RT
213
c.1

Genetic and Heuristic Algorithms
For
Regrouping Service Sites

By
HIBA TABBARA

Submitted in partial fulfillment of the requirements
For the degree of Master of Science

Thesis Advisor : Dr. NASHAT MANSOUR

Natural Science Division
LEBANESE AMERICAN UNIVERSITY

Beirut

June 2000

LEBANESE AMERICAN UNIVERSITY

GRADUATE STUDIES

We hereby approve the thesis of

Hiba Tabbara

Candidate for the *Master of Science* degree.

Dr. Nashat Mansour

Signatures Redacted

Dr. George Nasr

Signatures Redacted

Dr. Haidar Harmanani

Signatures Redacted

Date

05/07/2000

I grant to the **LEBANESE AMERICAN UNIVERSITY** the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or to its students, agents and employees. I further agree that the University may reproduce and provide single copies of the work, in any format other than in or from microforms, to the public for the cost of reproduction.

GENETIC AND HEURISTIC ALGORITHMS
FOR
REGROUPING SERVICE SITES

ABSTRACT

by

HIBA TABBARA

The problem of regrouping service sites into a smaller number of service centers, such that a number of criteria are satisfied, is a realistic problem. Each service center then serves several customer-sites (e.g. towns) in a region. The objectives of regrouping are usually to consolidate human resources, improve service quality, reduce the cost of services, and centralize company branches, in addition to other application-dependent objectives. This regrouping problem is intractable and needs to be automated.

Our approach is based on a weighted graph problem formulation, and the solution has two phases. In the first phase, the graph is decomposed into the required number of sub-graphs (regions) using a tuned hybrid genetic algorithm (Tuned HGA). The second phase finds a suitable center within each region by applying a heuristic algorithm. Genetic algorithms are stochastic algorithms based on the mechanics of natural evolution. They are adapted in our work by using a problem-specific

objective function for the fitness of individuals. The algorithm is hybridized by a hill-climbing procedure in order to direct the search into profitable search sub-domains. The results of the HGA are tuned by using a problem-specific iterative improvement heuristic (IIH) that aim to remove anomalies and hence improve the final solution's quality. We also explore using a pre-processing step to reduce the graph vertex granularity for the purpose of further reducing the objective function value and improving the solution's quality. In the second phase, the heuristic algorithm favors higher-weight and well-centered vertices for selection to be centers within a region.

We empirically explored the behavior of the Tuned HGA and the center-selecting heuristic algorithm using a number of graphs, representing service sites with their inner-site distances. The empirical results show that:

- (a) The two-phase approach can be used for solving this problem,
- (b) Hybridization of the GA improves both its solution quality and evolution time,
- (c) The tuning IIH does improve the results by removing most anomalies,
- (d) Breaking up graph vertex granularity can be useful for graphs where vertex weights vary significantly, and
- (e) The center selection heuristic selects reasonable centers within regions.

To my parents

Acknowledgments

I would like to thank my advisor Dr. Nashat Mansour very much for his guidance and sincere support throughout my M.S. studies in general, and thesis work in particular. Thanks is also due to Dr. Haidar Harmanani and Dr. George Nasr for being on my Thesis committee. I would also like to thank the chairman of the Natural Science Division, Dr. Ahmad Kabbani, for his encouragement throughout my graduate studies. I also wish to thank the Lebanese National Council for Scientific Research for partially supporting this thesis work.

At last but not at least, very special note of thanks to my family and friends for their long support and encouragement.

Contents

1. Introduction	1
1.1 Overview of the Regrouping Service Sites Problem	1
1.2 Previous Work	2
1.3 Scope and Purpose	4
2. Problem Description and Objective Function	7
2.1 Preface	7
2.2 Problem Description	8
2.3 Objective Function for Graph Decomposition	10
2.4 Numerical Example	14
3. Overview of Our Solution for the Graph Decomposition Problem	17
3.1 Two-phase Approach	17
3.2 Four Heuristics for Phase 1	18
4. Hybrid Genetic Algorithm	19
4.1 Background	19
4.2 Genetic Algorithm for Graph Decomposition	20
4.3 Population and Chromosomal Representation	21
4.4 Reproduction Scheme and Elitism	22
4.5 Genetic Operators	23
4.6 Crossover and Mutation	24
4.7 Hybridization	25
4.7.1 Random Hybridization	26
4.7.2 Hybridization with Best among Random Regions	27
4.7.3 Hybridization with Best among Nearest Regions	28
4.8 Termination Criteria, Efficiency and Output	29

5. Iterative Improvement Heuristic	33
5.1 High-Level Performance	33
5.2 Step 1: Preparatory Step	34
5.3 Step 2: Remapping Vertices of ‘Type 3’	35
5.4 Step 3: Remapping Vertices of ‘Type 4’	38
5.5 Step 4: Remapping Vertices of ‘Type 3’	40
5.6 Proof of Termination	41
6. Improvements to the Hybrid Genetic Algorithm	47
6.1 Tuning the Hybrid Genetic Algorithm Results	47
6.2 Break up of Vertices Weights for the HGA Input Graph	50
7. Heuristic Algorithm for Finding Centers	53
7.1 Heuristic Algorithm	54
7.2 Numerical Example	57
8. Empirical Results	61
8.1 Experimental Environment	62
8.2 Hybrid versus Non-Hybrid GA	71
8.3 Three Types of GA	77
8.4 HGA versus HGA with Reduced Vertex Granularity	80
8.5 HGA versus Tuned HGA	88
8.6 IIH versus Tuned HGA	116
8.7 Finding Centers for Tuned HGA	128
9. Discussion of Empirical Results	140
10. Conclusions and Further Work	146
Bibliography	148

List of Figures

Figure 2.1 A weighted graph example	8
Figure 2.2 A weighted graph example	14
Figure 4.1 Genetic Algorithm Outline	21
Figure 4.2 Ranking Selection Scheme	24
Figure 4.3 Illustration of the Crossover Operation	25
Figure 4.4 Illustration of the Mutation Operation	26
Figure 4.5 Algorithm for Random Hybridization	28
Figure 4.6 Algorithm for Hybridization with Best Among Random Regions	29
Figure 4.7 Algorithm for Hybridization with Best Among Nearest Regions	31
Figure 5.1 An Iterative Improvement Heuristic Algorithm	34
Figure 5.2 Algorithm for Remapping Vertices of ‘Type 3’	37
Figure 5.3 Algorithm for Remapping Vertices of ‘Type 4’	39
Figure 5.4 Another Algorithm for Remapping Vertices of ‘Type 3’	41
Figure 6.1 Data Flow Diagram for Tuning up of HGA Results	49
Figure 6.2 Data Flow Diagram for Breaking up of Vertices Weights	51
Figure 6.3 Breaking up of Vertices Weights Algorithm	52
Figure 7.1 Heuristic Algorithm for Finding Centers	54
Figure 7.2 Numerical Example Graph	57
Figure 8.1 SRDSS Graph, $N = 14$	65
Figure 8.2 30-Vertex Graph, $N = 30$	67

Figure 8.3 50-Vertex Graph, $N = 50$	68
Figure 8.4 100-Vertex Graph, $N = 100$	70
Figure 8.5 (a) Results for SRDSS Graph, ($N = 14, p = 3$), Non Hybrid GA	72
Figure 8.5 (b) Results for SRDSS Graph, ($N = 14, p = 3$), Hybrid GA	72
Figure 8.6 (a) Results for the 30-Vertex Graph, ($N = 14, p = 3$), Non Hybrid GA ...	74
Figure 8.6 (b) Results for the 30-Vertex Graph, ($N = 14, p = 3$), Hybrid GA	74
Figure 8.7 'Random Hybridization GA for the 50-Vertex Graph, ($N = 50, p = 5$) ..	77
Figure 8.8 'Hybridization with Best Among Nearest Regions' GA for the 50-Vertex Graph, ($N = 50, p = 5$)	78
Figure 8.9 'Hybridization with Best Among Random Regions' GA for the 50-Vertex Graph, ($N = 50, p = 5$)	78
Figure 8.10 'Random Hybridization GA with Break up of Weights for the 50-Vertex Graph, ($N = 50, p = 5$)	80
Figure 8.11 'Hybridization with Best Among Nearest Regions' GA with Break up of Weights for the 50-Vertex Graph, ($N = 50, p = 5$) ...	81
Figure 8.12 'Hybridization with Best Among Random Regions' GA with Break up of Weights for the 50-Vertex Graph, ($N = 50, p = 5$) ...	81
Figure 8.13 (a) Results for SRDSS Graph, ($N = 14, p = 3$), HGA	91
Figure 8.13 (b) Results for SRDSS Graph, ($N = 14, p = 3$), Tuned HGA	91
Figure 8.14 (a) Results for 30-Vertex Graph, ($N = 30, p = 5$), HGA	93
Figure 8.14 (b) Results for 30-Vertex Graph, ($N = 30, p = 5$), Tuned HGA	93
Figure 8.15 (a) Results for 50-Vertex Graph, ($N = 50, p = 3$), HGA	95
Figure 8.15 (b) Results for 50-Vertex Graph, ($N = 50, p = 3$), Tuned HGA	95
Figure 8.16 (a) Results for 50-Vertex Graph, ($N = 50, p = 4$), HGA	97
Figure 8.16 (b) Results for 50-Vertex Graph, ($N = 50, p = 4$), Tuned HGA	97

Figure 8.17 (a) Results for 50-Vertex Graph, ($N = 50, p = 5$), HGA	99
Figure 8.17 (b) Results for 50-Vertex Graph, ($N = 50, p = 5$), Tuned HGA	99
Figure 8.18 (a) Results for 50-Vertex Graph, ($N = 50, p = 8$), HGA	101
Figure 8.18 (b) Results for 50-Vertex Graph, ($N = 50, p = 8$), Tuned HGA	101
Figure 8.19 (a) Results for 100-Vertex Graph, ($N = 100, p = 5$), HGA	104
Figure 8.19 (b) Results for 100-Vertex Graph, ($N = 100, p = 5$), Tuned HGA	105
Figure 8.20 (a) Results for 100-Vertex Graph, ($N = 100, p = 10$), HGA	107
Figure 8.20 (b) Results for 100-Vertex Graph, ($N = 100, p = 10$), Tuned HGA	108
Figure 8.21 (a) Results for 100-Vertex Graph, ($N = 100, p = 15$), HGA	110
Figure 8.21 (b) Results for 100-Vertex Graph, ($N = 100, p = 15$), Tuned HGA	111
Figure 8.22 (a) Results for 100-Vertex Graph, ($N = 100, p = 20$), HGA	113
Figure 8.22 (b) Results for 100-Vertex Graph, ($N = 100, p = 20$), Tuned HGA	114
Figure 8.23 Results of IIH for SRDSS Graph, ($N = 14, p = 3$)	118
Figure 8.24 Results of IIH for 30-Vertex Graph, ($N = 30, p = 5$)	119
Figure 8.25 Results of IIH for 50-Vertex Graph, ($N = 50, p = 3$)	120
Figure 8.26 Results of IIH for 50-Vertex Graph, ($N = 50, p = 4$)	121
Figure 8.27 Results of IIH for 50-Vertex Graph, ($N = 50, p = 5$)	121
Figure 8.28 Results of IIH for 50-Vertex Graph, ($N = 50, p = 8$)	122
Figure 8.29 Results of IIH for 100-Vertex Graph, ($N = 100, p = 5$)	124
Figure 8.30 Results of IIH for 100-Vertex Graph, ($N = 100, p = 10$)	125
Figure 8.31 Results of IIH for 100-Vertex Graph, ($N = 100, p = 15$)	126
Figure 8.32 Results of IIH for 100-Vertex Graph, ($N = 100, p = 20$)	127
Figure 8.33 Finding Centers for SRDSS Tuned HGA Graph, ($N = 14, p = 2$)	129
Figure 8.34 Finding Centers for SRDSS Tuned HGA Graph, ($N = 14, p = 3$)	129
Figure 8.35 Finding Centers for SRDSS Tuned HGA Graph, ($N = 14, p = 4$)	130

Figure 8.36 Finding Centers for 30-Vertex Tuned HGA Graph, ($N = 30, p = 5$) ...	131
Figure 8.37 Finding Centers for 30-Vertex Tuned HGA Graph, ($N = 30, p = 6$) ...	131
Figure 8.38 Finding Centers for 50-Vertex Tuned HGA Graph, ($N = 50, p = 3$) ...	132
Figure 8.39 Finding Centers for 50-Vertex Tuned HGA Graph, ($N = 50, p = 4$) ...	133
Figure 8.40 Finding Centers for 50-Vertex Tuned HGA Graph, ($N = 50, p = 5$) ...	133
Figure 8.41 Finding Centers for 50-Vertex Tuned HGA Graph, ($N = 50, p = 7$) ...	134
Figure 8.42 Finding Centers for 50-Vertex Tuned HGA Graph, ($N = 50, p = 8$) ...	134
Figure 8.43 Finding Centers for 100-Vertex Tuned HGA Graph, ($N = 100, p = 5$)	136
Figure 8.44 Finding Centers for 100-Vertex Tuned HGA Graph, ($N = 100, p = 10$)	137
Figure 8.45 Finding Centers for 100-Vertex Tuned HGA Graph, ($N = 100, p = 15$)	138
Figure 8.46 Finding Centers for 100-Vertex Tuned HGA Graph, ($N = 100, p = 20$)	139

List of Tables

Table 8.1 List of considered towns for the SRDSS test case	65
Table 8.2 List of Service Sites for the 30-Vertex Test Case	66
Table 8.3 List of Service Sites for the 50-Vertex Test Case	67, 68
Table 8.4 List of Service Sites for the 100-Vertex Test Case	69
Table 8.5 Results of Non Hybrid GA versus HGA	76
Table 8.6 Numerical Results of HGA versus HGA with Reduced Granularity for the 14-Vertex Test Case	83
Table 8.7 Numerical Results of HGA versus HGA with Reduced Granularity for the 30-Vertex Test Case	84
Table 8.8 Numerical Results of HGA versus HGA with Reduced Granularity for the 50-Vertex Test Case	85, 86
Table 8.9 Numerical Results of HGA versus HGA with Reduced Granularity for the 100-Vertex Test Case	87
Table 8.10 Results of HGA versus Tuned HGA	89
Table 8.11 Results of IIH algorithm versus Tuned HGA	116

Chapter 1

Introduction

1.1 Overview of the Regrouping Service Sites Problem

Organizations or institutions with multiple offices or service sites, which are geographically spread over many towns/cities/countries, may sometimes be interested in consolidating their resources by regrouping them. Regrouping refers to replacing a number of service sites with a smaller number of these existing sites (which are referred to as centers). The service sites can be company's branch offices, storage centers, medical centers, schools, warehouses, goods shops...etc.

The objectives of regrouping are usually to consolidate human resources, improve service quality, reduce the cost of services, and centralize company branches, in addition to other application-dependent objectives. In order to satisfy these objectives, we can set application-dependent criteria/constraints for choosing a center to replace or serve a group of neighboring service sites. One constraint is to locate a given number of service sites, called centers, so that the total travel distance between service sites and their corresponding center is minimized. Another constraint is to have minimum deviation of the weights of sites allocated to a center from their

average weight, where the weight associated to each site represents its service demand. So, not only the distance from service sites to their centers is considered, but also the weight demand favors the location of these centers. Selection of a suitable objective function to implement such constraints is the most important issue raised in the process of solving the regrouping service sites problem.

Regrouping large numbers of sites (in different districts, regions or countries) is an interesting and realistic problem. However, solving this problem, in order to satisfy some criteria (set of constraints C), is a complex task, which is extremely difficult to do manually and needs to be automated.

1.2 Previous Work

Related, but different, problems are facility location problems usually modeled as p -median or p -centers problems. Previous work has addressed these similar problems. A number of operational models has been developed to deal with facility location optimization (Love et al., 1988; Francis et al., 1992; Current et al., 1990). Most classical location studies focus on the minimization of the mean (or total) distance (the median concept) or the minimization of the maximum distance (the center concept) to the service facilities. Several approaches to location problems have been previously proposed such as a lexicographic minimax approach (Ogryczak W., 1997) and a distribution approach (Ogryczak W., 1999). In (Ogryczak W., 2000), solution concepts to location problems are based on the bi-criteria optimization of the

mean distance and the absolute inequality measures which comply with the concept of equitable efficiency.

Location problems can be classified into three groups: those locating central facilities, semi-desirable facilities and obnoxious facilities. This type of central facilities location problems is similar but not exactly the same as the regrouping service site problem presented in this work. The similarity is that 'Central facilities are those to which people must travel to receive the service, or from which a service is provided to the whole community of interest. It is generally felt that the closer (usually measured in terms of distance or time) the facilities are to the users, the better the service provided' (Rahman and Smith, 2000). On the other hand, the difference is that one of the most popular models for public facility location problems is the p -median problem (Hakimi, 1964). This p -median problem can be defined as: given discrete demands, locate a number (p or less) of facilities so that the total weighted travel distance or time between facilities and demand centers is minimized. What differs from the regrouping service sites problem is that the later requires exactly p centers and not p or less. What also differs is that not only distances are measured, but also weights associated with the service sites in the regrouping service sites problem are considered. Related concepts are shared between the two problems. 'The most important issue raised in the process of solving location problems is the selection of a suitable criterion or objective function' (Rahman and Smith, 2000). The smaller the total (or average) weighted travel distance (time), the more convenient for the users to get to the nearest facility. Solutions which minimize the weighted travel distance may be inequitable, forcing a few users to travel far.

