# Solving
# Combinatorial Optimization Problems
# Using a New Algorithm
# Based on Gravitational Attraction

by

Barry Lynn Webster

A dissertation
submitted to the College of Engineering at
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

Melbourne, Florida
May, 2004

Solving Combinatorial Optimization Problems Using a New Algorithm
Based on Gravitational Attraction
a dissertation by
Barry Lynn Webster

Approved as to style and content

| | |
|---|---|
| P. J. Bernhard, Ph.D.<br>Associate Professor<br>Computer Sciences<br>Dissertation Advisor | O. Frieder, Ph.D.<br>Professor<br>Computer Sciences<br>Committee Member |
| W. P. Bond, Ph.D.<br>Associate Professor<br>Computer Sciences<br>Committee Member | J. Hadjilogiou, Ph.D.<br>Professor<br>Electrical and Computer Engineering<br>Committee Member |

W. D. Shoaff, Ph.D.
Associate Professor
Computer Sciences
Department Head

# Abstract

Solving Combinatorial Optimization Problems Using a New Algorithm

Based on Gravitational Attraction

by

Barry Lynn Webster

Dissertation Advisor: P. J. Bernhard, Ph.D.

This dissertation represents the culmination of research into the development of a new algorithm for locating optimal solutions to difficult problems. This new algorithm is founded upon one of the most basic concepts in nature – so basic that it is in fact one of the four primary forces in physics: gravity.

It is called the Gravitational Emulation Local Search algorithm, or GELS. Four variants of the algorithm were developed, representing combinations of two basic methods of operation and two modes of search space exploration. Following development, a series of experiments were conducted to assess the capabilities of this new algorithm. Three test problems were used (Traveling Salesman, Bin Packing, and File Assignment). Instances of these problems were generated using several different problem sizes, then solved using three well-known comparison

algorithms (Hill Climbing, Genetic Algorithm, and Simulated Annealing) in addition to the four variants of GELS.

The outcomes of the experiments were rigorously analyzed using a variety of statistical techniques. The results of the analyses showed that GELS was able to perform on a par with, and in many cases better than, the much more mature and extensively studied comparison algorithms. One of the GELS combinations achieved the best performance ratings of any algorithm in solving instances of Bin Packing, and finished in a virtual tie with Simulated Annealing for solving instances of File Assignment and for general purpose performance. Two of the four GELS combinations were also shown to outperform Hill Climbing and the Genetic Algorithm.

GELS also performed its task efficiently. Two of the four combinations were shown to be more efficient in locating their solutions than any of the comparison algorithms except Hill Climbing (a greedy algorithm known to produce solutions in very few steps). The solutions produced by GELS were thus not only of comparable or better quality than those of the comparison algorithms, but usually were arrived at more efficiently.

# Table of Contents

List of Keywords ..... ix

List of Exhibits ..... x

List of Abbreviations ..... xv

Acknowledgements ..... xvi

Dedication ..... xviii

1 Preliminary Material ..... 1

1.1 Introduction ..... 2

1.2 Background Information ..... 3

1.2.1 Combinatorial Optimization Problems ..... 4

1.2.1.1 The Traveling Salesman Problem ..... 9

1.2.1.2 The Bin Packing Problem ..... 10

1.2.1.3 The File Assignment Problem ..... 11

1.2.2 Existing Solution Methods ..... 12

1.2.2.1 Systematic Methods ..... 13

1.2.2.2 Stochastic (Heuristic) Methods ..... 14

1.2.2.3 Advantages/Disadvantages of Method Types ..... 16

1.2.3 Statistical Validation of Research Hypotheses ..... 19

1.2.3.1 Design of Experiments ..... 19

1.2.3.2 Evaluating Results of Experiments ..... 25

1.2.3.2.1 Graphical Analysis Tools ..... 26
1.2.3.2.2 Analysis of Variance ..... 33
1.2.3.2.3 Evaluating Experimental Results Using SPSS ..... 36
1.3 A New Method for Solving Combinatorial Optimization Problems ..... 42
2 Analysis and Evaluation of the GELS Algorithm ..... 47
2.1 Preliminary Work Done With GELS ..... 48
2.2 Current Research Completed Using GELS ..... 58
2.2.1 Premises of the Research ..... 58
2.2.2 Design of the Current Research Experiments ..... 61
2.2.3 Implementation of the Test Problems ..... 66
2.2.3.1 Traveling Salesman Problem Implementation ..... 67
2.2.3.2 Bin Packing Problem Implementation ..... 70
2.2.3.3 File Assignment Problem Implementation ..... 74
2.2.4 Implementation of the Test Algorithms ..... 78
2.2.4.1 Hill Climbing Implementation ..... 82
2.2.4.2 Simulated Annealing Implementation ..... 84
2.2.4.3 Genetic Algorithm Implementation ..... 87
2.2.4.4 GELS Implementation ..... 97
2.2.5 Validation of the Experimental Environment ..... 113
2.2.6 Performance of the Research Experiments ..... 117
2.2.7 Results of the Current Research Experiments ..... 121
2.2.7.1 Algorithm Performance Results ..... 121
2.2.7.1.1 Problem Size Ten Performance Results ..... 122

2.2.7.1.2 Problem Size Twenty Performance Results.....160
2.2.7.1.3 Problem Size Thirty Performance Results.....166
2.2.7.1.4 Problem Size Forty Performance Results.....173
2.2.7.1.5 Problem Size Fifty Performance Results.....180
2.2.7.1.6 Random Problem Size Performance Results.....187
2.2.7.2 Algorithm Efficiency Results.....195
2.2.7.2.1 Problem Size Ten Efficiency Results.....197
2.2.7.2.2 Problem Size Twenty Efficiency Results.....204
2.2.7.2.3 Problem Size Thirty Efficiency Results.....210
2.2.7.2.4 Problem Size Forty Efficiency Results.....216
2.2.7.2.5 Problem Size Fifty Efficiency Results.....221
2.2.7.2.6 Random Problem Size Efficiency Results.....226
3 Research Efforts Summary and Evaluation.....234
3.1 Interpretation of Research Results.....235
3.2 Overall Evaluation of GELS.....240
3.3 Conclusion.....243
References.....245

# List of Keywords

Algorithm Efficiency

Algorithm Performance

Artificial Intelligence

Bin Packing Problem

Combinatorial Optimization Problems

File Assignment Problem

Genetic Algorithm

Heuristic

Hill Climbing

Local Search

Monte Carlo Algorithm

NP-Hard

Objective Function

Optimization Algorithm

Simulated Annealing

Traveling Salesman Problem

# List of Exhibits

**Exhibit 1.** Example Box Plot Generated by SPSS .....28

**Exhibit 2.** Example Line Plot Generated by SPSS.....30

**Exhibit 3.** Example P-P Plot Generated by SPSS .....32

**Exhibit 4.** Example Scatter Plot Generated by SPSS.....33

**Exhibit 5.** Example ANOVA Results Generated by SPSS .....36

**Exhibit 6.** Example Kolmogorov-Smirnov Test Generated by SPSS .....38

**Exhibit 7.** Example Kruskal-Wallis Test Generated by SPSS .....39

**Exhibit 8.** Example Homogeneous Subsets List Generated by SPSS.....41

**Exhibit 9.** Average Difference from Optimal, Early Experiments.....55

**Exhibit 10.** Average Improvement over Random, Early Experiments .....56

**Exhibit 11.** Average Number of Iterations per Test, Early Experiments.....57

**Exhibit 12.** Box Plot, TSP Size 10 Performance.....123

**Exhibit 13.** Line Plot, TSP Size Ten Performance.....124

**Exhibit 14.** ANOVA Results, TSP Size Ten Performance .....125

**Exhibit 15.** Residual Normal P-P Plot, TSP Size Ten Performance .....126

**Exhibit 16.** Kolmogorov-Smirnov Test, TSP Size Ten Performance .....127

**Exhibit 17.** Predicted vs. Residual Plot, TSP Size Ten Performance .....128

**Exhibit 18.** Residual Trend Plot, TSP Size Ten Performance.....129

**Exhibit 19.** Kruskal-Wallis Test, TSP Size Ten Performance .....130

**Exhibit 20.** Homogeneous Subsets, TSP Size Ten Performance .....131

**Exhibit 21.** Box Plot, BPP Size Ten Performance .....133

**Exhibit 22.** Line Plot, BPP Size Ten Performance.....134

**Exhibit 23.** ANOVA Results, BPP Size Ten Performance .....135

**Exhibit 24.** Residual Normal P-P Plot, BPP Size Ten Performance.....136

**Exhibit 25.** Kolmogorov-Smirnov Test, BPP Size Ten Performance.....137

**Exhibit 26.** Predicted vs. Residual Plot, BPP Size Ten Performance .....138

**Exhibit 27.** Residual Trend Plot, BPP Size Ten Performance .....139

**Exhibit 28.** Homogeneous Subsets, BPP Size Ten Performance.....140

**Exhibit 29.** Box Plot, FAP Size Ten Performance.....143

**Exhibit 30.** Line Plot, FAP Size Ten Performance .....144

**Exhibit 31.** ANOVA Results, FAP Size Ten Performance.....145

**Exhibit 32.** Residual Normal P-P Plot, FAP Size Ten Performance.....146

**Exhibit 33.** Kolmogorov-Smirnov Test, FAP Size Ten Performance.....147

**Exhibit 34.** Predicted vs. Residual Plot, FAP Size Ten Performance.....148

**Exhibit 35.** Residual Trend Plot, FAP Size Ten Performance .....149

**Exhibit 36.** Homogeneous Subsets, FAP Size Ten Performance.....150

**Exhibit 37.** Box Plot, Composite Size Ten Performance.....151

**Exhibit 38.** Line Plot, Composite Size Ten Performance .....152

**Exhibit 39.** Problem Type Plot, Composite Size Ten Performance.....153

**Exhibit 40.** ANOVA Results, Composite Size Ten Performance.....154

**Exhibit 41.** Residual Normal P-P Plot, Composite Size Ten Performance.....155

**Exhibit 42.** Kolmogorov-Smirnov Test, Composite Size Ten Performance.....156

**Exhibit 43.** Predicted vs. Residual Plot, Composite Size Ten Performance .....157

**Exhibit 44.** Residual Trend Plot, Composite Size Ten Performance .....158

**Exhibit 45.** Homogeneous Subsets, Composite Size Ten Performance.....159

**Exhibit 46.** Homogeneous Subsets, TSP Size Twenty Performance .....161

**Exhibit 47.** Homogeneous Subsets, BPP Size Twenty Performance .....162

**Exhibit 48.** Homogeneous Subsets, FAP Size Twenty Performance.....163

**Exhibit 49.** Homogeneous Subsets, Composite Size Twenty Performance.....165

**Exhibit 50.** Problem Type Plot, Composite Size Twenty Performance.....166

**Exhibit 51.** Homogeneous Subsets, TSP Size Thirty Performance .....167

**Exhibit 52.** Homogeneous Subsets, BPP Size Thirty Performance .....168

**Exhibit 53.** Homogeneous Subsets, FAP Size Thirty Performance.....170

**Exhibit 54.** Homogeneous Subsets, Composite Size Thirty Performance.....171

**Exhibit 55.** Problem Type Plot, Composite Size Thirty Performance .....173

**Exhibit 56.** Homogeneous Subsets, TSP Size Forty Performance.....174

**Exhibit 57.** Homogeneous Subsets, BPP Size Forty Performance.....175

**Exhibit 58.** Homogeneous Subsets, FAP Size Forty Performance .....177

**Exhibit 59.** Homogeneous Subsets, Composite Size Forty Performance .....178

**Exhibit 60.** Problem Type Plot, Composite Size Forty Performance.....179

**Exhibit 61.** Homogeneous Subsets, TSP Size Fifty Performance.....181

**Exhibit 62.** Homogeneous Subsets, BPP Size Fifty Performance .....182

**Exhibit 63.** Homogeneous Subsets, FAP Size Fifty Performance .....183

**Exhibit 64.** Homogeneous Subsets, Composite Size Fifty Performance .....185

**Exhibit 65.** Problem Type Plot, Composite Size Fifty Performance .....186

**Exhibit 66.** Homogeneous Subsets, Random Size TSP Performance.....188

**Exhibit 67.** Homogeneous Subsets, Random Size BPP Performance.....189

**Exhibit 68.** Homogeneous Subsets, Random Size FAP Performance .....190

**Exhibit 69.** Homogeneous Subsets, Random Size Composite Performance .....192

**Exhibit 70.** Problem Type Plot, Random Size Composite Performance.....193

**Exhibit 71.** Problem Size Plot, Random Size Composite Performance.....195

**Exhibit 72.** Homogeneous Subsets, TSP Size Ten Efficiency .....200

**Exhibit 73.** Homogeneous Subsets, BPP Size Ten Efficiency.....201

**Exhibit 74.** Homogeneous Subsets, FAP Size Ten Efficiency.....202

**Exhibit 75.** Homogeneous Subsets, Composite Size Ten Efficiency.....203

**Exhibit 76.** Problem Type Plot, Composite Size Ten Efficiency.....204

**Exhibit 77.** Homogeneous Subsets, TSP Size Twenty Efficiency .....205

**Exhibit 78.** Homogeneous Subsets, BPP Size Twenty Efficiency.....206

**Exhibit 79.** Homogeneous Subsets, FAP Size Twenty Efficiency.....208

**Exhibit 80.** Homogeneous Subsets, Composite Size Twenty Efficiency.....209

**Exhibit 81.** Problem Type Plot, Composite Size Twenty Efficiency.....210

**Exhibit 82.** Homogeneous Subsets, TSP Size Thirty Efficiency .....211

**Exhibit 83.** Homogeneous Subsets, BPP Size Thirty Efficiency .....212

**Exhibit 84.** Homogeneous Subsets, FAP Size Thirty Efficiency.....213

**Exhibit 85.** Homogeneous Subsets, Composite Size Thirty Efficiency.....214

**Exhibit 86.** Problem Type Plot, Composite Size Thirty Efficiency.....215

**Exhibit 87.** Homogeneous Subsets, TSP Size Forty Efficiency.....216

**Exhibit 88.** Homogeneous Subsets, BPP Size Forty Efficiency .....217

**Exhibit 89.** Homogeneous Subsets, FAP Size Forty Efficiency .....218

**Exhibit 90.** Homogeneous Subsets, Composite Size Forty Efficiency .....219

**Exhibit 91.** Problem Type Plot, Composite Size Forty Efficiency .....220

**Exhibit 92.** Homogeneous Subsets, TSP Size Fifty Efficiency .....222

**Exhibit 93.** Homogeneous Subsets, BPP Size Fifty Efficiency .....223

**Exhibit 94.** Homogeneous Subsets, FAP Size Fifty Efficiency .....224

**Exhibit 95.** Homogeneous Subsets, Composite Size Fifty Efficiency .....225

**Exhibit 96.** Problem Type Plot, Composite Size Fifty Efficiency .....226

**Exhibit 97.** Homogeneous Subsets, TSP Random Size Efficiency.....227

**Exhibit 98.** Homogeneous Subsets, BPP Random Size Efficiency .....228

**Exhibit 99.** Homogeneous Subsets, FAP Random Size Efficiency .....229

**Exhibit 100.** Homogeneous Subsets, Composite Random Size Efficiency .....230

**Exhibit 101.** Problem Type Plot, Composite Random Size Efficiency .....231

**Exhibit 102.** Problem Size Plot, Composite Random Size Efficiency.....232

**Exhibit 103.** Summary of Algorithm Comparison Rankings.....236

**Exhibit 104.** Averages of Algorithm Comparison Rankings .....237

**Exhibit 105.** Algorithm Selection Suggestions .....238

# List of Abbreviations

| | |
|---|---|
| ANOVA: | Analysis of Variance |
| BPP: | Bin Packing Problem |
| DOE: | Design of Experiments |
| FAP: | File Assignment Problem |
| GA: | Genetic Algorithm |
| GELS: | Gravitational Emulation Local Search |
| HC: | Hill Climbing |
| MC: | Monte Carlo |
| OF: | Objective Function |
| RCBD: | Randomized Complete Block Design |
| SA: | Simulated Annealing |
| SPSS: | Statistical Package for the Social Sciences |
| SQL: | Structured Query Language |
| TA11: | GELS – Method 1, Movement Type 1 |
| TA12: | GELS – Method 1, Movement Type 2 |
| TA21: | GELS – Method 2, Movement Type 1 |
| TA22: | GELS – Method 2, Movement Type 2 |

# Acknowledgements

I would like first to thank my major advisor Dr. Bernhard for taking me on as a student, for his availability for the many meetings (sometimes on rather short notice), for his encouragement, his trust, and his assistance, without which this dissertation could not have become a reality.

I would like to thank next the members of my committee, both collectively and individually:

- To Dr. Hadjilogiou, for his willingness to join the committee after only the briefest of introductions
- To Dr. Bond, for his insight at the beginning of this endeavor into how the experiments should be conducted
- To Dr. Frieder, for his willingness to serve on a committee of mine not once but twice, for his encouragement throughout this long process, and certainly not least, for his ever-present wry humor

I would also like to thank Dr. Wade Shaw for his guidance in setting up the experiments and ensuring that the analysis of the results was done properly.

A note of thanks is also due to my supervisor Gary Smith and my co-workers Paul Geneczko, Steve Grant, Liz Price, Jeff Straehla, Scott Strmel, and Frank Van Langen, who put up with me being extremely distracted and frequently incoherent

during the many months that this work was progressing, and who also gave me many words of encouragement.

Thanks should also be extended to my parents Dewey and Winifred Webster, who both being school teachers instilled in me the value of an education, and to my late grandfather Dr. Myron Webster, who inspired me to go this far with that education.

Lastly, I would like to offer thanks to my wife Pawinee, who endured my seemingly endless hours of absorption in this effort without complaint, who gave me her unwavering support, and who took care of many of those “nuisance” items usually referred to as “Real Life” when I was off in my other world, that of completing this endeavor.

# Dedication

Normally, one would expect a dedication to go to one individual, or at most a small group of persons. However, there were many people who provided assistance and/or inspiration to complete this task, and in many ways. To give a dedication to any one of them would seem to elevate that one to a position of greater importance, and thus somewhat diminish the roles played by the others. Furthermore, there was a considerable amount of adversity encountered during this odyssey, and there were many temptations to decide that it was just not worth the trouble. The drive to see it through to completion came from a variety of sources, but in the end it came down to sheer willpower. Therefore, this work is dedicated to every person who had a hand in making it a reality, and to everyone who, facing adversity in pursuit of a lofty goal, will nonetheless find the will to succeed.

# 1 Preliminary Material

## 1.1 Introduction

When examining the types of problems that computers are undertaking to solve, one cannot help but notice the sheer size and complexity of some of those problems. From the very beginning, as the computational power and capabilities of computers have increased, so have the size and complexity of the problems that have been assigned to them. With each passing year, more and more problems that once were considered too large and/or too involved to be solved in a reasonable manner via computer are becoming viable, if not routine.

Yet even so, there remain many instances of problems that are very difficult to solve, even for powerful computers. These problems are sufficiently large and complex that it may not be feasible to produce a solution within a reasonable amount of time. Now, it is true that what constitutes a "reasonable amount of time" is subjective. For some applications, a solution that appears within several days may be acceptable (Harrell et al., 2000). For other applications, such as a real-time business intelligence provider, a solution appearing after only a few seconds have elapsed may be considered too slow (Hewlett-Packard, 2002). In any case, problems can be found that will tax the ability of the computer to provide a solution within a timeframe that meets the situation's standards of "reasonable" (Harrell et al., 2000, Hewlett-Packard, 2002, Kondrak et al., 1995). It is precisely those types of problems that are of interest herein.

This dissertation is the result of research into the development of a new algorithm to locate solutions to difficult problems in a reasonable amount of time. It is organized to outline the course of that research. In the first part, concepts necessary for an understanding of the development and evaluation of the new algorithm are presented. In the second part, the algorithm itself is discussed, from its beginnings to the present, with a focus on the results of a series of experiments conducted in order to determine the ability of the algorithm to accomplish its intended purpose. In the final part, the results are tied together and conclusions are formulated from the data. The document ends with a discussion of the direction in which this research is heading and a few concluding remarks.

## 1.2 Background Information

In discussions regarding this research, it was noted that a sizeable number of people, both without and within the computer sciences community, were at least somewhat unfamiliar with one or more of the concepts involved in the research. Certain of these concepts are critical to an understanding of the research material, and to an ability to place the disparate portions of the material into their proper context. Therefore, it was deemed prudent that there should be included herein a discussion of those critical concepts.

It is not intended that this discussion should be comprehensive. To make such an attempt is neither necessary nor feasible. Rather, for each concept only the

items that are directly pertinent to the research material will be discussed, and only to the level they were applied to that material.

First will be a discussion of combinatorial optimization problems. This is the class of problems that formed the focus of the research, the class of problems for which the new algorithm was designed to locate solutions. Differing methods currently in place for solving these problems will be mentioned, along with the advantages and disadvantages of each.

Next will be a discussion of some statistical methods that can be used to assist in drawing conclusions from experimental data. Various techniques for analyzing data and presenting conclusions, both visual and mathematical, will be identified. A brief tutorial will follow showing how each of the techniques is implemented by SPSS, the statistical package that was used to analyze the research data.

Finally, the new algorithm will be introduced. Its origins will be outlined, along with early experimentation leading up to the current set of experiments.

### 1.2.1 Combinatorial Optimization Problems

This research revolved around the development and design of a new algorithm. Yet, algorithms are designed to solve problems. Some algorithms are designed to solve specific problems, as with control mechanisms for various types of automated machinery (Harrell et. al., 2000). Other algorithms are designed to solve more general classes of problems, as with linear programming (Papadimitriou, 1994).

So, there must of necessity be a problem or class of problems for which the new algorithm was designed.

One of the most basic problems under study within the computer sciences is the *search problem*. This is actually a general class of problems with a similar goal. Simply put, that goal is to find something of interest within a designated search area. To state the problem formally,

Given: set S

key value k

SEARCH(S, k) = x, such that x is a pointer to an element of S with a value equal to k, or NIL if no such element exists within S (Corman et al., 1991).

That is, given a set of elements to search through, called the *search space*, and a particular value to search for, SEARCH will attempt to locate an element of that value within the search space. If such an element is found, SEARCH will return the location of that element within the search space. If no such element can be found, SEARCH will return a null value or otherwise indicate that the search failed.

Of interest herein is a special case of the more general search problem known as the *optimization search problem*, or *combinatorial optimization*. This too is a class of problems with a similar goal, and since it is a special case of the search problem, that goal is still to find something of interest within a given search space. However, with combinatorial optimization the "something of interest" is more restrictive than in the general case. Note that there may be multiple elements within the search space that satisfy the search condition. In the generalized search

problem, locating any one of those elements is sufficient. In combinatorial optimization, though, simply locating any element that satisfies the search condition is not enough.

Combinatorial optimization tacks on the additional requirement that the search locate an element that not only matches the search condition but is in some sense the "best" element that matches the search condition out of a possible many that satisfy said condition. The notion of which element is "best" is determined by what is known as the *objective function*.

As its name implies, the objective function is a function that can be applied to elements within the solution space to determine their relative ability to attain a specific objective. For example, suppose that the search space consists of all integers $i$ such that $0 < i < 101$. Suppose further that the objective function that is to be applied is $F(x) = x$. That is, the value of the objective function is simply the value of the element itself. For the purposes of this example the objective is defined to be maximizing $F(x)$.

Now, given that the search condition is to locate an element $e$ such that $e \bmod 10 = 0$ (i.e. $e$ is evenly divisible by ten), it is easy to see that the search condition will be satisfied by more than one element in the search space; in fact, it will be satisfied by ten different elements. If generalized search is being performed, then finding any one of those ten will suffice. However, since optimization search is being performed, it is necessary to apply the objective function to each element found matching the search condition (such an element

being called a *solution* to the problem). For this example, it can be seen that doing so will result in the element with the value 100 being declared the optimal solution, since it satisfies the search condition and has the highest value of the objective function F(x) of all the solutions.

From this example, it should be fairly obvious that combinatorial optimization is more complex than generalized search. Whereas with generalized search it is necessary only to find one solution, with combinatorial optimization it is necessary to find all possible solutions (either explicitly or implicitly), applying the objective function to each in order to determine which solution is optimal. The problem is further complicated by the fact that in many instances the search space is so large that searching every element to find all possible solutions is highly impractical, if not impossible (Freuder et. al., 1995). Also, the search space may contain many solutions that have equal or near-equal values for the objective function, and these solutions may be widely dispersed throughout the search space. Lastly, the solutions may consist of several parts, and there may be a complex interplay between the parts, meaning that the overall value of the objective function often cannot be determined by just looking for patterns in the solutions (Aarts and Lenstra, 1987).

It is precisely this type of problem that is of interest in this document: the class of problems where

- An optimal solution is desired, but where the search space is large enough that performing an exhaustive search is not an option.

- There is no guarantee that the optimal solution or solutions will be located near another solution or solutions of similar quality
- The solutions are multi-dimensional (contain more than one part)

Such problems constitute an interesting and challenging area of study. Combinatorial optimization appears in a variety of situations, from academic problems used to teach the principles of optimization to problems of everyday interest in industry. It is also used extensively in Artificial Intelligence (AI) applications. Search in and of itself is fundamental to the study of AI. It is so fundamental, in fact, that it has been said that almost all the basic methods used by AI applications are some variation of search (Newell et. al., 1977). Combinatorial optimization in particular is used by expert systems to choose the best course of action to take (Englemore and Morgan, 1988, Giarratano and Riley, 1993, Webster, 1995), by Constraint Satisfaction Problems to find the best solution that satisfies all constraints of the problem (Prosser, 1993, Tsang, 1996), and by intelligent scheduling systems to find the best out of a list of feasible schedules (Bresina, 1996, Bresina et. al., 1994). Conducting in-depth research on all combinatorial optimization problems is obviously beyond the sphere of a single research study, and so the scope needed to be narrowed to a few specific examples. The example problem types that were included in the research studies will now be described.

#### 1.2.1.1 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a very thoroughly studied problem that is popular in the classroom for introducing optimization and the challenges it presents. The idea of this problem is to imagine a salesman who is about to embark on a sales trip. There are a certain number of cities on his proposed itinerary, and he needs to visit all of them before returning to his home city. The restriction is that he is only allowed to visit each city once during his trip. Thus, TSP is a variation of a *Hamiltonian cycle* problem (Aho et. al., 1983).

There is defined to be some sort of cost involved in traveling between any two given cities on the itinerary. This cost may be thought of in monetary terms, as in the cost of an airline ticket to fly between the two cities, or in terms of the time it will take to travel between the cities, or some other kind of cost. It really doesn't matter; the important thing is that some type of cost will be incurred to the salesman as he moves from one city to another, and the cost to go from city A to city B might not be the same as it is to move from city B to city A.

Faced with a directive from his management to control costs of sales trips, the salesman wants to be as efficient as he can when planning his trip. His goal is therefore to minimize the total cost of completing his itinerary. Thus, a solution for TSP is a complete tour itinerary for the salesman, showing the order in which each city will be visited. The objective function value of the solution is the sum total of

costs to visit all the cities on the complete tour, with the objective of minimizing that value.

The salesman is presented with many choices. He may visit the cities on his itinerary in any order, but he has to visit them all and may only visit each one once. He may travel between any pair of cities, but travel between some pairs is more expensive than others. This multitude of options makes TSP a very complex problem to tackle (Aho et. al., 1983).

#### 1.2.1.2 The Bin Packing Problem

The Bin Packing Problem (BPP) is another very thoroughly studied problem that is popular in the classroom. There are a number of variations to this problem, but the one that is of interest herein is *1-Dimensional* Bin Packing. The premise of this problem is that there is a collection of objects of varying size, with the one restriction that all objects are less than a specified size *n*. There is also available an unlimited supply of bins of size *n* in which to store the objects. Each of the objects in the collection is to be placed in one of the available bins. Any number of objects may be placed in a particular bin, provided the combined size of the objects does not exceed the size of the bin.

The goal of the problem is to find the minimum number of bins required to store all the objects in the collection. Thus, a solution for BPP is an assignment of objects to bins, and the objective function value for the solution is the number of

bins that have been used to store the collection, with the objective being to minimize that value.

This simple problem, with its simple goal, is actually rather deceptive in its simplicity. When placing the objects into bins, there is no restriction on how the objects may be combined, save that of not exceeding the bin size. Many options are available, meaning that the choice of bin in which to place each object is far from trivial (Corman et. al., 1991).

#### 1.2.1.3 The File Assignment Problem

The File Assignment Problem (FAP) is of everyday interest to the business community. This problem supposes a computer system that has a certain number of hardware devices available for storage, each with its own performance characteristics. The usage of the system requires that a certain group of files will need to be stored, and will be accessed in a variety of manners (e.g. differing orders, read vs. write, etc.). There are a variety of costs associated with this process, including the cost of accessing a file on a given device, the cost of (potentially) maintaining copies of a file on different devices and keeping them consistent when the file changes, and the cost of coordinating the operations of the hardware devices (Dowdy and Foster, 1982).

The idea in this problem is to allocate each of the files that will be accessed to one or more of the available hardware devices in such a way as to minimize some

pre-determined cost (average access time, throughput, etc.). Thus, a solution for FAP is a mapping of files to devices, and the objective function value for the solution is a measurement of some cost associated with storing the files in that configuration, with the objective being to minimize that value.

FAP is of interest in many common settings. It is of interest to system managers, who want to know how best to arrange their systems to maximize performance for their users. It is also of interest to database administrators, who want to know how to store their data files to maximize the performance of their databases (Bernhard and Fox, 2000). Since the problem can be expressed in a number of variations and employed with a number of objective functions, its uses can be quite extensive. Since the problem also involves storing a group of files on a group of devices, and the many ways in which that could be accomplished, FAP is also a very complex problem (Bernhard and Fox, 2000).

### 1.2.2 Existing Solution Methods

There are currently a great many methods that are used to solve combinatorial optimization. One need only examine the ever-growing list of literature to see the collection of methods that have been offered for solving TSP alone. Once again, to attempt a comprehensive discussion of these myriad methods would be neither necessary nor prudent. As such, an overview of the general types of approaches

that are in use and some basic explanations will be offered, in order to compare and contrast these methods with the new algorithm under study.

#### 1.2.2.1 Systematic Methods

Systematic methods for solving combinatorial optimization problems are searches that follow a specified step-by-step procedure to systematically go through the search space until either the search has located its quarry or it becomes clear that the search cannot succeed. By utilizing a systematic method, the search has a guarantee that if a solution to the given problem exists within the search space, the method will locate it – a property known as *completeness* (Pearl, 1985). Systematic methods can be further subdivided into *uninformed* and *informed* methods.

Uninformed systematic searches presume no knowledge of the search space or the data within, and proceed solely on the basis of the proscribed procedure (Pearl, 1985). Examples of uninformed systematic search include:

- **Linear** – progresses through the search space entry by entry, treating it as if it was one long line of potential solution values
- **Backtracking** – performs a depth-first or breadth-first search of the search space, moving along a given path until a solution is found or a dead end is reached, then backing up and moving on to the next path
- **Forward Checking** – similar to backtracking, except that at each step of the path a check is performed to see if the next step in the path would

contribute to a viable solution, or whether that path does not contribute and should be abandoned in favor of another path

Informed systematic search is used in cases where something is known about the search space and/or the data within (Pearl, 1985). For example, the data might be sorted alphabetically, or might be arranged according to some other known organization. Informed search can take advantage of such situations to improve the search performance. Examples of informed systematic search include:

- **Yes/No (Binary Tree) Search** – at each step in the search path a decision is made to follow one of two paths that will lead to a solution; works when data are sorted or otherwise organized to accommodate the binary decisions, and is very efficient
- **Hashed/Indexed Search** – uses hash functions or indexes to point the search to a specific location or region where a solution will be located; again, works on organized data and is extremely efficient
- **Domain Ordering** – uses pre-processing to order or partially order the search space to enable the use of other informed search techniques

#### 1.2.2.2 Stochastic (Heuristic) Methods

On the other side of the table are the stochastic, or heuristic, techniques. These approaches apply a *heuristic* (or "rule-of-thumb") in an attempt to guide the search towards a solution (Pearl, 1985). The heuristic will make use of knowledge

regarding the objective function (i.e. what types of things are needed to produce a solution with good objective function values) to make decisions as the search progresses that will (hopefully) locate high-quality solutions. Heuristic searches generally are *local* searches, meaning that they search a very small area of the search space and then apply the heuristic to the search results to determine the path that the search will take (Aarts and Lenstra, 1997).

Heuristic searches can either take a *generate-and-test* approach or be *repair-based*. A technique that takes a generate-and-test approach will build a solution piece by piece until a complete solution is constructed (Pearl, 1985), while a repair-based technique will start with a given solution (which may or may not be a valid solution to the problem at hand) and repetitively alter it until a valid or better solution is found (Aarts and Lenstra, 1997). Examples of heuristic search include:

- **Hill Climbing** – at each step in the search process, follows the path that adds the most value to the objective function; stops when there is no longer any path that will result in a better objective function value
- **Simulated Annealing** – technique that simulates the physical process of annealing metal. If molten metal is allowed to cool too quickly, it will develop imperfections that will weaken the intended structure, so the temperature has to be lowered in a controlled fashion. In simulated annealing, at each step in the process an attempt is made to find a path with a better objective function value, but even if one exists there is a non-zero probability that the path will not be followed and some other

path will be taken in an attempt to find an even better solution at another location. The "temperature" in simulated annealing is a pre-set value that is gradually lowered after a specified number of changes are made to the current solution (or after a specified number of iterations – there are a number of variants of the general algorithm). The procedure stops when a certain temperature value threshold is reached.

- **Genetic Algorithms** – attempts to model the process of genetics in nature. Solutions are evaluated according to a "fitness rating" that corresponds to the objective function. At each step in the process, certain solutions with low fitness ratings are allowed to "die out", while others are "crossbred" (combined) and/or "mutated" (changed) in an attempt to make them into better solutions. The procedure stops after a specified number of "generations", or iterations of the process.

#### 1.2.2.3 Advantages/Disadvantages of Method Types

As could be reasonably expected, each of the general types of approach to solving combinatorial optimization problems comes with its own set of advantages and disadvantages. As mentioned, systematic techniques have the advantage that they are complete; that is, if a solution exists within the search space they are guaranteed to find it. A disadvantage of uninformed systematic searches is that for combinatorial optimization problems, completeness entails finding not just a valid

solution but every valid solution in the search space (either explicitly or implicitly, in order to determine which is the "best"), and as has been noted, for many classes of problems the search space is simply too large to render this feasible. A disadvantage of informed systematic searches is that, while they are very efficient at locating solutions, they pay the price by requiring that the search space be organized in some known fashion. Again, for many classes of problems a search space that comes organized in this way cannot be expected, and pre-processing the search space to make it sufficiently organized would once again take far too long.

Heuristic techniques, on the other hand, have the advantage that they are able to locate high-quality solutions in a relatively short amount of time, even for search spaces of immense size. A disadvantage of these techniques is that they are not complete, which means they are not guaranteed to find the best solution. In cases where a near-optimal solution is sufficient, this is not a problem. However, in situations where it is imperative that the absolute highest quality solution be found, this disadvantage becomes an issue.

A major reason why heuristic techniques do not always find the best solution is that their termination conditions often cause them to stop in *local optima* (sing. *local optimum*). Local optima are solutions that have better objective function values than other solutions that occur nearby in the search space; that is, they are the best solutions to be found within a local *neighborhood* of the search space (Aarts and Lenstra, 1997). However, local optima are not necessarily the best solutions to be found within the entire search space. Nevertheless, because of the

way the techniques are designed they often stop in one of these local optima instead of a global optimum.

When designing heuristic search techniques, there are several important things to keep in mind. The first is to develop a solid heuristic. Employing a weak or faulty heuristic renders moot the whole point of the technique and reduces the chances of the technique performing well and finding high quality solutions. Also important is choosing a mechanism for determining the local neighborhood. Since the neighborhood is what will be searched next, the choice of neighborhood function will strongly influence the direction that the search takes.

Another item that should be considered when designing heuristic search techniques is a mechanism to allow the technique to escape local optima and settle only on a global optimum. Having a technique that is too prone to stopping at local optima reduces the chances of it finding a global optimum, thus making the technique less effective. Given the fact that heuristic techniques are by nature incomplete, and that time constraints and search space sizes often render completeness infeasible anyway, a certain amount of risk of stopping on a local optimum is inevitable. In spite of this, attempts should be made to mitigate this risk and reduce it as much as possible.

### 1.2.3 Statistical Validation of Research Hypotheses

Being able to validate conclusions drawn from research is of vital importance. Results from a series of experiments may seem to confirm a hypothesis, but if the data can be somehow bolstered by other evidence, the conclusions will be placed on a much stronger footing. Statistical methods can be used for exactly that purpose. Using appropriate statistical methods can provide a solid mathematical backing for confirming or rejecting hypotheses constructed regarding the results of research experimentation.

It is for this reason that statistical validation was desirable for use within this research. Early experiments seemed to confirm that the new algorithm under development was effective in solving certain combinatorial optimization problems (see section 2.1), but only a rudimentary analysis of the data was conducted (due to the fact that at the time it was not known if the research would continue, and so only nominal indications were desired). The later experiments done for this research, however, utilized a much more rigorous set of statistical evaluations to test the results obtained. The techniques that were used to conduct these evaluations will now be described.

#### 1.2.3.1 Design of Experiments

Design of Experiments (DOE) means exactly what it says; it is the process of designing a set of experiments suitable for accomplishing the purposes of the

research. Using a proper DOE is critical to assuring that the experiments will produce the correct results – not in terms of getting the results one wanted but in terms of getting results that are untainted by extraneous or unwanted effects (Montgomery, 2001). For example, suppose that a series of experiments are being conducted to compare two pieces of like equipment from different manufacturers. One set of tests, with the two pieces of equipment operating in a given test machine, is run first thing Monday morning by first shift personnel. A second set of tests, with the two pieces of equipment operating in a different test machine, is run Thursday afternoon by second shift personnel after the two pieces of equipment have been operating all day. The results of the tests are compiled and the results announced.

Unwittingly, though, several effects have entered into the experiments that the designers did not consider and that may have compromised the results. First, the fact that one set of tests was run using one test machine and the other using a second test machine has introduced an effect based on the machines used, since these machines were not checked to verify that they yield equal test results for the same piece of equipment. Second, the fact that one set of tests was run on equipment that was just starting up and the other after the equipment was operating for awhile has introduced a warm-up effect, since the equipment being tested may need to operate for awhile before achieving a steady-state level of performance. Third, the fact that one set of tests was run by first shift personnel and the other by second shift personnel has introduced an effect of operator, since the different

personnel may have different levels of training and experience and may not operate the equipment in the same fashion.

None of these effects represent something that is of interest to the experimenters. They are not, for instance, interested in whether there is a differing level of operator capability between first and second shift personnel, at least not at this point. However, this and the other effects may have had a substantial influence on the final outcome of the tests. Now, it may be the case that the overall results of the tests would not have changed had these effects been accounted for, but the point is that this cannot be known for sure without having accounted for them, and the test results as obtained may be spurious.

When designing experiments, usually the researcher wants to determine whether a particular item or items have an effect on an outcome of interest. These items that may or may not have an effect are known as the *factors* that are being investigated (Walpole and Myers, 1972). Each factor, in turn, will have different settings to be used for determining whether the factor does in fact have an effect on the outcome of interest. These settings are known as the *levels* for the factor (Walpole and Myers, 1972). Normally what is done is to decide what factor(s) will be examined, and then design a set of experiments to test various levels of each factor and compare the results obtained at each level.

Along with deciding what factors and levels should be included in the experiments, it should be decided as to exactly what hypothesis will be evaluated. When comparing the effects of different levels of a factor, the normal procedure is

to put forth a hypothesis that the choice of level makes no difference, i.e. the factor does not have any statistically significant effect on the outcome (Walpole and Myers, 1972). This hypothesis is referred to as the *null hypothesis*, usually rendered symbolically as $H_0$. The counterpart to the null hypothesis is the *alternative hypothesis* (usually symbolized as $H_1$), which states the converse of the null hypothesis, i.e. that the choice of level for the factor does have a significant effect on the outcome. In simple experiments there may be only one null hypothesis under examination, while in more complex experiments there may be several.

The concepts of factors, levels, and hypothesis testing can be illustrated by returning to the example of the equipment tests. This example has one factor of interest, namely the piece of equipment being tested. This factor has two levels, one for each manufacturer of the piece of equipment. There will be a single null hypothesis that will be evaluated, which will be that there is no difference between the performances of the pieces of equipment based on which manufacturer supplied them. The experiments should be designed around testing the factor at each level, or in this case testing the piece of equipment from each manufacturer. But, what of the extraneous factors? It has already been shown that there are other elements in this example that could have an effect on the outcome. These elements are not of interest to the experiments, but since they could have an effect on the outcome they are factors and must be dealt with as such.

One way of dealing with the extraneous factors is to try to eliminate them from the experiments. In the example, the test machine factor could be eliminated by ensuring that all experiments are run on the same test machine. Likewise, the warm-up factor could be eliminated by ensuring that all experiments are run only after the equipment has been in operation for awhile. Finally, the personnel factor could be eliminated by ensuring that all experiments are conducted by the same operators.

Sometimes, however, eliminating the effect of the unwanted factors may not be feasible. It may not be possible, for instance, to conduct all the experiments on the same test machine or run the equipment for extended periods to get them to steady-state due to time constraints. It may also be the case that the first shift personnel were needed for other efforts and were unavailable for the second round of tests. In such cases as these, where the unwanted factors cannot be eliminated, there are techniques available for accounting for them without removing them.

Factors that do not have their effects included in the experimental results but that have been accounted for are said to be *blocked*. Blocking factors have had their effects statistically pooled into a *block* that can then be removed from the overall effects (Montgomery, 2001). In the case of the example, it would be wise to block on the test machine, warm-up, and personnel factors to account for and statistically remove their effects from the experiments.

In order to be able to properly account for the effects of factors, it is necessary that a reasonable number of *repetitions* of the experiment be performed. Just as

flipping a coin twice might yield one head and one tail, attempting to discover the true nature of the probabilities of coin flipping would be better accomplished by increasing the number of times the coin is flipped. Normally it is not necessary to conduct thousands of repetitions of an experiment. There are statistical techniques available that can produce excellent results with only a relatively small number of repetitions (from as few as around ten to a few hundred, depending on the number of factors and levels to be considered) (Montgomery, 2001). These techniques will be discussed in the next section.

It is also wise to include the element of randomness when designing experiments. Randomness helps to ensure that the results of one experiment will not be related to the results from a previous experiment (a condition known as *correlation*) (Montgomery, 2001). In the example, it may be the case that the test machines lose a little bit of their calibration with each successive test that is run. If all the pieces of equipment from manufacturer A are tested before the pieces from manufacturer B, it stands to reason that the results for the pieces from manufacturer A will be more accurate. While this form of correlation cannot be eliminated entirely (assuming it is infeasible to re-calibrate the machines after every test), it can be mitigated somewhat by randomizing the order in which the pieces from the two manufacturers are tested. Also, while in the example the simplest way to reduce correlation might be to alternate the testing of pieces from the two manufacturers, in many other problems it might not be so simple, and thus randomization is the recommended approach. A DOE that includes one or more

blocking factors (each containing all levels of the factor to be studied) and that has been randomized in its setup is called a Randomized Complete Block Design (RCBD) (Montgomery, 2001). Such a DOE was used in the course of this research to allow comparison of the results obtained by the new algorithm with those obtained by other algorithms for the same problem instances.

#### 1.2.3.2 Evaluating Results of Experiments

As important as it is to properly design an experiment to ensure that the correct results are obtained, it is equally important to properly evaluate the results to ensure that the correct conclusions are drawn. The benefits of a well-designed experiment are lost if the results of that experiment are improperly interpreted. There are two general categories of errors that may arise when interpreting the results of experiments: Type I errors and Type II errors (Walpole and Myers, 1972). Type I errors occur when, as a result of the interpretation of the experimental results, the null hypothesis is rejected when it is actually true. Type II errors occur when the null hypothesis is affirmed when it is actually false.

Knowing whether the null hypothesis is in fact true is never a matter of absolute certainty in an experimental setting (if it could be known with absolute certainty, then there would be no need of an experimental setting to test it). Instead, the idea is to know whether an asserted hypothesis is true with a given probability. Two symbols are used: $\alpha$, which represents the probability of a Type I error and is often

referred to as the *significance level*, and $\beta$, which represents the probability of a Type II error. The normal method of evaluating experimental results is to interpret the results using a specified significance level, which is set to a value representing the acceptable amount of risk of incorrectly rejecting a factual research hypothesis (Montgomery, 2001).

There are many statistical tools available to assist with the evaluation process. The types of tools that should be employed are dependent on the nature of the experiments and the null hypothesis to be tested. For instance, if only two levels of a factor are being compared, there are some tests that are suitable for that type of comparison. On the other hand, if multiple factor levels are to be simultaneously compared, there are different tests that are suitable for that type of comparison. Also, if a statistical package is being used to conduct the analysis, the types of tools available and how they are employed will be dependent on what tools are provided by the statistical package and how they are implemented by that package. Since the experiments conducted for this research effort utilized the SPSS statistical package to conduct the analysis of the results, the tools that were used to conduct that analysis and how they are presented by SPSS will be introduced.

#### 1.2.3.2.1 Graphical Analysis Tools

Some of the simplest and most aesthetically pleasing tools for analyzing experimental results are the graphical analysis tools. By presenting a visual

depiction of the data, graphical tools give the researcher a concise overall view of the results of the experiments. Sometimes these graphical views are even enough to provide the researcher with sufficient information to draw valid conclusions regarding research hypotheses, although one must be very careful when doing so since what appears to be a significant result on a graphic may not turn out to be so when rigorous mathematical tests are applied (and vice versa). Of the variety of graphical tools available, four general types were used in this research effort.

The first type of graphical tool used was the *box plot*. A box plot is used to display summary information regarding ranges of data values for one or more categories. For each category on the plot, there is a box with one line drawn through it. The box represents the values in the range that fall between the $25^{th}$ and the $75^{th}$ percentile, also called the *inter-quartile range* (quartiles being the $25^{th}$, $50^{th}$, and $75^{th}$ percentiles of the values), with the line representing the median value of the range. On opposite ends of the box are lines that extend out to some value in the range, often referred to as the *whiskers* of the diagram. There are several variations of exactly what values are represented by the end of the lines (Montgomery and Runger, 1999), but the SPSS package denotes the ends of the lines to be the maximum and minimum values of the range, excluding outliers (*outliers* are values that are considered to be significantly rare and extreme) (SPSS, 2001). The SPSS package also plots the outliers and their values outside the box plot range lines.

Exhibit 1 shows an example box plot generated by the SPSS package. It shows seven category levels of a particular factor along the X axis, and the value range scale along the Y axis. It also shows three examples of how outlier values are displayed on a box plot by SPSS.

