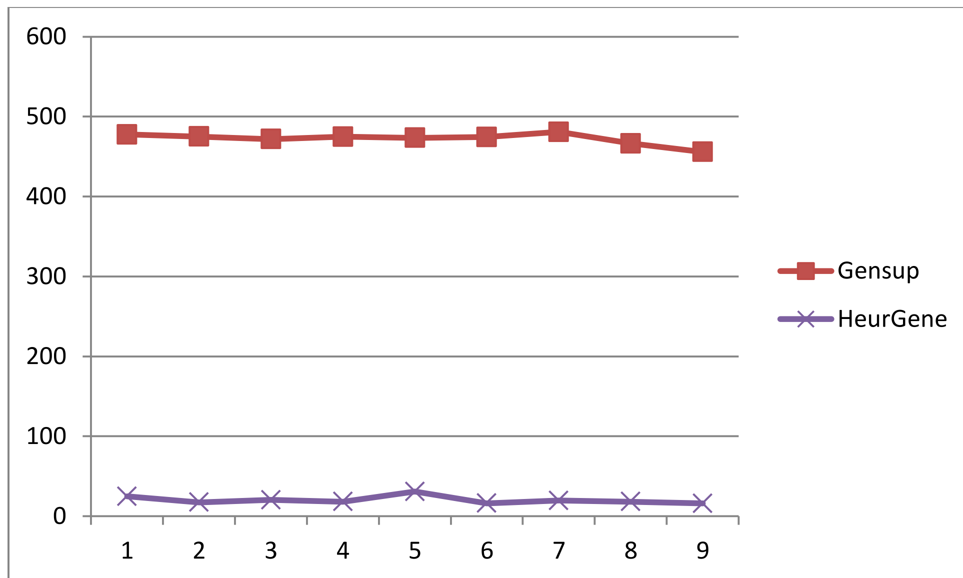

Figure 13: Comparison of Average CPU Run Times (Y-Axis in seconds) with 10,000 Cells with 1% Sensitive Cells

These results suggest that with tables of 10,000 cells and 1% sensitive cells, HeurGene is capable of producing a high quality solution within ~10% of GenSup, while requiring only ~5% the CPU time.

Test Results for Tables with 10,000 Cells and 3% Sensitive Cells

Table 9 shows that the GenSup algorithm produced the lowest average and median solutions cost as compared to the other strategies along with the lowest variance. The GenSup algorithm demonstrated a ~23% improvement over the hypercube solution cost, while the HeurGene algorithm yielded a ~13% improvement. GenSup produced the smallest average standard deviation across the ten datasets.

	HyperCube	GenSup	HeurGene
Average	2922	***2304***	2603
Median	2997	***2253***	2670
Highest	3484	2746	3020
Lowest	2253	1816	2009
Standard Deviation	343	287	305

Table 9: Comparison of Average Solution Costs with 10,000 Cells with 3% Sensitive Cells

Figure 14 gives a dataset-by-dataset comparison of the solution costs for the each method. The graph shows that both GenSup and HeurGene were, on average, able to improve the HyperCube's average solution cost, and that, in a dataset-by-dataset comparison, GenSup was able to consistently produce lower-cost solutions than HeurGene.

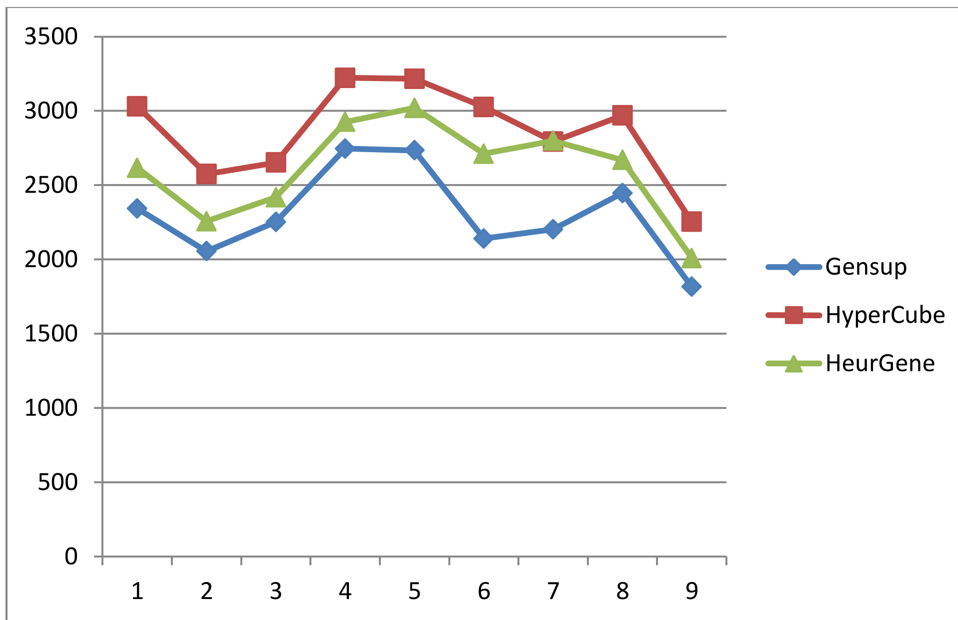

Figure 14: Comparison of Average Solution Costs (Y-Axis) with 10,000 Cells with 3% Sensitive Cells

A comparison of the run times presented in Table 10 shows that on average the HeurGene strategy was capable of producing solutions requiring ~8% the CPU time compared to GenSup. CPU times for the HyperCube method averaged 0.3106 seconds with a median time of 0.3113 seconds and a standard deviation of 0.0019.

	GenSup	HeurGene
Average	811.2831	64.5615
Median	824.3360	66.7520
Highest	952.6630	71.4950
Lowest	671.2850	56.5190
Standard Deviation	86.4002	4.7155

Table 10: Comparison of Average CPU Run Times (in seconds) with 10,000 Cells with 3% Sensitive Cells

Figure 15 gives a summary of the runtimes for each of the datasets. The graph shows that HeurGene's CPU execution times were consistently low across each of datasets. GenSup gave the highest CPU times and largest standard deviation. This was likely due to the number of generations that the GAs produced before their termination

conditions were satisfied.

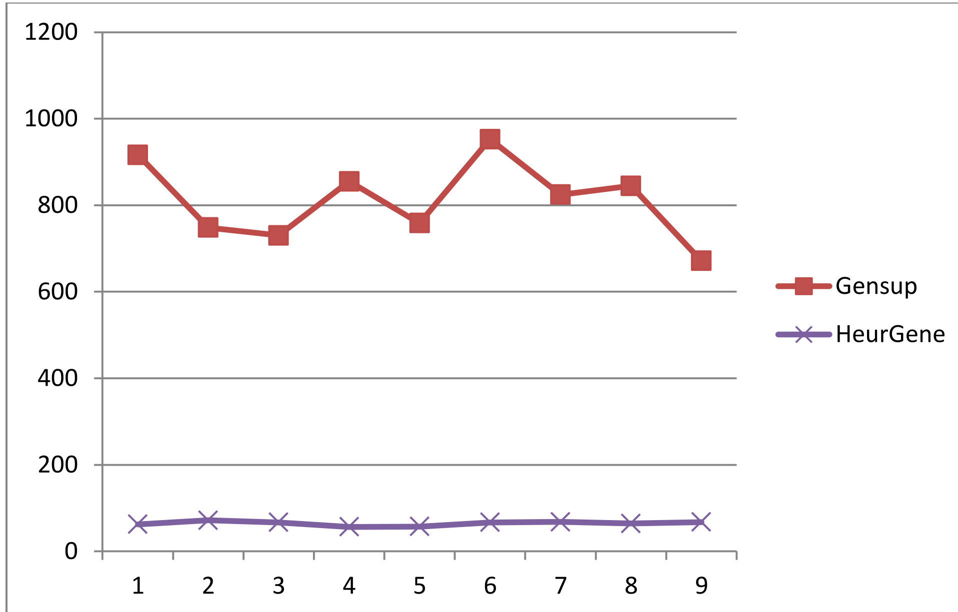

Figure 15: Comparison of Average CPU Run Times (Y-Axis in seconds) with 10,000 Cells with 3% Sensitive Cells

These results indicated that with tables of 10,000 cells and 1% sensitive cells, HeurGene is capable of producing a high-quality solution, within ~11% of GenSup, while requiring only ~8% the CPU time.

