Genetic Hyper-Heuristics for Graph Layout Problems

Behrooz Koohestani

A Thesis Submitted for the Degree of Doctor of Philosophy

Department of Computer Science
University of Essex

January 15, 2013

Acknowledgments

I would like to take this opportunity to express my deepest gratitude to my supervisor, Professor Riccardo Poli, for his continued guidance, assistance and encouragement throughout my PhD studies. I consider myself extremely fortunate and honored to have got the chance to do my PhD under the supervision of one of the undisputed world leaders in Evolutionary Computation and Genetic Programming. I have had a wonderful academic research experience at the University of Essex, and I owe this to my exceptional supervisor.

My special thanks go to my brother, Dr. Kambiz Koohestani who is the best brother and friend that anyone could have. Indeed, words fail me to express my appreciation to him for his endless kindness and support.

I am also deeply indebted to my parents who have provided me with immeasurable support, direction and encouragement through my entire life. It is certainly impossible to thank them enough for all they have done for me.

Abstract

Graph layout problems are a special class of combinatorial optimisation problems, aiming at discovering a linear layout of an input graph such that a certain objective function is optimised. A linear layout is a labeling of the nodes of a graph with unique integers. Graph layout problems are mainly NP-complete, meaning that a guaranteed optimal solution cannot be reached in polynomial time. Because a large number of problems in science and engineering can be formulated as graph layout problems, a variety of methods have been proposed for addressing them.

These methods are mainly *heuristic* in nature and based on graph-theoretic concepts. The best graph-theoretic heuristic algorithms can produce good-quality solutions in a short time, but, of course, they do not guarantee the optimality of the solutions obtained, and the solutions may be far from ideal. *Meta-heuristic* and *Hyper-heuristic* approaches are popular alternatives to classical optimisation techniques in a variety of domains. However, in the case of graph layout problems, there has been a limited exploration of such methods. Although meta-heuristics applied to these problems have shown to typically produce better results than graph-theoretic heuristics, in practice, graph-theoretic heuristics are much more preferred due to being substantially faster and more reliable.

This thesis presents *Genetic Hyper-Heuristics*, which are heuristic search algorithms, exploring the space of problem solvers. They are designed to automatically evolve heuristic algorithms for graph layout problems. The evolved heuristics are reusable, meaning that they are intended for use on unseen problems. The central organising element of the genetic hyper-heuristics is a specialised *Genetic Programming* system. In addition to its application as a hyper-heuristic, our proposed genetic programming system can be utilised as a meta-heuristic in order to solve this class of problems. This is also presented in the thesis.

In this study, we particularly focus on two of the most important graph layout problems, namely *bandwidth* and *envelope* reduction problems, and use them as testbeds for our algorithms. In order to assess the performance of the heuristics generated, we evaluate them on a large set of standard benchmark matrices from the Harwell-Boeing sparse matrix collection against the best bandwidth and envelope reduction algorithms from the literature. The results obtained show that the evolved heuristic algorithms are able to outperform state-of-the-art human-designed algorithms. In addition to being highly effective, the heuristics generated by our hyper-heuristic methods are very fast, and therefore suitable for practical use.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 An Overview of the Relevant Concepts

An optimisation problem is a computational problem in which the aim is to find the best of all feasible solutions, or, in other words, to discover a solution in the search space corresponding to the optimal (minimum or maximum) value of the objective function [131]. Optimisation problems can be categorised in several ways based on the existence or type of constraints, nature of design variables, physical structure of the problem, nature of the equations involved, deterministic nature of the variables, permissible value of the design variables, separability of the functions and number of objective functions [145]. Optimisation problems are also naturally divided into two distinct classes: those with continuous variables, and those with discrete variables, which are called *combinatorial problems* [131]. In general, in the continuous problems, we search for a set of real numbers, while in the combinatorial problems, we search for an object from a finite set or possibly countably infinite set, which may be an integer, set, permutation or graph [131]. Combinatorial optimisation problems are among the most difficult problems faced by computer scientists.

Graph layout problems are a special class of combinatorial optimisation problems, aiming at discovering a linear layout of an input graph such that a certain objective function is optimised [50]. A linear layout is a labeling of the nodes of a graph with unique integers from the set $\{1,...,n\}$ where n is the number of nodes in the graph. The list of important graph layout problems includes Bandwidth, Minimum Linear Arrangement, Cutwidth, Modified Cut, Vertex Separation, Sum Cut, Envelope, Edge Bisection and Vertex Bisection problems [50]. Graph layout problems are mainly NP-complete, meaning that a guaranteed optimal solution cannot be reached in polynomial time [49].

In this study, we particularly focus on two very important graph layout problems, i.e., *bandwidth* and *envelope* reduction problems, and employ them as testbeds for our proposed algorithms.

The bandwidth of a sparse matrix is the distance (in columns) from the main diagonal beyond which all elements of the matrix are zero [136]. The *bandwidth reduction problem* for a matrix consists of finding a permutation of rows and columns of the matrix which reduces the bandwidth. A suitable nodal numbering scheme can reduce the bandwidth in such a way as to bring all non-zero elements into the smallest possible band around the main diagonal.

The sum of the distances between the first non-zero element in each row and the main diagonal of a matrix is a property assessed by a quantity called the envelope size of the matrix [136]. Finding a permutation of rows and columns of a matrix which reduces the envelope size is a problem known as the *envelope reduction problem*. It should be noted that both of the problems mentioned above can also be stated in the context of graphs as we will see in the next chapter.

With respect to the fact that there are $N!$ possible permutations for an $N \times N$ matrix, the bandwidth and envelope reduction problems are considered, in general, very hard combinatorial optimisation problems. Indeed, minimising the bandwidth and envelope of symmetric sparse matrices have been proved to be NP-complete [14, 130].

The bandwidth and envelope reduction problems have a large number of applications in scientific and engineering fields, e.g., large linear systems, finite element methods, large scale power transmission systems, circuit design, VLSI design, data storage, chemical kinetics, network survivability, numerical geophysics, industrial electromagnetics, saving large hypertext media and topology compression of road networks [45, 47, 136]. The same is true for the other graph layout problems. As a result, numerous approaches, including exact, heuristic, and meta-heuristic methods have been proposed for the solution of these problems.

The exact methods can find optimal solutions (global optima), but they are extremely computationally intensive. By contrast, the heuristic methods, which are mainly graph-theoretic-based, cannot guarantee the optimality of the generated solutions, but they are generally very fast. Meta-heuristic methods are viable alternatives to classical optimisation algorithms, and capable of producing better solutions than the heuristic methods. However, in the case of graph layout problems, meta-heuristics are not generally preferred in practice, since they are stochastic in nature and extremely slow in execution compared to graph-theoretic heuristic methods. It should be noted that practically graph-theoretic heuristic algorithms are still the most widely used strategies for solving graph layout problems. We will discuss this in more detail in the next chapter.

Hyper-heuristics are considered as an emerging direction in modern search technology. Despite the fact that the term hyper-heuristic is relatively new, introduced by Cowling *et al.* [37] in 2001, the first research in this area dates back to the early 1960s. Hyper-heuristics can be defined as "heuristics to choose heuristics" [20]. The above definition is the most widely accepted definition of hyper-heuristics. In this context, a hyper-heuristic is, in fact, a high-level method capable of selecting and applying a suitable low-level heuristic at each decision-making point. Note that such a system needs a number of low-level heuristics, which should be provided by the human designer. Hyper-heuristics can also be thought of as "heuristics to generate heuristics" [26]. In this case, the high-level method is designed to intelligently combine simpler heuristics (provided by the human designer) in order to evolve new heuristics. The main distinguishing feature of hyper-heuristic approaches is that they search a space of heuristics rather than a space of candidate solutions directly, which is the case with most meta-heuristics. Research in the area of hyper-heuristics is motivated by the aim of increasing the level of generality at which search methodologies can operate.

Over the past decade, hyper-heuristics have been successfully applied to a variety of problems such as scheduling, timetabling, space allocation, bin packing, SAT, TSP, nurse rostering and vehicle routing [82]. However, to the best of our knowledge, no prior attempt to investigate the application of hyper-heuristics to the graph layout problems has been reported in the literature.

1.2 Objectives and Contributions

Considering the main characteristics, advantages and disadvantages of the algorithms proposed in the literature (see the previous section), it seems that there is a need for developing more effective, accurate and generally applicable graph-theoretic heuristics for solving graph layout problems. The main objective of this thesis is to address this need.

Developing effective and efficient heuristics for graph layout problems is generally hard, and such heuristic methods are often the result of years of research by a number of experts. Therefore, in this study, we aim to automate the process of heuristic generation in order to reduce the development time and the need for human intervention. However, the human designer still needs to identify the building blocks for heuristics, which is a crucial task.

This thesis presents *Genetic Hyper-Heuristics*, which are heuristic search algorithms, exploring the space of problem solvers. They are designed to automatically evolve heuristic algorithms for graph layout problems. The evolved heuristics are reusable, meaning that they are intended for use on unseen problems. The key component of the architecture of the genetic hyper-heuristics is a specialised *Genetic Programming* system. In addition to its application as a hyper-heuristic, our proposed genetic programming system can be utilised as a meta-heuristic in order to solve this class of problems. This is also discussed in the thesis.

In order to assess the performance of the heuristics generated, we evaluate them on a large set of standard benchmark matrices from the Harwell-Boeing sparse matrix collection against the best bandwidth and envelope reduction algorithms from the literature. The results obtained show that the evolved heuristic algorithms are able to outperform state-of-the-art human-designed algorithms. In addition, the heuristics generated by our hyper-heuristic methods are very fast, and therefore suitable for practical use.

The main contributions of this thesis can be summarised as follows:

- Presenting the first genetic programming system for addressing graph layout problems.
- Solving the bandwidth and envelope reduction problems using the proposed genetic programming system.
- Presenting the first hyper-heuristic framework capable of automatically generating heuristic algorithms for graph layout problems.
- Evolving human-competitive heuristics for the bandwidth and envelope re-

duction problems using the genetic hyper-heuristics.

- Proposing new ideas for using a level structure system without its conventional constraints, and also employing novel features in the process of prioritising nodes for the construction of permutations.

- Presenting a genetic hyper-heuristic approach to evolving a highly enhanced version of the well-known Sloan algorithm.

- Presenting a local search algorithm capable of being integrated with graph layout solvers in order to refine the ordering produced by them.

1.3 Publications

The following papers have reported subsets of the results presented in this thesis:

1. B. Koohestani and R. Poli. A genetic programming approach to the matrix bandwidth-minimization problem. In R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6239 of *Lecture Notes in Computer Science*, pages 482–491. Springer Berlin / Heidelberg, 2010.

2. B. Koohestani and R. Poli. A hyper-heuristic approach to evolving algorithms for bandwidth reduction based on genetic programming. In M. Bramer, M. Petridis, and L. Nolle, editors, *Research and Development in Intelligent Systems XXVIII*, pages 93–106. Springer London, 2011.

3. B. Koohestani and R. Poli. A genetic programming approach for evolving highly-competitive general algorithms for envelope reduction in sparse matrices. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and

M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 287–296. Springer Berlin / Heidelberg, 2012.

4. B. Koohestani and R. Poli. On the application of genetic programming to the envelope reduction problem. In *Computer Science and Electronic Engineering Conference (CEEC), 2012 4th*, pages 53–58. IEEE, 2012.

5. Two journal articles under review.

1.4 Thesis Outline

The outline of this thesis is as follows:

Chapter 2 provides background information about graph layout problems with particular attention to two important graph layout problems, namely the bandwidth and envelope reduction problems, which, as indicated above, are used as testbeds for the algorithms proposed in this study.

Chapter 3 presents a comprehensive review of the relevant literature on hyper-heuristics. The fundamentals of genetic programming are also presented, since in this work genetic programming is used both as a meta-heuristic and as a hyper-heuristic.

Chapter 4 describes a specialised genetic programming system, designed to act as the central organising element of genetic hyper-heuristics. The application of the proposed genetic programming system as a meta-heuristic to the bandwidth and envelope reduction problems is also discussed in detail.

Chapter 5 introduces a general framework for genetic hyper-heuristics. Also, based on the proposed framework, two genetic hyper-heuristic approaches are designed for evolving graph-theoretic bandwidth and envelope reduction heuristic algorithms. The heuristics generated are then compared to well-known human-designed algorithms.

Chapter 6 presents a genetic hyper-heuristic for evolving a highly enhanced version of the well-known Sloan algorithm, which is the most widely-used and promising algorithm for reducing the envelope size of sparse matrices and graphs.

Finally, the conclusions and future work are discussed in Chapter 7.

Chapter 2

Graph Layout Problems

2.1 Introduction

As indicated in Chapter 1, *Graph layout problems* are a special class of combinatorial optimisation problems. In order to address such problems, a linear layout of an input graph should be discovered in such a way that a particular cost function is optimised [50]. A linear layout is a labeling or a numbering of the vertices of a graph with distinct integers from the set $\{1,...,n\}$ [50]. Note that n is the number of vertices in the graph. Most of the graph layout problems have been proved to be NP-complete, which implies that there are no known polynomial time algorithms to guarantee finding optimal solutions to these problems. Since a wide range of problems in various disciplines can be formulated as graph layout problems, a variety of algorithms have been developed for solving this class of problems [49]. In this study, two of the most important graph layout problems, i.e., the *Bandwidth* and *Envelope* reduction problems are the focus of attention, and we use them as testbeds for our algorithms.

This chapter is organised as follows: In Sec. 2.2, graph theoretic definitions and notations of graph layout problems are presented. Sec. 2.3 provides a brief historical overview of graph layout problems together with some of their important applications. Sec. 2.4 briefly describes the advantages and disadvantages of the existing methods developed for addressing graph layout problems. In Sec. 2.5, the bandwidth and envelope reduction problems are presented in detail. Finally, Sec. 2.6 summarises this chapter.

2.2 Graph Theoretic Definitions and Notations

Given an undirected graph G, its vertex set is denoted by $V(G)$ and its edge set by $E(G)$. The notation uv stands for an undirected edge $\{u, v\}$. The degree of a vertex u in G is denoted as $deg(u) = deg_G(u)$ and the maximal degree of all the vertices in G as $\Delta(G)$. Note that the degree of a vertex is the number of vertices connected to that vertex. The notation $\Gamma(u) = \Gamma_G(u)$ stands for the neighbourhood of u in G, which is the set $\{v \in V(G) : uv \in E(G)\}$ [155].

A *linear layout* of an undirected graph $G = (V, E)$ with $n = |V|$ vertices is a bijective function $\varphi : V \to [n]$, where $[n] = \{1, ..., n\}$. The set of all layouts of G is denoted by $\Phi(G)$. A layout is also called a linear arrangement, a labeling or a numbering of the vertices of a graph [155].

Given a layout φ of a graph G and an integer i, first, the sets L and R are defined as: $L(i, \varphi, G) = \{u \in V : \varphi(u) \leqslant i\}$ and $R(i, \varphi, G) = \{u \in V : \varphi(u) > i\}$.

According to [155], layout measures for φ of G are then defined as follows:

- The *edge cut* at position i of φ:
  $\theta(i, \varphi, G) = |\{uv \in E \ : \ u \in L(i, \varphi, G) \wedge v \in R(i, \varphi, G)\}|$.

- The *modified edge cut* at position i of φ:
  $\zeta(i, \varphi, G) = |\{uv \in E \ : \ u \in L(i, \varphi, G) \wedge v \in R(i, \varphi, G) \wedge \varphi(u) \neq i\}|$.

- The *vertex cut* or *separation* at position i of φ:
  $\delta(i,\varphi,G) = |\{u \in L(i,\varphi,G) \ : \ \exists v \in R(i,\varphi,G) : uv \in E\}|$.

- The *length* of uv on φ:
  $\lambda(uv,\varphi,G) = |\varphi(u) - \varphi(v)|$.

In order to represent a layout φ of a graph G, a common way is to align its vertices on a line, mapping each vertex u to position $\varphi(u)$. This representation together with the definitions presented earlier are illustrated in Fig. 2.1.

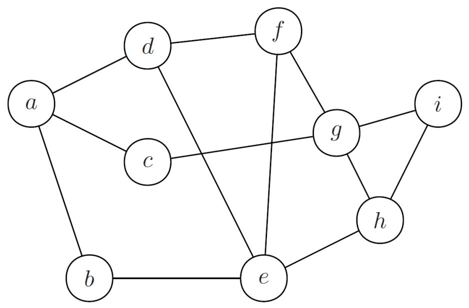
Given a layout φ of a graph $G = (V,E)$, the *reversed layout* of G is denoted by φ^R, and for all $u \in V$, it is calculated as $\varphi^R(u) = |V| - \varphi(u) + 1$.

A *layout cost* is a function F, associating to each layout φ of a graph G an integer $F(\varphi,G)$. If F is a layout cost, then the layout optimisation problem associated with F consists in discovering a layout $\varphi^* \in \Phi(G)$ of an input graph G such that $F(\varphi^*,G) = \underset{\varphi\in\Phi(G)}{min} F(\varphi,G)$ [155].

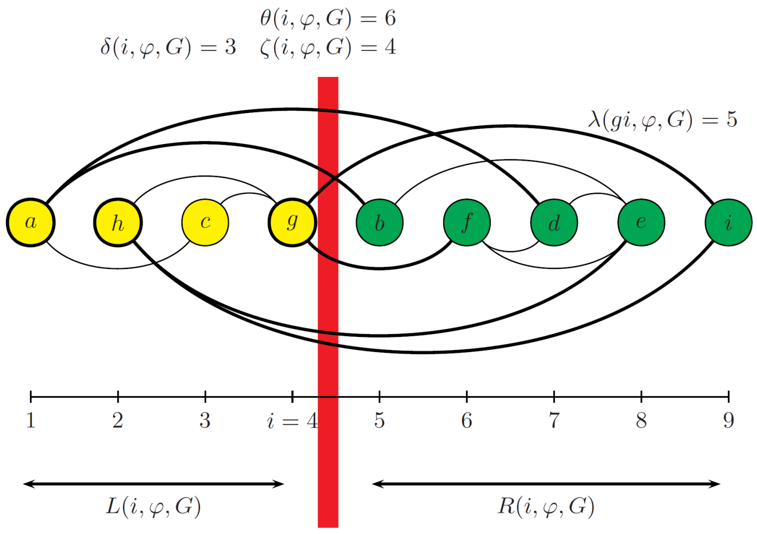
The important graph layout problems include Bandwidth, Minimum Linear Arrangement, Cutwidth, Modified Cut, Vertex Separation, Sum Cut, Envelope, Edge Bisection and Vertex Bisection problems [49, 50, 155]. The layout costs, which correspond to these problems are summarised in Table 2.1.

2.3 A Brief Historical Overview and Applications

The Minimum Linear Arrangement problem was first introduced in 1964 by Harper [78]. His aim was to create error-correcting codes with minimal average absolute errors on certain classes of graphs [78, 79]. Mitchison and Durbin [123] also considered this problem as an over-simplified model of some neural activity in the cortex. In addition, the Minimum Linear Arrangement problem has applications in single machine job scheduling [2, 146]. Note that the Optimal Linear



(a)



(b)

Fig. 2.1: (a) a graph $G = (V,E)$, (b) a graphical representation of the layout $\varphi = \{(a,1),(b,5),(c,3),(d,7),(e,8),(f,6),(g,4),(i,9),(h,2)\}$ of G and its related measures.

Table 2.1: Important graph layout problems together with their costs.

Problem	Cost
Bandwidth	$BW(\varphi, G) = \max_{uv \in E} \lambda(uv, \varphi, G)$
Minimum Linear Arrangement	$LA(\varphi, G) = \sum_{uv \in E} \lambda(uv, \varphi, G)$
Cutwidth	$CW(\varphi, G) = \max{}_{i=1}^{n} \theta(i, \varphi, G)$
Modified Cut	$MC(\varphi, G) = \sum_{i=1}^{n} \zeta(i, \varphi, G)$
Vertex Separation	$VS(\varphi, G) = \max{}_{i=1}^{n} \delta(i, \varphi, G)$
Sum Cut	$SC(\varphi, G) = \sum_{i=1}^{n} \delta(i, \varphi, G)$
Envelope	$ENV(\varphi, G) = \sum_{u \in V} \left(\varphi(u) - \min_{v \in \Gamma^{*}(u)} \varphi(v) \right)$
Edge Bisection	$EB(\varphi, G) = \theta\left(\left\lfloor \frac{n}{2} \right\rfloor, \varphi, G \right)$
Vertex Bisection	$VB(\varphi, G) = \delta\left(\left\lfloor \frac{n}{2} \right\rfloor, \varphi, G \right)$

Ordering, Edge Sum and Minimum-1-sum are alternative names for this problem.

The Bandwidth and Envelope problems have generated a great deal of interest since their discovery in the sixties. The Bandwidth and Envelope problems were originally presented for speeding up several computations on sparse matrices and reducing the amount of storage of sparse matrices respectively [136]. These two problems are closely related and have a wide range of applications in scientific and engineering fields, e.g., numerical analysis, large linear systems, finite element methods, large scale power transmission systems, circuit design, VLSI design, data storage, chemical kinetics, network survivability, numerical geophysics, industrial electromagnetics, saving large hypertext media and topology compression of road networks [34, 45, 47, 72].

The Cutwidth problem was first considered in the seventies as a theoretical model for the number of channels in an optimal layout of a circuit [1, 120]. The cutwidth of a graph multiplied by the order of the graph generally provides a measure of the area required to represent the graph in a VLSI layout [110]. Other applications of this problem include network reliability [84], automatic graph drawing [126] and information retrieval [17].

The Sum Cut problem was originally proposed in 1979 as a simplified version of the δ-operator problem [48]. This problem is approximately equivalent to the Interval Graph Completion problem [146], which has applications in archeology [87] and clone fingerprinting [85].

The Vertex Separation problem was motivated by the problem of discovering suitable separators for graphs [118], and it has applications in VLSI design [109].

The Edge Bisection problem has practical applications in parallel computing and VLSI circuits [16, 108]. This problem is also related to the complexity of sending messages to processors in interconnection networks via vertex-disjoint paths [95].

In section 2.5 of this chapter, the bandwidth and envelope reduction problems are described in detail.

2.4 Advantages and Disadvantages of Existing Methods

As mentioned earlier, graph layout problems have connections with a wide range of other problems in scientific and engineering fields, and therefore a variety of methods have been developed to tackle this class of problems. These methods can be classified into three main categories: *Exact*, *Heuristic* and *Meta-heuristic* methods.

The exact methods, such as dynamic programming, linear programming and branch-and-bound techniques are designed to discover the exact optimal solution (the global optimum) [137]. However, these techniques are extremely costly and time-consuming due to their high computational complexity. Therefore, the exact methods are very rarely used in practice for graph layout problems.

The heuristic methods proposed for graph layout problems are mainly based on graph-theoretic concepts. The best graph-theoretic heuristic algorithms can produce good-quality solutions in a short time, but they do not guarantee the optimality of the solutions obtained.