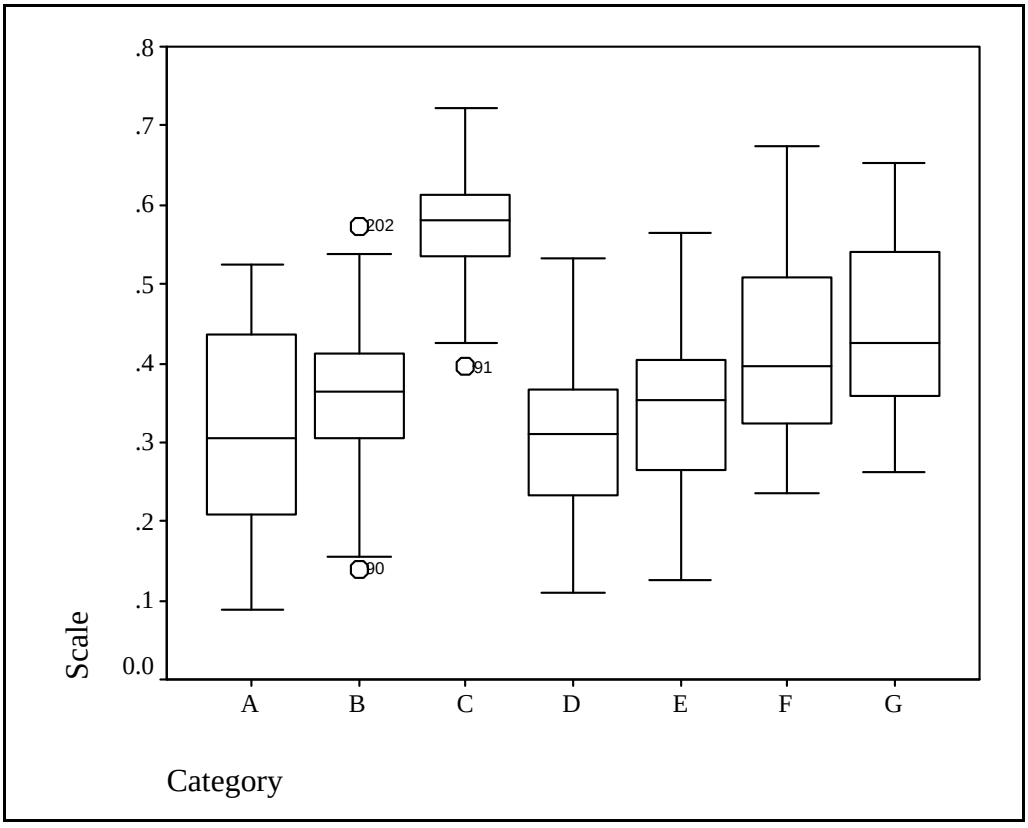


**Exhibit 1.** Example Box Plot Generated by SPSS

The second type of graphical tool used was the *line plot*. A line plot is used to show comparisons of a variable's value for two or more categories. The categories are listed along the X axis, with a scale of values along the Y axis. For each

category, a value is plotted, and then a reference line is drawn connecting the plotted points to assist in visualizing the comparison between the values. It is also possible to show the values for multiple variables in order to compare not only the values of those variables between categories, but also the values of those variables against the values of the other variables for the same category.

Exhibit 2 shows an example line plot generated by the SPSS package. It shows the comparison between values of three variables for seven categories. Different line types and point markers were used to distinguish between the lines and points for the three variables. A legend appears at the right to identify which variable uses which line types and point markers.

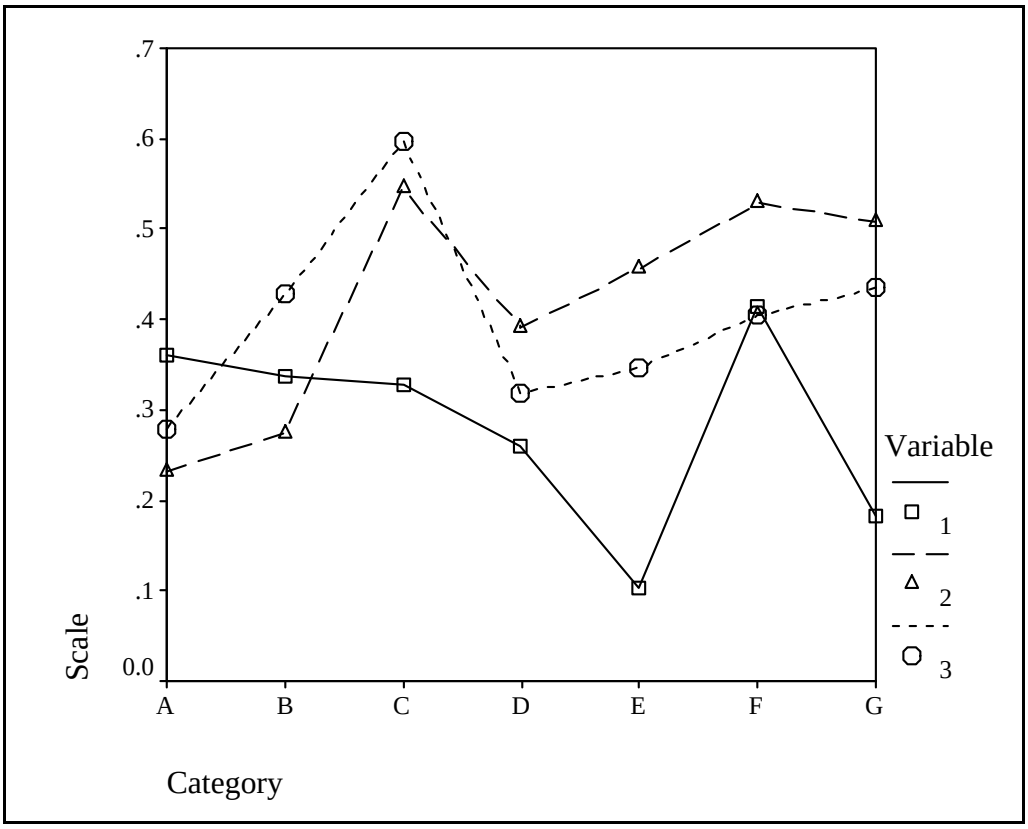


**Exhibit 2.** Example Line Plot Generated by SPSS

The third type of graphical tool used was the *P-P plot.* A P-P plot is used to show how well the observed data points of a variable match a particular type of probability distribution. Both axes of the plot are scaled in terms of *cumulative probability*, meaning that each point on an axis represents the proportion of values that occur with less than that probability (SPSS, 2001). The X axis represents the cumulative probability for the data points observed in the variable, and the Y axis the cumulative probability expected for the particular test distribution type. Points are plotted showing the relationship between the observed probabilities and the

expected. A straight line is marked on the plot showing where the relationship points should be plotted if the variable exactly matches the test distribution type. The closer the points actually are to this line, the closer the variable matches the test distribution type.

Exhibit 3 shows an example P-P plot generated by the SPSS package. In this example, the test distribution type of interest is the *normal* distribution, so the plot is showing how well the observed data points for the variable match those that would be expected for a normal distribution. The plotted points are always fairly close to the line in this example, but there are some places where they do stray somewhat. This raises an alert that the variable might not be normally distributed, and that further testing using other methods is called for to make a final determination.

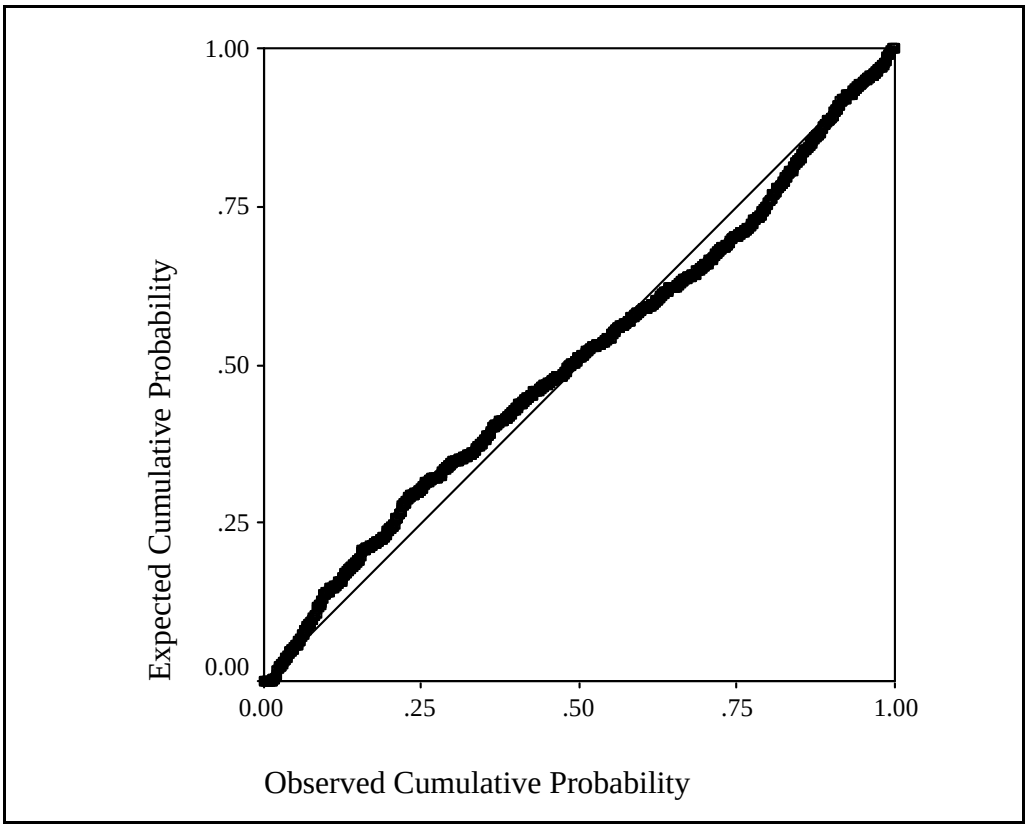


**Exhibit 3.** Example P-P Plot Generated by SPSS

The fourth and final graphical type used was the *scatter plot*. A simple scatter plot (there are other types, not used in the research effort and thus not discussed here) is used to mark one variable's values against those of another variable (SPSS, 2001). Because of this rather general nature, scatter plots can be used for a variety of purposes. As will be seen in later sections, this research effort did make use of scatter plots in a number of different ways to convey different types of information.

Exhibit 4 shows an example scatter plot generated by the SPSS package. Points are shown on the plot identifying the mapping of the values for the variable

represented by the X axis to the values of the variable represented by the Y axis. Reference lines can also be included on these plots to show a fit to the pattern of the plot, or the mean value of the plotted points with respect to one axis' values.

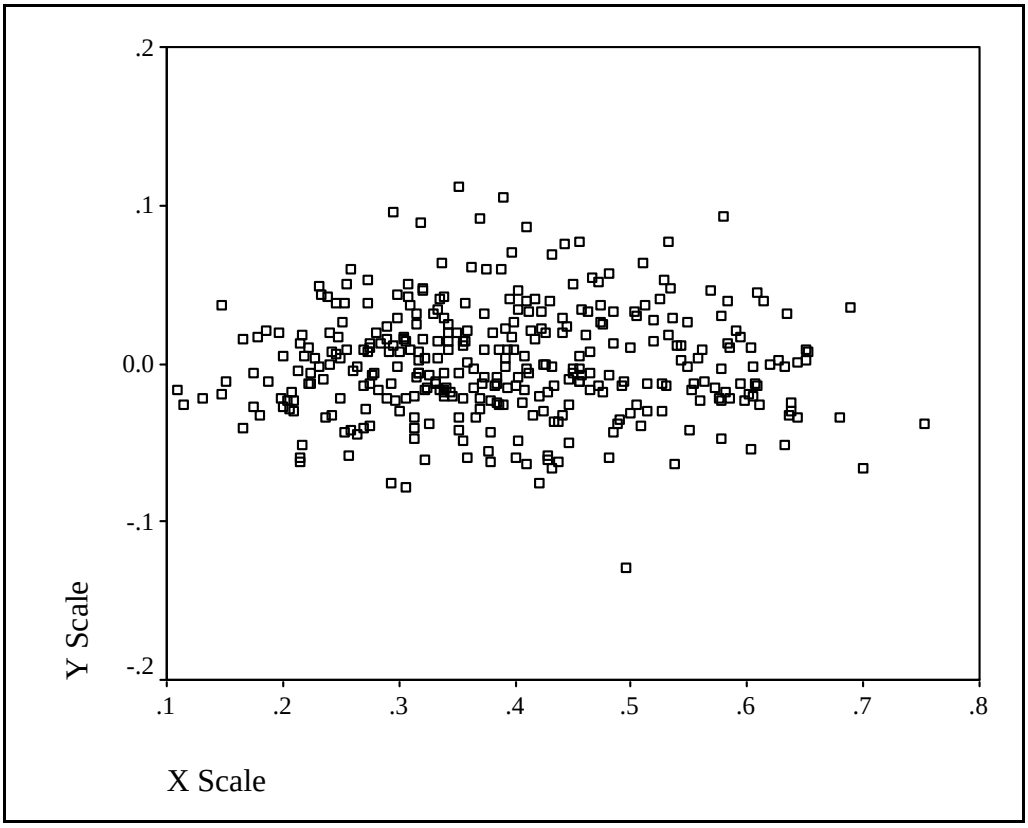


**Exhibit 4.** Example Scatter Plot Generated by SPSS

## 1.2.3.2.2 Analysis of Variance

Graphical analysis tools provide a very good means of presenting an overall summary view of experimental results. However, as mentioned earlier it is risky to develop conclusions regarding the experiments solely on the basis of the data

provided by the graphics. While graphical tools can provide good supporting evidence in support of conclusions, solid mathematical methods will render them much more concrete.

The choice of methods to use is dependent on the DOE employed since, as also mentioned previously, different statistical methods are apropos for different situations. For situations requiring a comparison of multiple factor levels, the *Analysis of Variance*, or ANOVA, serves very well. So well, in fact, that Montgomery states that ANOVA "is probably the most useful technique in the field of statistical inference" (Montgomery, 2001).

The ANOVA procedure contains a fair amount of mathematical calculations that are not germane to this discussion (and that are accomplished automatically by statistical packages anyway). The main point is that the ANOVA procedure recognizes that in any set of experiments there is bound to be a certain amount of variance within the results. This variance can stem from a variety of sources. Some of it is due to pure random error. Some of it can be due to the effects of particular factor levels. The ANOVA procedure uses its mathematical techniques to attempt to partition the variance in the experiments according to the sources that caused it. The objective is to determine if the variance attributable to the effects of factor levels is sufficiently larger than that which could be expected from random error. If so, this is a strong indication that the factor levels are having a significant effect on the outcome of the experiments.

To perform an ANOVA, a model of the experiments needs to be constructed. Included in the model should be all factors for which a comparison is desired, plus any blocking factors. Blocking factors need to be included since their effects could be significant and must be accounted for, even if those effects are not of interest. If it is suspected that there might be effects from interactions between two or more of the factors, those interactions also need to be included in the model.

Once the model has been constructed, the ANOVA calculations can be carried out. There are variations between statistical packages in how they present their ANOVA results, but generally there is a table showing a breakout of the total variance in the experiments, how much of it was attributed to each source, and some mechanism for determining which sources contributed a sufficient amount to be deemed as having a significant effect on experiment outcomes.

By examining the results of an ANOVA, the researcher is able to verify or reject a hypothesis that different factor levels have no effect on experimental outcome. The results of the ANOVA can be combined with the supporting evidence from the graphical tools to form a basis for forming solid research conclusions. The next section will show how this can be done using the SPSS statistical package.

### 1.2.3.2.3 Evaluating Experimental Results Using SPSS

When conducting experiments to compare the effects of multiple factor levels on a variable, as was done for this research effort, the evaluation of the experimental results will center on the ANOVA. Exhibit 5 shows an example ANOVA result generated by the SPSS package. The sources of variance in the experiment are listed on the left. The "Error" source represents the variance coming from random error. The "Corrected Model" and "Intercept" sources are those coming from the model itself. The "Factor 1" and "Factor 2" sources show the amount of variance due to the effects of those factors.

Dependent Variable: Performance Ratio

| Source | Type III Sum of Squares | df | Mean Square | F | Sig. | Partial Eta Squared | Noncent. Parameter | Observed Power[a] |
|---|---|---|---|---|---|---|---|---|
| Corrected Model | 6.007[b] | 55 | .109 | 28.126 | .000 | .840 | 1546.931 | 1.000 |
| Intercept | 71.094 | 1 | 71.094 | 18307.321 | .000 | .984 | 18307.321 | 1.000 |
| Factor A | 3.770 | 49 | 7.693E-02 | 19.811 | .000 | .768 | 970.750 | 1.000 |
| Factor B | 2.238 | 6 | .373 | 96.030 | .000 | .662 | 576.181 | 1.000 |
| Error | 1.142 | 294 | 3.883E-03 | | | | | |
| Total | 78.243 | 350 | | | | | | |
| Corrected Total | 7.149 | 349 | | | | | | |

a. Computed using alpha = .05

b. R Squared = .840 (Adjusted R Squared = .810)

**Exhibit 5.** Example ANOVA Results Generated by SPSS

There are quite a few numbers listed on the diagram, but the primary ones of interest are those in the column marked "Sig.". Recalling that statistical packages have some mechanism for determining which sources of variance are significant, SPSS uses the values in this column for that purpose. If a value in this column is

less than the researcher-specified significance level for the analysis, the corresponding source of variance is deemed significant (SPSS, 2001). In this example, the bottom of the diagram shows that a significance level of 0.05 (the "alpha" value) was used. Looking at the values in the "Sig." column, all are less than 0.05. Therefore, the model itself is shown to contain variance that cannot be attributed to random error alone, and thus something else must be contributing to the experimental outcomes. Since both Factor 1 and Factor 2 have values less than 0.05 they are shown to be contributors, and that their level settings do matter to the experimental outcomes.

Though the ANOVA results provide the focus of the evaluation, there are several other items that must be taken into consideration when performing the analysis. First, ANOVA assumes that the error in the experiments is random and normally distributed (Montgomery, 2001). To check this assumption, both a *Residual Normal P-P Plot* and a *Kolmogorov-Smirnov Normality Test* (Massey, 1951) can be generated. Both use the *residuals* of the experiments as their test basis. Residuals are differences between the observed value of a variable for a given factor level and its average value for that factor level, and are commonly used to check the adequacy of an ANOVA (Montgomery, 2001). An example of a P-P plot has already been shown. Exhibit 6 shows an example of the Kolmogorov-Smirnov test. The example shows the values used in the calculation of the test. The line marked "Asymp. Sig. (2-tailed)" shows the result. If this number is greater than 0.05, the test indicates that the variable being tested is normally

distributed (SPSS, 2001). In the example this value is 0.500, indicating the test variable is normally distributed.

| | | Residual for Dep. Var. |
|---|---|---|
| N | | 350 |
| Normal Parameters[a,b] | Mean | .0000000 |
| | Std. Deviation | .03467621 |
| Most Extreme Differences | Absolute | .044 |
| | Positive | .044 |
| | Negative | -.029 |
| Kolmogorov-Smirnov Z | | .827 |
| Asymp. Sig. (2-tailed) | | .500 |

a. Test distribution is Normal.

b. Calculated from data.

**Exhibit 6.** Example Kolmogorov-Smirnov Test Generated by SPSS

Another assumption of ANOVA is that the residuals should be non-structured (Montgomery, 2001). That is, when examining the residual values against predicted observation values or over time, they should not show any obvious patterns such as consistent widening or narrowing (sometimes called a *megaphone effect*). Scatter plots can be used to plot observed residuals versus predicted observation values or observed residuals versus time sequence to test this assumption. An example of a scatter plot has already been shown; the only difference would be what values are being represented by the two axes of the plot.

If the assumptions of the ANOVA do not hold, an alternative technique is available for testing hypotheses of equality of effect between factor levels. This technique is called the *Kruskal-Wallis* test (Kruskal and Wallis, 1952). Exhibit 7 shows an example of what this test looks like. The different factor levels are mathematically assigned a ranking, and the test determines whether the rankings are significantly different. The value on the line marked "Asymp. Sig." shows the conclusion of the test. If this value is less than 0.05, the factor level effects are determined to be different (SPSS, 2001). In the example, the value is 0.000, indicating a significant factor level difference.

**Ranking Test Statistics[a,b]**

| | Dependent Variable |
|---|---|
| Chi-Square | 97.397 |
| df | 6 |
| Asymp. Sig. | .000 |

a. Kruskal Wallis Test

b. Grouping Variable: Test Variable

**Exhibit 7.** Example Kruskal-Wallis Test Generated by SPSS

If the results of the Kruskal-Wallis test concur with those of the ANOVA, this indicates that the original ANOVA results are trustworthy in spite of the failed assumptions. If the results do not concur, the Kruskal-Wallis results should take precedence.

Since for many analyses of experiments (and indeed for this research effort) the goal of the analysis is to determine if there is any difference between factor levels, and since for the purposes of this research the interest was in finding optimal solutions to combinatorial optimization problems, once it has been determined by an ANOVA and/or a Kruskal-Wallis test that a particular factor does exert a significant effect on the outcome of the experiment it is important to know which levels of the factor tend to produce the best results. This can be accomplished by producing a list of *homogeneous subsets of factor levels*. This test will divide the possible factor level settings into subsets based on which level settings tend to produce results that are significantly different from other levels. That is, level settings that generate results that are statistically indistinguishable from each other will be grouped together into subsets. Using this list of subsets, it will be easy to see which level settings produce the best results.

Tukey (Tukey, 1953) and Duncan (Duncan, 1955) each developed a method for determining the homogeneous subsets, and SPSS can utilize each of these popular methods to generate a list of subsets. Exhibit 8 shows an example of a list of homogeneous subsets generated by SPSS using both of these techniques. In this example there are five factor levels for the given factor. Tukey's method assigns the five levels into four homogeneous subsets, with factor level 1 generating the highest value and factor levels 4 and 5 generating the lowest value (factors 4 and 5 being indistinguishable from each other). Duncan's method gives slightly different results, assigning each of the five factor levels into its own subset. Still, though,

level 1 produces the highest value and level 5 the lowest. So, though the Tukey and Duncan methods do not agree entirely on the subset designations, they both agree that factor level 1 will tend to produce the highest result value for the experiment, while factor level 5 will tend to produce the lowest result value.

| | | | Subset | | | | |
|---|---|---|---|---|---|---|---|
| | Factor Level | N | 1 | 2 | 3 | 4 | 5 |
| Tukey HSD[a,b] | 5 | 49 | .3203544 | | | | |
| | 4 | 56 | .3346349 | | | | |
| | 3 | 70 | | .3675620 | | | |
| | 2 | 98 | | | .4194931 | | |
| | 1 | 77 | | | | .4682279 | |
| | Sig. | | .231 | 1.000 | 1.000 | 1.000 | |
| Duncan[a,b,c] | 5 | 49 | .3203544 | | | | |
| | 4 | 56 | | .3346349 | | | |
| | 3 | 70 | | | .3675620 | | |
| | 2 | 98 | | | | .4194931 | |
| | 1 | 77 | | | | | .4682279 |
| | Sig. | | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.554E-03.

a. Uses Harmonic Mean Sample Size = 66.013.

b. The group sizes are unequal. The harmonic mean of the group sizes is used. Type I error levels are not guaranteed.

c. Alpha = .05.

**Exhibit 8.** Example Homogeneous Subsets List Generated by SPSS

By using statistical techniques such as ANOVA and Kruskal-Wallis, research hypotheses regarding equality of effect of factor levels on an experiment can be verified mathematically, giving the researcher (and the evaluators of the research) a high level of confidence in the validity of the research experiments. By further

evaluating the basis of an ANOVA to confirm its conclusions and using graphical tools as supporting evidence, the experimental results are made that much stronger. These methods were used in this research effort for just that purpose: to place the evaluation of the algorithm test experiments on a concrete foundation.

## 1.3 A New Method for Solving Combinatorial Optimization Problems

Having discussed the concepts behind combinatorial optimization problems and statistical analysis of research hypotheses, it is time to introduce the reason for which a basic knowledge of those concepts is necessary, and for which this research was embarked upon. That reason is the new algorithm developed for solving combinatorial optimization problems.

This new algorithm is called Gravitational Emulation Local Search, or GELS. As its name implies, GELS is a heuristic, local search technique. In addition, it is repair-based, and it belongs to the class of techniques that emulate some natural or physical process like simulated annealing and genetic algorithms.

GELS takes as its basis the natural principles of gravitational attraction. Gravity works in nature to cause objects to be pulled towards each other. The more massive the object, the more gravitational "tug" it exerts on other objects. Also, the closer two objects are to each other, the stronger the gravitational forces between them. This means that a given object will be more strongly attracted to a larger,

more massive object than to another object of lesser mass at a given distance, and it will also be more strongly attracted to an object close by than to another, more distant object having the same mass (Sears et. al., 1987).

GELS makes the attempt to emulate these processes of nature and use them to formulate a heuristic algorithm. The idea is to imagine the search space as being the universe. Contained within the search space are, hopefully, one or more valid solutions to the problem at hand. Each of these solutions has a "mass" that is represented by its objective function value. The better the solution's objective function value, the higher its mass. Locations within the search space that do not contain valid solutions are assigned a zero mass.

A small object represented as a pointer is moving through the search space. As it approaches a solution object, the mass of the solution object will cause the pointer object to be pulled towards it. Newton's laws of gravitational attraction are used to define how much gravitational "force" exists between the pointer object and the solution object.

As mentioned in a previous section, a heuristic technique also needs to consider how it will attempt to avoid stopping at a local optimum. GELS does this by virtue of the same principles of gravitation. In nature, when one object is being pulled by gravity towards another, the pulled object will pick up speed. In many instances, by the time the object being pulled reaches the object pulling it, it will have sufficient momentum to keep moving past the pulling object. In some instances, the pulling object's gravity will be sufficient to cause the pulled object to come

back to it, but in other instances the pulled object will move off towards other objects.

GELS attempts to avoid getting stuck at local optima by emulating this process. As the pointer object approaches a solution object, the "speed" of the pointer object increases. Once it reaches the solution (or passes by on one side or another), its speed in the original direction will decrease due to the fact that the solution object's gravity is now pulling it back the way it came. If the solution object's gravity is strong enough, the pointer object will stop altogether, terminating the procedure. But, if the pointer object's speed is sufficient, it will continue moving past the solution object. The intention is that the "momentum" of the pointer object as it moves through the search space will be such that it will be able to bypass solutions of lesser quality, stopping only on high-quality solutions with better objective function values (indeed, hopefully at a solution with the best objective function value, a global optimum).

At the time the idea for GELS was formulated, it was completely unknown as to whether it would perform at all as intended. It was not even clear that the algorithm would produce a reasonable solution at all. Gravity works very well in nature, but there are potential issues. For example, there are gravitational forces between the Earth and the Sun. However, instead of falling into the Sun, the interplay of the gravitational forces has caused the Earth to settle into a stable orbit around the Sun, continually moving instead of coming to rest. While this is very good for the inhabitants of Earth, it would not be good if the same type of event

would occur in GELS. This would mean that a solution would never be settled on and GELS would continue running indefinitely.

There were many questions to be answered, even at an initial level. Would an implementation of GELS, operating in an actual problem-solving environment, be able to produce valid solutions? If it did, what would be the quality of the solutions produced? How long would it take to find a solution? Would it be able to find a global optimum? Would it get stuck in an "orbit" around some solution without ever stopping?

The only way to answer even these most basic questions was to implement the technique and test it against actual problem instances. But there was another question that needed answering first: why bother to study this algorithm at all? There are already a host of algorithms available for solving combinatorial optimization problems, some of them with very good track records for producing high quality solutions. Given this situation, what would be the benefit of adding yet another algorithm to the mix?

The answer to this question was twofold. First, if further investigation would reveal that GELS could perform better than other algorithms, even occasionally on only one type of problem, then the question would become: why not use it? To be faced with solving a problem, having a solution method available that is likely to give the best solution, and then not using it, would seem to be illogical.

On the other hand, if further investigation would show that GELS did not perform well, the investigation would still have not been in vain. The literature is

replete with examples of algorithms that in general perform rather poorly, yet are still useful because they can be used as instruments for instruction on various concepts. For example, consider simple backtracking. This algorithm typically performs very poorly as a search method, yet even today it is used to teach principles of algorithmic procedure and systematic search. In the same manner, even if GELS would not be a top performing algorithm, it could still be used as an example of methods that emulate processes that occur in nature to solve problems, and thus the research would not have been wasted.

This, however, would be a worst-case scenario. It was never intended that the study of GELS would be pursued with the idea in mind that it would not work well. Rather, it was hoped that GELS would perform admirably, in a variety of situations. But, as stated it would not be possible to know for sure without fully developing the algorithm and putting it through some rigorous testing.

# 2 Analysis and Evaluation of the GELS Algorithm

## 2.1 Preliminary Work Done With GELS

To begin a study of the GELS method, a conceptual framework was developed, and a preliminary design was produced. At this point the algorithm was called the Gravitational Local Search Algorithm, or GLSA. This name was later changed to GELS since there are a number of places in the literature where the initials GLS are used to refer to Guided Local Search (Voudouris and Tsang, 1995), and another identifier was wanted to avoid confusing the two procedures. Two separate versions of GLSA were implemented, using the C programming language. The two versions operated in essentially the same fashion, but with two key differences. The first version, dubbed GLSA1, used as its heuristic Newton's equation for gravitational force between two objects, while the second, dubbed GLSA2, used as its heuristic Newton's method for gravitational field calculation. Additionally, in GLSA1 the pointer object moved through the search space one position at a time, while the pointer object in GLSA2 was allowed to move multiple positions at a time.

Each of the two procedures had operational parameters that the user could set to fine tune its performance. These parameters (and their four-character names as used in the procedure) were:

- Density (DENS) – represented the relative "density" of the search space. It was used as part of a calculation of "resistive" force that the pointer object would meet as it moved. This resistive force was intended to help prevent

the pointer object from never slowing down and stopping. It had a default value of 1.2 (the relative density of air) (Sears et. al., 1987).

- Drag (DRAG) – represented the "drag coefficient" of the pointer object. It was also used as part of the resistive force calculation, and had a default value of 0.5 (the drag coefficient of a relatively streamlined body) (Sears et. al., 1987).
- Friction (FRIC) – represented the "motion coefficient of friction" of the pointer object. It was also used as part of the resistive force calculation, and had a default value of 0.003 (the value for steel rolling on steel) (Sears et. al., 1987).
- Gravity (GRAV) – represented the coefficient of gravity acting between two objects, and was used only in the GLSA1 version of the procedure. It was used in the calculation of the gravitational force between the pointer object and an adjacent solution object. It had a default value of 6.672 (as appears in Newton's equation) (Sears et. al., 1987).
- Initial Velocity (IVEL) – represented the maximum permissible initial "speed" of the pointer object in any possible dimension of movement. It was used to put a bound on the initial speed of the pointer object as it began moving through the solution space when the procedure commenced. It had a default value of 10 (an arbitrary setting).

- Iteration Limit (ITER) – the maximum number of iterations that the procedure could perform for a given run before being forcibly terminated. It was used to ensure that even if the pointer object did get into an "orbit" condition as described previously, or the procedure encountered some other difficulty in finding a solution, the procedure would still terminate. If the procedure completed this number of iterations and the pointer object had not yet stopped moving, the procedure terminated regardless and returned the best solution seen to that point.
- Mass (MASS) – represented the "mass" of the pointer object. It was used in calculations where mass of an object was required. It had a default value of 1 (an arbitrary setting).
- Radius (RADI) – represented the distance between two objects, and was used only in the GLSA1 version of the algorithm. It was used in the calculation of the gravitational force between the pointer object and an adjacent solution object. It had a default value of 2 (an arbitrary setting).
- Silhouette (SILH) – represented the "silhouette area" of the pointer object as seen from the front. It was another item used as part of the resistive force calculation, and had a default value of 0.1 (an arbitrary setting).
- Threshold (THRE) – represented the threshold at which the speed of the pointer object in a given direction would be assumed to drop to zero. It was used to prevent the speed of the pointer object in any direction from

asymptotically approaching zero but not ever actually getting there due to the rounding off of numbers and the precision limits of the calculations. It had a default value of 2 (an arbitrary setting).

To test the procedure, a problem had to be identified or created. The problem needed to be simple enough that it could be easily generated and evaluated, yet still constitute a test of sufficient complexity that it would not be trivial. The problem instance that was eventually settled on had as its basis a 10x100 matrix. This matrix was populated with integer values ranging from zero to one hundred, and was to be searched for a ten-variable optimal solution. The optimality of the solution was to be decided by an objective function that consisted simply of the sum of the integer values assigned to each variable of the solution.

This problem type was chosen because it fit the bill both in terms of ease of generation and sufficient complexity. Each instance of the problem could easily be generated by randomly assigning integer values to every location within the matrix. The optimal solution could be determined during problem instance generation by keeping track of which assignments of matrix row values to variables yielded the highest sum. Yet, even though the problem instances were easy to generate and an optimal solution easy to determine, any procedure that would be used to search the matrix would not have this *a priori* knowledge and would simply be searching a large search space for a ten-variable solution. Thus, tests run using one or more of these solution procedures would be valid since there was a large space to search

and the techniques used by the solution would not be dependent on the “short cut” to finding the optimal solution.

Having chosen the problem, it was then necessary to select some solution methods to compare against GLSA. To this end, it was decided to utilize two methods. The first method would be a Monte Carlo, or random solution, whereby random assignments of elements from the search matrix would be made to each of the ten solution variables. The other method to be used would be basic Hill Climbing. The Monte Carlo solution was chosen because it was guaranteed to find a solution in time linear in the number of matrix rows, and because it would provide a good “starting point” for determining an average-quality solution that the other solution methods could then attempt to improve upon. Hill Climbing was chosen as the other method because it is a simple, well-known example of a local search technique that could serve as a basic benchmark for local search improvement over the Monte Carlo solution.

Together with the implementations of GLSA1 and GLSA2, the Monte Carlo and Hill Climbing procedures formed the test suite that would be used for the comparison tests. To complete the scenario, a set of experiments had to be devised. When creating an instance of the problem, the capability existed to specify an integer parameter that would represent the probability of non-zero entries within the search matrix. For example, by specifying a value of twenty for the parameter, each node of the matrix would have a 20% chance of being assigned a non-zero value when the matrix was generated. By adjusting the parameter value between

zero and one hundred, the relative number of occurrences of quality solutions within the search matrix could be determined beforehand.

With this capability in hand, the test scenario was set up as follows: a series of problem instances was generated, at varying levels of solution availability. Specifically, ten tests each were conducted at parameter settings ten, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, and one hundred, respectively. For each test, a problem instance was generated and the optimal solution for that instance was recorded. Then, the Monte Carlo procedure generated a solution, and the objective function value of that solution was recorded. Using the Monte Carlo solution as a starting point, the Hill Climbing procedure was then run, and the solution it generated and its associated objective function value was recorded. Finally, using the Monte Carlo solution again as a starting point, each of GLSA1 and GLSA2 was run and their respective solutions and associated objective function values were recorded. This resulted in a grand total of one hundred tests being run comparing the four methods (Monte Carlo, Hill Climbing, GLSA1, and GLSA2) against the same data sets, using the same starting points.

Approximately one dozen test scenarios were set up and run as described. The results were then collected and analyzed. The analysis showed that while none of the procedures "won" every test by generating the solution with the highest objective function value, some clearly performed better than others. As expected, the Monte Carlo procedure produced a solution very quickly (in a single step), but the solutions were generally of poor quality. The Hill Climbing procedure

generated solutions quickly (typically in two to five steps), and in almost all instances was able to improve upon the Monte Carlo solution.

While these results were interesting, of primary interest in running the test scenarios was seeing how the GLSA procedures would perform. To that end, the analysis indicated that both GLSA1 and GLSA2 were able to generate valid solutions, typically in twenty to twenty-five steps. There were a very few cases where the particular sequence of gravitational effects engendered by a problem instance caused GLSA2 to cycle through the search space back to the same point, where the sequence would repeat. This led to a potentially endless harmonic motion through the search space, and the algorithm had to be terminated by maximum iteration count.

Like Hill Climbing, the solutions generated by both GLSA1 and GLSA2 were in almost all instances better than the Monte Carlo solution. In addition, both GLSA1 and GLSA2 were in the overwhelming majority of instances able to generate solutions that were better than the solution produced by Hill Climbing. In fact, in many instances the solutions produced by GLSA1/GLSA2 were substantially better than the Hill Climbing solution. Lastly, the solutions produced by GLSA2 were on average better than those produced by GLSA1, and in a number of instances GLSA2 was able to locate the optimal solution when none of the other three methods had.

Exhibit 9 is a graph of the results of the experiments pertaining to solution quality (Webster and Bernhard, 2003). It shows the average difference between the

objective function value of the optimal solution and the objective function value of the solution produced by each of the algorithms. Since the objective function value was the sum of the assignments to each of the ten variables in a solution, and since the maximum value that could be assigned to a variable was one hundred, the maximum value of the objective function was one thousand. Thus, if the optimal solution value for a problem instance was nine hundred, and the solution value for one of the algorithm types for that problem instance was eight hundred, that algorithm registered a difference of one hundred for that problem instance. The values in Exhibit 9 represent the average such distances for each algorithm type over all problem instances tested.

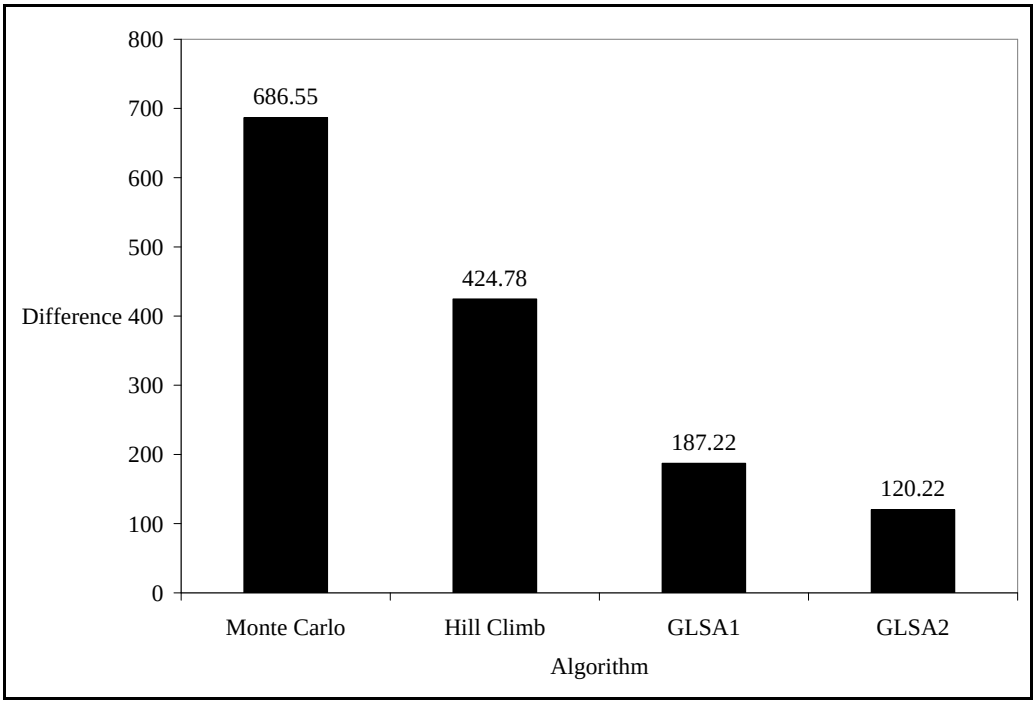


**Exhibit 9.** Average Difference from Optimal, Early Experiments

Exhibit 10 is another graph of the results of the experiments pertaining to solution quality (Webster and Bernhard, 2003). However, instead of showing the average distance of the algorithms' solution qualities from the optimal, it shows the average improvement in solution quality over that obtained by the random Monte Carlo solution. That is, if the objective function value of the Monte Carlo solution for a problem instance was four hundred, and the value for one of the algorithm types was five hundred, that algorithm posted a one hundred point improvement in solution quality over the Monte Carlo solution. Once again, the values shown in Exhibit 10 represent the average value of such improvements over all problem instances tested.

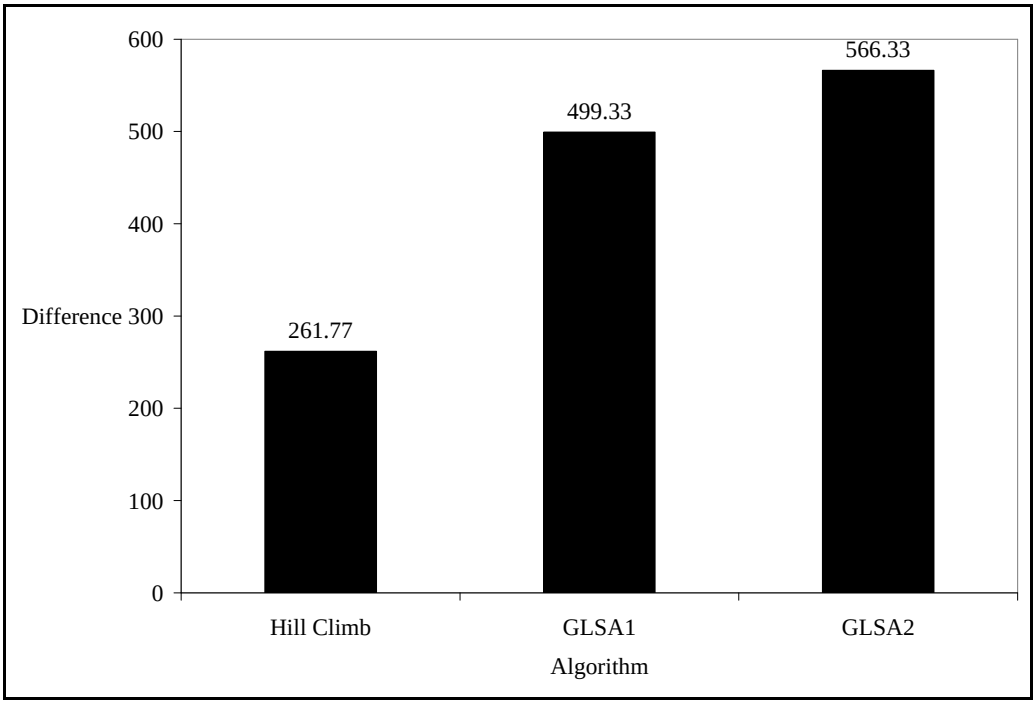


**Exhibit 10.** Average Improvement over Random, Early Experiments

Exhibit 11 is a graph of the results of the experiments pertaining to algorithm efficiency (Webster and Bernhard, 2003). It shows the average number of iterations each algorithm took to arrive at a solution over all problem instances tested. Since the number of iterations for the Monte Carlo solution was always one, it is not shown.

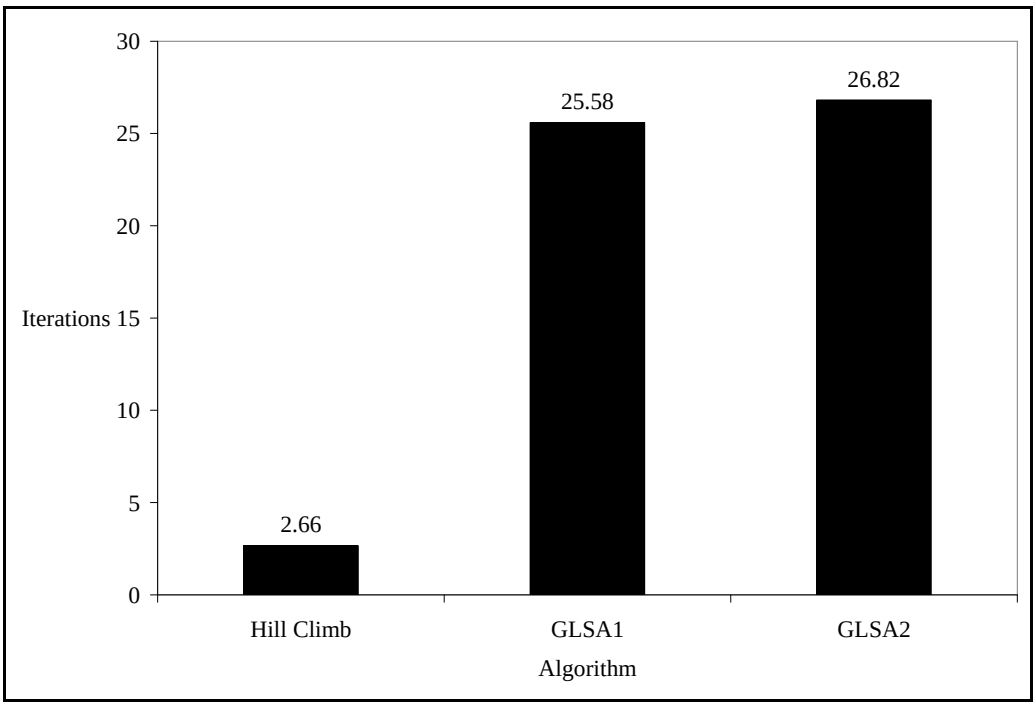


**Exhibit 11.** Average Number of Iterations per Test, Early Experiments

These early experiments with GLSA were very encouraging. The algorithm had shown, over hundreds of problem instances, that it could generate noticeably better objective function values than random solutions and Hill Climbing solutions. The improved results were obtained at a cost of an additional twenty-three or

twenty-four iterations of the GLSA algorithm (on average, with no perceptible increase in overall execution time), well worth the price.

## 2.2 Current Research Completed Using GELS

With the promising results of the early experimentation with the GELS method in hand, it was decided that research should begin in earnest on a much more rigorous set of experiments with the algorithm. This time a completely new set of experiments was designed from the ground up, one that would add the weight of statistical analysis to the raw data in addition to evaluation on multiple, difficult optimization problems. It was hoped that results could be produced similar to those obtained by the early experiments, which would then be bolstered by that statistical analysis. The details of this research, and its results, will now be described.

### 2.2.1 Premises of the Research

The first item to be decided for the new experiments was exactly what was to be tested. As discussed in the background material, a proper DOE needs at least one null hypothesis and its corresponding alternate hypothesis. The whole point of this follow-on set of experiments was to be how the GELS algorithm would perform in solving combinatorial optimization problems compared to other algorithms. Obviously, then, there should be a null hypothesis to make a statement regarding algorithm performance.

In keeping with the general format for stating null hypotheses (and the one best supported by the SPSS package), it was decided that there would be a null hypothesis (and corresponding alternate hypothesis) as follows:

$H_0$: There is no difference between the ability of the GELS algorithm to improve on the solution qualities (i.e. objective function values) of random solutions and the ability of other algorithms to improve on the same random solutions for the same combinatorial optimization problems

$H_1$: The GELS algorithm's ability to improve on the solution qualities of random solutions is significantly different from the ability of other algorithms to improve on the same random solutions for the same combinatorial optimization problems

Additionally, it was of interest to find out not only how well the GELS algorithm would perform in finding solutions, but also how efficiently it would perform in arriving at those solutions. This led to a second null/alternate hypothesis formulation:

$H_0$: There is no difference between the rate at which the GELS algorithm improves on the solution qualities of random solutions and the rate at which other algorithms improve on the same random solutions for the same combinatorial optimization problems

$H_1$: The GELS algorithm improves on the solution qualities of random solutions at a significantly different rate than the rate at which other

algorithms improve on the same random solutions for the same combinatorial optimization problems

For the sake of convenience, the first null hypothesis will hereafter be referred to as the *performance hypothesis* for the experiments, and the second null hypothesis will hereafter be referred to as the *efficiency hypothesis* for the experiments. On first glance, it may appear that these hypotheses are somewhat trivial. Specifically, how could it be reasonably expected that there is no difference between algorithms in their ability to improve on solution quality, or that there is no difference between the rate at which each algorithm achieves its results? It would seem that these hypotheses are designed to fail, that almost any set of experiments would be able to cause them to be rejected.