Discussion of Test Results for Tables with 100,000 Cells

Datasets with 100,000 cells were designed to test HeurGene's scalability. Owing to excessive run time requirements, the GenSup algorithm was run only once on tables with 0.5% and 1% sensitive cells. The remaining algorithms were run ten times against the datasets as previously noted.

Test Results for Tables with 100,000 Cells and 0.25% Sensitive Cells

Table 11 shows that the HeurGene algorithm produced the lowest average overall solution cost as compared to the other strategies. The GenSup algorithm demonstrated an average ~0.02% improvement over the HyperCube solution cost, while the HeurGene

algorithm yielded a ~9.7% improvement.

	HyperCube	GenSup	HeurGene
Average	17810	17777	***17230***
Median	17538	17394	***17053***
Highest	19296	19638	19147
Lowest	16619	16717	16158
Standard Deviation	792	1035	892

Table 11: Comparison of Average Solution Costs with 100,000 Cells with 0.25% Sensitive Cells

Figure 16 gives a comparison of the solution costs for different methods. The graph shows that GenSup failed to find solutions for datasets 5 and 10. This is the result of the HyperCube algorithm failing to find solutions for GenSup's the initial population. GenSup's increased costs over HyperCube for datasets 2 and 8 are likely due to the fact that HyperCube algorithm produced higher cost initial populations for the GAs. The graph suggests that HeurGene is consistently capable of improving on the HyperCube's solution costs.

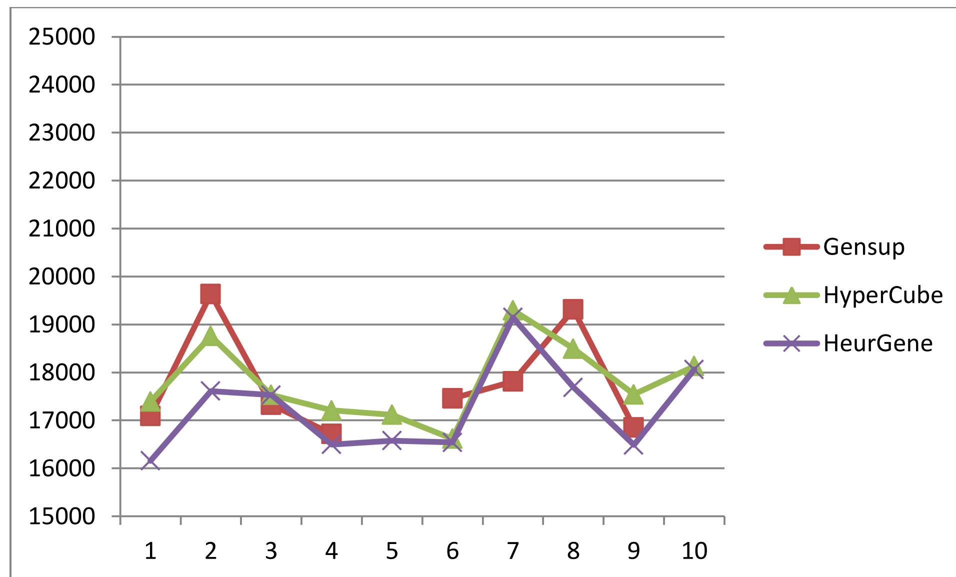

Figure 16: Comparison of Average Solution Costs (Y-Axis) with 100,000 Cells with 0.25% Sensitive Cells

A comparison of the run times presented in Table 12 shows that on average the HeurGene strategy was capable of producing solutions requiring ~5% the CPU time as compared to GenSup. CPU times for the HyperCube method averaged 2.85554 seconds with a median time of 2.85785 seconds and a standard deviation of 0.0217.

	GenSup	HeurGene
Average	12024.0222	511.1365
Median	12056.5141	511.4625
Highest	12321.6464	563.7090
Lowest	11691.6279	463.5860
Standard Deviation	199.8669	27.1727

Table 12: Comparison of Average CPU Times with 100,000 Cells with 0.25% Sensitive Cells

Figure 17 gives a detailed summary of the runtimes for each of the datasets. The graph shows that HeurGene's CPU execution times were consistently low across each of datasets.

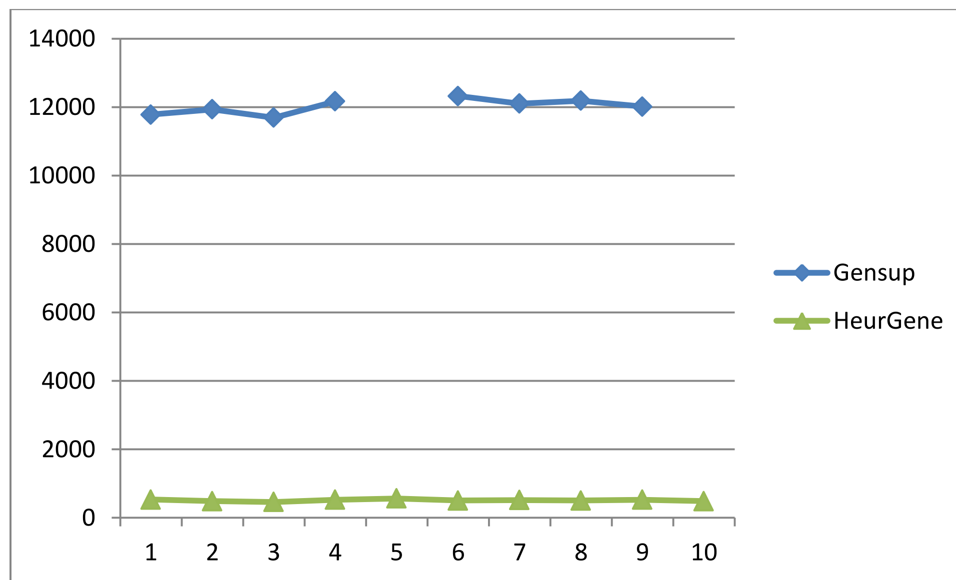

Figure 17: Comparison of Average CPU Times (Y-Axis in seconds) with 100,000 Cells with 0.25% Sensitive Cells

These results suggest that with tables of 100,000 cells and 0.25% sensitive cells,

HeurGene is capable of producing a high-quality solution that represents a ~9.7% improvement over the HyperCube. Additionally, HeurGene's CPU times were 5% those of GenSup.

Test Results for Tables with 100,000 Cells and 0.5% Sensitive Cells

Table 13 shows that the GenSup algorithm produced the lowest average overall solution cost as compared to the other strategies. The GenSup algorithm demonstrated an average ~ 11% improvements over the HyperCube solution cost while the HeurGene algorithm yielded ~4% average improvement. However, it must be noted that due to excessive runtime requirements, the GenSup algorithm was run only once and not ten times against each dataset as with the other strategies.

	HyperCube	**GenSup (1 Run)**	**HeurGene**
Average	17156	***15337***	16469
Median	16842	***15019***	16143
Highest	18930	17539	18287
Lowest	16276	14796	15593
Standard Deviation	854	853	910

Table 13: Comparison of Average Solution Costs with 100,000 Cells with 0.5% Sensitive Cells

Figure 18 gives a comparison of the solution costs for the different methods. Two outliers at datasets 5 and 6 were removed from the GenSup data as they were the result of the fact that the HyperCube algorithm created an excessively high-cost initial population.