Meta-heuristic methods such as genetic algorithms, tabu search and simulated annealing are search techniques, mostly based on natural metaphors [154]. Meta-heuristic approaches are viable alternatives to classical optimisation techniques in a variety of domains. However, in the case of graph layout problems, there has been a limited exploration of such methods. Although meta-heuristics applied to graph layout problems have shown to typically produce better results than graph-theoretic heuristics, in practice, graph-theoretic heuristics are much more preferred. For instance, the Cuthill-McKee and GPS algorithms introduced in 1969 and 1976 for bandwidth reduction problem [44, 67], and the Sloan algorithm introduced in 1989 for envelope reduction problem [156] are still the most widely used strategies for addressing these problems.

There are two main reasons for this preference. First, the meta-heuristic approaches proposed in the literature are extremely slow in execution compared to graph-theoretic heuristics, and therefore, they may be highly ineffective on large problem instances usually encountered in practice. Second, due to the stochastic nature of meta-heuristics, they normally generate a different solution in each run, so multiple runs may be required in order to obtain the desired result. The following is a typical example of the practical applications of graph layout problems, which can help illustrate the point.

A bandwidth reduction algorithm is generally used in conjunction with direct methods such as Gaussian elimination for finding efficient solutions to large systems of linear equations. In this case, the bandwidth reduction algorithm plays a key role in decreasing the complexity of the direct techniques used to solve the main problem, which results in substantial savings in computational time. It is clear that in such scenarios, algorithms specifically designed to address graph layout problems need to be as effective and efficient as possible. Otherwise, they may add a significant overhead to the main system, which is by no means desired.

2.5 Bandwidth and Envelope Reduction Problems

In this section, we start with a concise definition of the *bandwidth* and *envelope*. The bandwidth of a given symmetric square matrix A of order N with entries a_{ij} can be defined as the maximum of its row bandwidths, where the row bandwidth $b_i(A)$ for row i is the number of columns from the first non-zero entry in the row to the diagonal. Considering the fact that the envelope of A is the set of elements a_{ij} such that $0 < i - j \leq b_i(A)$ (j is the column index), the *envelope size* of A can be defined as the number of elements in the envelope, or in other words, the sum of its row bandwidths. Note that the envelope size is also referred to as variable band or profile [14, 41]. In the next subsections, the bandwidth and envelope reduction problems are described in detail.

2.5.1 Bandwidth Reduction Problem

The Bandwidth Reduction Problem (BRP) is a very well-known problem, familiar to applied mathematicians and arising in many applications in science and engineering. The BRP consists of labeling the vertices of a graph with integer labels

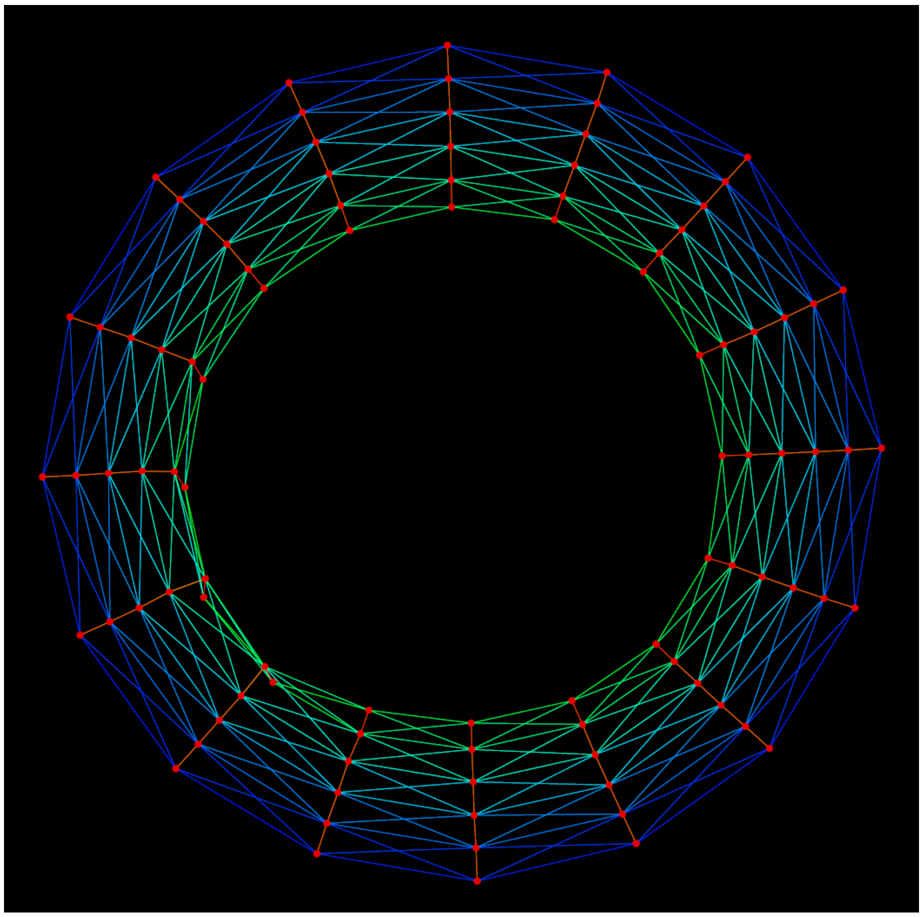


Fig. 2.2: CAN_96 undirected graph (from the Harwell-Boeing collection).

in such a way as to reduce the maximum absolute difference between the labels of adjacent vertices. The problem is isomorphic to the important problem of reordering the rows and columns of a symmetric matrix, such that its non-zero entries are maximally close to the main diagonal [136]. The mathematical definition of the BRP is provided in Sec. 2.5.3. Figs. 2.2–2.4 show an undirected graph together with its associated sparse matrix structure before and after reordering.

As mentioned in section 2.3, the BRP has many applications in different fields.

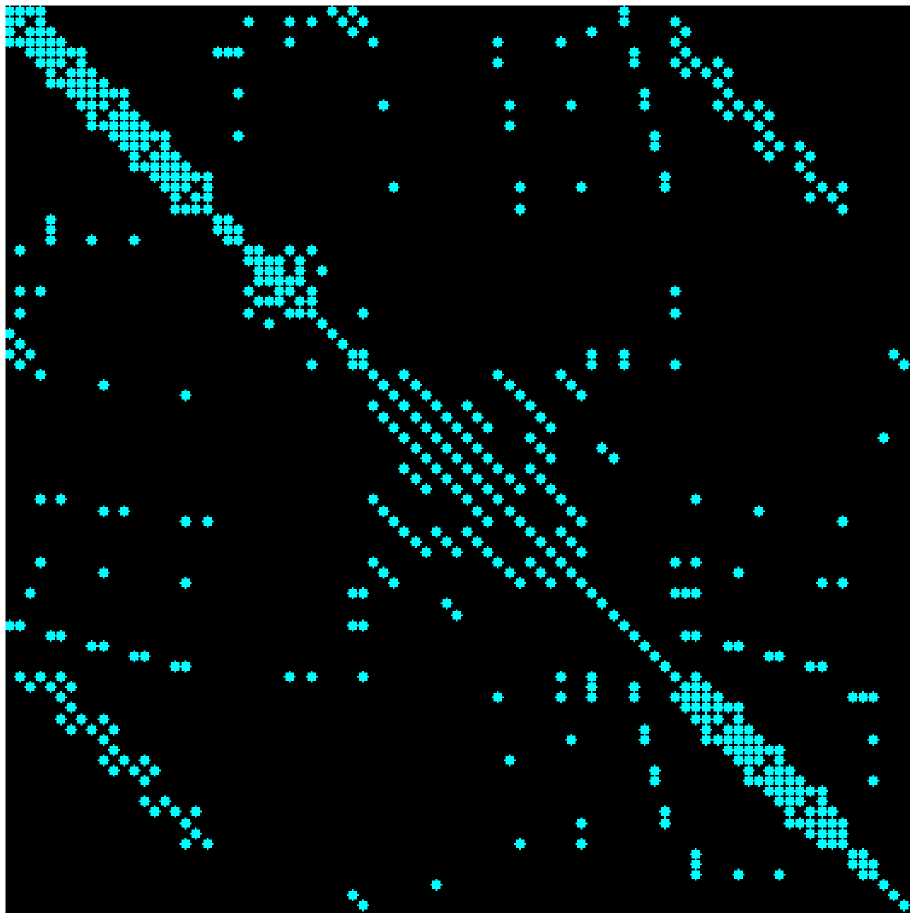


Fig. 2.3: Non-zero entries of the matrix associated with CAN_96 before reordering (the original matrix structure). A cyan dot indicates a nonzero entry.

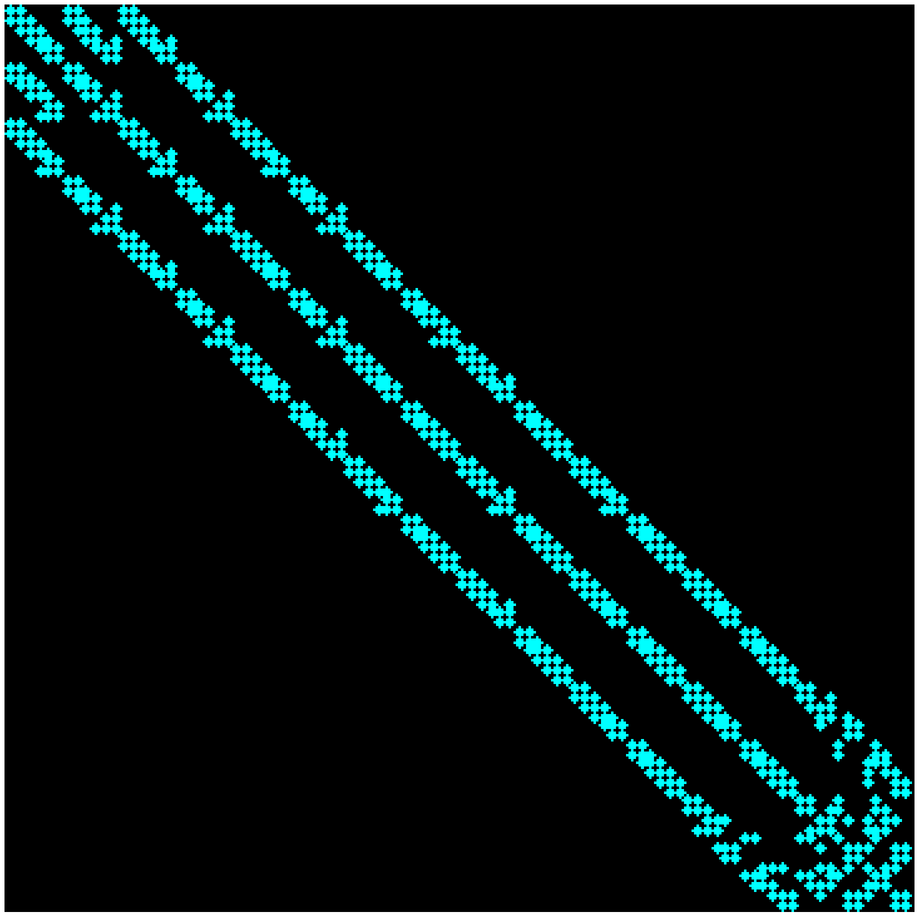


Fig. 2.4: Non-zero entries of the matrix associated with CAN_96 after applying an appropriate nodal numbering scheme. A cyan dot indicates a nonzero entry.

Among these applications, finding efficient solutions to large systems of linear equations through direct methods such as Gaussian Elimination is one of the most common applications of the BRP. The complexity of the direct techniques with an $n \times n$ matrix is $O(n^3)$, while for a banded matrix with bandwidth β, it is $O(\beta^2 n)$ and if $\beta \ll n$, a substantial saving in computational time can be achieved [116].

The problem of minimising the bandwidth size has been proved to be NP-complete [130]. This means that it is highly unlikely that there exists an algorithm, which finds the minimum bandwidth of a matrix in polynomial time. It has also been proved that this problem is NP-complete even for trees with a maximum degree of three, and only in very special cases it is possible to find the optimal ordering in polynomial time [61].

The first exact method for the BRP was proposed by Harary [77]. Gurari and Sudborough [73] and Del Corso and Manzini [35] also presented three other exact methods for addressing the problem. Note that these methods have been applied only to matrices of relatively small size, i.e., up to 100×100.

One of the earliest heuristic approaches for reducing the bandwidth of sparse symmetric matrices was introduced by Cuthill and McKee [44]. Their method is still one of the most widely used methods to (approximately) solve this problem. In this method, the nodes in the graph representation of a matrix are partitioned into equivalence classes based on their distance from a given root node. The partition is known as *level structure* for the given node (see Sec. 2.5.4 for more details). In Cuthill-McKee algorithm (CM), the root node for the level structure is normally chosen from the nodes of minimum degree in the graph. The permutation chosen to reduce the bandwidth of the matrix is then simply obtained by visiting the nodes in the level structure in increasing-distance order.

Gibbs, Poole and Stockmeyer [67] proposed a heuristic algorithm, known as GPS, that made more extensive use of level structures. They also provided a novel heuristic algorithm for finding the endpoints of a pseudo-diameter (i.e., a pair of vertices that are at nearly maximal distance) to be used as an appropriate starting node. The algorithm is substantially faster than the CM algorithm, and it can occasionally outperform it. However, it is significantly more complex to implement. The GPS algorithm has been shown to be an efficient method, in terms of both the quality of the solutions produced and the required computational effort, for the reduction of the bandwidth of finite-element matrices arising from structural engineering problems [55]. A new variant of the GPS algorithm has also been presented by Wang and Shi [159] in which a heuristic parameter called the "width-depth ratio" is incorporated into the original GPS algorithm for finding suitable pseudo-peripheral nodes.

Recently, meta-heuristic approaches have been tested to see if they can be viable alternatives to solve the BRP. Tabu search was employed by Marti *et al.* [121], while Lim *et al.* [114] used a hybrid between a genetic algorithm and hill-climbing to solve this problem. Piñana *et al.* [135] addressed the BRP by a greedy randomized adaptive search procedure combined with a path relinking strategy (GRASP-PR). Lim *et al.* also introduced two other hybrid algorithms: one combining ant colony optimization with hill-climbing [112] and one combining particle swarm optimization with hill-climbing [113]. A simulated annealing approach was proposed by Rodriguez-Tello *et al.* [148] to attack the problem. More recently, a Variable Neighbourhood Search meta-heuristic has also been presented for the BRP [124].

2.5.2 Envelope Reduction Problem

The Envelope Reduction Problem (ERP) consists of labeling the vertices of a graph with integer labels in such a way as to reduce the sum of the maximum positive difference between the labels of adjacent vertices. This problem can also be stated in the context of matrices, in which the objective is to find a reordering of the rows and columns of a symmetric matrix that reduces the sum of the distances of non-zero elements from the matrix's main diagonal [136]. The mathematical definition of the ERP is provided in Sec. 2.5.3.

An important class of nodal numbering schemes attempts to reduce the envelope size of sparse matrices. This is an essential task for solving a variety of real-world problems in science and engineering such as fluid mechanics simulations, finite element analysis and network analysis. Envelope reduction algorithms are still the method of choice for a large number of structural engineering applications, e.g., in the computational structural mechanics testbed (CSM) [96]. In fact, the major advantage of solving a system with reduced envelope size is that the solution requires a smaller number of arithmetic operations and/or storage locations.

Identical to the bandwidth, minimising the envelope size has been proved to be NP-complete [14, 117]. Because the ERP has connections with a wide range of other problems in different fields as mentioned in Sec. 2.3, a variety of methods have been proposed for reordering the rows and columns of sparse matrices in order to reduce the envelope size. These methods are mainly graph-theoretic heuristics.

George [64] in the study of envelope reduction algorithms observed that renumbering the CM ordering in a reverse way (RCM) often yielded a result superior to the original ordering in terms of reducing the envelope size without changing the bandwidth. Experimental evidence confirming the superior performance of the RCM over CM for matrices arising from the finite element method has been reported in [43, 119].

The GK (Gibbs-King) algorithm [66, 111] which is a variant of the GPS algorithm provides considerably better reduction of the envelope size in comparison with the original GPS, but it is often much slower in execution. Armstrong [4] reported that near-minimal envelope sizes could be obtained for a range of finite element grids by using a simulated annealing technique. The algorithm is extremely slow and thus unsuitable for practical use.

The Sloan algorithm [156, 157] offered a very significant improvement over the previous methods by introducing a new step in which the ordering obtained from a variant of the GPS algorithm was locally refined. This algorithm is one of the most important and widely used envelope reduction methods, because it is fast, accurate and capable of generating quality solutions. A number of enhancements to the original algorithm have also been proposed in [53, 147].

A very different algorithm was presented by Barnard *et al.* [15], who proposed the use of spectral analysis of the Laplacian matrix associated with the graph representing the non-zero elements in a sparse matrix as an effective method for the reduction of the envelope size of sparse matrices. In particular, the method permutes a sparse matrix based on the eigenvector associated with the first non-zero eigenvalue of the Laplacian matrix. This algorithm is iterative in nature and can simply be implemented in parallel.

Kumfert and Pothen [107] observed that the spectral algorithm could perform poorly on some problems. In order to fix this, they proposed a hybrid method that combined the spectral ordering with a modified version of the second phase of Sloan's algorithm. Numerical experiments have shown that, for large problems, the hybrid method with an appropriate choice of weights is superior to both the spectral and the Sloan algorithms in terms of the quality of the solutions produced. However, it requires significantly more CPU time than Sloan's algorithm.

Hu and Scott [81] introduced a multilevel algorithm for wavefront and envelope reduction. This algorithm is based on the multilevel algorithms used for graph partitioning and employs Sloan's algorithm. The algorithm does not appear to offer any improvement over the hybrid method proposed by Kumfert and Pothen (see the previous paragraph) in terms of ordering quality, but it is faster.

2.5.3 Mathematical Definitions of the BRP and ERP

Let A be an $N \times N$ symmetric matrix with entries a_{ij}. For the i^{th} row of A, the following relations can be defined:

$$f_i(A) = \min\left\{j : a_{ij} \neq 0\right\} \quad , \tag{2.1}$$

$$b_i(A) = i - f_i(A) \quad , \tag{2.2}$$

$$w_i(A) = |\{k : (k > i) \wedge (\exists l \leq i : a_{lk} \neq 0)\}| \quad . \tag{2.3}$$

in which, $f_i(A)$ is the column index of the first non-zero entry of the i^{th} row of A, $b_i(A)$ is the bandwidth of the i^{th} row of A and $w_i(A)$ is the frontwidth of the i^{th} row of A. According to [44] the bandwidth of A is defined as:

$$B(A) = \max_{(i,j):a_{ij} \neq 0} |i - j| \quad , \tag{2.4}$$

or, equivalently,

$$B(A) = \max\left\{b_i(A)\right\} \quad . \tag{2.5}$$

As stated in [119], the envelope and transpose envelope of A can be formulated as:

$$Env(A) = \{(i, j) : f_i(A) \leq j \leq i\} \quad , \tag{2.6}$$

$$Env^T(A) = \left\{(i, j) : (j \leq i) \wedge (\exists k \geq i : a_{kj} \neq 0)\right\} \quad , \tag{2.7}$$

$$Env^T(A) = Env(A^T) \ , \tag{2.8}$$

where A^T is the transpose of A about its minor diagonal. As a function of $b_i(A)$ or $w_i(A)$, the size of the envelope of A can be expressed as:

$$|Env(A)| = N + \sum_{i=1}^{N} b_i(A) = N + \sum_{i=1}^{N} w_i(A) \ . \tag{2.9}$$

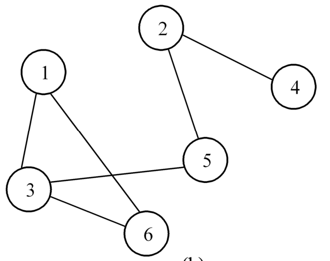
With respect to the definitions presented earlier, the BRP consists of finding a permutation of rows and columns of A which reduces $B(A)$, while the ERP consists of finding a permutation of rows and columns of A which reduces $|Env(A)|$. The ultimate aim, of course, is to reduce these quantities as much as possible. Since there are $N!$ possible permutations for an $N \times N$ matrix, the BRP and ERP are considered, in general, very difficult combinatorial optimisation problems. The definitions in Eqs. (2.1)–(2.9) as well as the process of bandwidth and envelope reduction are illustrated in Fig. 2.5.

While the equations reported above state the mathematical definitions of the bandwidth and envelope of a matrix, in order to develop fast algorithms for sparse matrices, one really needs to calculate such quantities from graphs associated with the matrices. An n-node graph can be associated with an $n \times n$ matrix A such that if element $a_{i,j}$ is non-zero then there is an edge from a vertex i to a vertex j in the graph. It should be noted that in this case, the diagonal entries of A are not considered as edges of the graph.

Algorithm 1 presents the process of calculation of the bandwidth and envelope of a graph $G = (V, E)$ with N vertices associated with a given matrix. The algorithm uses the function $Adj(i)$, which returns a list of integers that indicate which vertices of a graph are adjacent to vertex i (step 2). The list is simply constructed by reading the adjacency list of the graph. This list can also be generated by using the

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & * & 0 & * & 0 & 0 & * \\ 2 & 0 & * & 0 & * & * & 0 \\ 3 & * & 0 & * & 0 & * & * \\ 4 & 0 & * & 0 & * & 0 & 0 \\ 5 & 0 & * & * & 0 & * & 0 \\ 6 & * & 0 & * & 0 & 0 & * \end{array}$$

(a)



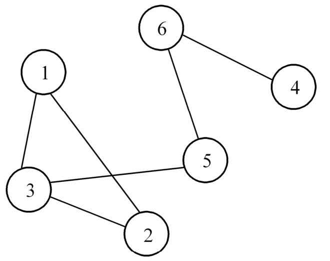
(b)

i	$f_i(A)$	$b_i(A)$	$w_i(A)$
1	1	0	2
2	2	0	4
3	1	2	3
4	2	2	2
5	2	3	1
6	1	5	0
$B(A)$		5	
$\|Env(A)\|$		18 (12+6)	18 (12+6)

(c)

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & * & * & * & 0 & 0 & 0 \\ 2 & * & * & * & 0 & 0 & 0 \\ 3 & * & * & * & 0 & * & 0 \\ 4 & 0 & 0 & 0 & * & 0 & * \\ 5 & 0 & 0 & * & 0 & * & * \\ 6 & 0 & 0 & 0 & * & * & * \end{array}$$

(d)



(e)

i	$f_i(A)$	$b_i(A)$	$w_i(A)$
1	1	0	2
2	1	1	1
3	1	2	1
4	4	0	2
5	3	2	1
6	4	2	0
$B(A)$		2	
$\|Env(A)\|$		13 (7+6)	13 (7+6)

(f)

Fig. 2.5: Example illustrating the definitions of the bandwidth and envelope, as well as the process of bandwidth and envelope reduction: (a) a matrix of order 6 with some non-zero entries (marked with *), (b) its corresponding graph, (c) the related bandwidth and envelope size calculations, (d) the same matrix after applying the permutation $P = (1, 6, 3, 4, 5, 2)$ to its rows and columns, (e) the corresponding graph of the new matrix obtained, (f) the bandwidth and envelope size calculations for the new matrix obtained.