This is in fact partially true, but there is more to the story. It is not a problem that the hypotheses are likely to be rejected. In experiments such as these, where multiple items are being compared, the SPSS package wants to take the base view that there is no difference between the items and then try to prove that view to be incorrect (SPSS, 2001). If successful in this, SPSS can then state an ordering to the items using the homogeneous subsets tests. In this manner, if an SPSS analysis succeeds in rejecting the performance and/or efficiency hypotheses, it can also tell whether the performance/efficiency of the GELS algorithm is not only different, but better or worse than other algorithms. So in that sense, the goal is much more than to simply reject the hypotheses and state that there are differences between the

algorithms; it is also to be able to state how much difference exists, and between which algorithms.

### 2.2.2 Design of the Current Research Experiments

Having decided upon the hypotheses to be tested in the research experiments, the next step was to prepare a DOE to test them. A complete experimental environment was needed, to consist of:

- A set of combinatorial optimization problem types to use as test problems
- A set of algorithm types to use as test algorithms
- A framework within which the tests of each of the test algorithms against each of the test problems would be conducted

To select problem types to act as test problems, the goal was to choose a small representative sample that are well known and sufficiently complex to present a genuine challenge to solution algorithms. To that end, three problem types were ultimately chosen: the Traveling Salesman problem (introduced in section 1.2.1.1), the Bin Packing problem (introduced in section 1.2.1.2) and the File Assignment problem (introduced in section 1.2.1.3).

Each of these problem types easily met the criteria for selection. They are all very familiar to and extremely well studied by researchers. They are also all very difficult problems to solve, belonging to the class of problems known as *NP-Hard* (Garey and Johnson, 1979). NP-Hard problems are among the most difficult to

solve, with most researchers believing that these problems are *intractable*, meaning that there is no known solution algorithm for them that can be accomplished in deterministic polynomial time (i.e. in $N^K$ steps for any input size N and constant value of K) (Cormen et. al., 1991). Some very good approximation algorithms exist to aid in the solution of these problems under the appropriate conditions (Arora, 1998, Karmarkar and Karp, 1982, Papadimitriou, 1994). However, these algorithms do not guarantee finding optimal solutions (hence the term "approximation"), and they do not change the fact that unless it can be shown that polynomial solutions exist for NP-Hard problems, such problems will remain difficult to solve optimally.

Once the set of test problems was defined, a set of test algorithms was needed to solve them. The goal here was to establish a small collection of well known algorithms suitable for comparison with the GELS procedure. Chosen for inclusion in this collection were Hill Climbing, Simulated Annealing, and a Genetic Algorithm, all introduced in section 1.2.2.2. The Hill Climbing algorithm was selected as a representative of a greedy algorithm, and also because of its prior use in the early experiments. Simulated Annealing and the Genetic Algorithm were selected because of their popularity and because they are, like GELS, representatives of algorithms that are based at least in part on processes that occur in nature.

With the sets of test problems and test algorithms in place, the one remaining item was to design the framework within which the test algorithms would be used

to solve the test problems. This involved a number of decisions that had to be made regarding how the problems would be set up, how the algorithms would be configured to solve them, and how the comparisons between the algorithms would be conducted. Of primary importance was rendering the "playing field" as level as possible in an attempt to remove as much bias from the experiments as possible.

To accomplish this, it was decided to retain the same general mechanism that had been successfully used in the early experiments (Webster and Bernhard, 2003). This mechanism operated in the following manner:

1. Generate an instance of a problem to be tested
2. Generate a Monte Carlo (random) solution to the problem instance
3. Using the Monte Carlo solution as a starting point, solve the problem instance using Hill Climbing
4. Again using the Monte Carlo solution as a starting point, solve the problem instance using Simulated Annealing
5. Again using the Monte Carlo solution as a starting point, solve the problem instance using the Genetic Algorithm
6. Once more using the Monte Carlo solution as a starting point, solve the problem instance using GELS
7. Repeat steps 1 through 6 for each problem instance to be tested

Using this mechanism provided several benefits. Generating a Monte Carlo solution for each problem instance could be expected, on average, to deliver an objective function value neither the worst possible nor the best, but somewhere in

the middle. By using this solution as the common starting point for all the other algorithms, it ensured that all algorithms had an equal opportunity to improve upon the same solution. Had each algorithm been allowed to have a different starting point, each could have begun in a different neighborhood of the solution space, and it could not have been known for certain whether improvements in solution quality obtained by each algorithm were due to its performance capability, or because it began in a more advantageous neighborhood. By following the mechanism it was guaranteed that for every problem instance tested, each algorithm would begin in the same neighborhood and would be forced to realize any improvement based solely on its own merits.

In addition, it was decided that each problem type would utilize a common neighborhood selection definition. In doing so, when local search neighborhoods were needed each algorithm would construct them in exactly the same way. Since the neighborhood plays such a vital role in determining how a local search algorithm will navigate through a search space, allowing each algorithm to determine its own neighborhood selection method could have given one algorithm an advantage over another by virtue of having neighborhoods that produced better search patterns.

When deciding on configurations to use for each of the test algorithms, at first it was thought that it might be a simpler and easier matter to use prepackaged procedures. There are a number of such packages available for use, and several were tried. Ingber provides a general purpose Simulated Annealing package called

Adaptive Simulated Annealing (ASA) (Ingber, 1993). Kliewer and Tschöke also describe a Simulated Annealing library called parSA (Kliewer and Tschöke, 1998). Goodman describes a Genetic Algorithm package called GALOPPS (Genetic ALgorithm Optimized for Portability and Parallelism System), produced by the Genetic Algorithm Research and Applications Group (GARAGe) at Michigan State University (Goodman, 1996). Wall at the Massachusetts Institute of Technology provides another Genetic Algorithm package called GALib (Wall, 1996).

After some examination, though, it was decided not to use any of the prepackaged procedures. Though very sophisticated and capable of operation with many different parameter settings, using these procedures would have introduced some of the same biases that attempts had been made to avert with the design of the problem types. Namely, in many cases they could not make the guarantee that all algorithms would use the same starting point and the same neighborhood selection method.

Ultimately, it was decided that the best means of ensuring that as much control as possible was maintained over the experimentation process was to develop a custom-made framework. Consequently, what emerged was a completely self-contained environment written in C++. Each of the test problems became a C++ class, with member functions to create problem instances, calculate objective function values, generate local search neighborhoods, solve the instances using each of the test algorithms, and output the results. This ensured that each test would have the same problem instance being solved using the same objective

function definitions, by the same algorithm configurations, with the same neighborhood selection methods, and all within the same environment, designed specifically for these experiments by the same developer.

### 2.2.3 Implementation of the Test Problems

Creating the classes that would implement each of the test problems, though obviously requiring elements unique to each of the problem types, had a common theme. Each class would have to be able to generate problem instances. Each class would require methods for determining local search neighborhoods, calculating objective function values, and selecting Monte Carlo solutions. Each class would require some way of keeping track of the solutions produced by the various algorithms for each problem instance, and would have to output them in a manner usable by SPSS for later analysis.

The process of keeping track of solutions was handled by variables that stored the solutions and their associated objective function values as produced by each test algorithm. Common print routines were then used to output the results to a flat text file that could later be loaded into SPSS. How each of the problems implemented the other necessities will now be discussed.

#### 2.2.3.1 Traveling Salesman Problem Implementation

To implement the TSP class, a specific version of the problem had to be chosen. As was the case with the other test problem types, over time the generic definition of TSP had evolved into several variations, each with its own special conditions. The variant implemented used *symmetric* costs; that is, if going from city A to city B incurs a given cost, then going from city B to city A incurs the same cost. This is one of the most straightforward versions of the problem, and was chosen for that reason.

Generating problem instances for TSP involved creating an NxN symmetric matrix of integer values representing the cost to move from any one city on the tour to any other, where N was the given size of the problem instance (i.e. the total number of cities on the tour). The matrix was populated by first setting all diagonal values to zero (since there is no cost for moving from city A to city A). One half of the matrix was then initialized by generating random values between one and ten (ten being an arbitrarily determined maximum cost for any one movement). The other half of the matrix was set to mirror the values of the first half to enforce symmetry of the matrix and of city movement costs (Stewart, 1973).

Determination of local search neighborhoods for TSP was accomplished by using a pair-wise rearrangement scheme (Aarts and Lenstra, 1997). The procedure for this rearrangement was as follows:

1. Start with a given solution (for which a neighborhood is to be generated) and an empty neighborhood
2. Swap the first and last elements in the given solution
3. Add the resulting solution to the neighborhood
4. Set an index variable to the second element in the given solution
5. Swap the element in the given solution indicated by the index variable with the preceding element in the given solution
6. Add the resulting solution to the neighborhood
7. Increment the index variable
8. Repeat steps 5 – 7 until the index variable reaches the last element in the given solution

Note that the given solution itself is never actually modified; each member of the neighborhood is produced by starting with the original solution as given and altering a copy of it to place in the neighborhood. By using this method, the local search neighborhood would consist of N members for a problem instance of size N, meaning that the size of the neighborhood would grow linearly in the size of the tour, as opposed to the higher growth rates of some other methods.

The objective function for TSP was calculated via the problem instance matrix previously described. Since the definition of TSP states that a tour begins and ends

at a home city, it was stipulated that problem instance tours would always begin and end at city 0. To calculate the objective function value for any given solution, the following procedure was accomplished:

1. Start with an objective function value of 0
2. Consult the problem instance matrix to find the cost of moving from city 0 to the first city on the tour
3. Add this cost to the objective function value
4. Consult the problem instance matrix to find the cost of moving from the current city to the next city on the tour
5. Add this cost to the objective function value
6. Repeat steps 4 and 5 for each successive city on the tour
7. Consult the problem instance matrix one last time to find the cost of moving from the last city on the tour back to city 0
8. Add this cost to the objective function value to yield the final total

The other item required for the TSP class definition was a mechanism for determining Monte Carlo solutions. This was done by completing the following procedure:

1. Start with an empty Monte Carlo Solution
2. Generate a random integer between 1 and N (N being the number of cities on the tour)
3. Make this value the first city on the Monte Carlo tour
4. Generate a random integer between 1 and N
5. If the city represented by this value is not already on the Monte Carlo tour, add it to the end of that tour, otherwise go back to step 4
6. Repeat steps 4 and 5 until all of the N cities have been included, giving a complete Monte Carlo tour for the problem instance

#### 2.2.3.2 Bin Packing Problem Implementation

As with TSP, there are a number of variants of BPP that have evolved, and one of them had to be selected for implementation. The particular version selected was the *1-Dimensional* BPP which, as described in the problem introduction, consists of adding objects of one dimension (size) to bins that are also of one dimension. This is in contrast to other variants such as the 2-Dimensionsal BPP, where the objects and bins have length and width dimensions, and was selected for its relative simplicity.

Generating problem instances for BPP was a simple process of generating a series of N random integer values, where N was the given size of (i.e. number of objects in) the problem instance. Each value would range between one and fifty (half the predefined size of a bin). Each value represented the size of one object to be put into a bin, and at the outset each object was assigned to its own bin. Thus, each problem instance would consist of N objects in N bins.

Determination of local search neighborhoods for BPP was a somewhat more complex process (Kochetov and Usmanova, 2001). To accomplish this task, the following procedure was used:

1. Start with a given solution (for which a neighborhood is to be generated) and an empty neighborhood
2. Establish a counting variable with an initial value of 1
3. Make the first bin in the given solution the current bin
4. If the current bin contains at least as many objects as the value of the counting variable, move a number of objects equal to the value of the counting variable to the next bin (if the current bin is the last bin in the given solution, move the object(s) to the first bin), and add the resulting solution to the neighborhood
5. Repeat step 4 for each bin in the given solution, or until there are N solutions in the neighborhood

6. If all bins in the given solution have been examined, and there are still less than N solutions in the neighborhood, increment the counting variable and go back to step 3

Objects were stored in bins as if in a queue. Thus, when moving a certain number of objects from one bin to another, the objects were taken from the front of the first bin's queue in order and added to the end of the second bin's queue in the same order. Again, as with TSP the original given solution was not actually modified, but copies of this solution were altered according to the aforementioned neighborhood generation procedure in order to be added to the neighborhood. Also as with TSP, this procedure produced a local search neighborhood that grew linearly in the size of the problem instance. Note that altering the original given solution as indicated in the procedure could lead to neighborhood solutions that were invalid (e.g. a bin with exceeded capacity). This situation was dealt with by assigning invalid solutions a particular objective function value, as will be seen momentarily.

Calculating the objective function for solutions was a simple matter of counting the number of non-empty bins – this number would then be the objective function value. There is normally not as much variation between high quality and low quality solutions for BPP as there is for TSP, and methods such as computing the average percent full over all bins can be used to further distinguish between solutions. While these methods may serve to widen the potentially narrow spread of solution qualities, it was decided that since the objective of BPP is to minimize

the number of bins to store all the objects, the objective function should be a direct reflection of this fact. Hence, the procedure of using the number of non-empty bins in a solution as its objective function value was retained. Any invalid solutions that may have appeared as a result of neighborhood generation were assigned an objective function value of N + 1. Since the worst possible valid solution for BPP has each object in its own bin (thus giving N non-empty bins), an objective function value of N + 1 ensured that invalid solutions would always have a worse value than any valid solution. Since, as stated, problem instances began with each object in its own bin, this guaranteed that at least one solution would be present with an objective function value less than N + 1, and hence no invalid solution could ever be returned by any algorithm as the best solution seen.

To generate Monte Carlo solutions for BPP, the following procedure was used:

1. Start with the original problem instance solution (each object in its own bin)
2. Select the object in the first bin as the current object
3. Randomly reassign the current object to one of the N bins in the original solution (possibly back where it started)
4. Check the validity of the reassignment (i.e. would assigning that object to that bin cause the bin's capacity to be exceeded)
5. If the reassignment is not valid, repeat steps 3 and 4 until the reassignment is valid
6. Repeat steps 3 – 6 for each object in the original problem instance solution

#### 2.2.3.3 File Assignment Problem Implementation

Like TSP and BPP, to implement the FAP class required the selection of a specific version of the problem. There were many considerations: how many devices to use, how many files to use, what kind of device accesses to account for and how to represent them, what kind of costing should be used to formulate an objective function, etc. Here again the decisions came down to matters of straightforwardness. It was never intended that the experiments should represent solution patterns for some exotic special cases of FAP, but rather that they would show solution patterns for a very generic form of the problem (this was in fact the case when making similar considerations for all test problem types).

Consequently, it was decided to build the problem instances around an actual benchmark for database access: TPC-H. TPC-H stands for Transaction Processing Performance Council – H, and is a standard for testing query processing efficiency for decision support databases (TPC, 2002). It posits a collection of eight database tables, against which twenty-two separate queries will be executed. Using this standard, a table was constructed listing the number of "hits" that would occur against each table in the database for a single execution of each of the queries. Hits were determined by examining the queries and looking for references to the tables in the "FROM" clause of the Structured Query Language (SQL) code used to formulate the queries. If a particular table was to be selected from in a given query, it would count as a hit against that table for each execution of that query.

Of course, the exact number of times a particular table will be accessed cannot be determined simply by examining a query. The number of accesses will be determined by the amount of data in the table, interactions with other tables, filters, etc., all data that are unavailable in these experiments due to the fact that hypothetic tables are to be stored on hypothetic devices. Once again, the principle of straightforwardness prevailed, and the rule became “one table reference, one hit”. That is, if a particular table was referenced once in the “FROM” clause of a given query, that table would register one hit per execution of that query. If the table was referenced multiple times, as could occur with table joins, the table would register one hit for each reference in the query.

Once this cross-reference of queries and table hits was constructed, it became possible to get a relative gauge of how frequently tables were being accessed by query executions. Armed with this information, problem instances could be constructed by generating a sequence of executions of any of the twenty-two queries, each one in the sequence randomly selected. Given a number of total queries to process, the total number of hits registered against each table could be calculated and noted. Each table was then assigned to one of the available devices in a round-robin fashion, beginning with the first device. The number of these devices was set to four, each with a simulated size of thirty-six gigabytes. This was done to mimic a recommended minimum setup for a commercial database installation, using commonly available disk sizes (Oracle, 2002). Each table was

set to a simulated size of eight hundred megabytes, in order to allow for the possibility of all tables being stored on a single device with room for overhead.

Defining local search neighborhoods for FAP was accomplished in much the same manner as for BPP. The only differences between the neighborhood generation procedure for BPP and the one for FAP were:

1. Instead of moving objects from one bin to the next, tables were moved from one device to the next
2. Since each device was capable of holding all the tables, there was no possibility of an invalid solution being generated by the procedure
3. The size of the neighborhood was relative to the number of tables and devices rather than the number of objects and bins

Other than these three differences, the neighborhood generation procedures for the two problem types were identical. This produced fixed-size neighborhoods, due to the fact that problem sizes for FAP were determined not by the number of tables and/or devices (which were static), but by the number of queries to be executed against them.

In defining the objective function for FAP, again there were many options. Dowdy and Foster, in their seminal paper on FAP, described a number of different cost indicators that could be optimized (Dowdy and Foster, 1982). The final selection was made based on a factor that would be of common interest to database administrators, namely disk contention. Database administrators, this author among them, are keenly interested in maintaining a high level of performance from

their databases, and one of the methods for doing this is to monitor storage devices for unbalanced access rates.

Along these lines, Bernhard and Fox outlined a method for using minimum database device contention as an objective function measure (Bernhard and Fox, 2000). This measure was the initial one implemented, but in test runs it was discovered that the same objective function values were being produced by all the test algorithms on almost all problem instances. This turned out to be because the most active tables in the TPC-H queries were setting a lower bound on the possible minimum device contention value. Consequently, it did not matter on which device these most active tables were placed or what other tables were placed with them; because of their influence the minimum device contention values were always identical for any given problem instance. Since Bernhard and Fox had success with this measure in their work, this situation would appear to be a function of the particular TPC-H query structure and device configuration used to implement the problem.

Because of this issue, the final objective function was defined to be the average device contention over all queries instead of the minimum device contention. This allowed some form of device contention to be retained as a measure of FAP objectives, and test runs showed that by using this measure it did make a difference on which device and with which other tables the most active tables were stored. Thus, the objective function value for FAP became the average number of hits per device over the course of a certain number of queries.

As with the local search neighborhood definitions, generating Monte Carlo solutions for FAP was done in much the same way as for BPP. Each of the eight TPC-H tables was randomly assigned to one of the four available devices. The Monte Carlo solution thus became a particular mapping of the tables to the devices.

### 2.2.4 Implementation of the Test Algorithms

Having defined classes that implemented each of the test problem types, it was then necessary to implement each of the test algorithm types. Each algorithm was implemented as a member function for each test problem type. This was done due to the different structures necessary to represent solutions for the different problem types. Each problem type required slightly different methods for accessing and manipulating solutions and solution parts, and performing such operations occupied significant sections of the activities of each algorithm. So, rather than implement the common parts of each algorithm as generic functions, with subroutines unique to each problem type for every instance of a solution operation (which would have constituted a goodly portion of the implementations anyway), it was decided to port the implementation of the algorithms between problem types and alter the solution operations as necessary.

Testing the performance and efficiency hypotheses for the algorithms, the primary reason for this research, required some overall standard to provide a basis of comparison for the algorithms' performance and efficiency. For the

performance hypothesis, the leading candidate for this standard would seem to be the objective function values. After all, these values are a direct representation of the quality of solutions. However, as pointed out already the objective function values can vary greatly between problem types, rendering the usage of objective function value alone misleading as a basis of comparison between problem types.

The performance measure that was eventually settled on is a ratio of change in objective function value with respect to the Monte Carlo solution for a problem instance. Stated formally, the measure is

$$\frac{MC - OF}{MC}$$

where $MC$ = Monte Carlo solution objective function value

$OF$ = algorithm objective function value

This measure expresses a particular algorithm's ability to improve on the original Monte Carlo solution as a percentage of that solution's objective function value, and thus the higher the measure, the better the performance. As a result, it is less dependent on the problem type than the objective function value alone. It is designed to function with minimization problems, a characteristic common to all three of the test problem types. It also normalizes values to between zero and one, meaning that the measure will generate the same maximum range of values regardless of problem type. It is true that problem types with larger objective function values will tend to have larger MC values, allowing for more margin for improvement and thus larger values for the measure than problem types with

smaller objective function values. However, this effect can be tempered during analysis by making each problem instance part of a blocking factor. By doing so, variances in MC values between problem instances can be accounted for. This makes the measure suitable for use as a performance measure for any one problem type by itself, or as a comparative measure between problem types.

Likewise, a standard measure of algorithm efficiency had to be adopted in order to test the efficiency hypothesis. The measure selected to serve in this capacity was a function of the number of solutions within the solution space that an algorithm examined prior to termination. It may not be clear why a count of the number of solutions examined would be the measure of efficiency for an algorithm, instead of an iteration count or something similar. Iteration count is not a very good candidate for an efficiency measure. Different algorithms follow different procedures, and a single iteration of one algorithm could involve a substantially different number of operations than a single iteration of another algorithm. Comparing algorithms on this count would not be comparing "apples to apples".

One item that does mean the same thing between algorithms is the count of solutions examined. All search algorithms examine solutions to see if they meet the criteria for solving their respective problems. For local search algorithms, the normal method is for the algorithm to examine a certain number of solutions within one or more successive neighborhoods and at some point terminate, returning the best solution it found from all solution examinations. The total number of solutions examined will then represent the total amount of search space that has been

examined to reach a conclusion. Some solutions may have been examined multiple times; still, the number of solutions examined will show the total amount of "territory" that was covered before the algorithm terminated. This is similar to the method of counting the number of consistency checks performed to determine the efficiency of algorithms for solving Constraint Satisfaction Problems (Prosser, 1993, Tsang, 1996), and it seemed to be an egalitarian and easily implemented method for use as an efficiency measure.

The actual measure used was a ratio of change in objective function value to number of solutions examined for the problem instance, change meaning relative to the Monte Carlo solution. The formal statement of this measure is:

$$\frac{MC - OF}{SE - 1}$$

where $MC$ = Monte Carlo solution objective function value

$OF$ = algorithm objective function value

$SE$ = number of solutions examined by algorithm

This measure represents the incremental amount of improvement in objective function value achieved by a particular algorithm per solution examined within the solution space, relative to the Monte Carlo solution, and consequently once again the higher the measure, the greater the efficiency of the algorithm. The MC value is obtained by examining a single solution (hence the "1" in the denominator). On the other hand, the algorithm obtained its solution by examining SE solutions. By relating the amount of improvement over the MC value that the algorithm was able

to achieve to the additional number of solutions it had to examine to get there, a picture of how efficiently the algorithm operates will emerge.

Like the performance measure, this efficiency measure is not immune to differences in expected MC values between problem instances. Once again, though, this can be mediated by blocking on the problem instance factor during the analysis phase. Also like the performance measure, the efficiency measure is designed for minimization problems to match the characteristics of the test problem types, and values are normalized to between zero and one. And, since the measure has the same meaning regardless of problem type or instance, it is suitable for a single problem type or multiple types.

Having established the performance and efficiency measures for the experiments, the implementation of each of the test algorithms contained a mechanism for recording the final objective function value obtained by the algorithm for the problem instance and the number of solutions examined during its run. This allowed for later calculation of the performance and efficiency measures to be loaded into SPSS for analysis. The specific procedures that each of the test algorithms followed to obtain those values will now be discussed.

#### 2.2.4.1 Hill Climbing Implementation

As it is a representative of a greedy algorithm, the aim of the Hill Climbing (HC) algorithm is quite simple: get more with every turn. The “more” in this case

meant improvement in objective function value. There are two general versions of basic HC that were considered, *first fit* and *best fit*. The first fit version will move to the first solution it finds within the local search neighborhood that has a better objective function value than the current solution, while the best fit version will examine all solutions in the neighborhood and then move to the one that offers the best improvement in objective function value over the current solution. While the algorithm was implemented to operate in either mode, for the purposes of these experiments the best fit mode would be used as it offered the "purest" form of greedy pursuit and was likely to generate better results.

The base procedure of HC was as follows:

1. Make the Monte Carlo solution the current solution
2. Generate a local search neighborhood for the current solution
3. Find the solution in the neighborhood with the best objective function value
4. If that value is better than the value for the current solution, make that solution the current solution and go back to step 2
5. Record the current solution, its objective function value, and the total number of solutions examined

In step 3, for each solution in the neighborhood that was checked to see if it is better than the current solution, the count of solutions examined was incremented by one (having been initialized to zero at the outset of the procedure). There were no necessary variations in the implementation between TSP, BPP, and FAP, so the

base procedure was used virtually unchanged for each test problem type, except for how solutions were constituted.

### 2.2.4.2 Simulated Annealing Implementation

The base procedure used to implement the Simulated Annealing (SA) test algorithm was an adaptation of the *Metropolis algorithm* (Metropolis et. al., 1953). This is a general identifier used to describe an algorithm that will always follow a selected path if that path leads to a higher quality solution, and will also occasionally follow a selected path if that path leads to a lower quality solution. The Metropolis algorithm is based in turn on the *Maxwell-Boltzmann probability distribution*. This is a continuous probability distribution representing energy states of a system (Sears et. al., 1987). The theory is that the total energy within a system in thermal equilibrium at a given temperature T will be distributed among various energy states E according to the equation

$$P(E) = \frac{e^{-E/kT}}{A}$$

where A is a normalization constant

*k* is a constant value known as *Boltzmann's constant*

This distribution shows that even in systems with low overall energy, there could be points of relatively high energy, and vice versa. The classical form of SA uses the probability equation of the Maxwell-Boltzmann distribution to determine probabilities of moving along a path to lower quality solutions than the current

solution. It begins at a given "temperature" and examines successive solutions one at a time. If the examined solution is better than the current solution, it will become the current solution. If the examined solution is not better than the current solution, it will still become the current solution with Maxwell-Boltzmann probability. Gradually the temperature value is lowered, reducing the probability of accepting lower quality solutions. When a certain temperature is reached, the algorithm terminates.

The rate at which the temperature is lowered is called the *annealing rate*, and is usually a parameter setting. Other parameters are usually available to determine how many times a new solution must be accepted (or how many times a new solution acceptance must be attempted, or both) prior to each temperature reduction, and the terminating temperature threshold. There are theoretically an infinite number of possible combinations of values that could be set for these parameters, so obviously it would be impossible to test them all. Instead, the parameters were implemented to be adjustable at run time and defaulted to values commonly in use, particularly for TSP. Specifically, the initial temperature defaulted to two thousand and was set to be reduced by an annealing rate of 0.01 for each ten solutions accepted or one hundred solutions examined, whichever came first. The termination threshold defaulted to 0.01 as well.

Thus, the base SA procedure became:

1. Make the Monte Carlo solution the current solution

2. Reset the number of solutions accepted and number of solutions attempted to zero
3. Generate a local search neighborhood for the current solution
4. Randomly select a candidate solution from the neighborhood
5. If the objective function value of the candidate solution is better than that of the current solution, make the candidate solution the current solution
6. If the objective function value of the candidate solution is not better than that of the current solution, generate a random number between zero and one, and compare it to the Maxwell-Boltzmann probability value for the difference between the objective function value of the candidate solution and that of the current solution for the current temperature (excluding the normalization constant and Boltzmann's constant); if the probability value is greater than the random number, make the candidate solution the current solution
7. If the candidate solution was accepted in either step 5 or step 6, increment the number of solutions accepted by one
8. Increment the number of solutions attempted by one
9. If the number of solutions accepted has reached ten, or the number of solutions attempted has reached one hundred, reduce the temperature by the annealing rate and go back to step 3
10. If the current temperature is greater than the termination threshold, go back to step 2

11. Record the best solution seen during the run, its objective function value, and the total number of solutions examined

Each time a solution was selected to be a candidate, this constituted an examination of a solution, and hence in step 4 the count of solutions examined was incremented by one (again having started at zero). Once again there were no necessary variations in the implementation between TSP, BPP, and FAP, so the only difference in the procedure between problem types was in how solutions were constituted.

#### 2.2.4.3 Genetic Algorithm Implementation

The implementation that was decided upon for a Genetic Algorithm (GA) was to use three components commonly found in many other such algorithms. The first of these components was a method for simulating Darwinian natural selection, or "survival of the fittest" as it is colloquially (and somewhat incorrectly) known. This method had to evaluate members of a "population" of solutions according to their relative "fitness", which in this case was represented by the quality of the solution – the objective function value. Then, over successive "generations" (iterations) of the algorithm, the method would have to decide which members of the population would survive to remain members of the population. The decision of which solutions would survive would have to be directly related to their fitness; that is, higher quality solutions had to stand a better chance of surviving than

solutions of lesser quality, though no absolute guarantee was required. This component was intended to build up the overall quality of the population solutions.

Because the population was the venue for performing local search operations, it supplanted the usual neighborhood as used by the other algorithm types. The population was the neighborhood, which is typical behavior for genetic algorithms. This meant that the usual methods for generating and manipulating neighborhoods would not be used, but would be handled by GA itself. Initial populations were the one exception to this rule. An initial population was generated at the start of a GA run, and was generated by taking the Monte Carlo solution and generating a standard neighborhood for that solution. The initial population was then established by taking solutions from that neighborhood. Tests showed that the required running time, even for small problem sizes, increased dramatically after the population size began to go over twenty. These increased running times were not accompanied by increases in ability to achieve higher quality solutions, and so the population size for GA was set to a fixed value of twenty. Solutions were taken from the neighborhood in a round-robin fashion until twenty were accumulated in the initial population.

The second component chosen for inclusion in GA was a method for simulating genetic recombination, often called *breeding* or *crossover*. This method would take pairs of solutions from the population and combine them to produce one or more "offspring" containing elements of both "parent" solutions. It would also decide which pairs of solutions would become parents each generation. This component

was intended to create newer, higher quality solutions to add to the population by merging two established solutions (solutions that, by virtue of surviving to remain in the population to breed, would already be of higher quality than other solutions that did not survive).

The third component chosen was a method for simulating genetic mutation. This method would select solutions from the population and perform some random change to reconstitute them as different solutions. It would decide what forms of changes could take place, and at what rate. This component was intended to introduce random changes into the population in the hope that they would be beneficial, raising the quality of the solution in the same way random genetic mutations introduced into a population of plants or animals sometimes produce beneficial changes that make them better able to flourish in their environment.

The selection component was implemented to use a probabilistic selection based on the best fitness currently in the population. The current population was examined to find the solution with the best fitness (objective function value). The probability of a particular solution remaining in the population then became the ratio of that solution's fitness to the population's best fitness. One by one each solution in the population would be examined, and a random number between zero and one would be generated. If the solution's ratio was greater than the random number, it was moved into a new population. Once all solutions had been examined, if the new population still did not have the maximum number of members then the process would repeat until the new population was full.

The crossover component was implemented to first generate a random number between zero and one. This number was then compared to the crossover probability, a run-time parameter set to 0.25 (a value commonly used for this purpose). If the crossover probability was greater than the random number, crossover was triggered to occur. The actual crossover process involved taking each successive pair of solutions in the new population (containing the solutions chosen by the selection process from the original population as just described) and redistributing their respective solution elements. Care had to be taken when doing this, since such redistributions could easily result in solutions that were invalid. Also, solutions for each problem type were different, requiring a different crossover mechanism for each.

For TSP, the crossover mechanism operated by moving through the tour list for each successive pair of solutions in the new population. At each stop in the tour, it randomly selected either the city at that stop for the first solution, or the city at that stop for the second solution. It then rebuilt the two solutions by placing the selected city value at the first stop in the solutions that had not yet been rebuilt. A list was kept of the cities that had been used in order to prevent duplicates and ensure that the solutions would still be valid after being rebuilt. To show how this worked, consider the following pair of solutions:

$S_1 = \{3\ 1\ 2\ 4\ 5\}$

$S_2 = \{2\ 5\ 4\ 1\ 3\}$

The mechanism would start at the first stop on the tour and randomly select one of the values, say in this case 3. Since none of the parts of the solutions had been built yet, it would assign city 3 to the first stop for both solutions. Moving on, suppose 1 was the value selected for stop two. This value would then be assigned to the second stop for both solutions, since the first stop had already been rebuilt. For stop three, suppose the value 4 is selected. That value would be assigned to the third stop for both solutions. Suppose then that for stop four, the value 1 was selected. Since this value had already been used, this selection was skipped. Finally, at stop five, suppose the value 3 was selected. Once again, this value had already been used, so the selection was skipped.

All stops in the original pair of solutions had now been examined, but due to duplicates being selected and one value being missed, only three of the stops had been rebuilt. The remaining two values, 2 and 5, would be assigned to the remaining two stops in order, making the rebuilt solutions look as follows:

$S_1 = \{3\ 1\ 4\ 2\ 5\}$

$S_2 = \{3\ 1\ 4\ 2\ 5\}$

In this way, a single new solution was manufactured from the two original solutions by recombining their elements, and the validity of the new solution was also ensured. Both of the original solutions within the new population were overwritten with this new solution, and both were retained within the new population to keep the population size constant.

For BPP, the crossover mechanism again moved through successive pairs of solutions in the new population, taking the elements in each corresponding pair of bins (one from the first solution and one from the second) for each pair of solutions and randomly assigning them back to either the bin in the first solution or the bin in the second solution. This produced new solutions that were rearrangements of the originals. Lists of the original solution contents and original bin contents were kept to ensure that all elements (and no others) originally in each solution and in each bin pairing from those solutions were still in the rearrangement in order to preserve the validity of the solutions, and to ensure that no bin would have its capacity exceeded. As an example of this mechanism, consider the following pair of solutions:

$S_1$ = { $B_1$: 10, 30

$B_2$: 20, 30

$B_3$: 40, 50 }

$S_2$ = { $B_1$: 20, 40

$B_2$: 10, 30, 30

$B_3$: 50 }

Note that it is not a problem to have duplicate elements, since they represent a size, not an identifier. The crossover mechanism would first place all elements from $S_1$ into a group, and all elements from $S_2$ into another group. The mechanism would then examine $B_1$ with $B_1$, putting all their elements (10, 30, 20, and 40) into a group. Suppose then that elements 10 and 40 were assigned back to $S_1$, and

elements 20 and 30 were assigned back to $S_2$. All elements from the group were accounted for and assigned, and there were sufficient elements in the groups for each solution to cover the assignments, so the procedure would move on to the second pair of bins, putting their elements into a group. Suppose that elements 20 and two of the 30 elements were assigned to the first solution and elements 10 and the other 30 were assigned to the second solution. Again, all of the original elements from the bin group were accounted for and assigned, and there were sufficient remaining elements in the groups of original solution elements, so the procedure would move on to the final pair of bins, putting their elements into a group. Suppose that from this group, elements 40 and the first 50 were assigned to the second solution. Then the next random draw indicated that the last 50 element should also be assigned to the second solution. But, this could not happen since there was only one 50 element originally in the second solution, and it had already been assigned back to a bin in that solution. To preserve validity of the solutions, the second 50 element would have to be assigned to the first solution, which still had an available 50 element that had not been assigned. Also, assigning a second 50 element to the second solution would have exceeded the capacity (100) of the bin. The draw also failed the capacity check and the element was instead assigned to the first solution.

The final arrangement of the solution pair would have been:

$S_1$ = { $B_1$: 10, 40

$B_2$: 20, 30, 30

$B_3$: 50 }

$S_2$ = { $B_1$: 20, 30

$B_2$: 10, 30

$B_3$: 40, 50}

In this manner, the original solutions were rearranged to produce new solutions. The content of each solution remained the same, but the contents of the bins have been altered, though not so as to violate the capacity of any bin. The validity of the solutions has been ensured, and now there are two new solutions to work with. Note that in this example applying crossover did not improve the quality of either solution (each had three occupied bins to begin with, and it remained so after crossover), and this could be the case the majority of the time. However, in some instances the quality could improve. In fact, if the 50's in the example would have been 10's instead, a different random draw could have cut the second solution down to two bins. Also, though a crossover may not actually improve a solution, it could set the stage for an improvement in a later generation.

For FAP, the crossover mechanism operated in a manner virtually identical to that for BPP. The only difference was that instead of reassigning elements between bins as in BPP, files were reassigned between devices. Exactly the same methods of reassignment and solution validity preservation were used, with the exception

that checks to verify that devices did not exceed capacity were not necessary since all files were capable of being stored on a single device.

Implementing the mutation component was a matter of injecting a random change into solutions at a certain rate. This rate was set to be 0.01, or one percent of the time, again a common value used for such purposes. For each solution in the new population (now containing the solutions chosen from the original population by the selection mechanism, plus any changes to those solutions that may have occurred if crossover had been triggered), a random number between zero and one was generated. If this number was less than the mutation rate value, mutation was triggered to occur.

For TSP, mutations were done by randomly selecting a point in the solution representing a tour stop. The city at this point was then swapped with the city at the previous tour stop (or the last stop, if the first stop was the point selected). This produced a new solution, and since none of the cities in the solution had changed but only their ordering, the solution was still valid.

For BPP and FAP, mutations were done by randomly selecting a non-empty bin (or device, for FAP). The first element in this bin/device was moved to another randomly selected bin/device. The first element from this second bin/device was then moved to the first bin/device. Performing this maneuver did carry with it a slight risk in BPP that one of the bins would exceed capacity, rendering the solution invalid. However, changing elements between bins seemed to be the only intra-solution mutation that made sense, as swapping bins would accomplish nothing and

simply changing the value of elements within bins carried with it the risk of corrupting the solution and/or converting it into a different problem instance altogether. If by some chance a bin did exceed capacity and the solution became invalid, its probability of being retained in the next generation dropped to zero and thus it was guaranteed to be eliminated. If somehow this happened to all solutions in the population in the same generation (an extremely unlikely event), a mechanism was in place to generate a completely new population (the same mechanism that was used to generate an initial population).

The three individual components (selection, crossover, and mutation) needed to be combined into a single GA. This was accomplished by making each component a stage in the process of a generation of the population. Starting with a current population (either an initial population or one from the previous generation), first selection would occur to determine which solutions would survive in a new population. Then, crossover would be applied to the new population. Finally, mutation would be applied to the new population. The member solutions as they appeared in the new population following the execution of all three stages would then become the current population and would be the input generation for the next round, when all three components would repeat.

Thus, the general form of GA was:

1. Generate an initial population from the Monte Carlo solution
2. Apply the probabilistic selection to the current population to create a new population

3. Probabilistically apply crossover to the new population
4. Probabilistically apply mutation to the new population
5. Remove the old population and make the new population the current population
6. If the number of generations has not reached the specified limit, go back to step 2
7. Record the best solution seen during the run, its objective function value, and the total number of solutions examined

Each time a current population was established (either through initialization or through transitioning from a new population), each solution in the current population would be examined to see if any solutions had a better objective function value than seen to that point. If so, a new "best solution" would be registered. After examining each solution, the count of solutions examined would be incremented. The total number of generations allowed was arbitrarily set to ten thousand, a value that allowed a substantial number of solutions to be examined without extending run times tremendously for larger problem sizes.

#### 2.2.4.4 GELS Implementation

During early experimentation, the GELS algorithm had been known as GLSA and had consisted of two versions: one that was based on the gravitational attraction between two objects and allowed navigation only to adjacent positions

within the solution space, and another that was based on gravitational field attractions and allowed navigation to non-adjacent positions. Along with a change in name prior to the current experiments came a number of changes to the algorithm itself. As development of the algorithm progressed, it became clear that with the transition from hypothetical search spaces to genuine search spaces for actual problem types, alterations to the algorithm were necessary.

Definitions of neighborhoods had changed, from being adjacent positions within the search space to being solutions that were slight variants of the current solution. Objective function values of neighboring solutions were no longer randomly determined but were now functions of those solutions. Some solutions no longer merely had poor objective function values, but were completely unusable since they did not form valid solutions to the problem at hand.

Many adjustments were made to the algorithm. Along the way, the original two methods of operation were significantly reworked. In addition, it was discovered that several of the parameters in the original model had either become so sensitive to adjustment that finding points of equilibrium was extremely difficult or had become redundant in their effect on the outcome. In the end, two modes of operation and two methods of navigation remained, albeit different from the originals, and only five parameters.

GELS has several elements in common with the other algorithms in the test suite. Like SA, GELS is based on a formula describing a process that occurs in nature. In SA, this formula is the Maxwell-Boltzmann distribution of energy states.

In GELS, the formula is Newton's law of gravitational force between two objects, expressed as

$$F = \frac{Gm_1m_2}{r^2}$$

where G = the gravitational constant, ~ 6.672

$m_1$ = the mass of the first object

$m_2$ = the mass of the second object

r = the radius of the distance between the two objects

Like HC, GELS will navigate towards better solutions. In HC, this occurs as direct movement to solutions with better objective function values. In GELS, movement occurs generally towards solutions with higher "gravity" (meaning better objective function values). Like GA, there is an expectation of iterative improvement, and both algorithms can be terminated by iteration count.

Along with the elements that GELS has in common with the other algorithms, there are also some striking differences. Unlike SA, GELS will not always move to a solution with a better objective function value, even if one is presented to it, and it will not move to solutions with worse objective function values on a probabilistic basis, but deterministically according to its rules of motion. Unlike HC, GELS will not always move directly towards the solution in the neighborhood with the best objective function value, and it will not always stop on a locally optimal solution, but can move off in an attempt to locate even better solutions. Unlike GA, GELS does not proceed by randomly altering solutions, but by examining existing

solutions, and it has distinct termination conditions that can cause the algorithm to complete prior to reaching a specified iteration count.

So, though it does have some elements of randomness within it, GELS does not proceed strictly probabilistically. Though it uses local search neighborhoods to look for solutions, it does not always move through them in the same fashion. And, though it does have some behaviors characteristic of greedy algorithms, it does not always seek to follow the best path or grab the most resources. GELS uses a law that governs the motion of objects in physical space to guide the motion of a search through a complex search space.

The two methods of operation for GELS utilize the same gravitational force formula, but in slightly different ways. The first method applies the formula to a single solution within the local search neighborhood to determine the gravitational force between that solution and the current solutions, while the second method applies the formula to all solutions within the neighborhood and tracks the gravitational force between each of them and the current solution individually. The procession of the search through the search space is governed by the gravitational forces, either in a single direction or in all directions, as determined by the formula.

The two modes of movement through the search space, or *stepping* modes, differ only in how much of the search space they span. The first mode (called *single stepping*) allows movement only to solutions within the current local search neighborhood, while the second mode (called *multiple stepping*) allows movement to solutions well outside of the neighborhood. Each stepping mode can be used

with either method of operation. This results in four total variants of how GELS can conduct a search. Shorthand monikers have been assigned to identify each variant. Each moniker takes the form “TA” (for “test algorithm”) followed by a 1 or a 2 to identify which method of operation is being used, followed by another 1 or 2 to identify whether the algorithm is using single stepping or multiple stepping, respectively. Thus, to identify an instance of the algorithm using the second method of operation and single stepping, the moniker used to identify that instance would be TA21.

GELS maintains a vector, the size of which is determined by the number of dimensions in a solution. For example, a ten-city TSP tour would generate a vector with ten elements, a twenty-city tour would generate a vector with twenty elements, and so on. This vector’s values represent the relative “velocity” in each dimension. The velocity is a measure of how much of a tendency there is to bypass solutions. The higher the velocity, the more the tendency to bypass solutions (this is the GELS mechanism for escaping local optima). If the multiple stepping mode is being used, the velocity is also used to determine how far past the local search neighborhood the search will relocate. There is also a pointer to identify which of the elements in the vector is the current “direction of movement”.

Both methods of operation and both stepping modes have been combined into a single module. The choice of which method and which stepping mode to use can be made at run time. Other parameters that are available are:

- Maximum velocity – defines the maximum value that any element within the velocity vector can have; used to prevent velocities that become too large to be usable
- Radius – sets the radius value in the gravitational force formula; used to determine how quickly the gravitational force can increase or decrease
- Iterations – defines the number of iterations of the algorithm that will be allowed to complete before it is automatically terminated; used to ensure that the algorithm will terminate

The settings of these parameters for the current experiments were arrived at through trial-and-error during the development of GELS. Some settings caused the algorithm to run too long; others caused conditions where numbers were becoming too large, causing the algorithm to behave erratically. After a number of tests, the values settled on were 10 for the maximum velocity, 4 for the radius, and 10,000 for the iterations.

The algorithm begins by initializing the current solution, velocity vector, and direction of movement. As with all the other test algorithms, the initial current solution is set to the Monte Carlo solution. For each dimension in the velocity vector, a random integer between one and the maximum velocity is chosen, and this becomes the value of the element at that dimension. A minimum value of one is set

to ensure that there will be a non-zero velocity component in each dimension at the outset. After all dimensions of the vector have received values, the one having the largest value is set as the initial direction of movement (in the event of a tie, the first element in the vector having the largest value will be selected).

Next, a series of iterations of the algorithm are executed. What happens in each iteration will be dependent on which method of operation has been selected, and each will be described separately. The algorithm will terminate when one of two conditions occurs: either all of the elements in the velocity vector have gone to zero, or the maximum allowable number of iterations has been completed.

One iteration of the first method of operation consists of first selecting a candidate solution. This solution will be the solution in the local search neighborhood having the same ordinal identifier as the current direction of movement indicator, i.e. if the indicator is currently set to five, the fifth solution in the neighborhood will become the candidate solution. Once selected, the candidate solution's objective function value is checked to see if it is the best one seen to this point. If so, the candidate solution is marked as being the best solution seen so far. The count of number of solutions examined is also incremented at this point.

Next, the gravitational force between the current solution and the selected candidate solution is calculated. Newton's formula is used, with the alteration that the two masses in the numerator of the equation are replaced by the value of the difference between the objective function value of the candidate solution and that

of the current solution. The value of the gravitational force between the two solutions then becomes:

$$F = \frac{G(CU - CA)}{R^2}$$

where G = 6.672

CU = objective function value of the current solution

CA = objective function value of the candidate solution

R = value of the radius parameter

This formula is designed to be a positive value if the objective function value of the current solution is larger than that of the candidate solution and negative if the candidate's value is larger. This is because the gravitational pull should be towards solutions with better objective function values. Since the problem types used in the experiments are all minimization problems, a lesser objective function value is better. Thus, if the candidate solution is better it will have a lower objective function value, making CU – CA a positive value.

It would have been possible to replace the two mass numbers in Newton's formula by the objective function values of the two solutions and apply the appropriate sign based on which one was larger. It was decided to use the difference between them instead simply because using the multiplication led to much larger values (and much larger ranges of values) and made the determination of a radius value suitable for several different problem sizes more difficult. It was

believed that using the difference tended to normalize the values somewhat, without loss of the spirit of what Newton's formula was intended to indicate.