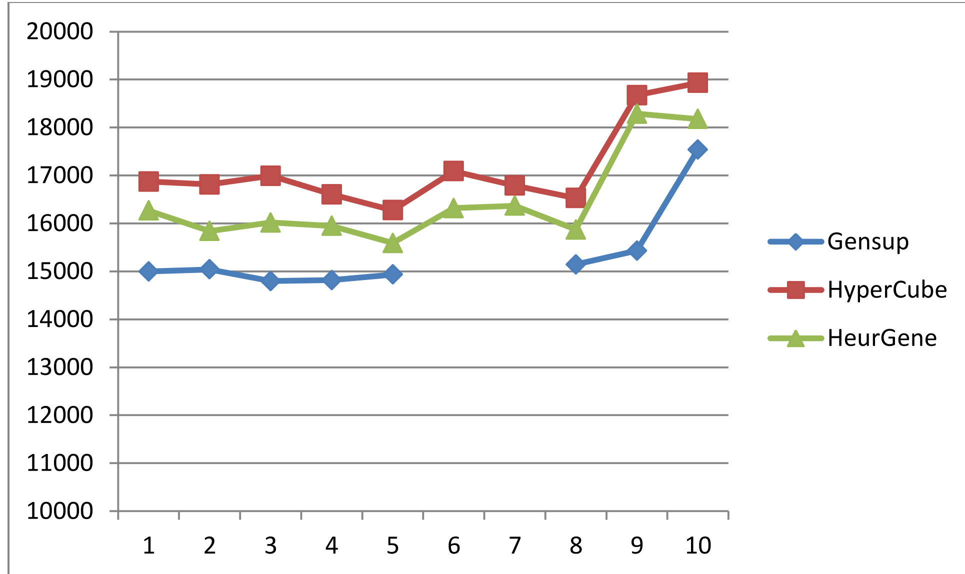

Figure 18: Comparison of Average Solution Costs (Y-Axis) with 100,000 Cells with 0.5% Sensitive Cells

A comparison of the run times presented in Table 14 shows that, on average, the HeurGene strategy was capable of producing solutions requiring ~10% of the CPU time as compared to GenSup. The GenSup data represents an aggregated result for a single run of each of the ten datasets as the execution time was in excess of six hours per dataset. CPU times for the HyperCube method averaged 5.84 seconds with a median time of 5.67 seconds and a standard deviation of 0.22.

	GenSup (1 Run)	HeurGene
Average	22452.63	2089.52
Median	22452.63	2064.21
Highest	22452.63	2213.72
Lowest	22452.63	2028.44
Standard Deviation		61.55

Table 14: Comparison of Average CPU Times (in seconds) with 100,000 Cells with 0.5% Sensitive Cells

Figure 19 gives a summary of the average runtimes for each of the datasets. The graph shows that HeurGene's CPU execution times were consistently low across each of

the datasets.

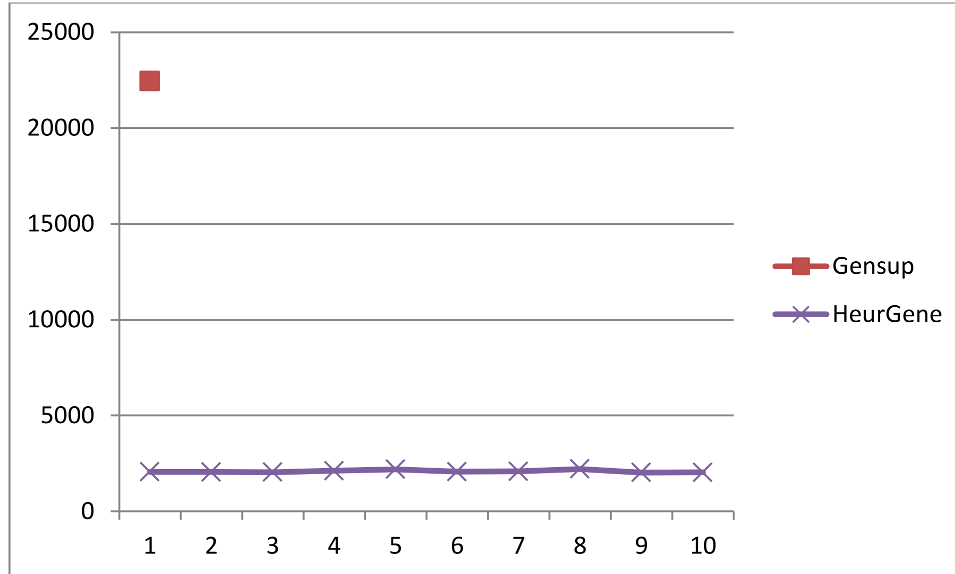

Figure 19: Comparison of Average CPU Times (Y-Axis in seconds), Using a Single Dataset, with 100,000 Cells with 0.5% Sensitive Cells

These results show that with tables of 100,000 cells and 0.5% sensitive cells, HeurGene is able to consistently keep CPU time lower across each of the datasets. An analysis of GenSup's execution times showed that for each of the datasets, it ran for 1,000 generations, the hardcoded limit for the number of allowable generations. As a result, all GenSup's run times will likely be similar for each of the datasets. HeurGene ran an average of 31 generations.

Test Results for Tables with 100,000 Cells and 1% Sensitive Cells

Table 15 shows that GenSup consistently produced the lowest average and median solution costs. GenSup's average solution cost improvement over the HyperCube was ~16% versus HeurGene's ~4%. GenSup also produced a very low variance as compared to HeurGene and HyperCube. Note that the GenSup results represent the average for a single test run on of each dataset.

	HyperCube	GenSup (1 Run)	HeurGene
Average	9418	***7868***	9065
Median	9526	***7812***	9097
Highest	10118	8379	9797
Lowest	8545	7803	8039
Standard Deviation	570	170	563

Table 15: Comparison of Average Solution Costs with 100,000 Cells with 1% Sensitive Cells

Figure 20 gives a dataset-by-dataset comparison for each of the methods. The graph shows that GenSup produced the lowest average and median cost with a ~16% average improvement over the HyperCube algorithm, while HeurGene generated a ~4% average improvement.

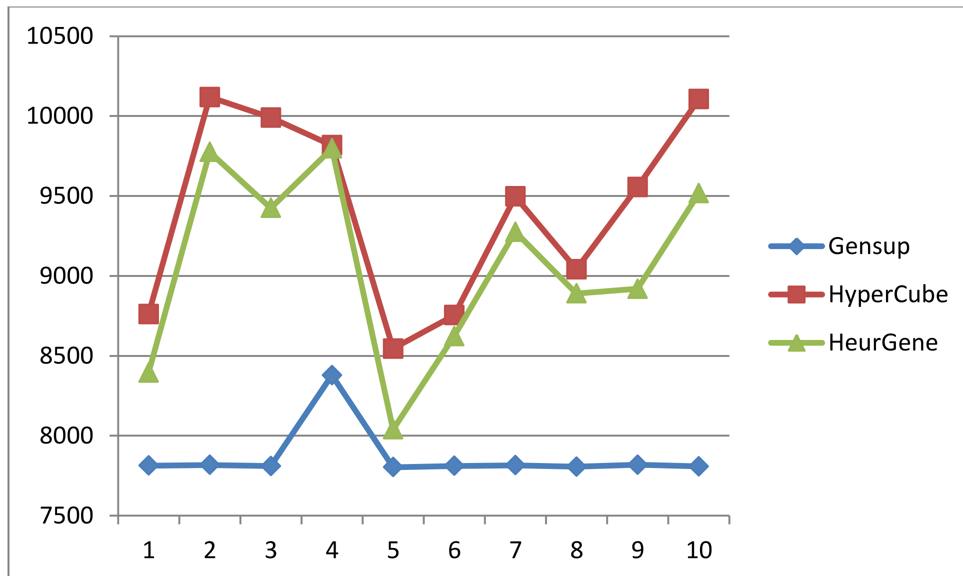

Figure 20: Comparison of Average Solution Costs (Y-Axis) with 100,000 Cells with 1% Sensitive Cells

Table 16 gives the CPU times for each of the algorithms tested. Note that due to CPU times in excess of 20 hours per dataset, the GenSup algorithm was executed only once for each of the datasets. CPU times for the HyperCube method averaged 11.39 seconds with a median time of 11.34 seconds and a standard deviation of 0.097.