Algorithm 1 Calculation of the bandwidth and envelope size for a graph $G = (V, E)$ with N vertices.

```
1:  for $i = 1, \cdots, N$ do
2:      $l_i = Adj(i)$ {$Adj(i)$ returns the list of all vertices adjacent to vertex $i$}
3:      for each vertex $k \in l_i$ do
4:          $b_i^k = i - k$
5:          if $b_i^k < 0$ then
6:              $b_i^k = 0$
7:          end if
8:      end for
9:      $b_i = \max b_i^k$
10: end for
11: return $\max b_i$ {the bandwidth of $G$} or $N + \sum_{i=1}^{N} b_i$ {the envelope size of $G$}
```

adjacency matrix of the graph, which is not recommended because of the need for scanning the entire matrix. For example, the function would return the following results if applied to nodes 1 through 6 in the graph shown in Fig. 2.5 (b): $\{3,6\}$, $\{4,5\}$, $\{1,5,6\}$, $\{2\}$, $\{2,3\}$, $\{1,3\}$.

Next, for each vertex k in the list l, the maximum positive difference between the labels of adjacent vertices are calculated (steps 3-9). This process should be repeated for each vertex of graph G. The bandwidth/envelope size is then computed and designated as the final output of the algorithm (step 11).

2.5.4 Some Useful Definitions Related to the BRP and ERP

Level Structures. One of the most important concepts in many graph-theoretic bandwidth and envelope reduction algorithms is that of *level structure* [67]. A level structure, $L(G)$, of a graph G is a partition of the set $V(G)$ into levels $L_1, L_2, \ldots, L_K$ such that:

1. all vertices adjacent to vertices in level L_1 are in either level L_1 or L_2,
2. all vertices adjacent to vertices in level L_K are in either level L_K or L_{K-1},
3. for $1 < i < K$, all vertices adjacent to vertices in level L_i are in level L_{i-1}, L_i, or L_{i+1}.

To each vertex $v \in V(G)$, there corresponds a particular level structure $L_v(G)$ called the *level structure rooted at* v. Its levels are determined by:

1. $L_1 = \{v\}$,
2. for $i > 1$, L_i is the set of all those vertices adjacent to vertices of level L_{i-1} not yet assigned to a level.

Note that if a vertex belongs to L_j, then the distance between that vertex and the root vertex is $j-1$. Fig. 2.6 illustrates an example of level structures.

For any level structure L, a numbering f_L of G assigns consecutive integers level by level, first to the vertices of level L_1, then to those of L_2, and so on [67]. In a nodal numbering scheme based on the level structures such as [44, 64, 67], there are then two important elements that influence performance. The first is the method used to specify a suitable starting vertex. This tends to be chosen from either the vertices of minimum degree or the *pseudo-peripheral vertices* [63] in a graph. Recall that vertices $v1$ and $v2$ are called pseudo-peripheral vertices if they are at nearly maximum distance, i.e., the length of a shortest path connecting them is close to the diameter of G. The second element is the method of numbering the vertices located in each level. One of the most effective approaches is to number the vertices of each level based on their increasing degree. This is the method adopted

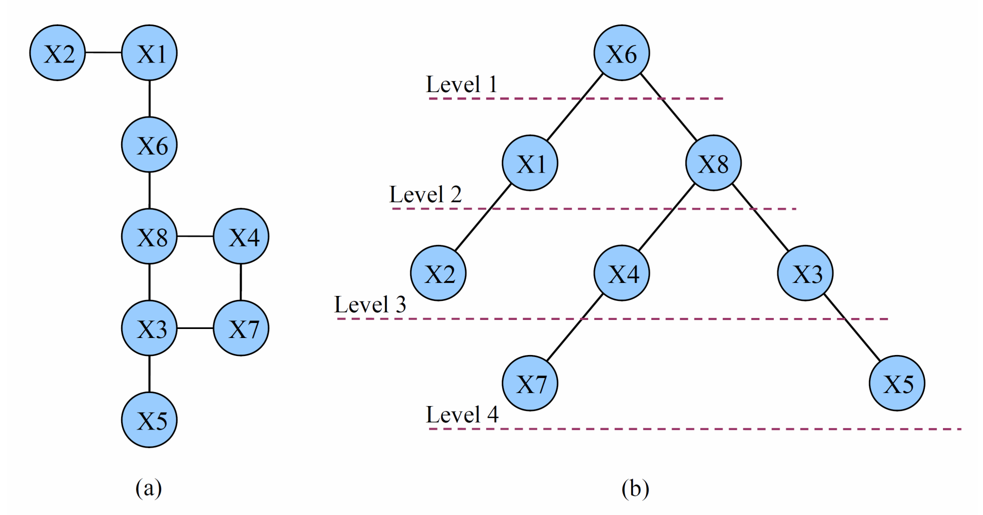


Fig. 2.6: An example of level structures: (a) an 8-node graph, (b) its corresponding rooted level structure at vertex X6.

by the RCM algorithm. In this process, it is very likely that a level contains ties, which are vertices with the same degree. The most common strategy for dealing with this issue is to break ties randomly.

Laplacian Matrix. Consider an undirected, connected graph $G = (V,E)$ and the adjacency matrix $A(G)$ of G. Let $d(v)$ denote the degree of a vertex $v \in V$, and let $D(G)$ denote the diagonal matrix with diagonal entries $d(v_i)$. The matrix $L = L(G) = D(G) - A(G)$ is called the Laplacian matrix of G. In other words:

$$L = l_{ij} = \begin{cases} -1 & \text{if } i \neq j \ \& \ a_{ij} \neq 0 \ , \\ 0 & \text{if } i \neq j \ \& \ a_{ij} = 0 \ , \\ d(v_i) & \text{if } i = j \ . \end{cases} \tag{2.10}$$

For example, the Laplacian matrix of the graph shown in Fig. 2.7 is obtained as follows:

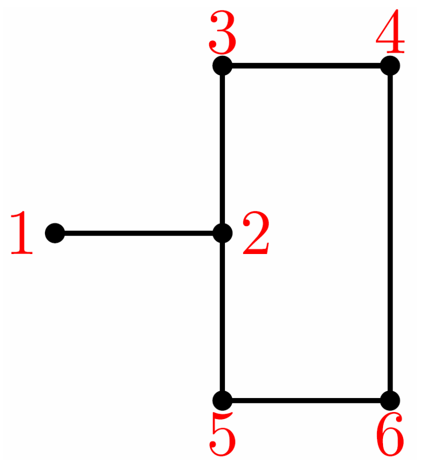


Fig. 2.7: An undirected, connected graph $G = (V, E)$.

$$A(G) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \qquad D(G) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

$$L(G) = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & -1 \\ 0 & -1 & 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

Let λ_2 denote the smallest positive eigenvalue of L. An eigenvector corresponding to λ_2 is called a *Fiedler vector*. Barnard *et al.* [15] observed that by sorting the components of the Fiedler vector in monotonically non-decreasing (or non-increasing) order, a permutation vector is obtained by which the envelope size of a sparse matrix could be reduced.

2.6 Chapter Summary

The aim of this chapter was to provide an overview on a special class of combinatorial optimisation problems, i.e., graph layout problems. These problems have been introduced and discussed from a graph-theoretic point of view in this chapter. Their historical background and applications together with a brief overview of the advantages and disadvantages of existing methods developed for addressing them have also been presented. The main focus of the chapter was on two of the most important graph layout problems, namely the bandwidth and envelope reduction problems, since they were employed as testbeds for this research.

The next chapter presents a comprehensive review of the relevant literature on hyper-heuristics. The fundamentals of genetic programming are also presented.

Chapter 3

Hyper-Heuristics and Genetic Programming

3.1 Introduction

Hyper-heuristics represent a search methodology, which is motivated by the aim of automating the process of selecting or combining simpler heuristics in order to tackle hard computational problems. Unlike meta-heuristics, hyper-heuristics operate on a search space of heuristics rather than directly on a search space of candidate solutions. Hyper-heuristics can be classified into two main categories. The first category aims to intelligently choose heuristics from a set of human designed heuristics, usually taken from the literature. The second category aims to generate new heuristics (which do not exist at present) from a set of potential heuristic components. The main focus of the work presented in this thesis is on the second category. Genetic programming, which is an evolutionary computation technique, is used in the present study in order to automatically generate graph-theoretic heuristic algorithms for the bandwidth and envelope reduction problems. The reason for choosing genetic programming is that a number of experts [13, 26, 102, 103, 104, 150] have pointed out the suitability of genetic programming over other machine learning methods for automatic heuristic generation.

This chapter is organised as follows: In Sec. 3.2, a very brief description of heuristics and meta-heuristics is presented. Sec. 3.3 provides a detailed review of the key literature on hyper-heuristics considering the two main categories of this methodology. In Sec. 3.4, a brief introduction to genetic programming is presented, which provides an explanation of its main concepts and fundamentals. Finally, Sec. 3.5 summarises this chapter.

3.2 Heuristics and Meta-Heuristics

Since hyper-heuristics are closely related to heuristics and meta-heuristics, it is useful to start with a very brief description of these methods.

According to Pearl [133], heuristics are intelligent search strategies for computer problem solving. A heuristic can also be considered as a rule-of-thumb or educated guess, which reduces the search required for finding a solution. Heuristic algorithms are designed to speed up the process of discovering a satisfactory solution. However, the optimal solution is not guaranteed to be obtained.

Meta-heuristic methods such as genetic algorithms [80], tabu search [68, 69] and simulated annealing [94] are heuristic search techniques, mostly based on natural metaphors. These methods are designed to address complex optimisation problems where classical optimisation techniques fail to be effective. According to Osman and Laporte [129], a meta-heuristic can be defined as an iterative generation process, which guides a subordinate heuristic by intelligently combining different concepts for exploring and exploiting the search space, and uses learning strategies to structure information in order to find near-optimal solutions.

Despite the fact that meta-heuristics are generally effective, there can often be some reluctance to employ them for addressing real-world problems, and in practice very simple search techniques are often preferred even if those techniques produce relatively inferior results. In Sec. 2.4 of the previous chapter, we described two main reasons for such a preference in relation to addressing graph layout problems. According to [149], other reasons might include:

1. In these kinds of search methods, the need for making a significant range of parameter or algorithm choices may cause difficulties for inexperienced users.

2. The state of the art in these techniques for coping with real world problems tends to represent problem-specific special purpose techniques, which are principally resource intensive to develop and implement.

3. Two identical runs may produce different answers to the same problem, which is mainly due to the fact that these methods often involve making some probabilistic choices.

4. There is a limited knowledge or understanding of the average or worst-case behaviour of some of these methods.

5. These techniques are typically very slow in comparison with simple heuristic methods.

Research on hyper-heuristics is an effort to answer these justifiable criticisms, as we will see in the next section.

3.3 Hyper-Heuristics

Hyper-heuristics are considered as an emerging methodology in search and optimisation. The term "hyper-heuristic" was first introduced by Cowling *et al.* [37]. According to their definition, a hyper-heuristic manages the choice of which low-level heuristic method should be applied at any given time, depending upon the characteristics of the heuristics and the region of the solution space currently under exploration. Burke *et al.* [20] also defined hyper-heuristics as "heuristics to choose heuristics". The latter definition is the most widely accepted definition of hyper-heuristics, which has frequently appeared in the literature since 2003.

Although the term hyper-heuristics is relatively new, the first research in this area dates back to the early 1960s. For instance, Fisher and Thompson [57, 58] developed a method that combined local job shop scheduling rules using a probabilistic learning technique. This method can be considered as a hyper-heuristic, since the learning algorithm first selects which of two heuristics to apply, and then the chosen heuristic selects the next job for the machine.

In a classic hyper-heuristic framework, there is a high-level method serving as central control unit and a set of low-level heuristics. The high-level method first examines each state of the target problem and then selects which low-level heuristic to apply to the problem in any particular state. Considering the fact that each problem-specific heuristic has its own weaknesses and strengths, combining a set of heuristics in such a way as to allow one heuristic to compensate for the weakness of another may lead to a better algorithmic performance [149].

A meta-heuristic operates directly on a search space of candidate solutions in order to find optimal or near-optimal solutions, while a hyper-heuristic operates on a search space of heuristics used to solve the target problem, and the aim is to find or generate high-quality heuristics for a problem, a certain class of instances of a problem or even for a particular instance. This is actually the main difference between meta-heuristics and hyper-heuristics [20, 143].

The broad aim of research on hyper-heuristics is to develop algorithms for addressing a whole range of problems, which are fast, reliable in terms of both quality and repeatability and with good worst-case behaviour, and more generally applicable compared to heuristics and meta-heuristics [149].

Hyper-heuristics can be placed into two main categories [22]. In the following subsections, these two categories together with a comprehensive review of the literature relevant to each category are presented.

3.3.1 First Category: Heuristics to Choose Heuristics

In this category, a hyper-heuristic is provided with a set of effective human-designed heuristics. These heuristics can be obtained from the literature. The hyper-heuristic then tries to choose which heuristic or sequence of heuristics to apply to a given problem instance in any particular state. In the following paragraphs, a review of the key literature on this category of hyper-heuristics is presented.

In 2001, Cowling *et al.* [37] introduced the concept of a hyper-heuristic as an approach that operates at a higher level of abstraction than current meta-heuristic approaches. In this research, they presented three different categories of hyper-heuristic approaches, namely random approaches, greedy approaches and choice-function-based approaches. Their hyper-heuristic methods do not employ problem-specific information other than that provided by a range of simple, easy and cheap to implement, knowledge-poor heuristics. Each low-level heuristic communicates with the hyper-heuristic using a common problem-independent interface architecture. This hyper-heuristic framework can either select to call a low-level heuristic in order to see what would happen if the low-level heuristic were used, or to permit the low-level heuristic to change the current solution. They applied their proposed hyper-heuristics to a real-world sales summit scheduling problem. The results reported were far superior to those provided by the previous system used to generate schedules. Some work similar to this has been reported in [20, 38, 39]

A Genetic Algorithm (GA) based hyper-heuristic (hyper-GA) for scheduling geographically distributed training staff and courses was presented by Cowling *et al.* [36]. The aim of the hyper-GA was to evolve a good-quality heuristic for each given instance of the problem and use this to find a solution by applying an appropriate ordering from a set of low-level heuristics. The hyper-GA employs a GA as the high-level method, with low-level heuristics as the genes in the chromosome. In this system, there are twelve problem-specific low-level heuristics, which are designed to accept a current solution and modify it locally in an attempt to return an improved solution. At each generation, the hyper-GA can call the set of low-level heuristics and apply them in any sequence. These low-level heuristics are considered in three groups: add, add-swap, and add-delete. The evolution results in sequences of these low-level heuristics, and these low-level heuristics are applied to the problem according to the sequence specified. The nature of the low-level heuristics applied varies from one generation to the next, and the hyper-GA seems to have some ability to determine when to call each heuristic, across a number of different problem instances. The reported results showed that the hyper-GA considerably outperformed both a GA and MA (Memetic Algorithm), and very significantly outperformed its component heuristics. Han *et al.* [74, 75, 76] also presented three different hyper-GA approaches for the trainer scheduling problem.

Burke *et al.* [21] developed a new hyper-heuristic method using Case-Based Reasoning (CBR) for solving course timetabling problems. The overall goal of their approach was to investigate CBR as a selector to choose (predict) the best (or a reasonably good) heuristic for the problem in hand according to the knowledge obtained from solving previous similar problems in order to avoid a large amount of computational time and effort on the comparison and choosing different heuristics. In their CBR system, the previous most similar cases provide information that facilitates the prediction of the best heuristic for the target case. Knowledge discovery techniques are also employed in order to extract knowledge of meaningful relationships within the case-based heuristic selector via iterative training processes on cases of course timetabling problems. There are two iterative training stages used in this system. The first stage attempts to discover the representation of cases with an appropriate set of features and weights. The second stage trains the case base such that it contains the appropriate collection of source cases. Both processes are performed iteratively with the aim of obtaining the highest accuracy on retrievals for predictions of heuristics for target cases. The results from this study indicated the possible advantages of employing knowledge discovery techniques in the course timetabling domain. This work has been extended in [33].

A tabu-search hyper-heuristic was introduced by Burke *et al.* [31]. They applied their method to two different domains of nurse scheduling and university course timetabling. Different sets of local search heuristics were employed for each of the two problems, but the hyper-heuristic was left unchanged. This hyper-heuristic maintains a ranking of its low level heuristics based on their performance, and applies the one with the highest rank at each decision point to the problem. If a heuristic is applied and does not result in a better solution, it is placed in the tabu list, and therefore it is not used for a number of iterations. The tabu search hyper-heuristic does not receive any information about the domain in which it operates. Therefore, this makes it possible to apply the hyper-heuristic to a different problem domain without the need for modification. Similar work on a tabu search hyper-heuristic was presented by Kendall and Hussin [88]. They also presented an improved version of this algorithm and applied it to a real world timetabling problem from the MARA University of Technology [89].

Kendall and Mohamad [90] proposed a hyper-heuristic system for the channel assignment problem, which is very important in the mobile communications industry. Their objective was to find the minimum frequency bandwidth considering different traffic demand distributions within the mobile network. In that research, the channel demand requirement and the minimum channel reuse distance are simultaneously considered in order to avoid the effect of call interference within the same cell or adjacent cells. In this system, there are four low level heuristics (LLHs), defined as simple local search operators, which are problem dependent. LLHs are called to produce a move from the current solution to the new solution. The proposed hyper-heuristic is responsible for the acceptance of a new solution with respect to 4 different types of acceptance criteria. These are as follows. *Accept all moves (AM)*: all the returned solutions from the LLHs are accepted. *Only improving moves (OI)*: only moves improving the current solution are accepted. *Monte Carlo (MC)*: the acceptance of the worst solution is based on a probability distribution. *Record-to-Record Travel (RRT)*: any configuration, which is not much worse than the current solution, is accepted. The idea of the RRT acceptance criterion was proposed by Dueck [52], and it is a variant of the Great Deluge acceptance criterion. Based on the experimental results reported, the RRT hyper-heuristic seems to be superior to the other hyper-heuristics proposed in this study.

An Ant algorithm hyper-heuristic for the project presentation scheduling problem was designed by Burke *et al.* [30]. This algorithm is based on a network in which vertices represent heuristics, and directional transitional arcs exist between heuristics if it is possible to apply one immediately after the other. A number of ants are then created, each of which represents a hyper-heuristic agent supplied with an initial solution in the solution space and access to the heuristics and evaluation functions. The ants are scattered uniformly among the vertices of the network.

The ants then build a sequence of heuristics (a path) by traversing the network. At each decision point, each ant chooses the next vertex that it will visit, traverses the arc to that vertex, and applies the heuristic represented by that vertex to its current solution. Vertices and arcs may recur within the path. After visiting a certain number of heuristics, the ant pauses to evaluate the traversed path and to lay an amount of pheromone on each edge in that path according to the improvement in the quality of the solution during the entire path. Each ant then proceeds to generate its next path. This algorithm continues for as many cycles as required. Note that according to [30], a cycle is defined as the time taken between all ants beginning their paths and all ants completing their paths. Another ant algorithm hyper-heuristic has been presented in [42].

Burke *et al.* [32] proposed a graph-based hyper-heuristic for educational timetabling problems. In this study, an investigation of a simple generic hyper-heuristic approach on a set of broadly used constructive graph colouring heuristics in timetabling is presented. A tabu search approach is utilised within the hyper-heuristic framework to search for permutations of graph heuristics. These heuristics are then used for generating timetables in exam and course timetabling problems. In fact, their method has two stages, and the tabu search employs permutations upon a different number of graph heuristics during these stages. The proposed hyper-heuristics were tested on both exam and course benchmark timetabling problems and were compared against state-of-the-art approaches. The results were within the range of the best results reported in the literature.

Simulated annealing was used as a hyper-heuristic in [51] for determining shipper sizes for storage and transportation. This simulated annealing algorithm is based on the tabu search hyper-heuristic presented in [31]. The difference is that when the heuristic is chosen by the tabu search, the move it generates is accepted according to the simulated annealing algorithm. Issues related to memory length in a simulated annealing hyper-heuristic are addressed in [11].

Bai *et al.* [12] investigated heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. For brevity, here we only focus on three types of hyper-heuristics applied to the above-mentioned problem. *Tabu search-based hyper-heuristic (TSHH)*: the main idea behind this algorithm is the incorporation of a tabu list into the heuristic selection mechanism which prevents the selection of some low-level heuristics at certain stages of the search. *Simulated annealing hyper-heuristic (SAHH)*: in this algorithm, a simulated annealing algorithm is used as an acceptance criterion. At each iteration, the algorithm selects a heuristic from the set of low-level heuristics available and applies it to the current solution. If the solution generated by this heuristic is better than the current solution, it is accepted. Otherwise, it is accepted according to a Metropolis probability. The temperature of the simulated annealing is then modified. When the stopping conditions are met, the system terminates and produces the best solution found so far. *Tabu search simulated annealing hyper-heuristic (TSSAHH)*: this hybrid hyper-heuristic is, in fact, a compound of the tabu search hyper-heuristic and the simulated annealing hyper-heuristic described above in which the low-level heuristics are selected using the same rule as in TSHH, but candidate solutions are accepted according to the simulated annealing probability (similar to [51]). In this study, TSSAHH and SAHH performed significantly better than the other algorithms under test.

3.3.2 Second Category: Heuristics to Generate Heuristics

This category of hyper-heuristics aims to generate new heuristics by intelligently combining potential heuristic components, supplied by a human designer. The main focus of the work presented in this thesis is on the second category, which is less well studied in the literature. There is a clear distinction between this category and the first category described in Sec. 3.3.1, in which the hyper-heuristic is provided with a set of complete functioning heuristics, and it must then choose between them. The newly generated heuristics may be "disposable" in the sense that they are produced to address just one problem instance (or a single set of problems). In other words, they are not intended for use on unseen problem instances. Alternatively, the generated heuristics may be "reusable", meaning that they are produced for the purpose of reusing them on unseen instances of a certain class of problems.

This approach has a number of potential advantages. Considering the fact that every problem instance is different, a new heuristic or a variation of a previously created heuristic may be required in order to find the best possible solution to an instance of a problem. There is no doubt that it would be inefficient or even impossible for a human to manually create a new heuristic for every problem instance. In general, human-created heuristics are designed to effectively address many problem instances rather than only one problem instance. For example, "best-fit" is a general human-designed heuristic for addressing one dimensional bin packing problem [91], and performs well on a wide range of bin packing instances. However, over a set of instances with piece sizes defined over a certain distribution, the "best-fit" algorithm can be easily outperformed by heuristics, which are tailored to that distribution of piece sizes [27].