Some work has been done on the regrouping service sites problem, where a prototype was developed for a decision support system (Mansour and Rishani, 1998). However, it fails to guarantee that the requested number of centers will be exactly satisfied. The application of the modified p -centers algorithm yields a number of centers that may be equal or close to p . That is, the solution is not guaranteed to include exactly p centers as required by the problem specification.

The above work was extended in (Dana, 1998), where a 2-phase Genetic-based method that meets the problem specification and finds exactly p centers was proposed. The method is based on the weighted graph model and on solving the problem in two phases: decomposing the graph model into p (required number of centers) sub-graphs (the graph decomposition problem) using a Genetic Algorithm (GA), followed by finding a suitable center within each sub-graph using a heuristic algorithm. Such an exact solution maintains the flexibility offered by the graph problem formulation and provides more accurate solutions by finding the exact number of centers.

1.3 Scope and Purpose

The purpose of the regrouping service sites problem is to assign a central service site to a group of sites which they will seek to receive the service, such that some objective function can be maximized or minimized subject to some constraints. Finding optimal central sites for the regrouping service sites is our main objective.

In this thesis, an improved genetic algorithm, called Tuned HGA, for the graph decomposition problem is developed and tested. Our genetic algorithm differs from the existing implementation in three ways. First, we use hybridization and present some Hybrid Genetic Algorithms (HGA). Second, we add a pre-processing step to reduce the graph vertex granularity. Third, we tune the results of the HGA by applying a heuristic algorithm called Iterative Improvement Heuristic (IIH) to them. The proposed HGA is compared to the existing GA used in (Dana, 1998), and to the HGA with reduced vertex granularity. The improved Tuned Hybrid Genetic Algorithm (Tuned HGA) is compared to the HGA, and to the heuristic algorithm IIH when applied to random input.

We start our research by extending the GA presented in (Dana, 1998) in order to get better quality solutions. This is done through presenting three different types of Hybrid GA's with comparable results: 'Random Hybridization', 'Hybridization with Best among Random Regions', and 'Hybridization with Best among Nearest Regions'.

Then, we present a pre-processing step that breaks up weights of the input service sites. This is done in order to improve the results of the above Hybrid GA's.

Moreover, in order to tune the results of the Hybrid GA's (HGA's), we present an Iterative Improvement Heuristic (IIH) algorithm. This algorithm is applied to the output produced by the HGA to improve its results. Other than the HGA results, the IIH algorithm can be applied on a randomly generated input.

We implement the operation of our solution algorithms and apply them to several test cases to produce comparable experimental results that show high quality improvements.

This thesis is organized as follows: Chapter 2 describes the regrouping service sites problem and its objective function. In this chapter, the problem initial graph formulation is explained in details along with its graph decomposing method. The objective function used by the HGA is stated and explained through an example. Chapter 3 presents an overview of our solution to the graph decomposition problem. In Chapter 4, the Hybrid Genetic Algorithm used for Graph Decomposition is presented, where the three types of hybridization are proposed. Chapter 5 includes the Iterative Improvement Heuristic algorithm and its proof of termination. Two improvements to HGA are presented in Chapter 6. The first section includes the improvement (called Tuned HGA), which is done by applying the IIH algorithm to the output produced by HGA in order to tune its results. The second section explains the pre-processing step (called Breaking up of Vertex Weights for the HGA Input Graph) that re-builds the input graph given to the HGA by breaking up the weights of its vertices. In Chapter 7, the heuristic procedure for finding centers is discussed. Chapter 8 includes our experimental results, followed by a discussion of these results in Chapter 9. Chapter 10 contains the conclusion and our insights for further research work.

Chapter 2

Problem Description and Objective Function

2.1 Preface

In this thesis, we describe a method for solving the regrouping service sites problem described in the previous chapter.

The method is based on a weighted graph representation of the problem as shown in Figure 2.1 below, and is divided into a preparation stage and followed by two different phases. In the preparation stage, a distance matrix of the weighted graph may be produced using Equation 2.1. From this, a so-called minimum distance matrix is derived using Dijkstra's algorithm (Wirth, 1976). Moreover, we calculate a service sites weights vector using Equation 2.2. Afterwards, in Phase 1, a Genetic Algorithm that uses the complete minimum distances matrix and the vertices weight vector decomposes the weighted graph into p sub-graphs (or regions), where p is the required number of desired centers to be found. Each region will have a minimal total sum of inner distances, and the total weight of the graph will be uniformly distributed among all regions. Finally, in Phase 2, a heuristic algorithm is applied in order to find one center for each region.

Figure 2.1 A weighted graph example

In Section 2.2, we present the problem description. In Section 2.3, we present the formulation of the objective function of Phase 1, followed by a related numerical example in Section 2.4.

2.2 Problem Description

The sites regrouping problem can be stated as follows: given N sites, let p ($p < N$) be the number of sites to be selected as centers into which the other sites can be regrouped (merged). Also, let C be the set of user-defined constraints. The problem can thus be expressed in the following form: find p centers, to replace or serve N sites, such that the distance from the location of a center to a ‘maximum’ number of nearby service sites is minimized, subject to C .

Let $G = \langle V, E \rangle$ be a weighted graph, representing a user-defined region, where each $v_i \in V$ represents a service site and $|V| = N$. We let $(v_i, v_j) \in E$ denote an edge of E with the weight e_{ij} representing the geographical distance between vertices v_i and v_j . This edge e_{ij} measures the direct connection distance between v_i and v_j . Vertices of the graph may be mapped to a grid of X and Y axis, as we have done in the testing phase, where v_i and v_j will be of coordinates (X_i, Y_i) and (X_j, Y_j) respectively, and e_{ij} may be calculated using the following equation:

$$e_{ij} = \sqrt{(X_j - X_i)^2 + (Y_j - Y_i)^2} \quad (2.1)$$

In addition, we use W to be a set of weights such that each w_i belongs to W is the weight associated with the vertex v_i . A vertex weight is computed as a weighted sum of some user defined factors. Associated with each factor, a weight percentage is allocated to it in order to reflect its importance relative to the other factors. The vertex weight of w_i is given by the following equation:

$$w_i = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3 + \dots + \alpha_N P_N \quad (2.2)$$

The value of a coefficient α_i is used to give a higher weight to the better values of the factor i , where $i = 1, \dots, N$. Guidelines for determining these values can be proposed and are application-dependent.

A solution to the regrouping service site problem is represented by a set of vertices $V'' \subset V$ where $|V''|$ would be equal to the desired (required) number of centers p . Moreover, for each center $v_i \in V''$, there is a set of vertices $V_i' \subset V$ associated to it, such that v_i will replace or serve its related set V_i' .

2.3 Objective Function for Graph Decomposition

Our goal is to decompose the graph into p regions (sub-graphs). Two factors are to be considered here: the total weight of a single region services sites and its total inner distance. The weight of a region (sub-graph) R is given by the following equation:

$$W_R = |D_R| + \mu / 10 * p * (N / \text{Card}_R) * C_R \quad (2.3)$$

where p is the number of service centers and N is the number of service sites.

D_R represents the deviation of the sub-graph's vertices weight from the average of all weights of all sub-graphs; that is

$$D_R = \left(\sum_{v_i \in R} w_i \right) - \left(\sum_{v_i \in G} w_i / p \right)$$

C_R represents the total sum of the inner distances within a sub-graph; that is

$$C_R = \left(\sum_{v_i, v_j \in R} e_{ij} \right)$$

And Card_R represents the number of vertices in the sub-graph R .

μ is a scaling coefficient needed due to the difference in the units of the values of D_R and C_R , and μ is calculated according to the following equation:

$$\mu = \left(\left| \sum_{v_i \in G} w_i - \left(\sum_{v_i \in G} w_i / N \right) \right| \right) / \left(\sum_{v_i, v_j \in R} e_{ij} \right) \quad (2.4)$$

In an ideal situation, weights of all regions are equal. Moreover, each region must have a minimal total sum of inner distances. In practice, we need to minimize

D_R and C_R for all regions in order to be closest to the ideal situation. Consequently, we minimize the maximum W_R for all regions. Thus, the objective function to be minimized in Phase 1 may be

$$\phi_1 = \text{MAX}_{\text{all } R}(W_R) \quad (2.5)$$

An alternative objective function can be to minimize the average of W_R .

$$\phi_2 = \sum_{\text{all } R} W_R / p \quad (2.6)$$

We use the difference in the objective function, ΔOF , for the purpose of improving the algorithm efficiency. The ΔOF resulting from remapping site s from region l to region k , is calculated efficiently to enhance the overall performance of the Genetic Algorithm (since such a remapping will be done extensively during the hybridization operation) and the Iterative Improvement Heuristic. When we remap a site s from region l to region k , ΔOF is calculated as follows:

$$\begin{aligned} \Delta OF &= OF_{\text{new}} - OF_{\text{old}} \\ &= \sum_{\text{all } R} W_{R \text{ new}} / p - \sum_{\text{all } R} W_{R \text{ old}} / p \\ &= 1/p [(W_{R k \text{ new}} - W_{R k \text{ old}}) + (W_{R l \text{ new}} - W_{R l \text{ old}})] \\ &= [(W_{R k \text{ new}} - W_{R k \text{ old}})] = |D_{R k \text{ new}}| + \mu/10 * p * (N / \text{Card}_{R k \text{ new}}) * C_{R k \text{ new}} - \\ &\quad |D_{R k \text{ old}}| + \mu/10 * p * (N / \text{Card}_{R k \text{ old}}) * C_{R k \text{ old}} \end{aligned}$$

$$\text{where } |D_{R k}| = |(\sum_{v_i \in R k} w_i) - (\sum_{v_i \in G} w_i / p)|$$

$$\text{and } C_{Rk} = \left(\sum_{v_i, v_j \in Rk} e_{ij} \right)$$

Similarly, $[(W_{Rl \text{ new}} - W_{Rl \text{ old}})]$ can be calculated as above.

Hence,

$$\Delta OF = 1/p (|D_{Rk \text{ new}}| - |D_{Rk \text{ old}}| + |D_{Rl \text{ new}}| - |D_{Rl \text{ old}}|) + \mu/10 * N * \\ (C_{Rk \text{ new}} / \text{Card}_{Rk \text{ new}} - C_{Rk \text{ old}} / \text{Card}_{Rk \text{ old}} + C_{Rl \text{ new}} / \text{Card}_{Rl \text{ new}} - C_{Rl \text{ old}} / \text{Card}_{Rl \text{ old}})$$

Using ΔOF produces more efficient results, rather than calculating two objective functions, one before remapping a site from one region to another and the other after the remapping takes place. When a site s is mapped from region l to a new region k , only updates to the weights factors D_R 's, and the edges factors C_R 's are carried out according to the following algorithm:

Calculate all D_R 's for region = 1, ..., p

Calculate all C_R 's for region = 1, ..., p

Calculate all $Card_R$'s for region = 1, ..., p

Loop for sites $s = s_1, s_2, \dots, s_x$

Move s from region l to region k

Set the values of $C_{Rk\text{old}}$ and $C_{Rl\text{old}}$ from C_R 's

Set the values of $D_{Rk\text{old}}$ and $D_{Rl\text{old}}$ from D_R 's

Set the values of $Card_{Rk\text{old}}$ and $Card_{Rl\text{old}}$ from $Card_R$'s

$$C_{Rk\text{new}} = C_{Rk\text{old}} + \left(\sum_{i \in Rk} e_{is} \right) \quad \{\text{edges from site } s \text{ to every site } i \text{ in region } k\}$$

$$C_{Rl\text{new}} = C_{Rl\text{old}} - \left(\sum_{i \in Rl} e_{is} \right)$$

$$D_{Rk\text{new}} = D_{Rk\text{old}} + w_s \quad \{\text{weight of site } s\}$$

$$D_{Rl\text{new}} = D_{Rl\text{old}} - w_s$$

$$Card_{Rk\text{new}} = Card_{Rk\text{old}} + 1$$

$$Card_{Rl\text{new}} = Card_{Rl\text{old}} - 1$$

$$\Delta OF = 1/p (|D_{Rk\text{new}}| - |D_{Rk\text{old}}| + |D_{Rl\text{new}}| - |D_{Rl\text{old}}|) + \mu/10 * N *$$

$$(C_{Rk\text{new}}/Card_{Rk\text{new}} - C_{Rk\text{old}}/Card_{Rk\text{old}} + C_{Rl\text{new}}/Card_{Rl\text{new}} - C_{Rl\text{old}}/Card_{Rl\text{old}})$$

If ($\Delta OF < 0$), then

Update C_R 's, D_R 's, and $Card_R$'s for regions k and l

End Loop

2.4 Numerical Example

In this section, we present a numerical example to illustrate the calculation of a region objective function. We will consider the following graph:

Figure 2.2 A weighted graph example

We let $p = 2$ represent the number of required sites, and $N = 4$ represent the total number of sites.

The following table is the minimum distance matrix of the distances between the vertices numbered 1, 2, 3 and 4 of the weighted graph shown in Figure 2.2.

Minimum Distance Matrix

	1	2	3	4
1	0	200	150	427.2
2	200	0	250	250
3	150	250	0	400
4	427.2	250	400	0

The weight vector of the service sites (vertices) of the graph in Figure 2.2. is :

1	5
2	45
3	17
4	23

$$D_R = \left(\sum_{v_i \in R} w_i \right) - \left(\sum_{v_i \in G} w_i / p \right)$$

$$\sum_{v_i \in G} w_i / p = (5 + 45 + 17 + 23) / 2 = 45$$

$$C_R = \left(\sum_{v_i, v_j \in R} e_{ij} \right)$$

$$\sum_{v_i, v_j \in R} e_{ij} = 200 + 150 + 250 + 427.2 + 250 + 400 = 1677.2$$

$$\left| \sum_{v_i \in G} w_i - \left(\sum_{v_i \in G} w_i / N \right) \right| = \left| (5 + 45 + 17 + 23) - ((5 + 45 + 17 + 23) / 4) \right| = 67.5$$

Therefore, μ , using equation (2.4), is 0.04.

We randomly partition the graph into 2 regions (sub-graphs).

Region 1 (R1) contains service sites 1 and 4.

Region 2 (R2) contains service sites 2 and 3.

Using equation (2.3),

$$D_{R1} = (W_1 + W_2) - 45 = (5 + 23) - 45 = 17$$

$$C_{R1} = (e_{14}) = 427.2 / \text{Card}(R1) = 427.2 / 2 = 213.6$$

$$W_{R1} = 17 + 0.04 / 10 * 2 * 4/2 * 213.6 = 17 + 3.4176 = 20.4176$$

Similarly,

$$D_{R2} = (W_2 + W_3) - 45 = (45 + 17) - 45 = 17$$

$$C_{R2} = (e_{23}) = 250 / \text{Card}(R2) = 250 / 2 = 125$$

$$W_{R2} = 17 + 0.04 / 10 * 2 * 4/2 * 125 = 17 + 2 = 19$$

Therefore, the objective function to be minimized in Phase 1 is

$$\varphi_1 = \underset{\text{all } R}{\text{MAX}}(W_R) = 20.4176$$

or

$$\varphi_2 = \sum_{\text{all } R} W_R / p = 19.7088$$

Chapter 3

Overview of our Solution for the Graph Decomposition Problem

3.1 Two-phase Approach

Our approach for solving the regrouping service sites problem is a two-phase approach solution. The first phase determines p sub-graphs (regions) of the weighted graph $G = \langle V, E \rangle$ using the HGA, where p is the number of required centers. Each sub-graph is represented by a weighted graph $G' = \langle V', E' \rangle$, where each $v_i \in V'$ is a service site, and for each edge $(v_i, v_j) \in E'$, e_{ij} represents the geographical distance between vertices v_i and v_j , $V' \subset V$ and $E' \subset E$. Moreover, we let w_i denote the weight associated with each vertex v_i . The second phase determines the center for each region, that is the vertex that will replace or serve its region's set of vertices V' . The set of vertices $V'' \subset V$, where $v_i \in V''$ is a center for its region i and $|V''| = p$, will be the solution of the regrouping service sites problem.

3.2 Four Heuristics for Phase 1

In the first phase of our approach which determines the regions that decompose the weighted graph $G = \langle V, E \rangle$, there are four heuristics suggested and tested to produce comparable results. The following are the four heuristic algorithms:

1. *Hybrid Genetic Algorithm (HGA)* discussed in Chapter 4,
2. *Iterative Improvement Heuristic Algorithm (IIH)* illustrated in Chapter 5,
3. *Hybrid Genetic Algorithm with Tuning up results using IIH* explained in Section 6.1, and
4. *Hybrid Genetic Algorithm with Breaking up vertices weights* explained in Section 6.2.

The empirical results of these algorithms are compared, and the best among these heuristics (Tuned HGA) outputs the intermediate solution, which determines the sub-graphs of the initial graph G .

Chapter 4

Hybrid Genetic Algorithm

4.1 Background

Genetic Algorithms are based on the mechanics of natural evolution (Goldberg, 1989). They derive their behavior from a metaphor of some of the mechanisms of evolution in nature.

A genetic search uses the mechanics of natural selection and natural genetics to evolve a population of initial solutions into a near-optimal solution (Chambers, Vol. I, 1995; Chambers, Vol. II, 1995).

Genetic Algorithms work on a population of possible solutions. These solutions are called individuals or chromosomes. The individuals in the population then go through a process of simulated 'evolution'.

The initial generation operates on a population of randomly generated individuals. From there on, the genetic operations: reproduction, crossover, mutation, and hybridization, in concert with the fitness measure, operate to improve the population by creating a new offspring population from the current population. Therefore, iteration of generations is created in a successive progression towards increasing uniformity until 95% of the population share the same value.

The reproduction operation involves selecting according to fitness, an individual from the current population of individuals and allowing it to survive by copying it into the new population. Then two mates are randomly selected and the genetic operators crossover, mutation and hybridization are applied to create new offspring individuals, which replace the parents, and the process is repeated for many generations.