	GenSup (1 Run)	HeurGene
Average	74161.45	7383.42
Median	74161.45	7325.55
Highest	74161.45	7788.72
Lowest	74161.45	7207.65
Standard Deviation		170.30

Table 16: Comparison of Average CPU Times (in seconds) with 100,000 Cells with 1% Sensitive Cells

The results presented in Figure 21 demonstrate that with tables of this type, HeurGene is able to consistently keep CPU time lower across each of the datasets. An analysis of GenSup's execution times showed that for each of the datasets, it ran for 1,000 generations, the hardcoded limit for the number of allowable generations. As a result, all of GenSup's run times were likely to be similar for each of the datasets. HeurGene ran an average of 45 generations.

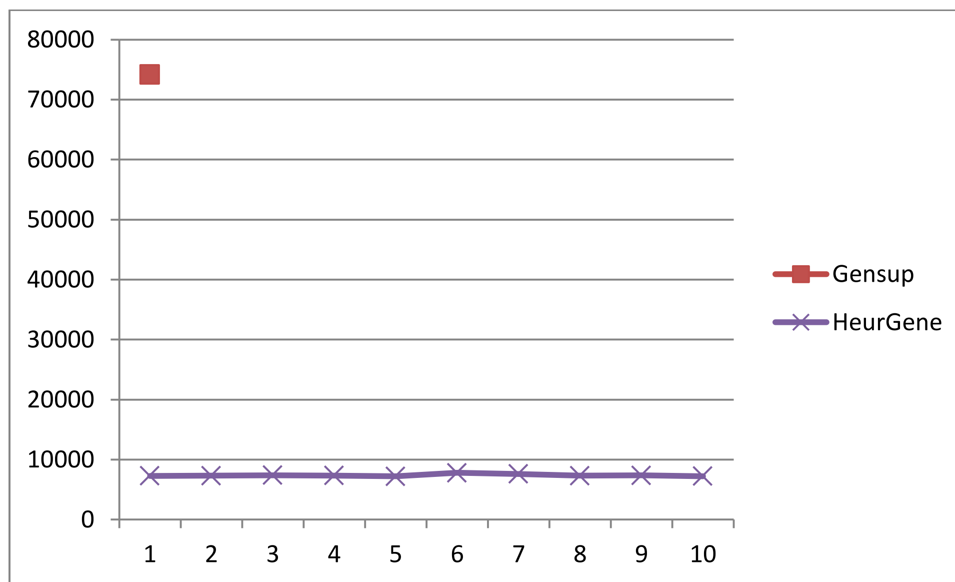

Figure 21: Comparison of Average CPU Times (Y-Axis in seconds), Using a Single Dataset, with 100,000 Cells with 1% Sensitive Cells

The results of the test at 1% sensitive cells demonstrated that HeurGene was able to make improvements in solution cost over the HyperCube solution while maintaining

low CPU time as compared to GenSup.

Test Results for Tables with 100,000 Cells and 3% Sensitive Cells

Table 17 shows that GenSup gave the lowest average solution cost from ten test runs using ten datasets. Both GenSup and HeurGene showed only modest improvement over the HyperCube algorithm with very low solution costs. This indicates that tables required a very small number of complementary suppressions to be made safe.

	HyperCube	GenSup	HeurGene
Average	145	***134***	143
Median	120	***114***	120
Highest	298	298	298
Lowest	41	41	41
Standard Deviation	78	84	78

Table 17: Comparison of Average Solution Costs with 100,000 Cells with 3% Sensitive Cells

Figure 22 gives a dataset-by-dataset comparison for each of the methods. The plot shows that for datasets 1 through 7, all three strategies returned the same solution cost. This reflects the fact that neither GenSup nor HeurGene improved on the HyperCube's solution. Dataset 8 required no additional suppressions.

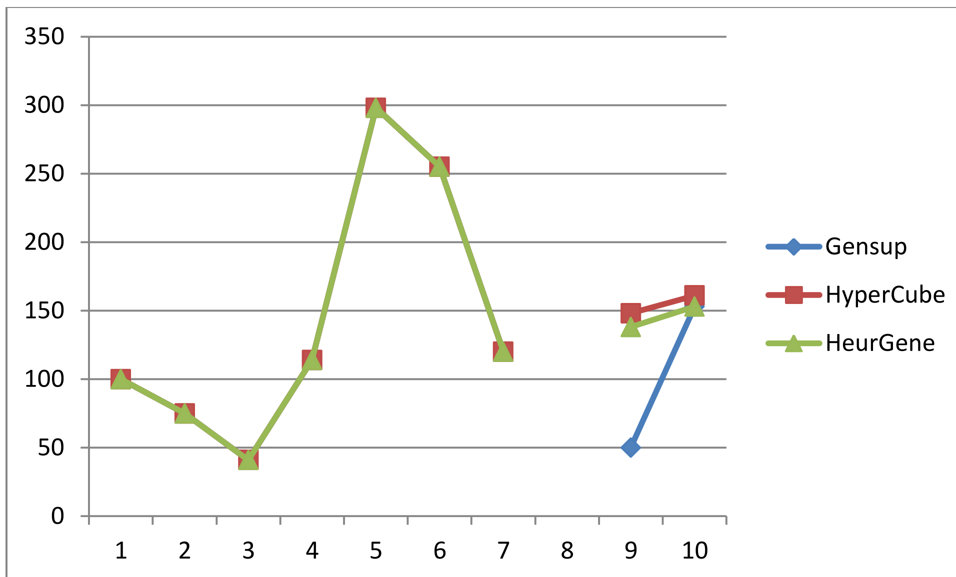

Figure 22: Comparison of Average Solution Costs (Y-Axis) with 100,000 Cells with 3% Sensitive Cells

Table 18 gives the CPU times for the GenSup and HeurGene methods. Unlike all the previous tests, GenSup yielded a lower average and median run time as compared to HeurGene. CPU times for the HyperCube method averaged 34.41 seconds with a median time of 34.38 seconds and a standard deviation of 0.09.

	GenSup	**HeurGene**
Average	***3781.76***	12434.86
Median	***3696.40***	12601.48
Highest	4582.76	13584.11
Lowest	3454.93	11444.25
Standard Deviation	343.32	721.48

Table 18: Comparison of Average CPU Times (in seconds) with 100,000 Cells with 3% Sensitive Cells

Figure 23 gives a dataset-by-dataset comparison of each of the CPU times. It shows that HeurGene required three times the CPU time of GenSup. Dataset 8 required no additional suppressions and was therefore not included in the CPU times.