A reasonable solution to the problem mentioned above is automating the heuristic design process by a computer system. This offers the chance to simply specify the range of problem instances that a heuristic will be applicable to, and then generate that heuristic with minimal human effort spent in the process. Such a heuristic may be cable of producing better solutions than any human-created heuristic, because it is designed specifically to address that range of problem instances. Although a highly specialised heuristic can produce the best possible solution to an instance or a single set of problems, in practice, heuristic algorithms, capable of effectively addressing a wider range of problems are more desirable. This is, in fact, in agreement with the broad aim of research on hyper-heuristics. In the rest of this section, a review of the key literature on the second category of hyper-heuristics is presented.

Fukunaga [59, 60] proposed an automated heuristic discovery system for the SAT problem. The most effective heuristics for the SAT problem can be expressed as different combinations of a particular set of components. It is argued in [59] that humans are particularly good at identifying suitable heuristic components, but the process of combining them can benefit from automation. In that paper, an evolutionary algorithm (i.e., a heavily modified genetic programming system named CLASS) is used to evolve human-competitive heuristics which consist of a number potential components. This research can be considered the first to examine a genetic programming system as a hyper-heuristic. Some issues related to [59] were addressed in [60] by implementing the system in Lisp for faster evaluation, and demonstrating that good quality heuristics could still be generated without initialising the population with complex hand-coded heuristics and components.

Tavares *et al.* [158] presented a methodology for evolving an evolutionary algorithm. They specified the main components of a generic evolutionary algorithm, including initialisation, selection, and the use of genetic operators, and described how each of these components could be evolved individually by a genetic programming approach. Oltean [127] also introduced a linear genetic programming approach in order to generate evolutionary algorithms. A standard individual in a linear genetic programming system represents a series of instructions which manipulate values in a memory array. The memory positions are referred to as "registers". He describes an evolutionary algorithm population as the memory array, with each member of the population stored in one register. The genetic operators are the instructions operating on the memory array.

Burke *et al.* [24] employed genetic programming in order to evolve Bin-Packing heuristics. In this approach, a genetic programming system selects between a set of low level building blocks to create a heuristic that performs well in the environment provided for it. In this case, the building blocks are the function and terminal sets, and the environment is the data set provided for the system. Their system evolves a control program which decides whether or not to put a given piece into a given bin. An individual in the population is evaluated by rating the results produced when the algorithm is run. It should be noted that each individual in the population is not constrained by whether or not it is allowed to put the current piece in the current bin. However, this may result in the creation of an illegal solution with a high penalty. In this study, they indicated that a heuristic created by a human, namely the "first fit" heuristic (a well-known human-designed heuristic for the bin-packing problem), could also be evolved by genetic programming. They generated the heuristic without any human input except for the specification of the building-blocks. It was reported that the generated heuristics were the same as high quality heuristics created by humans. Some similar work on bin-packing problem has been reported in [3, 23, 27, 28, 29, 83].

Poli *et al.* [144] also proposed to use a form of genetic programming as a hyper-heuristic to attack the one-dimensional bin-packing problem with discrete item sizes. The structure in which their heuristics operate is based on matching the item-size histogram with the bin-gap histogram, and it is motivated by the observation that space is wasted when placing a piece in a bin leaves a smaller available space than the size of the smallest piece still to be packed. This method is controlled by a heuristic function which decides how to prioritise items in order to minimise the difference between histograms. They employed genetic programming to evolve such a function. This work differs from that presented in [24], since in [144] a form of linear register-based genetic programming system is used, and the problem addressed is offline bin packing rather than online. Another hyper-heuristic based on linear genetic programming was introduced by Keller and Poli [86]. They used their genetic programming system as a hyper-heuristic for addressing Travelling Salesman Problem (TSP) (see also [128] for another study on the TSP).

Bader-El-Den and Poli [5] presented a hyper-heuristic based on genetic programming for evolving local-search 3-SAT heuristics. The aim of this research was to produce disposable heuristics, which are evolved and used for a specific subset of instances of a problem as defined earlier. The genetic programming system used in this study was grammar-based. In order to construct an appropriate grammar to guide the hyper-heuristic, a number of well-know local-search heuristics were employed and decomposed into their basic components. These heuristics are as follows: *GSAT* [153], *HSAT* [62], *GWSAT* [151] and *WalkSAT* [152]. Next, a simple but flexible grammar was designed in order to give the hyper-heuristic enough freedom for evolving new heuristics. They compared their method with other well-known evolutionary and local search heuristics and reported that the proposed approach was competitive with the methods under test. Some work similar to this has also been presented in [6, 7, 9]. Subsequent studies by the same team of researchers indicated that the hyper-heuristic system proposed in [5] could also evolve constructive heuristics for timetabling problems [8, 10].

Kumar *et al.* [106] introduced a genetic programming system for evolving bi-objective knapsack heuristics. This is the first work in which heuristics for a multi-objective problem have been automatically generated by a hyper-heuristic approach. The technique used in that research has several similarities with the bin packing methodology presented in [82]. Further work by Kumar *et al.* [105] has indicated that heuristics can also be generated by genetic programming for a multi-objective minimum spanning tree problem.

Recently, Burke *et al.* [25] have proposed a grammatical evolution approach in order to automatically evolve good quality local search heuristics, which are capable of maintaining their performance on new problem instances. The main aim of this research was to automatically generate effective neighbourhood move operators producing landscapes where it was easy to reach a good quality solution. The evolved move operators could only accept moves that did not deteriorate the solution. The results reported indicated that this approach was effective.

3.4 Genetic Programming

Genetic Programming (GP) [102, 122, 143] is an evolutionary algorithm inspired by biological evolution. GP is capable of solving problems automatically without the need for the user to specify the structure of the solutions in advance. In fact, it is a specialisation of Genetic Algorithms (GAs) in which the evolving individuals are computer programs rather than fixed-length vectors of parameters. GP has a random nature identical to GAs and is able to stochastically transform populations of programs into new and, hopefully better, populations of programs. GP has been utilised as an automatic programming tool, a machine learning tool, and an automatic problem-solving engine since its early beginning.

As described in Sec. 3.3.2, GP has also been successfully used as a hyper-heuristic. For example, GP has evolved competitive SAT solvers [9, 59, 92], state-of-the-art or better than state-of-the-art bin packing algorithms [24, 144], particle swarm optimisers [142], evolutionary algorithms [127], TSP solvers [86, 128], 2-D strip packing heuristics [23], bi-objective knapsack heuristics [106] and local search heuristics [25]. In this thesis, GP is employed as a hyper-heuristic in order to automatically generate graph-theoretic heuristic algorithms for the bandwidth and envelope reduction problems. The fundamentals of GP are described in detail in the following sections.

3.4.1 Program Representation

In GP, individuals (computer programs) that form the population are generally represented as tree structures. A program tree is composed of *terminals* and *functions*. The terminals are the leaves of the tree (nodes without branches), and as stated in [143] they typically include:

- Variables (e.g., x, y, z),
- Constant values (e.g., 3, 6.5, Boolean constant NIL),
- Functions with no arguments (e.g., rand, go_left).

The functions are the internal nodes of the tree (nodes with branches), and according to [102] they may comprise:

- Arithmetic operations (e.g., $+$, $-$, $*$),
- Mathematical functions (e.g., sin, cos, exp, log),
- Boolean operations (e.g., AND, OR, NOT),
- Conditional operators (e.g., If-Then-Else),
- Functions causing iteration (e.g., FOR, REPEAT, WHILE),
- Any of other domain-specific functions defined by the designer.

Note that unlike the leaf nodes having no successors, the internal nodes can have one or more children nodes, and they return a value, which is the result of performing different operations (e.g., arithmetic, Boolean etc) on the values returned

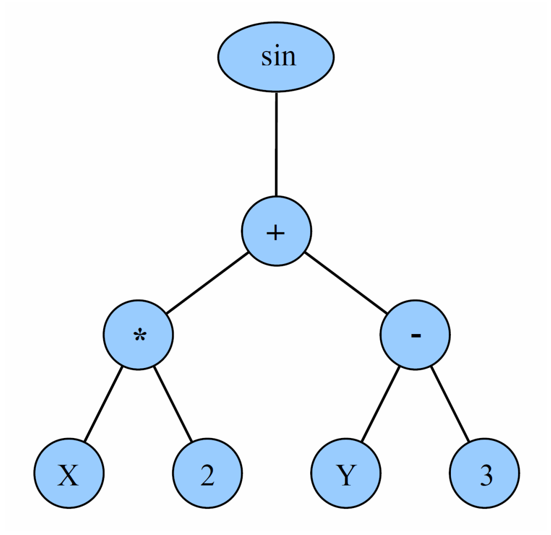


Fig. 3.1: Tree representation of the program $sin((x*2)+(y-3))$.

by their children nodes. The functions and terminals selected for generating program trees together form the *primitive set* of a GP system. Fig. 3.1 shows an example in which the tree representation of the program $sin((x*2)+(y-3))$ is illustrated. In this example, terminals $= \{x, y, 2, 3\}$, functions $= \{+, -, *, sin\}$ and primitive set $= \{x, y, 2, 3, +, -, *, sin\}$.

Tree representations are the most common type of representation in standard GP. However, there are other forms of GP in which programs are represented in different ways. For instance, linear GP [18, 134] in which programs are linear sequences of instructions, and graph-based GP [138, 139] that evolves graph-like programs.

3.4.2 Initialisation

GP is similar to other evolutionary algorithms in terms of the initialisation of the population, in that the individuals (program trees) in the initial population are normally generated randomly. There are different approaches for the generation of a random initial population in GP. The full, grow and ramped half-and-half methods are the most widely used tree-creation techniques [102, 143]. For the three approaches mentioned here, a predefined maximum depth needs to be specified in order to limit the size of the initial trees.

The full method randomly selects a function from the function set for every node until the maximum depth is reached, where only a terminal can be selected. This leads to the generation of a tree in which all of the terminal nodes (leaves) are at the same depth. On the contrary, the grow method randomly selects a primitive (function or terminal) from the primitive set for every node until the maximum depth is reached, where a terminal must be selected. This results in trees with varied structures. The ramped half-and-half method, proposed by Koza [102] is actually a combination of the full and grow methods in which the first half of the initial population is generated with the full method, and the second half is generated with the grow method. This is performed using a range of depth limits to guarantee that trees having a variety of sizes and shapes are generated [143].

3.4.3 Fitness Evaluation

By initialising the population, in fact, the search space that GP will explore is defined. At this stage, computer programs in the population need to be assessed in order to determine their merits. This is, of course, a problem specific issue. The fitness measure is actually an indication of how good a computer program is at addressing a given problem. This quantity is then used in the selection phase as we will see in the next section.

There are a number of different ways of measuring the fitness. For instance, it can be measured in terms of the amount of error between system's output and a desired output, the amount of time needed for a system to reach a desired state, the accuracy of a program in identifying patterns or categorising objects, the payoff that a game-playing program produces, the compliance of a structure with user-defined design standards and so on [143].

Since individuals are represented as computer programs in a GP system, the obvious method of testing the effectiveness of the solutions is to execute all the programs in the population. To achieve this goal, the most common strategy is to incorporate an interpreter into the algorithm to evaluate the evolved programs. In the process of interpreting a given program tree, all nodes in the tree should be executed in such a way as to ensure that nodes are executed only after the value of their arguments is determined. This is normally achieved by traversing the tree in a recursive way starting from the root node, and suspending the assessment of each node until the values of its children (i.e., arguments) are determined [143].

3.4.4 Selection

In the selection phase, individual programs are probabilistically selected from the population based on their fitness values. This means that better computer programs are normally favoured over inferior programs. The genetic operators (e.g., crossover and mutation) are then applied to the selected individuals for creating a new generation of individual programs.

Tournament selection [102, 143], is the most commonly used method for the selection of individuals in a GP system. However, any standard evolutionary-computation-based selection method can also be employed for this purpose.

In tournament selection, a group of individuals (determined by the tournament size parameter) are chosen randomly from the population, and the one with the best fitness value is then selected to be the parent. This selection method automatically rescales fitness, and therefore the selection pressure on the population remains constant [143].

3.4.5 Genetic Operations

In GP, the genetic operations are applied to the program trees chosen by a selection process for creating a new generation of individual programs. The fundamental genetic operations in a GP system are as follows:

Crossover

Subtree crossover [102, 143] is the most common form of crossover in GP. This operator requires two parent individuals and produces one offspring. A crossover point (a node) is chosen at random in each parent, and as a result two subtrees rooted at the crossover points are defined. The offspring is then created by replacing the first parent's subtree with the second parent's subtree [143]. It should be noted that in this process copies of the parents are normally utilised in order to avoid disrupting the original parents. Note also that it is possible to define a crossover operation producing two offspring, but this is not common in practice. Fig. 3.2 illustrates an example of subtree crossover.

Mutation

In GP, subtree mutation and point mutation [102, 143] are the most widely used mutation techniques. The subtree mutation operator needs one parent individual and generates one offspring. A mutation point (a node) is chosen randomly in the parent. The subtree defined by the mutation point is then deleted, and a new subtree is randomly generated in its place [143]. Fig. 3.3 illustrates an example of subtree mutation. The point mutation is approximately the same as the bit-flip mutation [70], which is generally used in genetic algorithms. In point mutation, first, a random node is chosen. A new primitive with the same arity is then selected

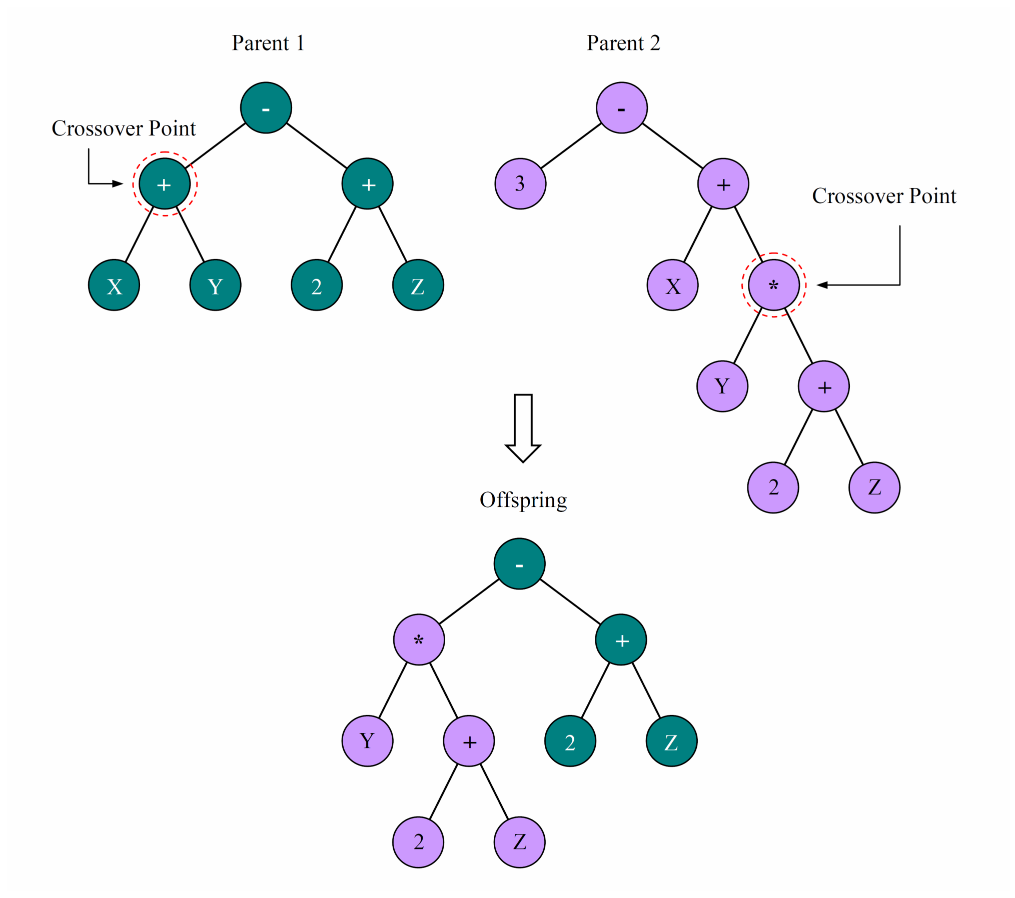


Fig. 3.2: An example of subtree crossover.

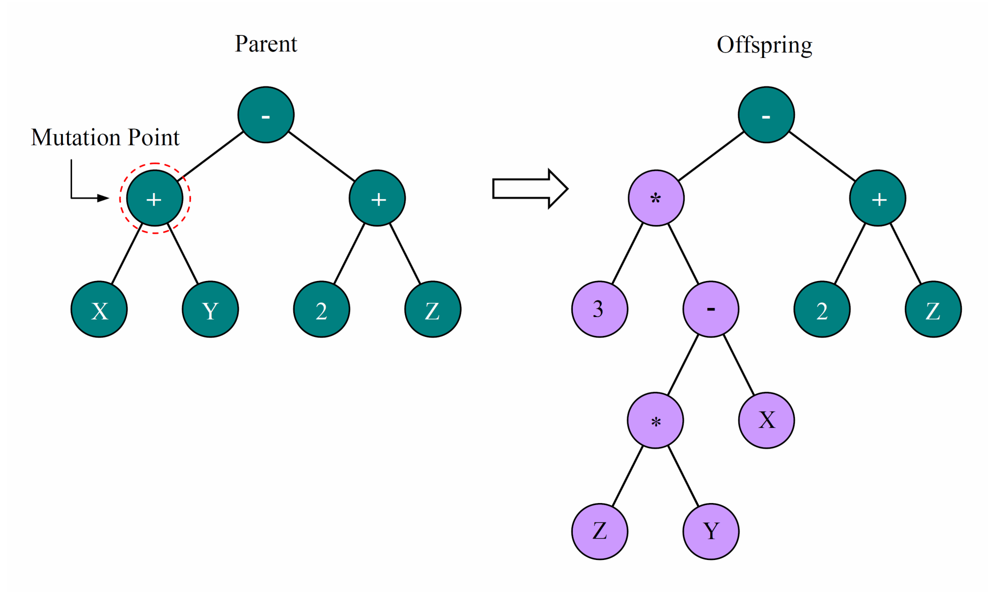


Fig. 3.3: An example of subtree mutation.

at random from the primitive set, and the primitive stored in the random node is replaced with the new primitive.

Reproduction

The reproduction operation copies a single individual, probabilistically selected based on fitness, into the next generation of the population without any alteration [143]. This operator provides a mechanism for retaining good program trees in the population.

Elitism

During a GP run, there is a high probability of losing the best program tree(s) in the population after applying the genetic operations. Elitism is used to prevent this problem by directly copying one or a few program trees having the best fitness value(s) in the current population, into the new population [143].

3.4.6 Termination and Solution Designation

The termination condition may be specified as a maximum number of generations or a problem-specific success predicate [143]. Normally, the best-so-far individual is then designated as the result of a run. However, it may be required to return additional individuals and data if necessary for the target problem [143].

3.4.7 Flowchart of Genetic Programming

In order to apply a GP system to a given problem, first, the terminal set, the function set, the fitness measure, the run parameters (e.g., the population size, the maximum program size etc) and the termination criterion should be determined, and after that a run of GP can be launched [143]. GP is problem-independent in the sense that the flowchart specifying the basic sequence of executional steps is not modified for each new run or each new problem [102]. In addition, there is usually no human intervention or interaction during a run of GP. Fig. 3.4 is a flowchart illustrating the executional steps of a run of GP.

3.5 Chapter Summary

The main aim of this chapter was to give an overview of hyper-heuristics: an emerging methodology in search and optimisation. In this chapter, first, heuristics and meta-heuristics have been briefly introduced because of their close relationship with hyper-heuristics. Then, a comprehensive review of the key literature on hyper-heuristics considering the two main categories of this methodology ("heuristics to choose heuristics" and "heuristics to generate heuristics") has been provided. Because in this thesis genetic programming is used both as a meta-heuristic and as a hyper-heuristic, its main concepts and fundamentals have also been presented in

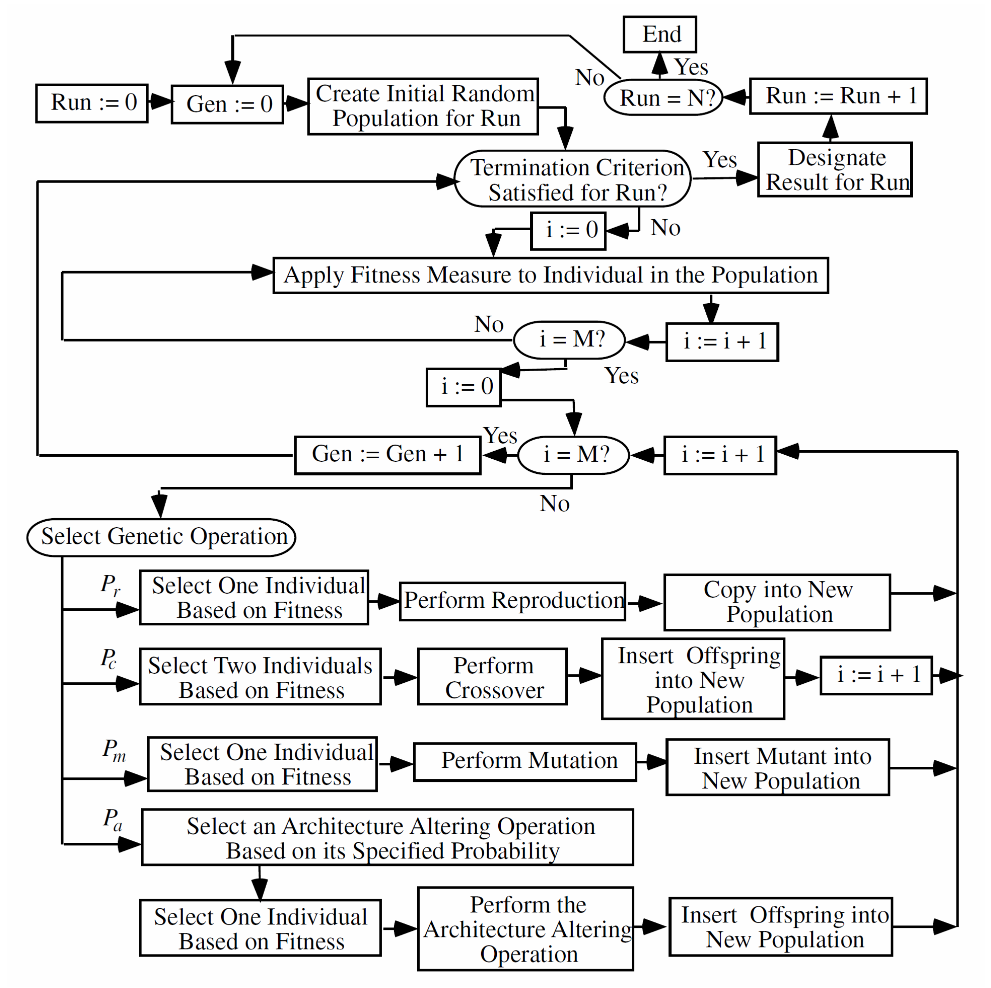

End
Yes
No
Run := 0
Gen := 0
Create Initial Random Population for Run
Run = N?
Run := Run + 1
Termination Criterion Satisfied for Run?
Yes
Designate Result for Run
i := 0
No
Apply Fitness Measure to Individual in the Population
No
i = M?
i := i + 1
i := 0
Yes
Yes
Gen := Gen + 1
i = M?
i := i + 1
No
Select Genetic Operation
P_r
Select One Individual Based on Fitness
Perform Reproduction
Copy into New Population
P_c
Select Two Individuals Based on Fitness
Perform Crossover
Insert Offspring into New Population
i := i + 1
P_m
Select One Individual Based on Fitness
Perform Mutation
Insert Mutant into New Population
P_a
Select an Architecture Altering Operation Based on its Specified Probability
Select One Individual Based on Fitness
Perform the Architecture Altering Operation
Insert Offspring into New Population

Fig. 3.4: Flowchart of GP (reproduced from [102]).

this chapter.