Having calculated the relative gravitational force in the current direction of movement, the velocity vector can now be updated. The force value, positive or negative, is added to the component of the velocity vector at the position of the current direction of movement. If doing so makes the value exceed the maximum velocity parameter setting, it is set to the maximum. If the update would cause the value to go negative, it is set to zero.

Note that this process is emulating the acceleration effect that a gravitational force would have, and that in actual physics the acceleration would be calculated by dividing the force by the mass of the object acted upon. However, the object being acted upon would be the solution pointer object moving through the search space, not either of the solution objects themselves (which do not move). Including a mass value for the solution pointer would have meant only that a constant value was being included in the calculation. This would have involved an additional computation that did not add any value to the process, so it was excluded.

If the value of the velocity vector for the current direction of movement has decreased as a result of the update, a check is done to see if it should remain the current direction. As at the start of the algorithm, each element in the vector is examined to find the largest, which will become the new direction indicator. This check is not necessary if the vector element value increased as a result of the

update, since the element updated was already the largest element, and increasing its value cannot make it smaller than one of the others.

Performing an iteration of the algorithm using the second method is very much like the first method. Gravitational forces are calculated, the velocity vector is updated, and a new direction of movement is determined. The only difference is that instead of calculating the force value and updating the vector only for the current direction, the calculation and update are performed for each element in the vector, using as a candidate solution the objective function value of the neighborhood solution corresponding to the index of the element within the vector. Since values are updated for the entire vector, the check for new direction is always performed. Each candidate solution will generate a check for best solution seen and an incrementing of the count of number of solutions examined.

Once an iteration has completed, be it using the first method or the second, the solution pointer is relocated within the search space according to the stepping mode. If single stepping is set, the pointer will be relocated to the solution in the local search neighborhood identified by the current direction of movement. This solution will be checked to evaluate if it qualifies as best solution seen, and the count of number of solutions examined will be incremented.

If multiple stepping is set, the pointer will also be relocated to the neighborhood solution in the current direction. However, if the element value in the velocity vector at the index of the current direction is greater than one, a new neighborhood is generated for the solution to which the pointer just relocated, and the pointer will

move again to the neighborhood solution corresponding to the direction of movement. This process of generating a new neighborhood and moving the solution pointer will repeat a number of times equal to the value of the velocity vector at the current direction index. At each stop of the pointer during this process, a check will be done to see if the solution being pointed to is the best solution seen, and the count of solutions examined will be incremented.

A variation to this process had to be made for TSP. Recall that the method used by TSP to generate solutions for local search neighborhoods is to swap successive pairs of elements in the original solution. If the standard process for multiple stepping is followed in this case, the process of generating neighborhoods will trigger an oscillation in movement, and the solution pointer will not actually move multiple steps beyond the original neighborhood. To see this, consider the following TSP solution: {1 4 2 3 5}. Suppose that the current direction is 3, and element three of the velocity vector is currently set to a value of 5. When it comes time to relocate the solution pointer, it will need to move five times. The first time it moves, it will move to the third solution in the neighborhood. In producing that neighborhood, the third solution would have been produced by swapping the second and third elements in the original solution, meaning the pointer would move to solution {1 2 4 3 5}. On the second move, the third solution in the neighborhood generated for this solution would again be produced by swapping the second and third elements, meaning the pointer would move to solution {1 4 2 3 5}. But, this is right back where it started. A third move would take the pointer back to

{1 2 4 3 5}, a fourth back to {1 4 2 3 5}, and the fifth back to {1 2 4 3 5}. Obviously, the pointer is not really moving five steps beyond the original neighborhood.

To counter this problem, the multiple-stepping procedure was modified for TSP. Instead of moving at each step to the neighborhood solution in the current direction of movement, the pointer would move to a random neighborhood solution. This made the chances of a prolonged oscillation process occurring extremely small. This modified procedure was only required for TSP, as the neighborhood generation procedures for BPP and FAP required completely different methods from TSP, and the problem did not manifest itself there.

After completing the stepping process, the stopping point for the solution pointer is made the current solution, and the count of available iterations remaining is decremented by one (having been initialized at the start of the algorithm to the value specified by the iterations parameter). If there are available iterations remaining, and if there is at least one non-zero value remaining in the velocity vector, the entire procedure consisting of: a) generate a neighborhood for the current solution, b) follow either method one or method two to calculate gravitational forces and update the velocity vector and current direction of movement, and c) perform either single or multiple stepping, is repeated.

A pseudo-code outline of the procedures just described for GELS is as follows:

CurrentSolution = BestSolution = MonteCarloSolution

SolutionsExamined = 0

```
IterationsRemaining = MaxIterationsParameter
VelocitySum = 0
for each Index in VelocityVector
    VelocityVector[Index] = random integer between 1 and MaxVelocityParameter
    VelocitySum = VelocitySum + VelocityVector[Index]
end for
Direction = MaximumValueIn (VelocityVector)
while (VelocitySum > 0 and IterationsRemaining > 0)
    GenerateNeighborhood (CurrentSolution)
    if MethodOneSelected
        CandidateSolution = Neighborhood (Direction)
        if ObjectiveFunction (CandidateSolution) < ObjectiveFunction
            (BestSolution)
            BestSolution = CandidateSolution
        end if
        SolutionsExamined = SolutionsExamined + 1
        Force = Integer (6.672 * (ObjectiveFunction (CurrentSolution) –
            ObjectiveFunction (CandidateSolution)) / RadiusParameter ** 2)
        VelocityVector[Direction] = VelocityVector[Direction] + Force
        if VelocityVector[Direction] < 0
            VelocityVector[Direction] = 0
        end if
```

```
    if VelocityVector[Direction] > MaxVelocityParameter
        VelocityVector[Direction] = MaxVelocityParameter
    end if
    VelocitySum = 0
    for each Index in VelocityVector
        VelocitySum = VelocitySum + VelocityVector[Index]
    end for
    Direction = MaximumValueIn (VelocityVector)
else if MethodTwoSelected
    for each Index in Neighborhood
        CandidateSolution = Neighborhood (Index)
        if ObjectiveFunction (CandidateSolution) < ObjectiveFunction
            (BestSolution)
            BestSolution = CandidateSolution
        end if
        SolutionsExamined = SolutionsExamined + 1
        Force = Integer (6.672 * (ObjectiveFunction (CurrentSolution) –
            ObjectiveFunction (CandidateSolution)) / RadiusParameter ** 2)
        VelocityVector[Index] = VelocityVector[Index] + Force
        if VelocityVector[Index] < 0
            VelocityVector[Index] = 0
        end if
```

```
            if VelocityVector[Index] > MaxVelocityParameter
                VelocityVector[Index] = MaxVelocityParameter
            end if
        end for
        VelocitySum = 0
        for each Index in VelocityVector
            VelocitySum = VelocitySum + VelocityVector[Index]
        end for
        Direction = MaximumValueIn (VelocityVector)
    end if
    if SingleSteppingSelected
        if TSPProblemBeingSolved
            CurrentSolution = Neighborhood[random]
        else
            CurrentSolution = Neighborhood[Direction]
        end if
        if ObjectiveFunction (CurrentSolution) < ObjectiveFunction (BestSolution)
            BestSolution = CurrentSolution
        end if
        SolutionsExamined = SolutionsExamined + 1
    else if MultipleSteppingSelected
        for 1 to VelocityVector[Direction]
```

```
            if TSPProblemBeingSolved
                CurrentSolution = Neighborhood[random]
            else
                CurrentSolution = Neighborhood[Direction]
            end if
            if ObjectiveFunction (CurrentSolution) < ObjectiveFunction
                (BestSolution)
                BestSolution = CurrentSolution
            end if
            SolutionsExamined = SolutionsExamined + 1
            GenerateNeighborhood (CurrentSolution)
        end for
    end if
    IterationsRemaining = IterationsRemaining – 1
end while
return BestSolution, ObjectiveFunction (BestSolution), SolutionsExamined
```

This is the version of the GELS algorithm that was used in the algorithm comparison experiments. Each possible configuration of the algorithm (TA11, TA12, TA21, and TA22) was treated as a separate algorithm for the purposes of those experiments, complete with its own set of run statistics. This was done because not only was it of interest to discover how GELS would perform against

the other algorithm types, but also how each variant of GELS would perform against the other variants.

### 2.2.5 Validation of the Experimental Environment

As alluded to in the discussions regarding implementation of the problem and algorithm types, a fair amount of ad hoc testing was done in the course of the development effort, prior to the pieces being put together into a cohesive experimental environment. When it came time to put those pieces together, a series of tests were conducted to confirm that the environment was operating properly.

The first C++ class to be placed into the environment was TSP. The infrastructure of the problem was put into place (i.e. class definition, necessary program control variables, etc.), and then the implementation of the problem instance generator was initiated. The generator was tested by generating problem instances and printing out the resulting tour movement cost matrix. Each matrix was checked to ensure that all costs were between the specified minimum and maximum values of 1 and 10, respectively (except for the diagonal of the matrix, which should have been all zeroes). Each matrix was also checked to ensure that it was in fact symmetric, with all costs of movement from any given city A to any other city B equal to the cost of movement from city B to city A.

Once the problem instance generator was shown to be operating correctly, the implementations for the Monte Carlo solution generator and objective function

value calculator were added. Solutions to problem instances produced by the generator were verified to be valid TSP tours, with every city in the tour being accounted for with no duplicates. The quality of the Monte Carlo solutions being produced was gauged by evaluating them with the objective function value calculator, which was in turn validated by matching the values calculated against manual calculations done by using the cost matrix to find the cost for each individual leg of the tour and then summing all the costs.

Next, the implementation for the Hill Climbing algorithm was added. This in turn required that the procedure for local search neighborhood generation be in place. The neighborhood generator was tested by using a symbolic debugger to step through HC, and at each point that a new neighborhood was required it was verified that the generated neighborhood was correct. Stepping through HC continued to ensure that it was finding the best solution in the local search neighborhood and was following the best fit greedy pattern by moving to the best solution found in the neighborhood with a better objective function value than the current solution. Finally, it was verified that HC would terminate when it could no longer find any solutions in the neighborhood better than the current solution.

The next item to be added was the implementation for the Simulated Annealing algorithm. Since the methods for local search neighborhood generation and objective function value calculation were already present, the SA procedure was set up to use these procedures. The symbolic debugger was used to step through the SA procedure, verifying that it was following its proscribed steps correctly, and

that individual pieces (like the formula for calculating the Maxwell-Boltzmann probability values) were working properly. Entire runs of SA were followed in the debugger, using break points and watch points to view its progress and ensure that things were in order.

Next to be added was the implementation for the Genetic Algorithm. This followed the same procedure as for SA, hooking GA to the existing objective function value calculator (the local search neighborhood generator not being required) and using the symbolic debugger to verify that each of the GA components (population selection, crossover, and mutation) was operating properly. Also, as with SA complete runs of GA were done through the debugger to verify its progress.

Once its development efforts were finalized, the implementation for GELS was added in. Again, GELS was set up to use the existing local search neighborhood generator and objective function value calculator. As was done with SA and GA, the symbolic debugger was used to verify that each of the designed phases of operation for GELS was correctly functioning, and complete runs were accomplished to observe and verify operations.

The addition of the GELS implementation rounded out the set of test algorithms to be used for TSP. All that remained was to add in the output method to write the results of the algorithm comparisons to file for later uploading into SPSS. Once this was done some complete tests involving all the algorithms and different problem sizes were conducted, both to verify the correctness of the output routine

and to verify that the system would still function correctly during multiple consecutive runs and with different (sometimes quite large) problem sizes.

With the completion of testing for TSP, work began on BPP. Testing of BPP followed the same format as for TSP. First, the class definitions and infrastructure were put together. Then, the common-use procedures (problem instance generator, local search neighborhood generator, and objective function calculator) were added and tested, using the same testing methods as for TSP. The algorithms were then added in one by one in the order MC, HC, SA, GA, and GELS, including the BPP-specific modifications necessary for GA already mentioned and the removal of the special multiple stepping modification needed for GELS only on TSP. Each algorithm was in turn tested in the same manner as was done for TSP. The output routine was then added and complete runs were done both of BPP standalone and in concert with TSP. The output file was examined to verify the correct writing and identification of results for BPP and TSP together.

Once testing was completed on BPP, the final test problem type, FAP, was put together. The steps and methods used in assembling and validating FAP were exactly the same as those for TSP and BPP. When they were completed, complete runs of all three problem types together were done. Results of these runs were compared with those of previous runs of each of the problem types by itself to see if they appeared to be consistent. At this point everything appeared to be functioning properly, which meant that the entire package was now ready to conduct the official algorithm comparison experiments.

### 2.2.6 Performance of the Research Experiments

With the completion of the development and testing efforts for the experimental environment, the DOE for the algorithm comparison experiments could be put into action. This design was fairly simple; generate a series of problem instances for each of the test problem types, and for each problem instance produce a solution from each of Monte Carlo, Hill Climbing, Simulated Annealing, Genetic Algorithm, GELS method one with single stepping, GELS method one with multiple stepping, GELS method two with single stepping, and GELS method two with multiple stepping. Then, the data would be analyzed by SPSS to either confirm or reject the performance and efficiency hypotheses.

When analyzing the data, problem instance was used as a blocking factor. The reason for this was because of the inherent variability of the Monte Carlo solutions. Since they are random, a Monte Carlo solution for one problem instance might be relatively poor in relation to the optimal solution. This would mean that each of the test algorithms would have ample room to improve on it. In another problem instance, however, the Monte Carlo solution might be very close to the optimal. In this case, improving on it would be very difficult no matter what the solution algorithm. Since the Monte Carlo solutions serve as the starting point for all the other algorithms, their relation to the optimal solution will influence the overall results of every experiment. This influence must be accounted for, even if its cause is not of particular interest to the experiments. The relative quality of the Monte

Carlo solutions in relation to the optimal was not known during the experiments, and indeed could not have been known for certain without exhaustive search of each problem instance search space, a completely infeasible task. Therefore, the relative quality of the Monte Carlo solutions for each problem instance was not an item of interest. Yet, it influenced the outcomes of the experiments, as just explained, and thus had to be accounted for to ensure the validity of the interpretation of the results; hence, its inclusion in the experiments as a blocking factor.

The number of experiments to be included in the series was effectively dictated by SPSS. The version of SPSS that was available was the student edition, which limits the number of cases in a single analysis to fifteen hundred (SPSS, 2001), with each case amounting to a single line in the output file. Because of the way SPSS handles its analyses, each case needed to be structured as follows:

{Run No.} {Problem Type} {Algorithm Type} {Performance Value} {Efficiency Value}

The "Run No." field represented the problem instance count, used for blocking of the problem instance factor. The "Problem Type" field identified which of the test problem types the problem instance was generated for, and the "Algorithm Type" field identified which of the test algorithm types was being used to solve the problem instance. These two fields were used to allow grouping of the results by problem and algorithm. The "Performance Value" and "Efficiency Value" fields

contained the metrics used in the analysis. As an example, the SA solution for the tenth TSP problem instance generated would have a case that looked something like the following:

10 TSP SA 0.75 0.11

Structuring cases in this manner allowed SPSS to conduct all of the necessary analysis of the data, but it did limit the number of problem instances experiments that could be conducted. By generating fifty problem instances for each problem type of a given size, twelve hundred cases were created (8 algorithm types x 3 problem types x 50 problem instances = 1,200 cases), falling within the limits of the SPSS student edition. A few more problem instances per problem type could have been generated and still have met the SPSS case limitation, but fifty provided a convenient round number for use in calculations and, as it turned out, generated more than sufficient data for SPSS to produce meaningful conclusions.

To evaluate the effect of different problem sizes on the algorithm comparison, it was decided to conduct separate experiments, each with a different problem size, which would allow the effect to be analyzed while meeting the case limitation. Problem sizes of ten, twenty, thirty, forty, and fifty were each to be analyzed. For TSP, problem size was determined by the number of cities to be included in a tour. For BPP, size was determined by the number of objects to be placed in bins. For FAP, since the number of devices and files was predetermined, size was

determined by the number of queries that would be executed. Each increment of problem size represented one thousand TPC-H queries, e.g. a problem size of ten meant that ten thousand randomly-ordered queries from the TPC-H standard would be executed for that run.

It was also decided that a single set of experiments encompassing all problem sizes would be run, to further evaluate the effect of problem size within a single analysis. Of course, this meant reducing the number of problem instances per problem size. To accommodate the inclusion of problem size as a factor, the case structure for this set of experiments needed to be altered to look like the following:

{Run No.} {Prob. Type} {Prob. Size} {Alg. Type} {Performance Value} {Efficiency Value}

To avoid any possible unforeseen side effects of running problem sizes in a particular order, the size for each problem instance was determined randomly out of the original sizes of ten, twenty, thirty, forty, and fifty. This meant a possible inequality in the number of cases of each problem size, which would place some limitations on the analysis (some statistical tests want equal numbers of cases for each factor value), but it was in keeping with recommended DOE techniques.

Thus, there were a total of six sets of experiments – one for each of the five defined problem sizes, and one consisting of random problem sizes. Each set was designed to allow statistical comparison of the algorithm types, and provided the

SPSS tool with a grand total of 7,200 cases with which to perform that comparison – well more than SPSS required to accomplish the task.

### 2.2.7 Results of the Current Research Experiments

With the DOE for the algorithm comparison experiments defined, the six sets of experiments were run according to the design and the data collected into output files to be uploaded into SPSS for analysis. As the aim of the experiments was to either confirm or reject both the performance and the efficiency hypotheses, the analysis of each experiment's data was conducted in two phases. First, the performance results from each set of experiments were analyzed to compare the relative performance of each of the algorithms against random solutions and against each other. Then, a second analysis was done on the efficiency results from each set of experiments to compare the relative efficiency of each of the algorithms in the same manner. The results of the analyses are presented here in the same order in which they were conducted; first the performance data analysis, then the efficiency data analysis.

#### 2.2.7.1 Algorithm Performance Results

The analysis of the algorithm performance results was conducted by first loading the raw data into SPSS. Then, for each of the three test problem types a box plot and line plot were generated to give any visual indications of a difference

in algorithm performance. Next, an ANOVA was run to see if any statistically significant difference could be detected between the performances recorded for each algorithm type. A fourth box plot and line plot were also generated, and a fourth ANOVA run, each consisting of composite data over all problem types in order to detect any significant difference between algorithm performances across all three problem types considered together. The outcome of each ANOVA was evaluated and if necessary corroborated using the tests described in section 1.2.3.2.3. If a significant difference was determined to exist, an ordering of the algorithms by relative performance was established according to the homogeneous subsets defined by the analysis. This procedure was then repeated for each of the five sets of experiments using a fixed problem size and the one set of experiments using random problem sizes.

#### 2.2.7.1.1 Problem Size Ten Performance Results

Exhibit 12 shows the box plot that was generated for the set of experiments encompassing TSP problem instances of size ten. This plot seems to show a very slight edge in performance capability for SA, but the SA box does not completely fall outside the borders of the boxes for several other algorithms. Also, the median lines for SA, TA21, and TA22 are at about the same level. The box widths are roughly the same, with those for HC, TA11, and TA12 being slightly larger. The whiskers are also roughly the same length, with those for HC and TA12 being a

little longer. These two items show that the inter-quartile and min/max ranges for each of the algorithms are comparable, indicating similar variances between algorithm types.

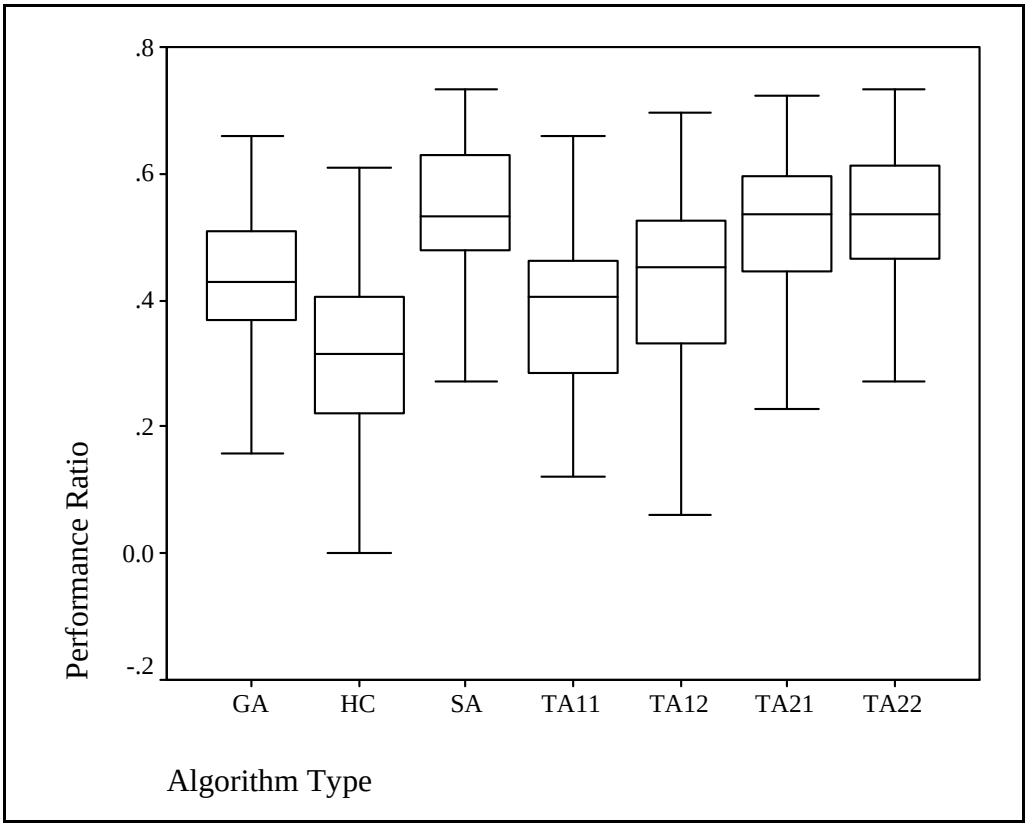


**Exhibit 12.** Box Plot, TSP Size 10 Performance

Exhibit 13 shows the line plot of marginal means that was generated for the same data. This plot shows average performance values that are apparently very similar for SA, TA21, and TA22, and better than the rest. The ANOVA for this problem group should be able to confirm if this is in fact the case.

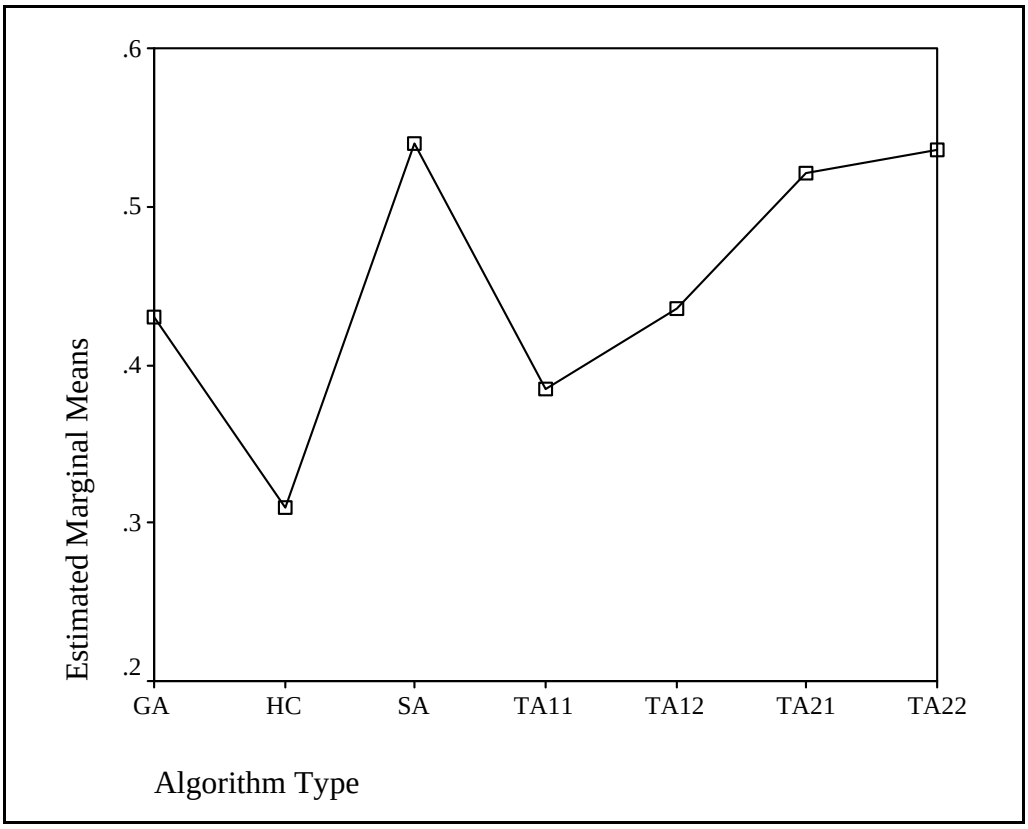


**Exhibit 13.** Line Plot, TSP Size Ten Performance

Exhibit 14 shows the results of the ANOVA. The significance values are all zero, giving a strong indication that variances in performance values for the model in general, and the run number and algorithm type factors in particular, cannot be attributed to random error alone. The run number and algorithm type are almost certainly affecting the performance values being produced. It is good, then, that the run number (problem instance) was included as a blocking factor. Had it been left out, its influence on the model would not have been accounted for and could have distorted the effects seen for the algorithm type factor.

Dependent Variable: Performance Ratio

| Source | Type III Sum of Squares | df | Mean Square | F | Sig. | Partial Eta Squared | Noncent. Parameter | Observed Power[a] |
|---|---|---|---|---|---|---|---|---|
| Corrected Model | 6.007[b] | 55 | .109 | 28.126 | .000 | .840 | 1546.931 | 1.000 |
| Intercept | 71.094 | 1 | 71.094 | 18307.321 | .000 | .984 | 18307.321 | 1.000 |
| Factor A | 3.770 | 49 | 7.693E-02 | 19.811 | .000 | .768 | 970.750 | 1.000 |
| Factor B | 2.238 | 6 | .373 | 96.030 | .000 | .662 | 576.181 | 1.000 |
| Error | 1.142 | 294 | 3.883E-03 | | | | | |
| Total | 78.243 | 350 | | | | | | |
| Corrected Total | 7.149 | 349 | | | | | | |

a. Computed using alpha = .05

b. R Squared = .840 (Adjusted R Squared = .810)

**Exhibit 14.** ANOVA Results, TSP Size Ten Performance

The results of the ANOVA needed to be confirmed by testing to verify that the assumed conditions for a valid ANOVA were in fact present. Exhibit 15 shows the P-P plot of the ANOVA residuals. There is evidence of departure from the normal marker line in two sections of the plot, leading to suspicion that the residuals might not be normally distributed. A Kolmogorov-Smirnov normality test was used at this point to provide a more precise indication.

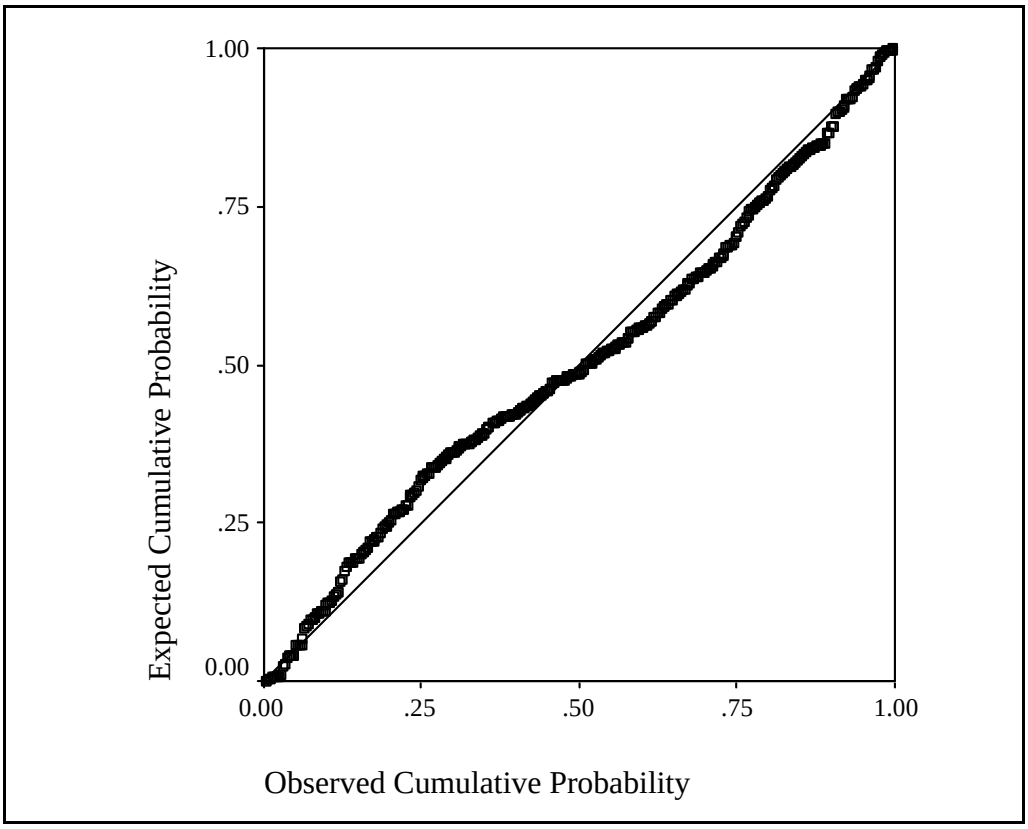


**Exhibit 15.** Residual Normal P-P Plot, TSP Size Ten Performance

Exhibit 16 shows the results of the Kolmogorov-Smirnov test. Here there is a problem. The significance factor shown on the last line is 0.037, which is lower than the threshold of 0.05. This means that the Kolmogorov-Smirnov test has rejected the hypothesis of normally distributed residuals. This cast some doubt on the validity of the ANOVA results, and further evidence was required to corroborate those results.

| | | Residual for Performance Ratio |
|---|---|---|
| N | | 350 |
| Normal Parameters[a,b] | Mean | .0000000 |
| | Std. Deviation | .05719588 |
| Most Extreme Differences | Absolute | .076 |
| | Positive | .058 |
| | Negative | -.076 |
| Kolmogorov-Smirnov Z | | 1.413 |
| Asymp. Sig. (2-tailed) | | .037 |

a. Test distribution is Normal.

b. Calculated from data.

**Exhibit 16.** Kolmogorov-Smirnov Test, TSP Size Ten Performance

However, before examining that additional evidence the other condition of non-structured residuals was tested. The first of these tests was the plot of the predicted values versus observed residuals, shown in Exhibit 17. This plot shows if there is a pattern of relationship between the residuals and the predicted values of performance based on average outputs for given inputs. While there is an indication of narrowing in the plotted points towards the right of the diagram, there is no overall "megaphone" shape or similar pattern discernable. Thus, the plot gives no reasonable indication of a serious problem.

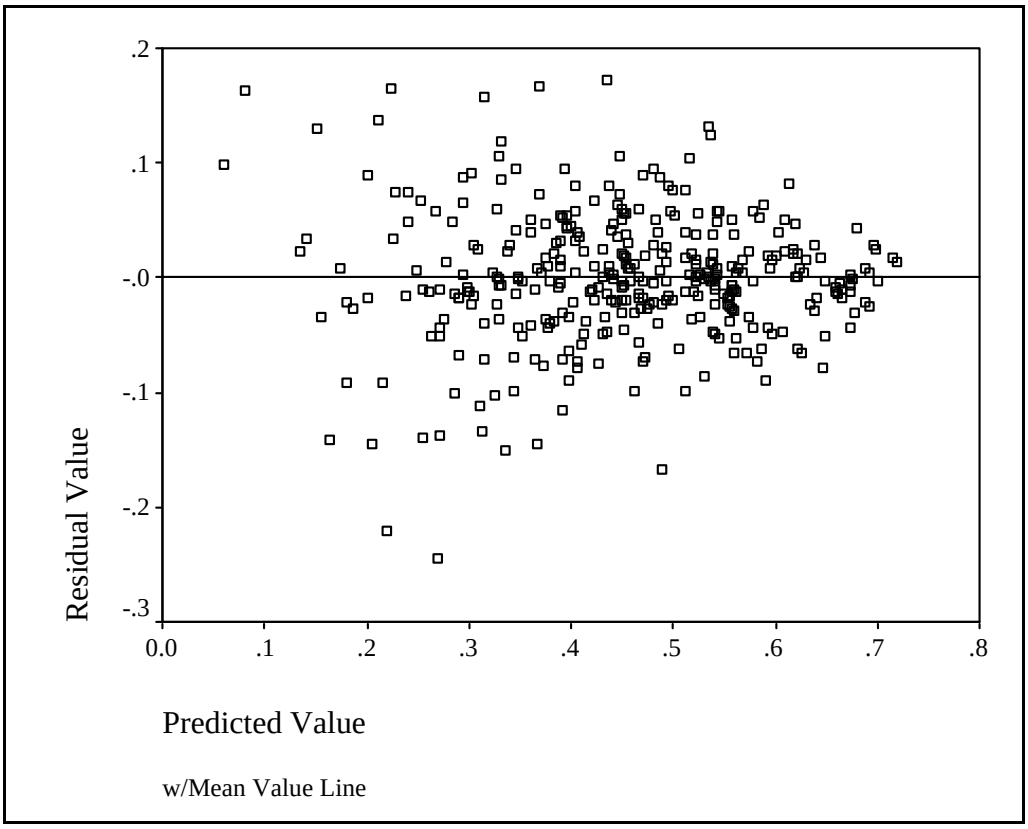


**Exhibit 17.** Predicted vs. Residual Plot, TSP Size Ten Performance

The other test of the assumption of non-structured residuals was the plot of residuals over time, shown in Exhibit 18. This plot shows if there is a trend for residuals to become larger or smaller with each additional experiment run (in this case, problem instance). The plotted points resemble a tube, with no obvious narrowing or widening, and thus there is no indication of a serious problem with structured residuals here either.

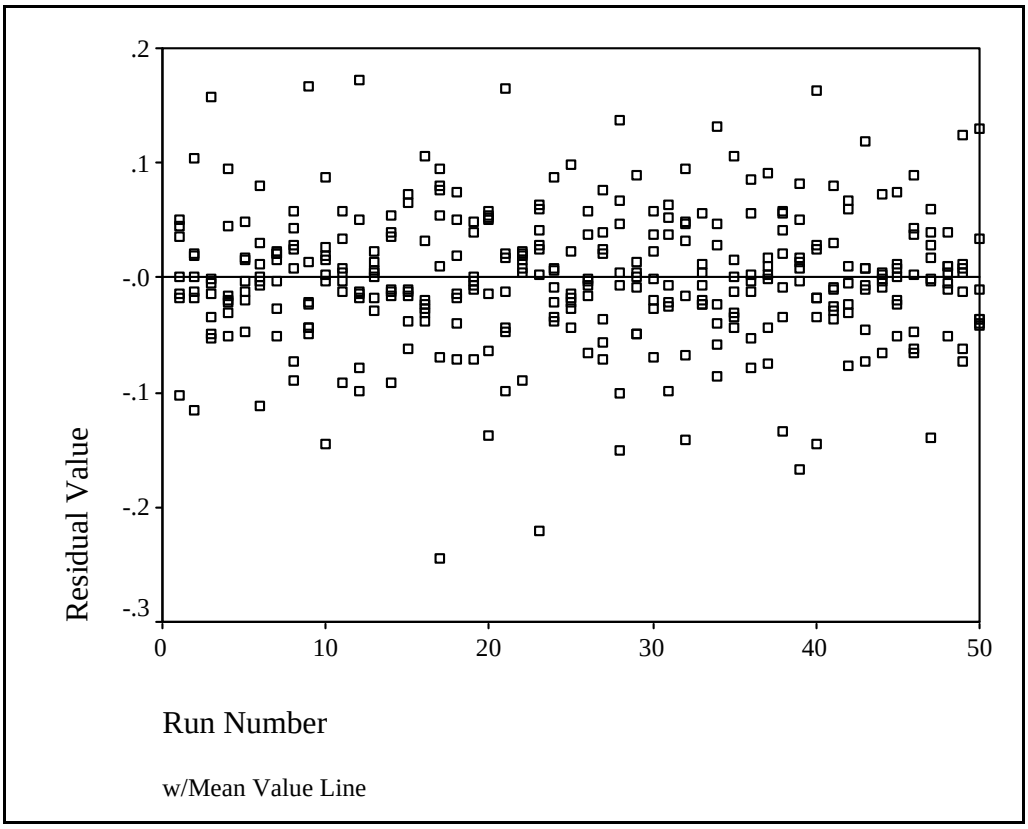


**Exhibit 18.** Residual Trend Plot, TSP Size Ten Performance

The tests for non-structured residuals did not reveal any cause for alarm. However, the normality of the residual distribution was rejected by the Kolmogorov-Smirnov test. Since the residuals could not be assumed to be normally distributed, one of the assumptions of the ANOVA had been violated, and it needed additional evidence to corroborate its conclusions. The Kruskal-Wallis test could be used for this purpose, as it is a non-parametric test not subject to the same assumptions as the ANOVA. The test was conducted using algorithm type as the test factor.

Exhibit 19 shows the results of this test. The significance value shown on the last line is zero, below the threshold of 0.05, meaning that the test accepts the hypothesis that there is a difference between the algorithm types. This result is in agreement with the results obtained by the ANOVA, lending credence to those results in spite of the failed test for residual normality. Since both the ANOVA and the Kruskal-Wallis Test agree on this count, it can be concluded that there is a significant difference between the performance values obtained by the different algorithms for TSP problem instances of size ten.

**Ranking Test Statistics[a,b]**

| | Performance Ratio |
|---|---|
| Chi-Square | 107.785 |
| df | 6 |
| Asymp. Sig. | .000 |

a. Kruskal Wallis Test

b. Grouping Variable: Algorithm Type

**Exhibit 19.** Kruskal-Wallis Test, TSP Size Ten Performance

To find out exactly what differences were considered significant, the homogeneous subsets for algorithm performance were generated, using both the Tukey and Duncan methods. The results are shown in Exhibit 20. In this case both methods have generated four subsets. Since the subsets are automatically arranged by increasing performance ratio, and since better performance of an algorithm is

indicated by higher values for this measure, the best performing algorithms will be in subset 4, and the worst in subset 1. Both the Tukey and Duncan methods also agree on which algorithms should be placed in which subsets. The subset assignments indicate that the performance of SA, TA22, and TA21 are the best, and furthermore, that performance between the three is statistically indistinguishable. The next best performers were TA12 and GA, also indistinguishable. Next down the list was TA11, and HC came in as the worst performer of the group.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | HC | 50 | .3097601 | | | |
| | TA11 | 50 | | .3847190 | | |
| | GA | 50 | | | .4302296 | |
| | TA12 | 50 | | | .4350807 | |
| | TA21 | 50 | | | | .5206225 |
| | TA22 | 50 | | | | .5352588 |
| | SA | 50 | | | | .5391901 |
| | Sig. | | 1.000 | 1.000 | 1.000 | .751 |
| Duncan[a,b] | HC | 50 | .3097601 | | | |
| | TA11 | 50 | | .3847190 | | |
| | GA | 50 | | | .4302296 | |
| | TA12 | 50 | | | .4350807 | |
| | TA21 | 50 | | | | .5206225 |
| | TA22 | 50 | | | | .5352588 |
| | SA | 50 | | | | .5391901 |
| | Sig. | | 1.000 | 1.000 | .697 | .162 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 3.883E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 20.** Homogeneous Subsets, TSP Size Ten Performance

The results of the analysis for TSP problem instances of size ten led to the conclusion that the performance hypothesis could be safely rejected for this group of problem instances. There is a statistically significant difference between the performances of the different algorithm types. The analysis also concluded that Simulated Annealing, GELS method two with single stepping, and GELS method two with multiple stepping are the best performing for problem instances of this type, with performances that are statistically indistinguishable from each other.

To continue with the analysis process, BPP problem instances of size ten were considered. Exhibit 21 shows the box plot for this group of problem instances. The boxes for GA and TA21 appear to be a little bit higher, but otherwise it appears to be a rather tight grouping. Also, this time the box sizes for GA and TA21 are notably smaller than the others, and the whiskers for GA are considerably shorter. This indicates a wider variance in performance than there was for the TSP size ten problem instances, except for GA which apparently had a very tight variance. Still though, there are no outlier markers on the diagram, which would be an indication of some extreme values having an undue effect on the variance.

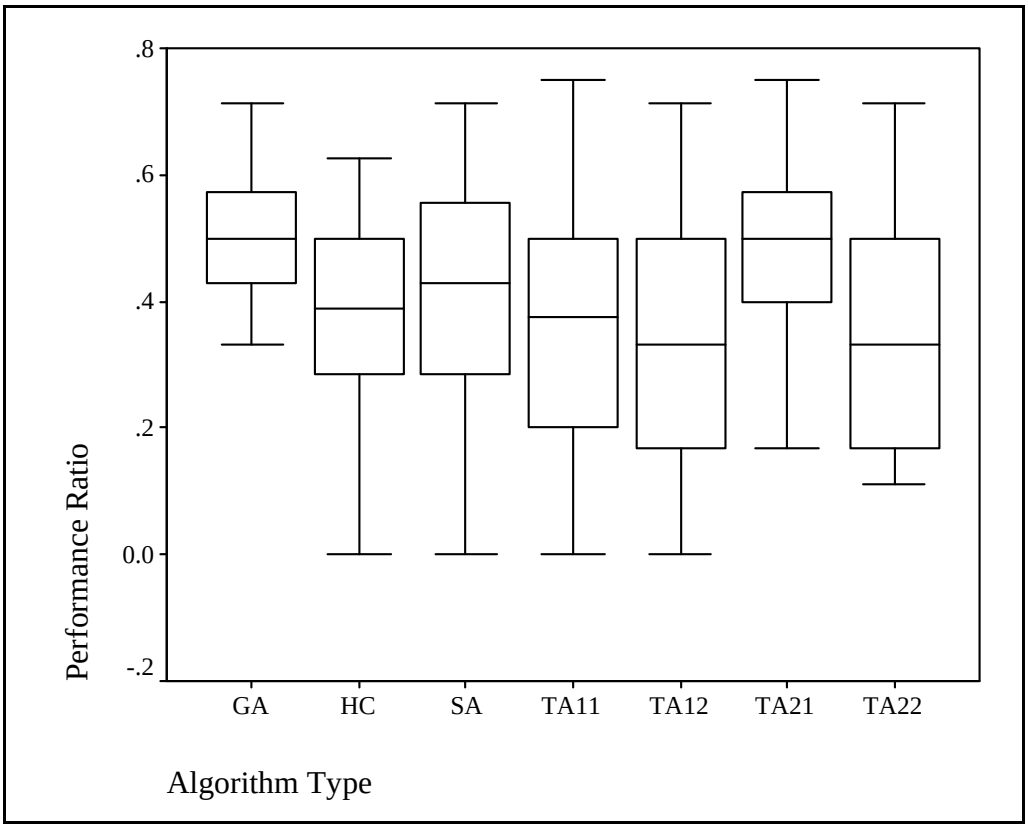


**Exhibit 21.** Box Plot, BPP Size Ten Performance

Exhibit 22 shows the line plot of marginal means for this group of problems. Again, GA and TA21 appear higher than the others, strengthening the suspicion that they are the best performers. In any event, there appears to be a significant difference between the performance of some of the algorithms, and the ANOVA should be able to confirm this.

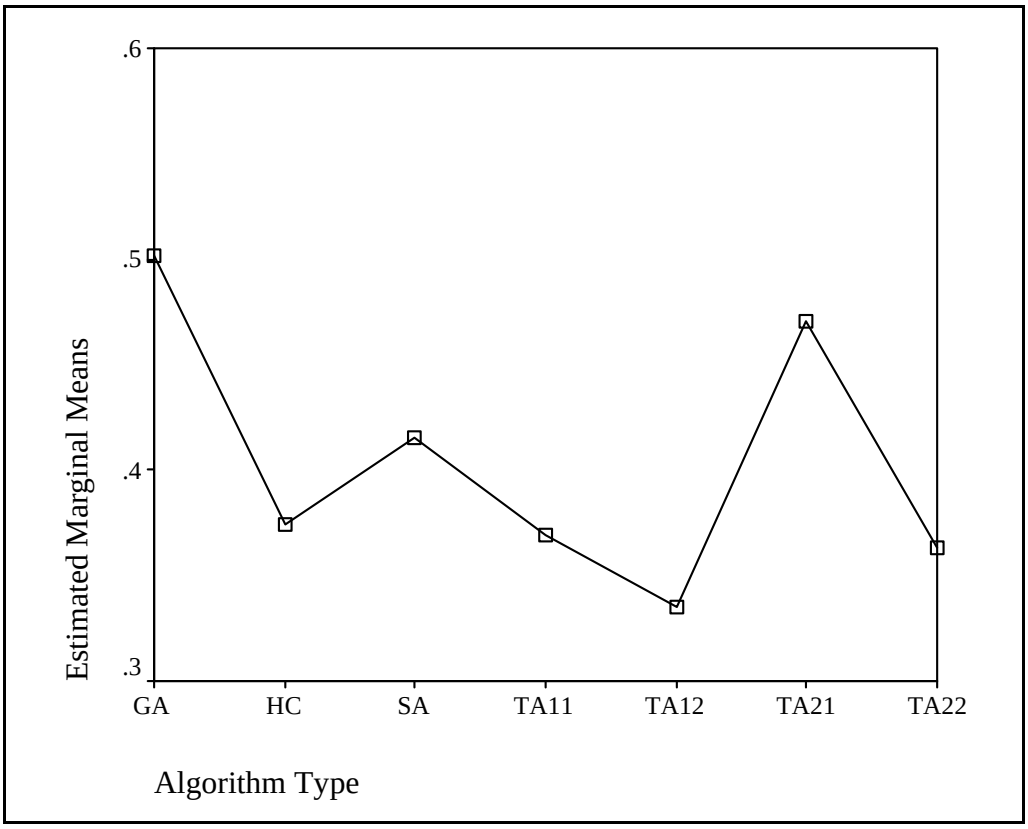


**Exhibit 22.** Line Plot, BPP Size Ten Performance

The results of the ANOVA are shown in Exhibit 23. All significance values are at zero, below the threshold of 0.05, indicating that the model contains significant amounts of variance that cannot be attributed to random error, and that choice of algorithm does make a difference to performance. The run number again is significant, making it a good thing that it was set up as a blocking factor.

Dependent Variable: Performance Ratio

| Source | Type III Sum of Squares | df | Mean Square | F | Sig. | Partial Eta Squared | Noncent. Parameter | Observed Power[a] |
|---|---|---|---|---|---|---|---|---|
| Corrected Model | 5.819[b] | 55 | .106 | 9.109 | .000 | .630 | 501.014 | 1.000 |
| Intercept | 57.154 | 1 | 57.154 | 4920.861 | .000 | .944 | 4920.861 | 1.000 |
| Run Number | 4.693 | 49 | 9.577E-02 | 8.246 | .000 | .579 | 404.037 | 1.000 |
| Algorithm Type | 1.126 | 6 | .188 | 16.163 | .000 | .248 | 96.977 | 1.000 |
| Error | 3.415 | 294 | 1.161E-02 | | | | | |
| Total | 66.388 | 350 | | | | | | |
| Corrected Total | 9.234 | 349 | | | | | | |

a. Computed using alpha = .05

b. R Squared = .630 (Adjusted R Squared = .561)

**Exhibit 23.** ANOVA Results, BPP Size Ten Performance

To validate the results of the ANOVA, the assumptive preconditions of normal distribution of and no structure to the residuals were tested. The P-P plot of the residuals is shown in Exhibit 24. There appears to be a very good fit for this plot, with only one small deviation near the middle. This would lead to the belief that the assumption of residual normality can be upheld, but in order to confirm this the Kolmogorov-Smirnov test was run.