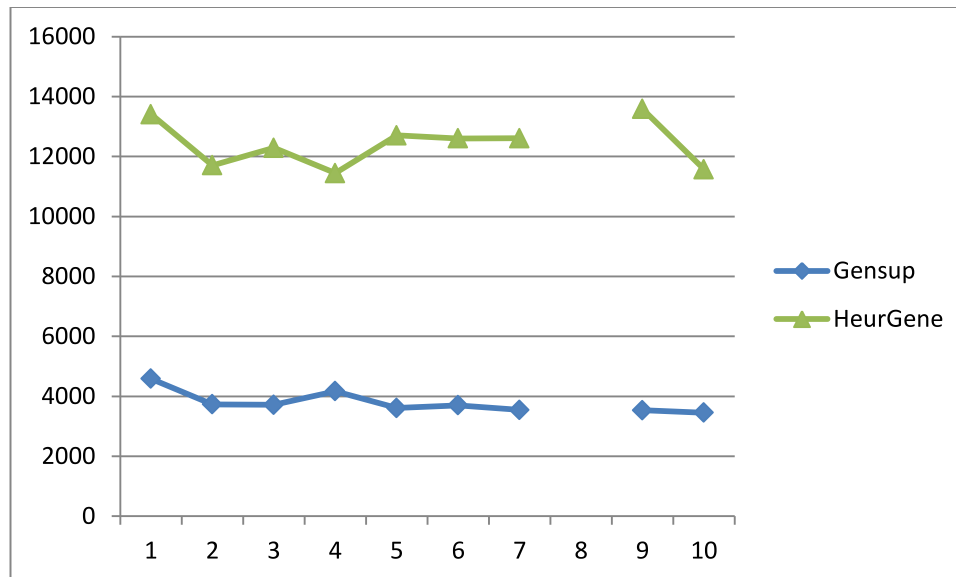

Figure 23: Comparison of Average CPU Times (Y-Axis in seconds) with 100,000 Cells with 3% Sensitive Cells

The results showed that HeurGene required three times the CPU time of GenSup and 360 times that of HyperCube without delivering significant reductions in solution cost over either strategy. The difference in CPU times reflects the nature of how the crossover routines are implemented. GenSup terminated after [100] generations without an improvement in solution cost. This represents [100] crossover operations. HeurGene also terminated at [100] generations without a cost improvement, but made 100 attempts at each generation to perform a crossover operation meeting the algorithm's heuristic's threshold requirements. This resulted in [10,000] total crossover attempts. The data suggests that HeurGene ran at or near [200] generations, or [20,000] crossover attempts, before program termination, while GenSup terminated after an average of 166 generations with 166 crossover operations.

Summary

The results of the experiments demonstrated that HeurGene was able to produce average solutions costs lower than the HyperCube's solutions and slightly better than

GenSup with tables of 10,000 cells at 0.5% sensitive and 100,000 cells at 0.25% sensitive cells. For all other percentages of sensitive cells, GenSup produced solutions of lower cost. However, GenSup's lower solution costs required ten to 20 times the CPU seconds to run to termination. A comparison of the solution cost over processing time was made using the following formula $(C_h - C_*)/t_*$, where (1) C_* is the solution cost for either GenSup or HeurGene; (2) C_h is the solution cost for HyperCube; (3) and t_* is CPU run time for either GenSup or HeurGene. This ratio gives a relative index of the reduction in solution costs compared to the HyperCube in cost per CPU second. The results for the tables at 10,000 cells are given in table 19.

Sensitive Cells	**GenSup**	**HeurGene**
3%	0.7621	4.9420
1%	3.0570	40.5636
0.5%	1.0803	29.0881

Table 19: Comparison of Improvement Ratios in Tables of 10,000 Cells

This comparison shows that HeurGene was able to provide better performance per CPU second than GenSup at all percentages of sensitive cells. The results for the tables at 100,000 cells are given in table 20.

Sensitive Cells	**GenSup**	**HeurGene**
3%	0.0031	0.0002
1%	0.0128	0.0478
0.5%	0.0836	0.3290
0.25%	0.0027	1.1336

Table 20: Comparison of Improvement Ratios in Tables of 100,000 Cells

The comparison shows that HeurGene outperformed GenSup at 0.25, 0.5 and 0.1 percentages of sensitive cells.

Chapter 5

Conclusions, Implications, Recommendations and Summary

Conclusions

The goal of this research was to develop an improved Genetic Algorithm that generated low-cost solutions without introducing excessive CPU overhead. To this end, this research investigated the use of:

1) a heuristically-directed crossover operation,
2) a crossover and mutation operation that produced few or no infeasible solutions,
3) a mutation operation that targeted specific cells for mutation,
4) a portfolio of selection and replacement strategies to increase genetic diversity.

Given the outcomes of this research, the following research questions can be addressed:

Can crossover and mutation operations be designed that produce few or no infeasible solutions?

Yes. Both the crossover and mutation operations were designed to check for complementary suppressions shared between circuits of protection in order to avoid removing them from S_2 and producing infeasible solutions. This resulted in a reduction in infeasible solutions from generation to generation.

Will this method provide for improvement in the cost of the solutions?

Conditionally. When the percentage of sensitive cells is small, 0.25% and 0.5%, the HyperCube algorithm used to generate the initial population produced sufficient genetic diversity to allow HeurGene to produce cost improvements over the Shortest

Path, HyperCube and GenSup methods. However, at sensitive cells percentages at 1% and above there was insufficient diversity in the chromosomal makeup to keep HeurGene from converging before realizing a cost improvement that was the same as or better than GenSup.

Does the computational overhead associated with the genetic algorithm negate its benefits?

No. HeurGene's use of directed, solution-preserving crossover and mutation operations reduced CPU time as compared to GenSup. However, this reduction came at the cost of premature convergence, which accounted in part for the low CPU times as compared to GenSup.

Will a portfolio of deterministic and probabilistic selection and replacement rules maintain sufficient genetic diversity to avoid premature convergence?

Not given the current configuration. The selection and replacement strategies in their current implementation were found to be unable to promote sufficient genetic diversity to prevent premature convergence. However, the same strategies in a different configuration or a configuration modified to evaluate chromosomes based on other than solution cost might prove more effective.

Preliminary Testing

Preliminary tests were conducted on tables of 10,000 cells with 0.5%, 1%, and 3% sensitive cells to determine if any one permutation of selection and replacement strategies offered an advantage. Evaluation was based on solution cost and included six permutations of selection and replacement strategies: Fitness Proportional Roulette-Wheel/Kill Tournament, Fitness Proportional Roulette-Wheel/Similar, Random

Tournament/Kill Tournament, Random Tournament/Similar, Elite/Elite and Probabilistic. The probabilistic strategy used a random number generator to select between the other five strategies at each generation.

The outcome of the tests was inconclusive as no one strategy provided significantly lower-costs. As a result, the Probabilistic strategy was selected for inclusion in this research based on its consistently producing low-cost solutions.

This research focused on four strategies: Shortest Path, HyperCube, GenSup and HeurGene. In addition, the HyperCube algorithm was used to provide the initial populations for GenSup and HeurGene. The effects of each of the major components of HeurGene on the outcome are presented:

Initial population

The initial population was generated using the same HyperCube algorithm used in the HyperCube testing. Sensitive cells were protected using a circuit of suppressions forming a cube. The order in which the sensitive cells were protected was randomized for the purpose of creating a population of genetically distinct individuals. Since the HeurGene algorithm specifically selects circuits of protection for crossover and mutation based on their potential to produce lower-cost offspring, its ability to locate suitable circuits is bound by the diversity found in the current population. HeurGene was programmed to make 100 attempts to locate suitable circuits. When this failed, HeurGene selected two new parents for mating and repeated the process. As the current population became more homogeneous, HeurGene became progressively less capable of producing lower-cost offspring and terminated after 100 mating attempts without success. This helped contribute to HeurGene’s inability to find lower-cost solutions as compared to

GenSup and also contributed to HeurGene's low CPU times.

Selection and Replacement

A portfolio of selection and replacement strategies was tested as part of this research. A random number generator was used to select the strategy. Fitness Proportional Roulette-Wheel and Random Tournament selections were each used ~45% of the time. These were paired with either Kill Tournament or Similar replacement strategies, each of which was selected for ~45% of the test runs. Elite selection and replacement was used for ~10% of the test runs. Preliminary testing found no advantage in any pairing of selection and replacement strategies. A side effect of the strategies was that it was possible for less fit offspring to replace more fit parents. This resulted in not only the loss of low-cost solutions, but also the loss of low-cost circuits of protection.