Typically, the best individual that appeared in any generation of a run (i.e. best-so-far individual) is designated as the result produced by the Genetic Algorithm. In the next sub-section, we present a Genetic Algorithm for decomposing a graph into p sub-graphs such that an objective function (Equation 2.6) is minimized.

Genetic algorithms have been used for graph decomposition related problems, such as graph partitioning. They have also been applied to engineering problems such as mapping data to multicomputers (Mansour and Fox, 1994) and VLSI placement problems. Some other applications in which GAs have been used include the dynamic layout problem (Conway and Venkataramanan, 1994; Balakrishnan and Cheng, 2000), quadratic assignment problem (Tate and Smith, 1995), job shop scheduling (Davis, 1985), and constrained optimization (Homaifar, 1994).

4.2 Genetic Algorithm for Graph Decomposition

In this section, we describe how Genetic Algorithms can be adapted for solving the regrouping service sites problem. We present an outline of the algorithm in Figure 4.1, and its components are described in the subsequent sections.

```

Random generation of initial population size POP;
Evaluate fitness of individuals;
Repeat
    Preserve_the_Fittest (elitism);
    Rank individuals and allocate reproduction trials;
    For  $i := 1$  to POP incrementing  $i$  by 2
        Randomly select two parents from list of reproduction trials;
        Apply crossover and mutation
        Apply hybridization
    End for
    Evaluate fitness of offspring
    Preserve_the_Fittest (elitism);
Until convergence
Solution = Fittest.
  
```

Figure 4.1 Genetic Algorithm Outline

The input of the Genetic Algorithm consists of the number of service sites and their corresponding names, the minimum distance matrix and the weight vector. The desired number of centers to be found must also be specified.

The Genetic Algorithm operates in order to decompose the graph of service sites into sub-graphs (regions) such that the objective function ϕ_2 (Equation 2.6) is minimized. That is, the Genetic Algorithm minimizes each region's weight. Moreover, each region should have a minimum total sum of inner distances.

4.3 Population and Chromosomal Representation

In applying Genetic Algorithms to the regrouping service sites problem, chromosomes (individuals) represent a possible grouping solution. The population of chromosomes is represented by 'Init-pop', a two-dimensional array of integers. An individual in the population is encoded as N-element vector $[X_1, X_2, X_3, \dots, X_N]$, where X_i 's are called genes, and each individual corresponds to a candidate solution to the regrouping service sites problem (i.e. a set of region distribution). The value of each element i represents a service site number. The value assigned to each gene is an integer number representing the region to which a service site is allocated. For example, if we have 5 service sites which must be mapped to 2 regions, a possible solution could be to assign service sites 1,2 and 4 to region 1 and service sites 3 and 5 to region 2, then the encoded chromosome will be as follows (1,1,2,1,2).

The size of the population is chosen as a multiple of $2 * N$, N being the number of service sites. This will guarantee a complete crossover operation. In our implementation, we make the size of the population to be equal to $2 * N * p$. The initial generation is created at random.

4.4 Reproduction Scheme and Elitism

In the Genetic Algorithm, the whole population is considered a single reproduction unit within which random selection is performed. The fitness function to be maximized is $1/\phi_2$. Our reproduction scheme involves elitism ranking, followed by random selection of mates from the list of reproduction trials assigned to ranked individuals.

In the ranking scheme (Baker, 1985), the individuals are sorted by fitness values. After sorting, each individual is assigned a rank based on a scale of equidistant values of the population. The ranks assigned to the fittest and least fit individuals are 1.2 and 0.8 respectively. Individuals with ranks greater than 1 are first assigned single copies, then the fractional part of their ranks and the ranks of the lower half of individuals are treated as possibilities for assignment of copies. Figure 4.2 illustrates the ranking selection in a population of four individuals. It has been found that the above ranking based selection with maximum rank of 1.2 produces a survival percentage of 92% to 98% in different generations.

Elitism is used to maintain population diversity. Presentation of the fittest individual is done by replacing the least-fit individual by the new fittest if it is better than the current fittest; else if it is worse, we replace the worst individual by the current fittest. This will help preserving our current fittest into the next generation.

1	2	3	4	5	6	7	8	Individual	Fitness	Rank	Copies
1	2	1	2	1	3	2	3	0	3	1.07	1
3	1	2	1	2	3	2	3	1	5	0.93	1
1	2	3	1	3	1	3	2	2	1	1.2	2
1	2	3	1	2	2	3	1	3	6	0.8	0

8 service sites mapped to 3 regions.

Result: Duplication of chromosome 0 and 1 by one copy,
chromosome 2 by two copies.

Figure 4.2 Ranking Selection Scheme

4.5 Genetic Operators

The genetic operators used in our Genetic Algorithm are crossover, mutation and hybridization. Crossover is carried out before mutation. Then, hybridization is applied on the resulting individuals. The probability of crossover is considered to be 0.7 and the mutation rate is set to be 0.3 (Grefenstette, 1986). Crossover is done at a

higher rate than mutation and this is because crossover plays an important role in evolution; however, mutation is usually used to tune the solution. Hybridization is applied on each individual. It works to improve the individuals, depending on the value of the objective function. The following sections give description of each genetic operator.

4.6 Crossover and Mutation

In our Genetic Algorithm, two-endpoint crossover is replaced by multi-point crossover which is applied on a pair of individuals. The number of genes to be swapped is randomly selected between 1 and N. Then, genes are selected randomly, and their swapping between the two individuals is carried out. An illustration of this crossover operation is shown in Figure 4.3.

Figure 4.3 Illustration of the Crossover Operation

Mutation is carried out on a single chromosome by selecting randomly one position and changing its content. The new value of the gene is a randomly selected region number. Figure 4.4 illustrates the mutation operation. A special mechanism is applied to insure that the crossover and mutation operations will not lead to an unfeasible chromosome.

Figure 4.4 Illustration of the Mutation Operation

4.7 Hybridization

The hybridization operation is applied on each chromosome after crossover and mutation. The value of each gene from 1 to N is changed to be another randomly selected region number, while keeping a copy of its original value. Then, the objective function is checked for improvement of the quality of the individual. If the objective function is better, the new value of the gene is accepted. Else, its old value

is restored. In other words, each site moves from one region to a new region in case the chromosome yields a better objective function. The new region of the site is selected according to the hybridization type. Three types of hybridization are implemented in our work, and their experimental results compared. The following sections will describe how hybridization is applied in each case.

4.7.1 Random Hybridization

In the random hybridization, the value of each gene is changed to a randomly selected region number. Each time a gene is considered, the random generator produces a random number r between 1 and p , which is the number of regions. If the objective function is improved, r will be the new value of the gene; in other words, the new region of the site under consideration. Else, the gene will maintain its original value. Figure 4.5 presents the algorithm for applying random hybridization on a chromosome (individual) of size N which is the number of genes (sites).

```

For  $i := 1$  to  $N$       {go thru each gene of the chromosome}

    Save the original value of the site (gene)  $i$  in a variable  $temp$ 

    Set the value of site  $i$  to a random region number between 1 and  $p$ 

    Calculate  $\Delta OF$ 

    If ( $\Delta OF < 0$ ) then

        Let site  $i$  maintain its new value, and

        Update accordingly variables for calculating  $\Delta OF$ 

    Else

        Restore the original region number  $temp$  of site  $i$ 

End Loop

```

Figure 4.5 Algorithm for Random Hybridization

4.7.2 Hybridization with Best among Random Regions

This type of hybridization is similar to random hybridization in the way the new region of the site is selected, i.e. randomly. What differs is that not only one random region number r is considered to be the new value of the gene, but x number of random regions $r_1, r_2, \dots, r_x$ are substituted in turn to be the gene value, and the best among them; i.e., the one that produces the best objective function is selected in case this change improves the original chromosome. x is set to be equal to the ceiling of half of the number of regions p . The corresponding algorithm is shown in Figure 4.6 below.

```

 $x := \text{ceiling} ( p / 2 )$     {  $p$  is the number of regions}

For  $i := 1$  to  $N$            {go thru each gene of the chromosome}

    Save the original value of the site (gene)  $i$  in a variable  $temp$ 

     $Min\_ \Delta OF := 0$  {initializing the minimum  $\Delta OF$  to compare with}

    For  $j := 1$  to  $x$ 

        Set the value of site  $i$  to a random region number

        Calculate  $\Delta OF$ 

        If  $( \Delta OF < Min\_ \Delta OF )$  then

            Let site  $i$  maintain its new value, and

            Update accordingly variables for calculating  $\Delta OF$  ,

             $Min\_ \Delta OF := \Delta OF$     {new value of  $Min\_ \Delta OF$  }

        Else

            Restore the original region number  $temp$  of site  $i$ 

    End Loop {For  $j$ }

End Loop {For  $i$ }

```

Figure 4.6 Algorithm for Hybridization with Best among Random Regions

4.7.3 Hybridization with Best among Nearest Regions

In this case, new region numbers of the sites (genes) are not produced randomly, but nearest regions to the sites are considered. The region that contains the nearer site to the site under consideration is the nearer region. As in the previous type

of hybridization, x regions $r_1, r_2, \dots, r_x$ take in turn the value of the gene, where x is equal to the ceiling of half of the number of regions p . The best among $r_1, r_2, \dots$, and r_x , i.e. the region that improves the objective function the most, is the one which is accepted and replaces the old region; this is done in case the new region yields a better chromosome. Else, the old region of the site is maintained. In this type of hybridization, the regions $r_1, r_2, \dots, r_x$ are not generated as random numbers, but rather the result of a function that outputs the nearest x regions in their sequential order, from the 1st nearest region to the x^{th} nearest region, is produced. The algorithm is presented in Figure 4.7.

```

 $x := \text{ceiling} ( p / 2 )$     {  $p$  is the number of regions}

For  $i := 1$  to  $N$            {go thru each gene of the chromosome}

    Get the nearest  $x$  regions, and

    Store the region numbers in an array  $x\_regions$  with entries from 1 to  $x$ 

    Save the original value of the site (gene)  $i$  in a variable  $temp$ 

     $Min\_ \Delta OF := 0$     {initializing the minimum  $\Delta OF$  to compare with}

    For  $j := 1$  to  $x$ 

        Set the value of site  $i$  to region number in  $x\_regions [j]$ 

        Calculate  $\Delta OF$ 

        If  $( \Delta OF < Min\_ \Delta OF )$  then

            Let site  $i$  maintain its new value, and

            Update accordingly variables for calculating  $\Delta OF$  ,

             $Min\_ \Delta OF := \Delta OF$     {new value of  $Min\_ \Delta OF$  }

        Else

            Restore the original region number  $temp$  of site  $i$ 

    End Loop {For  $j$ }

End Loop {For  $i$ }

```

Figure 4.7 Algorithm for Hybridization with Best among Nearest Regions

4.8 Termination Criterion, Efficiency and Output

The Genetic Algorithm terminates when the best-so-far string does not change its fitness value for 50 consecutive generations. The complexity of the algorithm is equal to the product of the maximum number of generations multiplied by the cube of the population size.

Once converged, our service sites are mapped to the desired number of regions. Each region is represented as a graph, then a heuristic algorithm is used to find the required center inside each region.

Chapter 5

Iterative Improvement Heuristic

5.1 High-Level Description

In this chapter, we present an Iterative Improvement Heuristic (IIH) algorithm that can be applied as a tuning step to the output of the Genetic Algorithm in order to improve its results. This heuristic algorithm also works on random input that is not produced by the Genetic Algorithm, i.e. a graph with sites that have randomly allocated region numbers. There are three main steps of our IIH algorithm after which a better output result is produced. These steps are illustrated in the following sections. The IIH algorithm is presented in Figure 5.1.

- Use the output of the GA or generate randomly a mapping of sites (vertices) to regions (sub-graphs).
- **Step 1:** Visit all vertices in the sub-graphs and mark them as of:
 - ‘Type 1’: ‘Inner’ points,
 - ‘Type 2’: ‘Boundary’ points,
 - ‘Type 3’: ‘Misplaced within one region’ points, or
 - ‘Type 4’: ‘Misplaced within more than one region’ points
- **Step 2:** Remap all vertices of ‘Type 3’
- **Step 3:** Remap all vertices of ‘Type 4’
- **Step 4:** If some vertices become of ‘Type 3’ due to the previous step, Remap these vertices.

Figure 5.1 An Iterative Improvement Heuristic Algorithm

5.2 Step 1: Preparatory Step

The Iterative Improvement Heuristic algorithm goes through all vertices of the graph and determine their types. There are four types of vertices (points) to examine:

- ‘Type 1’: ‘Inner’ points,
- ‘Type 2’: ‘Boundary’ points,
- ‘Type 3’: ‘Misplaced within one region’ points, and
- ‘Type 4’: ‘Misplaced within more than one region’ points.

We mark each region by a certain color, so the words ‘region’ and ‘color’ are used interchangeably. An ‘inner’ point is a site which is allocated a region number, and all its neighboring sites are mapped to the same region number, i.e. a site with color i lying within the same color i . A ‘boundary’ point is a site that has a region number which it shares with one or more of its neighboring sites, while other neighboring sites are mapped to different region number(s). In other words, it is a vertex of color i at the boundary of region i but also neighboring one or more colors. A ‘misplaced within one region’ point is a site which is allocated region number i , and all its neighboring sites are mapped to a different region j . This is a vertex of color i lying within color j . A ‘misplaced within more than one region’ point is a site which has region number i , and none of its neighboring sites shares the same region number i , and at least 2 neighboring sites are mapped to 2 different regions.

5.3 Step 2: Remapping Vertices of ‘Type 3’

In this step, all vertices of ‘Type 3’, which are misplaced within one region, are visited successively. A vertex v of color (region) i , that is misplaced within region j , is remapped to region j . Then, v attempts to swap regions with another vertex of color (region) j , so that the objective function is improved. More than one vertex of color j may exist, but there are some more conditions to select the vertex w with which v may swap regions. Each such vertex w must be of ‘Type 2’, ‘Type 3’ or ‘Type 4’. Also, w must have at least one neighbor u of color i , and u must not be of ‘Type 3’ or ‘Type 4’. The condition for w to have such a neighbor u is because we want to try to swap regions only with vertices that may improve the objective

function. If u is remapped to region i , u already has at least one neighbor of color i , and this neighbor is not misplaced; i.e., it has another neighbor of color i . This insures the fact that we are trying to re-map w , of region j , to region i because w is lying within a neighborhood of region i . If such a vertex w exists, or more than one vertex, $w_1, \dots, w_x$, that have the same case as w , exist, we try to color this/these $w(s)$ with color i , and calculate ΔOF for each trial. The vertex w , that yields the minimum negative ΔOF , is selected to be remapped to region i ; i.e., vertex w swaps its region with vertex v .

Iterations over vertices of the graph continue until no more ‘Type 3’ vertices are found. The algorithm for this step is shown in Figure 5.2.

Find vertex v of 'Type 3' {i.e. vertex of color i lying within color j }

Loop until no more 'Type 3' vertices are found

- Save OF into OF_1 and Save color i
- Color this vertex v with color j , and update OF into OF_2
- Go through all 'Boundary' points (i.e. of 'Type 2') of color j , and 'Misplaced' points (i.e. of 'Type 3' or 'Type 4') of color j , that have neighbor(s) of color i which is/are not of 'Type 3' or 'Type 4'.
(don't carry on if these types of points are not found).
- If points of the previous conditions are found, attempt to color them with i , and find the vertex w that yields largest negative ΔOF , and color it with i (i.e. swap this vertex w with v)
- Save OF into OF_3
- If $(OF_3 < OF_2)$ and $(OF_3 < OF_1)$, then
accept the swap of the vertex w with vertex v
- Re-mark all vertices, and find if there is more vertex v of 'Type 3'

End Loop

Figure 5.2 An Algorithm for Remapping Vertices of 'Type 3'

5.4 Step 3: Remapping Vertices of ‘Type 4’

At this stage, we must not have vertices of ‘Type 3’, so we do not check for that type anymore. Here, we want to locate vertices of ‘Type 4’ that are misplaced within more than one region. We visit each vertex v of this type, and remap it to a new region. The steps followed to remap ‘Type 4’ points are similar to the steps of the algorithm presented in the previous section that remaps points of ‘Type 3’. The difference is in the way vertex v is remapped. We color vertex v with the best of the neighboring colors by checking the objective function. After this iterative procedure for remapping vertices of ‘Type 4’, there is no more vertices of this type. Figure 5.3 presents the steps of this algorithm.

We show below, through an example, that the algorithm presented above may result in changing the type of a/some point/points to become of ‘Type 3’.

Consider a vertex v of ‘Type 4’, whose color (region) is i . And, v has different neighbors of different colors, one of which it remaps to, say color (region) j . As the IIH algorithm depicts, it attempts to swap the contents of vertex v with another vertex, say w , of color j . Consider the case where we have improvement of the objective function, so swapping takes place, and vertex w of color j is remapped to color i . Assume w has a neighbor u of color k , and u is of ‘Type 4’. u has all its neighbors of color i , except neighbor w . w used to be of color j with respect to u , but when the swap occurred, w has been remapped to color i . Hence, all neighbors of u are now of color i . And, u becomes of ‘Type 3’ instead of ‘Type 4’.

Hence, a/some vertex/vertices may become of 'Type 3' while remapping another/other 'Type 4' vertex/vertices. Therefore, we have added to the IHH a step, discussed in the following section, to remap this/these points again.