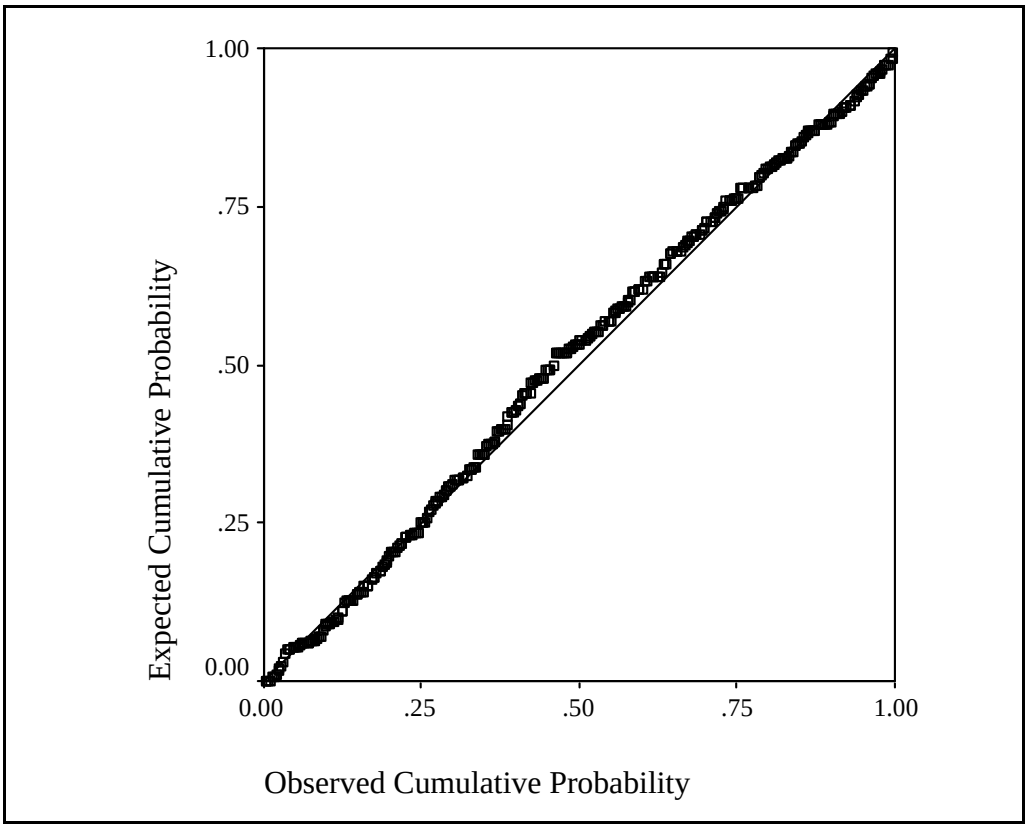


**Exhibit 24.** Residual Normal P-P Plot, BPP Size Ten Performance

Exhibit 25 shows the results of the Kolmogorov-Smirnov test. The significance value of the test is 0.169, above the threshold of 0.05, meaning that the test accepts the normality of the distribution. This confirms what was seen in the P-P plot, and verifies the assumption of normally distributed residuals.

| | | Residual for Performance Ratio |
|---|---|---|
| N | | 350 |
| Normal Parameters[a,b] | Mean | .0000000 |
| | Std. Deviation | .09891547 |
| Most Extreme Differences | Absolute | .059 |
| | Positive | .025 |
| | Negative | -.059 |
| Kolmogorov-Smirnov Z | | 1.111 |
| Asymp. Sig. (2-tailed) | | .169 |

a. Test distribution is Normal.

b. Calculated from data.

**Exhibit 25.** Kolmogorov-Smirnov Test, BPP Size Ten Performance

To test the assumption of non-structured residuals, the predicted values versus residuals and residual trend plots were examined. Exhibit 26 shows the first of these plots. There is a slight "pinching" of the plot at both ends but no overall pattern of widening or narrowing, and the majority of points are clustered within an equidistant band around the mean value line, so this plot is no cause for alarm.

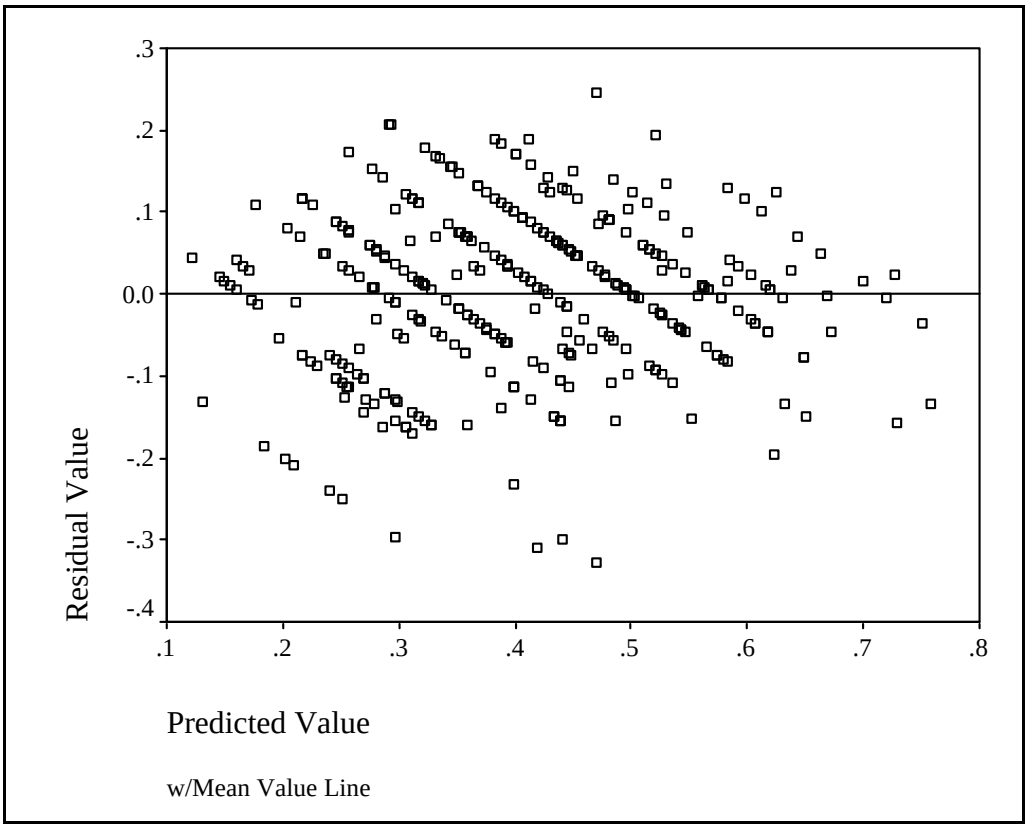


**Exhibit 26.** Predicted vs. Residual Plot, BPP Size Ten Performance

Exhibit 27 shows the residual trend plot. This plot looks very good as well, with no indication at all of any narrowing or widening over the runs of the experiment. The predicted values versus residuals and residual trend plots provide good evidence that the residuals are non-structured, and that this assumption can therefore be made.

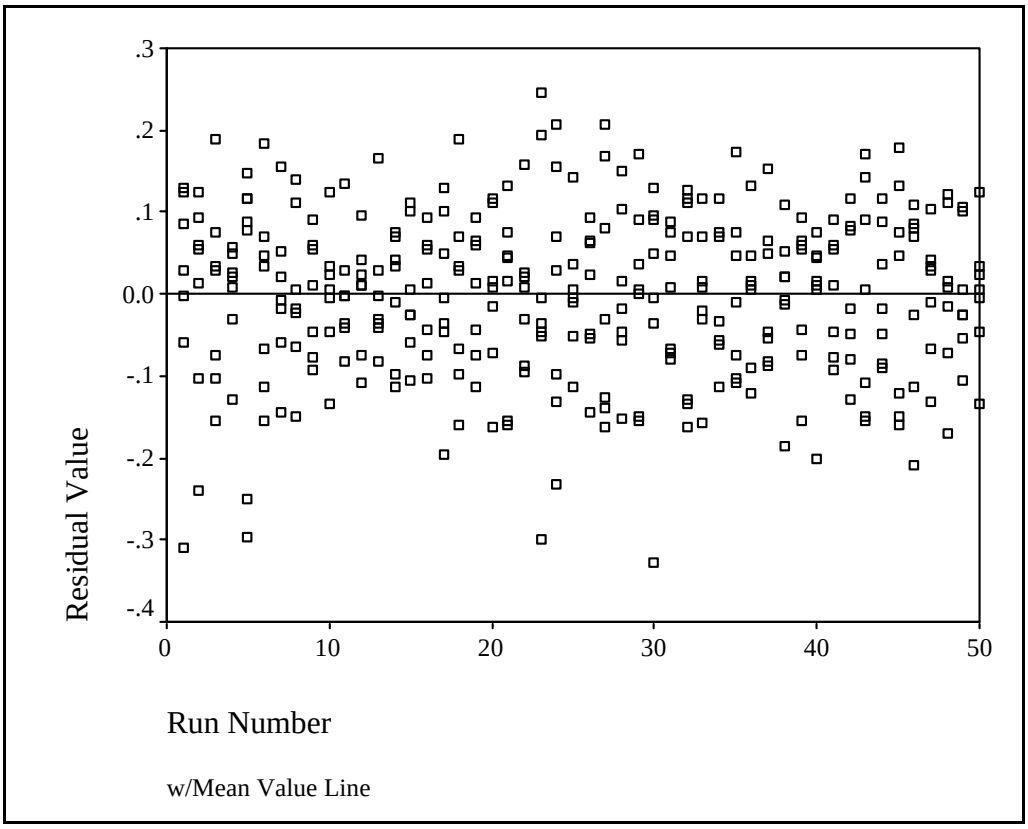


**Exhibit 27.** Residual Trend Plot, BPP Size Ten Performance

Since the assumptions of normal distribution and no structure to the residuals can be upheld, the original ANOVA results can be considered solid. Those results indicated that there is a difference between the performances of the different algorithms. To see which differences can be considered significant, the homogeneous subsets were generated.

These results are shown in Exhibit 28. In this case, the Tukey and Duncan methods have generated a different number of subsets. The Tukey method places GA and TA21 in the top subset, TA21 and SA in the next subset, SA, HC, TA11,

and TA22 in the next, and HC, TA11, TA22, and TA12 in the next, for a total of four subsets. The Duncan method, on the other hand, places GA and TA21 in the top subset, SA and HC in the second, and HC, TA11, TA22, and TA12 in the last for a total of three subsets.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | TA12 | 50 | .3349286 | | | |
| | TA22 | 50 | .3635794 | .3635794 | | |
| | TA11 | 50 | .3689206 | .3689206 | | |
| | HC | 50 | .3744444 | .3744444 | | |
| | SA | 50 | | .4149444 | .4149444 | |
| | TA21 | 50 | | | .4706349 | .4706349 |
| | GA | 50 | | | | .5012540 |
| | Sig. | | .527 | .209 | .135 | .790 |
| Duncan[a,b] | TA12 | 50 | .3349286 | | | |
| | TA22 | 50 | .3635794 | | | |
| | TA11 | 50 | .3689206 | | | |
| | HC | 50 | .3744444 | .3744444 | | |
| | SA | 50 | | .4149444 | | |
| | TA21 | 50 | | | .4706349 | |
| | GA | 50 | | | .5012540 | |
| | Sig. | | .095 | .061 | .157 | |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.161E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 28.** Homogeneous Subsets, BPP Size Ten Performance

Notice that in several cases the same algorithm appears in two subsets simultaneously. The homogeneous subsets methods generate groups of factor

levels that cannot be distinguished statistically. It may be the case that there is not enough of a difference between factor levels A and B to be considered statistically significant. There may also not be enough of a difference between factor levels B and C to be considered significant. This would mean that A and B cannot be distinguished and would be placed in a subset, and B and C cannot be distinguished and would be placed in another subset. However, the difference between A and C may be just enough to be significant. Thus, factor levels A and C would appear in different subsets, but factor level B would appear in a subset with A and in a subset with C at the same time.

Such is the case here. For example, in Tukey's method GA and TA21 appear in the top subset, but TA21 also appears in the next subset with SA. This is saying that there is not enough of a difference in performance between GA and TA21 to be significant, so they are placed in a subset, and there is also not enough of a difference between TA21 and SA to be significant, so they are placed in a subset. However, there is enough of a difference between GA and SA to be significant, so they appear in different subsets. In this way, TA21 appears in two different subsets at the same time.

The results of the analysis of BPP problem instances of size ten showed a significant difference in performance between the test algorithms. Thus, the performance hypothesis can be rejected for this group of problem instances. Further, though there was a difference in the number of homogeneous subsets between the Tukey and Duncan methods, both agreed that the top performers in

this category of problem were the Genetic Algorithm and GELS method two with single stepping.

Continuing on, the next group of problem instances to be analyzed was the FAP instances of size ten. The box plot for this group appears in Exhibit 29. Here there would appear to be a distinct difference between algorithm performances, with the boxes for SA and TA21 falling completely outside several of the others. Since these boxes are above the others, it also indicates that SA and TA21 are the best performers. Box sizes for HC, SA, and TA21 are relatively small compared to the others, and whisker lengths for SA and TA21 are comparatively shorter than the others. There are also outlier values for GA, HC, and TA21. Overall, this indicates lower variability in performance for SA and TA21 in particular. Should these two algorithms be confirmed to have the best performance (which from the diagram would appear to be the case), they would not only be better performing but better performing with a lower chance of returning a poor solution.

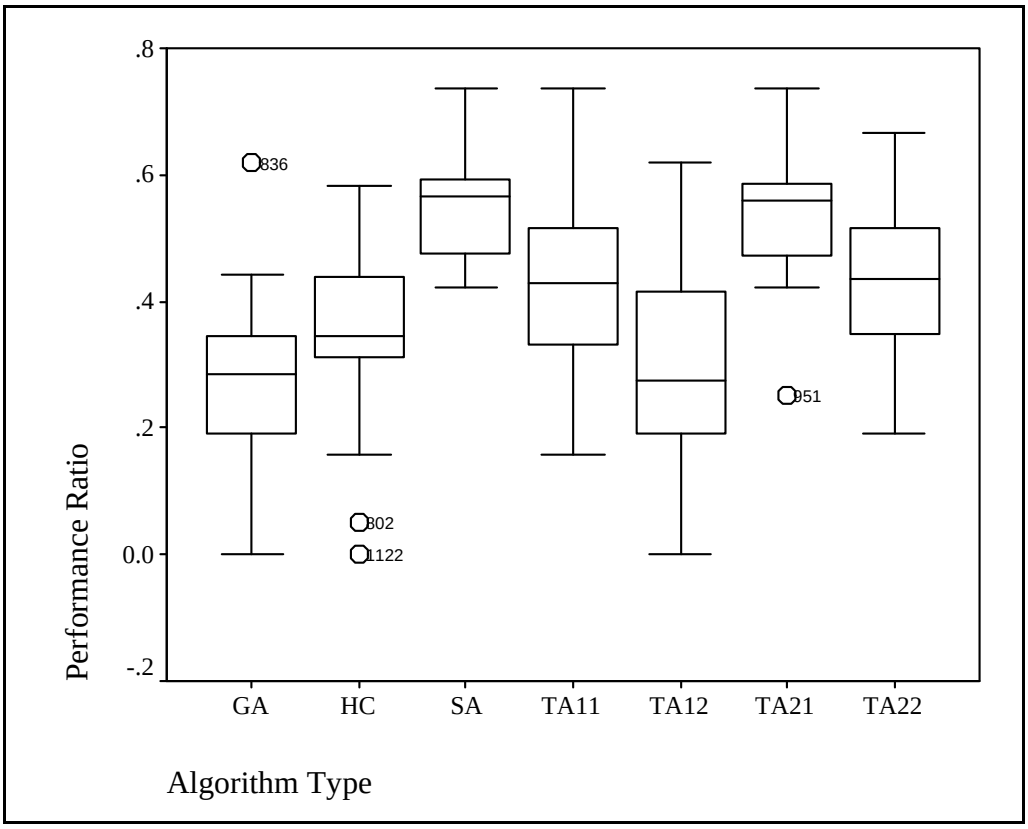


**Exhibit 29.** Box Plot, FAP Size Ten Performance

The line plot for these data appears in Exhibit 30. This plot agrees with the box plot that there is a definite difference between the performances of the different algorithms, and that SA and TA21 are the best performers. It is expected that the ANOVA will confirm this.

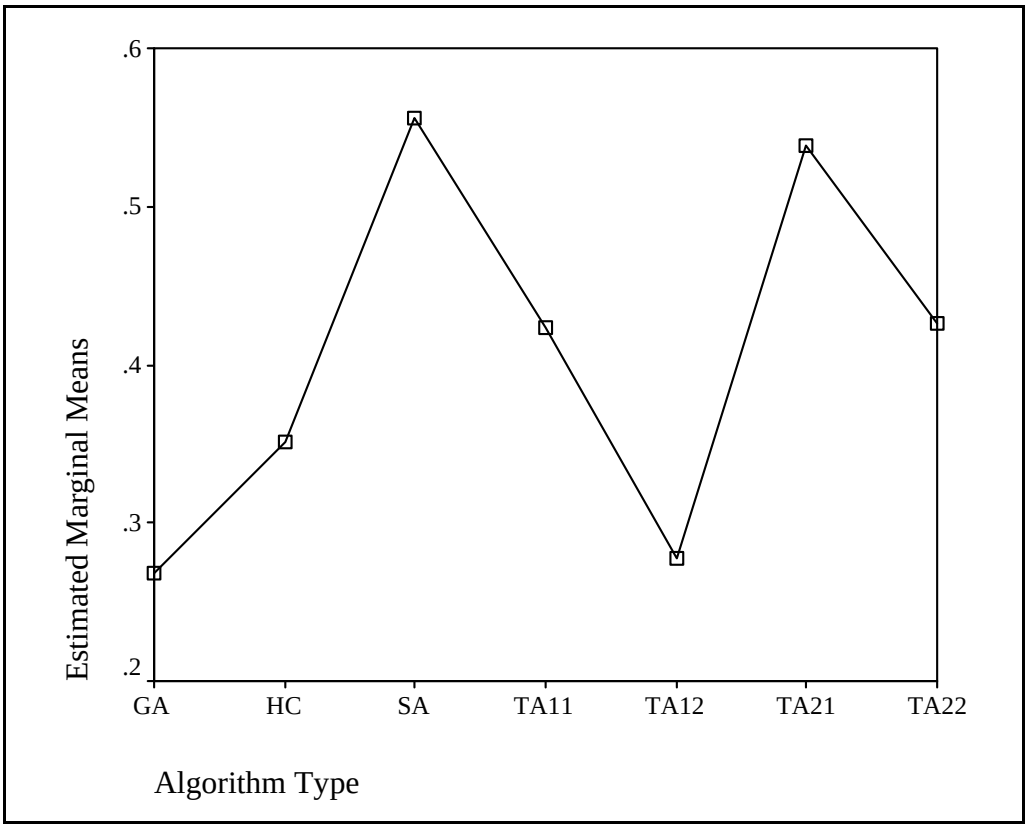


**Exhibit 30.** Line Plot, FAP Size Ten Performance

Exhibit 31 shows the ANOVA results. Once again, the significance values are all zero. The model almost certainly contains variance that cannot be explained by random error, the algorithm type factor is playing a significant role in determining the outcome, and the run number factor is exerting a significant influence and was rightly blocked.

Dependent Variable: Performance Ratio

| Source | Type III Sum of Squares | df | Mean Square | F | Sig. | Partial Eta Squared | Noncent. Parameter | Observed Power[a] |
|---|---|---|---|---|---|---|---|---|
| Corrected Model | 6.829[b] | 55 | .124 | 18.366 | .000 | .775 | 1010.106 | 1.000 |
| Intercept | 57.586 | 1 | 57.586 | 8518.208 | .000 | .967 | 8518.208 | 1.000 |
| Run Number | 2.862 | 49 | 5.841E-02 | 8.640 | .000 | .590 | 423.336 | 1.000 |
| Algorithm Type | 3.967 | 6 | .661 | 97.795 | .000 | .666 | 586.770 | 1.000 |
| Error | 1.988 | 294 | 6.760E-03 | | | | | |
| Total | 66.403 | 350 | | | | | | |
| Corrected Total | 8.816 | 349 | | | | | | |

a. Computed using alpha = .05

b. R Squared = .775 (Adjusted R Squared = .732)

**Exhibit 31.** ANOVA Results, FAP Size Ten Performance

To confirm the results of the ANOVA, the tests of the assumptive conditions were performed. Exhibit 32 shows the P-P plot of the residuals for this group of problems. Most of the points are right on the line, and those that are not are very close to it. This would lead to the belief that the residuals are normally distributed.

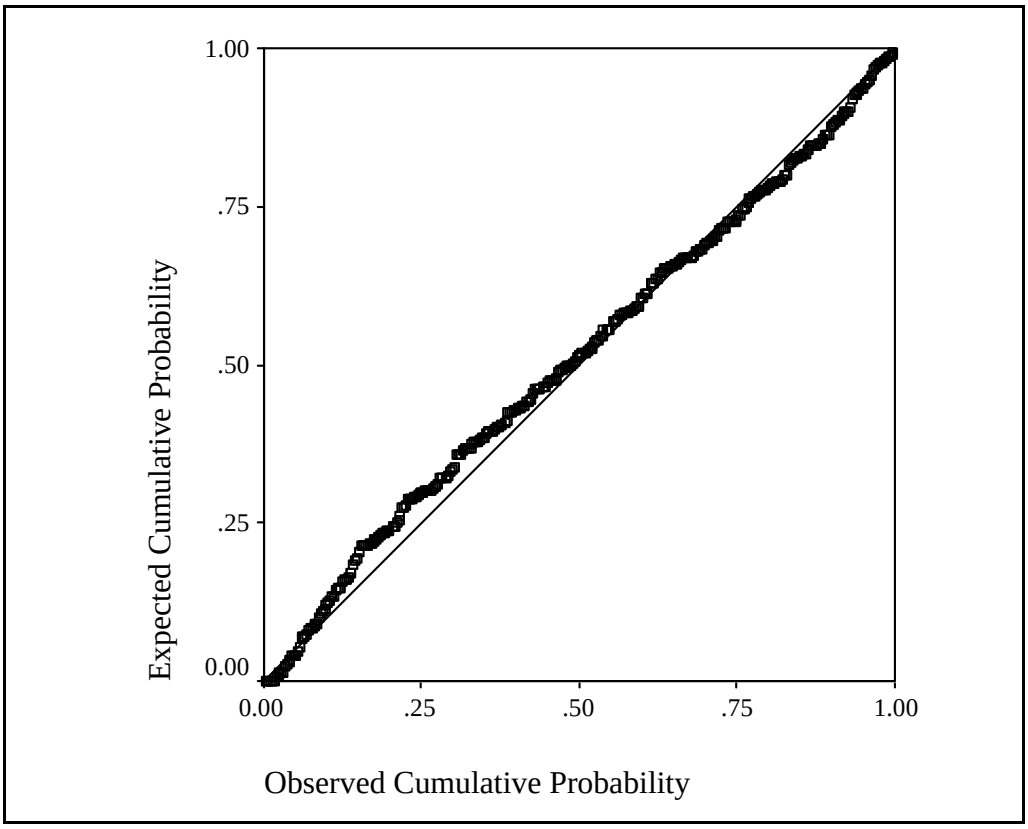


**Exhibit 32.** Residual Normal P-P Plot, FAP Size Ten Performance

The Kolmogorov-Smirnov test for normality of the residuals, shown in Exhibit 33, confirms this belief. The significance value of 0.144 is above the threshold of 0.05, so the test accepts the assumption that the residuals are normally distributed.

| | | Residual for Performance Ratio |
|---|---|---|
| N | | 350 |
| Normal Parameters[a,b] | Mean | .0000000 |
| | Std. Deviation | .07546525 |
| Most Extreme Differences | Absolute | .061 |
| | Positive | .035 |
| | Negative | -.061 |
| Kolmogorov-Smirnov Z | | 1.147 |
| Asymp. Sig. (2-tailed) | | .144 |

a. Test distribution is Normal.

b. Calculated from data.

**Exhibit 33.** Kolmogorov-Smirnov Test, FAP Size Ten Performance

With the assumption of the normality of the residuals upheld, the tests for non-structured residuals were conducted. Exhibit 34 shows the first of these tests, the plot of predicted values versus residuals. Again the ends of the plot appear somewhat pinched, but a close look shows that this effect is due to a small number of individual points. Most of the points are clustered together in a fairly tight arrangement with no indication of a consistent pattern, so this plot should not raise any warning flags.

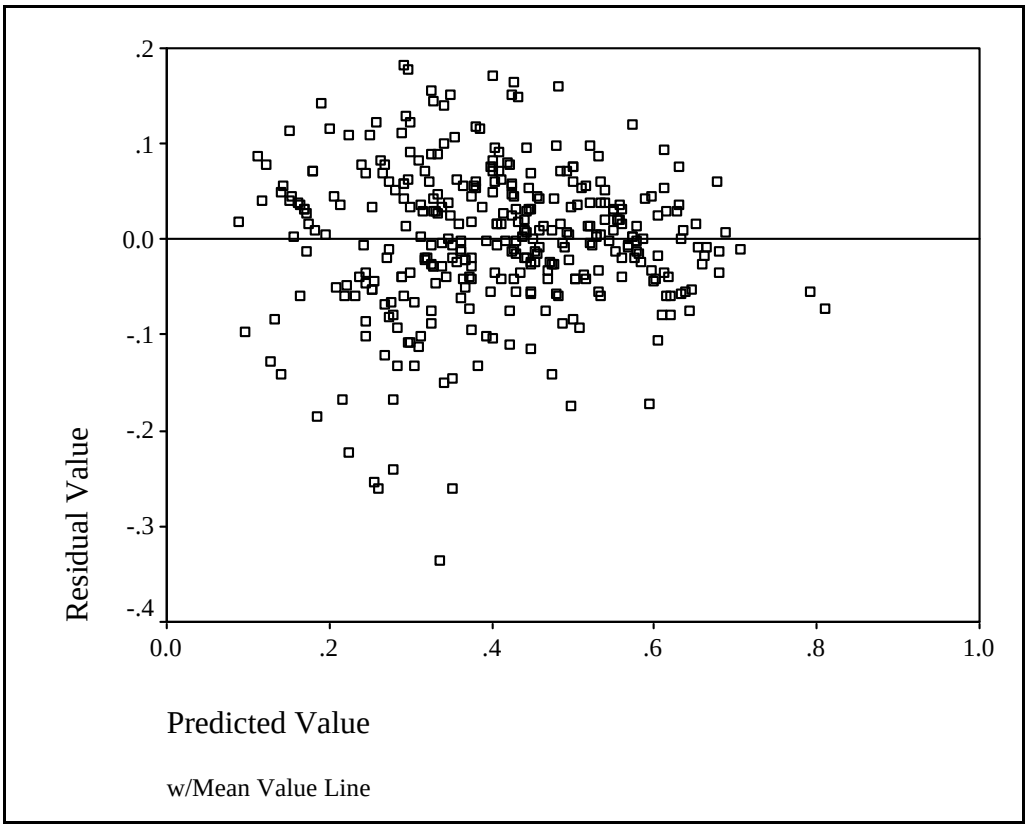


**Exhibit 34.** Predicted vs. Residual Plot, FAP Size Ten Performance

Exhibit 35 shows the plot of residual trend, the other test for non-structured residuals. Again, there are a few outliers, but there is no indication of consistent widening or narrowing, and so there are no warning flags here either. The assumption of non-structured residuals appears reasonable.

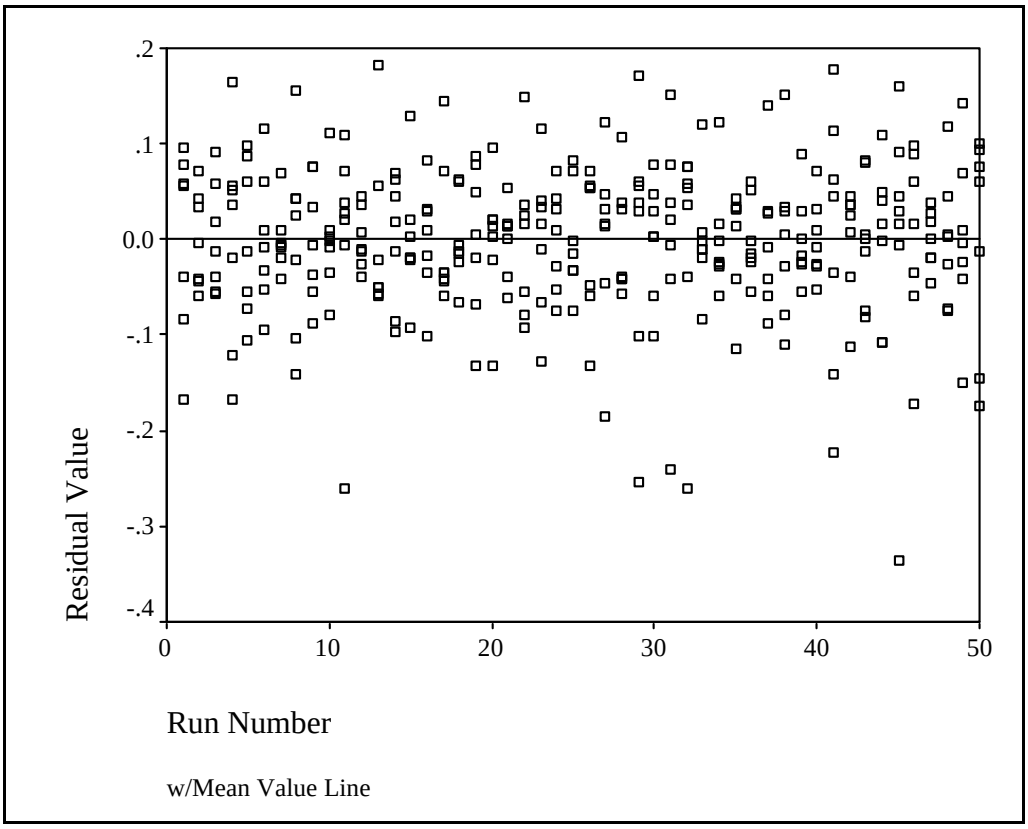


**Exhibit 35.** Residual Trend Plot, FAP Size Ten Performance

Having validated the results of the ANOVA, with its conclusion that there is a difference between the performances of the algorithms, the homogeneous subsets of those differences were generated. Exhibit 36 shows those subsets. Tukey and Duncan are in agreement as to the number and contents of the subsets. Both methods place SA and TA21 in the top subset, TA22 and TA11 in the next subset, HC by itself in the next, and finally TA12 and GA in the lowest subset.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 50 | .2676534 | | | |
| | TA12 | 50 | .2773018 | | | |
| | HC | 50 | | .3508193 | | |
| | TA11 | 50 | | | .4238876 | |
| | TA22 | 50 | | | .4258730 | |
| | TA21 | 50 | | | | .5381334 |
| | SA | 50 | | | | .5557158 |
| | Sig. | | .997 | 1.000 | 1.000 | .937 |
| Duncan[a,b] | GA | 50 | .2676534 | | | |
| | TA12 | 50 | .2773018 | | | |
| | HC | 50 | | .3508193 | | |
| | TA11 | 50 | | | .4238876 | |
| | TA22 | 50 | | | .4258730 | |
| | TA21 | 50 | | | | .5381334 |
| | SA | 50 | | | | .5557158 |
| | Sig. | | .558 | 1.000 | .904 | .286 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 6.760E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 36.** Homogeneous Subsets, FAP Size Ten Performance

The result of the analysis for FAP size ten problem instances indicates that the performance hypothesis can be rejected for this group. There is a significant difference in the performance of the algorithms, with Simulated Annealing and GELS method two with single stepping together claiming the best performance.

To complete the analysis of the size ten problems, the composite analysis of all size ten problem instances over all three problem types was conducted. The box plot for this appears in Exhibit 37. Here the box sizes and whisker lengths for SA

and TA21 appear smaller than for the others, indicating that they have smaller variances in performance, even though both of them have outliers. SA and TA21 also appear slightly higher (and therefore would have better performance), but they also overlap several of the other boxes, so this cannot be considered conclusive. The variances in the performances of GA, TA11, and TA12 are extreme, fully encompassing the entire sizes of the others, pointing to a wide range in performance for those algorithms on problem instances of this type.

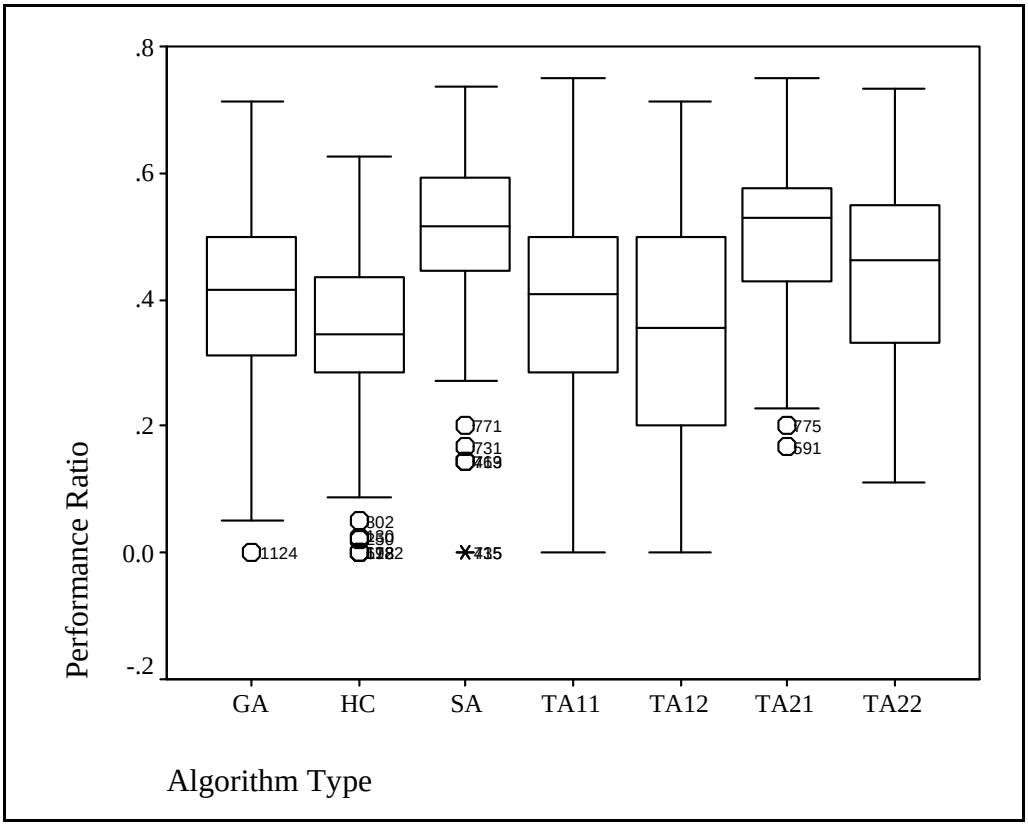


**Exhibit 37.** Box Plot, Composite Size Ten Performance

The composite line plot is shown in Exhibit 38. This plot is even more suggestive of SA and TA21 being the better-performing algorithms for this group of problems than was the box plot. It also suggests that the ANOVA will show the performances of the two to be about the same.

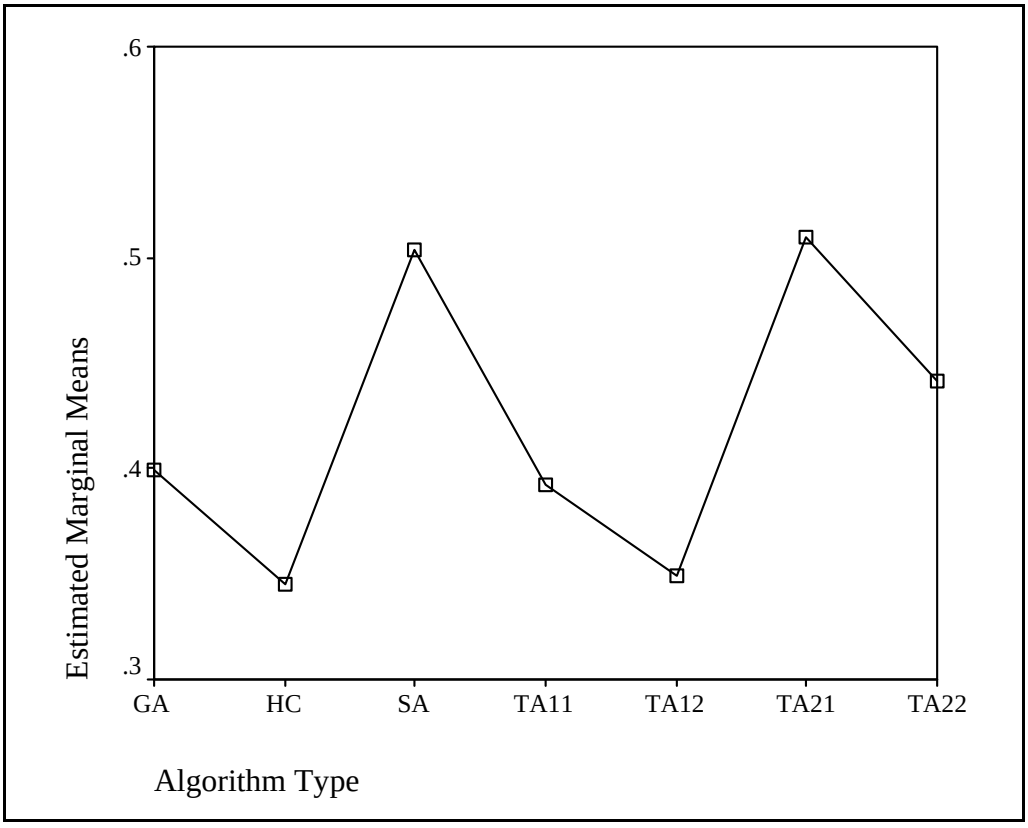


**Exhibit 38.** Line Plot, Composite Size Ten Performance

One other line plot for this group of problems is shown in Exhibit 39. This plot gives a side-by-side comparison of the performances of the algorithms for each problem type. If there was any interaction between the problem type factor and the

algorithm type factor when solving size ten problems, some of the lines would cross. If no such interaction was present, the lines would remain virtually parallel, varying only with the algorithm type.

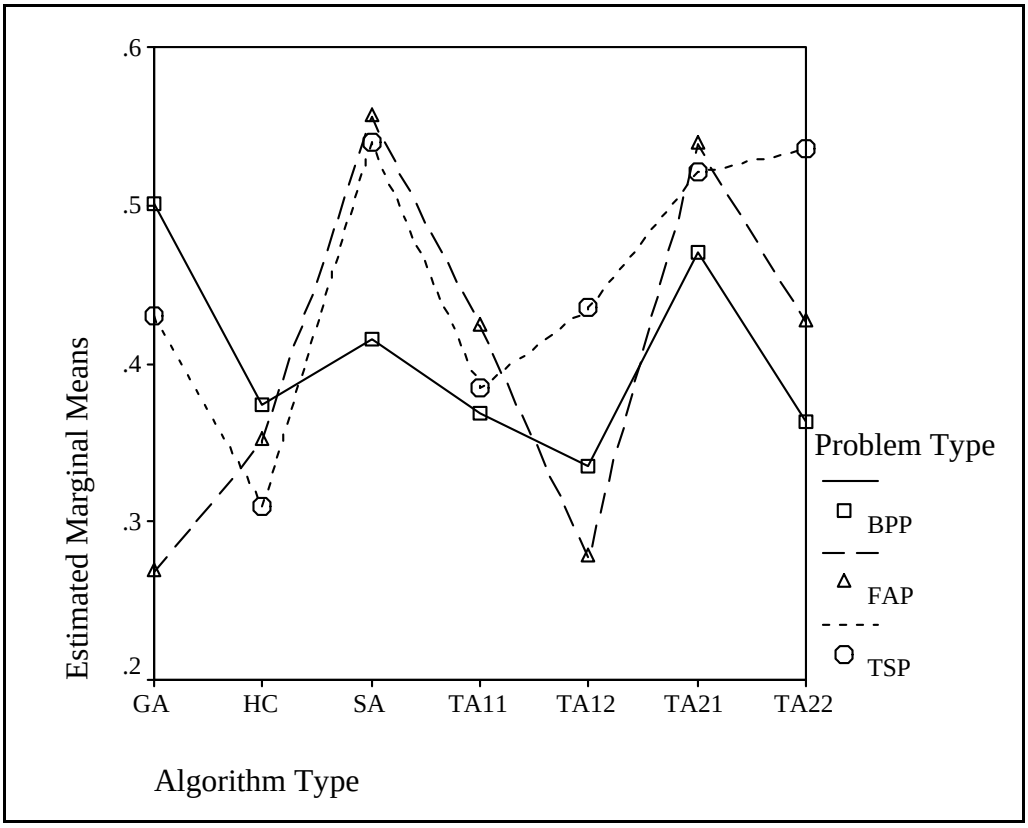


**Exhibit 39.** Problem Type Plot, Composite Size Ten Performance

As can be seen in the plot, there are several points at which the lines cross. This indicates that the algorithms' ability to solve size ten problems is not independent of the type of problem being solved. For instance, looking at GA it appears that if this algorithm is to be used it would most likely produce its best

solutions on a size ten BPP problem, and considerably worse on an FAP problem of the same size. The fact that algorithms perform with different capabilities on different types of problems should come as no surprise. What is noteworthy, however, is that because of the nature of the experiments being conducted, this difference can here be statistically verified and viewed for specific types of problems.

The composite ANOVA for size ten problems is shown in Exhibit 40. All the significance values are zero, including the one for the interaction of the problem type factor with the algorithm type factor. The choice of algorithm for solving size ten problems does have a significant effect on performance, and this effect is not independent of the type of problem being solved.

Dependent Variable: Performance Ratio

| Source | Type III Sum of Squares | df | Mean Square | F | Sig. | Partial Eta Squared | Noncent. Parameter | Observed Power[a] |
|---|---|---|---|---|---|---|---|---|
| Corrected Model | 11.920[b] | 69 | .173 | 12.295 | .000 | .464 | 848.386 | 1.000 |
| Intercept | 185.344 | 1 | 185.344 | 13191.343 | .000 | .931 | 13191.343 | 1.000 |
| Run Number | 4.099 | 49 | 8.365E-02 | 5.954 | .000 | .229 | 291.735 | 1.000 |
| Problem Type | .491 | 2 | .245 | 17.455 | .000 | .034 | 34.911 | 1.000 |
| Algorithm Type | 4.092 | 6 | .682 | 48.543 | .000 | .229 | 291.260 | 1.000 |
| Problem Type vs. Algorithm Type | 3.238 | 12 | .270 | 19.207 | .000 | .190 | 230.480 | 1.000 |
| Error | 13.769 | 980 | 1.405E-02 | | | | | |
| Total | 211.034 | 1050 | | | | | | |
| Corrected Total | 25.690 | 1049 | | | | | | |

a. Computed using alpha = .05

b. R Squared = .464 (Adjusted R Squared = .426)

**Exhibit 40.** ANOVA Results, Composite Size Ten Performance

To verify the results of the ANOVA, the checks on the assumptive conditions were performed. The P-P plot for checking the normality of the residuals is shown in Exhibit 41. The points in this plot look almost like a straight line; there can be little doubt that the residuals are normally distributed. However, the Kolmogorov-Smirnov test was still run to confirm this assertion.

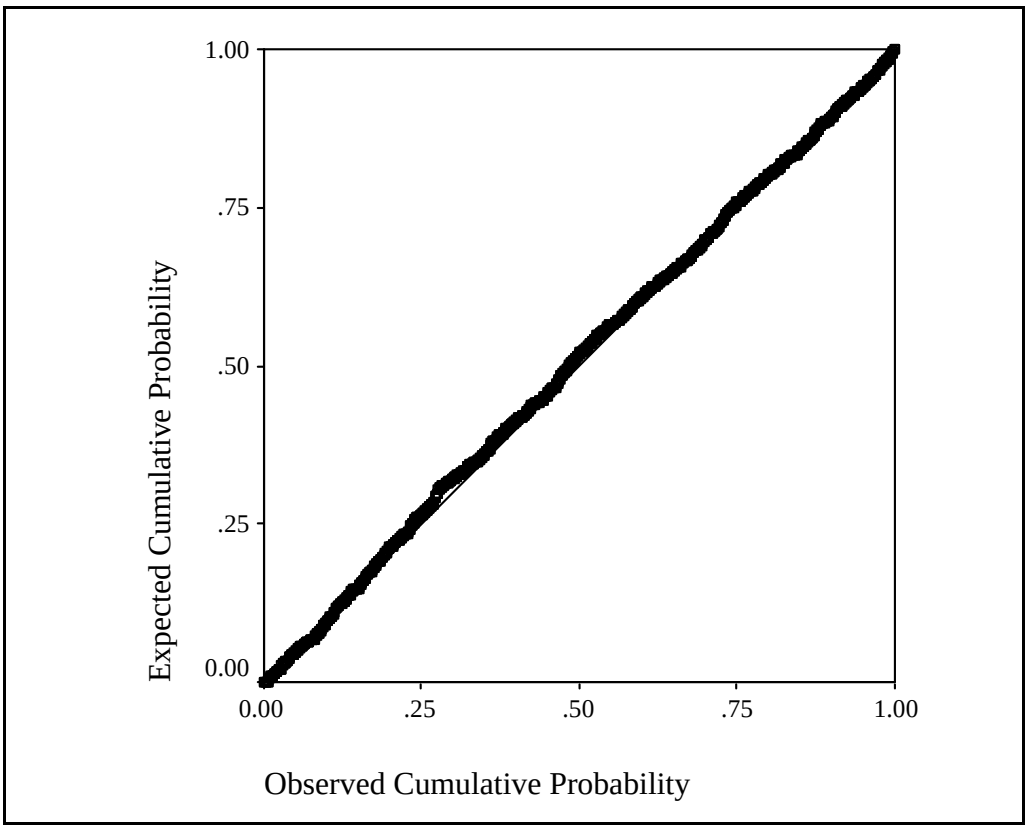


**Exhibit 41.** Residual Normal P-P Plot, Composite Size Ten Performance

The results of this test are given in Exhibit 42. The significance value is 0.382, above the 0.05 threshold as expected. The case for the normality of the residuals is upheld.

| | | Residual for Performance Ratio |
|---|---|---|
| N | | 1050 |
| Normal Parameters[a,b] | Mean | .0000000 |
| | Std. Deviation | .11456977 |
| Most Extreme Differences | Absolute | .028 |
| | Positive | .012 |
| | Negative | -.028 |
| Kolmogorov-Smirnov Z | | .908 |
| Asymp. Sig. (2-tailed) | | .382 |

a. Test distribution is Normal.

b. Calculated from data.