Crossover

A heuristic-based, solution-preserving, crossover operation was developed as part of this research. Crossover worked on the level of the circuits protecting the sensitive cells by crossing over entire circuits of suppressed cells. Unlike previous research, the circuits crossed could be non-cube, complex circuits with up to 32 cells. This was made possible by maintaining a list of complementary suppressions protecting each of the sensitive cells. The heuristic likewise worked on the level of the circuits, requiring that the crossover operation produce a lower-cost offspring than the lowest cost parent. Profiling of the crossover code showed a reduction in solution cost at each generation. However, this also resulted in accelerated convergence as low-cost circuits of protection were accumulated into the lowest cost members of the current population, resulting in accelerated loss of genetic diversity.

Mutation

The mutation operation has two steps: (1) to find redundant complementary suppressions created by the crossover routine and remove them from the set S_2; and (2) to merge suppressions in the sensitive cell's list of complementary suppressions, allowing for the crossover of complex circuits (circuits with greater than four suppressions) in future generations. This second goal is unique to HeurGene and allowed for reductions in solution cost with minimal CPU overhead by promoting convergence when the combining of circuits had reached saturation. Unfortunately, the use of complex circuits also reduced the likelihood of finding suitable circuits for crossover in successive generations due to an increase in the number of sensitive cells sharing complementary suppressions in their circuits of protection. This contributed to HeurGene reaching the maximum-generations-without-change-limit and terminating prematurely.

Implications

Prior to this research, no GA had been developed that examined solution-preserving, directed crossover and mutation operations that acted on circuits of suppressions protecting sensitive cells with greater than four suppressions. Additionally, this research explored a probabilistic portfolio of selection and replacement strategies to balance selective pressure with genetic diversity. From this research the contributions can be considered:

Selection and Replacement

The probabilistic portfolio of selection and replacement strategies was intended to increase genetic diversity in an attempt to lessen selective pressure caused by genetic operators and mitigate premature convergence of the GA at local optima. The results of

preliminary testing were inconclusive, demonstrating that no pairing of probabilistic selection and replacement strategies performed better than elite selection and replacement. As a result, a probabilistic approach was adapted for this research that pooled elite selection and replacement with Proportional Roulette-Wheel and Random Tournament selection paired with either Kill Tournament or Similar replacement.

The probabilistic approach selected for this research proved unable to mitigate the effects of crossover and mutation operations that increased selective pressure on the GA's population. This outcome was in part due to the nature of the initial population. A modified HyperCube algorithm that worked to form rectangular circuits of protection on individual sensitive cells was used to seed the GA's initial population. The order in which the sensitive cells are protected was re-sequenced for each individual in order to induce genetic diversity into the population. When the percentage of sensitive cells was small, $\leq 0.5\%$, the cube method was capable of producing an initial population of genetically dissimilar individuals. However, as the percentage of sensitive cells increases to $\geq 1\%$, the cube algorithm formed fewer unique circuits between individuals. Instead, circuits were formed from primarily sensitive cells, requiring the addition of less complementary suppressions. This reduced the overall number of unique circuits and negated the effect of protecting the sensitive cells based on a random ordering. As a result, the genetic diversity from individual to individual decreased. Without an initial genetically diverse population, the selection and replacement strategies were unable to introduce diversity into the current population.

This research suggests that with a large percentage of sensitive cells, $\geq 1\%$, in randomly generated populations of $> 10{,}000$ cells, if the selection and replacement

strategies are to be effective, they need to focus on the circuits protecting the sensitive cells and not on the solution cost. A method to measure the genetic dissimilarity will need to be developed to help maintain individuals in the population based on the diversity of their circuits of protection in order to preserve promising genetic sequences that might exist in less fit individuals.

Crossover

This research demonstrated that a heuristically-directed, solution-preserving crossover operation, acting on individual circuits of protection, is capable of quickly accumulating low-cost circuits protecting sensitive cells in most fit offspring through successive generations while minimizing infeasible solutions. By utilizing a sensitive cell-based list of complementary suppressions, the operator was able to crossover circuits of protection with greater than four suppressions, preserving the feasibility of the solutions in the offspring.

A heuristic was applied to the crossover that selected circuits of protection for crossover based on their ability to reduce overall solution cost. This resulted in at least one of the offspring being of lower-cost than one of its parents at each generation. However, this had the disadvantage of reducing genetic diversity. This was due to there being fewer circuits of protection available to satisfy the heuristic function at each generation. This resulted in the GA quickly reaching its maximum number of generations-without-change limit and terminating.

The result of this portion of the research suggests the need for selection strategies that focus on the circuits protecting the sensitive cells. Rather than selecting parents for the current population based on overall solution cost, it is advantageous to select parents

based on dissimilarities in the circuits of protection. This makes it more likely that the crossover operations will find circuits that satisfy the heuristic function, resulting in a successful operation. Also, further development of directed, solution-preserving crossover algorithms needs to be considered in order to completely eliminate the creation of infeasible offspring and improve solution cost at each generation.

Mutation

The mutation operation developed for this research was designed to remove unneeded complementary suppressions introduced by the crossover operation. This resulted in the new configuration of complementary suppressions forming larger circuits. The new circuits could then be used by successive generations to further lower solution costs by forming even larger, more complex circuits that would provide for lower-cost offspring. However, the creation of complex, non-rectangular circuits introduced two conditions that had a negative effect on solution cost.

First, as the number of sensitive cells in a circuit increased, it became progressively less likely that the heuristic function that directed the crossover operation would correctly evaluate the cost of the circuit and flag it as suitable for crossover, resulting in more frequent failure of the crossover operations and increasing the likelihood of premature convergence. Second, as the circuits became large there was an increased probability of the crossover operation creating an infeasible solution.

Although the mutation operation examined in this research did not perform as expected, it did indicate the types of problems that can occur with circuits protecting sensitive cells as they become large and non-rectangular. Continued research in this area could result in the correction of these problems. In addition, research on a mutation

operation that strategically adds complementary suppressions allowing for further combinations of circuits of protection may allow for lower-cost solutions.

Recommendations

The results of this research suggest multiple areas of future research. First, a survey of current heuristic approaches that provides initial populations specifically for GA's operating on large tables needs to be conducted. The focus of the research would be on producing genetic diversity and not solution cost. This research is necessary to accommodate tables where the number of sensitive cells is large and HyperCube-like algorithms provide insufficient genetic diversity for the genetic operators to substantially improve upon the parent's solutions.

Second, this research suggests the need for selection and replacement strategies that factor in genetic diversity on the level of the circuits protecting the sensitive cells, as well as the solution cost. When genetic diversity is low, the need to identify promising genetic sequences becomes acute. Selection and replacement strategies that act on the level of circuits could be more effective in mitigating selective pressure and maintaining genetic diversity.

Third, further research on solution-preserving, directed crossover operators working on the level of the circuits protecting sensitive cells should be conducted. Continued development of solution-preserving crossover operations would allow for more complex circuits of protection to be exchanged, further lowering solution costs while decreasing CPU time.

Fourth, further research on large tables should focus on heuristic-based solutions that are not dependent upon probabilistic functions. This conclusion responds to the

genetic operators' inability to adequately explore the search space presented in large tables.

Fifth, comparisons using Improvement Ratios suggest the need for research into GAs where program termination is determined by a cost-benefit ratio rather than a fixed number of generations. If the goal of the GA is to improve upon the solution cost of the initial parent population without over-committing CPU resources, the use of a ratio indicating the current improvement with respect to CPU time can be an indicator of when a predesigned point of diminishing returns has been reached.