Find vertex v of 'Type 4' {i.e. v of color i lying within colors $j_1, j_2, \dots$ }

Loop until no more 'Type 4' vertices are found

- Save OF into OF_1 and Save color i
- Color this vertex v with the best of the neighboring colors (i.e. color j), and update OF into OF_2
- Go through all 'Boundary' points (i.e. of Type 2) of color j or 'Misplaced' points (i.e. of Type 4) of color j , that have neighbor(s) of color i that is/are not of 'Type 3' or 'Type 4'.
(don't carry on if these types of points are not found)
- **If** points of the previous conditions are found, attempt to color them with i , and find the vertex w that yields largest negative ΔOF , and color it with i (i.e. swap this vertex w with v)
- Save OF into OF_3
- **If** ($OF_3 < OF_2$) and ($OF_3 < OF_1$), then
accept the swap of the vertex w with vertex v
- Re-mark all vertices, and find if there is more vertex v of 'Type 4'

End Loop

Figure 5.3 Algorithm for Remapping Vertices of 'Type 4'

5.5 Step 4: Remapping Vertices of ‘Type 3’

In this step, we remap the vertices of ‘Type 3’ introduced in the previous step (if any). This remapping is done in a way similar to the one presented in Step 2; however, no swapping is performed in order not to introduce ‘Type 4’. We show below that swapping may introduce ‘Type 4’ by changing a vertex of ‘Type 3’ to become of ‘Type 4’.

Consider a vertex v of ‘Type 3’ whose color (region) is i . And v has all its neighbors of color (region) j . The ‘Remapping Vertices of ‘Type 3’’ in Step 2 of the IHH algorithm attempts to swap the contents of vertex v with another vertex, say w , of color j . Consider the case where the requested conditions are satisfied, and we have improvement of the objective function. So, swapping takes place, and vertex w of color j is remapped to color i . Assume w has a neighbor u of color k , and u was of ‘Type 3’ before the swapping. u had all its neighbors of color j . w used to be of color j with respect to u , but when the swapping occurred, w has been remapped to color i . Hence, a neighbor of u is now of color i . And, u becomes of ‘Type 4’ instead of ‘Type 3’.

The algorithm for remapping vertices of ‘Type 3’, without introducing ‘Type 4’, is shown in Figure 5.4.

Find vertex v of 'Type 3' {i.e. vertex of color i lying within color j }

Loop until no more 'Type 3' vertices are found

- Save OF into OF_1 and Save color i
- Color this vertex v with color j , and update OF into OF_2
- Re-mark all vertices, and find if there is more vertex v of 'Type 3'

End Loop

Figure 5.4 Another Algorithm for Remapping Vertices of 'Type 3'

After this last step, no more 'Type 3' or 'Type 4' vertices are found as will be shown in the formal proof of termination of the IHH algorithm presented in the next section.

5.6 Proof of Termination

To prove that the Iterative Improvement Heuristic algorithm terminates, we must prove the following three statements:

- 1) After the 'Remapping Vertices of Type 3' step, no more 'Type 3' vertices remain.
- 2) After the 'Remapping Vertices of Type 4' step, no more 'Type 4' vertices remain, but 'Type 3' vertices may be introduced.
- 3) If new 'Type 3' is introduced in the previous step, do 'Remapping Vertices of Type 3' again. After this step, no more 'Type 3' vertices remain, and no more 'Type 4' vertices are introduced.

To prove the first statement, let us follow the steps of the algorithm for ‘Remapping Vertices of Type 3’. When we recolor vertex v (of color i) with color j , vertex v is no more of ‘Type 3’ but of ‘Type 1’ (‘inner’ vertex) since it has the same color as all its surrounding vertices. Then, we attempt to swap this vertex with another vertex w of ‘Type 2’, ‘Type 3’, or ‘Type 4’ in order to try to improve the objective function. There are two cases to check. First, no such vertex w is found, and in this case, we would have changed the ‘Type 3’ vertex v to ‘Type 1’, and no new ‘Type 3’ vertices are introduced. In the second case, we find this vertex w of color j and recolor it with color i . A precondition to this remapping is that vertex w must have a neighbor u of color i , and this neighbor u is not of ‘Type 3’ or ‘Type 4’. If w is of ‘Type 3’, it is then of color j and lying within color i , and recoloring it with color i will remap this ‘Type 3’ vertex w to become of ‘Type 1’, and hence no new ‘Type 3’ vertex is introduced. If w is of ‘Type 2’ or ‘Type 4’, new ‘Type 3’ vertices may be introduced because the neighboring vertices type may be changed according to the new color of w . In this case, the next iteration visits these newly introduced ‘Type 3’ vertices and remaps them in the same way. A first argument to prove that this algorithm for remapping vertices of ‘Type 3’ terminates (no more ‘Type 3’ vertices exist) is that once a ‘Type 3’ vertex is remapped, it becomes of ‘Type 1’; it is marked as visited and is not visited as of ‘Type 3’ anymore. A second argument is that vertices are countable, and existing ‘Type 3’ vertices are finite in number, and hence if there is newly introduced ‘Type 3’ vertices, they are also finite in number, and then their type is changed in the next iteration to ‘Type 1’. Because of these two arguments, and because the number of vertices N is finite, no matter how many ‘Type 3’ vertices there are and may be introduced, each vertex may become of ‘Type 3’ only once and will certainly change to ‘Type 1’ (proved before). Let us mark this vertex as

visited. Hence, we reach a stage that no more vertices may become of ‘Type 3’ because they are already visited, and they have an upper limit N .

We want to prove the second statement that after the ‘Remapping Vertices of Type 4’ step, no more ‘Type 4’ vertices exist. ‘Type 3’ vertices may be introduced. We have given an example, in Section 5.4 above, that shows this case. Now, we prove that no more ‘Type 4’ vertices exist after the ‘Remapping Vertices of Type 4’ step is completed. For each vertex v of ‘Type 4’, that has color (region) i and lies within regions that may be different than i , where at least 2 such different neighboring colors, say j and k , must exist, assume the ITH algorithm remaps vertex v to region j . Vertex v is no more of ‘Type 4’ because it already has a neighbor of color j , and it is not possible that it becomes of ‘Type 3’ because of the same reason. v does not become of ‘Type 1’ since it has minimum one neighboring vertex of color different than its own. Hence, it becomes of ‘Type 2’. We attempt to swap this vertex v with another vertex w of ‘Type 2’ or ‘Type 4’ in order to try to improve the objective function. Similar to ‘Remapping Vertices of Type 3’ step, two cases are considered. In the first case, no such vertex w is found, and vertex v has already changed its type from ‘Type 4’ to ‘Type 2’, and hence ‘Type 4’ vertex is remapped to become of ‘Type 2’. In the other case, a vertex w of color j is found and recolored with color i . A precondition to this remapping is that vertex w must have a neighbor u of color i , and this neighbor u is not of ‘Type 3’ or ‘Type 4’. New ‘Type 4’ vertices may be introduced because the neighboring vertices types may be changed according to the new color of w . Also, vertices neighboring to w may become of ‘Type 3’, when w is remapped to a different region, and we have given an example of this case in Section 5.4. In the next iteration, these newly introduced ‘Type 4’ vertices are remapped to

new regions, and swapping with other vertices may be encountered, and hence there is a probability of having new 'Type 4' vertices introduced. The next iteration goes through the same process. In order to insure termination of this step; i.e., have no more 'Type 4' vertices, we must prove that there are no oscillating cases that lead a vertex v_i of 'Type 4' to take on the original color after remapping (and may be swapping) in the same iteration. If this is proved, then we know that any 'Type 4' vertex of the same color does not reappear infinitely in the same iteration, and hence we reach a point in time where we have no more newly introduced 'Type 4' vertices.

We use Contradiction argument to prove the following statement. If a 'Type 4' vertex v_i , of color x , is remapped (and re-colored), it does not take on the same color x again during a certain iteration; i.e., no oscillating case leads a vertex v_i of 'Type 4' to take on the original color after remapping (and may be swapping). The argument is presented as follows:

Assume we have the two vertices that will swap colors,

v_i of 'Type 4' and color x , and

v_j of color y

Calculate objective function OF_1

Color v_i with color y and Calculate OF_2

Color v_j with color x and Calculate OF_3

Then, $(OF_3 < OF_2)$ and $(OF_3 < OF_1)$.

Assume v_i becomes of 'Type 4' again, and

it swaps colors with v_j in an oscillating case, then:

Color v_i with color x and Calculate OF_4

Color v_j with color y and Calculate OF_5 , in this case, we have:

$OF_1 = OF_5$ since we have returned to the initial situation

Then, $(OF_1 < OF_4)$ and

$(OF_1 < OF_3)$ which is **impossible** (contradiction with $OF_3 < OF_1$)

Hence, v_j is not recolored with color y , which means

No swapping occurs, so

v_i is no more of 'Type 4'

Therefore,

no 'Type 4' recurrences occur on a vertex v_i , with the same color x , in the same iteration.

Now, we want to prove the third statement that 'If new 'Type 3' is introduced in the previous step, remap vertices of 'Type 3' again. After this step, no 'Type 3' vertices must exist, and no 'Type 4' vertices must be introduced'.

As we have mentioned above, after remapping vertices of 'Type 4' to become of different types, there may be some vertices that become of 'Type 3'. Then, there is another step to remap vertices of 'Type 3' again. This fourth step is explained in Section 5.5. And, the corresponding algorithm is similar to the one presented the first time 'Type 3' vertices are remapped in Step 2, but no swapping is done in order not to introduce 'Type 4' again. To prove the first part of the statement is to prove that no more 'Type 3' vertices exist after 'Remapping Vertices of Type 3' again. We have previously proved the first statement that 'After the 'Remapping Vertices of Type 3' step, no more 'Type 3' vertices must exist'. The same matter applies here. So, as proved earlier, no 'Type 3' vertices exist after remapping vertices of 'Type 3' again. The second part of the statement is to prove that no 'Type 4' is introduced after the

'Remapping Vertices of Type 3' again. As explained above, while proving the first statement: when remapping a vertex v of 'Type 3' from its region to its surrounding region, it becomes of 'Type 1'. In this step (Step 4), no swapping of vertex v with any other vertex takes place in order not to introduce 'Type 4' vertices. Hence, 'Remapping Vertices of Type 3' again only results in remapping a vertex of 'Type 3' to become of 'Type 1', and clearly, no 'Type 4' vertices are introduced in this case.

Hence, the three statements that insure the termination of the IHH algorithm are proved.

Chapter 6

Improvements to the Hybrid Genetic Algorithm

6.1 Tuning the Hybrid Genetic Algorithm Results

After applying the HGA to decompose a given initial graph into the required number of regions (sub-graphs), we apply the Iterative Improvement Heuristic (IIH) algorithm, presented in Chapter 5, on its output to tune the results. This improves the results produced by the HGA.

We have applied the HGA, proposed in Chapter 4, to a number of test cases which are presented in the experimental results in Chapter 8. The resulting solutions of the graph decomposition problem are examined, and the following remarks on the distribution of the sites among regions are encountered.

Sometimes, there is a site that belongs to a certain region r_1 , but its location is among sites of other regions $r_2, \dots, r_x$ where $x \geq 2$. This kind of misplacement is not examined by our HGA since it does not check the surrounding region(s) while stepping through the algorithm. The HGA depends on two main factors: randomness and objective function. Irregularities, such as the site misplaced within other regions,

can be produced by the random allocation procedure of a site to a certain region because the objective function may reflect some improvement. The value of the objective function depends on the weight balance among the regions versus the internal distances of the regions which must be minimized.

We apply the IIH algorithm explained in Chapter 5 to the output graph of the HGA that is already decomposed into sub-graphs. This will tune up the results of the HGA by eliminating the irregularities of the distribution of vertices among sub-graphs.

The IIH algorithm determines each vertex type according to its region (color) and the neighboring sites' regions (colors). Depending on the surrounding of each misplaced vertex v , the IIH algorithm attempts to remap the vertex to the most appropriate region. The objective function is also used to decide if we may swap the remapped vertex with another vertex, and if this is the case, which vertex is selected that improves the objective function the most. The objective function must remain balanced between the weight factor and the inner distance factor. As we are remapping a misplaced vertex v from region r_1 to another region r_2 , the weight and inner distance of r_1 are decreasing, while the weight and inner distance of r_2 are increasing. Hence, the objective function of region r_1 is less than it used to be before the remapping takes place, while the objective function of region r_2 is more. In order to keep the objective function balanced among regions, we attempt to remap a vertex of r_2 to region r_1 . In this way, we are trying to swap 2 sites between regions r_1 and r_2 . To choose which vertex w of r_2 is swapped with vertex v , we calculate ΔOF for each possibility and accordingly may select the vertex w that improves the objective

function the most; this is done in case the objective function becomes better than it used to be before remapping of the misplaced vertex v .

The IIH algorithm needs to know if there is an edge e_{ij} between any two sites i and j , in order to determine the neighbors of each site, and accordingly its type. This is input to the IIH algorithm. The distribution of the sites (vertices) among the required number of regions (sub-graphs) is given by the HGA output. Re-allocation of vertices between sub-graphs can be done by the IIH algorithm, and then the resulting solution of the graph decomposition is output. The following figure shows the corresponding data flow diagram.

Figure 6.1 Data Flow Diagram for Tuning up HGA Results

6.2 Breakup of Vertices Weights for the HGA Input Graph

Breaking up of weights of vertices for the graph to be decomposed into sub-graphs can be done as a pre-processing step to the HGA for our graph decomposition problem. The pre-processing of the input graph is implemented as an independent unit that results in rebuilding the input graph for the HGA, so that this graph is weight balanced.

Heavy weights of vertices may affect the weight factor of the objective function. When weights are close to each other in value, there is some added flexibility to distribute weights among regions more evenly, and hence deviation of regions' weights from the average weight of vertices is minimized which may result in improving the objective function. On the other hand, the inner distances of the regions are affected because there are some sites which are created as a result of breaking up of heavy sites, and hence each new site adds distances to reach all other sites in the graph. These newly added distances get into account in the calculation of the inner distances of the regions, and increase the total objective function. This is a local phenomenon that results from the pre-processing step (black box) , and the resulting updates over the graph weights and distances are feeded as input to the HGA that decompose the rebuilt graph as any other input graph.

The advantage of this pre-processing step is that the balanced weights are in a more 'free' environment to move from one region to another. We do not encounter the following situation anymore: a heavy vertex cannot swap regions with a light vertex although the inner distances of the regions are less, because the weight factor

of the objective function is disturbed more than its inner distance factor. The diagram in Figure 6.2 explains the data flow between the pre-processing step and the HGA.

Figure 6.2 Data Flow Diagram for Breaking up of Vertices Weights

In the break up process, we divide each vertex v with heavy weight w into two new vertices v_1 and v_2 such that:

the weight of v_1 = the weight of v_2 = $w / 2$.

A heavy weight w is a weight that is more than one and a half the average weight of vertices of the graph.

Although the two newly created vertices have the same physical location on the graph, and hence the same physical name, v_1 and v_2 are treated as two different sites with their corresponding weights. One of these vertices has the same gene number as the original vertex v , but has its weight updated. The other vertex number is augmented to the chromosome as a new gene with its own weight ($w / 2$). The minimum distance matrix is extended to include the distances from the new site to every other site. These distances are equal to the distances from the original site to every other site. These new weight matrices and minimum distance matrices are used

to rebuild the input graph for the HGA to produce a finer-grain graph that is decomposed into regions. The algorithm for the pre-processing step is presented in Figure 6.3.

$$Avg_weight := \sum_{i=1}^N w_i / N$$

For $i := 1$ to N { i is the site under process }

If ($w_i > 1.33 * Avg_weight$) then

 Break up weight w_i into two equal weights $w_i / 2$

 Update weight of site i to be equal to $w_i / 2$

 Add a new site j with weight $w_i / 2$

 Update the minimum distance matrix to include j 's distances

 Save the Broken up sites correspondance

End If

End Loop

Rebuild Graph

Figure 6.3 Breaking up of Vertices Weights Algorithm

Chapter 7

Heuristic Algorithm for Finding Centers

After applying the Genetic Algorithm for decomposing our initial graph into the required number of regions (sub-graph), a heuristic algorithm for the second phase of the regrouping service sites problem is applied in order to find one center for each region (sub-graph). In this chapter, we present the heuristic algorithm developed in (Dana, 1998) which uses Equation 7.1 given below, and we propose some new equations (Equations 7.2 – 7.6) to be used in this algorithm that add some more flexibility in order to improve the results. In the second section, we give a numerical example.

We recall that a sub-graph is represented by a weighted graph $G' = \langle V', E' \rangle$, where $v_i \in V'$ represents a service site, $(v_i, v_j) \in E'$ with e_{ij} representing the geographical distance between v_i and v_j , $V' \subset V$ and $E' \subset E$. We let w_i be the weight associated with vertex v_i . We want to find one center for each sub-graph G' ; that is, the vertex which replaces or serves the sub-graph G' 's set of vertices V' . Each center is chosen such that it favors heavily weighted vertices and is easily accessible from the different vertices in the sub-graph. The set of vertices $V'' \subset V$, where $v_i \in V''$ is

a center for its region i and $|V''| = p$, where p is the required number of centers, is the solution of the regrouping service sites problem.

7.1. Heuristic Algorithm

1. Determine the minimum distance matrix of G and the weight w_i of each vertex $v_i \in V'$.
2. Set $\lambda = 0$ and $\Delta\lambda = 0.5$.
3. Increase λ by $\Delta\lambda$.
4. Calculate the penetration value δ_i for all $v_i \in V'$.
5. Find the reachable sets $\varphi_\lambda(v_i)$ for all $v_i \in V'$.
6. For every $\varphi_\lambda(v_i)$, for all $v_i \in V'$, check if a vertex (service site) can be reached from all other centers within the penetration value δ_i . Otherwise, go to step 3

Figure 7.1 Heuristic Algorithm

An outline of the heuristic algorithm is given in Figure 7.1. Step 1 is an initialization one. The minimum distance matrix is found when Dijkstra algorithm is applied. Moreover, the weights are given as input to the heuristic algorithm. λ is an edge distance incremented in each iteration (Step 3).