**Exhibit 42.** Kolmogorov-Smirnov Test, Composite Size Ten Performance

Checking the other ANOVA assumptive condition, non-structured residuals, the plot of predicted values versus residuals is shown in Exhibit 43. Once again there is the pinching effect on the ends of the plot, but some of this at least can be explained by the nature of the performance ratio. It cannot exceed one, nor can it go less than zero. Consequently, the farther the performance ratio from mid-range, the less room there is for the residuals to fluctuate. Thus, a narrowing at the extremes of this plot is somewhat to be expected, and certainly no cause for alarm.

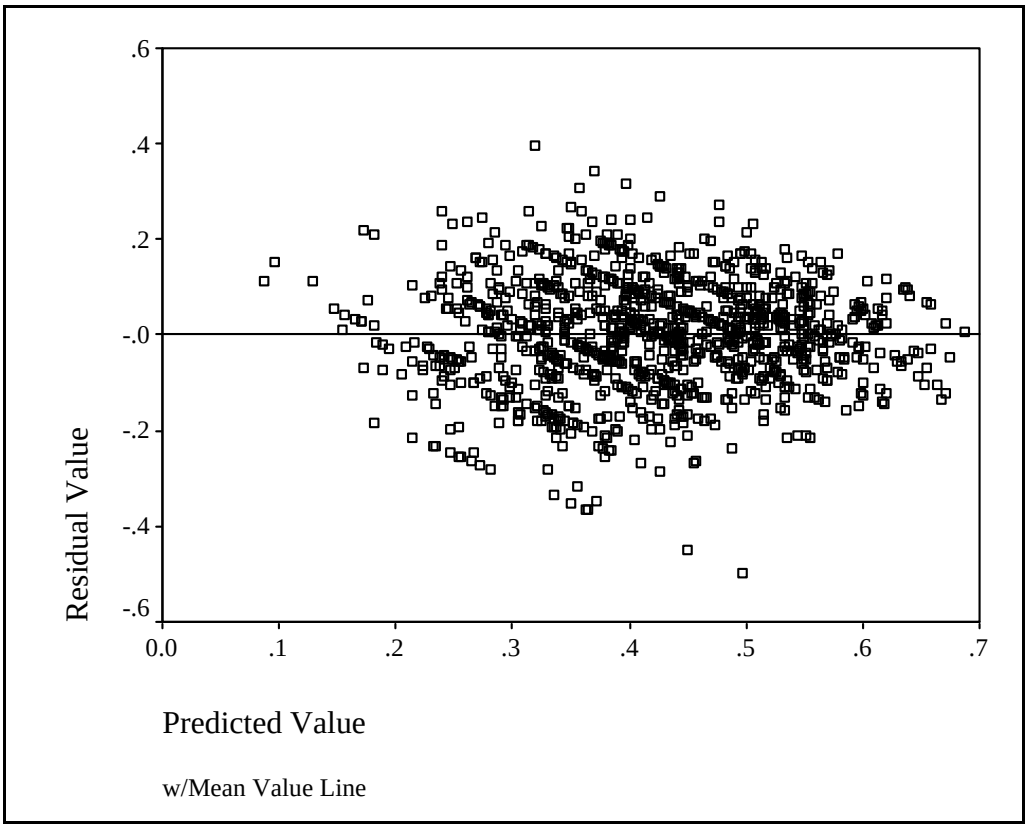


**Exhibit 43.** Predicted vs. Residual Plot, Composite Size Ten Performance

The other check for non-structured residuals, the plot of residual trend, is shown in Exhibit 44. This plot looks almost like a cylinder, with no hint of narrowing or widening, so there is certainly not a problem here.

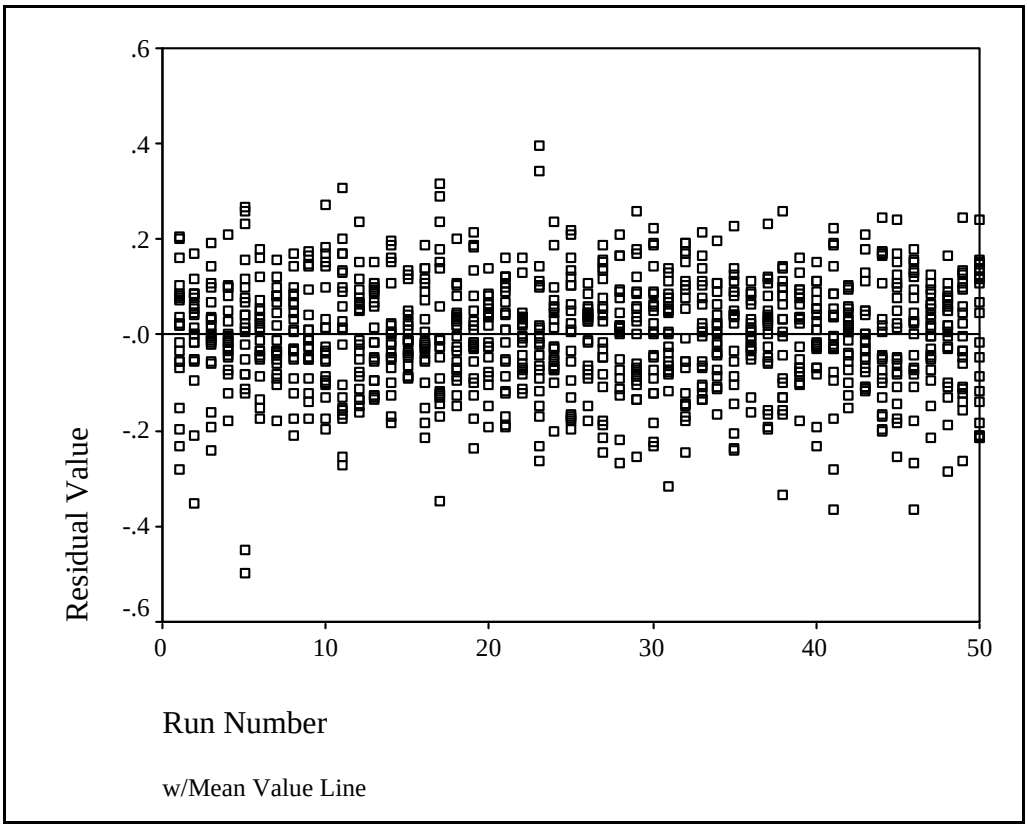


**Exhibit 44.** Residual Trend Plot, Composite Size Ten Performance

With no problems found for either residual normality or residual structures, the results of the ANOVA appear solid. Given this, the homogeneous subsets for the experiment set as a whole were generated. These are shown in Exhibit 45. Tukey's and Duncan's methods are in full agreement in this case. Both have generated four subsets, with TA21 and SA occupying the top spot. In the next subset is TA22 by itself, then GA and TA11 in the next subset, and finally TA12 and HC in the final subset.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | HC | 150 | .3450079 | | | |
| | TA12 | 150 | .3491037 | | | |
| | TA11 | 150 | | .3925091 | | |
| | GA | 150 | | .3997123 | | |
| | TA22 | 150 | | | .4415704 | |
| | SA | 150 | | | | .5032835 |
| | TA21 | 150 | | | | .5097969 |
| | Sig. | | 1.000 | .998 | 1.000 | .999 |
| Duncan[a,b] | HC | 150 | .3450079 | | | |
| | TA12 | 150 | .3491037 | | | |
| | TA11 | 150 | | .3925091 | | |
| | GA | 150 | | .3997123 | | |
| | TA22 | 150 | | | .4415704 | |
| | SA | 150 | | | | .5032835 |
| | TA21 | 150 | | | | .5097969 |
| | Sig. | | .765 | .599 | 1.000 | .634 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.405E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 45.** Homogeneous Subsets, Composite Size Ten Performance

The performance hypothesis has been solidly rejected for the composite set of size ten problems. There is a definite difference in the performances of the algorithms across problem types of this size, and there is a measurable difference between how the algorithms perform on different algorithm types. For problem instances of size ten across all test problems, GELS method two with single stepping was indistinguishable from Simulated Annealing in being the best performers.

#### 2.2.7.1.2 Problem Size Twenty Performance Results

The next portion of the analysis involved the consideration of problem instances of size twenty. All of the methods, tests, tools, and diagrams used in performing this part of the analysis were exactly the same as were used in the analysis of the set of problem instances of size ten. Such was also the case for the examination of the other remaining experiment sets (size thirty, size forty, size fifty, and the random size problem instances). Displaying the individual results of all the tests would require many hundreds of pages; consequently, the diagrams of the tests have been omitted, and the summary results will be given.

For TSP problem instances of size twenty, the analysis showed that the performance hypothesis could be rejected for this group of problems. With a significance value of zero in the ANOVA, the algorithm type factor demonstrated that it had a significant effect on performance. The ANOVA results were verified through testing of the residuals, and the homogeneous subsets were calculated.

The subsets produced are shown in Exhibit 46. From this diagram, it can be seen that though the count of the subsets generated by the two methods was different, the ordering of the algorithms within them was not. The only difference between them was that Tukey's method put GELS method two with multiple stepping in the same subset with GELS method two with single stepping, while Duncan's method separated them. Both agreed that the best performer for this group of problem instances was Simulated Annealing. The best performing of the

GELS combinations were method two with multiple stepping and method two with single stepping, which came in second and third, respectively, in the Duncan method, and were tied for second best performance in the Tukey method.

| | | | Subset | | | | |
|---|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 | 5 |
| Tukey HSD[a,b] | GA | 50 | .3120485 | | | | |
| | HC | 50 | .3148042 | | | | |
| | TA11 | 50 | .3214439 | | | | |
| | TA12 | 50 | | .3518877 | | | |
| | TA21 | 50 | | | .4277217 | | |
| | TA22 | 50 | | | .4560472 | | |
| | SA | 50 | | | | .5674781 | |
| | Sig. | | .966 | 1.000 | .072 | 1.000 | |
| Duncan[a,b] | GA | 50 | .3120485 | | | | |
| | HC | 50 | .3148042 | | | | |
| | TA11 | 50 | .3214439 | | | | |
| | TA12 | 50 | | .3518877 | | | |
| | TA21 | 50 | | | .4277217 | | |
| | TA22 | 50 | | | | .4560472 | |
| | SA | 50 | | | | | .5674781 |
| | Sig. | | .380 | 1.000 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 2.499E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 46.** Homogeneous Subsets, TSP Size Twenty Performance

For BPP problem instances of size twenty, the performance hypothesis was easily rejected again. The ANOVA results, verified by checks on the assumptions, showed all zeroes for the significance values, indicating significant effect of algorithm type on performance. When constructing the homogeneous subsets, this time Tukey's method created three subsets, and Duncan's method created five. The

results are shown in Exhibit 47. Once again, though the overall number of subsets differed between the two methods, both agreed that the best performers for this group of problem instances were the Genetic Algorithm and GELS method two with single stepping together.

| | | | Subset | | | | |
|---|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 | 5 |
| Tukey HSD[a,b] | TA12 | 50 | .1953327 | | | | |
| | TA22 | 50 | .2396508 | | | | |
| | TA11 | 50 | | .2974454 | | | |
| | SA | 50 | | .3332408 | | | |
| | HC | 50 | | .3409857 | | | |
| | TA21 | 50 | | | .4217291 | | |
| | GA | 50 | | | .4221470 | | |
| | Sig. | | .121 | .135 | 1.000 | | |
| Duncan[a,b] | TA12 | 50 | .1953327 | | | | |
| | TA22 | 50 | | .2396508 | | | |
| | TA11 | 50 | | | .2974454 | | |
| | SA | 50 | | | | .3332408 | |
| | HC | 50 | | | | .3409857 | |
| | TA21 | 50 | | | | | .4217291 |
| | GA | 50 | | | | | .4221470 |
| | Sig. | | 1.000 | 1.000 | 1.000 | .646 | .980 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 7.101E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 47.** Homogeneous Subsets, BPP Size Twenty Performance

For FAP problem instances of size twenty, the ANOVA results (verified by checks on the assumptions) showed that the performance hypothesis could be safely rejected for this group of problem instances. The homogeneous subsets generated are given in Exhibit 48. This time both methods agreed on the number of

subsets, but not on the content for each individual subset. Nevertheless, both agreed on the contents of the top subset, naming the best performers for this group of problems as a three-way tie between Simulated Annealing, GELS method two with multiple stepping, and GELS method one with multiple stepping.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | GA | 50 | .3397125 | | | |
| | HC | 50 | .4023753 | | | |
| | TA11 | 50 | | .6658073 | | |
| | TA21 | 50 | | .7057492 | .7057492 | |
| | TA12 | 50 | | | .7790635 | .7790635 |
| | TA22 | 50 | | | | .7925096 |
| | SA | 50 | | | | .7996660 |
| | Sig. | | .317 | .812 | .151 | .992 |
| Duncan[a,b] | GA | 50 | .3397125 | | | |
| | HC | 50 | | .4023753 | | |
| | TA11 | 50 | | | .6658073 | |
| | TA21 | 50 | | | .7057492 | |
| | TA12 | 50 | | | | .7790635 |
| | TA22 | 50 | | | | .7925096 |
| | SA | 50 | | | | .7996660 |
| | Sig. | | 1.000 | 1.000 | .168 | .506 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 2.092E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 48.** Homogeneous Subsets, FAP Size Twenty Performance

In looking at the composite view of all size twenty problem instances, the ANOVA once again rejected the performance hypothesis, meaning that it

determined that for any given problem instance of size twenty there is a significant difference in performance between the algorithm types, regardless of which of the test problem types is being solved. The interesting item here is that while the assumption of normal distribution of the residuals held for all three problem types individually, the Kolmogorov-Smirnov test rejected this assumption for the composite case. This meant that the ANOVA results had to be corroborated by additional evidence. Fortunately, the results of the Kruskal-Wallis test conducted on the algorithms' performances concurred with those of the ANOVA, providing that necessary corroboration.

Having affirmed the difference in performance between the algorithms, the homogeneous subsets were generated, and the results are shown in Exhibit 49. This time the two methods were in complete agreement, creating four subsets and naming the overall best performer for the size twenty set of experiments to be Simulated Annealing. The best performances of the GELS combinations were GELS method two with single stepping and GELS method two with multiple stepping, which finished together in the second spot.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | HC | 150 | .3527217 | | | |
| | GA | 150 | .3579693 | | | |
| | TA11 | 150 | | .4282322 | | |
| | TA12 | 150 | | .4420946 | | |
| | TA22 | 150 | | | .4960692 | |
| | TA21 | 150 | | | .5184000 | |
| | SA | 150 | | | | .5667950 |
| | Sig. | | 1.000 | .970 | .757 | 1.000 |
| Duncan[a,b] | HC | 150 | .3527217 | | | |
| | GA | 150 | .3579693 | | | |
| | TA11 | 150 | | .4282322 | | |
| | TA12 | 150 | | .4420946 | | |
| | TA22 | 150 | | | .4960692 | |
| | TA21 | 150 | | | .5184000 | |
| | SA | 150 | | | | .5667950 |
| | Sig. | | .728 | .359 | .139 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.710E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 49.** Homogeneous Subsets, Composite Size Twenty Performance

The results of the ANOVA also asserted a significant effect of problem type on algorithm performance, indicating the presence of some interaction between the problem type and algorithm type factors. A visual depiction of the interaction detected is shown in Exhibit 50. While there is not as much interaction as was shown for size ten problem instances, some line crossing can be seen, as can notably better performances obtained overall for FAP problem instances than for instances of the other problem types.

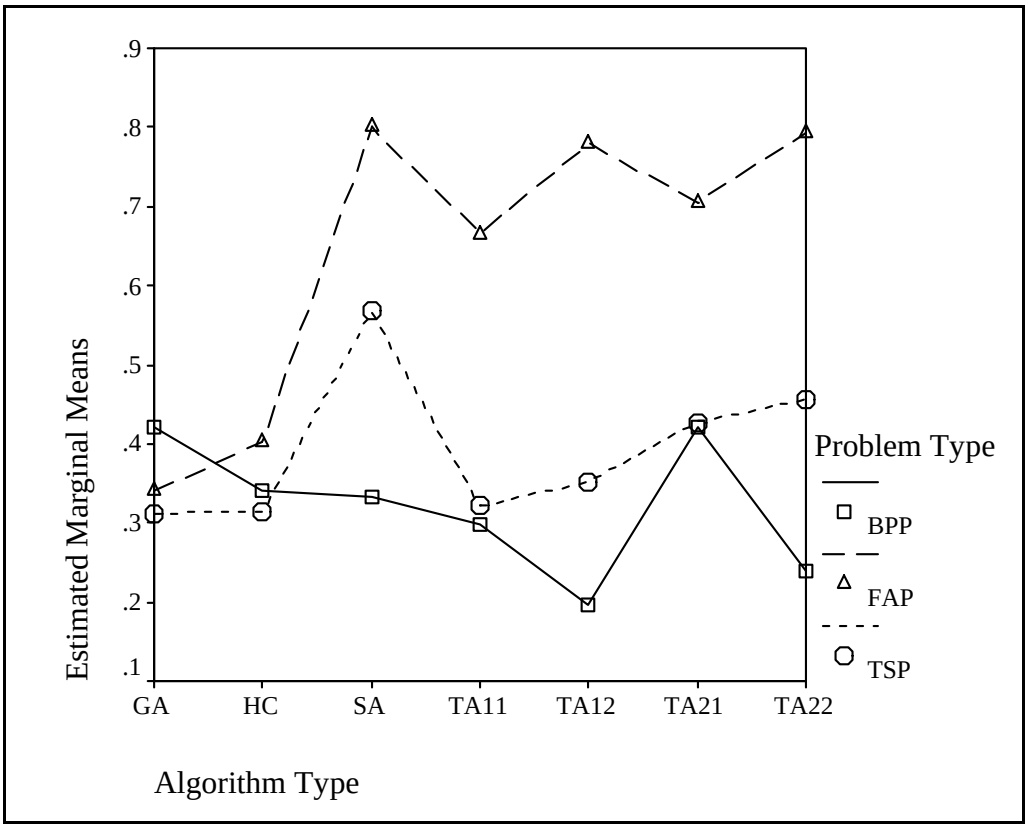


**Exhibit 50.** Problem Type Plot, Composite Size Twenty Performance

## 2.2.7.1.3 Problem Size Thirty Performance Results

The next set of experiments to be analyzed was the set of problem instances of size thirty. For TSP problem instances of this size, the ANOVA results, confirmed by checks of the assumptions, showed that the algorithm type factor was having a significant effect on performance, thus rejecting the performance hypothesis for this group of problems. In constructing the homogeneous subsets, the Tukey and Duncan methods were in complete agreement on the arrangement. The sets they generated are shown in Exhibit 51. Both generated five subsets, and gave the nod

to Simulated Annealing as the best performer for this group of problems. The best result for GELS was for method two with multiple stepping, which was placed in the second best subset.

| | | | Subset | | | | |
|---|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 | 5 |
| Tukey HSD[a,b] | GA | 50 | .2541734 | | | | |
| | TA11 | 50 | .2583976 | | | | |
| | TA12 | 50 | | .3055335 | | | |
| | HC | 50 | | .3218444 | | | |
| | TA21 | 50 | | | .3591655 | | |
| | TA22 | 50 | | | | .3900728 | |
| | SA | 50 | | | | | .5640582 |
| | Sig. | | .999 | .491 | 1.000 | 1.000 | 1.000 |
| Duncan[a,b] | GA | 50 | .2541734 | | | | |
| | TA11 | 50 | .2583976 | | | | |
| | TA12 | 50 | | .3055335 | | | |
| | HC | 50 | | .3218444 | | | |
| | TA21 | 50 | | | .3591655 | | |
| | TA22 | 50 | | | | .3900728 | |
| | SA | 50 | | | | | .5640582 |
| | Sig. | | .626 | .060 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.870E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 51.** Homogeneous Subsets, TSP Size Thirty Performance

For the BPP problem instances of size thirty, the ANOVA rejected the performance hypothesis for this group of problems, asserting that the choice of algorithm did have a significant effect on the performance results. The ANOVA results were verified by checks on the assumptions, and the homogeneous subsets were generated. The results of this are shown in Exhibit 52. Tukey's method

produced four subsets, while Duncan's method produced five. The Tukey method also named GELS method two with single stepping to be the best performer for this group of problems, in a tie with the Genetic Algorithm. The Duncan method, however, differentiated between the two, naming GELS method two with single stepping to the top spot by itself and placing the Genetic Algorithm in second place. In either case, though, GELS method two with single stepping was indicated as the top performer for this group of problems.

| | | | Subset | | | | |
|---|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 | 5 |
| Tukey HSD[a,b] | TA12 | 50 | .1341031 | | | | |
| | TA22 | 50 | .1762473 | | | | |
| | TA11 | 50 | | .2510508 | | | |
| | SA | 50 | | .2731156 | | | |
| | HC | 50 | | | .3443429 | | |
| | GA | 50 | | | .3744462 | .3744462 | |
| | TA21 | 50 | | | | .4228052 | |
| | Sig. | | .216 | .878 | .622 | .098 | |
| Duncan[a,b] | TA12 | 50 | .1341031 | | | | |
| | TA22 | 50 | | .1762473 | | | |
| | TA11 | 50 | | | .2510508 | | |
| | SA | 50 | | | .2731156 | | |
| | HC | 50 | | | | .3443429 | |
| | GA | 50 | | | | .3744462 | |
| | TA21 | 50 | | | | | .4228052 |
| | Sig. | | 1.000 | 1.000 | .216 | .092 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 7.921E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 52.** Homogeneous Subsets, BPP Size Thirty Performance

For the FAP problems of size thirty, the ANOVA again indicated that the algorithm type factor was having a significant effect on performance for this group of problems, and therefore the performance hypothesis should be rejected for this group of problems. These results were confirmed by the checks on the assumptions, and the homogeneous subsets were generated. The results of this are given in Exhibit 53. Tukey's method generated three subsets, while Duncan's method generated four, but both methods agreed on a three-way tie between Simulated Annealing, GELS method two with single stepping, and GELS method two with multiple stepping for the best performing algorithm for this group of problems.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 50 | .2800623 | | | |
| | HC | 50 | | .3974322 | | |
| | TA11 | 50 | | .4094645 | | |
| | TA12 | 50 | | .4505783 | | |
| | TA22 | 50 | | | .5350326 | |
| | TA21 | 50 | | | .5428214 | |
| | SA | 50 | | | .5700861 | |
| | Sig. | | 1.000 | .297 | .772 | |
| Duncan[a,b] | GA | 50 | .2800623 | | | |
| | HC | 50 | | .3974322 | | |
| | TA11 | 50 | | .4094645 | .4094645 | |
| | TA12 | 50 | | | .4505783 | |
| | TA22 | 50 | | | | .5350326 |
| | TA21 | 50 | | | | .5428214 |
| | SA | 50 | | | | .5700861 |
| | Sig. | | 1.000 | .618 | .089 | .173 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.455E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 53.** Homogeneous Subsets, FAP Size Thirty Performance

For the composite of all problem instances of size thirty, the ANOVA indicated that in the overall case for this set of experiments the performance hypothesis should be rejected. It found a significant effect of algorithm type on performance, and these results were confirmed through checks on the assumptive conditions of the ANOVA. The homogeneous subsets generated are shown in Exhibit 54. In this case both the Tukey and the Duncan methods are in complete agreement. Both created three subsets, and both named Simulated Annealing and GELS method two

with single stepping to be in a tie for the overall best performer in the set of problem instances of size thirty.

| | | | Subset | | |
|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 |
| Tukey HSD[a,b] | TA12 | 150 | .2967383 | | |
| | GA | 150 | .3028940 | | |
| | TA11 | 150 | .3063043 | | |
| | HC | 150 | | .3545398 | |
| | TA22 | 150 | | .3671176 | |
| | TA21 | 150 | | | .4415974 |
| | SA | 150 | | | .4690866 |
| | Sig. | | .996 | .984 | .565 |
| Duncan[a,b] | TA12 | 150 | .2967383 | | |
| | GA | 150 | .3028940 | | |
| | TA11 | 150 | .3063043 | | |
| | HC | 150 | | .3545398 | |
| | TA22 | 150 | | .3671176 | |
| | TA21 | 150 | | | .4415974 |
| | SA | 150 | | | .4690866 |
| | Sig. | | .565 | .417 | .076 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.798E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 54.** Homogeneous Subsets, Composite Size Thirty Performance

The ANOVA also found a significant effect of the problem type on the performances of the algorithms. A graph of this is shown in Exhibit 55. It shows several places where the lines cross, indicating the interaction between the problem

type being solved and the algorithm being used to solve it. The lines for FAP and TSP problem instances have a similar shape, suggesting that these two problem types have similar effects on the algorithms. The line for BPP problem instances, on the other hand, has nearly a completely different shape, suggesting an entirely different effect on algorithms attempting to solve them. That being the case, it would suggest further that a different selection of algorithm for solving BPP problem instances might be apropos. One counter-indication to this is the spike on BPP for GELS method two with single stepping, showing its high level of performance for those problem instances, and its comparable level of performance on instances of the other problem types.

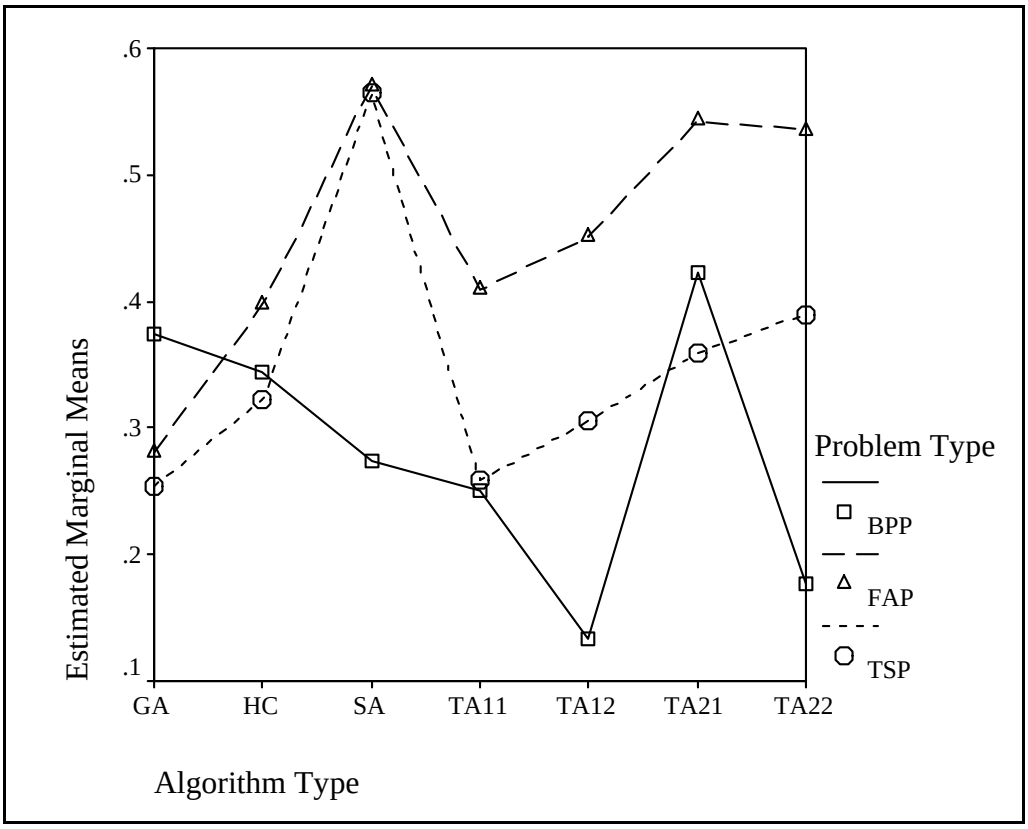


**Exhibit 55.** Problem Type Plot, Composite Size Thirty Performance

## 2.2.7.1.4 Problem Size Forty Performance Results

Next to be analyzed was the set of problem instances of size forty. For the TSP problem instances in this group, the confirmed ANOVA results led to the rejection of the performance hypothesis for this group of problem instances. The homogeneous subsets were generated, giving the results shown in Exhibit 56. Once again Tukey's method generated one fewer subset than did Duncan's method, but both agreed that the top performer for this group of problem instances was Simulated Annealing, with GELS method two with multiple stepping being the

highest rated of the GELS combinations, in the second slot. Of note here is that the Duncan method found enough difference between each of the algorithms that it was almost willing to put each one in its own subset, with only Hill Climbing and GELS method two with single stepping being placed together.

| | | | Subset | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 | 5 | 6 |
| Tukey HSD[a,b] | GA | 50 | .2427152 | | | | | |
| | TA11 | 50 | | .2675271 | | | | |
| | TA12 | 50 | | .2884991 | | | | |
| | TA21 | 50 | | | .3317160 | | | |
| | HC | 50 | | | .3370489 | | | |
| | TA22 | 50 | | | | .3637347 | | |
| | SA | 50 | | | | | .5688824 | |
| | Sig. | | 1.000 | .067 | .991 | 1.000 | 1.000 | |
| Duncan[a,b] | GA | 50 | .2427152 | | | | | |
| | TA11 | 50 | | .2675271 | | | | |
| | TA12 | 50 | | | .2884991 | | | |
| | TA21 | 50 | | | | .3317160 | | |
| | HC | 50 | | | | .3370489 | | |
| | TA22 | 50 | | | | | .3637347 | |
| | SA | 50 | | | | | | .5688824 |
| | Sig. | | 1.000 | 1.000 | 1.000 | .467 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.342E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 56.** Homogeneous Subsets, TSP Size Forty Performance

For the size forty BPP problem instance group, the ANOVA indicated rejection of the performance hypothesis. However, it failed the test for normality of the residuals, registering only a 0.022 significance rating on the Kolmogorov-Smirnov test. Needing additional evidence to corroborate the ANOVA results, a Kruskal-Wallis test was run. This test was able to affirm the results of the ANOVA with a

perfect score, providing reassurance of its conclusions. Having corroborated the ANOVA results, the homogeneous subsets were generated, and are shown in Exhibit 57. Again, the Tukey method produced one fewer subset than the Duncan method. Both methods placed GELS method two with single stepping as the best performer for this group of problems, with Duncan's method having placing it there by itself and Tukey's method matching it with Hill Climbing (a rather surprising finish for HC given its performance in the other problem instance groups).

| | | | Subset | | | | |
|---|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 | 5 |
| Tukey HSD[a,b] | TA12 | 50 | .1126178 | | | | |
| | TA22 | 50 | .1605698 | | | | |
| | SA | 50 | | .2357028 | | | |
| | TA11 | 50 | | .2461047 | | | |
| | GA | 50 | | | .3333715 | | |
| | HC | 50 | | | .3573521 | .3573521 | |
| | TA21 | 50 | | | | .3998529 | |
| | Sig. | | .149 | .998 | .865 | .272 | |
| Duncan[a,b] | TA12 | 50 | .1126178 | | | | |
| | TA22 | 50 | | .1605698 | | | |
| | SA | 50 | | | .2357028 | | |
| | TA11 | 50 | | | .2461047 | | |
| | GA | 50 | | | | .3333715 | |
| | HC | 50 | | | | .3573521 | |
| | TA21 | 50 | | | | | .3998529 |
| | Sig. | | 1.000 | 1.000 | .582 | .205 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 8.911E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 57.** Homogeneous Subsets, BPP Size Forty Performance

For the FAP problem instances of size forty, the ANOVA again rejected the performance hypothesis for this group of problems. This time, the ANOVA results were confirmed by check of the assumptive conditions, and the homogeneous subsets were generated. The results of this are shown in Exhibit 58. For the first time, there were only two subsets generated, identical between the Tukey and Duncan methods. In the subset indicative of best performance for this group of problem instances, Simulated Annealing was grouped together with all four of the GELS combinations, all statistically indistinguishable.

| | Algorithm Type | N | Subset 1 | Subset 2 |
|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 50 | .3056053 | |
| | HC | 50 | .3491229 | |
| | TA12 | 50 | | .5738065 |
| | TA11 | 50 | | .5805035 |
| | TA21 | 50 | | .6003191 |
| | TA22 | 50 | | .6180262 |
| | SA | 50 | | .6200262 |
| | Sig. | | .638 | .569 |
| Duncan[a,b] | GA | 50 | .3056053 | |
| | HC | 50 | .3491229 | |
| | TA12 | 50 | | .5738065 |
| | TA11 | 50 | | .5805035 |
| | TA21 | 50 | | .6003191 |
| | TA22 | 50 | | .6180262 |
| | SA | 50 | | .6200262 |
| | Sig. | | .096 | .117 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.703E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 58.** Homogeneous Subsets, FAP Size Forty Performance

The composite analysis for all problem instances of size forty was then performed. The ANOVA indicated a difference in performance between the algorithms across problem types. However, it completely failed the check of normality of the residuals, scoring a zero on the Kolmogorov-Smirnov test. Fortunately, a Kruskal-Wallis test comparing the algorithm types found solid evidence of its significance as a factor, bolstering the ANOVA results. Since the

non-parametric test had agreed with the ANOVA, there was sufficient evidence of difference between the performances of the algorithms to warrant rejection of the performance hypothesis for the size forty experiment set, and the homogeneous subsets were generated as shown in Exhibit 59. This time there was a two-subset difference between the Tukey and the Duncan methods, with the Duncan method again being more discriminatory in its selection of subsets than was Tukey. Duncan's method put Simulated Annealing alone at the top, with GELS method two with single stepping in the second spot, while the Tukey method could not distinguish between those them as the best performers for this set of experiments.

| | | | Subset | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 | 5 | 6 |
| Tukey HSD[a,b] | GA | 150 | .2938973 | | | | | |
| | TA12 | 150 | .3249745 | .3249745 | | | | |
| | HC | 150 | | .3478413 | .3478413 | | | |
| | TA11 | 150 | | .3647117 | .3647117 | | | |
| | TA22 | 150 | | | .3807769 | | | |
| | TA21 | 150 | | | | .4439627 | | |
| | SA | 150 | | | | .4748705 | | |
| | Sig. | | .275 | .065 | .211 | .282 | | |
| Duncan[a,b] | GA | 150 | .2938973 | | | | | |
| | TA12 | 150 | | .3249745 | | | | |
| | HC | 150 | | .3478413 | .3478413 | | | |
| | TA11 | 150 | | | .3647117 | .3647117 | | |
| | TA22 | 150 | | | | .3807769 | | |
| | TA21 | 150 | | | | | .4439627 | |
| | SA | 150 | | | | | | .4748705 |
| | Sig. | | 1.000 | .100 | .224 | .247 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.444E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 59.** Homogeneous Subsets, Composite Size Forty Performance

Again, the ANOVA results also found that the performance of the algorithms was not independent of the problem type being solved, and that there was interaction between the two factors as shown in Exhibit 60. There are several points where the lines cross, highlighting the interaction between problem type and algorithm performance, and there is that same general shape between the FAP and TSP lines, with a markedly different shape to the BPP line, as seen previously, again indicating a different effect on algorithm performance from BPP problem instances than from instances of the other problem types.

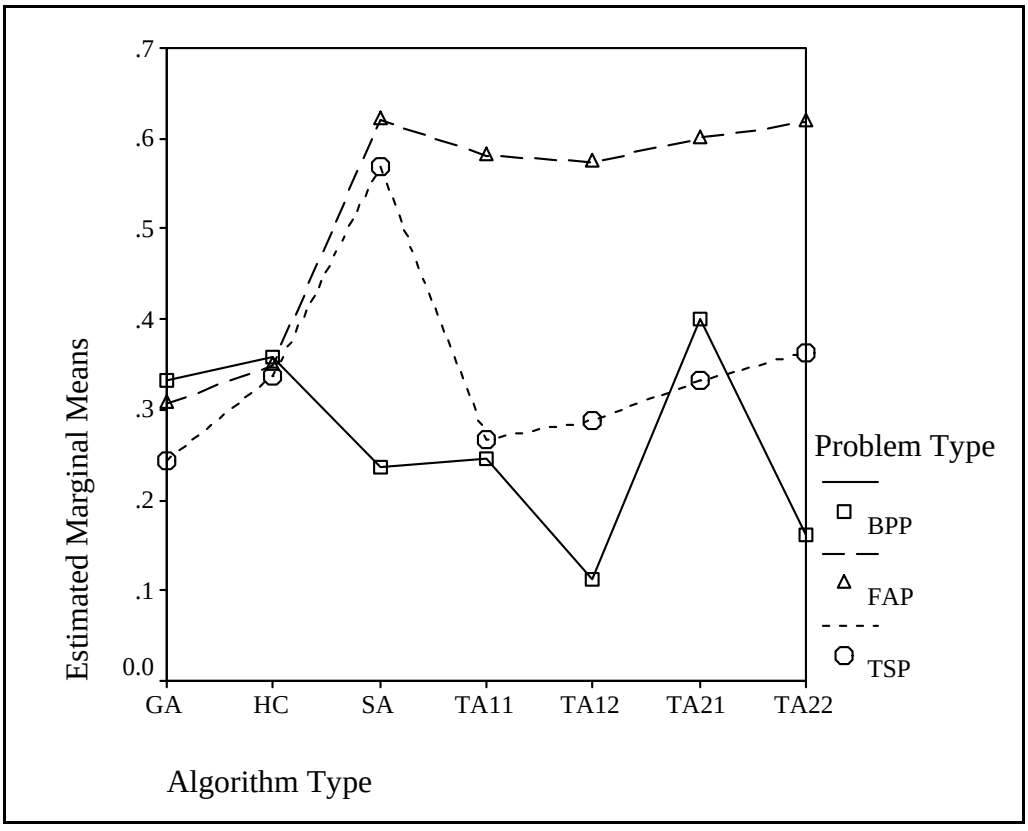


**Exhibit 60.** Problem Type Plot, Composite Size Forty Performance

#### 2.2.7.1.5 Problem Size Fifty Performance Results

The set of experiments consisting of problem instances of size fifty was next analyzed. For the TSP problem instances of this size, the ANOVA indicated rejection of the performance hypothesis for this group of problem instances. These results were confirmed through the necessary checks, and the list of homogeneous subsets was generated. Exhibit 61 shows this list. While once before the Duncan procedure was almost prepared to place each algorithm in its own subset, this time it actually did do so. Even the Tukey method placed only two of the algorithms in the same subset. Both methods put Simulated Annealing in the top spot, with the best performing GELS combination being method two with multiple stepping, in third place in Duncan and tied for second in Tukey.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 50 | .1944183 | | | | | | |
| | TA11 | 50 | | .2372775 | | | | | |
| | TA12 | 50 | | | .2608414 | | | | |
| | TA21 | 50 | | | | .2882691 | | | |
| | TA22 | 50 | | | | | .3119696 | | |
| | HC | 50 | | | | | .3329833 | | |
| | SA | 50 | | | | | | .5478063 | |
| | Sig. | | 1.000 | 1.000 | 1.000 | 1.000 | .057 | 1.000 | |
| Duncan[a,b] | GA | 50 | .1944183 | | | | | | |
| | TA11 | 50 | | .2372775 | | | | | |
| | TA12 | 50 | | | .2608414 | | | | |
| | TA21 | 50 | | | | .2882691 | | | |
| | TA22 | 50 | | | | | .3119696 | | |
| | HC | 50 | | | | | | .3329833 | |
| | SA | 50 | | | | | | | .5478063 |
| | Sig. | | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.295E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 61.** Homogeneous Subsets, TSP Size Fifty Performance

For the BPP problem instances of size fifty, the confirmed ANOVA results rejected the performance hypothesis. The homogeneous subsets for this group of problems are shown in Exhibit 62. Once again the Tukey method generated one less subset than the Duncan method, but both placed GELS method two with single stepping in a tie with Hill Climbing (another high finish) as the best performers for this group of problem instances.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | TA12 | 50 | .0809039 | | | | |
| | TA22 | 50 | .1109964 | | | | |
| | SA | 50 | | .1618859 | | | |
| | TA11 | 50 | | .1877836 | | | |
| | GA | 50 | | | .2880904 | | |
| | HC | 50 | | | | .3420107 | |
| | TA21 | 50 | | | | .3442204 | |
| | Sig. | | .332 | .521 | 1.000 | 1.000 | |
| Duncan[a,b] | TA12 | 50 | .0809039 | | | | |
| | TA22 | 50 | | .1109964 | | | |
| | SA | 50 | | | .1618859 | | |
| | TA11 | 50 | | | .1877836 | | |
| | GA | 50 | | | | .2880904 | |
| | HC | 50 | | | | | .3420107 |
| | TA21 | 50 | | | | | .3442204 |
| | Sig. | | 1.000 | 1.000 | .066 | 1.000 | .875 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 4.942E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 62.** Homogeneous Subsets, BPP Size Fifty Performance

For the FAP problem instances of size fifty, the ANOVA found a difference between the algorithm performances. A check of the normality of the residuals failed, however, with only a 0.003 significance on the Kolmogorov-Smirnov test. To provide additional evidence, a Kruskal-Wallis test was performed on the algorithm type factor. This test was able to corroborate the ANOVA, with the same findings of a difference in algorithm performance. Thus, the list of homogeneous subsets could be generated, and is shown in Exhibit 63. Only three subsets, identical in each method, were produced. The best performing algorithms

for this group of problem instances was determined to be a four-way tie, consisting of Simulated Annealing and all of the GELS combinations with the exception of method one with multiple stepping, which was placed in the secondary subset.

| | | | Subset | | |
|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 |
| Tukey HSD[a,b] | GA | 50 | .3408493 | | |
| | HC | 50 | .4022736 | | |
| | TA12 | 50 | | .7182439 | |
| | TA11 | 50 | | | .8237925 |
| | TA21 | 50 | | | .8513245 |
| | TA22 | 50 | | | .8646275 |
| | SA | 50 | | | .8715972 |
| | Sig. | | .516 | 1.000 | .780 |
| Duncan[a,b] | GA | 50 | .3408493 | | |
| | HC | 50 | .4022736 | | |
| | TA12 | 50 | | .7182439 | |
| | TA11 | 50 | | | .8237925 |
| | TA21 | 50 | | | .8513245 |
| | TA22 | 50 | | | .8646275 |
| | SA | 50 | | | .8715972 |
| | Sig. | | .066 | 1.000 | .194 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 2.761E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 63.** Homogeneous Subsets, FAP Size Fifty Performance

Looking at the composite of all problem instances of size fifty, the ANOVA found sufficient cause to reject the performance hypothesis for this set of

experiments. Once more, though, it completely failed the Kolmogorov-Smirnov test of normality of the residuals (with a significance of zero), calling the results into question. A run of a Kruskal-Wallis test against the algorithm type factor was fortunately able to solidly reaffirm the ANOVA results, providing the necessary additional evidence to declare the rejection of the performance hypothesis valid. The list of homogeneous subsets was then generated, shown in Exhibit 64. Once again there is a one-set difference between the Tukey and Duncan methods. The Duncan method declared Simulated Annealing to be the best performing algorithm for this group of problems, with GELS method two with single stepping in second place. Tukey's method, on the other hand, placed both algorithms in the same top slot for performance.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 150 | .2744527 | | | | |
| | TA12 | 150 | | .3533297 | | | |
| | HC | 150 | | .3590892 | | | |
| | TA11 | 150 | | | .4162845 | | |
| | TA22 | 150 | | | .4291978 | | |
| | TA21 | 150 | | | | .4946046 | |
| | SA | 150 | | | | .5270964 | |
| | Sig. | | 1.000 | 1.000 | .977 | .305 | |
| Duncan[a,b] | GA | 150 | .2744527 | | | | |
| | TA12 | 150 | | .3533297 | | | |
| | HC | 150 | | .3590892 | | | |
| | TA11 | 150 | | | .4162845 | | |
| | TA22 | 150 | | | .4291978 | | |
| | TA21 | 150 | | | | .4946046 | |
| | SA | 150 | | | | | .5270964 |
| | Sig. | | 1.000 | .699 | .385 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.658E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 64.** Homogeneous Subsets, Composite Size Fifty Performance

The ANOVA noted an effect of problem type on algorithm performance for this set of experiments also. Exhibit 65 shows a line plot that displays this effect. As with the previous experiment sets, there is a similar shape to the lines for FAP and TSP, suggesting that these two problem types have a similar effect on the algorithms. BPP, however, seems again to have a different effect, as evidenced by its different shape and the fact that the line for BPP crosses the FAP line in three places. In particular, BPP seems to have a much more adverse effect on the performance of Simulated Annealing than the other problems, since the

performance on them for SA goes up, but goes down for BPP. As seen previously, there is also a notably higher overall performance by the algorithms on FAP problem instances than on instances of the other problem types, but also the most overall variability in the quality of the performances between the different algorithms. BPP and TSP problem instances, while not having as high levels of performance as for FAP, nevertheless had notably less variability in the performance ratios between algorithm types.