The next chapter describes a specialised genetic programming system, designed to act as the central organising element of the genetic hyper-heuristics — our own approach to the graph layout problems. The application of the proposed genetic programming system as a meta-heuristic to the bandwidth and envelope reduction problems is then discussed in detail.

Chapter 4

A Specialised Genetic Programming System for Graph Layout Problems

4.1 Introduction

In the previous chapter, hyper-heuristics were defined, and a comprehensive review of the key literature on this methodology was provided. Considering the importance of Genetic Programming (GP) in the current study, a brief introduction to this evolutionary computation technique, providing an explanation of the main concepts was also presented. This chapter describes our initial work towards developing *Genetic Hyper-Heuristics* (the main contribution of this thesis). The central organising element of the genetic hyper-heuristics is a specialised GP system, which incorporates a large number of components required for addressing graph layout problems. The proposed GP system can be utilised both as a meta-heuristic and as a hyper-heuristic in order to solve this class of problems. In this chapter, we describe the system and its utilisation as a meta-heuristic in detail. Both as a sanity check and also due to a total lack of any previous application of GP to graph layout problems, we employ this GP system as a meta-heuristic in order to address the bandwidth and envelope reduction problems (BRP and ERP, see Sec. 2.5 for more details). This allows us to explore the benefits and drawbacks of a meta-heuristic GP approach to these problems. The results obtained on a wide-ranging set of standard benchmark matrices from the Harwell-Boeing sparse matrix collection show that the proposed method compares very favourably with state-of-the-art algorithms from the literature.

This chapter is organised as follows: In Sec. 4.2, our specialised GP system for addressing graph layout problems is described in detail. Sec. 4.3 presents the application of the proposed GP system as a meta-heuristic to the BRP followed by the related experimental results. Sec. 4.4 details the application of the proposed GP system as a meta-heuristic to the ERP and provides a description of our experiments and the interpretation and discussion of the results. Finally, Sec. 4.5 summarises this chapter.

4.2 Our Specialised GP System

We designed a tree-based GP system with additional decoding steps required for graph layout problems and our particular choice of representation for its solutions. As we will see later, this GP system can be used both as a meta-heuristic and as a hyper-heuristic in order to address graph layout problems. The system is implemented in C# utilising the Microsoft .NET technology. The system also works under Linux using Mono. A high-level description of the main operations of the proposed GP system is given in Algorithm 2. Below, the elements of the system are described in detail.

Algorithm 2 The main operations of the proposed GP system.

```
1: Randomly generate an initial population of programs.
2: repeat
3:     Execute each program in the population.
4:     Create the permutation represented by each program.
5:     Apply each permutation to the adjacency list of the initial graph (or the initial ma-
       trix) and generate new adjacency lists.
6:     Compute the layout cost for the adjacency lists obtained in the previous step.
7:     Determine the fitness value of each program in the population.
8:     Perform selection to choose individual program(s) from the population based on
       fitness to participate in genetic operations.
9:     Create a new generation of individual program(s) by applying genetic operations
       with specified probabilities.
10: until the maximum number of generations is reached.
11: return the permutation represented by the best program in the last generation.
```

4.2.1 Representation

Individuals in our GP system are tree-like expressions. However, for efficiency reasons, they are internally stored as linear arrays using a flattened representation for trees. One may wonder then how we transform program trees into permutations, which are needed to actually solve graph layout problems. We accomplish this using a simple trick: we interpret the *outputs* produced by a program tree when executed *over a set of fitness cases* as a permutation.

To understand how this works, let us imagine that GP was used to evolve a function of one variable X, and that the fitness cases tested the function for input values X_1, X_2, X_3, X_4. If a GP program produced the values 2.3, 3.5, 0.7 and -0.4, we could interpret them as a permutation by adopting the following approach.

First, we associate with each element in the sequence its position. Thus, in the sequence mentioned above 2.3 is in first position, 3.5 is in second position, etc. This can be expressed more explicitly as $2.3_1,\ 3.5_2,\ 0.7_3,\ -0.4_4$, where the subscripts refer to the positions associated with each element in the sequence. Then, we sort the sequence in ascending order, but we still remember the original positions. This leads to the sequence $-0.4_4,\ 0.7_3,\ 2.3_1,\ 3.5_2$. We finally construct the permutation from the subscripts, which can be expressed as 4, 3, 1, 2.

In this study, we adopt the procedure explained above for the representation of permutations. This decoding process will be described in more detail and further exemplified in Sec. 4.2.3.

4.2.2 Generating an Initial Population

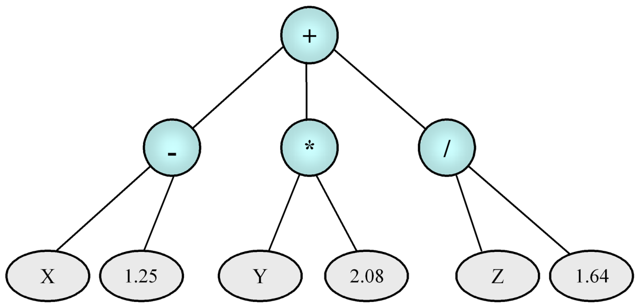
The initial population in our GP system is generated randomly (Algorithm 2, step 1) using a modified version of the ramped half-and-half method (see Sec. 3.4.2 for more details). In our system, during the process of tree initialisation, terminals are not drawn with equal probability from the terminal set. Instead, we artificially increase the chance of selecting the variables, ensuring that each program tree contains at least one variable. Trees without variables are a problem for our system because they produce a constant output and, thus, they represent the permutation $\sigma = (1, 2, \cdots, n)$. Such a permutation is useless, since it produces no change in the layout cost.

4.2.3 Fitness Evaluation

Since, individuals are represented as computer programs in a GP system, the obvious method of testing the effectiveness of the solutions is to execute all the programs in the population (Algorithm 2, step 3). As briefly mentioned in Sec. 4.2.1, in the GP system described, we need to transform the individuals (program trees) in the population into permutations, which are used for solving graph layout problems. The decoding process proposed for the creation of a permutation from a given program tree (Algorithm 2, step 4) is as follows.

For each program tree, the interpreter is called N times, where N is the number of nodes of a given graph or the dimension of a given matrix. Each call of the interpreter executes the selected program with respect to the different values of the independent variables. The outputs obtained from each execution of the given program tree are stored in a one-dimensional array. This array is then sorted in ascending order while also recording the position that each element originally had in the unsorted array. Reading such positions sequentially from the sorted array produces the permutation associated with the original program tree. The decoding process, as explained above, is exemplified for a 6×6 matrix (or its corresponding 6-node graph) in Fig. 4.1.

Next, the permutations obtained from the previous phase should be applied to the adjacency list of the initial graph (or the initial matrix), leading to the generation of new adjacency lists (Algorithm 2, step 5). The layout cost for the adjacency lists obtained is then computed (Algorithm 2, step 6). A list of well-known graph layout problems together with their cost functions was presented in Table 2.1. Subsequently, in order to determine the fitness value of each program in the population (Algorithm 2, step 7), the standard objective function (or a better figure of merit) of the related graph layout problem can be utilised. For example, the standard objective function of the bandwidth and envelope reduction problems (our testbeds in this study) are given in Eq. (2.5) and Eq. (2.9) respectively.



(a)

Order	X	Y	Z	Execution results
1	0.81	0.28	0.78	0.6180
2	-0.25	0.46	-0.34	-0.7505
3	-0.13	0.58	0.75	0.2837
4	0.56	-0.24	-0.19	-1.3050
5	0.74	-0.92	0.26	-2.2651
6	-0.66	0.43	-0.11	-1.0826

(b)

1	2	3	4	5	6
0.6180	-0.7505	0.2837	-1.3050	-2.2651	-1.0826

(c)

5	4	6	2	3	1
-2.2651	-1.3050	-1.0826	-0.7505	0.2837	0.6180

(d)

1	2	3	4	5	6
5	4	6	2	3	1

(e)

Fig. 4.1: Example illustrating the process of generating a permutation from a program tree for a 6×6 matrix or its corresponding 6-node graph: (a) a program tree with three input variables X, Y and Z, (b) six fitness cases and the corresponding outputs produced by the program tree, (c) execution results stored in a one-dimensional array, (d) sorted results, (e) the permutation generated from (a).

4.2.4 Selection and Genetic Operations

In our GP system, tournament selection is used to choose individual program(s) from the population based on fitness to participate in genetic operations (Algorithm 2, step 8). New individual programs are created by applying the genetic operations of reproduction, sub-tree crossover and point mutation with specified probabilities (Algorithm 2, step 9). Elitism is also used to ensure that the best individual in one generation is transferred unaltered to the next.

4.2.5 Termination

The termination criterion used is based on the predetermined maximum number of generations to be run (Algorithm 2, step 10). Finally, the permutation extracted from the best program tree appearing in the last generation is designated as the final result of a run (Algorithm 2, step 11). Because we use elitism as mentioned earlier, this also corresponds to the best program evolved throughout the run.

4.3 Application of our GP System to the BRP

Our initial experiments to test the effectiveness of the proposed GP system were carried out on the Bandwidth Reduction Problem (BRP) (for more details, see Sec. 2.5.1). In order to address the BRP, we employed the GP system presented in Sec. 4.2 as a meta-heuristic. In that case, our GP system operates directly on a search space of candidate solutions, i.e., permutations represented by program trees. The primitive set used includes the functions $+$, $-$, $\times$, % (protected division), sin and abs. The terminals include three floating-point input variables X, Y and Z as well as $c_1, \cdots, c_{100}$ which are uniformly-distributed random constants with floating-point values in the interval $[-5.0, +5.0]$.

The values of X, Y and Z for each fitness case are carefully chosen during the initialisation of the system in such a way as to maximise the chance that their values across the fitness cases represent suitable permutations. This is performed with the aim of assisting the GP system by biasing its search. More specifically, the X values were set to be the components of the eigenvector associated with the first non-zero eigenvalue of the Laplacian matrix (see Sec. 2.5.4) related to the matrix to be optimised. The incorporation of such components into the GP system as the values of an external input is motivated by the concept of the algebraic connectivity of graphs (for more details, see [15, 46, 56]). The values of Y and Z, instead, were obtained by using the notion of the level structures (see Sec. 2.5.4), which is at the basis of other high-performance BRP and ERP solvers, e.g., the CM, GPS and RCM algorithms. We used the approach followed in [116, 124], in which the initial solutions were generated by picking random nodes in a graph and building permutations using their associated level structures. In other words, for initialising Y and Z, we picked two random nodes and built their level structures. By traversing these structures, we constructed two permutations. Then, we converted these permutations into two floating point arrays such that if sorted as explained in Sec. 4.2.1, they would produce those permutations back. Finally, we used such arrays to provide the fitness case inputs associated with the Y and Z variables.

It should be noted that the time required for the process of the initialisation of the independent variables X, Y and Z is negligible compared to the total computational time.

The standard objective function for the BRP is simply that given in Eq. (2.5) and its calculation is straightforward. However, although this is precisely the value that needs to be minimised, it is not an ideal fitness function for a metaheuristic search. The main reason for this is that there may be many candidate solutions which have formally the same bandwidth, but which are quite different, with some being much closer to better solutions than others.

For example, in Fig. 4.2, at first sight it might seem that labeling 1 and 2 have no priority over each other considering the bandwidth measures produced, which are both equal to 4. However, there is no doubt that the labeling 1 is much closer to an ideal solution (labeling 3) than labeling 2. In fact, by a simple exchange of vertices 1 and 6, we can easily obtain labeling 3 from labeling 1.

A more effective approach is to use a fitness function which incorporates information about how close the candidate solution is to better areas of the search space. As suggested in [97], the incorporation of the profile into the fitness function used for the BRP can help capture this information. Therefore, in this work, we used the product of the bandwidth and profile as our fitness function, i.e.,

$$f(p) = B(A_{\sigma(p)}) \times P(A_{\sigma(p)}) \quad . \tag{4.1}$$

where $B(A)$ and $P(A)$ are the bandwidth and profile (see Sec. 2.5 for details) of a matrix A obtained after applying $\sigma(p)$, which is the permutation associated with program tree p.

4.3.1 Experimental Results

In the experiments conducted, we used a set of 15 instances from the well-known Harwell-Boeing sparse matrix collection (available from `http://math.nist.gov/MatrixMarket/data/Harwell-Boeing`). As mentioned briefly in Sec. 1.2, this collection includes benchmark matrices arising from problems in linear systems, least squares and eigenvalue calculations.

In order to evaluate the performance of our GP method, we compared it against two high-performance methods. Firstly, we used the RCM algorithm contained in the MATLAB library (RCMM). This is based closely on the SPARSPAK implementation described by George and Liu [65]. This algorithm is still one of the best

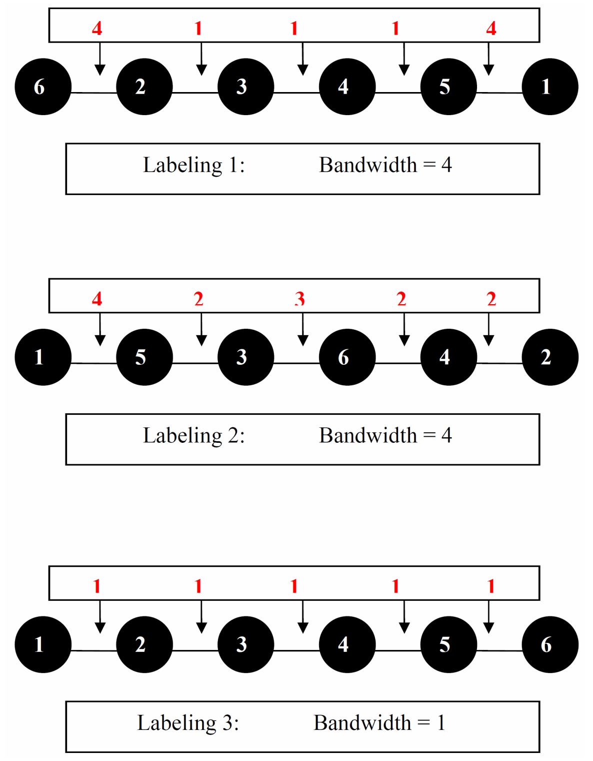

4
1
1
1
4
6
2
3
4
5
1
Labeling 1: Bandwidth = 4
4
2
3
2
2
1
5
3
6
4
2
Labeling 2: Bandwidth = 4
1
1
1
1
1
1
2
3
4
5
6
Labeling 3: Bandwidth = 1

Fig. 4.2: The effect of different labeling of a graph on bandwidth reduction.

and most widely used methods for the BRP. Indeed, the results reported in [54] and [115] indicate that the RCMM is also superior to the well-known GPS algorithm. In addition, we compared our method with the widely-used spectral analysis approach presented in [15].

The parameters of the runs are given in Table 4.1. Note that we do not claim any optimality in relation to the parameters reported. They were selected from a range of possible initialisation parameters, after conducting a number of experiments on the test problems, considering both the quality of solutions and run times. In general, the parameter set used in the experiments produces good-quality solutions in reasonable times. Note also that by increasing the population size and the maxi-

Table 4.1: Parameters used in our experiments for assessing the performance of the proposed GP system employed as a meta-heuristic for the BRP and ERP.

Parameter	Value
Maximum Number of Generations	100
Maximum Length of any GP Program	500
Maximum Depth of Initial Programs	3
Population Size	1000
Tournament Size	5
Elitism Rate	0.1%
Reproduction Rate	0.9%
Crossover Rate	70%
Mutation Rate	29%
Mutation Per Node	0.05%

mum length of each program tree, considerably better results could be produced by our GP system, but this also increases the execution time of the algorithm.

All the experiments were performed on an Intel Celeron(M) Processor 1.86 GHz. Each method was tested on our set of 15 benchmark matrices, which had sizes of up to 100×100. Considering the non-deterministic nature of GP, 10 independent runs were executed for each of the selected benchmark instances and performance values (bandwidth values) were averaged across such runs. Table 4.2 shows a performance comparison of the algorithms under test.

A simple inspection of the results of the experiments reported in Table 4.2 reveals that our GP approach is superior to both the RCMM algorithm and the spectral algorithm with respect to the mean of the bandwidth values and the number of the best results obtained. We also performed a paired Wilcoxon signed-rank

Table 4.2: Performance comparison of our GP approach with the RCMM and Spectral algorithms.

Harwell-Boeing Matrix	Dimension	RCMM $B(A)$	Spectral $B(A)$	GP Approach Mean $B(A)$ (10 runs)
ash85	85×85	13	17	**12.0**
bcspwr01	39×39	*5*	11	*5.0*
bcspwr02	49×49	13	12	**10.4**
bcsstk01	48×48	27	**20**	24.5
can_24	24×24	7	6	**5.6**
can_61	61×61	19	14	**13.4**
can_62	62×62	9	10	**8.0**
can_73	73×73	27	27	**23.3**
can_96	96×96	23	18	**15.6**
dwt_59	59×59	8	10	**7.2**
dwt_66	66×66	*3*	*3*	*3.0*
dwt_72	72×72	7	12	**6.0**
dwt_87	87×87	17	19	**13.3**
lap_25	25×25	9	7	**6.0**
nos4	100×100	12	**11**	11.3
Mean of $B(A)$ values		13.27	13.13	10.97
Standard deviation		7.73	5.98	6.16
Standard error of the mean		2.00	1.54	1.59

$B(A)$ = bandwidth of a matrix A

Best results are shown in boldface, while ties are shown in italics.

test in order to determine the statistical significance of the results and obtained a two-tailed p-value of 0.000 for the RCMM algorithm, and 0.007 for the Spectral algorithm, which indicate that performance differences between our approach and these algorithms are statistically significant.

To give an idea of the reliability of the GP system, in Table 4.3 we provide its run-by-run results over the 15 benchmark problems. As one can see, the algorithm has produced very good results in all runs. It is useful to note that the total running times for the GP approach were approximately 2 hours. We will provide more details on the run times in the following section.

4.4 Application of our GP System to the ERP

Following the success of our initial experiments on the BRP, we decided to apply the proposed GP system to the ERP (for more details, see Sec. 2.5.2) in order to further analyse our method and examine its applicability to other graph layout problems. We employed the same GP system as described in Sec. 4.3. The only change made to the system was the replacement of the fitness function used with the standard objective function of the ERP given in Eq. (2.9). We also replaced the test problems used in the previous work with much larger and more diverse benchmark matrices with sizes ranging from 59×59 to 2680×2680 from Everstine's sparse matrix collection (DWT) [55], included in the Harwell-Boeing database.

4.4.1 Experimental Results

In this section, the experiments carried out to evaluate the performance of our GP system employed as a meta-heuristic for addressing the ERP are presented. All the experiments were performed on an AMD Athlon(tm) Dual-core Processor 2.20 GHz. We report the envelope size obtained for each of the benchmark problems

Table 4.3: Run-by-run results of our GP system for the BRP.

	Benchmark Matrix														
Run	ash85	bcspwr01	bcspwr02	bcsstk01	can_24	can_61	can_62	can_73	can_96	dwt_59	dwt_66	dwt_72	dwt_87	lap_25	nos4
1	11	5	10	24	6	13	9	23	16	8	3	8	13	6	12
2	12	5	10	26	5	14	8	23	16	7	3	8	13	6	11
3	12	5	10	25	6	13	8	23	17	7	3	8	14	6	12
4	12	5	12	24	7	13	8	23	15	7	3	8	13	6	11
5	12	5	10	23	7	13	8	23	16	7	3	8	14	6	12
6	13	5	10	25	5	14	7	25	15	7	3	8	13	6	11
7	11	5	10	25	5	14	8	23	16	7	3	8	13	6	11
8	14	5	10	25	5	13	8	24	15	7	3	9	13	6	11
9	12	5	10	25	5	14	8	23	15	8	3	8	14	6	11
10	11	5	12	23	5	13	8	23	15	7	3	8	13	6	11
Best	11	5	10	23	5	13	7	23	15	7	3	8	13	6	11
Worst	14	5	12	26	7	14	9	25	17	8	3	9	14	6	12
Mean	12.0	5.0	10.4	24.5	5.6	13.4	8.0	23.3	15.6	7.2	3.0	8.1	13.3	6.0	11.3
Standard deviation	0.943	0	0.843	0.972	0.843	0.516	0.471	0.675	0.699	0.422	0	0.316	0.483	0	0.483
Standard error	0.298	0	0.267	0.307	0.267	0.163	0.149	0.213	0.221	0.133	0	0.1	0.153	0	0.153

as well as the time required to solve each problem instance on this computer. The evolution of fitness related to a subset of benchmark instances during runs for the best, worst and average fitness values obtained is also presented and discussed.

Comparison between our GP System and Reference Algorithms

In order to assess the performance of the proposed GP system, we compared it against four best-known and high-performance algorithms, i.e., RCM, GK, Spectral and Sloan as summarised in Sec. 2.5.2. Each method was tested on our set of benchmark matrices, i.e., the DWT set. After performing a number of experiments on the DWT set, we decided to employ the same set of parameters used for the BRP as listed in Table 4.1. Because of the non-deterministic nature of GP, 20 independent runs were executed for each of the selected benchmark instances.

In order to objectively judge how well our GP system performed compared to the other algorithms under test, which are all deterministic, we divided the results obtained from 20 runs on each of the test problems into 4 separate groups of 5. Then, the best solution found in each group was picked, leading to four best-of-group solutions for each problem. These are a sample from the distribution of solutions that one might expect in a realistic situation where users would run the system a few times (5 in this case) and pick the best permutation overall as their final result. We, therefore, subjected them to statistical analysis. Tables 4.4 and 4.5 report the run-by-run results obtained from our GP system over the benchmark problems and the grouping system used.

Table 4.6 shows the envelopes resulting from the permutations produced by the GP system, together with the envelopes produced by the RCM, GK, Spectral and Sloan algorithms for the DWT test problems. Note that all the results associated with the RCM, GK and Sloan algorithms were taken from [156] and [157], and for the Spectral algorithm, we used a MATLAB program to compute the related

Table 4.4: Run-by-run results of our GP system for the ERP and the grouping system used (Part 1).