For each vertex $v_i \in V'$, we 'penetrate' along all possible routes from v_i up to a distance δ_i . The penetration value δ_i is given by one of the following Equations:

$$(i) \quad \delta_i = \lambda * w_i / \sum w_j, \quad (7.1)$$

where w_i is the weight of vertex v_i , and

$\sum w_j$ such that $j = 1, \dots, |V'|$ is the sum of the weights of all vertices $v_j \in V'$.

δ_i is set in this way in order to bias the solution in favor of heavily weighted vertices.

δ_i is calculated in Step 4.

$$(ii) \quad \delta_i = \lambda * f(w_i), \quad (7.2)$$

where $f(w_i)$ is a scaled value for w_i relative to the other vertices weights.

Scaling is done according to the following steps:

- Find $w_{\min}$ and $w_{\max}$ within a region, where $w_{\min}$ is the minimum vertex weight among all other vertices' weights, and $w_{\max}$ is their maximum vertex weight.

$$\bullet \quad r_1 = w_{\max} / w_{\min}, \quad (7.3)$$

r_1 is the ratio to be scaled in a proportional scale.

$$\bullet \quad r_2 = \phi r_1, \quad \text{if } (r_2 > 1) \quad (7.4)$$

r_2 is the new scaled ratio which is compressed according to ϕ ,

where ϕ is a user-defined compression coefficient.

If $(r_2 > 1)$, continue with the rest of the steps to find $f(w_i)$ to calculate δ_i ,

Else, do not continue with the rest of the steps, and

calculate δ_i according to Equation 7.1.

7.2. Numerical Example

Consider the graph shown in Figure 7.2. This example uses Equation 7.1 for δ_i , and is taken from (Dana, 1998).

Figure 7.2. Numerical Example Graph

Step 1. Initialization

Distance matrix

	1	2	3	4
1	0	4	2	-
2	4	0	-	-
3	2	-	0	30
4	-	-	30	0

- Find x according to Equation 7.5, where $(1 - x) \leq f(w_i) \leq (1 + x)$

$$r_2 = (1 + x) / (1 - x)$$

$$\Rightarrow x = (r_2 - 1) / (r_2 + 1), \text{ if } r_2 > 1 \quad (7.5)$$

- for each w_i , find $f(w_i)$ according to Equation 7.6.

To map the values (scaled values), we have to re-sort vertices weights according to the new scaling, which implies:

$$\frac{w_i - w_{\min}}{1 - x} = \frac{w_{\max} - w_i}{1 + x}$$

$$\Rightarrow f(w_i) = \frac{(w_i - w_{\min})(1 + x) + (w_{\max} - w_i)(1 - x)}{w_{\max} - w_{\min}}, \quad (7.6)$$

In Step 5, $\phi_\lambda(v_i)$ is the reachable set from v_i , defined as the set of all vertices $y \in V$ from which v_i is reachable within a distance δ_i . ϕ_λ is calculated using the matrix of minimum distances of G . Each vertex is either reachable or not within a distance δ_i from a vertex v_i . Consequently, each vertex v_i can be associated with a binary vector $(j_1, j_2, \dots, j_N)$ of length N (the number of vertices or service sites), where $j_k = 1$ if the vertex v_k is reachable within δ_i from v_i , and $j_k = 0$ otherwise.

The result of Step 6 is the required center. If for the same value of λ , more than one center is found, the one with the highest weight value is favored. This is because we are not only taking into consideration distances between service sites, but also the corresponding weight for each service site. The running time of the heuristic algorithm is $O(N^2)$.

Minimum distance matrix

	1	2	3	4
1	0	4	2	32
2	4	0	6	36
3	2	6	0	30
4	32	36	30	0

Weight vector

1	45
2	31
3	10
4	25

Step 2. Set $\lambda = 0$ and $\Delta\lambda = 0.5$

Step 3. $\lambda = \lambda + \Delta\lambda$

Step 4. To find sets $\phi_{\lambda}(v_i)$, for all $v_i \in V$, we calculate the penetration value δ_i , for all i .

Using Equation 7.1, we have:

$$\delta_1 = 0.5 * 45 / 111 = 0.203$$

$$\delta_2 = 0.5 * 31 / 111 = 0.140$$

$$\delta_3 = 0.5 * 10 / 111 = 0.045$$

$$\delta_4 = 0.5 * 25 / 111 = 0.113$$

Minimum distance matrix

	1	2	3	4
1	0	4	2	32
2	4	0	6	36
3	2	6	0	30
4	32	36	30	0

Weight vector

1	45
2	31
3	10
4	25

Step 2. Set $\lambda = 0$ and $\Delta\lambda = 0.5$

Step 3. $\lambda = \lambda + \Delta\lambda$

Step 4. To find sets $\phi_{\lambda}(v_i)$, for all $v_i \in V$, we calculate the penetration value δ_i , for all i .

Using Equation 7.1, we have:

$$\delta_1 = 0.5 * 45 / 111 = 0.203$$

$$\delta_2 = 0.5 * 31 / 111 = 0.140$$

$$\delta_3 = 0.5 * 10 / 111 = 0.045$$

$$\delta_4 = 0.5 * 25 / 111 = 0.113$$

Minimum distance matrix

	1	2	3	4
1	0	4	2	32
2	4	0	6	36
3	2	6	0	30
4	32	36	30	0

Weight vector

1	45
2	31
3	10
4	25

Step 2. Set $\lambda = 0$ and $\Delta\lambda = 0.5$

Step 3. $\lambda = \lambda + \Delta\lambda$

Step 4. To find sets $\phi_{\lambda}(v_i)$, for all $v_i \in V$, we calculate the penetration value δ_i , for all i .

Using Equation 7.1, we have:

$$\delta_1 = 0.5 * 45 / 111 = 0.203$$

$$\delta_2 = 0.5 * 31 / 111 = 0.140$$

$$\delta_3 = 0.5 * 10 / 111 = 0.045$$

$$\delta_4 = 0.5 * 25 / 111 = 0.113$$

Calculate $\varphi_{0.5}(v_i)$

$\varphi_{0.5}(1)$	1	$\varphi_{0.5}(2)$	0	$\varphi_{0.5}(3)$	0	$\varphi_{0.5}(4)$	0
	0		1		0		0
	0		0		1		0
	0		0		0		1

Step 5. No valid solution found at $\lambda = 0.5$

Iterate until $\lambda = 79$,

$$\delta_1 = 79 * 45 / 111 = 32.027$$

$$\delta_2 = 79 * 31 / 111 = 22.063$$

$$\delta_3 = 79 * 10 / 111 = 7.117$$

$$\delta_4 = 79 * 25 / 111 = 17.792$$

Calculate $\varphi_{79}(v_i)$

$\varphi_{79}(1)$	1	$\varphi_{79}(2)$	1	$\varphi_{79}(3)$	1	$\varphi_{79}(4)$	0
	1		1		1		0
	1		1		0		0
	1		0		0		1

At $\lambda = 79$, our center is vertex 1. With penetration = 32.027, center 1 can reach centers 2, 3 and 4 with distances equal to 4, 2 and 32 km, respectively. All other centers do not have a penetration value that can cover their required distances.

Step 8. Solution: the required center is vertex 1 with respective weight of 45.

Chapter 8

Empirical Results

In this chapter, we present the empirical results of applying our genetic-based and heuristic-based approaches to a number of cases for decomposing a weighed graph into the required number of sub-graphs. Furthermore, our new 2-phase genetic-based method for finding service centers is tested, and the corresponding results are shown. First, we describe the experimental environment, and the type and size of input data used to carry out the testing phase.

In Section 8.2, the HGA described in Chapter 4 is tested and compared with the Non Hybrid GA. In Section 8.3, results of the 3 types of hybridization used in the GA are compared. In Section 8.4, we present the results of reducing graph granularity by breaking up heavy vertices weights for the HGA input graph. In Section 8.5, we present some experimental results produced by applying HGA, and we compare them with the output resulting from tuning up of these HGA results. This Tuning up of HGA results, done by applying the IIH algorithm to the HGA output, is discussed in Chapter 6. Section 8.6 demonstrates the results of applying the heuristic IIH algorithm presented in Chapter 5, and it compares them with the tuned HGA results.

Finally, Section 8.7 presents the results of Phase 2 of our approach, which finds the required number of centers of the sub-graphs decomposed by the Tuned HGA method.

8.1 Experimental Environment

Experiments are carried out on Pentium III 450 MHz, 128 MB of RAM microcomputer. The implementation is done using Borland Delphi Client/Server Suite Version 3.0. A realistic case example is taken from the Schools Regrouping Decision Support System (SRDSS) (Mansour and Rishani, 1998). Other test cases are arbitrarily generated examples designed to demonstrate the operations of our methods.

Each test case consists of some input data that forms a weighed graph, with a certain number of vertices N . Three input files are needed for the HGA, two files that contain the weights and names of the vertices, respectively, and one file that contains the minimum distance matrix. The weights of the vertices of the graph are stored in a text file that has N lines where each line contains the weight of a vertex. The names of the vertices are stored similarly in another text file. In order to calculate the minimum distance matrix of the vertices of the graph, we have applied Dijkstra algorithm on the distances of the edges between any two connected vertices. The distances are calculated, as illustrated in Chapter 2, using the coordinates of the vertices. There are two ways to enter the values of these coordinates, either graphically or numerically. Graphically, a module is implemented to allow the allocation of points (vertices) on an image file and the drawing of lines between

connected points. Drawing tools like Canvas, Dot button and Straight Line button are used to draw the graph. A text file f_1 is created to store the coordinates of the vertices. Each time we press down on the mouse to designate a point (vertex) v on the image, v is given a number for identification, and its corresponding coordinates are stored on a new line of f_1 . When all the vertices are entered, f_1 contains their coordinates consecutively. Furthermore, another file f_2 is used to store the cases where there are connections between any two vertices. If two vertices are connected by a straight line (edge), the two corresponding identification numbers are printed next to each other on a separate line of f_2 . We can drag the mouse between two connected vertices to draw a straight line (edge) between them. Numerically, the coordinates of the vertices can be written on f_1 , and their corresponding connections on f_2 . The two files, f_1 and f_2 , are input to a program that calculates the distances between any two connected vertices. The distance matrix is stored in an output file f_3 . f_3 is given to the program that implements the Dijkstra algorithm to output the minimum distance matrix. This minimum distance matrix is stored in a file that is input to the HGA.

Moreover, the graph decomposition solutions produced by the GA's or the IIH algorithm are drawn using a program that has 3 input files: one file f_1 which includes the graph vertices coordinates, another file f_2 which includes their corresponding connections, and the third file which includes the result of the graph decomposition produced by some GA or IIH algorithm. The edges between vertices drawn on the graph reflect the real distances calculated from the coordinates of the vertices. Hence, this gives us visual demonstration of the distribution of service sites on the graph, and the relative distances among each other. In these graphs, each region R_i , where i is the number of the region, has its vertices colored with a certain color. Moreover, we

let $W(R_i)$ and $D(R_i)$ denote the weight and the inner distance of R_i , respectively. For each region R_i , these two values are written using the corresponding color of R_i . We also let avr_weight be the average weight of the required number of regions, and $Q2$ the value of the GA objective function. These two amounts are written in a box under the graph. The avr_weight can be used to calculate the deviation of each region's weight from it. The objective function is of much significance to reflect the quality of the graph decomposition result produced by any GA.

The size of the SRDSS realistic case graph is 14 vertices. Other graphs used to produce the empirical results in this chapter have $N = 30$, $N = 50$, or $N = 100$ vertices, where N is the number of service sites. The number of sub-graphs or required centers p varies from 3 to 20 depending on the size of the graph. Figures 8.1 - 8.4 show these graphs used in our empirical work.

The following test case is the SRDSS realistic case. Table 8.1 shows selected towns for this test case and their corresponding weights. Figure 8.1 shows a graphical representation of the selected towns, where each vertex represents a town, and the distance of the edge between any two vertices is the geographical distance between the corresponding two towns. The weight of each vertex (town) is written next to it.

Table 8.1 List of considered towns for the SRDSS test case

Town	Town weight
1. Kahale	12.5
2. Bsous	12.5
3. Choueifete	100
4. Aley	62.5
5. Sofar	12.5
6. Chanay	12.5
7. Ain Anoub	12.5
8. Bchamoun	62.5
9. Kaifoun	12.5
10. Aaramoun	37.5
11. Mechrefe	12.5
12. Maasrati	12.5
13. Aabey	12.5
14. Bnaiye	12.5

Figure 8.1 SRDSS Graph, N = 14

As another test case, we consider a graph with 30 vertices (Service Sites). Table 8.2 shows the corresponding weights of the sites. Figure 8.2 shows a graphical representation of the graph for this test case.

Table 8.2 List of Service Sites for the 30-Vertex Test Case

Service Site	Weight
1	29
2	40
3	31
4	85
5	66
6	94
7	28
8	52
9	94
10	69
11	82
12	52
13	53
14	36
15	92
16	69
17	66
18	56
19	27
20	38
21	26
22	97
23	82
24	98
25	44
26	81
27	59
28	85
29	97
30	93

Figure 8.2 30-Vertex Graph, $N = 30$

A third test case consisting of a weighed graph that has 50 vertices is shown graphically in Figure 8.3. Table 8.3 specifies the weights of the Service Sites (vertices of the graph).

Table 8.3 List of Service Sites for the 50-Vertex Test Case

Service Site	Weight
1	29
2	40
3	31
4	85
5	66
6	94
7	28
8	52
9	94
10	69

Service Site	Weight
26	81
27	59
28	85
29	97
30	93
31	77
32	60
33	96
34	45
35	80

Table 8.3 Continued

Service Site	Weight	Service Site	Weight
11	82	36	70
12	52	37	50
13	53	38	55
14	36	39	69
15	92	40	99
16	69	41	33
17	66	42	44
18	56	43	39
19	27	44	98
20	38	45	28
21	26	46	72
22	97	47	97
23	82	48	62
24	98	49	35
25	44	50	60

Figure 8.3 50-Vertex Graph, N = 50

The last test case consisting of a weighed graph that has 100 vertices is shown graphically in Figure 8.14. Table 8.7 specifies the weights of the 100 Service Sites (vertices).

Table 8.4 List of Service Sites for the 100-Vertex Test Case

Service Site	Weight	Service Site	Weight	Service Site	Weight	Service Site	Weight
1	29	26	81	51	29	76	81
2	40	27	59	52	40	77	59
3	31	28	85	53	31	78	85
4	85	29	97	54	85	79	97
5	66	30	93	55	66	80	93
6	94	31	77	56	94	81	77
7	28	32	60	57	28	82	60
8	52	33	96	58	52	83	96
9	94	34	45	59	94	84	45
10	69	35	80	60	69	85	80
11	82	36	70	61	82	86	70
12	52	37	50	62	52	87	50
13	53	38	55	63	53	88	55
14	36	39	69	64	36	89	69
15	92	40	99	65	92	90	99
16	69	41	33	66	69	91	33
17	66	42	44	67	66	92	44
18	56	43	39	68	56	93	39
19	27	44	98	69	27	94	98
20	38	45	28	70	38	95	28
21	26	46	72	71	26	96	72
22	97	47	97	72	97	97	97
23	82	48	62	73	82	98	62
24	98	49	35	74	98	99	35
25	44	50	60	75	44	100	60

Figure 8.14 Graph of the 100-Vertex Test Case

8.2 Hybrid versus Non-Hybrid GA

In this section, experimental results produced, when random hybridization is implemented in the GA for solving the re-grouping problem, are compared to the results of the non-hybrid genetic approach.

The result of applying the Non Hybrid GA to the SRDSS test case is depicted in Figure 8.5 (a). Figure 8.5 (b) shows the output of HGA for the same test case. In this example, the graph of this test case is decomposed into 3 regions (sub-graphs). As explained before, in these graphs and in the following ones, each region R_i , where i is the number of the region, has its vertices colored with a certain color. Moreover, we let $W(R_i)$ and $D(R_i)$ denote the weight and the inner distance of R_i , respectively. We also let avr_weight be the average weight of the required number of regions, and $Q2$ the value of the GA objective function.

(a) Non Hybrid GA

(b) Hybrid GA

Figure 8.5 Results for SRDSS Graph, ($N = 14$, $p = 3$)

As shown in Figure 8.5 (a), the Non Hybrid GA has produced a graph decomposed into regions, whose sites are not well distributed among these regions. Each sub-graph has many of its sites not neighboring each other, and this results in increasing its inner distance factor. For example, the region R2 (green color) has the three towns: AinAnoub, Abey and Maasrati, not neighboring each other. Similarly, region R3 (blue color) has a high inner distance factor due to the fact that, to go from one town of R3, for example, Chanay, to another town of R3, for example Bneye, we have to pass through other towns of other regions; in this example: Sofar and Mechrefe of region R1 (red color), and Maasrati of region R2. The graph decomposition using the Non Hybrid GA produces a higher objective function when compared to the HGA. This results in a much better solution when using the HGA for solving the graph decomposition problem. In this section, we apply the 'Random Hybridization' to the GA to implement the HGA and compare its results to the Non Hybrid GA results.