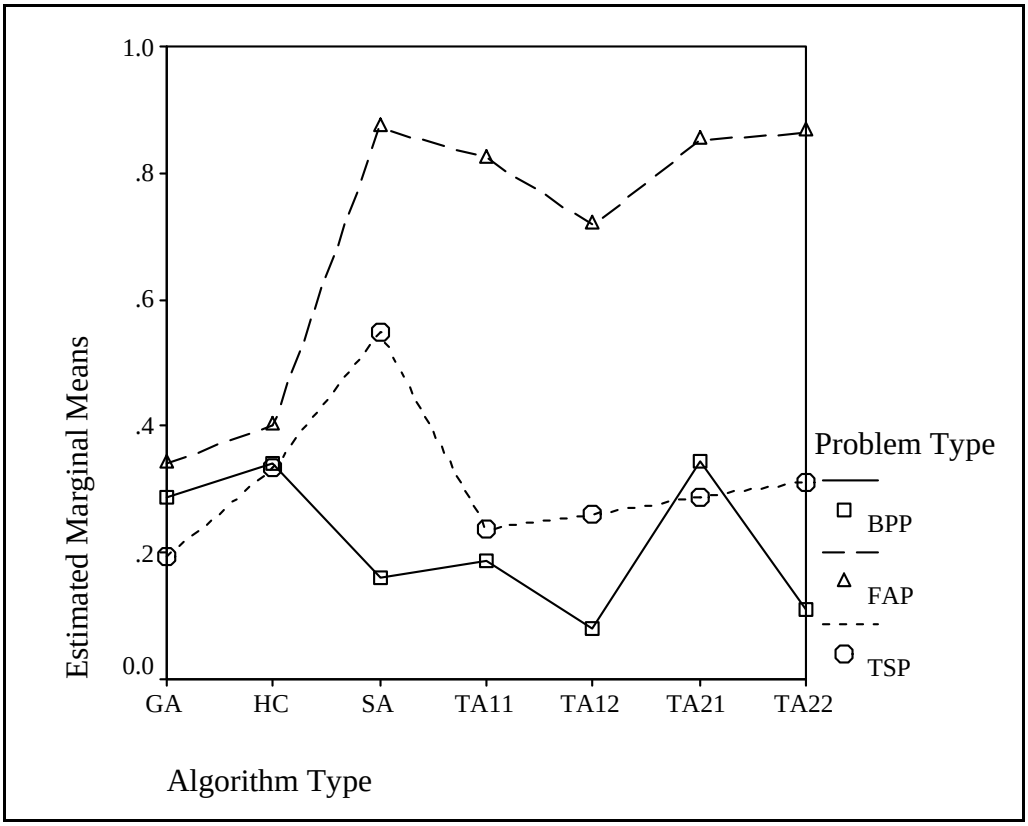


**Exhibit 65.** Problem Type Plot, Composite Size Fifty Performance

#### 2.2.7.1.6 Random Problem Size Performance Results

The final set of experiments to be analyzed for performance was the set consisting of problem instances with randomly assigned problem sizes. For the group of random TSP problem sizes, the ANOVA found significant differences between algorithm performances, thus rejecting the performance hypothesis. This finding was verified through the necessary checks, followed by generation of the homogeneous subsets, which is shown in Exhibit 66. Here the Tukey and Duncan methods generated the same subsets, as well as the same content for each of the subsets. Simulated Annealing was shown to have the best performance for this group of problems, with the highest rated of the GELS combinations being method two with multiple stepping, in second place.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 50 | .3064558 | | | | |
| | TA11 | 50 | .3108164 | | | | |
| | TA12 | 50 | | .3500400 | | | |
| | HC | 50 | | .3564982 | | | |
| | TA21 | 50 | | | .4100139 | | |
| | TA22 | 50 | | | | .4398797 | |
| | SA | 50 | | | | | .5728985 |
| | Sig. | | .998 | .983 | 1.000 | 1.000 | 1.000 |
| Duncan[a,b] | GA | 50 | .3064558 | | | | |
| | TA11 | 50 | .3108164 | | | | |
| | TA12 | 50 | | .3500400 | | | |
| | HC | 50 | | .3564982 | | | |
| | TA21 | 50 | | | .4100139 | | |
| | TA22 | 50 | | | | .4398797 | |
| | SA | 50 | | | | | .5728985 |
| | Sig. | | .581 | .413 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.554E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 66.** Homogeneous Subsets, Random Size TSP Performance

For the random size BPP problem instances the ANOVA again rejected the performance hypothesis, finding significant differences among the algorithm performances. The necessary checks revealed no problems, and the homogeneous subsets were generated. These results are shown in Exhibit 67. Once again the Tukey method produced one less subset than the Duncan method, which named GELS method two with single stepping and the Genetic Algorithm to the highest performing subset. The Tukey method also placed those two algorithms in the top subset, adding in Hill Climbing for a three-way tie.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | TA12 | 50 | .1446418 | | | | |
| | TA22 | 50 | .1890416 | .1890416 | | | |
| | TA11 | 50 | | .2410511 | .2410511 | | |
| | SA | 50 | | | .2551412 | | |
| | HC | 50 | | | | .3348845 | |
| | GA | 50 | | | | .3583587 | |
| | TA21 | 50 | | | | .3907435 | |
| | Sig. | | .236 | .096 | .990 | .057 | |
| Duncan[a,b] | TA12 | 50 | .1446418 | | | | |
| | TA22 | 50 | | .1890416 | | | |
| | TA11 | 50 | | | .2410511 | | |
| | SA | 50 | | | .2551412 | | |
| | HC | 50 | | | | .3348845 | |
| | GA | 50 | | | | .3583587 | .3583587 |
| | TA21 | 50 | | | | | .3907435 |
| | Sig. | | 1.000 | 1.000 | .461 | .220 | .091 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 9.116E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 67.** Homogeneous Subsets, Random Size BPP Performance

For the FAP random sized problem instances the ANOVA once again found a significant difference between the performances of the algorithms, leading to a rejection of the performance hypothesis for this group of problem instances. These results were affirmed by the necessary checks, and the homogeneous subsets were generated, as shown in Exhibit 68. Again, the Tukey method produced one fewer subset than the Duncan method. Duncan's method had a two-way tie for the top performance between Simulated Annealing, while Tukey's method had a four-way tie between the two algorithms just mentioned and the rest of the GELS

combinations, with the exception of method one with multiple stepping (which was placed in the next lower subset).

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | GA | 50 | .2973995 | | | |
| | HC | 50 | .3389672 | | | |
| | TA12 | 50 | | .5825441 | | |
| | TA11 | 50 | | .6112350 | .6112350 | |
| | TA22 | 50 | | .6245444 | .6245444 | |
| | TA21 | 50 | | .6526007 | .6526007 | |
| | SA | 50 | | | .6761866 | |
| | Sig. | | .632 | .074 | .124 | |
| Duncan[a,b] | GA | 50 | .2973995 | | | |
| | HC | 50 | .3389672 | | | |
| | TA12 | 50 | | .5825441 | | |
| | TA11 | 50 | | .6112350 | .6112350 | |
| | TA22 | 50 | | .6245444 | .6245444 | |
| | TA21 | 50 | | | .6526007 | .6526007 |
| | SA | 50 | | | | .6761866 |
| | Sig. | | .095 | .111 | .116 | .342 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.535E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 68.** Homogeneous Subsets, Random Size FAP Performance

The final portion of the performance analysis was for the composite of the random sized problem instances. The ANOVA for this set of experiments found a significant difference between the performances of the algorithms, but in conducting the necessary checks, it was found to fail the test for normally

distributed residuals, scoring only a significance value of 0.01 on the Kolmogorov-Smirnov test. Additional evidence was required, and a Kruskal-Wallis test of the algorithm type factor showed the same significant difference, thus agreeing with and affirming the ANOVA conclusion to reject the performance hypothesis for this set of experiments. Exhibit 69 shows the homogeneous subsets that were generated. Yet again, the Tukey method produced one less subset than the Duncan method. However, both methods had the same content for their respective top subsets, naming Simulated Annealing and GELS method two with single stepping as the co-best performers for the set of random sized experiments.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 150 | .3207380 | | | | |
| | HC | 150 | .3434500 | | | | |
| | TA12 | 150 | .3590753 | .3590753 | | | |
| | TA11 | 150 | | .3877008 | .3877008 | | |
| | TA22 | 150 | | | .4178219 | | |
| | TA21 | 150 | | | | .4844527 | |
| | SA | 150 | | | | .5014088 | |
| | Sig. | | .135 | .466 | .401 | .916 | |
| Duncan[a,b] | GA | 150 | .3207380 | | | | |
| | HC | 150 | .3434500 | .3434500 | | | |
| | TA12 | 150 | | .3590753 | .3590753 | | |
| | TA11 | 150 | | | .3877008 | | |
| | TA22 | 150 | | | | .4178219 | |
| | TA21 | 150 | | | | | .4844527 |
| | SA | 150 | | | | | .5014088 |
| | Sig. | | .128 | .294 | .055 | 1.000 | .255 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.663E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 69.** Homogeneous Subsets, Random Size Composite Performance

Once again, the ANOVA for this set of experiments also found a significant effect of problem type on the performance of the algorithms. This effect is illustrated in Exhibit 70. Again, there is somewhat of a similarity between the shape of the lines for FAP and TSP problem instances, but a notable difference in the shape of the BPP line. At several points the line for BPP is decreasing from one point to the next while the lines for the others are increasing, and again BPP seemed to have a rather severe adverse effect on Simulated Annealing. Performance ratios are again somewhat higher in general for FAP problem

instances and somewhat lower in general for BPP problem instances, although lower ratios for BPP are not altogether surprising since there is usually less of an opportunity to improve upon the objective function values of random solutions given the typically smaller range of these values within problem instances of any size in comparison to instances of the other problem types. Still, though, the BPP problem instances seem to be having an adverse effect on some of the algorithms (Simulated Annealing in particular), but not others (the Genetic Algorithm and GELS method two with single stepping, for instance).

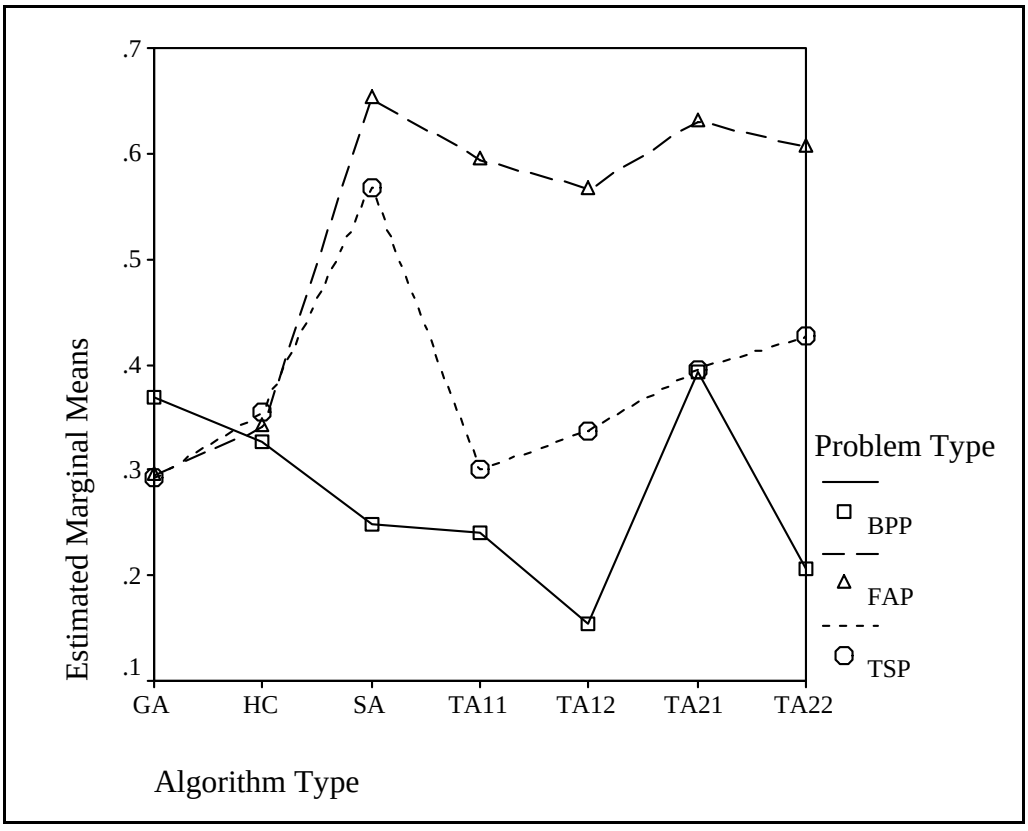


**Exhibit 70.** Problem Type Plot, Random Size Composite Performance

Due to the fact that this set of experiments contained both problem type and problem size as factors, this enabled the ANOVA to include both of them in its analysis and look for interactions. Consequently, the ANOVA also found evidence of a significant effect of problem size on the algorithms' ability to produce high quality solutions to them. This effect is displayed in Exhibit 71. The interaction between the two factors can be seen in the crossing lines. However, while in plots of performance for the different problem types there were notably different shapes to some of the lines (particularly for BPP), here all the lines have formed more or less the same shape. Also, the difference between the highest point and the lowest point for any of the algorithms is comparable to the distance for any of the others. This indicates that the problem size is having a fairly consistent effect on each of the algorithms.

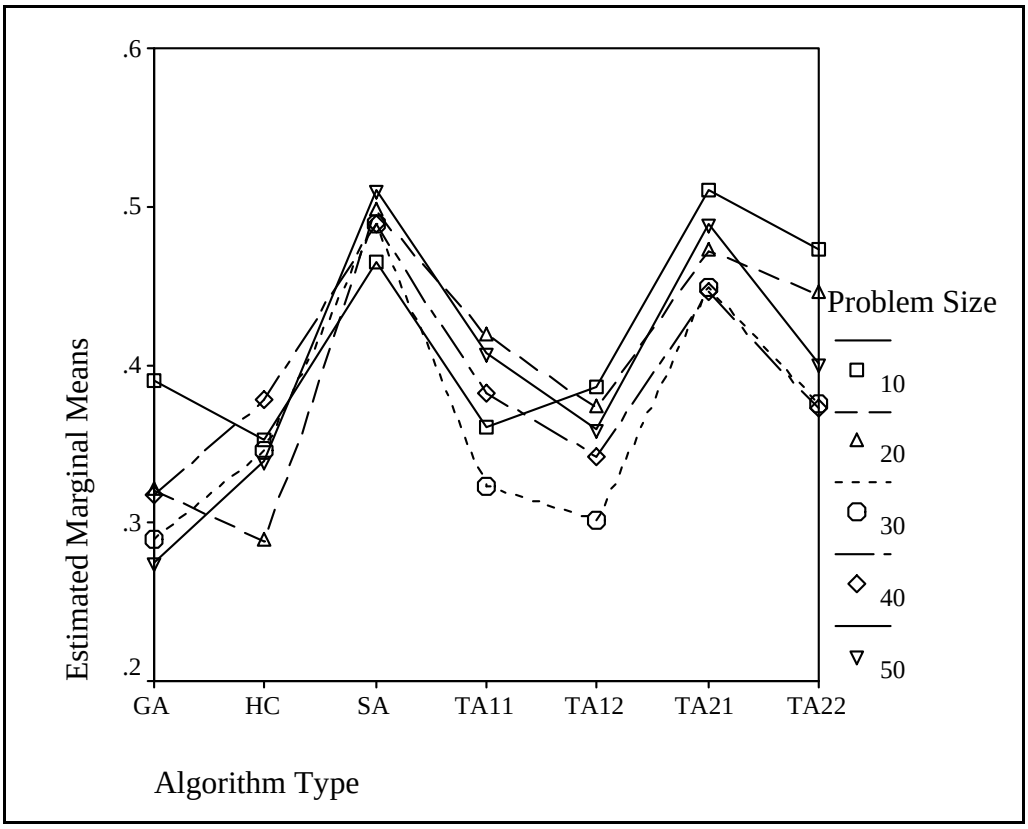


**Exhibit 71.** Problem Size Plot, Random Size Composite Performance

## 2.2.7.2 Algorithm Efficiency Results

Recalling that there were two hypotheses in the original DOE, the second phase of the analysis process was to investigate the efficiency of the algorithms. That is, now that an analysis has been completed of how good the solutions produced by the algorithms were, the second phase was to investigate how quickly the algorithms arrived at those solutions. This is of interest because in situations such as in these experiments where suboptimal solutions are being delivered, an algorithm that can produce solutions better than those of another algorithm is more

valuable. However, if the solutions produced by that algorithm are only slightly better than those of the other, and it took much longer for that algorithm to produce the solution than it took for the other, the value of that algorithm is reduced.

The sets of experiments to investigate algorithm efficiency mirrored those conducted to test algorithm performance. A total of six sets of experiments were conducted, consisting of a set each for problem instance sizes of ten, twenty, thirty, forty, and fifty, plus a set of random size problem instances, just as was done for the algorithm performance analysis phase. Exactly the same problem instances that were used to analyze performance were used to analyze efficiency, and exactly the same set of tests, tools, and procedures were used in the analysis process, so as with the last five sets of performance data only the summary results of the efficiency investigations will be shown.

The only difference between the algorithm performance analysis and the algorithm efficiency analysis is the metric under study. In the performance phase this metric measured an algorithm's ability to improve upon random solutions. In the efficiency phase, the metric measured how many solutions were examined by an algorithm before arriving at its ultimate solution. In this way, the qualities of the solutions being produced by an algorithm were still being taken into consideration, but they were being tempered by the portion of the search space that was being covered in producing those solutions.

#### 2.2.7.2.1 Problem Size Ten Efficiency Results

Beginning with the set of problem instances of size ten, the ANOVA showed that there was a significant difference in algorithm performance, enough to reject the efficiency hypothesis for this group of problem instances. However, it failed the Kolmogorov-Smirnov test for normal distribution of the residuals. As will be seen with the other problem groups and experiment sets, this situation was present for every problem group of every experiment set in the efficiency analysis.

This situation is due in large part to the nature of the algorithms. Hill Climbing is a purely greedy algorithm, and as such it tends to terminate very quickly. Thus, any gains in the quality of solutions produced by the algorithm are obtained at a cost of very few solutions examined, giving it a very high efficiency ratio. But, the Genetic Algorithm is by design iterative, repeating the same procedure over and over until a defined iteration count has been reached. Any gains in quality of solution are usually obtained only by examining and evaluating large numbers of solution candidates, meaning its efficiency ratio will tend to be rather small.

So, the efficiency ratio became a double-edged sword. On the one hand it was a convenient and uniform way of evaluating the efficiency of algorithms across problem type and size, and for comparing those efficiencies. On the other hand, because of the way the algorithms operate the ratios for Hill Climbing were very high, and those of the remaining algorithms substantially lower, resulting in

experiment case residuals that stood a very low chance of being normally distributed.

This problem could have been alleviated somewhat by excluding Hill Climbing from the analysis, yet this would have meant that there would have been no means of comparison between it and the other algorithms other than by rough estimates. Instead, the HC data were included in the analysis, and Kruskal-Wallis tests were employed to ensure that the ANOVA results were accurate in spite of non-normal residuals. With that said, the Kruskal-Wallis test of the algorithms' efficiency for the TSP size ten problems was conducted, and matched the results of the ANOVA, confirming the decision to reject the efficiency hypothesis.

The ANOVA also discovered, for the first time, that the run number did not play a significant role in the results, as evidenced by its significance value of 0.195 (above the threshold of 0.05). This result would turn out to also be commonplace during the efficiency analyses, not occurring for every problem group in every experiment set but in a large percentage of them. This too can be explained by the nature of the experiments. In the performance phase, problem instances each began with a Monte Carlo solution and the algorithms attempted to improve on it. The quality of the Monte Carlo solution was random, so the opportunity to improve on it varied from problem instance to problem instance. This innate variability affected the outcome, so the individual experiment runs had to be accounted for and blocked.

For the efficiency phase, however, this effect was not as potent. It was still present in that gains in quality were still part of the metric, but it was dampened by being only part of the ratio, a part that was usually much smaller than the number of solutions examined. This dampening often meant that the effect was reduced sufficiently that it was no longer considered significant by the ANOVA. Nevertheless, it did not hurt to keep the factor blocked in the experiments. In those situations where it still played a significant effect, it was still necessary for it to be blocked. In the situations where it was not significant, removing it as a blocking factor would have taken the variability in the experiments assigned to that factor and put it back into the general "pool" to be allocated elsewhere. Since there was already more than a sufficient amount of data to be able to give the ANOVA room to make its determinations, adding this additional material into the mix was not needed.

Returning to the analysis of the TSP problem instances of size ten, once the ANOVA results were confirmed by the Kruskal-Wallis test, the homogeneous subsets for efficiency were generated, as shown in Exhibit 72. As expected, and as will become the norm for these analyses, Hill Climbing finished as the most efficient of the algorithms by a large margin. Of more interest here is the second place finisher. Duncan's method determined this to be GELS method one with single stepping, while Tukey's method grouped that algorithm with GELS method one with multiple stepping.

| | Algorithm Type | N | Subset 1 | Subset 2 | Subset 3 |
|---|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 50 | .0001206 | | |
| | SA | 50 | .0005898 | | |
| | TA22 | 50 | .0007808 | | |
| | TA21 | 50 | .0011020 | | |
| | TA12 | 50 | .0152340 | .0152340 | |
| | TA11 | 50 | | .0414254 | |
| | HC | 50 | | | .4089060 |
| | Sig. | | .887 | .348 | 1.000 |
| Duncan[a,b] | GA | 50 | .0001206 | | |
| | SA | 50 | .0005898 | | |
| | TA22 | 50 | .0007808 | | |
| | TA21 | 50 | .0011020 | | |
| | TA12 | 50 | .0152340 | | |
| | TA11 | 50 | | .0414254 | |
| | HC | 50 | | | .4089060 |
| | Sig. | | .286 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 3.846E-03.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 72.** Homogeneous Subsets, TSP Size Ten Efficiency

For the BPP problem instances, the ANOVA determined that there was a significant difference between the efficiencies (as will always be the case because of Hill Climbing), and this was backed up by the Kruskal-Wallis test. The homogeneous subsets are shown in Exhibit 73. Tukey's method produced only two subsets, while Duncan's produced three. Again, Hill Climbing claimed the top spot in both methods. Since this is uniformly the case, it will hereafter go unmentioned,

concentrating instead on the next algorithm(s) in the list. The diagrams can be used as references for the degree of difference between the Hill Climbing efficiencies and the others. In this case, Tukey's method placed all the algorithms except Hill Climbing into the second subset, while Duncan's method managed to detect enough of a difference to award GELS method two with single stepping the second spot.

| | | | Subset | | |
|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 |
| Tukey HSD[a,b] | TA22 | 50 | .0000124 | | |
| | GA | 50 | .0000169 | | |
| | TA12 | 50 | .0000229 | | |
| | TA11 | 50 | .0001250 | | |
| | SA | 50 | .0001293 | | |
| | TA21 | 50 | .0027439 | | |
| | HC | 50 | | .0707447 | |
| | Sig. | | .129 | 1.000 | |
| Duncan[a,b] | TA22 | 50 | .0000124 | | |
| | GA | 50 | .0000169 | | |
| | TA12 | 50 | .0000229 | | |
| | TA11 | 50 | .0001250 | | |
| | SA | 50 | .0001293 | | |
| | TA21 | 50 | | .0027439 | |
| | HC | 50 | | | .0707447 |
| | Sig. | | .923 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 2.755E-05.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 73.** Homogeneous Subsets, BPP Size Ten Efficiency

For the FAP problem instances, the ANOVA was conducted and corroborated, and the homogeneous subsets were generated as shown in Exhibit 74. Again the Tukey method generated one less subset than the Duncan method, and has assigned the second best efficiency to GELS method one with single stepping and GELS method two with single stepping together. Duncan's method assigned each of the two to its own subset, with the former in second place and the latter in third.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | GA | 50 | .0000357 | | | |
| | TA22 | 50 | .0000806 | | | |
| | SA | 50 | .0002585 | | | |
| | TA12 | 50 | .0033077 | | | |
| | TA21 | 50 | | .1162469 | | |
| | TA11 | 50 | | .1935029 | | |
| | HC | 50 | | | .9620087 | |
| | Sig. | | 1.000 | .281 | 1.000 | |
| Duncan[a,b] | GA | 50 | .0000357 | | | |
| | TA22 | 50 | .0000806 | | | |
| | SA | 50 | .0002585 | | | |
| | TA12 | 50 | .0033077 | | | |
| | TA21 | 50 | | .1162469 | | |
| | TA11 | 50 | | | .1935029 | |
| | HC | 50 | | | | .9620087 |
| | Sig. | | .933 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 2.994E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 74.** Homogeneous Subsets, FAP Size Ten Efficiency

In considering the problem instances of size ten as a whole, the ANOVA was conducted and corroborated. The homogeneous subsets generated are shown in Exhibit 75. Both methods put GELS method one with single stepping into the second place subset.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | GA | 150 | .0000577 | | | |
| | TA22 | 150 | .0002913 | | | |
| | SA | 150 | .0003259 | | | |
| | TA12 | 150 | .0061882 | .0061882 | | |
| | TA21 | 150 | | .0400309 | | |
| | TA11 | 150 | | | .0783511 | |
| | HC | 150 | | | | .4805531 |
| | Sig. | | .999 | .103 | 1.000 | 1.000 |
| Duncan[a,b] | GA | 150 | .0000577 | | | |
| | TA22 | 150 | .0002913 | | | |
| | SA | 150 | .0003259 | | | |
| | TA12 | 150 | .0061882 | | | |
| | TA21 | 150 | | .0400309 | | |
| | TA11 | 150 | | | .0783511 | |
| | HC | 150 | | | | .4805531 |
| | Sig. | | .664 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.192E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 75.** Homogeneous Subsets, Composite Size Ten Efficiency

In determining the effect of problem type for this set of experiments, the ANOVA found a significant interaction between problem type and algorithm

efficiency, as displayed in Exhibit 76. It shows the expected high values for Hill Climbing, and it also shows a consistent shape for the lines, indicating that problem type had a relatively consistent effect on each of the algorithms. It is also apparent that the best efficiencies overall were attained against FAP problem instances.

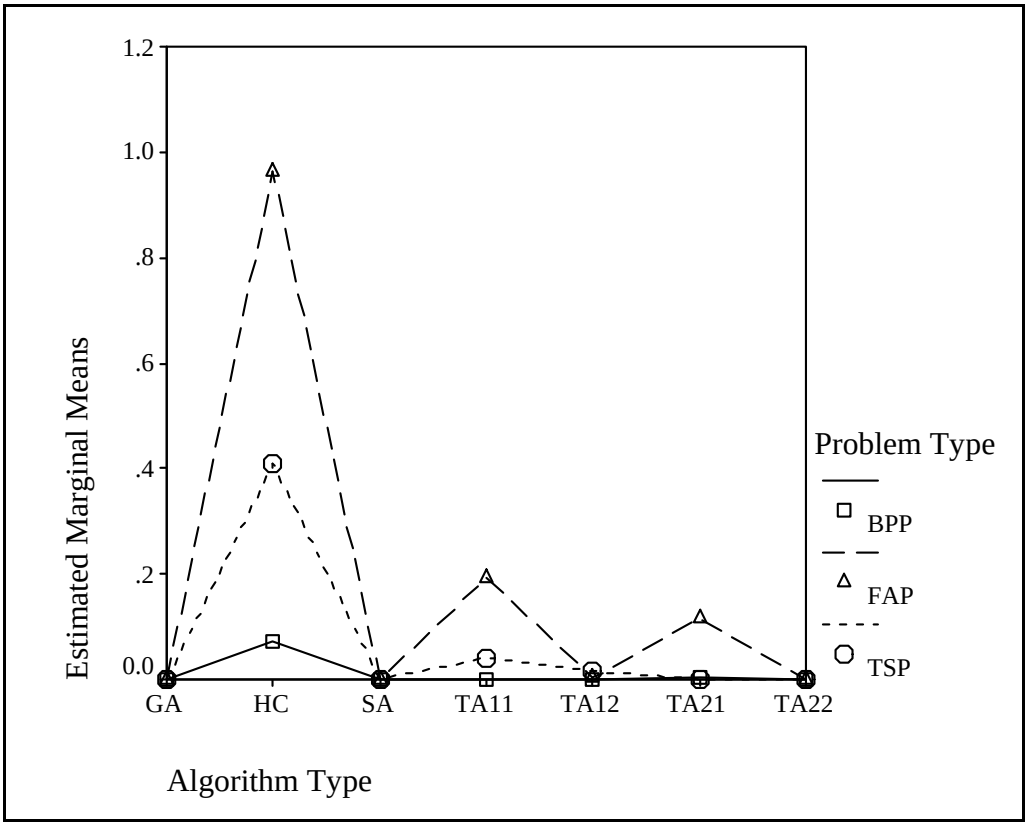


**Exhibit 76.** Problem Type Plot, Composite Size Ten Efficiency

## 2.2.7.2.2 Problem Size Twenty Efficiency Results

For the TSP problem instances of size twenty, the ANOVA was conducted and corroborated, and the homogeneous subsets generated as shown in Exhibit 77. The

Tukey method again produced one less subset than the Duncan method, and placed GELS method one with single stepping and GELS method one with multiple stepping in the second place subset together. The Duncan method placed GELS method one with single stepping in its own subset in the second spot, with GELS method one with multiple stepping also in its own subset in the third spot.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | TA22 | 50 | .0001691 | | | |
| | GA | 50 | .0001755 | | | |
| | TA21 | 50 | .0002921 | | | |
| | SA | 50 | .0011623 | | | |
| | TA12 | 50 | .0114639 | .0114639 | | |
| | TA11 | 50 | | .0234045 | | |
| | HC | 50 | | | .2309011 | |
| | Sig. | | .185 | .136 | 1.000 | |
| Duncan[a,b] | TA22 | 50 | .0001691 | | | |
| | GA | 50 | .0001755 | | | |
| | TA21 | 50 | .0002921 | | | |
| | SA | 50 | .0011623 | | | |
| | TA12 | 50 | | .0114639 | | |
| | TA11 | 50 | | | .0234045 | |
| | HC | 50 | | | | .2309011 |
| | Sig. | | .848 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 5.349E-04.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 77.** Homogeneous Subsets, TSP Size Twenty Efficiency

For the BPP problem instances, the ANOVA was conducted and corroborated, and the homogeneous subsets were generated as shown in Exhibit 78. Here the two methods are in complete agreement, placing GELS method two with single stepping alone in the second spot, with all the rest aside from Hill Climbing in the third subset together.

| | | | Subset | | |
|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 |
| Tukey HSD[a,b] | TA22 | 50 | .0000105 | | |
| | TA12 | 50 | .0000243 | | |
| | GA | 50 | .0000275 | | |
| | TA11 | 50 | .0001950 | | |
| | SA | 50 | .0002410 | | |
| | TA21 | 50 | | .0050334 | |
| | HC | 50 | | | .0404029 |
| | Sig. | | .993 | 1.000 | 1.000 |
| Duncan[a,b] | TA22 | 50 | .0000105 | | |
| | TA12 | 50 | .0000243 | | |
| | GA | 50 | .0000275 | | |
| | TA11 | 50 | .0001950 | | |
| | SA | 50 | .0002410 | | |
| | TA21 | 50 | | .0050334 | |
| | HC | 50 | | | .0404029 |
| | Sig. | | .544 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 2.720E-06.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 78.** Homogeneous Subsets, BPP Size Twenty Efficiency

For the FAP problem instances, the ANOVA was again conducted and corroborated, with the homogeneous subsets coming out as shown in Exhibit 79. The Tukey and Duncan methods agreed on the number of subsets to be produced, but disagreed on their contents. Duncan's method put GELS method two with single stepping in the second subset along with GELS method one with single stepping, while Tukey's method added GELS method two with multiple stepping to that grouping.

| | Algorithm Type | N | Subset 1 | Subset 2 | Subset 3 |
|---|---|---|---|---|---|
| Tukey HSD[a,b] | GA | 50 | .0000443 | | |
| | SA | 50 | .0003463 | | |
| | TA12 | 50 | .0265885 | | |
| | TA22 | 50 | .1098495 | .1098495 | |
| | TA11 | 50 | | .3089587 | |
| | TA21 | 50 | | .3134450 | |
| | HC | 50 | | | 1.2401905 |
| | Sig. | | .880 | .252 | 1.000 |
| Duncan[a,b] | GA | 50 | .0000443 | | |
| | SA | 50 | .0003463 | | |
| | TA12 | 50 | .0265885 | | |
| | TA22 | 50 | .1098495 | | |
| | TA11 | 50 | | .3089587 | |
| | TA21 | 50 | | .3134450 | |
| | HC | 50 | | | 1.2401905 |
| | Sig. | | .266 | .960 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = .198.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 79.** Homogeneous Subsets, FAP Size Twenty Efficiency

In considering the problem instances of size twenty together as a whole, once again the ANOVA was conducted and corroborated, leading to the homogeneous subsets shown in Exhibit 80. The Tukey and Duncan methods again agreed on number of subsets, but disagreed on their contents. Duncan's method declared GELS method one with single stepping and GELS method two with single stepping to be in a tie for the second best efficiency for this set of experiments, while

Tukey's method added GELS method two with multiple stepping to this grouping for a three-way tie.

| | | | Subset | | |
|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 |
| Tukey HSD[a,b] | GA | 150 | .0000824 | | |
| | SA | 150 | .0005832 | | |
| | TA12 | 150 | .0126922 | | |
| | TA22 | 150 | .0366764 | .0366764 | |
| | TA21 | 150 | | .1062568 | |
| | TA11 | 150 | | .1108527 | |
| | HC | 150 | | | .5038315 |
| | Sig. | | .899 | .197 | 1.000 |
| Duncan[a,b] | GA | 150 | .0000824 | | |
| | SA | 150 | .0005832 | | |
| | TA12 | 150 | .0126922 | | |
| | TA22 | 150 | .0366764 | | |
| | TA21 | 150 | | .1062568 | |
| | TA11 | 150 | | .1108527 | |
| | HC | 150 | | | .5038315 |
| | Sig. | | .285 | .882 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 7.128E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 80.** Homogeneous Subsets, Composite Size Twenty Efficiency

The ANOVA for this experiment set also found a significant interaction between problem type and algorithm efficiency. This is visualized in Exhibit 81. Again there is a similar shape to the lines, indicating a comparable effect of

problem type on each of the algorithms. Also, the FAP problem instances again produced notably higher efficiency ratios from the algorithms than the other two problem types.

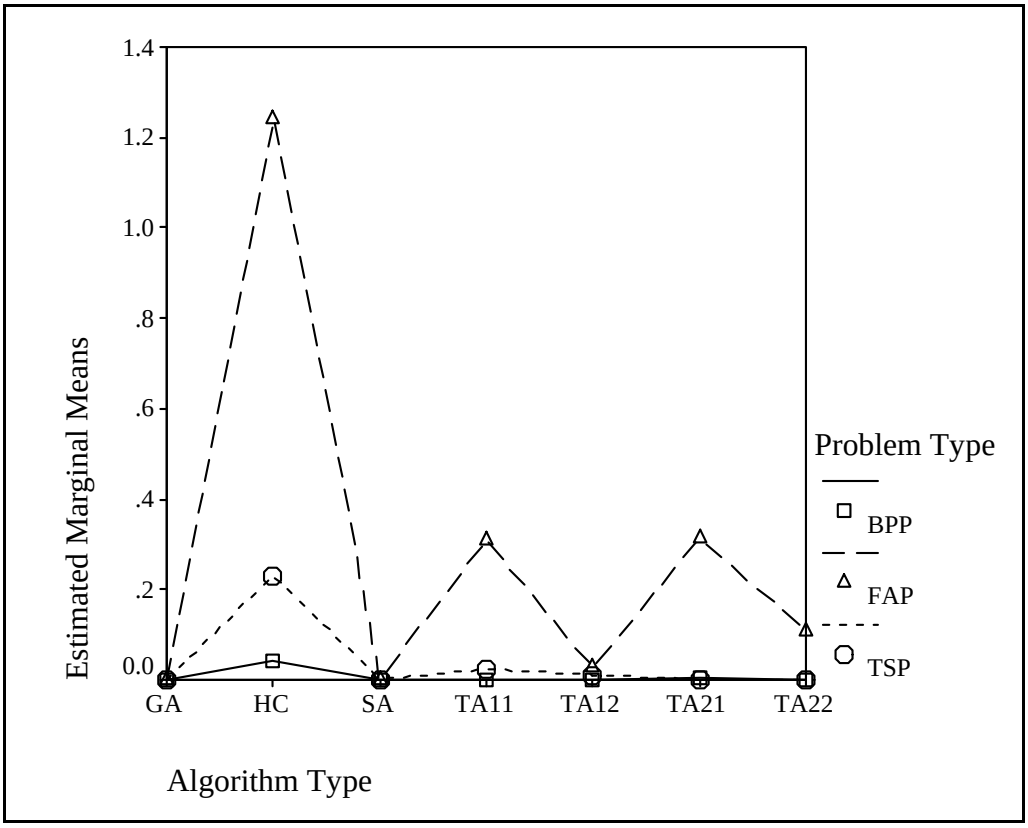


**Exhibit 81.** Problem Type Plot, Composite Size Twenty Efficiency

## 2.2.7.2.3 Problem Size Thirty Efficiency Results

Moving on to the analysis of the problem instances of size thirty, the validated ANOVA results for the TSP problem instances of this size led to the homogeneous subsets shown in Exhibit 82. Once again the Tukey and Duncan methods agreed

on count of subsets but not quite on content, with Tukey's method twice placing the same algorithm in two different subsets. Both methods agreed, though, that the second best efficiency for this group of problem instances should go to GELS method one with single stepping. Duncan's method put this algorithm alone in second place, while Tukey's method grouped it into a tie with GELS method one with multiple stepping.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | TA22 | 50 | .0001603 | | | |
| | TA21 | 50 | .0001907 | | | |
| | GA | 50 | .0002107 | | | |
| | SA | 50 | .0016618 | .0016618 | | |
| | TA12 | 50 | | .0087461 | .0087461 | |
| | TA11 | 50 | | | .0149772 | |
| | HC | 50 | | | | .1603784 |
| | Sig. | | .996 | .054 | .134 | 1.000 |
| Duncan[a,b] | TA22 | 50 | .0001603 | | | |
| | TA21 | 50 | .0001907 | | | |
| | GA | 50 | .0002107 | | | |
| | SA | 50 | .0016618 | | | |
| | TA12 | 50 | | .0087461 | | |
| | TA11 | 50 | | | .0149772 | |
| | HC | 50 | | | | .1603784 |
| | Sig. | | .578 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.452E-04.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 82.** Homogeneous Subsets, TSP Size Thirty Efficiency

For the BPP problem instances, the validated ANOVA led to the homogeneous subsets shown in Exhibit 83. This time the Tukey method generated three subsets as opposed to four by the Duncan method, but both agreed that the second best efficiency for this group of problem instances belonged to GELS method two with single stepping.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | TA22 | 50 | .0000086 | | | |
| | TA12 | 50 | .0000241 | | | |
| | GA | 50 | .0000368 | | | |
| | TA11 | 50 | .0002490 | | | |
| | SA | 50 | .0003573 | | | |
| | TA21 | 50 | | .0039249 | | |
| | HC | 50 | | | .0288668 | |
| | Sig. | | .183 | 1.000 | 1.000 | |
| Duncan[a,b] | TA22 | 50 | .0000086 | | | |
| | TA12 | 50 | .0000241 | | | |
| | GA | 50 | .0000368 | | | |
| | TA11 | 50 | .0002490 | .0002490 | | |
| | SA | 50 | | .0003573 | | |
| | TA21 | 50 | | | .0039249 | |
| | HC | 50 | | | | .0288668 |
| | Sig. | | .126 | .448 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 5.080E-07.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 83.** Homogeneous Subsets, BPP Size Thirty Efficiency

For the FAP problem instances, the validated ANOVA led to the homogeneous subsets shown in Exhibit 84. In this case, both the Tukey and the Duncan method could not distinguish between the efficiencies of any of the algorithms other than Hill Climbing, so all finished in a tie for second best.

| | | | Subset | |
|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 |
| Tukey HSD[a,b] | GA | 50 | .0000222 | |
| | TA12 | 50 | .0000683 | |
| | SA | 50 | .0001443 | |
| | TA22 | 50 | .0072575 | |
| | TA11 | 50 | .0283337 | |
| | TA21 | 50 | .0779681 | |
| | HC | 50 | | .6522771 |
| | Sig. | | .592 | 1.000 |
| Duncan[a,b] | GA | 50 | .0000222 | |
| | TA12 | 50 | .0000683 | |
| | SA | 50 | .0001443 | |
| | TA22 | 50 | .0072575 | |
| | TA11 | 50 | .0283337 | |
| | TA21 | 50 | .0779681 | |
| | HC | 50 | | .6522771 |
| | Sig. | | .133 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 5.038E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 84.** Homogeneous Subsets, FAP Size Thirty Efficiency

In considering all problem instances of size thirty together, the validated ANOVA led to the generation of the homogeneous subsets shown in Exhibit 85. Like the FAP problem instances, neither the Tukey nor the Duncan method could make any distinction between the efficiencies of any of the algorithms other than Hill Climbing. Consequently, for the size thirty experiment set all algorithms besides Hill Climbing finished with the second best efficiency.

| | | | Subset | |
|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 |
| Tukey HSD[a,b] | GA | 150 | .0000899 | |
| | SA | 150 | .0007211 | |
| | TA22 | 150 | .0024755 | |
| | TA12 | 150 | .0029462 | |
| | TA11 | 150 | .0145200 | |
| | TA21 | 150 | .0273612 | |
| | HC | 150 | | .2805074 |
| | Sig. | | .554 | 1.000 |
| Duncan[a,b] | GA | 150 | .0000899 | |
| | SA | 150 | .0007211 | |
| | TA22 | 150 | .0024755 | |
| | TA12 | 150 | .0029462 | |
| | TA11 | 150 | .0145200 | |
| | TA21 | 150 | .0273612 | |
| | HC | 150 | | .2805074 |
| | Sig. | | .120 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.739E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 85.** Homogeneous Subsets, Composite Size Thirty Efficiency

The validated ANOVA for this experiment set once again found a significant interaction between problem type and algorithm efficiency, as shown in Exhibit 86. Once again there is a consistent shape to the lines indicative of a similar effect of problem type on each of the algorithms, and the FAP problem instances in general garnered the highest efficiencies from the algorithms.

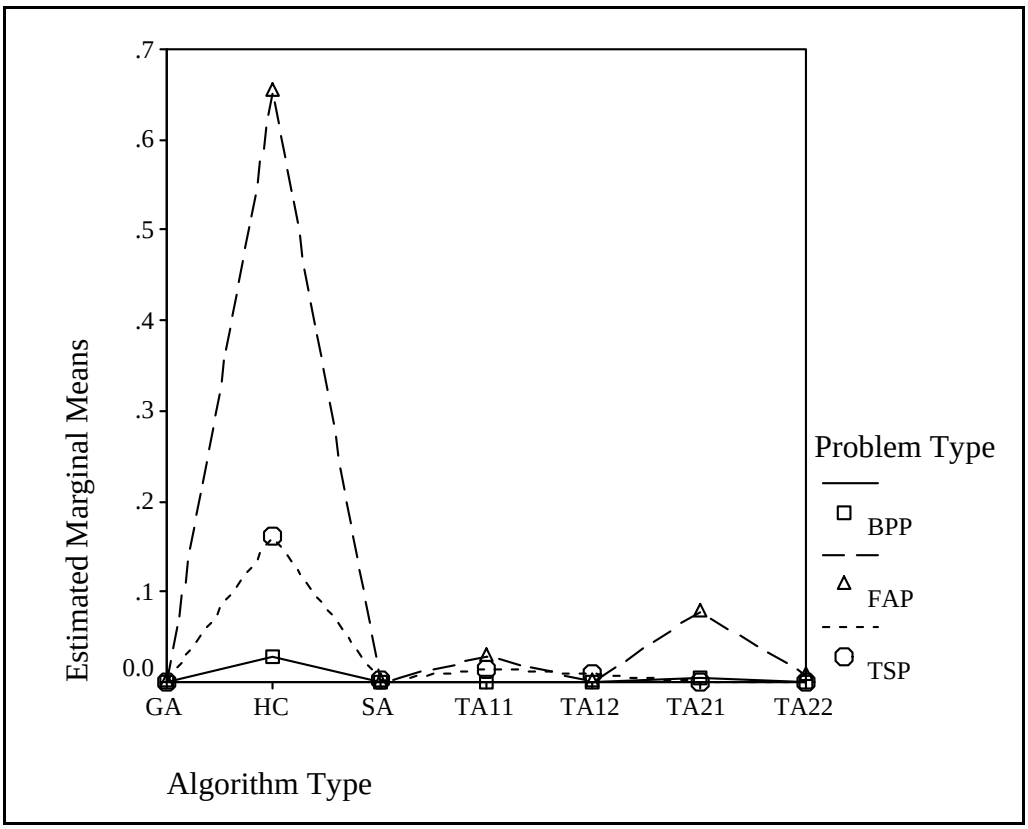


**Exhibit 86.** Problem Type Plot, Composite Size Thirty Efficiency

## 2.2.7.2.4 Problem Size Forty Efficiency Results

Moving along to the examination of the set of experiments of size forty, the validated ANOVA for the TSP problem instances of this size led to the generation of the homogeneous subsets shown in Exhibit 87. Both the Tukey and Duncan methods were in complete agreement in this case, putting GELS method one with single stepping into the slot for the second best efficiency.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | TA22 | 50 | .0001660 | | | |
| | TA21 | 50 | .0001850 | | | |
| | GA | 50 | .0002784 | | | |
| | SA | 50 | .0022785 | | | |
| | TA12 | 50 | | .0078855 | | |
| | TA11 | 50 | | | .0129658 | |
| | HC | 50 | | | | .1236567 |
| | Sig. | | .791 | 1.000 | 1.000 | 1.000 |
| Duncan[a,b] | TA22 | 50 | .0001660 | | | |
| | TA21 | 50 | .0001850 | | | |
| | GA | 50 | .0002784 | | | |
| | SA | 50 | .0022785 | | | |
| | TA12 | 50 | | .0078855 | | |
| | TA11 | 50 | | | .0129658 | |
| | HC | 50 | | | | .1236567 |
| | Sig. | | .200 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 5.531E-05.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 87.** Homogeneous Subsets, TSP Size Forty Efficiency

For the BPP problem instances, the validated ANOVA led to the homogenous subsets shown in Exhibit 88. The Tukey and Duncan methods have agreed on the number of subsets, but Tukey's method placed three of the algorithms in two subsets. Both methods, however, agreed that the second best efficiency for this group of problems was GELS method two with single stepping.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | TA22 | 50 | .0000084 | | | |
| | TA12 | 50 | .0000274 | .0000274 | | |
| | GA | 50 | .0000440 | .0000440 | | |
| | TA11 | 50 | .0003240 | .0003240 | | |
| | SA | 50 | | .0003562 | | |
| | TA21 | 50 | | | .0032085 | |
| | HC | 50 | | | | .0225612 |
| | Sig. | | .075 | .054 | 1.000 | 1.000 |
| Duncan[a,b] | TA22 | 50 | .0000084 | | | |
| | TA12 | 50 | .0000274 | | | |
| | GA | 50 | .0000440 | | | |
| | TA11 | 50 | | .0003240 | | |
| | SA | 50 | | .0003562 | | |
| | TA21 | 50 | | | .0032085 | |
| | HC | 50 | | | | .0225612 |
| | Sig. | | .767 | .773 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 3.128E-07.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 88.** Homogeneous Subsets, BPP Size Forty Efficiency

For the FAP problem instances, the validated ANOVA led to the generation of the homogeneous subsets shown in Exhibit 89. As with the FAP problem instances for the size thirty experiment set, both Tukey's method and Duncan's method were unable to make a distinction between the efficiencies of any of the algorithms besides Hill Climbing. Therefore, all of them fell into the second place position for this group of problems.

| | | | Subset | |
|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 |
| Tukey HSD[a,b] | GA | 50 | .0000194 | |
| | TA12 | 50 | .0000690 | |
| | SA | 50 | .0001212 | |
| | TA22 | 50 | .0051760 | |
| | TA11 | 50 | .0079355 | |
| | TA21 | 50 | .0469712 | |
| | HC | 50 | | .5885714 |
| | Sig. | | .928 | 1.000 |
| Duncan[a,b] | GA | 50 | .0000194 | |
| | TA12 | 50 | .0000690 | |
| | SA | 50 | .0001212 | |
| | TA22 | 50 | .0051760 | |
| | TA11 | 50 | .0079355 | |
| | TA21 | 50 | .0469712 | |
| | HC | 50 | | .5885714 |
| | Sig. | | .347 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 4.554E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 89.** Homogeneous Subsets, FAP Size Forty Efficiency

In considering all the size forty problem instances as a whole, the validated ANOVA led to the generation of the homogeneous subsets shown in Exhibit 90. Like the FAP problem instances of size forty and the composite experiment set of size thirty, neither the Tukey nor the Duncan method was able to distinguish between the efficiencies of any of the algorithms other than Hill Climbing, and all were placed in the second place subset together.

| | | | Subset | |
|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 |
| Tukey HSD[a,b] | GA | 150 | .0001139 | |
| | SA | 150 | .0009187 | |
| | TA22 | 150 | .0017835 | |
| | TA12 | 150 | .0026606 | |
| | TA11 | 150 | .0070751 | |
| | TA21 | 150 | .0167882 | |
| | HC | 150 | | .2449298 |
| | Sig. | | .907 | 1.000 |
| Duncan[a,b] | GA | 150 | .0001139 | |
| | SA | 150 | .0009187 | |
| | TA22 | 150 | .0017835 | |
| | TA12 | 150 | .0026606 | |
| | TA11 | 150 | .0070751 | |
| | TA21 | 150 | .0167882 | |
| | HC | 150 | | .2449298 |
| | Sig. | | .319 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.536E-02.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 90.** Homogeneous Subsets, Composite Size Forty Efficiency

The ANOVA for this set of experiments also noted an interaction between the problem type and algorithm efficiency. Exhibit 91 shows the plot of the efficiencies for the three problem types. The patterns of similar shape of the lines and generally better efficiency ratios for FAP continued for this experiment set, although the ratios seem to be on the decrease compared to the smaller problem size experiments.