		Everstine Collection (DWT)												
	Run	59	66	72	87	162	193	209	221	245	307	310	361	419
	1	284	193	240	554	1559	4604	3165	1991	2772	7509	3006	5075	7805
	2	285	193	236	553	1610	4812	3205	1988	2699	7275	3006	5075	7654
Group 1	3	285	193	240	545	1617	4801	3125	2006	2751	7513	3006	5075	7856
	4	281	193	233	524	1621	4880	3183	1984	2764	7516	3006	5075	7921
	5	281	193	238	561	1506	4835	3126	1995	2736	7492	3006	5075	7911
Best in Group 1		*281*	*193*	*233*	*524*	*1506*	*4604*	*3125*	*1984*	*2699*	*7275*	*3006*	*5075*	*7654*
	6	281	193	236	553	1605	4842	3151	1997	2633	7483	3006	5060	7816
	7	282	193	238	556	1625	4842	3144	1978	2734	7361	3006	5075	7826
Group 2	8	279	193	237	552	1618	4738	3173	1976	2690	7468	3006	5075	7873
	9	284	193	237	554	1613	4795	3161	1996	2730	7518	3006	5075	7737
	10	281	193	235	541	1611	4842	3166	1986	2797	7373	3006	5075	7577
Best in Group 2		*279*	*193*	*235*	*541*	*1605*	*4738*	*3144*	*1976*	*2633*	*7361*	*3006*	*5060*	*7577*
	11	284	193	233	520	1602	4830	3168	1982	2677	7483	3006	5075	7733
	12	283	193	237	551	1622	4620	3159	1976	2654	7477	2976	5060	7885
Group 3	13	281	193	236	548	1617	4900	3126	2006	2805	7466	3006	5075	7751
	14	283	193	233	549	1620	4848	3140	1976	2728	7536	3006	5075	7786
	15	278	193	236	562	1628	4887	3164	1990	2795	7461	3006	5060	7805
Best in Group 3		*278*	*193*	*233*	*520*	*1602*	*4620*	*3126*	*1976*	*2654*	*7461*	*2976*	*5060*	*7733*
	16	286	193	237	555	1523	4896	3157	1982	2706	7442	3006	5060	7852
	17	279	193	238	559	1518	4878	3214	1990	2747	7522	3006	5075	7906
Group 4	18	284	193	235	567	1620	4801	3148	1989	2626	7499	3006	5060	7917
	19	283	193	235	521	1612	4812	3136	1995	2657	7457	3006	5075	7913
	20	282	193	237	543	1524	4841	3187	1986	2683	7377	3006	5075	7636
Best in Group 4		*279*	*193*	*235*	*521*	*1518*	*4801*	*3136*	*1982*	*2626*	*7377*	*3006*	*5060*	*7636*
Best (all groups)		278	193	233	520	1506	4604	3125	1976	2626	7275	2976	5060	7577
Worst (all groups)		281	193	235	541	1605	4801	3144	1984	2699	7361	3006	5075	7733
Mean of the best in each group		279.25	193	234.00	526.50	1557.75	4690.75	3132.75	1979.50	2653.00	7368.50	2998.50	5063.75	7650.00
Standard deviation		1.26	0.0	1.15	9.81	53.07	94.73	9.00	4.12	32.89	76.22	15.00	7.50	64.37
Standard error		0.63	0.0	0.58	4.91	26.53	47.36	4.50	2.06	16.45	38.11	7.50	3.75	32.18

Table 4.5: Run-by-run results of our GP system for the ERP and the grouping system used (Part 2).

		Everstine Collection (DWT)										
	Run	503	592	758	869	878	918	992	1005	1007	1242	2680
Group 1	1	13914	10428	7403	15210	19363	18075	33460	31666	20818	41945	91304
	2	13948	10298	7408	15070	19197	18055	33372	31650	20739	41724	91870
	3	13836	10254	7392	15093	19373	18016	33312	31616	20837	41220	91989
	4	13731	10224	7414	15204	19310	18027	33356	31736	20697	41800	91656
	5	13972	10288	7414	14926	19325	18064	33382	31716	20773	41775	91983
Best in Group 1		*13731*	*10224*	*7392*	*14926*	*19197*	*18016*	*33312*	*31616*	*20697*	*41220*	*91304*
Group 2	6	13977	10254	7395	14962	19317	17885	33138	31655	20779	41827	91909
	7	14021	10236	7412	14260	19361	17889	33572	31887	20820	41830	92082
	8	14144	10249	7436	15224	19194	17947	33244	31817	20752	41944	92092
	9	13989	10288	7387	15005	19640	17948	33134	31690	20650	41828	91998
	10	14093	10414	7407	15267	19390	17985	33400	31756	20758	41876	91984
Best in Group 2		*13977*	*10236*	*7387*	*14260*	*19194*	*17885*	*33134*	*31655*	*20650*	*41827*	*91909*
Group 3	11	13581	10204	7452	15264	19455	17998	33508	31494	20859	41858	91997
	12	13773	10193	7394	15042	19249	18006	33194	31844	20768	41898	91941
	13	13841	10421	7404	15517	19404	17919	33394	31798	20837	41934	92221
	14	14023	10239	7369	15345	19415	18097	33340	31750	20782	41745	92199
	15	14118	10259	7393	15017	19362	17952	33142	31500	20953	41757	92236
Best in Group 3		*13581*	*10193*	*7369*	*15017*	*19249*	*17919*	*33142*	*31494*	*20768*	*41745*	*91941*
Group 4	16	13836	10299	7402	15197	19306	18085	33722	31784	20784	42093	91988
	17	13879	10273	7399	15161	19493	18026	33242	31734	20882	41739	91083
	18	14140	10298	7417	15039	19276	17967	33268	31758	20900	41795	92086
	19	14199	10329	7405	15117	19405	17966	33328	31945	20746	41872	92066
	20	14041	10304	7448	15597	19570	17903	33402	31685	20802	41945	92021
Best in Group 4		*13836*	*10273*	*7399*	*15039*	*19276*	*17903*	*33242*	*31685*	*20746*	*41739*	*91083*
Best (all groups)		13581	10193	7369	14260	19194	17885	33134	31494	20650	41220	91083
Worst (all groups)		13977	10273	7399	15039	19276	18016	33312	31685	20768	41827	91941
Mean of the best in each group		13781.25	10231.50	7386.75	14810.50	19229.00	17930.75	33207.50	31612.50	20715.25	41632.75	91559.25
Standard deviation		167.27	33.07	12.82	370.25	40.24	58.51	85.25	83.90	52.66	278.08	432.06
Standard error		83.64	16.54	6.41	185.12	20.12	29.25	42.63	41.95	26.33	139.04	216.03

envelopes. In terms of our GP system, we report the mean of the best candidate solutions selected from each group (GPm) as well as the best result obtained from the 20 runs for each benchmark problem (GPb). We also report the mean of the envelopes obtained by each algorithm over the 24 DWT test problems and the number of wins and draws obtained by each algorithm.

Statistical Analysis

Non-parametric statistical tests were carried out in order to see if the relative performance differences observed in Table 4.6 are statistically significant. We proceeded as follows. First, the results obtained by our GP system in the groups (whose mean was labelled GPm in Table 4.6) and the results produced by each of the reference algorithms (RCM, GK, Spectral and Sloan) were paired resulting in four paired samples: GP/RCM, GP/GK, etc. Then, we ran the Wilcoxon signed-rank test, which is a reliable and widely used nonparametric test for paired data. The test was performed using the SPSS software package (ver. 16.0). The results of the tests are summarised in Tables 4.7–4.9, which report the descriptive statistics and the ranks and test statistics, respectively.

In this study, we used the Monte Carlo method for computing the significance levels of the statistics based on $n = 24$ (problem instances) $\times 4$ (groups of runs) $=$ 96 degrees of freedom, with starting seed 2000000 and the confidence interval of 95% as shown in Table 4.9. It should be noted that the Z values reported in Table 4.9 are based on the negative ranks (see Table 4.8).

A simple inspection of the results reported in Table 4.6 reveals that when taking the best result out of 5 runs (GPm configuration), our GP system outperforms all four algorithms under test with respect to the mean of the envelope values and the number of the best solutions obtained, even more so considering *GPb*. The statistical analysis performed on the best-of-group results also confirms the superiority of

Table 4.6: Comparison of the proposed GP approach against the RCM, GK, Spectral and Sloan algorithms.

	Envelope size					
DWT	RCM	GK	Spectral	Sloan	GPm	GPb
DWT 59	314	314	290	294	279.25	278
DWT 66	217	193†	195	193†	193†	193†
DWT 72	244	244	245	244	234.00	233
DWT 87	696	682	622	525	526.50	520
DWT 162	1641	1579	1751	1554	1557.75	1506
DWT 193	5505	4609	5149	4618	4690.75	4604
DWT 209	3819	4032	3323	3316	3132.75	3125
DWT 221	2225	2154	2030	2052	1979.50	1976
DWT 245	4179	3813	2874	2676	2653.00	2626
DWT 307	8132	8132	7722	7550	7368.50	7275
DWT 310	3006	3006	3102	2982	2998.50	2976
DWT 361	5075	5060†	5339	5062	5063.75	5060†
DWT 419	8649	8073	8953	7255	7650.00	7577
DWT 503	15319	15042	14814	14436	13781.25	13581
DWT 592	11440	10925	10782	10073	10231.50	10193
DWT 758	8580	8175	7563	7598	7386.75	7369
DWT 869	19293	15728	16261	14909	14810.50	14260
DWT 878	22391	19696	19896	20537	19229.00	19194
DWT 918	23105	20498	18603	17236	17930.75	17885
DWT 992	38128	34068	35748	33928	33207.50	33134
DWT 1005	43068	40141	32613	36396	31612.50	31494
DWT 1007	24703	22465	21474	22669	20715.25	20650
DWT 1242	50052	52952	43661	36822	41632.75	41220
DWT 2680	105663	99271	92513	89686	91559.25	91083
Mean	16893.50	15868.83	14813.46	14275.46	14184.34	14083.83
Wins/Draws (excluding GPb)	0/0	2/1	0/0	8/1	13/1	n/a
Wins/Draws (excluding GPm)	0/0	0/2	0/0	5/1	n/a	17/2

† = ties
GPm = mean of the best results obtained by GP in each group
GPb = best result obtained by GP in 20 independent runs

Table 4.7: Descriptive statistics.

						Percentiles		
Algorithm	n	Mean	Std. Deviation	Minimum	Maximum	25th	50th (Median)	75th
GPm	96	14184.3438	19758.69114	193.00	91941.00	2144.5000	7382.0000	18899.5000
RCM	96	16893.5000	23244.54168	217.00	105663.00	2420.2500	8356.0000	22926.5000
GK	96	15868.8333	22086.21071	193.00	99271.00	2367.0000	8102.5000	20297.5000
Spectral	96	14813.4583	20138.26281	195.00	92513.00	2241.0000	7642.5000	19572.7500
Sloan	96	14275.4583	19463.98680	193.00	89686.00	2208.0000	7402.5000	19711.7500

Table 4.8: Ranks.

		n	Mean Rank	Sum of Ranks
$RCM - GPm$	Negative Ranks	0^a	.00	.00
	Positive Ranks	92^b	46.50	4278.00
	Ties	4^c		
	Total	96		
$GK - GPm$	Negative Ranks	6^d	12.00	72.00
	Positive Ranks	80^e	45.86	3669.00
	Ties	10^f		
	Total	96		
$Spectral - GPm$	Negative Ranks	0^g	.00	.00
	Positive Ranks	96^h	48.50	4656.00
	Ties	0^i		
	Total	96		
$Sloan - GPm$	Negative Ranks	34^j	50.51	1717.50
	Positive Ranks	58^k	44.15	2560.50
	Ties	4^l		
	Total	96		

a. $RCM < GPm$	b. $RCM > GPm$	c. $RCM = GPm$
d. $GK < GPm$	e. $GK > GPm$	f. $GK = GPm$
g. $Spectral < GPm$	h. $Spectral > GPm$	i. $Spectral = GPm$
j. $Sloan < GPm$	k. $Sloan > GPm$	l. $Sloan = GPm$

Table 4.9: Test statistics.

			$RCM-GPm$	$GK-GPm$	$Spectral-GPm$	$Sloan-GPm$
Monte Carlo Sig. (1-tailed)	**P-values**		**.000**	**.000**	**.000**	**.031**
	95% Confidence Interval	Lower Bound	.000	.000	.000	.000
		Upper Bound	.031	.031	.031	.066
Z			-8.329	-7.744	-8.507	-1.641

our GP system over the other methods tested by providing the related p-values. As indicated in Table 4.9, this analysis yielded a one-tailed p-value of $p = 0.000$ for the *RCM* algorithm, $p = 0.000$ for the *GK* algorithm, $p = 0.000$ for the *Spectral* algorithm and $p = 0.031$ for the *Sloan* algorithm, which clearly demonstrates that the performance differences between our GP approach and these algorithms are statistically significant.

According to [156, 157], the RCM algorithm is typically 140% faster than the GK algorithm and 50% faster than the Sloan algorithm. The Spectral algorithm [15] is the slowest of our four reference algorithms. Overall, in our tests, the execution times of these algorithms were very short, ranging from a fraction of a second to a few seconds. Instead, on average, GP took of the order of hundreds of seconds. This is illustrated by Table 4.10, which presents computing times for the GP approach and Sloan algorithm for each problem instance. As one can see there is a 3-order-of-magnitude difference between the two algorithms.

This, of course, is not surprising with respect to the fact that the reference algorithms are heuristic and based on graph-theoretic concepts, while our algorithm is based on GP, which is a metaheuristic algorithm. Indeed, metaheuristics have been applied successfully to the BRP, which has similarities with the ERP (see Sec. 2.5.1 and Sec. 2.5.3), typically producing better results in comparison with classic graph theoretic-based methods (e.g., CM, RCM, GPS etc). However, the the most successful graph-theoretic BRP heuristic solvers can be thousands to

Table 4.10: Computing times for the GP approach and Sloan algorithm for each problem instance.

	CPU time (seconds)	
DWT	Sloan	GP approach
DWT 59	0.02	12.01
DWT 66	0.02	12.65
DWT 72	0.01	13.23
DWT 87	0.03	22.18
DWT 162	0.07	30.22
DWT 193	0.22	54.22
DWT 209	0.12	59.32
DWT 221	0.11	69.21
DWT 245	0.14	68.97
DWT 307	0.20	107.69
DWT 310	0.17	44.58
DWT 361	0.16	74.92
DWT 419	0.26	155.82
DWT 503	0.40	207.36
DWT 592	0.38	278.20
DWT 758	0.44	332.28
DWT 869	0.48	417.18
DWT 878	0.58	484.76
DWT 918	0.61	610.15
DWT 992	1.09	540.37
DWT 1005	0.81	604.85
DWT 1007	0.71	745.94
DWT 1242	0.98	881.73
DWT 2680	2.41	3900.50
Mean	0.4341	405.3475
SD	0.5120	772.8665
SEM	0.1045	157.7607

SD = Standard Deviation
SEM = Standard Error of the Mean

hundreds of thousands of times faster than the corresponding metaheuristics [148].

While this high computational effort limits the applicability of our algorithm, there are still many situations, where the method is very valuable. Firstly, we should remember that if P $\neq$ NP, there exists no efficient algorithm for determining the minimum envelope and bandwidth of a matrix (indeed, for all problems tackled in this work, these quantities are unknown). So, any algorithm capable of further reducing these quantities is valuable regardless of execution-time issues. For example, the results obtained by our algorithm can now be used as a reference for comparing the performance of other algorithms. There are also situations, where one really needs to care about the accuracy and quality of the solutions rather than computational times. Finally, we should note that, unlike traditional methods, our system can easily be parallelised to exploit the power of modern multi-core machines and GPUs, so that runs that now take a few minutes could be completed in a few seconds or less.

Fitness Evolution

In order to analyse the evolution of fitness during our runs, we divided the 24 test problems used in our experiments into 3 groups, including small-sized (59 up to 221), medium-sized (245 up to 758) and large-sized (869 up to 2680) problems. Then, we selected the largest instance in each group as a representative problem to report fitness evolution. For each run, the best, worst and average fitness values in each generation were recorded. The mean of these fitness values (best, worst and average) for each generation were then computed and used in order to visualise the fitness evolution as illustrated in Figs. 4.3–4.5. Note that due to the big differences in the best, average and worst fitness values, in these figures, we used three scales in each plot. Also, note that the error bars represent the standard error of the mean.

Starting from the plots of the best fitness, it is clear that in all three problems,

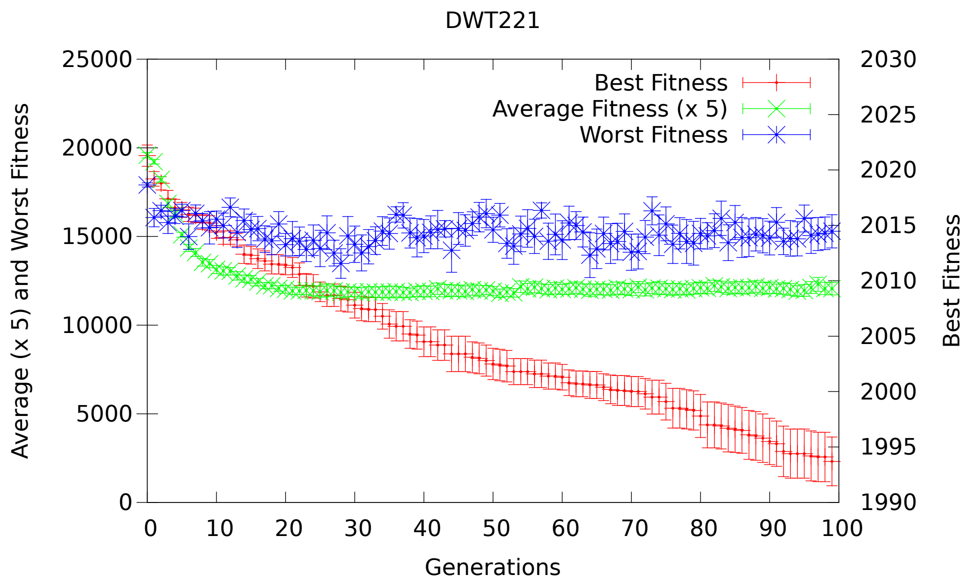


Fig. 4.3: The best, worst and average fitness values (based on 10 independent runs) associated with the DWT 221 as a function of the number of generations.

the best fitness continues to improve until the end of runs, showing no signs of stagnation, although the rate of improvement slows down slightly in the larger instances. As shown by the very small standard errors, the process is very reliable, effectively leading to good quality solutions in most runs. Turning to the worst fitnesses, we see that they start by improving slightly to later either plateauing or then getting slightly worse again and then plateauing. A similar profile is followed by the average-fitness plots. However, the upward trend shown by the averages in the late stages of a run is not present in the medians (not reported), which are much more robust to outliers. All this is an indication that as the solutions (permutations) get more and more optimised, they become more and more fragile to changes, leading to a substantial fraction of really bad offspring. Since such offspring have fitnesses one order of magnitude worse than the average, the average is heavily

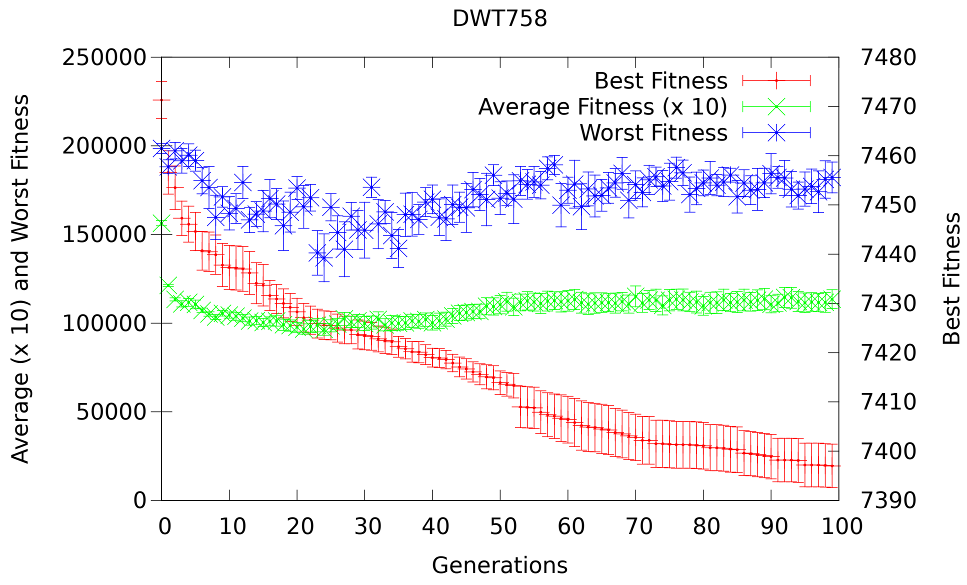


Fig. 4.4: The best, worst and average fitness values (based on 10 independent runs) associated with the DWT 758 as a function of the number of generations.

influenced by such individuals.

4.5 Chapter Summary

In this chapter, we have detailed our initial work towards the development of the *Genetic Hyper-Heuristics*. We have also introduced a specialised GP system, which forms the core of the genetic hyper-heuristics as we will see in the next chapter. The proposed GP system is capable of being used as a meta-heuristic and also as a hyper-heuristic in order to tackle graph layout problems. Our preliminary experiments for testing the effectiveness of the GP system were performed on the BRP. The system was employed as a meta-heuristic and compared with two widely-used and high-performance methods, i.e. the RCMM (RCM algorithm con-

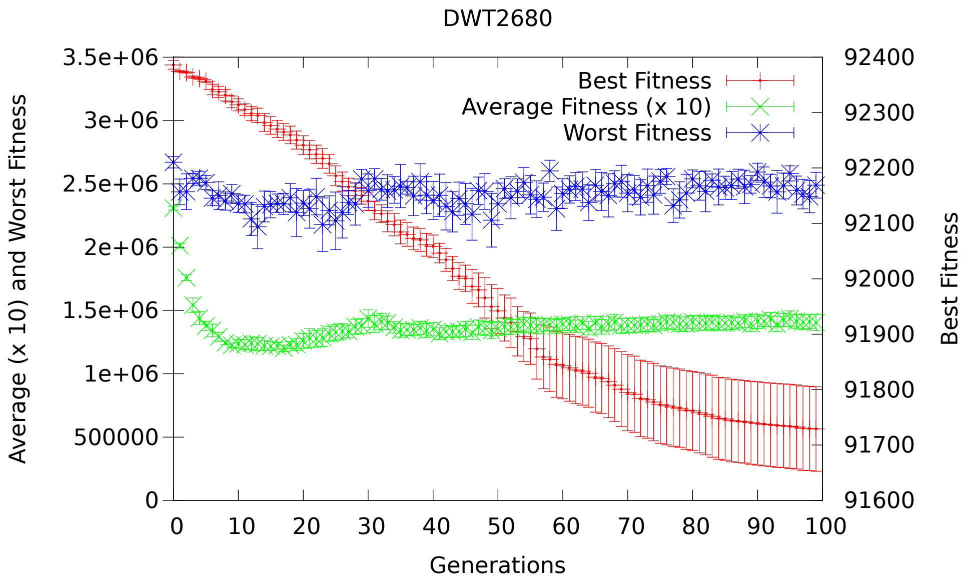


Fig. 4.5: The best, worst and average fitness values (based on 10 independent runs) associated with the DWT 2680 as a function of the number of generations.

tained in the MATLAB library) and Spectral algorithms. The results obtained were very satisfactory and encouraged us to further analyse our method and examine its applicability to other graph layout problems. We therefore applied the GP system to the ERP and performed a substantial number of experiments with much larger and more diverse benchmark matrices.