Figure 8.5 (b) shows the higher quality result using the HGA for the SRDSS graph, where $N = 14$ and $p = 3$. The inner distances of the sub-graphs are much less in this figure when applying HGA instead of Non Hybrid GA. The towns of the regions are mostly neighboring each other. But, there is an anomaly produced, where a town, named Kayfoun, of region R3 (blue color), is not connected directly to the other towns of region R3. On the other hand, the inner weights of the regions are well balanced among themselves. The objective function produced using the HGA is much minimized, and the quality of the solution is much better.

In Figure 8.6 (a), the result of applying the Non Hybrid GA to the 30-Vertex graph, where $N = 30$ and $p = 5$, is shown. Figure 8.6 (b) shows the output of HGA for the same test case. In both figures, the graph is decomposed into 5 regions.

(a) Non Hybrid GA

(b) Hybrid GA

Figure 8.6 Results for the 30-Vertex Graph, ($N = 30$, $p = 5$)

Figure 8.6 (b), when using the HGA, shows a lower objective function and higher quality result than Figure 8.6 (a), when using the Non Hybrid GA, of the graph decomposition problem for the 30-Vertex graph.

The sites of the regions of Figure 8.6 (a) are not well organized, and they are distributed widely, which results in big inner distances. Each region is colored with a certain color, and as shown in this figure, the sites (colored with its region color) of a certain sub-graph are not grouped in a way that there are direct connections (edges) among them. They are mingled among each other, and to go from one site to another site of a certain region, we pass through other site(s) of other region(s). This is not our aim, and the objective function produced ($Q2 = 185.86$) reflects the ‘bad’ quality of this result.

On the other hand, in Figure 8.6 (b), sites are well grouped among regions. Inner distances of these regions are minimized, where long edges are placed outside the regions. Mostly, there are direct connections (edges) between consecutive sites of each region. For example, as shown in the figure, region R2 (green color) has all its sites adjacent to each other. There is a site of region R1 (red color) between two blue sites. This is the result of favoring the weight factor over the inner distance factor of the objective function. Placing this site, and not another neighboring site to red sites, in the red region, results in balancing the weights among the regions, which may be in this case more significant to the objective function than lowering the inner distances of the regions, so that the objective function is less. The objective function ($Q2 = 74.79$), when applying the HGA in Figure 8.6 (b), is much less than the one produced when applying the Non Hybrid GA in Figure 8.6 (a), and the resulting graph is a better quality result for the 30-Vertex graph decomposition.

Table 8.5 includes the results of applying the Non Hybrid GA versus the HGA to test cases 1 to 14. For each test case, the *# of sites* and the *# of regions* refer to the number of vertices and sub-graphs of the considered graph, respectively. $\phi 2$ is the value of the objective function of the applied GA, *Texec* is the execution time of the algorithm, and *# gen* is the number of generations required to produce the GA results.

Table 8.5 Results of Non Hybrid GA versus HGA

Test case	# of sites	# of regions	Non Hybrid GA			Hybrid GA		
			$\phi 2$	Texec	# gen	$\phi 2$	Texec	# gen
#1	14	2	40.65	0:0:0:110	53	33.61	0:0:0:600	66
#2	14	3	40.52	0:0:0:330	50	25.31	0:0:0:880	61
#3	14	4	37.36	0:0:0:830	89	36.15	0:0:1:260	61
#4	30	5	185.86	0:0:3:290	50	74.79	0:0:15:600	101
#5	30	6	176.96	0:0:4:60	50	79.56	0:0:18:670	92
#6	50	3	311.79	0:0:6:860	50	201.99	0:0:20:50	61
#7	50	4	313.28	0:0:8:680	50	172.09	0:0:35:480	73
#8	50	5	309.57	0:0:11:40	50	149.59	0:0:48:830	87
#9	50	7	301.18	0:0:15:600	50	117.47	0:1:8:280	86
#10	50	8	302.89	0:0:20:330	50	120.58	0:1:37:980	103
#11	100	5	632.28	0:1:4:320	50	328.23	0:5:28:230	89
#12	100	10	626.68	0:2:56:650	50	214.77	0:20:7:100	145
#13	100	15	603.72	0:4:42:650	50	171.79	0:44:9:830	176
#14	100	20	581.04	0:8:41:630	50	144.37	0:58:46:270	146

The above table shows the much less objective function produced by applying the HGA to the 14 test cases graph decomposition, when compared to the Non Hybrid GA. This implies that the quality of the solutions produced by the HGA is much better than that produced by the Non Hybrid GA. The price paid for this improvement is the amount of time needed more by the HGA as shown in Table 8.5.

8.3 Three Types of HGA

In this section, the three types of hybridization explained in Section 4.7 are tested and compared, and their experimental results are presented. Figure 8.7 shows the result of applying the 'Random Hybridization' GA to the 50-vertex test case graph to decompose it into 5 regions. Figure 8.8 shows the output of 'Hybridization with Best Among Nearest Regions' GA for the same test case. Figure 8.9 shows the output of 'Hybridization with Best Among Random Regions' GA for the same test case. Each graph is decomposed into 5 sub-graphs.

Figure 8.7 'Random Hybridization' GA
for the 50-Vertex Graph, (N = 50, p = 5)

Figure 8.8 'Hybridization with Best among Nearest Regions' GA
for the 50-Vertex Graph, ($N = 50$, $p = 5$)

Figure 8.9 'Hybridization with Best among Random Regions' GA
for the 50-Vertex Graph, ($N = 50$, $p = 5$)

In Figure 8.8, ‘Hybridization with Best among Nearest Regions’ GA produces a better result for the 50-Vertex graph ($N = 50$, $p = 5$) than ‘Random Hybridization’ GA shown in Figure 8.7. This improvement is reflected visually from the graph and numerically from the value of the objective function. The ‘Hybridization with Best among Random Regions’ GA for the same test case is depicted in Figure 8.9. The objective function value of the solution produced is between the other two types’ objective function values.

Note that, in these test case results for the three types of hybridization for GA, we observe that Figure 8.9 visually reflects a better resulting decomposed graph than the one in Figure 8.8. In Figure 8.8, there is a vertex of region R3 (yellow color) among regions R1 (red color) and R5 (gray color) which looks anomalous. However, the objective function is better may be because the distribution of weights among regions is more balanced. Remember that, not only the regions’ inner distances factor affect the objective function, but also the weights of the regions.

We cannot conclude that one type of hybridization produces better results than another one. As will be seen in Tables 8.6 – 8.9 in the following section, when test cases are compared to each other, the three types of hybridization applied to GA produce different quality solutions for each test case. In parallel processing, we can run the three Hybrid GA’s on a certain case, and then choose the best solution produced depending on the value of the objective function.

8.4 HGA versus HGA with Reduced Granularity

Empirical results of the HGA for the three types of hybridization are compared to the empirical results produced when the breaking up of heavy vertices weights, for the HGA input weights, is carried out. The same test case, presented in the previous section, is used here to demonstrate the effect of the breaking up of vertices weights for the HGA. Figure 8.10 shows the corresponding result of applying the Random Hybridization GA to the 50-Vertex test case in order to decompose its graph into 5 sub-graphs. Figure 8.11 shows the corresponding output of 'Hybridization with Best Among Nearest Regions' GA for the same test case. Figure 8.12 shows the corresponding output of 'Hybridization with Best Among Random Regions' GA for the same test case.

Figure 8.10 'Random Hybridization' GA with Break up of Weights
for the 50-Vertex Graph, (N = 50, p = 5)

Figure 8.11 'Hybridization with Best among Nearest Regions' GA
with Break up of Weights for the 50-Vertex Graph, ($N = 50$, $p = 5$)

Figure 8.12 'Hybridization with Best among Random Regions' GA
with Break up of Weights for the 50-Vertex Graph, ($N = 50$, $p = 5$)

The results of breaking up of weights of the vertices of the 50-Vertex graph, given as HGA input, are shown in the above 3 figures, each using one of the three types of hybridization. This produces reduced granularity effect on the input graph. Each heavily weighted vertex of the graph is broken up into two vertices of moderate weights. In Figure 8.10, some vertices are broken up, and there is one vertex which is broken up into two vertices, one of which is mapped to region R2 (green color) and the second to R3 (yellow color). Similarly, in Figure 8.11, for another type of hybridization, a vertex is broken up into two vertices, one of which is mapped to region R1 (red color) and the second to R3 (yellow color). In Figure 8.12, for the third type of hybridization, two heavily weighed vertices are broken up, where the resulting moderately weighed vertices are mapped to different regions. The figures show some improvement when the graph granularity is reduced for the HGA, and this is reflected by the value of the objective function produced.

Numerical results are given in Tables 8.6 – 8.9 for the 14 considered test cases. Each table is divided into sections (sub-tables), each corresponding to a test case. Table 8.6 corresponds to the SRDSS graph ($N = 14$), with sections for the cases: ($p = 2$), ($p = 3$), and ($p = 4$). Table 8.7 corresponds to the 30-Vertex graph ($N = 30$), with sections for the cases: ($p = 5$), and ($p = 6$). Table 8.8 corresponds to the 50-Vertex graph ($N = 50$), with sections for the cases: ($p = 3$), ($p = 4$), ($p = 5$), ($p = 7$), and ($p = 8$). Table 8.9 corresponds to the 100-Vertex graph ($N = 100$), with sections for the cases: ($p = 5$), ($p = 10$), ($p = 15$), and ($p = 20$).

Table 8.6 Numerical Results of HGA versus HGA with Reduced Granularity
for the 14-Vertex Test Case

14 sites 2 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	33.61	32.95	33.61	31.07	31.07	31.07
Texec	0:0:0:600	0:0:0:380	0:0:0:610	0:0:1:540	0:0:0:550	0:0:1:430
# gen	66	52	66	97	51	97

14 sites 3 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	25.31	30.60	25.31	22.25	24.16	24.16
Texec	0:0:0:880	0:0:1:100	0:0:1:270	0:0:1:320	0:0:1:590	0:0:1:820
# gen	61	53	55	58	51	56

14 sites 4 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	36.15	35.92	30.01	23.27	31.26	30.48
Texec	0:0:1:260	0:0:1:420	0:0:2:970	0:0:2:200	0:0:1:980	0:0:2:640
# gen	61	51	89	71	51	60

Table 8.7 Numerical Results of HGA versus HGA with Reduced Granularity
for the 30-Vertex Test Case

30 sites 5 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	74.79	80.60	80.60	78.91	71.54	75.79
Texec	0:0:15:600	0:0:15:160	0:0:19:720	0:0:15:760	0:0:27:190	0:0:25:920
# gen	101	54	60	78	69	54

30 sites 6 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	79.56	80.52	81.12	112.92	83.36	83.36
Texec	0:0:18:670	0:0:25:100	0:0:21:40	0:0:30:50	0:0:23:950	0:0:29:500
# gen	92	74	54	116	53	60

Table 8.8 Numerical Results of HGA versus HGA with Reduced Granularity
for the 50-Vertex Test Case

50 sites 3 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	201.99	208.54	211.96	194.28	197.32	198.66
Texec	0:0:20:50	0:0:31:420	0:0:58:270	0:1:38:320	0:0:42:290	0:1:34:200
# gen	61	63	111	182	57	115

50 sites 4 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	172.09	199.21	184.72	176.29	178.36	194.53
Texec	0:0:35:480	0:0:33:450	0:0:40:590	0:0:42:70	0:0:45:750	0:1:3:380
# gen	73	52	57	60	53	60

50 sites 5 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	149.59	142.68	147.07	143.27	140.90	137.25
Texec	0:0:48:830	0:2:11:490	0:2:47:850	0:1:28:650	0:3:16:740	0:1:41:60
# gen	87	122	111	109	127	62

Table 8.8 Continued

50 sites 7 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	117.47	117.47	114.61	116.74	112.75	132.72
Texec	0:1:8:280	0:2:7:420	0:2:14:740	0:2:2:160	0:4:59:390	0:3:40:90
# gen	86	64	68	106	110	79

50 sites 8 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	120.58	124.13	133.37	113.16	135.24	122.28
Texec	0:1:37:990	0:2:27:480	0:2:9:20	0:1:42:880	0:2:50:0	0:5:12:360
# gen	103	74	58	78	53	99

Table 8.9 Numerical Results of HGA versus HGA with Reduced Granularity
for the 100-Vertex Test Case

100 sites 5 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	328.23	318.28	303.82	311.92	314.64	306.43
Texec	0:5:28:230	0:7:43:1	0:8:17:9	0:13:40:8	0:17:27:2	0:19:41:3
# gen	89	71	71	93	98	102

100 sites 10 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	214.77	225.37	220.54	223.56	331.79	231.83
Texec	0:20:7:100	0:19:2:950	0:21:28:6	0:20:21:4	0:38:15:170	0:58:49:60
# gen	145	58	65	93	74	117

100 sites 15 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	171.79	172.22	172.92	214.33	175.38	173.08
Texec	0:44:9:830	0:40:10:460	0:49:5:220	0:42:46:880	1:4:23:620	1:53:4:450
# gen	176	56	64	139	57	103

100 sites 20 regions	Hybrid	Genetic	Algorithm	HGA with	Reduced	Granularity
	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions	Random Hybridize	Best of X Nearest Regions	Best of X Random Regions
ϕ 2	144.37	162.70	155.44	151.63	155.01	187.04
Texec	0:58:46:270	1:04:38:620	1:11:48:007	1:13:50:006	1:40:41:006	1:53:15:280
# gen	146	56	65	131	55	60

In most of the test cases shown in Tables 8.6 – 8.9, improvement is observed when the graph vertex granularity is reduced for the HGA, and this is reflected by the value of the objective function produced. Few exceptions of this are encountered. Heavily weighted vertices of the graph are broken up into two vertices of moderate weights. As the experimental results show, this helps in balancing the weights of the regions where weights are more flexible to move among regions.

8.5 HGA versus Tuned HGA Results

In Section 6.1, ‘Tuning up of HGA Results’ by applying the IIH algorithm to the output produced by HGA has been presented. From this section and on, we apply ‘Random Hybridization’ to GA to produce the Hybrid GA (HGA). In this section, the ‘Random Hybridization’ GA (HGA) results are compared to the results produced after their tuning by the IIH algorithm. Table 8.10 shows these results for the 14 test cases. The objective function (ϕ_2), execution time (**T_{exec}**), and number of generations (**#gen**) produced by HGA and Tuned HGA are given in the columns labeled **HGA** and **Tuned HGA**, respectively.

Table 8.10 Results of HGA versus Tuned HGA

Test case	# of sites	# of regions	HGA			Tuned HGA		
			ϕ 2	Texec	# gen	ϕ 2	Texec	# gen
#1	14	2	33.61	0:0:0:600	66	33.61	0:0:0:600	66
#2	14	3	25.31	0:0:0:880	61	25.71	0:0:0:930	61
#3	14	4	36.15	0:0:1:260	61	36.15	0:0:1:260	61
#4	30	5	74.79	0:0:15:600	101	86.08	0:0:15:710	101
#5	30	6	79.56	0:0:18:670	92	79.56	0:0:18:670	92
#6	50	3	201.99	0:0:20:50	61	197.11	0:0:20:220	61
#7	50	4	172.09	0:0:35:480	73	172.23	0:0:35:810	73
#8	50	5	149.59	0:0:48:830	87	184.05	0:0:48:990	87
#9	50	7	117.47	0:1:8:280	86	117.47	0:1:8:280	86
#10	50	8	120.58	0:1:37:980	103	118.99	0:1:42:980	103
#11	100	5	328.23	0:5:28:230	89	330.80	0:5:28:550	89
#12	100	10	214.77	0:20:7:100	145	213.91	0:20:7:380	145
#13	100	15	171.79	0:44:9:830	176	174.43	0:44:10:100	176
#14	100	20	144.37	0:58:46:270	146	144.16	0:58:46:540	146

Experimentation is done for Tuning up of HGA results and comparing the produced results to the HGA results, and the above table shows the 14 test cases numerical results of HGA versus Tuned HGA. The objective function produced, when Tuning up of HGA results, may not be better, but visually the solutions decomposed graphs look better, and anomalies are removed. This is shown in Figures 8.13 - 8.22. When Tuning up of HGA results, no more vertices are misplaced within other regions.

In the following figures, the graphs decomposed into regions (colors), resulting from applying HGA, are shown versus those resulting from applying Tuned HGA. 'Random hybridization' is applied in the HGA for those test cases. Each following couple of figures corresponds to the same test case, where the first figure represents the decomposed graph produced by the HGA, and the second represents the decomposed graph produced by the Tuned HGA.

Figures 8.13 (a) and (b) correspond to the SRDSS test case, where $N = 14$ and $p = 3$.

(a) HGA

(b) Tuned HGA

Figure 8.13 Results for SRDSS Graph, (N = 14, p = 3)

In Figure 8.13 (a), a town named 'Kayfoun' of Region R3 (blue color) is misplaced between 3 towns: two ('Bsous' and 'Aley') of R1 (red color) and one ('Aramoun) of R2 (green color). When tuning up of HGA results for this SRDSS graph in Figure 8.13 (b), this 'misplaced within two regions R1 and R2' town 'Kayfoun' is swapped with a town of R1, namely 'Bsous'. The objective functions of the two approaches are approximately equal, but visually it is observed that after tuning up of the HGA results, the graph is decomposed with no more misplaced sites among sub-graphs.

Figures 8.14 (a) and (b) correspond to the 30-Vertex test case whose graph is decomposed into 5 regions ($N = 30$, $p = 5$).