Summary

Cell suppression can be defined as a method of Statistical Disclosure Control in which the sensitive data in a statistical table are blocked from publication by suppressing their value. This is accomplished by setting the value of the sensitive cell to null (Fischetti & Salazar, 1998).

A cell in a table is denoted by (i, j), where i is the row location and j is the column location in table T, with m rows and n columns, such that $T = \{(i, j) \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ (de Carvalho et al., 1994). A primary suppression is a sensitive cell suppressed from publication. The set of primary suppressions $(i, j) \in S_1$ is a subset of $S_1 \subseteq T$. S_1 is protected by lower and upper bounds l_{ij} and u_{ij} respectively, with a protection interval defined as $P_{ij} = [a_{ij} - l_{ij}, a_{ij} + u_{ij}]$ (Fischetti & Salazar, 1998; Almeida & Carvalho, 2005). The set of complementary suppressions is denoted by $S_2 = \{(i, j) \in A\}$ (de Carvalho et al., 1994).

A table is considered safe if each sensitive cell in S_1 is both right and left protected. S_2 is considered feasible if all cells in S_1 get protected when the values $S_1 \cup S_2$

are omitted from the table or set to null. Each cell in S_1 is assigned a weight of zero and each cell in S_2 is given a non-negative weight $w_{ij} = |a_{ij}|$, reflecting the loss of information due to suppression of non-sensitive cells. The cost of the complementary suppressions can be expressed as: $\sum_{(i,j)\in S_2} w_{ij}$ (Almeida et al., 2006). The goal of the CSP can be expressed as finding a lowest cost set for S_2 where all cells in S_1 are protected.

The primary goal of the proposed research was to develop an improved GA for the CSP that generated low-cost solutions without introducing excessive additional CPU overhead. To achieve this objective, the following primary goals needed to be realized:

1) development of crossover and mutation operators that improve upon existing methods, and
2) development of selection and replacement strategies that provide sufficient chromosomal diversity at each generation to avoid premature convergence.

Previous methods for finding a minimal set of suppressions use network flow approaches to evaluate the conditions under which a sensitive cell is considered protected. These methods typically examine one sensitive cell at a time and build a system of sub-networks to protect each of the cells. However, these methods often produce low-quality solutions due to oversuppression. The need to minimize oversuppression leads to the computationally costly post-processing of solutions. Shortest path algorithms have been used to find low-cost solutions, but come with a high runtime costs, especially for large graphs.

Genetic algorithms are typified by an initial parent population composed of chromosomal representations of a solution space and ranked by a fitness function, which allows for selection of most fit pairs for mating. Offspring are created through a process

of crossover and mutation with the more fit individuals replacing the less fit members of the parent population according to the fitness function (Russell & Norvig, 2010). The process is repeated until a stopping condition is met. The evolutionary process takes advantage of the fitter individuals produced by the genetic operators and increases their relative frequency in the population such that they are more likely to reproduce, producing fitter offspring (Smith, 2007).

The application of GAs improves upon other methods by providing lower-cost solutions with relatively low computational cost. However, the nature of a GA's crossover and mutation operations tends to disturb existing solutions, requiring that offspring undergo repair or replacement, increasing runtime costs. Most GAs lack the ability to operate on sets of cells, which prevents them from locating lower-cost solutions. After both crossover and mutation, the solutions have to be checked for feasibility and rejected or repaired if infeasible. This is due to the complexity of the sub-network of cells produced as a function of the GA's evolutionary process.

The initial population of parent chromosomes was created using a hypercube method on a two-dimensional table that formed a circuit of suppressions in the form of a rectangle (Giessing & Repsilber, 2002). Once the execution of the HyperCube code was completed, a separate function searched S_2 for redundant complementary suppressions. Redundant complementary suppressions were removed from S_2 and the solution tested for feasibility. This process provided low-cost initial populations for the GAs.

Two parents were chosen from the current population using one of the strategies from the portfolio of available selection strategies. The strategy used was selected at random and included: Elite, Proportional Roulette-Wheel and Random Tournament

selection.

A sensitive cell was selected at random and its protection circuit crossed between the selected parents. The solution cost of the resulting offspring was then compared to the goal state and the offspring either accepted or rejected. This cycle was repeated until the offspring were accepted or 100 attempts passed without success. If the crossover was successful, the offspring underwent mutation. When the operation failed, two new parents were selected for mating and the sequence repeated.

Next, a sensitive cell in the offspring was selected at random and the complementary suppressions in its circuit of protection checked for redundancy. Once a redundant complementary suppression was found, it was removed from S_2 and the new chromosomal representation $S_1 \cup S_2$ was tested for feasibility.

Replacement of members of the current population by the offspring was performed using Elite, Kill Tournament or Similar replacement strategies. The strategy used was selected at random unless the Elite strategy was used for selection, in which case the Elite replacement strategy was used.

Evaluation of the results was based on solution cost and CPU time requirements. The results of the experiments demonstrated that HeurGene was able to produce average solutions that were slightly better than GenSup with tables of 10,000 cells at 0.5% sensitive and 100,000 cells at 0.25% sensitive cells. For all other percentages of sensitive cells, GenSup produced solutions of lower-cost. However, GenSup's lower solution costs came at the expense of CPU time, with GenSup requiring ten to 20 the times the CPU time to run to termination as compared to HeurGene.

To better evaluate HeurGene's ability to produce reductions in solution cost as a

function of CPU time, a solution improvement ratio was developed. This was expressed as $^{(C_h - C_*)}/_{t_{r*}}$, where (1) C_* is the solution cost for either GenSup or HeurGene; (2) C_h is the solution cost for HyperCube; and (3) t_{r*} is CPU run time for either GenSup or HeurGene.

The improvement ratio demonstrated that, given the termination criteria for the GAs of 100 generations without cost improvement or 1,000 total generations, HeurGene was able to efficiently produce reductions in solution cost as compared to GenSup. Figure 24 demonstrates that HeurGene's cost improvement ratios outperformed GenSup's.

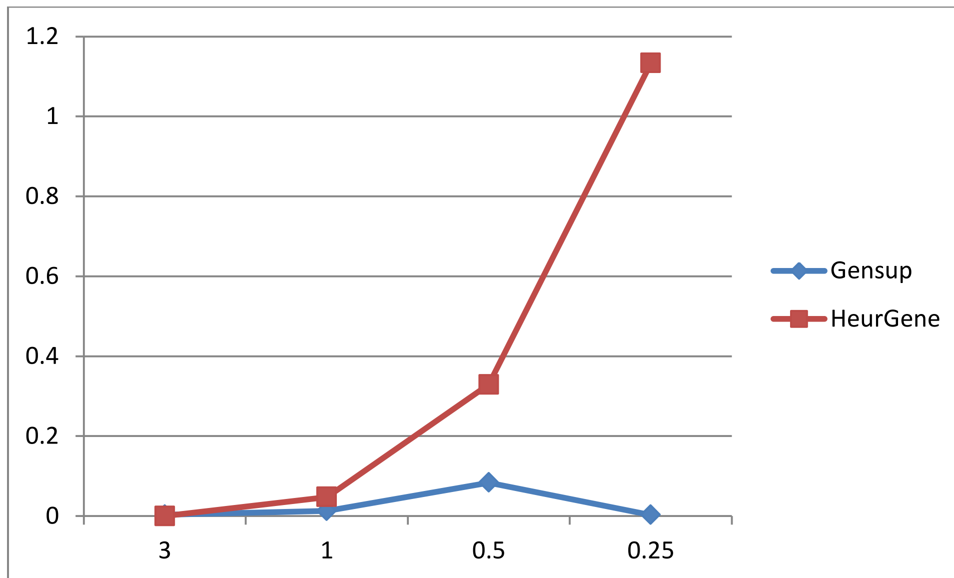

Figure 24: Comparison of Improvement Ratios (Y-Axis) at 100,000 Cells at Different Sensitive Cell Percentages (X-Axis)

The results of this research suggest the following areas for research. First, a survey of heuristic approaches, to provide genetic diversity in initial populations specifically for GAs operating on large tables, needs to be conducted. Second, selection and replacement operators that factor genetic diversity on the level of the circuits protecting the sensitive cells need to be developed. Third, further research on solution-

preserving, directed crossover operators working on the level of the circuits protecting sensitive cells should be conducted. Fourth, further research on large tables should focus on Heuristic base solutions that are not dependent upon probabilistic functions. Fifth, research needs to be conducted into GAs where program termination is determined by a cost benefit-ratio rather than a fixed number of generations.