Our method was evaluated against four of the best-known and broadly used envelope reduction algorithms, namely the RCM, GK, Spectral and Sloan. These graph-theoretic heuristic algorithms were significantly faster than our approach, which was expected given that the approach requires runs of a GP system. However, from the point of view of envelope reduction, the results obtained indicate that the proposed method compares very favourably with the four reference algorithms, *outperforming them in most cases*.

The experiments conducted also reveal that the GP system presented can generally be applied to other graph layout problems with some small modifications or additional decoding steps, as expected.

Chapter 5

Evolving Heuristic Algorithms for Graph Layout Problems

5.1 Introduction

In Chapter 4, a Genetic Programming (GP) system, specifically designed for addressing graph layout problems was introduced. The idea behind the development of this GP system was that it would be the key component of the architecture of *Genetic Hyper-Heuristics* as mentioned earlier. The application of the proposed GP system as a *meta-heuristic* to the bandwidth and envelope reduction problems (BRP and ERP) was also discussed in detail. In this chapter, a general framework for genetic hyper-heuristics, aimed at automatically evolving reusable heuristic algorithms for graph layout problems is presented. The framework acts as an offline learning mechanism which generates heuristics by learning from a set of training instances. This framework falls into the second category of hyper-heuristic methods, in which the aim is to generate new heuristics from a set of potential heuristic components (see Sec. 3.3.2 for more details).

Based on the proposed framework, two genetic hyper-heuristic approaches are designed for evolving graph-theoretic bandwidth and envelope reduction algorithms respectively. These two approaches are, in fact, specialisations of the genetic hyper-heuristic framework. We test the best of such evolved algorithms on a large set of standard benchmarks from the Harwell-Boeing sparse matrix collection against the best-known human-designed methods from the literature. Our algorithms outperform these methods by a significant margin, clearly indicating the promise of the approaches presented. To the best of our knowledge, there has been no research investigating the application of hyper-heuristics to the graph layout problems thus far.

In addition, in this chapter, we propose new ideas for using a level structure system without its conventional constraints, and also employing novel features in the process of prioritising nodes for the construction of permutations.

This chapter is organised as follows: In Sec. 5.2, a general framework for genetic hyper-heuristics is presented. Sec. 5.3 and Sec. 5.4 detail two genetic hyper-heuristic approaches for the BRP and ERP and provide a complete description of our experiments as well as the interpretation and discussion of the results. Finally, Sec. 5.5 summarises this chapter.

5.2 Genetic Hyper-Heuristics

As we described in Sec. 4.1 of the previous chapter, the hyper-heuristic methods proposed in this thesis are generally termed Genetic Hyper-Heuristics. We also use the abbreviation GHH to denote a genetic hyper-heuristic. The key component of GHHs is the specialised GP system presented and analysed in Chapter 4. The proposed GP system uses a set of low level building blocks (primitive set) to generate a new heuristic that performs well in the environment (training set) provided for the system. In fact, a GHH acts as an *offline learning* [19] mechanism which generates new heuristics by gathering knowledge from a set of training instances and intelligently combining the building blocks given to the system in the evolution process. This leads to the creation of reusable heuristic algorithms, which are intended for use on unseen problems. More specifically, a GHH approach is first applied to an independent training set of graphs/matrices (learning takes place in this phase), and then the best evolved algorithm is employed to directly address the targeted problem instance(s) (from a totally different set of data).

The pseudocode in Algorithm 3 shows a general framework for the genetic hyper-heuristics. The proposed framework is, in fact, the result of the transformation of the GP system presented in Chapter 4 into an offline learning method. This is accomplished by incorporating a loop, that iterates for each graph (or matrix) instance in the training set, into the GP system (Algorithm 3, step 6). Consequently, the fitness of a program tree is computed as the total of the layout costs of the solutions that it creates when run on each problem instance in the training set (Algorithm 3, step 12). Note that if there is a significant difference in size between each problem instance in the training set, it would be reasonable to normalise the layout cost of each instance with respect to the size of the matrix. Otherwise, there might always be a bias towards larger instances, since the layout cost associated with a large matrix could be much bigger than a small matrix. One possibility for dealing with this issue is to divide the layout cost of each matrix in the training set by its size. Note also that in our framework, program trees in the population represent heuristic algorithms (which are compositions of the functions and terminals suitable for graph layout problems), and the outputs produced by a program tree can be interpreted as a permutation (see Sec. 4.2.1 and Sec. 4.2.3 for details).

As mentioned earlier, GHHs are designed to generate reusable heuristics, since having such heuristics can increase the speed of solving new unseen instances of

Algorithm 3 The general framework for genetic hyper-heuristics.

```
1: Randomly generate an initial population of programs.
2: repeat
3:   for each program $p \in$ population do
4:     $fitness[p] = 0$
5:   end for
6:   for each graph (or matrix) instance $i \in$ training set do
7:     for each program $p \in$ population do
8:       Execute $p$.
9:       Create permutation $\sigma$ represented by $p$.
10:      Apply $\sigma$ to the adjacency list of $i$, and generate a new adjacency list.
11:      Compute the layout cost for the adjacency list thus obtained.
12:      $fitness[p] = fitness[p] + LayoutCost(i, p)$.
13:    end for
14:  end for
15:  Perform selection to choose individual program(s) from the population based on
     fitness to participate in genetic operations.
16:  Create a new generation of individual program(s) by applying genetic operations
     with specified probabilities.
17: until the maximum number of generations is reached.
18: return the best program tree appearing in the last generation.
```

graph layout problems. Although, in this study, it is not our intention to create disposable heuristics, which are generally produced just for a specific instance of a problem, this is easily possible by employing a GHH as an *online learning* [19] method. In that case, the GHH learns while solving a given instance of a problem, and there is no need to train the system before using it.

In the next sections, two genetic hyper-heuristic approaches based on the proposed framework are presented for evolving graph-theoretic bandwidth and envelope reduction algorithms respectively.

5.3 A Genetic Hyper-Heuristic Approach for the BRP

As mentioned in Sec. 2.5.4, the concept of level structures is of great importance to most graph-theoretic bandwidth and envelope reduction algorithms (e.g., the CM, RCM and GPS algorithms). The key element of such algorithms that significantly influences performance is a particular strategy by which they prioritise nodes in a level structure for insertion into a permutation (i.e. a numbering scheme). This strategy is, in fact, the "brain" of algorithms of this type.

In order to generate graph-theoretic heuristic algorithms for the BRP (see Sec. 2.5.1), we designed a Genetic Hyper-Heuristic (GHH1) by employing the general framework proposed in the previous section. In this work, our aim is to evolve more intelligent brains rather than common components of these algorithms (i.e., a level structure system plus a method for finding a suitable starting vertex). Therefore, we incorporate such components into GHH1 to be used as the basic structure of a BRP solver. By adopting this approach, GHH1 can concentrate on evolving the most important part of the algorithm, i.e., its brain. A description of the operations of the system is given in Algorithm 4.

Naturally, the key component of GHH1 is the specialised GP system presented in Chapter 4. Therefore, the methods used in GHH1 for initial population generation, fitness evaluation, selection, genetic operations and termination are the same as in that GP system.

Note that in GHH1, the fitness of a program tree is the total of the bandwidths of the solutions that it creates when run on each problem instance in the training set (Algorithm 4, step 21), and the fitness, rather obviously, needs to be minimised by the system. Note also that in order to control excessive code growth or bloat, the

Algorithm 4 GHH1 for the BRP.

```
 1: Randomly generate an initial population of programs from the available primitives.
 2: repeat
 3:     for each program $p \in$ population do
 4:         $fitness[p] = 0$
 5:     end for
 6:     for each instance $i \in$ training set do
 7:         Select a starting vertex $s$.
 8:         Construct level structure $L$ rooted at $s$.
 9:         for each program $p \in$ population do
10:             Insert $s$ into array $perm[1...n]$.
11:             for each level $l \in L$ do
12:                 for each vertex $v \in l$ do
13:                     Execute $p$.
14:                 end for
15:                 Create permutation $\sigma$ represented by $p$.
16:                 Sort vertices in $l$ in order given by $\sigma$.
17:                 Store the ordered vertices in $perm[\,]$ sequentially.
18:             end for
19:             Apply $perm[\,]$ to the adjacency list of the graph (or matrix) $i$, and generate a
                new adjacency list.
20:             Compute the $bandwidth$ for the adjacency list thus obtained.
21:             $fitness[p] = fitness[p] + bandwidth(i, p)$.
22:         end for
23:     end for
24:     Perform selection to choose individual program(s) from the population based on
        fitness to participate in genetic operations.
25:     Create a new generation of individual programs by applying genetic operations with
        specified probabilities.
26: until the maximum number of generations is reached.
27: return the best program tree appearing in the last generation.
```

Table 5.1: Primitive set used in GHH1.

Primitive set	Arity	Description
+	2	Adds two inputs
-	2	Subtracts second input from first input
$*$	2	Multiplies two inputs
N	0	Returns the number of vertices of a given graph (or the dimension of a given matrix)
SDV	0	Returns the sum of degrees of vertices connected to a vertex
Constants	0	Uniformly-distributed random constants with floating-point values in the interval $[-1.0, +1.0]$

Tarpeian method (which artificially weeds out a proportion of above-average-size trees) [140, 141] was utilised in the system.

5.3.1 Specialised Primitives

In nodal numbering schemes based on the level structures such as the CM and RCM, the vertices in each level are numbered in order of increasing degree. We should note, however, that our primitive set (shown in Table 5.1) does not include a function which returns the degree of the nodes in a graph. We did this for the following reason. In preliminary runs with such a primitive, we obtained relatively weak results. This is somehow surprising, since the degree of a node is what the CM and RCM algorithms use for prioritising nodes within a level. We believe that evolution found it particularly difficult to use such a primitive due to the problem of ties.

In a level, there are often nodes with the same degree which result in ties. As mentioned in Sec. 2.5.4, the normal strategy for dealing with this issue is to break ties randomly. Although this can resolve the problem somehow, it leads to a strong non-determinism in the fitness evaluation. This may hamper the evolutionary search in that a lucky fitness evaluation may lead to an inferior individual to be selected over and over again. The problem is particularly severe if the number of ties is large.

By using the sum of the degrees of the vertices connected to a vertex, `SDV`, in place of a vertex degree, the likelihood of ties in a level is considerably reduced. Also, some additional information related to the vertices located in the following level is captured by `SDV`, which is effectively equivalent to the product of a vertex degree and the mean degree of the vertex's children.

In addition, the primitive `N`, which returns the number of vertices of a given graph (or the dimension of a given matrix) is designed to capture information about problem size. Note that `N` is constant for each problem.

5.3.2 Numbering Vertices Located in Each Level

As shown in Algorithm 4, first, a level structure rooted at a suitable starting vertex is constructed (Step 8). The root vertex is then assigned the number 1, and inserted into the first position of array $perm[1...n]$, which is the result array (Step 10).

For each vertex v located in level l, the GP interpreter is called k times, where k is the number of vertices in l. Each call of the interpreter executes the selected program with respect to the different values returned by `SDV` (Step 13). The outputs obtained from each execution of the given program are stored in a one-dimensional array, and the permutation associated with the original program tree is produced as described in Sec. 4.2.1 (Step 15). The vertices located in l are then ordered based on the permutation generated (Step 16).

The ordered vertices are sequentially assigned the number 2 for the first element, number 3 for the second element, etc., and are stored in $perm[]$ (Step 17). This process is repeated for each successive level in the rooted level structure until the vertices of all levels have been numbered. Note that after the termination of this process, the indices of $perm[]$ correspond to the numbers assigned to the vertices.

In order to compute the bandwidth, *perm*[] should be applied to the adjacency list of the initial graph (or the initial matrix) (Steps 19 and 20).

5.3.3 Training Set

We used a training set of 25 benchmark instances from the Harwell-Boeing sparse matrix collection. This is a collection of standard test matrices arising from problems in finite element grids, linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The benchmark matrices were selected from 5 different sets in this collection, namely BCSSTRUC1 (dynamic analyses in structural engineering), BCSSTRUC3 (dynamic analyses in structural engineering), CANNES (finite-element structures problems in aircraft design), LANPRO (linear equations in structural engineering), and LSHAPE (finite-element model problems) with sizes ranging from 24×24 to 960×960. Note that we employed this training set only to evolve the heuristics. The performance of the evolved heuristics was then evaluated using two completely separate test sets, as discussed in the next section.

5.3.4 Test Problems

In order to compare the performance of the algorithms evolved by our proposed hyper-heuristic methods with high-performance algorithms from the literature, we employed Everstine's sparse matrix collection (DWT set) [55] and BCSPWR set as test sets. In Sec. 4.4, the DWT set was referred to briefly. Below, we provide more information on this set.

The DWT set consists of 30 sparse matrices from NASTRAN (a finite element analysis program) users working in U.S. Navy, Army, Air Force and NASA laboratories. The collection has been widely used in benchmarks, and it is, in fact, a subset of the Harwell-Boeing sparse matrix collection. Since the DWT benchmark matrices have been collected from a diverse range of finite element grids in a variety of engineering disciplines, they seem large and diversified enough to be used reliably for assessing the performance of bandwidth and envelope reduction algorithms. DWT set is comparable with the CANNES and LSHAPE sets used in our training set in terms of its discipline and the class of problems. Note that 6 matrices out of 30 in this collection, namely DWT 198, DWT 234, DWT 346, DWT 492, DWT 512 and DWT 607 were not included in our experiments because we found that the Laplacian matrix associated with each of these matrices had more than one zero eigenvalue.[1]

The BCSPWR set (power network patterns) consists of 10 sparse matrices included in the Harwell-Boeing database, and it has been broadly used as examples of networks rather than grid problems. This set is entirely in a different class compared to the training set used and the DWT set. Our aim of selecting the BCSPWR set was to assess the ability of the evolved algorithms to generalising to unseen situations. A complete description of the 34 test problems used in our experiments is presented in Table 5.2. This collection includes benchmark matrices of sizes ranging from 39×39 to 5300×5300.

5.3.5 Experimental Results

We performed ten independent runs of GHH1 with the training set specified above, recording the corresponding best-of-run individual in each.[2] The parameters of the

[1]This, in fact, means that their corresponding graphs have more than one component and in such a scenario, studying each component of the system separately is more likely to result in an efficient analysis. It is probable that in the original process of generating these 6 matrices from their actual numerical values, a rounding error has occurred leading to some entries being ignored.

[2]Note that due to the high computational load involved in the use of hyper-heuristics, one can normally only perform a very small number of runs. However, this is normally considered acceptable,

Table 5.2: Test problems from Everstine's sparse matrix collection (DWT) and BCSPWR set together with their statistics.

Set	Instance	Dimension	Entries	Original Envelope	Comment
DWT	DWT 59	59 × 59	267	464	PSI, 2D frame
	DWT 66	66 × 66	320	640	PSI, Truss
	DWT 72	72 × 72	222	244	PSI, Grillage
	DWT 87	87 × 87	541	2336	PSI, Tower
	DWT 162	162 × 162	1182	2806	PSI, Plate with hole
	DWT 193	193 × 193	3493	7953	PSI, Knee prosthesis
	DWT 209	209 × 209	1743	9712	PSI, Console
	DWT 221	221 × 221	1629	10131	PSI, Hull-tank region
	DWT 245	245 × 245	1461	4179	PSI, Carriage
	DWT 307	307 × 307	523	8132	PSI, Power supply housing
	DWT 310	310 × 310	2448	3006	PSI, Hull with refinement
	DWT 361	361 × 361	2953	5445	PSI, Cylinder with cap
	DWT 419	419 × 419	3563	40145	PSI, Barge
	DWT 503	503 × 503	6027	36417	PSI, Baseplate
	DWT 592	592 × 592	5104	29397	PSI, CVA bent
	DWT 758	758 × 758	5994	23871	PSI
	DWT 869	869 × 869	7281	20397	PSI
	DWT 878	878 × 878	7448	26933	PSI, Plate with insert
	DWT 918	918 × 918	7384	109273	PSI, Beam with cutouts
	DWT 992	992 × 992	16744	263298	PSI, Mirror
	DWT 1005	1005 × 1005	8621	122075	PSI, Baseplate
	DWT 1007	1007 × 1007	8575	26793	PSI
	DWT 1242	1242 × 1242	10426	111430	PSI, Sea chest
	DWT 2680	2680 × 2680	25026	590543	PSI, Destroyer
BCSPWR	BCSPWR01	39 × 39	85	331	PSI, Standard IEEE test power system – New England
	BCSPWR02	49 × 49	108	426	PSI, Small test power system – WSCC
	BCSPWR03	118 × 118	297	1406	PSI, IEEE standard 118 bus test case power network
	BCSPWR04	274 × 274	943	21289	PSI, Equivalenced representation of US power network
	BCSPWR05	443 × 443	1033	36691	PSI, Equivalenced representation of western US power network
	BCSPWR06	1454 × 1454	3377	77060	PSI, Western US power network – 1454 bus
	BCSPWR07	1612 × 1612	3718	90669	PSI, Western US power network – 1612 bus
	BCSPWR08	1624 × 1624	3837	96410	PSI, Western US power network – 1624 bus
	BCSPWR09	1723 × 1723	4117	474238	PSI, Western US power network – 1723 bus
	BCSPWR10	5300 × 5300	13571	6127500	PSI, Western US power network – 5300 bus

PSI = Pattern Symmetric Indefinite

Table 5.3: Parameters used in our experiments with GHH1.

Parameter	Value
Maximum Number of Generations	100
Maximum Depth of Initial Programs	3
Population Size	1000
Tournament Size	3
Elitism Rate	0.1%
Reproduction Rate	0.9%
Crossover Rate	80%
Mutation Rate	19%
Mutation Per Node	0.05%

runs are presented in Table 5.3.[3]

We then selected as our overall best evolved heuristic the best program tree from these ten best-of-run results. The simplified version of this evolved heuristic for ordering nodes in a level is as follows:

$$0.179492928171 \times \mathtt{SDV}^3 + 0.292849834929 \times \mathtt{SDV}^2 - 0.208926175433 \times \mathtt{N}$$
$$- 0.736485142138 \times \mathtt{N} \times \mathtt{SDV} - 1.77524579882 \times \mathtt{SDV} - 1.75681383404$$

which is also shown graphically in Fig. 5.1. What is interesting about this result is that the function evolved is *not* monotonically increasing in `SDV` (for any

since whenever focusing on human-competitive results one is more interested in the algorithms resulting from the application of a hyper-heuristic than the analysis of the hyper-heuristic itself.

[3]It should be noted that the parameters reported were chosen from a range of possible initialisation parameters after performing a number of experiments on the DWT and BCSPWR sets, taking into consideration both the quality of solutions and run times. However, we cannot guarantee the optimality of these parameters.

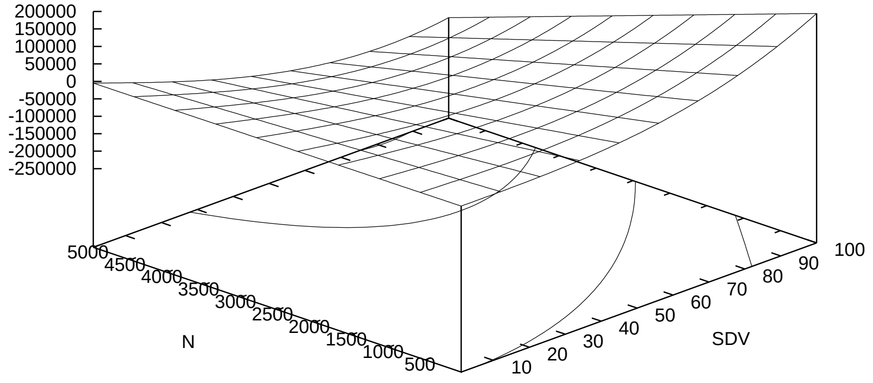


Fig. 5.1: Plot of the best heuristic evolved by GHH1 for the BRP.

given N). Indeed, the minimum of the function as N varies is given by $\mathtt{SDV} = \sqrt{1.5863264966 \times \mathtt{N} + 4.16677290311}/1.07695756903 - 0.543846560629$.

In other words, for small values of N, the system tries to select first nodes with minimum SDV, which is consistent with the standard strategy of sorting nodes by degree, as it is done, for example, in the CM and RCM algorithms. However, as N increases, the heuristic function evolved by GHH1 becomes concave and the system starts favouring nodes with intermediate values of SDV over either very big or very small SDVs. Presumably, while preferring nodes with small degrees is generally a good strategy, it can sometimes force the algorithm along very narrow paths (where one low degree node is followed by another low degree node) which do not give enough choice to the algorithm.

We then incorporated this heuristic into a level structure system as explained in Sec. 2.5.4, and carried out a substantial number of experiments. Note that in the experiments conducted in this work, we used 24 instances from the DWT set as well as the six largest instances from the BCSPWR set as test problems (see Sec. 5.3.4 for details).

In order to assess the performance of the best heuristic evolved by GHH1, we compared it against two well-known and high-performance algorithms, i.e., the RCM contained in the MATLAB library (RCMM), and spectral (see Sec. 4.3.1 for details). All the experiments were performed on an AMD Athlon(tm) Dual-core Processor 2.20 GHz. In addition to reporting the bandwidth obtained by each method under test for each benchmark problem, we also measured the time required to solve each problem instance on this computer.

Table 5.4 shows a performance comparison of the algorithms under test. The results of the tests reveal that the best heuristic generated by GHH1 is far superior to both the RCMM algorithm and the spectral algorithm with respect to the mean of the bandwidth values, the run times and the number of the best results obtained under both criteria (shown in the "Wins/Draws" row). A Wilcoxon signed-rank test, which is a reliable and widely used nonparametric test for paired data, also reveals that the performance differences between GHH1's best evolved heuristic and these algorithms are highly statistically significant ($p = 0.000$).

5.4 A Genetic Hyper-Heuristic Approach for the ERP

Following the strategy adopted in the previous section, we designed a Genetic Hyper-Heuristic (GHH2) in order to generate graph-theoretic heuristic algorithms for the ERP. A description of GHH2 is given in Algorithm 5.

Note that unlike most of the previous nodal numbering schemes for sparse matrices (including the best heuristic evolved by GHH1) which prioritise nodes

Table 5.4: Comparison of GHH1's best evolved heuristic against the RCMM and Spectral algorithms.