(a) HGA

(b) Tuned HGA

Figure 8.14 Results for 30-Vertex Graph, ($N = 30$, $p = 5$)

In Figure 8.14 (a), a service site of Region R1 (red color) is misplaced between 2 service sites of R3 (blue color). When tuning up of HGA results for this 30_VerTEX graph in Figure 8.14 (b), this ‘misplaced within one region’ service site is swapped with a service site of R3. The objective function has increased from 74.49 to 86.08 because there is also a weight factor that affects its value. After tuning up of the HGA results, the weights of the two regions R1 and R3 have been altered to deviate more away from the avr-weight, and hence this increases the objective function. The improvement done on their inner distances, when swapping of the misplaced service site is done, has affected the objective function less than the weight factor. This has resulted in increasing the objective function, but it is observed that after tuning up of the HGA results, the graph decomposition result in Figure 8.14 (b) is better than in Figure 8.14 (a), and no more misplaced sites are encountered.

Figures 8.15 (a) and (b) show the resulting graphs of the 50-Vertex test case decomposed into 3 regions ($N = 50$, $p = 3$).

(a) HGA

(b) Tuned HGA

Figure 8.15 Results for the 50-Vertex Graph. ($N = 50$, $p = 3$)

The HGA result of the 50-Vertex graph, decomposed into 3 regions, is shown above in Figure 8.15 (a), where a service site of Region R3 (blue color) is misplaced between 2 service sites of R2 (grey color). When tuning up of this HGA result in Figure 8.14 (b), this ‘misplaced within one region’ service site is swapped with a service site of R2. Moreover, the objective function has decreased from 201.99 to 197.11 since the weight factor is still balanced between the two regions (compare $W(R2)$ and $W(R3)$ in the two figures), and their inner distances: $D(R2)$ and $D(R3)$ are less. It is also observed visually that after tuning up of the HGA results, the graph is decomposed with no more misplaced sites among sub-graphs.

Figures 8.16 (a) and (b) correspond to the same test case, but the graphs are decomposed into 4 regions ($p = 4$).

(a) HGA

(b) Tuned HGA

Figure 8.16 Results for the 50-Vertex Graph, ($N = 50$, $p = 4$)

Figure 8.16 (a) of the resulting decomposed graph when HGA is applied for the 50-Vertex graph, where $N = 50$ and $p = 3$, includes some misplaced vertices. We have numbered these misplaced vertices and some other vertices with which swapping occurs. The number of each vertex is written to its right. Vertex 3 is of region R1 (red color), and misplaced within region R4 (yellow color). Vertex 35 is of region R3 (blue color) and is between two vertices, one of region R1 (red) and the other of region R2 (gray). Vertex 35 is of 'Type 4', which is the 'misplaced within more than one region' type. Vertex 37 is of region R3 (blue) and is between two vertices of region R2 (gray), so it is misplaced within one region. Vertex 46 is of region R1 (red) and misplaced within region R3 (blue). When tuning up of this HGA result, which is shown in Figure 8.16 (a), by applying to it the IIH algorithm, the decomposed graph in Figure 8.16 (b) is the resulting solution. In this Tuned HGA result, vertex 3 swaps regions with vertex 1, i.e., site 3 becomes of region R4 (yellow), and site 1 becomes of region R1 (red). This makes vertex 5 (yellow region) become misplaced within region R1 (red region). In the next iteration, this site swaps regions (colors) with site 9. Similarly, vertex 37 swaps regions with vertex 39. The 'misplaced within one region' site, that is numbered 46, swaps regions with site 35 which is 'a misplaced within more than one region' site. There are no more vertices that are misplaced within one region (of 'Type 3'). Also, the vertex that is misplaced within more than one region, namely vertex 35, is remapped during the 'Remapping of Type 3' step, and is no more of 'Type 4'. The objective function is not altered much (from 172.09 to 172.23), but after tuning up of the HGA results, the graph is decomposed with no more misplaced sites among sub-graphs, and visually it is observed that this result is much better. In Figures 8.17 (a) and (b), the graphs of the same test case are decomposed into 5 regions ($N = 50$, $p = 5$).

(a) HGA

(b) Tuned HGA

Figure 8.17 Results for the 50-Vertex Graph, ($N = 50$, $p = 5$)

The HGA result of the 50-Vertex graph, decomposed into 5 regions, is shown above in Figure 8.17 (a), where a service site of Region R1 (red color) is misplaced within region R2 (green color). Another service site of region R3 (yellow color) is misplaced within 3 regions: R1 (red color), R2 (green color), and R5 (gray color). When tuning up of this HGA result, the result is shown in Figure 8.17 (b). This ‘misplaced within one region’ service site of region R1 (red) is remapped to the region it is misplaced in, namely R2 (green); however, no swapping occurred with any other vertex of region R2, because the objective function is not minimized if swapping is done. The ‘misplaced within more than one region’ site, of region R3 (yellow color), is remapped to region R1 (red color), and not R2 (green color) or R5 (gray color), because the value of the objective function is better in this case. No swapping is done for this site for the same reason, which is that the objective function is not minimized. Although the objective function has increased from 149.59 to 184.05, the Tuned HGA result looks better than the HGA result without tuning up. The objective function is worse because there is also the weight factor that affects its value, and not only the inner distances factor of the regions. Although the average inner distances of the regions is improved when swapping of misplaced sites is done, the weights of those regions becomes more deviated from the avr-weight, and this increases the objective function. However, it is observed that after tuning up of the HGA results, the graph decomposition result, shown in Figure 8.17 (b), is better than it is in Figure 8.17 (a) because no more misplaced sites are encountered.

Figures 8.18 (a) and (b) also correspond to the 50-Vertex test case where the graphs are decomposed into 8 regions ($p = 8$).

(a) HGA

(b) Tuned HGA

Figure 8.18 Results for the 50-Vertex Graph, ($N = 50$, $p = 8$)

Figure 8.18 (a) of the resulting decomposed graph when HGA is applied for the 50-Vertex graph, where $N = 50$ and $p = 8$, includes some misplaced vertices. Vertices 3 and 4 are of region R7 (yellow color) and R6 (gray color), respectively. They are misplaced within each other's regions, and they are of 'Type 3'. Vertex 5 is of region R7 (yellow color) and is between two vertices, one of region R1 (red color) and the other of region R6 (gray color). Vertex 5 is of 'Type 4', which is the 'misplaced within more than one region' type. Vertex 28 is of region R8 (black color) and only neighboring a vertex of R1 (red color), so it is of 'Type 3'. Vertex 38 is also of the 'Misplaced within One Region' type. Vertex 39 is of 'Type 4' and misplaced within 2 regions R8 (black color) and R5 (light blue color). When tuning up of this HGA result shown in Figure 8.18 (a) by applying to it the IIH algorithm, the decomposed graph, which is shown in Figure 8.18 (b), is the resulting solution. In this Tuned HGA result, vertex 3 is remapped to region R6 (gray color); however, no swapping occurs with any other vertex of region R6, because the objective function is not minimized if swapping is done. Vertex 28 swaps regions with vertex 26; i.e., site 28 becomes of region R1 (red color), and site 26 becomes of region R8 (black color). Vertex 38 swaps regions with vertex 39, and the two vertices are not of 'Type 3' anymore. Then, there is the 'Remapping Vertices of Type 4' step, where vertex 5, which is 'misplaced within more than one region', is remapped to region R6 (gray color), and not R1 (red color), because the value of the objective function is better in this case. It has also swapped regions with vertex 4 of region R6 (gray color). Moreover, the objective function has decreased from 120.58 to 118.99 since the weight factors are still balanced among regions, and their inner distances are less. The tuned HGA result is better, visually and numerically, in this case.

When Tuning up of HGA results, no more vertices are misplaced within other regions. An observation is that there may be produced, even after Tuned HGA, a 'region within another region' case. This is encountered in Figure 8.18 (b) where there is a vertex v of region R8 (black color) between two vertices of region R1 (red color) although v is not misplaced. In the same graph, region R3 (blue color) crosses region R8 (black color). This is the 'region within region' case, where a region R_i is intersected with another region R_j ; i.e., to go from one vertex of R_i to another vertex of the same region R_i , one may pass through a vertex of the other region R_j .

The 100-Vertex test case results are shown in Figures 8.19 – 8.22. In Figures 8.19 (a) and (b), the corresponding graphs are decomposed into 5 regions ($N = 100$, $p = 5$).

Figure 8.19 Results for the 100-Vertex Graph, ($N = 100$, $p = 5$)

Figure 8.19 Results for the 100-Vertex Graph, ($N = 100$, $p = 5$)

The HGA result of the 100-Vertex graph, decomposed into 5 regions, is shown above in Figure 8.19 (a), where a service site of Region R3 (green) is misplaced between 2 service sites of R2 (red), and it is of 'Type 3'. When tuning up of this HGA result in Figure 8.19 (b), this 'misplaced within one region' service site is swapped with a service site of R2. The objective function is not altered much (from 328.23 to 330.8), but after tuning up of the HGA results, the graph is decomposed with no more misplaced sites among sub-graphs, and visually it is observed that this result is much better.

It is also observed that the 'region within another region' case is encountered in Figures 8.19 (a) and (b). Even after Tuning up of HGA results, a 'region within another region' case is not amended. There are two situations where this case is encountered in the 100-Vertex graph, where $p = 5$. Region R1 (light blue color) has a subset of it (more than one vertex, or else it is considered of 'Type 3' or 'Type 4' vertex) within region R4 (dark blue color). Also, region R4 (dark blue color) has a subset of it within region R1 (light blue color). The 'region within region' case is not detected, when tuning up of HGA results takes place, since the IIH algorithm only marks single vertices as misplaced, and not regions.

In Figures 8.20 (a) and (b), the graphs of the same test case are decomposed into 10 regions ($N = 100$, $p = 10$).

Figure 8.20 Results for the 100-Vertex Graph, ($N = 100$, $p = 10$)

Figure 8.20 Results for the 100-Vertex Graph. ($N = 100$, $p = 10$)

Figure 8.20 (a) shows the resulting decomposed graph when HGA is applied to the 100-Vertex graph, for $N = 100$ and $p = 10$, where a service site of Region R10 (black) is misplaced between 2 service sites of R6 (yellow) and R8 (gray), so it is of 'Type 4'. When tuning up of this HGA result in Figure 8.20 (b), this 'misplaced within more than one region' service site is remapped to region R8 (gray), and not R6 (yellow) because the value of the objective function is better in this case. It has also swapped regions with a vertex of region R8 (gray). The objective function is not altered much (from 214.77 to 213.91), but after tuning up of the HGA results, the graph is decomposed with no more misplaced sites among sub-graphs, and visually it is observed that this result is much better.

The graphs in Figures 8.21 (a) and (b) are decomposed into 15 regions ($N = 100, p = 15$).

(a) HGA

Figure 8.21 Results for the 100-Vertex Graph, ($N = 100$, $p = 15$)

Figure 8.21 Results for the 100-Vertex Graph. (N = 100, p = 15)

Figure 8.21 (a) shows the resulting decomposed graph when HGA is applied to the 100-Vertex graph, for $N = 100$ and $p = 15$, where a service site of region R14 (blue) is only neighboring a site of region R2 (red). So, this service site is of 'Type 3'. When tuning up of this HGA result in Figure 8.21 (b), this 'misplaced within one region' service site is remapped to region R2 (red). It has not swapped regions with any vertex of region R2 since this swapping does not improve the objective function. The objective function has increased a little (from 171.79 to 174.43), but after tuning up of the HGA results, the graph is decomposed with no more misplaced sites among sub-graphs, and visually it is observed that this result is much better.

Figures 8.22 (a) and (b) correspond to the 100-Vertex test case where the graphs are decomposed into 20 regions ($N = 100$, $p = 20$).

Figure 8.22 Results for the 100-Vertex Graph, ($N = 100$, $p = 20$)

(b) Tuned HGA

Figure 8.22 Results for the 100-Vertex Graph, ($N = 100$, $p = 20$)

The HGA result of the 100-Vertex graph, decomposed into 20 regions, is shown above in Figure 8.22 (a), where a service site of Region R8 (gray) is misplaced within 2 regions R10 (pink) and R9 (brown with black dot), so it is of ‘Type 4’. When tuning up of this HGA result in Figure 8.22 (b), this ‘misplaced within more than one region’ service site is remapped to region R10 (pink), and not R9 (brown with black dot) because the value of the objective function is better in this case. It has also swapped regions with a vertex of region R10 (pink). The objective function is not altered much (from 144.37 to 144.16), but after tuning up of the HGA results, the graph is decomposed with no more misplaced sites among sub-graphs, and visually it is observed that this result is much better.

8.6 IIH versus Tuned HGA Results

The IIH algorithm presented in Chapter 5 may work on random input which is not produced by the Genetic Algorithm; i.e., a graph with vertices (service sites) that have randomly allocated region numbers. In this case, IIH is applied to decompose graphs into regions (sub-graphs). In this section, we present the results produced by applying the IIH to a number of test cases. These results are compared to those produced by the Tuned HGA. Table 8.11 shows these results for the 14 test cases. The objective function ($\phi 2$) and execution time (**Texec**) produced by the IIH algorithm and Tuned HGA are given in the columns labeled **IIH** and **Tuned HGA**, respectively.

Table 8.11 Results of IIH algorithm versus Tuned HGA

Test case	# of sites	# of regions	IIH			Tuned HGA		
			$\phi 2$	Texec		$\phi 2$	Texec	# gen
#1	14	2	51.45	0:0:0:110		33.61	0:0:0:600	66
#2	14	3	89.89	0:0:0:160		25.71	0:0:0:930	61
#3	14	4	74.55	0:0:0:110		36.15	0:0:1:260	61
#4	30	5	114.87	0:0:0:550		86.08	0:0:15:710	101
#5	30	6	174.71	0:0:0:550		79.56	0:0:18:670	92
#6	50	3	308.63	0:0:1:50		197.11	0:0:20:220	61
#7	50	4	303.92	0:0:1:160		172.23	0:0:35:810	73
#8	50	5	302.07	0:0:1:100		184.05	0:0:48:990	87
#9	50	7	405.47	0:0:1:320		117.47	0:1:8:280	86
#10	50	8	315.38	0:0:1:150		118.99	0:1:42:980	103
#11	100	5	556.67	0:0:4:450		330.80	0:5:28:550	89
#12	100	10	502.00	0:0:5:160		213.91	0:20:7:380	145
#13	100	15	387.94	0:0:6:590		174.43	0:44:10:100	176
#14	100	20	323.59	0:0:7:200		144.16	0:58:46:540	146

The quality of the solutions produced by Tuning up of HGA results is much better than those produced by the IIH algorithm, and this is shown numerically in Table 8.11 above for the 14 test cases. The objective function has increased significantly when moving from the Tuned HGA results to the IIH algorithm results, which, in general, means that the resulting solutions from the IIH algorithm are worse than those resulting from the Tuned HGA. This is also shown graphically in the following figures.

In the following figures, the graphs, which are decomposed into regions (colors), resulting from applying the IIH algorithm, are depicted. The same test cases of the previous section are used in this section. The IIH algorithm is applied to those test cases in order to produce the decomposed graphs shown in Figures 8.23 - 8.32, versus those graphs resulting from Tuned HGA, presented in the previous section.

Figure 8.23 shows the results of the IIH algorithm when applied to the SRDSS graph, where $N = 14$ and $p = 3$.

Figure 8.23 Results of IIH for SRDSS Graph, ($N = 14$, $p = 3$)

In the above figure, the decomposed graph produced by the IIH algorithm, is much worse than the one produced by the Tuned HGA shown in Figure 8.13 (b). This is reflected by the weight distribution among regions and the value of the objective function. The weights of regions R2 (green) and R3 (blue) are deviated much from the average region weight. The objective function shows this 'not good' weight distribution because its value has increased from 25.71 to 89.89.

Figure 8.24 shows the 30-Vertex test case whose graph is decomposed into 5 regions.

Figure 8.24 Results of IHH for 30-Vertex Graph, ($N = 30$, $p = 5$)

The 30-Vertex graph, decomposed into 5 regions and shown in the above figure, is produced by the IHH algorithm, and this result is worse than the one produced by the Tuned HGA shown in Figure 8.14 (b). Each of regions R2 (green color) and R5 (gray color) is divided into two parts that are separated from each other. This increases the inner distances of the two regions, and hence affects the value of the objective function. The inner distances of R2 and R5 are big, and the objective function value has increased from 86.08 to 114.87.

Figure 8.25 shows the resulting graph of the 50-Vertex test case, which is decomposed into 3 regions.

Figure 8.25 Results of IIH for the 50-Vertex Graph. ($N = 50$, $p = 3$)

In the above figure, the decomposed graph, produced by the IIH algorithm, is worse than the one produced by the Tuned HGA shown in Figure 8.15 (b). Each of regions R2 (gray color) and R3 (blue color) is divided into three parts that are separated from each other. This increases the inner distances of the two regions, and hence affects the value of the objective function, which has increased from 197.11 to 308.63.

Figure 8.26 corresponds to the 50-Vertex test case, where the graph is decomposed into 4 regions. In Figure 8.27, the graph of the same test case is decomposed into 5 regions.

Figure 8.26 Results of IHH for the 50-Vertex Graph, (N = 50, p = 4)

Figure 8.27 Results of IHH for the 50-Vertex Graph, (N = 50, p = 5)

In Figures 8.26 and 8.27, the decomposed graphs produced, by the IIH algorithm, are observed to be worse than the ones produced by the Tuned HGA shown in Figure 8.16 (b) and 8.17 (b). Most of the regions are divided into several parts, which are separated widely from each other. This increases the inner distances of the corresponding regions, and hence affects the values of the objective functions of the two test cases, which have increased from 172.23 to 290.5 and from 184.05 to 298.84, respectively.

Figure 8.28 corresponds to the 50-Vertex test case, where the graph is decomposed into 8 regions, and the same observation, as above, is noted.