About Appendix

Appendix A contains sample output for the Unicode files generated for each of the datasets for each execution of each method.

Appendix A: Sample Output

This section of the data file provides a summary of the runtimes, solution costs, number of generations and number of failures for each dataset.

```
File: ds10000cells01_0.txt
Run times: Fastest 10.584000, Slowest 16.504999, Average 11.841100
Costs: Highest 8625, Lowest 7224, Average 7951
Generations: Highest 110, Lowest 0, Average 11
Failures: 0
```

Figure 25: Sample HeurGene Summary Output

This section of the data file was designed to allow for easy importing of data to MS Excel for analysis. The section consists of two parts: 1) Runtimes and 2) Costs. Both parts record the fastest/highest, slowest/lowest and average run time or cost for each of the datasets tested.

```
Run times summary:
        fastest,    slowest,    average
      10.584000,  16.504999,  11.841100
       9.528000,  24.878000,  14.225200
      11.491000,  17.386000,  13.268500
      11.319000,  20.450001,  14.730200
      10.811000,  18.143000,  13.565800
      10.779000,  30.888000,  14.335000
      10.109000,  16.184999,  12.839600
      10.826000,  19.624001,  13.128900
      10.593000,  18.283001,  13.165000
      11.123000,  15.912000,  12.634700

Costs summary:
      highest,  lowest, average
         8625,    7224,    7951
         8506,    6392,    7138
         8094,    6330,    7124
         8212,    6971,    7516
         7615,    6429,    6846
         7124,    5754,    6387
         6707,    5562,    6068
         7176,    6099,    6540
         6797,    5268,    5857
         6773,    5792,    6359
```

Figure 26: Sample HeurGene Summary Data Output

Reference List

Almeida, M. T., & Carvalho, F. D. (2005). Exact disclosure prevention in two-dimensional statistical tables. *Computers & Operations Research 32*, 2919-2936.

Almeida, M. T., Schütz, G., & Carvalho, F. D. (2006). Cell suppression problem: A genetic-based approach. *Computers and Operations Research*, 1613-1623.

Castro, J. (2012). Recent advances in optimization techniques for statistical tabular data protection. *European Journal of Operational Research, Volume 216, Issue 2*, 257–269.

Cox, L. H. (1980). Suppression Methodology and Statistical Disclosure Control. *Journal of the American Statistical Association*, 377-385.

Cox, L. H. (1995). Network Models for Complementary Cell Suppression. *Journal of the American Statistical Association*, 1453-1462.

de Carvalho, F. D., Dellaert, N. P., & de Sanches Osorio, M. (1994). Statistical Disclosure in Two-Dimensional Tables: General Tables. *Journal of the American Statistical Association, Vol. 89, No. 428*, 1547-1557.

De Jong, K. A., & Sarma, J. (1993). Generation Gaps Revisited. *Foundations of Genetic Algorithms, Vol. 2*, 19-28.

Ditrich, E. J. (2010). An Evolutionary Method for Complementary Cell Suppression. Ann Arbor, Michigan, USA: ProQuest LLC.

Ewald, R., Schulz, R., & Uhrmacher, A. M. (2010). Selecting Simulation Algorithm Portfolios by Genetic Algorithms. *Principles of Advanced and Distributed Simulation (PADS), 2010 IEEE Workshop on* (pp. 1-9). IEEE.

Fischetti, M., & Salazar, J. (1998). Models and algorithms for the 2-dimensional cell problem in statistical disclosure control. *Mathematical Programming, Vol 84*, 283-312.

Fischetti, M., & Salazar, J. (1999). Models and algorithms for the 2-dimensional cell suppression problem in statistical disclosure control. *Math. Program. 84*, 283–312.

Fukunaga, A. S. (2000). Genetic Algorithm Portfolios. *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on* (pp. 1304 - 1311). IEEE.

Galan, S. F., & Mengshoel, O. J. (2010). Generalized Crowding for Genetic Algorithms. *GECCO '10 Proceedings of the 12th annual conference on Genetic and*

evolutionary computation (pp. 775-782). New York, NY: ACM.

Giessing, S., & Repsilber, D. (2002). Tools and Strategies to Protect Multiple Tables with the GHQUAR Cell Suppression Engine. *Inference Control in Statistical Databases, From Theory to Practice*, 181-192.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning.* Boston, MA: Addison Wesley Longman, Inc.

Gupta, D., & Ghafir, S. (2012). An Overview of methods maintaining Diversity in Genetic Algorithms. *International Journal of Emerging Technology and Advanced Engineering*, 56-60.

Kelly, J. P., Golden, B. L., & Assad, A. A. (1992). Cell suppression: disclosure protection for sensitive tabular data. *Networks, 22*, 397-417.

Krasnogor, N., & Smith, J. (2005). A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues. *IEEE Transactions on Evolutionary Computation, Vol. 9, No. 5*, 474-488.

Kratica, J., Čangalović, M., & Kovačević-Vijčić, V. (2009). Computing minimal doubly resolving sets of graphs. *Computers & Operations Research 36*, 2149-2159.

Lozano, M., Herrera, F., & Cano, J. (2005). Replacement Strategies to Maintain Useful Diversity in Steady-State Genetic Algorithms. *Soft Computing: Methodologies and Applications*, 85-96.

Lu, H., & Li, Y. (2008). Practical Inference Control for Data Cubes. *IEEE Transactions on Dependable and Secure Computing*, 87-98.

Mashohor, S., Evans, J. R., & Arslan, T. (2005). Elitist selection schemes for genetic algorithm based printed circuit board inspection system. *Evolutionary Computation, 2005. The 2005 IEEE Congress on* (pp. 974-978). IEEE.

Mengshoel, O. J., & Goldberg, D. E. (2008). The crowding approach to niching in genetic algorithms. *Evolutionary Computation*, 315-354.

Razali, N. M., & Geraghty, J. (2011). Genetic Algorithm Performance wit Different Selection Strategies in Solving TSP. *Proceedings of the World Congress on Engineering 2011, Vol II* (pp. 1134-1139). London, U.K.: WCE.

Razali, N. M., & Geraghty, J. (2011). Genetic Algorithm Performance with Different Selection Strategies in Solving TSP. *Proceedings of the World Congress on Engineering 2011, Vol II* (pp. 1134-1139). London, U.K.: WCE.

Russell, S., & Norvig, P. (2010). *Artificial Intelligence A Modern Approach, 3rd ed.* Prentice Hall.

Salazar-González, J.-J. (2008). Statistical confidentiality: Optimization techniques to protect tables. *Computers & Operations Research 35*, 1638 – 1651.

Smith, J. (2007). On Replacement Strategies in Steady State Evolutionary Algorithms. *Evolutionary Computation 15(1)*, 29-59.

Smith, J., Clark, A., & Staggemeier, A. (2009). A Genetic Approach to Statistical Disclosure Control. *Genetic and Evolutionary Computation Conference, GECCO'09* (pp. 1625-1632). Montréal Québec: ACM.

Vavak, F., & Fogarty, T. C. (1996). Comparison of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments. *In IEEE International Conference on Evolutionary Computation* (pp. 192-195). IEEE.