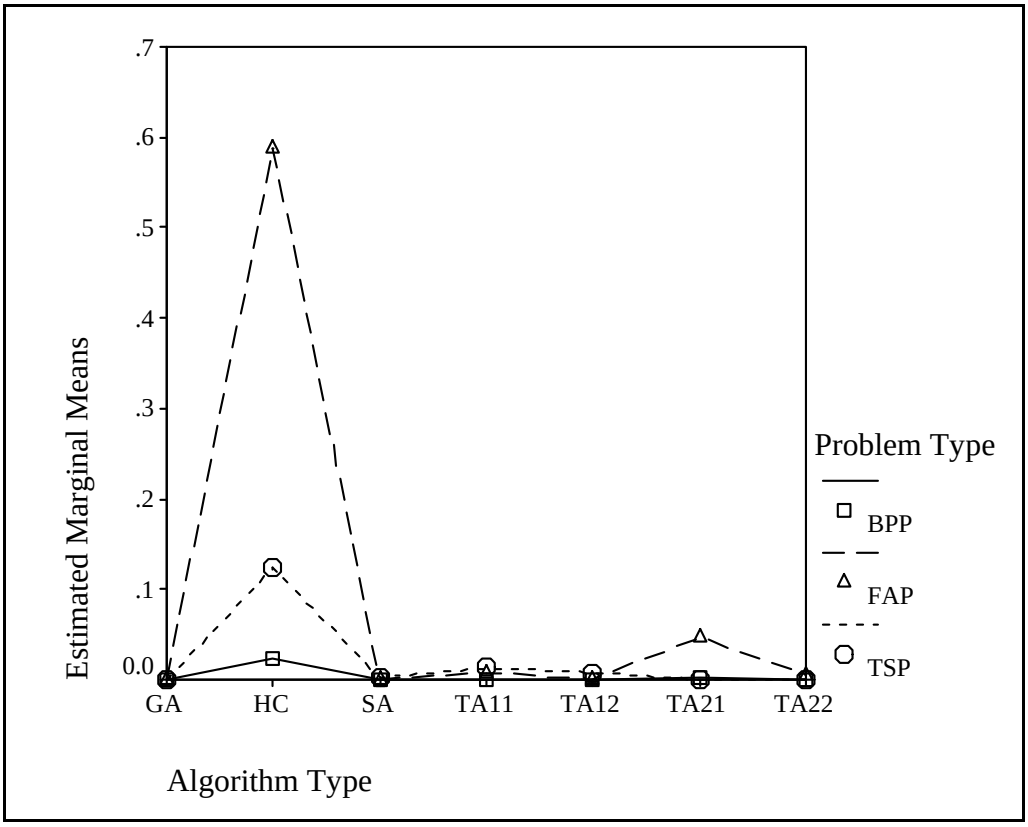


**Exhibit 91.** Problem Type Plot, Composite Size Forty Efficiency

#### 2.2.7.2.5 Problem Size Fifty Efficiency Results

Next to be analyzed was the experiment set consisting of problem instances of size fifty. For the TSP problem instances in this set, the validated ANOVA preceded the generation of the homogeneous subsets as shown in Exhibit 92. Though the Tukey and Duncan methods produced essentially the same subsets, the Tukey method did twice place an algorithm in two different subsets. The Duncan method named GELS method one with single stepping to the second place slot by itself, while the Tukey method grouped it together with GELS method one with multiple stepping.

| | | | Subset | | | |
|---|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 | 4 |
| Tukey HSD[a,b] | TA22 | 50 | .0001466 | | | |
| | TA21 | 50 | .0001594 | | | |
| | GA | 50 | .0002751 | | | |
| | SA | 50 | .0027098 | .0027098 | | |
| | TA12 | 50 | | .0063639 | .0063639 | |
| | TA11 | 50 | | | .0101841 | |
| | HC | 50 | | | | .1042789 |
| | Sig. | | .504 | .112 | .083 | 1.000 |
| Duncan[a,b] | TA22 | 50 | .0001466 | | | |
| | TA21 | 50 | .0001594 | | | |
| | GA | 50 | .0002751 | | | |
| | SA | 50 | .0027098 | | | |
| | TA12 | 50 | | .0063639 | | |
| | TA11 | 50 | | | .0101841 | |
| | HC | 50 | | | | .1042789 |
| | Sig. | | .089 | 1.000 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 4.715E-05.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 92.** Homogeneous Subsets, TSP Size Fifty Efficiency

For the BPP problem instances, the validated ANOVA led to the generation of the homogeneous subsets, and these are shown in Exhibit 93. In this case both the Tukey and Duncan methods produced the same subsets, and named GELS method two with single stepping as the second best efficiency for this group of problems.

| | Algorithm Type | N | Subset 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| Tukey HSD[a,b] | TA22 | 50 | .0000060 | | | |
| | TA12 | 50 | .0000240 | | | |
| | GA | 50 | .0000470 | | | |
| | TA11 | 50 | | .0003040 | | |
| | SA | 50 | | .0003534 | | |
| | TA21 | 50 | | | .0026562 | |
| | HC | 50 | | | | .0183102 |
| | Sig. | | .996 | .988 | 1.000 | 1.000 |
| Duncan[a,b] | TA22 | 50 | .0000060 | | | |
| | TA12 | 50 | .0000240 | | | |
| | GA | 50 | .0000470 | | | |
| | TA11 | 50 | | .0003040 | | |
| | SA | 50 | | .0003534 | | |
| | TA21 | 50 | | | .0026562 | |
| | HC | 50 | | | | .0183102 |
| | Sig. | | .554 | .446 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 1.047E-07.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 93.** Homogeneous Subsets, BPP Size Fifty Efficiency

For the FAP problem instances, the validated ANOVA led to the homogeneous subsets shown in Exhibit 94. Like the FAP problem instances of its predecessors, both Tukey's method and Duncan's method failed to distinguish between the efficiencies of any of the algorithms except Hill Climbing, placing them all into the same second place subset.

| | | | Subset | |
|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 |
| Tukey HSD[a,b] | GA | 50 | .0000185 | |
| | TA12 | 50 | .0000871 | |
| | SA | 50 | .0001758 | |
| | TA11 | 50 | .0057499 | |
| | TA22 | 50 | .0064368 | |
| | TA21 | 50 | .0303496 | |
| | HC | 50 | | .5137097 |
| | Sig. | | .956 | 1.000 |
| Duncan[a,b] | GA | 50 | .0000185 | |
| | TA12 | 50 | .0000871 | |
| | SA | 50 | .0001758 | |
| | TA11 | 50 | .0057499 | |
| | TA22 | 50 | .0064368 | |
| | TA21 | 50 | .0303496 | |
| | HC | 50 | | .5137097 |
| | Sig. | | .398 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 2.351E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 94.** Homogeneous Subsets, FAP Size Fifty Efficiency

In examining the composite results for the size fifty experiment set, the validated ANOVA was followed by the generation of the homogeneous subsets shown in Exhibit 95. Once again, as with the FAP and composite problem groups of other sizes, neither Tukey's nor Duncan's method could distinguish between any of the algorithms other than Hill Climbing, and thus there was no outright second place finisher in efficiency for this set of experiments.

| | | | Subset | |
|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 |
| Tukey HSD[a,b] | GA | 150 | .0001135 | |
| | SA | 150 | .0010797 | |
| | TA12 | 150 | .0021583 | |
| | TA22 | 150 | .0021965 | |
| | TA11 | 150 | .0054127 | |
| | TA21 | 150 | .0110551 | |
| | HC | 150 | | .2120996 |
| | Sig. | | .938 | 1.000 |
| Duncan[a,b] | GA | 150 | .0001135 | |
| | SA | 150 | .0010797 | |
| | TA12 | 150 | .0021583 | |
| | TA22 | 150 | .0021965 | |
| | TA11 | 150 | .0054127 | |
| | TA21 | 150 | .0110551 | |
| | HC | 150 | | .2120996 |
| | Sig. | | .363 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 7.922E-03.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 95.** Homogeneous Subsets, Composite Size Fifty Efficiency

Along with these results, the ANOVA had once again detected an interaction between problem type and algorithm efficiency, as displayed in Exhibit 96. The now-familiar patterns of similar line shape, general superiority of efficiency by algorithms for FAP problem instances, and a decreasing level of efficiency for FAP problem instances were once more noted in this diagram.

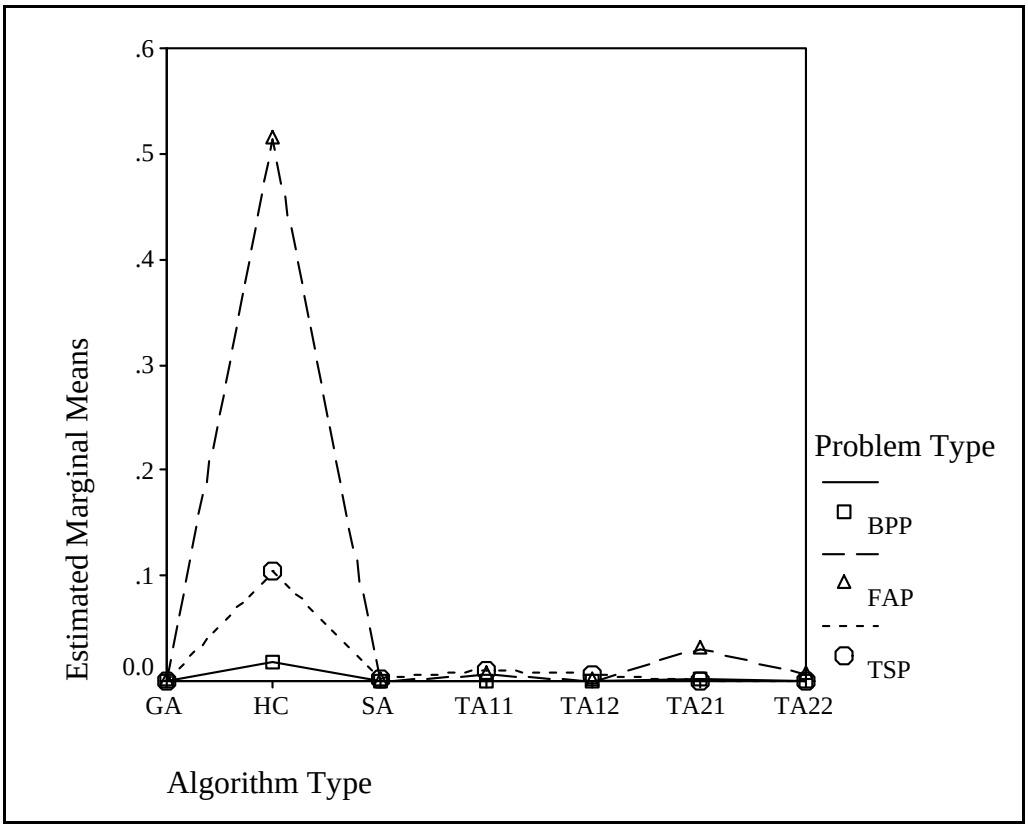


**Exhibit 96.** Problem Type Plot, Composite Size Fifty Efficiency

## 2.2.7.2.6 Random Problem Size Efficiency Results

To round out the analyses of efficiency, the set of problem instances having randomly assigned problem sizes was examined. The ANOVA for the TSP problem instance group was validated and followed by the generation of the homogeneous subsets, shown in Exhibit 97. The lists produced by the two methods were essentially the same, except that the Duncan method put GELS method one with single stepping in second place by itself, while the Tukey method coupled it with GELS method one with multiple stepping (which appeared in two subsets).

| | Algorithm Type | N | Subset 1 | Subset 2 | Subset 3 |
|---|---|---|---|---|---|
| Tukey HSD[a,b] | TA22 | 50 | .0001679 | | |
| | GA | 50 | .0002075 | | |
| | TA21 | 50 | .0002601 | | |
| | SA | 50 | .0015647 | | |
| | TA12 | 50 | .0105850 | .0105850 | |
| | TA11 | 50 | | .0229078 | |
| | HC | 50 | | | .2409103 |
| | Sig. | | .408 | .210 | 1.000 |
| Duncan[a,b] | TA22 | 50 | .0001679 | | |
| | GA | 50 | .0002075 | | |
| | TA21 | 50 | .0002601 | | |
| | SA | 50 | .0015647 | | |
| | TA12 | 50 | .0105850 | | |
| | TA11 | 50 | | .0229078 | |
| | HC | 50 | | | .2409103 |
| | Sig. | | .073 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 6.689E-04.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 97.** Homogeneous Subsets, TSP Random Size Efficiency

For the BPP problem instances, the validated ANOVA was again followed by the generation of the homogeneous subsets, which appear in Exhibit 98. The lists produced by the two methods are identical, putting GELS method two with single stepping into the number two slot for efficiency for this group of problems.

| | | | Subset | | |
|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 |
| Tukey HSD[a,b] | TA22 | 50 | .0000084 | | |
| | TA12 | 50 | .0000240 | | |
| | GA | 50 | .0000362 | | |
| | TA11 | 50 | .0002640 | | |
| | SA | 50 | .0002916 | | |
| | TA21 | 50 | | .0033279 | |
| | HC | 50 | | | .0321323 |
| | Sig. | | .993 | 1.000 | 1.000 |
| Duncan[a,b] | TA22 | 50 | .0000084 | | |
| | TA12 | 50 | .0000240 | | |
| | GA | 50 | .0000362 | | |
| | TA11 | 50 | .0002640 | | |
| | SA | 50 | .0002916 | | |
| | TA21 | 50 | | .0033279 | |
| | HC | 50 | | | .0321323 |
| | Sig. | | .549 | 1.000 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 4.217E-06.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 98.** Homogeneous Subsets, BPP Random Size Efficiency

For the FAP problem instances, the validated ANOVA was again followed by the generation of the homogeneous subsets, as shown in Exhibit 99. Like other FAP problem instance groups, the Tukey method was unable to distinguish between the algorithms for efficiency other than Hill Climbing. However, in this case the Duncan method was able to make somewhat of a distinction, placing

GELS method one with single stepping and GELS method two with single stepping into a second place subset apart from the others.

| | | | Subset | | |
|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 |
| Tukey HSD[a,b] | GA | 50 | .0000226 | | |
| | SA | 50 | .0001906 | | |
| | TA12 | 50 | .0045733 | | |
| | TA22 | 50 | .0060165 | | |
| | TA21 | 50 | .0592630 | | |
| | TA11 | 50 | .0881797 | | |
| | HC | 50 | | .6348052 | |
| | Sig. | | .124 | 1.000 | |
| Duncan[a,b] | GA | 50 | .0000226 | | |
| | SA | 50 | .0001906 | | |
| | TA12 | 50 | .0045733 | | |
| | TA22 | 50 | .0060165 | | |
| | TA21 | 50 | .0592630 | .0592630 | |
| | TA11 | 50 | | .0881797 | |
| | HC | 50 | | | .6348052 |
| | Sig. | | .119 | .391 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 2.830E-02.

a. Uses Harmonic Mean Sample Size = 50.000.

b. Alpha = .05.

**Exhibit 99.** Homogeneous Subsets, FAP Random Size Efficiency

To complete the second phase of analysis, the problem instances with randomly generated problem sizes were examined across problem types. Once more the ANOVA was conducted and corroborated, and the homogeneous subsets were

generated as shown in Exhibit 100. Both the Tukey and Duncan methods generated three subsets. Duncan's method awarded the second spot in a tie to GELS method one with single stepping and GELS method two with single stepping. Tukey's method concurred with this assessment, but added GELS method one with multiple stepping to make it a three-way tie for the second best efficiency for this set of experiments.

| | | | Subset | | |
|---|---|---|---|---|---|
| | Algorithm Type | N | 1 | 2 | 3 |
| Tukey HSD[a,b] | GA | 150 | .0000888 | | |
| | SA | 150 | .0006823 | | |
| | TA22 | 150 | .0020643 | | |
| | TA12 | 150 | .0050608 | .0050608 | |
| | TA21 | 150 | .0209503 | .0209503 | |
| | TA11 | 150 | | .0371172 | |
| | HC | 150 | | | .3026159 |
| | Sig. | | .541 | .080 | 1.000 |
| Duncan[a,b] | GA | 150 | .0000888 | | |
| | SA | 150 | .0006823 | | |
| | TA22 | 150 | .0020643 | | |
| | TA12 | 150 | .0050608 | | |
| | TA21 | 150 | .0209503 | .0209503 | |
| | TA11 | 150 | | .0371172 | |
| | HC | 150 | | | .3026159 |
| | Sig. | | .108 | .161 | 1.000 |

Means for groups in homogeneous subsets are displayed.
Based on Type III Sum of Squares
The error term is Mean Square(Error) = 9.959E-03.

a. Uses Harmonic Mean Sample Size = 150.000.

b. Alpha = .05.

**Exhibit 100.** Homogeneous Subsets, Composite Random Size Efficiency

Once more, the ANOVA found an interaction between problem type and algorithm efficiency, and this is displayed in Exhibit 101. The same patterns appear as for the other experiment sets, with comparable line shapes indicating similar effect of the problem types on each algorithm and a generally higher efficiency obtained by FAP problem instances.

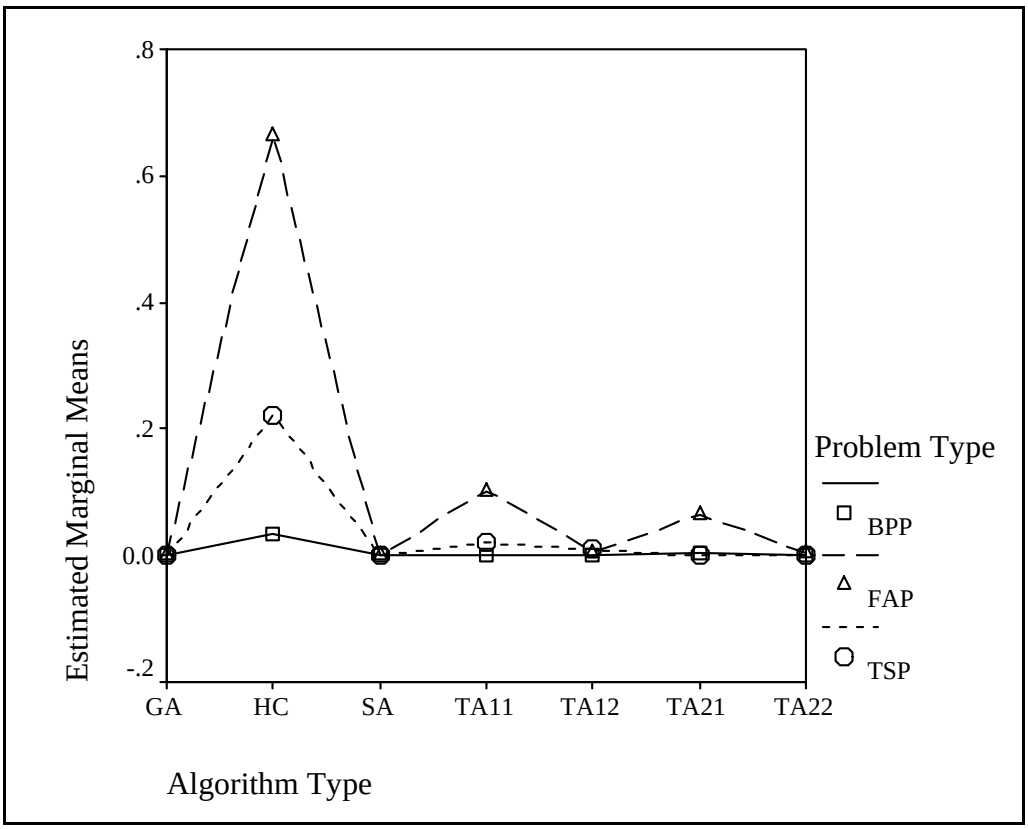


**Exhibit 101.** Problem Type Plot, Composite Random Size Efficiency

Finally, the ANOVA also found an interaction between the problem size and the algorithm efficiency, as displayed in Exhibit 102. Like the plot showing the

effect of problem type on the algorithms' efficiency, this plot shows a very consistent shape to the lines, indicating that the problem size, like the type, has a similar effect on the efficiencies of all algorithms. There also appears a downward trend in efficiency for each algorithm (where visible) with increasing problem size.

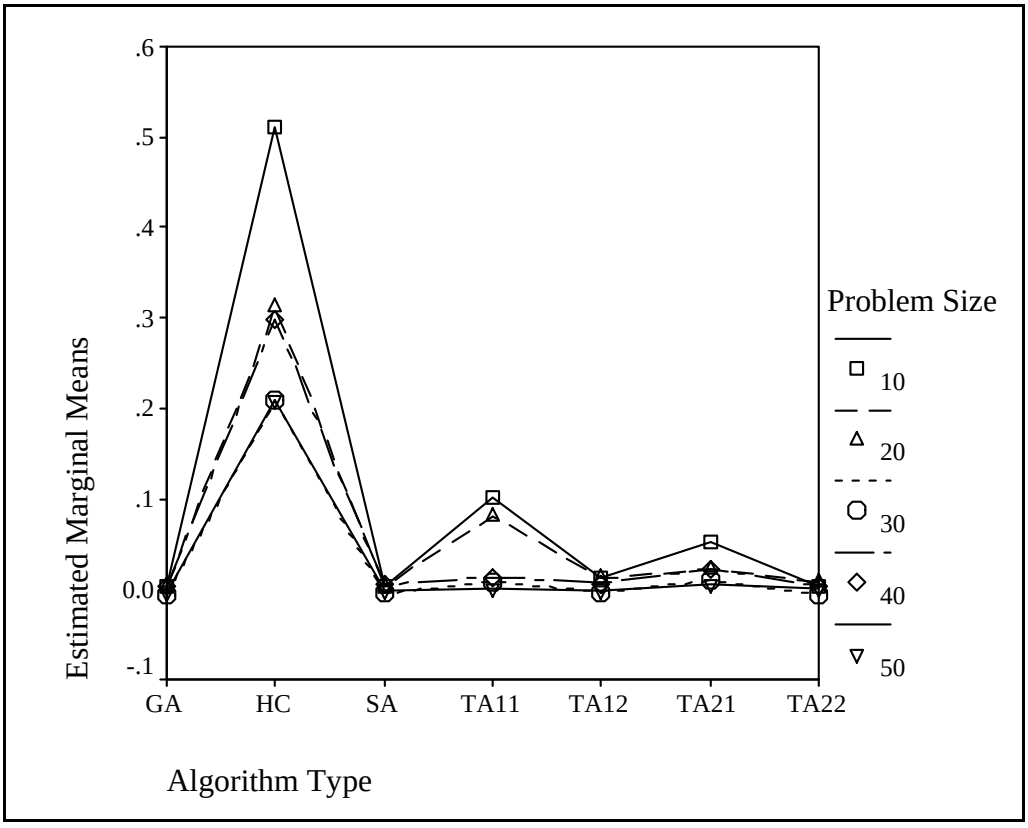


**Exhibit 102.** Problem Size Plot, Composite Random Size Efficiency

This completed the two phases of analysis comparing the algorithms' performance and efficiency. In all cases, both the performance and the efficiency hypotheses had been rejected, showing solid evidence that there was indeed a

significant difference in the ability of the different algorithms to produce high quality solutions to the generated problem instances, and also a significant difference in how much of the search space was examined by the different algorithms in producing their solutions. At every step, the results of the experiments were carefully analyzed, maintaining a watch on any and all necessary conditions to ensure that they were met and if not, that alternate sources of evidence were examined to verify the original results. All of this was done in an effort to ensure the integrity of the analyses and provide a firm grounding for establishing conclusions. In this way, any conclusions that would be drawn regarding the experiments in general or the GELS algorithm in particular would be based on solid statistical procedures and not merely on the wishful thinking of the author.

# 3 Research Efforts Summary and Evaluation

## 3.1 Interpretation of Research Results

In interpreting the results of the two phases of analysis, it was evident that there was no clear "winner" in terms of best performance and efficiency over all experiment sets. No one algorithm had managed to be that dominant, even in a single category (with the exception of Hill Climbing efficiency, for the reasons already noted). In order to better bring the results into focus and provide a clearer picture of the results for drawing appropriate conclusions, the results data needed to be consolidated and collated.

Exhibit 103 shows a concise view of all the algorithm comparison results. For each problem type ("COM" is used here to identify the composite of all problem instances across problem types) and size, the ranking of each algorithm is given in terms of which homogeneous subset it was placed into. Thus, a value of 1 indicates the algorithm was in the top ranked subset, higher values indicate lower ranked subsets. The rankings are identified as "P" for performance and "E" for efficiency, and each entry consists of the Tukey method ranking and the Duncan method ranking, respectively, separated by a slash.

| | | Algorithm Rankings | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **HC** | | **GA** | | **SA** | | **TA11** | | **TA12** | | **TA21** | | **TA22** | |
| Type | **Size** | **P** | **E** | **P** | **E** | **P** | **E** | **P** | **E** | **P** | **E** | **P** | **E** | **P** | **E** |
| TSP | **10** | 4/4 | 1/1 | 2/2 | 3/3 | 1/1 | 3/3 | 3/3 | 2/2 | 2/2 | 2/3 | 1/1 | 3/3 | 1/1 | 3/3 |
| | **20** | 4/5 | 1/1 | 4/5 | 3/4 | 1/1 | 3/4 | 4/5 | 2/2 | 3/4 | 2/3 | 2/3 | 3/4 | 2/2 | 3/4 |
| | **30** | 4/4 | 1/1 | 5/5 | 4/4 | 1/1 | 3/4 | 5/5 | 2/2 | 4/4 | 2/3 | 3/3 | 4/4 | 2/2 | 4/4 |
| | **40** | 3/3 | 1/1 | 5/6 | 3/4 | 1/1 | 3/4 | 4/5 | 2/2 | 4/4 | 2/3 | 3/3 | 3/4 | 2/2 | 3/4 |
| | **50** | 2/2 | 1/1 | 6/7 | 4/4 | 1/1 | 3/4 | 5/6 | 2/2 | 4/5 | 2/3 | 3/4 | 4/4 | 2/3 | 4/4 |
| | **R** | 4/4 | 1/1 | 5/5 | 3/3 | 1/1 | 3/3 | 5/5 | 2/2 | 4/4 | 2/3 | 3/3 | 3/3 | 2/2 | 3/3 |
| BPP | **10** | 3/2 | 1/1 | 1/1 | 2/3 | 2/2 | 2/3 | 3/3 | 2/3 | 4/3 | 2/3 | 1/1 | 2/2 | 3/3 | 2/3 |
| | **20** | 2/2 | 1/1 | 1/1 | 3/3 | 2/2 | 3/3 | 2/3 | 3/3 | 3/5 | 3/3 | 1/1 | 2/2 | 3/4 | 3/3 |
| | **30** | 2/2 | 1/1 | 1/2 | 3/4 | 3/3 | 3/3 | 3/3 | 3/3 | 4/5 | 3/4 | 1/1 | 2/2 | 4/4 | 3/4 |
| | **40** | 1/2 | 1/1 | 2/2 | 3/4 | 3/3 | 3/3 | 3/3 | 3/3 | 4/5 | 3/4 | 1/1 | 2/2 | 4/4 | 4/4 |
| | **50** | 1/1 | 1/1 | 2/2 | 4/4 | 3/3 | 3/3 | 3/3 | 3/3 | 4/5 | 4/4 | 1/1 | 2/2 | 4/4 | 4/4 |
| | **R** | 1/2 | 1/1 | 1/1 | 3/3 | 2/3 | 3/3 | 2/3 | 3/3 | 4/5 | 3/3 | 1/1 | 2/2 | 3/4 | 3/3 |
| FAP | **10** | 3/3 | 1/1 | 4/4 | 3/4 | 1/1 | 3/4 | 2/2 | 2/2 | 4/4 | 3/4 | 1/1 | 2/3 | 2/2 | 3/4 |
| | **20** | 4/3 | 1/1 | 4/4 | 3/3 | 1/1 | 3/3 | 3/2 | 2/2 | 1/1 | 3/3 | 2/2 | 2/2 | 1/1 | 2/3 |
| | **30** | 2/3 | 1/1 | 3/4 | 2/2 | 1/1 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 1/1 | 2/2 | 1/1 | 2/2 |
| | **40** | 2/2 | 1/1 | 2/2 | 2/2 | 1/1 | 2/2 | 1/1 | 2/2 | 1/1 | 2/2 | 1/1 | 2/2 | 1/1 | 2/2 |
| | **50** | 3/3 | 1/1 | 3/3 | 2/2 | 1/1 | 2/2 | 1/1 | 2/2 | 2/2 | 2/2 | 1/1 | 2/2 | 1/1 | 2/2 |
| | **R** | 3/4 | 1/1 | 3/4 | 2/3 | 1/1 | 2/3 | 1/2 | 2/2 | 2/3 | 2/3 | 1/1 | 2/2 | 1/2 | 2/3 |
| COM | **10** | 4/4 | 1/1 | 3/3 | 4/4 | 1/1 | 4/4 | 3/3 | 2/2 | 4/4 | 3/4 | 1/1 | 3/3 | 2/2 | 4/4 |
| | **20** | 4/4 | 1/1 | 4/4 | 3/3 | 1/1 | 3/3 | 3/3 | 2/2 | 3/3 | 3/3 | 2/2 | 2/2 | 2/2 | 2/3 |
| | **30** | 2/2 | 1/1 | 3/3 | 2/2 | 1/1 | 2/2 | 3/3 | 2/2 | 3/3 | 2/2 | 1/1 | 2/2 | 2/2 | 2/2 |
| | **40** | 2/4 | 1/1 | 4/6 | 2/2 | 1/1 | 2/2 | 2/3 | 2/2 | 3/5 | 2/2 | 1/2 | 2/2 | 2/3 | 2/2 |
| | **50** | 3/4 | 1/1 | 4/5 | 2/2 | 1/1 | 2/2 | 2/3 | 2/2 | 3/4 | 2/2 | 1/2 | 2/2 | 2/3 | 2/2 |
| | **R** | 4/4 | 1/1 | 4/5 | 3/3 | 1/1 | 3/3 | 2/3 | 2/2 | 3/3 | 2/3 | 1/1 | 2/2 | 2/2 | 3/3 |

**Exhibit 103.** Summary of Algorithm Comparison Rankings

By taking a simple average of the rankings over each of the problem types, Exhibit 104 is produced. Here the averages for the Tukey method are located in the rows marked with a "T" in the column labeled "M" (for "Method"). The Duncan method averages are located in the "D" rows.

| | | Average Algorithm Rankings | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **HC** | | **GA** | | **SA** | | **TA11** | | **TA12** | | **TA21** | | **TA22** | |
| Type | **M** | **P** | **E** | **P** | **E** | **P** | **E** | **P** | **E** | **P** | **E** | **P** | **E** | **P** | **E** |
| TSP | **T** | 3.50 | 1.00 | 4.50 | 3.33 | 1.00 | 3.00 | 4.33 | 2.00 | 3.50 | 2.00 | 2.50 | 3.33 | 1.83 | 3.33 |
| | **D** | 3.67 | 1.00 | 5.00 | 3.67 | 1.00 | 3.67 | 4.83 | 2.00 | 3.83 | 3.00 | 2.83 | 3.67 | 2.00 | 3.67 |
| BPP | **T** | 1.67 | 1.00 | 1.33 | 3.00 | 2.50 | 2.83 | 2.67 | 2.83 | 3.83 | 3.00 | 1.00 | 2.00 | 3.50 | 3.17 |
| | **D** | 1.83 | 1.00 | 1.50 | 3.50 | 2.67 | 3.00 | 3.00 | 3.00 | 4.67 | 3.50 | 1.00 | 2.00 | 3.83 | 3.50 |
| FAP | **T** | 2.83 | 1.00 | 3.17 | 2.33 | 1.00 | 2.33 | 1.67 | 2.00 | 2.00 | 2.33 | 1.17 | 2.00 | 1.17 | 2.17 |
| | **D** | 3.00 | 1.00 | 3.50 | 2.67 | 1.00 | 2.67 | 1.67 | 2.00 | 2.17 | 2.67 | 1.17 | 2.17 | 1.33 | 2.67 |
| COM | **T** | 3.17 | 1.00 | 3.67 | 2.67 | 1.00 | 2.67 | 2.50 | 2.00 | 3.17 | 2.33 | 1.17 | 2.17 | 2.00 | 2.50 |
| | **D** | 3.67 | 1.00 | 4.33 | 2.67 | 1.00 | 2.67 | 3.00 | 2.00 | 3.67 | 2.67 | 1.50 | 2.17 | 2.33 | 2.67 |

**Exhibit 104.** Averages of Algorithm Comparison Rankings

Using the numbers in Exhibits 103 and 104 as a guide, Exhibit 105 shows some basic recommendations that can be made regarding choice of algorithm for solving the various problem types. These are only suggestions based on the experimental analyses, and cannot be considered hard-and-fast rules for selecting an algorithm to solve a problem. For each set of selection criteria, the top three candidate algorithms are shown in order, separated by a slash.

| If the primary interest is… | And the secondary interest is… | And the problem to be solved is… | The solution algorithm of choice should likely be… |
|---|---|---|---|
| **Performance** | N/A | TSP | SA / TA22 / TA21 |
| **Performance** | Efficiency | TSP | SA / TA22 / HC |
| **Efficiency** | N/A | TSP | HC / TA11 / TA12 |
| **Efficiency** | Performance | TSP | HC / SA / TA11 |
| **Performance** | N/A | BPP | TA21 / GA / HC |
| **Performance** | Efficiency | BPP | TA21 / HC / GA |
| **Efficiency** | N/A | BPP | HC / TA21 / (SA, TA11) |
| **Efficiency** | Performance | BPP | HC / TA21 / GA |
| **Performance** | N/A | FAP | SA / TA21 / TA22 |
| **Performance** | Efficiency | FAP | TA21 / SA / TA22 |
| **Efficiency** | N/A | FAP | HC / TA11 / TA21 |
| **Efficiency** | Performance | FAP | HC / TA21 / TA11 |
| **Performance** | N/A | Any | SA / TA21 / TA22 |
| **Performance** | Efficiency | Any | SA / TA21 / TA22 |
| **Efficiency** | N/A | Any | HC / TA11 / TA21 |
| **Efficiency** | Performance | Any | HC / TA21 / SA |

**Exhibit 105.** Algorithm Selection Suggestions

Based on the data from the experiments and the summaries in Exhibits 103, 104, and 105, a number of observations can be made regarding the results of the research:

1. For the TSP problems, Simulated Annealing was the overall best. It had the best performance ratings, and though its efficiency was mediocre it had much better performance numbers than the algorithms with better efficiencies.
2. For the BPP problems, GELS method two with single stepping was the overall best. It had the best performance ratings, and the best efficiency ratings aside from Hill Climbing (which didn't have the performance numbers that GELS had).
3. For the FAP problems, it was a close call. Simulated Annealing and GELS method two with single stepping were virtually tied in terms of performance, with Simulated Annealing rated number one for all problem sizes by both the Tukey and Duncan methods, and GELS rated number one for all problem sizes but one (for which it was rated number two), also by both methods. GELS also had slightly better efficiency numbers.
4. For the overall composite cases, it was another close call. Again, Simulated Annealing and GELS method two with single stepping were virtually tied in terms of performance, with Simulated Annealing rated number one for all problem sizes by both the Tukey and Duncan methods, and GELS rated number one for all problem sizes but one by the Tukey method and three of

the problem sizes by the Duncan method (the other three being number two). And again, GELS had the better efficiency numbers.

5. The performance of GELS method one with single stepping was mediocre, with the exception of a good showing on the FAP problems. Its one shining spot was its efficiency, second only to Hill Climbing (and having better performance than Hill Climbing most of the time).
6. The performance of GELS method one with multiple stepping was also mediocre, across the board. It also had very mediocre efficiency numbers.
7. The performance of GELS method two with single stepping was very good, winning one category of problems outright (the BPP problems) and coming very close to winning two others, including the overall. It also had very good efficiency numbers. It did seem to have some difficulty with TSP, posting numbers for both performance and efficiency that were substantially worse than it received for the other problem types.
8. The performance of GELS method two with multiple stepping showed some bright spots, but it had some problems with efficiency, posting quite mediocre numbers.

## 3.2 Overall Evaluation of GELS

In the research experimentation, the GELS algorithm in its various combinations gave a very good showing. The method two variations had very

good performance, finishing near the top of the rankings. Method two with single stepping was the clear performance winner for BPP problem instances, and it came very close to having the best performance for FAP problem instances, and again for all problem types in general. The single stepping variations of the algorithm also finished at the top of the rankings for efficiency, besting all algorithms except the greedy Hill Climbing.

The other algorithms used in the study have been studied and optimized for years, and the parameters that were used to run them during the experiments were set to values that had been found over the course of much study to be suited for producing good results for the types of problems in use. The GELS algorithm, however, had only been under study for a relatively short period of time. The parameters used to run it had undergone a number of changes during its development, and there certainly was not a period of many years of tweaking and tuning behind the settings that were used for them during the experiments. In spite of these handicaps, GELS was able to go head-to-head with the much more mature algorithms on very well studied problems and in many cases beat them in terms of both quality of solutions produced and efficiency of search.

This then is the contribution of GELS to the literature. It is novel; a search of the literature at the beginning of this research revealed nothing that referenced the use of the principles of gravitation to guide the search of an optimization algorithm. Furthermore, it cannot be classified as merely a variation on the theme of another algorithm. Though it does always tend to move towards better solutions, it is not

purely greedy because it does not always move towards the best solutions available. Though it contains some elements of randomness, its movement through a search space is not random, but quite deterministic. And though it contains several elements in common with other algorithms, such as a heuristic to guide the search and a mechanism for escaping local optima, by definition all local search algorithms will contain those elements, and GELS employs them in a different way than the others. Finally, though new it was able to withstand the rigors of statistical examination on a variety of problem types and have that examination report operation on a par with if not better than the other algorithms.

Still, there are many opportunities for future research. Many of the opportunities lie in further study of the algorithm and its operation, such as:

- Investigating the algorithm heuristic to find out if using the mass components instead of just the difference between objective function values can be made cost-effective and beneficial
- Investigating the use of a fixed number of elements in the velocity vector
- Reintroducing the concept of resistive force, used in the early experiments but found to be too cumbersome for use in the research experiments (but perhaps could be useful if "streamlined" and made easier to control)
- Experimentation with different mechanisms for updating the velocity vector
- Experimentation with different mechanisms for multiple step motion

- Testing different combinations of parameter values to find settings inclined to produce better solutions
- Inserting some additional randomness into the procedure, like occasional random events that cause movement direction to shift
- Attempt to put more “intelligence” into the algorithm, e.g. allowing it to automatically alter its operation as it acquires information about the problem and senses the need for adjustment

Of course, there is also the possibility of conducting further studies with GELS using different problem types and comparison algorithms. It is the suspicion of this author, based on tantalizing data received during the early experiments with the algorithm, that the multiple step movement option would prove to be quite useful in problems where the search space is sparse, that is, contains very few valid solutions. Designing some experiments with algorithms of that nature to test this theory would provide useful information, regardless of whether or not the theory turned out to be correct.

## 3.3 Conclusion

To say that the research experimentation revealed the GELS algorithm to bring revolutionary new capabilities in solving combinatorial optimization problems to the table would be a falsehood. But, to say that it is useful only as a potential teaching vehicle would also be incorrect. It outperformed Hill Climbing and a

Genetic Algorithm, two styles of algorithm that are in widespread use to solve a variety of combinatorial optimization problems. It was tested against several common problem types and sizes which, although limited in number by the restrictions of the available analysis tool, nonetheless provided more than enough cases for the statistical analysis to produce solid backing for its capabilities.

For GELS to be relegated to an occasional mention in passing as an example of optimization algorithms that emulate natural processes to produce solutions would be to ignore that statistical backing. Certainly more study is required before the full capabilities and usefulness of the GELS algorithm will be known, but this research has demonstrated that such an undertaking would be worthwhile.

# References


Aarts, E. and J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*. New York, NY: John Wiley & Sons, Ltd., 1997.

Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullmann. *Data Structures and Algorithms*. Reading, MA: Addison-Wesley, 1983.

Arora, Sanjeev. "Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems." *Journal of the ACM*, vol. 45, iss. 5 (September 1998): 753-782.

Bernhard, Philip J. and Kevin L. Fox. "Experimental Evaluation of Techniques for Database File Assignment." N.P., 2000.

Bresina, John. "Heuristic-Biased Stochastic Sampling." *Proceedings of the 13th National Conference on Artificial Intelligence* (1996).

Bresina, John, Mark Drummond, and Keith Swanson. "Search Space Characterization for a Telescope Scheduling Application." Working notes of the AAAI Fall Symposium, *Planning and Learning: On to Real Applications*, 1994.

Corman, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1991.

Dowdy, Lawrence W. and Derrell V. Foster. “Comparative Models of the File Assignment Problem.” *ACM Computing Surveys*, vol. 14, no. 2 (June 1982).

Duncan, D. B. “Multiple Range and Multiple F Tests.” *Biometrics*, vol. 11 (1955): 1-42.

Englemore, Robert S. and Anthony J. Morgan, eds. *Blackboard Systems*. Reading, MA: Addison-Wesley, 1988.

Freuder et al. “Systematic Versus Stochastic Constraint Satisfaction.” *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (1995): 2027-2032.

Garey, Michael R. and Johnson, David S. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. New York, NY: W. H. Freeman, 1979.

Giarratano, Joseph C. and Gary Riley. *Expert Systems: Principles and Programming*, 2nd ed. Boston, MA: PWS Publishing Company, 1993.

Goodman, Erik D. "An Introduction to GALOPPS -- the Genetic ALgorithm Optimized for Portability and Parallelism System, Release 3.2." Technical Report 96-07-01, Intelligent Systems Laboratory and Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University (July 16, 1996).

Harrell, C., B. Ghosh, and R. Bowden. *Simulation Using ProModel*, 3rd ed. Boston, MA: McGraw-Hill, 2000.

Hewlett-Packard Corporation. “Zero Latency Enterprise Architecture.” White paper (2002).

Ingber, Lester. “Adaptive Simulated Annealing (ASA).” Global optimization C-Code, Caltech Alumni Association, Pasadena, CA (1993).

Karmarkar, N. and R. M. Karp. “An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem.” *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science (1982): 312-320*.

Kliewer, Georg and Stefan Tschöke. "A general parallel simulated annealing library (parSA) and its applications in industry." PAREO'98: First meeting of the PAREO working group on Parallel Processing in Operations Research, Versailles, France, July 8-10, 1998.

Kochetov, Yuri and Anjelika Usmanova. “Probabilistic Tabu Search with Exponential Neighborhood for Bin Packing Problem.” *Proceedings of the 4th Metaheuristics International Conference* (2001): 619-623.

Kondrak, Grzegorz, and Peter van Beek. “A Theoretical Evaluation of Selected Backtracking Algorithms.” *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (1995): 541-547.

Kruskal, W. H. and W. A. Wallis. “Use of Ranks on One Criterion Variance Analysis.” *Journal of the American Statistical Association*, vol. 47 (1952): 583-621. Corrections appear in vol. 48: 907-911.

Massey, F. J. Jr. “The Kolmogorov-Smirnov Test of Goodness of Fit.” *Journal of the American Statistical Association*, vol. 46 (1951).

Metropolis, N., A. Rosenblurb, M. Rosenblurb, A. Teller, and E. Teller. "Equation of State Calculations by Fast Computing Machines." *Journal of Chem. Physics*, vol. 21 (1953):1087-1092.

Montgomery, Douglas C. *Design and Analysis of Experiments*, 5th ed. New York, NY: John Wiley & Sons, Ltd., 2001.

Montgomery, Douglas C. and G. C. Runger. *Applied Statistics and Probability for Engineers*, 2nd ed. New York, NY: John Wiley & Sons, Ltd., 1999.

Newell, A., J. McDermott, and C. L. Forgy. *Artificial Intelligence: A Self-Paced Introductory Course*. Computer Science Department, Carnegie-Mellon University, 1977.

Oracle Corporation. *Oracle9i Administrator's Reference, Release 2 (9.2.0.1.0) for UNIX Systems: AIX-Based Systems, Compaq Tru64 UNIX, HP 9000 Series HP-UX, Linux Intel, and Sun Solaris*. Oracle Corporation, 2002.

Papadimitriou, Christos H. *Computational Complexity*. Reading, MA: 1994.

Pearl, Judea. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1985.

Prosser, Patrick. "Hybrid Algorithms for the Constraint Satisfaction Problem." *Computational Intelligence*, vol. 9, no. 3 (1993).

Sears, Francis W., Mark W. Zemansky, and Hugh D. Young. *University Physics*, 7th ed. Reading, MA: Addison-Wesley, 1987.

SPSS, Inc. *SPSS 11.0 Student Edition*. N.P., 2001.

Stewart, G. W. *Introduction to Matrix Computations*. New York, NY: Academic Press, 1973.

Transaction Processing Performance Council (TPC). *TPC Benchmark™ H (Decision Support)*. Standard Specification, Revision 2.1.0 (2002).

Tsang, Edward. *Foundations of Constraint Satisfaction*. London, UK: Academic Press Limited, 1996.

Tukey, J. W. "The Problem of Multiple Comparisons." N.P., Princeton University, (1953).

Voudouris, Chris and Edward Tsang, "Guided Local Search." Technical Report CSM-247, Department of Computer Science, University of Essex, UK (August 1995).

Wall, Matthew. "GALib: A C++ Library of GeneticAlgorithm Components." User documentation for GALib, version 2.4, documentation revision B (August 1996).

Walpole, Ronald E. and Raymond H. Myers. *Probability and Statistics for Engineers and Scientists*, 2nd ed. New York, NY: Macmillan Publishing Co., Inc., 1972.

Webster, Barry. An Object-Oriented Blackboard Expert System for Selecting Professional Baseball Players to Comprise a Team. Masters thesis, Florida Institute of Technology, 1995.

Webster, Barry and Philip J. Bernhard, “A Local Search Optimization Algorithm Based on Natural Principles of Gravitation.” *Proceedings of the International Conference on Information and Knowledge Engineering*, vol. 1. (2003): 255-261.