Figure 8.28 Results of IIH for the 50-Vertex Graph, ($N = 50$, $p = 8$)

The 100-Vertex test case results are shown in Figures 8.29 – 8.32. In Figure 8.29, the corresponding graph is decomposed into 5 regions. In Figure 8.30, the graph of the same test case is decomposed into 10 regions. The graph in Figure 8.31 is decomposed into 15 regions. Figure 8.32 corresponds to the same 100-Vertex test case, where the graph is decomposed into 20 regions.

In these figures of the 100-Vertex test case, the decomposed graphs, which is produced by the IIH algorithm, are observed to be much worse than the ones produced by the Tuned HGA shown in Figures 8.19 (b) - 8.22 (b). Most of the regions are divided into several parts, which are separated widely from each other. This increases the inner distances of the corresponding regions, and hence affects the values of the objective functions of the four 100-Vertex graph decomposition results. The value of the objective function has increased from 330.8 to 556.67 when $p = 5$. When the 100-Vertex graph is decomposed into 10 regions, the value of the objective function has increased from 213.91 to 502. For $p = 15$ and $p = 20$, the objective functions have increased from 174.43 to 387.94 and from 144.16 to 323.59, respectively.

Figure 8.29 Result of I/H for the 100-Vertex Graph. ($N = 100$, $p = 5$)

Figure 8.30 Result of IHH for the 100-Vertex Graph, ($N = 100$, $p = 10$)

Figure 8.31 Result of I/H for the 100-Vertex Graph. ($N = 100$, $p = 15$)

8.7 Finding Centers for Tuned HGA

In this section, we present the test cases results for the Phase 2 algorithm to find the required number of centers. For each region (sub-graph), a center is found using the heuristic algorithm for finding centers presented in Chapter 7. In the following figures, the centers of the decomposed graphs produced by the Tuned HGA first phase are located. In each graph, the center for the region with color i is circled using the same color.

Figures 8.33 - 8.35 correspond to the SRDSS test case whose graph has 14 vertices. In Figure 8.33, 2 centers of the 2 regions are found. In Figures 8.34 and 8.35, 3 and 4 centers are located, respectively.

The centers of the sub-graphs, which are located by the heuristic algorithm proposed in Chapter 7, are quite effective results.

Figure 8.33 Finding Centers for the SRDSS Tuned HGA Graph, (N = 14, p = 2)

Figure 8.34 Finding Centers for the SRDSS Tuned HGA Graph, (N = 14, p = 3)

Figure 8.35 Finding Centers for the SRDSS Tuned HGA Graph, (N = 14, p = 4)

Figure 8.36 corresponds to the 30-Vertex test case whose graph is decomposed into 5 regions. Figure 8.37 corresponds to the same 30-Vertex test case, but whose graph is decomposed into 6 regions.

Figure 8.36 Finding Centers for the 30-Vertex Tuned HGA Graph, ($N = 30$, $p = 5$) Figure 8.37 Finding Centers for the 30-Vertex Tuned HGA Graph, ($N = 30$, $p = 6$)

Figures 8.38 - 8.42 show the resulting graphs of the 50-Vertex test case where 3, 4, 5, 7, and 8 centers are found, respectively.

Figure 8.38 Finding Centers for the 50-Vertex Tuned HGA Graph, (N = 50, p = 3)

Figure 8.39 Finding Centers for the 50-Vertex Tuned HGA Graph, ($N = 50$, $p = 4$) Figure 8.40 Finding Centers for the 50-Vertex Tuned HGA Graph, ($N = 50$, $p = 5$)

Figure 8.41 Finding Centers for the 50-Vertex Tuned HGA Graph, (N = 50, p = 7)

Figure 8.42 Finding Centers for the 50-Vertex Tuned HGA Graph, (N = 50, p = 8)

The 100-Vertex test case results are shown in Figures 8.43 - 8.46 where 5, 10, 15, and 20 centers of the corresponding graphs are located, respectively.

Figure 8.43 Finding centers for the 100-Vertex Tuned HGA Graph. ($N = 100$, $p = 5$)

Figure 8.44 Finding centers for the 100-Vertex Tuned HGA Graph, ($N = 100$, $p = 10$)

Figure 8.45 Finding centers for the 100-Vertex Tuned HGA Graph, ($N = 100$, $p = 15$)

Chapter 9

Discussion of Empirical Results

Discussion of the experimental results of our work is presented in this chapter. Conclusions are inferred about the quality of our solutions. Some algorithms presented in this thesis imply important quality improvements. We deduce the following observations.

The objective function of the HGA applied for the graph decomposition problem reflects the quality of the resulting solution. The lower value objective function seems to produce the better result, but not all the time. The results of decomposing a weighted graph into sub-graphs (regions) are measured up according to two factors: each region weight factor and its inner distance factor. Weights of regions must be well-balanced so that the solution is well-balanced. Also, inner distances of the regions must be minimized in order to produce a better quality solution. These two criteria are observed throughout the testing phase. The better the weight and inner distance factors of the solution are, the less its objective function is, and better results are usually produced. As we minimize the HGA objective function, we are improving these factors. Consequently, we are improving the solutions, as shown in the experimental results.

The difference in the objective function ΔOF , which is presented in Chapter 2, improves the running time of the HGA. ΔOF is calculated, instead of calculating two objective functions: OF_1 and OF_2 and then deducting them to produce their difference. ΔOF is used extensively in the hybridization phase of the HGA, and hence produces more efficient solutions for the graph decomposition problem.

When we add hybridization to the Genetic Algorithm for solving the graph decomposition problem, the quality of the solutions is much more improved. In Section 8.2, we have tested the Hybrid GA (HGA) versus the Non Hybrid GA, and the experimental results show the improvement done. These results are presented in Table 8.5. Three factors of the applied GA are compared: the objective function $\phi 2$, the execution time of the algorithm T_{exec} , and the number of generations required to produce the GA results $\# gen$. 14 test cases results are presented in this table, and they show clearly that the objective function of the HGA is much less than the one of the Non HGA. Hence, the quality of the test cases solutions is much improved when applying hybridization to GA. However, the drawback of this quality improvement is that it takes more time. As the size of each test case increases, applying the HGA is more expensive in time than applying the Non HGA. In general, as the test cases results show, the $\# gen$ required to produce the HGA results is more than the one to produce the Non HGA results. The graphs in Figures 8.5 (a) and (b) for the SRDSS test case depict the improvement of solution of HGA versus Non HGA. Similarly, the graphs in Figures 8.6 (a) and (b) for the 30-Vertex test case show the same kind of improvement.

The experimental results of the three types of hybridization of Section 4.7 are presented in Sections 8.3 and 8.4. In Tables 8.6 – 8.9, the 14 test cases numerical

results, which include $\phi 2$, *Texec*, and *# gen* are shown. We observe that there is some improvement among these three types of HGA's, but for each test case, it is not necessarily the same hybridization type that produces the better result. In Figure 8.8, 'Hybridization with Best among Nearest Regions' GA produces a better result for the 50-Vertex Test Case than 'Random Hybridization' GA shown in Figure 8.7. This improvement is reflected visually from the graph and numerically from the value of the objective function. The 'Hybridization with Best among Random Regions' GA for the same test case is depicted in Figure 8.9. The objective function value of the solution produced is between the other two types' objective function values.

Note that in these test case results for the three types of HGAs, we observe that Figure 8.9 visually reflects a better resulting decomposed graph than the one in Figure 8.8. In Figure 8.8, there is a vertex of region R3 (yellow color) among regions R1 (red color) and R5 (gray color) which looks anomalous. However, the objective function is better because the distribution of weights among regions is more balanced. Remember that not only the regions' inner distances factor affect the objective function, but also the weights of the regions.

We cannot conclude that one type of hybridization produces better results than another one. As seen in Tables 8.6 – 8.9, when test cases are compared to each other, the three types of hybridization applied to GA produce different quality solutions for each test case. In parallel processing, we can run the three Hybrid GA's on a certain case, and then choose the best solution produced depending on the value of the objective function.

We note that, throughout this work, in case we are not discussing the three types of hybridization, and we want to compare the HGA to other approaches for

solving the regrouping service sites problem, ‘Random Hybridization’ is applied to have the HGA.

In Section 8.4, the results of breaking up of weights of the vertices of the graph, given as HGA input, are presented. This produces reduced granularity effect on the input graph. Each heavily weighted vertices of the graph is broken up into two vertices of moderate weights. As the experimental results show, this helps in balancing the weights of the regions. Weights are more flexible to move among regions. In Figure 8.10, some vertices are broken up, and there is one vertex which is broken up into two vertices, one of which is mapped to region R2 (green color) and the second to R3 (yellow color). Similarly, in Figures 8.11 and 8.12 for the other two types of HGA. The figures show some improvement when the graph granularity is reduced for the HGA, and this is reflected by the value of the objective function produced. In most of the other test cases shown in Tables 8.6 – 8.9, improvement is observed. Few exceptions of this are encountered.

In Section 8.5, experimentation is done for Tuning up of HGA results and comparing the produced results to the HGA results. Table 8.10 shows the 14 test cases numerical results of HGA versus Tuned HGA. The objective function produced, when Tuning up of HGA results, may not be better, but visually the solutions decomposed graphs look better, and anomalies are removed, and this is shown in Figures 8.13 - 8.22. When Tuning up of HGA results, no more vertices are misplaced within other regions. Note that tuning up of HGA results is done to the HGA without the reduced vertex granularity.

An observation is that there may be produced, even after Tuned HGA, some ‘region within another region’ cases. This is encountered in Figure 8.18 (b) where there is a vertex v of region R8 (black color) between two vertices of region R1 (red

color) although v is not misplaced. In the same graph, region R3 (blue color) crosses region R8 (black color). This is the 'region within region' case, where a region R_i is intersected with another region R_j ; i.e., to go from one vertex of R_i to another vertex of the same region R_i , one may pass through a vertex of the other region R_j . It is also observed that the 'region within another region' case is encountered in Figures 8.19 (a) and (b). Even after Tuned HGA, a 'region within another region' case is not amended. There are two situations where this case is encountered in the 100-Vertex graph, where $p = 5$, which have been explained in Section 8.5. The 'region within region' case is not detected, when tuning up of HGA results takes place, since the IIH algorithm only marks single vertices as misplaced, and not regions.

In Section 8.6, the results produced by the Tuned HGA are compared to the heuristic algorithm: the IIH. The quality of the solutions produced by the Tuned HGA is much better than those produced by the IIH algorithm, and this is shown numerically in Table 8.11 for the 14 test cases. The objective function has increased significantly when moving from the Tuned HGA results to the IIH algorithm results, which in general, means that the resulting solutions from the IIH algorithm are worse than those resulting from the Tuned HGA. This is also shown graphically in Figures 8.23 - 8.32 which depict the IIH solutions. We compare them to the previous graphs for Tuned HGA, and visually conclude that the results of the Tuned HGA are much better in quality although more expensive in time.

The IIH algorithm works on mapping misplaced single vertices in order to be adjacent to another/other vertex/vertices of the same region. This probably produces, as shown in the figures in Section 8.6, graphs that are decomposed into regions, such that these regions are divided into several subsets, which are separated from each other by relatively big distances. This reflects the quality of the IIH algorithm which

is much worse than the quality of the Tuned HGA results, which is also shown by the value of the objective functions produced. The reason for this is that the IHH algorithm fails to take into consideration the weight distribution among regions and their deviation from the average weight. It also does not work to minimize the inner distances of the regions. This is because the objective function is not taken as a measure of quality to be minimized (in our case) subject to these constraints.

We conclude that the Tuned HGA, which is the result of applying the IHH algorithm to the 'Random Hybridization' GA, without breaking up of vertices weights, produces the best quality graph decomposition solutions. Section 8.7 presents the experimental results of finding centers for the Tuned HGA decomposed graphs. This is done by applying the heuristic algorithm presented in Chapter 7 to the 14 test cases, and the results are shown in Figures 8.33 - 8.46.

The centers of the sub-graphs, which are located by the heuristic algorithm, are quite effective results although sometimes not a central service site, in terms of distance, is located because weights also affect the choice of the service site to be elected as a center.

An advantage of the proposed approach is that because the Tuned HGA is genetic in nature, it depends on random selections. Hence, in off-line processing or parallel processing, we can run any GA, specifically the Tuned HGA, several times with different starting seeds (for random number generation), and then, choose the best solution produced.

Chapter 10

Conclusions and Further Work

In our work, the purpose of the regrouping service sites problem is to assign a central service site to a group of sites which they seek to receive the service, such that some objective function can be maximized or minimized subject to some constraints. Finding optimal central sites for the regrouping service sites is our main objective. A genetic algorithm has been previously developed to solve this problem.

In this thesis, we have proposed an improved genetic algorithm to regroup service sites into a smaller number of service centers. This method, called the Tuned hybrid genetic algorithm (Tuned HGA), is developed to give much better results. It is tested against the HGA also proposed in this work, and to the heuristic algorithm IIH, when applied to random input, which we have developed to tune the results of the HGA. The HGA is compared to the existing Non Hybrid GA used to solve this problem, and to another proposed HGA with reduced vertex granularity. The genetic algorithms we have developed and tested differ from the existing implementation in three ways. First, we use hybridization and present some Hybrid Genetic Algorithms (HGA). Second, we add a pre-processing step to reduce the graph vertex granularity. Third, we tune the results of the HGA by applying a heuristic algorithm called Iterative Improvement Heuristic (IIH) to them.

The empirical results show that the proposed Tuned HGA is quite effective and gives much better results than other existing and proposed approaches.

Future research can be directed towards several issues such as:

- Allowing the prior selection of centers by users before applying the algorithm,
 - Implementing a better user interface,
 - Defining the sites weights by applying approaches such as Fuzzy Logic,
 - Improving the Iterative Improvement Heuristic algorithm, and attempting to remove regions' anomalies such as the 'Region within Region' case,
 - Re-implementing the heuristic algorithm to find centers for the service sites using Equation 7.2,
 - Tuning up of HGA results with break up of graph vertices weights.
-

Bibliography

- Baker, J. E., 1985. Adaptive selective methods for Genetic Algorithms. *International Conference On Genetic Algorithm*, 101-110.
- Balakrishnan, J., Cheng, C.H., 2000. Genetic search and the dynamic layout problem. *Computers & Operations Research* 27, 587-593.
- Brassard, G., Bratley, D.A., 1988. *Algorithmics Theory and Practice*. Prentice-Hall International Inc., London.
- Canto, M., 1997. *Mastering Delphi 3*. CA: Sybex, San Fransisco.
- Canto, M., Good, T.M., 1998. *Delphi: Developer's Handbook*. CA: Sybex, San Fransisco.
- Chambers, L.D., 1995. *Practical handbook of genetic algorithms, vol. I*. Boca Raton, FL: CRC Press.
- Chambers, L.D., 1995. *Practical handbook of genetic algorithms, vol. II, Applications*. Boca Raton, FL: CRC Press.
- Christofides, N., 1975. *Graph Theory an Algorithmic Approach*. Academic Press Inc, London, 79-105.
- Conway, D.G., Venkataramanan, M.A., 1994. Genetic search and the dynamic facility layout problem. *Computers and Operations Research* 21, 955-960.
- Current, J., Min, H., Schilling, D., 1990. Multiobjective analysis of facility location decisions. *European Journal of Operational Research* 49, 295-307.
- Dana, T., 1998. *MS Project*. Lebanese American University.
- Davis, L., 1985. Job shop scheduling with genetic algorithm. *Proceedings of the International Conference on Genetic Algorithms and their Applications*, 136-140.

- Davis, L., 1991. *Handbook of Genetic Algorithms*. Van Nostrand ReinHold, NewYork.
- Francis, R.L., McGinnis, L.F., White, J.A., 1992. *Facility Layout and Location: An Analytical Approach*. Prentice-Hall International Inc., Englewood Cliffs.
- Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Massachusetts.
- Grefenstette, J.J., 1986. Optimization of control parameters for Genetic Algorithms. *IEEE Trans. On systems, Man and Cybernetics* 16(1), Jan.-Feb., 122-128.
- Hakimi, S.L., 1964. Optimal locations of switching centers and absolute centers and medians of a graph. *Operations Research* 12, 450-459.
- Homaifar, A., Qi, C.X., Lai, S.H., 1994. Constrained optimization via genetic algorithms. *Simulation* 62, 242-253.
- Love, R.F., Morris, J.G., Wesolowsky, G.O., 1988. *Facilities Location: Models & Methods*. North-Holland, New York.
- Mansour, N., Fox., G.C., 1994. Allocating data to distributed-memory multiprocessors by genetic algorithms. *Concurrency: Practice and Experience* 6, 485-504.
-
- Mansour, N., Rishani, G., 1998. A Decision Support System for Regrouping Public Schools. *Proceedings of the IASTED International Conference Computer Systems and Applications*, Mar. 30-Apr. 2, x-y.
- Ogryczak, W., 1997. On the lexicographic minimax approach to location problems. *European Journal of Operational Research* 100, 566-585.
- Ogryczak, W., 1999. On the distribution approach to location problems. *Computers and Industrial Engineering* 37, 595-612.

- Ogryczak, W., 2000. Inequality measures and equitable approaches to location problems. *European Journal of Operational Research* 122, 374-391.
- Rahman, S.-u., Smith, D.K., 2000. Use of location-allocation models in health service development planning in developing nations. *European Journal of Operational Research* 123, 437-452.
- Rishani, G., 1997. *MS Project*. Lebanese American University.
- Tate, D.M., Smith, A.E., 1995. A genetic approach to the quadratic assignment problem. *Computers & Operational Research* 22, 73-83.
- Weiss, M.A., 1995. *Data Structures and Algorithm Analysis*. The Benjamin/Cummings Publishing Company, Inc.
- Wirth, N., 1976. *Algorithms + Data Structures = Programs*. Prentice-Hall International Inc., Englewood Cliffs.
-