		Bandwidth			Run time		
Instance	Dimension	RCMM	Spectral	GHH1	RCMM	Spectral	GHH1
DWT 59	59 × 59	**8**	10	**8**	0.00923	0.01445	**0.00169**
DWT 66	66 × 66	**3**	**3**	**3**	0.00608	0.03477	**0.00189**
DWT 72	72 × 72	**7**	12	**7**	0.00762	0.02110	**0.00210**
DWT 87	87 × 87	**18**	19	**18**	0.00951	0.03599	**0.00135**
DWT 162	162 × 162	**16**	26	**16**	0.00738	0.04329	**0.00273**
DWT 193	193 × 193	54	**45**	49	0.00965	0.06501	**0.00381**
DWT 209	209 × 209	**33**	52	34	0.01079	0.07493	**0.00318**
DWT 221	221 × 221	**15**	23	19	0.01220	0.07522	**0.00953**
DWT 245	245 × 245	55	90	**43**	0.00845	0.09125	**0.00844**
DWT 307	307 × 307	44	64	**39**	0.01178	0.21649	**0.00990**
DWT 310	310 × 310	15	18	**13**	0.01040	0.18425	**0.00932**
DWT 361	361 × 361	**15**	24	**15**	0.01054	0.32843	**0.00761**
DWT 419	419 × 419	34	65	**33**	0.01516	0.41680	**0.00754**
DWT 503	503 × 503	64	91	**51**	0.01707	0.86690	**0.00991**
DWT 592	592 × 592	42	101	**41**	0.01976	1.21548	**0.01009**
DWT 758	758 × 758	29	40	**26**	0.03121	2.49725	**0.01988**
DWT 869	869 × 869	43	149	**41**	0.03846	3.53698	**0.01102**
DWT 878	878 × 878	46	214	**44**	0.03831	3.73173	**0.01127**
DWT 918	918 × 918	57	82	**44**	0.04098	4.13962	**0.02502**
DWT 992	992 × 992	65	**60**	63	0.05488	4.44688	**0.01661**
DWT 1005	1005 × 1005	104	148	**101**	0.05092	5.86841	**0.01711**
DWT 1007	1007 × 1007	**38**	81	**38**	0.04824	5.56310	**0.02455**
DWT 1242	1242 × 1242	**92**	142	94	0.07462	9.96307	**0.02809**
DWT 2680	2680 × 2680	**69**	161	**69**	0.36711	91.7894	**0.03876**
Mean		40.25	71.67	37.87	0.03793	5.63420	0.01172
BCSPWR05	443 × 443	68	132	**56**	0.02329	0.460892	**0.01886**
BCSPWR06	1454 × 1454	126	204	**103**	0.10788	15.5191	**0.02912**
BCSPWR07	1612 × 1612	140	192	**119**	0.12310	19.1069	**0.03150**
BCSPWR08	1624 × 1624	135	341	**111**	0.12659	18.4791	**0.03898**
BCSPWR09	1723 × 1723	133	241	**126**	0.14560	22.4614	**0.04590**
BCSPWR10	5300 × 5300	285	554	**278**	3.19833	674.3833	**0.05982**
Mean		147.83	277.33	132.17	0.62079	125.06844	0.03736
Wins/Draws		3/8	2/1	17/8	0/0	0/0	30/0

Numbers in bold face are the best results.

Algorithm 5 GHH2 for the ERP.

```
1: Randomly generate an initial population of programs from the available primitives.
2: repeat
3:   Initialise the fitness of each program $p \in$ population to 0.
4:   for each instance $G_i \in$ training set of ERPs do
5:     Select a starting vertex $s$ and construct a level structure rooted at $s$.
6:     for each program $p \in$ population do
7:       $l \leftarrow$ empty list
8:       for each vertex $v \in V(G_i)$ do
9:         Insert $s$ into array $perm[1...n]$ and update $l$.
10:        Scan $l$ and if $l.count = 0$, then break.
11:        for each vertex $v' \in l$ do
12:          Execute $p$.
13:        end for
14:        Create permutation $\sigma$ represented by $p$.
15:        Sort vertices in $l$ in order given by $\sigma$.
16:        $s \leftarrow$ first element of the ordered list $l$; $l.remove(s)$.
17:      end for
18:      Apply $perm$ to the adjacency list of the graph $G_i$.
19:      Compute the $envelope$.
20:      $fitness[p] = fitness[p] + envelope(G_i, p)$.
21:    end for
22:  end for
23:  Apply selection.
24:  Produce a new generation of individual programs.
25: until the termination condition is met.
26: return the best program tree.
```

Table 5.5: Primitive set used in GHH2.

Primitive set	Arity	Description
+	2	Adds two inputs
-	2	Subtracts second input from first input
*	2	Multiplies two inputs
`ED`	0	Returns the number of unvisited vertices connected to each vertex
`DFSV`	0	Returns the distance from starting vertex for each vertex
Constants	0	Uniformly-distributed random constants in the interval $[-1.0, +1.0]$

at each level in a level structure independently, heuristics evolved by GHH2 are capable of exploring and sorting vertices located beyond a specific level. More details on Algorithm 5 are provided below.

5.4.1 Specialised Primitives

To make it feasible for GHH2 to exploit the new possibilities offered by its ability to explore and prioritise vertices located at different depths in a level structure, we provided it with two specially designed primitives, `ED` and `DFSV` (see Table 5.5).

The primitive `ED`, which stands for *Effective Degree*, is motivated by the common method of sorting vertices in a level structure based on their degree in classical bandwidth and envelope reduction algorithms. The degree of a vertex is the number of vertices connected to that vertex, and there is no doubt that it is of fundamental importance to nodal numbering schemes, e.g., ERP solvers. However, we found that prioritising nodes with the primitive `ED`, which does not include the vertices already visited when counting the number of vertices connected to a vertex, provides more accurate guidance.

In a level structure system, all vertices in a level are located at the same distances from the root vertex. Since in most ERP solvers based on the level structures nodes are sorted only within each level before moving to the next, node distance from the root is an irrelevant feature for such algorithms. However, in GHH2, while we use the notion of level structures, after the first step of the algorithm, nodes from different levels will be present in the list l. These nodes will thus have different distances from the root node. The primitive `DFSV` captures this information, and use it as a criterion for prioritising vertices.[4] In fact, by incorporating the primitive `DFSV` into the system, we aim to take into consideration the global structure of a graph.

5.4.2 Vertex Selection

Let us analyse Algorithm 5 from a vertex selection point of view. First, a level structure rooted at a suitable starting vertex s (vertex of minimum degree or a *pseudo-peripheral vertex* as explained in Sec. 2.5.4) is constructed (Step 5). Next, an empty list l is formed for each program p in the population (Step 7). The vertex s is then inserted into the first position of array *perm*, and l is updated (Step 9). The update process includes finding all unvisited vertices connected to s and inserting them into l. Note that further vertices will sequentially be assigned to s and inserted in the second, third, etc. positions in *perm*.

Next, the GP interpreter is called k times, where k is the number of vertices in l (Step 12). Each call of the interpreter executes the selected program with respect to the different values returned by `ED` and `DFSV`. The outputs obtained from each execution of the given program are stored in a one-dimensional array, and the permutation associated with the original program tree is produced as described in Sec. 4.2.1 (Step 14). The vertices located in l are then sorted based on the permutation produced (Step 15).

In Step 16, the first element of l is then removed and considered as a new

[4]Sloan [156] also uses a distance quantity in his algorithm, but he computes distances from the end node of a pseudo-diameter.

starting vertex. This process is repeated for each vertex in $V(G_i)$ until all the vertices of graph G_i have been numbered. Finally, *perm* is applied to the adjacency list of the initial graph (or matrix) (Step 18), a new adjacency list is generated, and its envelope is computed (Step 19).

5.4.3 Results

In this work, we used the same training and test sets as described in Sec. 5.3.3 and Sec. 5.3.4. Ten independent runs of GHH2 with the training set were performed, and the corresponding best-of-run individual in each was recorded (see footnote 2 on page 93). The parameters of the runs are given in Table 5.6 (see also footnote 3 on page 95). The best program tree from these ten best-of-run results was then selected as the overall best evolved heuristic.

The simplified version of the best heuristic evolved by GHH2 is as follows:

```
(((((((DFSV + ED) + (ED * ED)) * (((DFSV * ((0.616301555473498 + (DFSV
* (DFSV - -0.156489470580821))) + ((DFSV - -0.778113556456805) * ((DFSV
* 0.680593788009413) + (DFSV * ED))))) - (DFSV - -0.778113556456805))
- 0.273723254573403)) - 0.616301555473498) + (((ED - ED) + DFSV)
- -0.163010843639733)) + ((DFSV + (0.316761550175381 * DFSV)) -
0.889497244679135)) + -0.00709300954225148)
```

This function is also shown graphically in Fig. 5.2. The function is monotonic in both `ED` and `DFSV`. For small values of `ED`, nodes closer to the root are preferred over nodes further away. So, in a highly sparse matrix the algorithm behaves similarly to the RCM algorithm. However, if there are significant differences in `ED` values, the algorithm looks ahead and may prefer a deeper node with a lower `ED` to a closer one with a higher `ED`, thus exhibiting a previously totally unexplored strategy.

Table 5.6: Parameters used in our experiments with GHH2.

Parameter	Value
Maximum Number of Generations	100
Maximum Depth of Initial Programs	3
Population Size	2000
Tournament Size	4
Elitism Rate	0.1%
Reproduction Rate	0.9%
Crossover Rate	70%
Mutation Rate	29%
Mutation Per Node	0.05%

Next, the best heuristic evolved by GHH2 was incorporated into a level structure system, and experiments were carried out in order to evaluate its performance against the RCM and GK algorithms. In practice, both algorithms are still among the best and most widely used methods for envelope reduction. This heuristic was also tested against GHH1-E, which is the best heuristic produced by an envelope-minimising version of GHH1 described in Sec. 5.3. Unlike GHH2's heuristics, GHH1-E is constrained to operate at only one level of a level structure at a time.

A performance comparison of the algorithms under test is shown in Table 5.7. All the results associated with the RCM and GK on the DWT set were taken from [156]. Because there were no results available in the literature for the BCSPWR set, we used a highly enhanced version of the RCM algorithm contained in the MATLAB library to compute the related envelopes. Note that we do not report the results of GK algorithm on the BCSPWR problems as we did not have access to the original code, or a reliable reimplementation.

Table 5.7: Comparison of GHH2's best evolved heuristic against the RCM, GK and GHH1-E.

		Envelope size			
Instance	Dimension	RCM	GK	GHH1-E	GHH2
DWT 59	59×59	314	314	327	**297**
DWT 66	66×66	217	**193**	**193**	194
DWT 72	72×72	**244**	**244**	355	291
DWT 87	87×87	696	682	685	**556**
DWT 162	162×162	1641	**1579**	1611	1610
DWT 193	193×193	5505	**4609**	4851	5196
DWT 209	209×209	3819	4032	3851	**3580**
DWT 221	221×221	2225	2154	2335	**2053**
DWT 245	245×245	4179	3813	4884	**3081**
DWT 307	307×307	8132	8132	8644	**7693**
DWT 310	310×310	3006	3006	3045	**2974**
DWT 361	361×361	5075	**5060**	**5060**	**5060**
DWT 419	419×419	8649	8073	8635	**7411**
DWT 503	503×503	15319	15042	15139	**13759**
DWT 592	592×592	11440	**10925**	11933	11160
DWT 758	758×758	8580	**8175**	8479	8250
DWT 869	869×869	19293	15728	16942	**15296**
DWT 878	878×878	22391	**19696**	22074	21572
DWT 918	918×918	23105	**20498**	22032	22471
DWT 992	992×992	38128	**34068**	37288	37288
DWT 1005	1005×1005	43068	40141	41525	**38107**
DWT 1007	1007×1007	24703	**22465**	24692	24156
DWT 1242	1242×1242	50052	52952	50515	**44666**
DWT 2680	2680×2680	105663	99271	105967	**92500**
Mean		16893.50	15868.83	16710.92	**15384.20**
Wins/Draws		0/1	8/3	0/2	**13/1**
BCSPWR05	443×443	11227	*NA*	10246	**5377**
BCSPWR06	1454×1454	64636	*NA*	55897	**29499**
BCSPWR07	1612×1612	75956	*NA*	65675	**32664**
BCSPWR08	1624×1624	79811	*NA*	80057	**33045**
BCSPWR09	1723×1723	80983	*NA*	76222	**42477**
BCSPWR10	5300×5300	672545	*NA*	655482	**296313**
Mean		164193.00	*NA*	157263.20	**73229.16**
Wins/Draws		0/0	*NA*	0/0	**6/0**

Numbers in bold face are the best results.

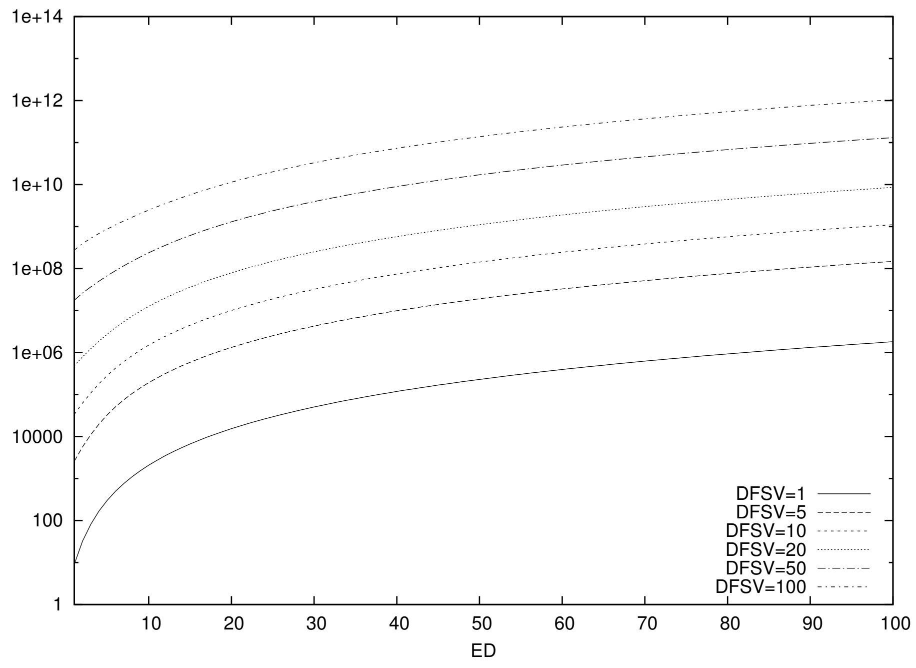


Fig. 5.2: Plot of the best heuristic evolved by GHH2 for the ERP.

As indicated in Table 5.7, the results of GHH2 are extremely encouraging, considering both the mean of the envelope sizes and the number of the best results obtained (presented in the "Wins/Draws" rows). GHH2's best evolved heuristic outperforms the RCM, GK and GHH1-E by a significant margin, and produces extremely good results for the BCSPWR set.

We also measured the time required for our method to solve each problem instance on an AMD Athlon(tm) Dual-core Processor 2.20 GHz. The running times for DWT59 (the smallest instance) and BCSPWR10 (the largest instance) were 0.0307 and 1.0094 seconds, respectively, while the average running time was 0.1605 seconds. This reveals that our evolved algorithm is not only very effective but also extremely efficient.

5.5 Chapter Summary

In this chapter, we have introduced a general framework for genetic hyper-heuristics, which are heuristic search algorithms designed to generate new heuristics for graph layout problems from a set of potential heuristic components. We have also proposed two genetic hyper-heuristic approaches (GHH1 and GHH2) based on our framework for evolving graph-theoretic bandwidth and envelope reduction algorithms. The best heuristic produced by GHH1 for the BRP is very interesting and unconventional, and it essentially goes against the accepted practice of ordering nodes by degree, particularly for large BRP instances. The best heuristic generated by GHH2 for the ERP is also novel, since it incorporates new ideas for using a level structure system without its conventional vertex numbering scheme and employs new features for prioritising vertices.

We tested the generated heuristic algorithms against the best-known human-designed methods from the literature on a large set of standard benchmarks from the DWT and BCSPWR collections. Our algorithms performed extremely well both on benchmark instances from the same class as the training set and also on large problem instances from a totally different class, clearly confirming the effectiveness of the proposed approaches. In addition, the results obtained showed that the evolved heuristics were highly efficient considering the fact that their average running times were very short, i.e., a fraction of a second.

Chapter 6

Evolving a Highly Enhanced Version of the Sloan Algorithm

6.1 Introduction

As we described in Chapter 2, the Envelope Reduction Problem (ERP) is an exceptionally hard graph layout problem, which has a considerable number of applications in various scientific and engineering disciplines. As a result, several methods have been developed for reducing the envelope size of sparse matrices among which the Sloan algorithm, introduced in 1989, offered a substantial improvement over previous algorithms. This sophisticated algorithm is still considered the best option for the solution of the ERP, because it can generally produce quality solutions for different types and sizes of problem instances in a short time. In this chapter, a hyper-heuristic approach based on the framework proposed in Chapter 5 is presented for automatically evolving an enhanced version of the Sloan algorithm. We also present a local search algorithm and integrate it with the new algorithm produced by our hyper-heuristic in order to further improve the quality of the solutions. The new algorithms are evaluated on a large set of standard benchmarks against six state-of-the-art algorithms from the literature. Our algorithms outperform all the methods under test by a wide margin. To the best of our knowledge, no prior attempt to utilize a hyper-heuristic method for evolving ERP solvers has been reported in the literature.

This chapter is organised as follows: Sec. 6.2 briefly describes the general structure of the Sloan algorithm. In Sec. 6.3, our proposed hyper-heuristic method is described in detail. In Sec. 6.4, a hybrid algorithm for tackling the ERP is introduced. Sec. 6.5 provides some information about handling graphs with multiple connected components. In Sec. 6.6, the numerical experiments and the related statistical analysis are discussed. Finally, Sec. 6.7 summarises this chapter.

6.2 The General Structure of Sloan Algorithm

As mentioned earlier, our hyper-heuristic method evolves a variant of the Sloan algorithm. It is therefore useful to first consider the general structure of this algorithm. The two distinct phases of the original Sloan algorithm are as follows.

6.2.1 Finding a Pseudodiameter

The algorithm starts with finding a pseudodiameter and then uses it to provide the second phase (vertex labelling) with a start vertex and a target end vertex. Note that for a graph G, a pseudodiameter can be either a diameter or a shortest path between two nodes in G such that their distance apart is slightly less than the diameter length. Following [63, 67], Sloan found a pseudodiameter by constructing level structures (see Sec. 2.5.4 for more details).

In addition, Paulino et al. [132] proposed the use of the first and last nodes of the spectral permutation to define a pseudodiameter, without constructing level structures. This approach was incorporated into the hybrid algorithm presented by Kumfert and Pothen [107] (see Sec. 2.5.2).

6.2.2 Labelling the Vertices of a Graph

In the labelling phase, the vertices of a graph G are divided into four groups based on their status, i.e., *post-active*, *active*, *pre-active* and *inactive*. Any vertex which has been assigned a new label is defined as post-active. A vertex is considered to be active if it is adjacent to a post-active vertex without having a post-active status. Any vertex adjacent to an active vertex with no post-active or active status is labelled as pre-active. If a vertex is not post-active, active or pre-active, it is inactive.

Then, the start vertex s selected in the first phase is relabelled as vertex one, and a list of eligible vertices to be labelled next is constructed. This list is composed of vertices which are either active or pre-active. The next vertex to be labelled is selected from the list in such a way as to maximise the priority function

$$P(i) = (W_1 \times d\,(i,e)) - (W_2 \times c_i) \quad . \tag{6.1}$$

where W_1 and W_2 are positive integer weights, and c_i is the *current degree* [156, 157] of vertex i. The current degree of a post-active vertex equals zero. For an active vertex, this quantity is equal to the number of its pre-active neighbours. Each pre-active vertex has a current degree equal to the number of its pre-active and inactive neighbours plus one, while the current degree of an inactive vertex equals its degree plus one.

In the first term of the priority function, $d\,(i,e)$ is the distance between vertex i and the target end vertex e (selected in the first phase). Considering $P(i)$, it is clear that a small c_i and a large $d\,(i,e)$ ensures a high priority. After the selection of a vertex for labelling, it is deleted from the list, and then the list is updated based on the connectivity information for the graph. The second phase of the algorithm is repeated until all the vertices have been labelled. Note that Sloan recommends the pair $(1,2)$ for $(W_1,\ W_2)$, and updates the priority function by incrementing it by W_2 (*updating value*). However, the best choice for the weights seems to be problem-dependent.

6.3 Our Hyper-Heuristic Method

In order to automatically generate an enhanced version of Sloan's algorithm, we designed a Genetic Hyper-Heuristic (GHH3) by utilising the general framework presented in Chapter 5. Algorithm 6 shows how GHH3 works.

As mentioned in Sec. 6.2.2, the Sloan algorithm has a priority function for determining the next vertex to be labelled at each stage. This priority function together with its updating value play a critical role in influencing the quality of the solutions obtained. For example, by setting $W_1 = 0$ and $W_2 = 1$ in the priority function, Sloan's robust algorithm becomes almost identical to the King algorithm [93], which is not considered a powerful ERP solver, and may produce very poor orderings. Therefore, we decided to focus on the priority function and the related updating value of the Sloan algorithm rather than the general structure of the algorithm.

In our system, GHH3 is given a training set of matrices as input, and it produces an improved version of the Sloan algorithm as its output. To cope with such a complex task, following the strategy described in Sec. 5.3, we provide GHH3 with the "skeleton" of the Sloan algorithm, and then we ask GHH3 to evolve the "brain" of this algorithm, which is its decision-making element or in other words, its priority function and updating value. The best algorithm created by the GHH3 in our experiments is called $S^\star$.

Algorithm 6 Our proposed hyper-heuristic method (GHH3).

```
1: Randomly generate an initial population of programs from $\{primitives\}$.
2: Assign each program $p$ in the population a floating-point random constant $c_p$.
3: repeat
4:     for each $p \in$ population do
5:         $fitness[p] \leftarrow 0$
6:     end for
7:     for each instance $G_i \in$ training set of sparse matrices do
8:         Find the endpoints of a pseudo-diameter, vertices $s$ and $e$.
9:         Construct a level structure rooted at $e$.
10:        for each $p \in$ population do
11:            for each vertex $v \in V(G_i)$ do
12:                $status[v] \leftarrow$ inactive
13:                $P[v] \leftarrow$ Execute $p$
14:            end for
15:            $status[s] \leftarrow$ preactive; $l \leftarrow$ empty list; $l.add(s)$; $j \leftarrow 1$
16:            repeat
17:                Find vertex $top$ which has the highest priority value.
18:                Update $l$ and $P$.
19:                Insert $top$ into array $perm[j]$; $j$++
20:                Update $l$ and $P$.
21:            until $l.count \neq 0$
22:            Apply $perm$ to the adjacency list of graph $G_i$.
23:            Compute the $envelope$ for the adjacency list thus obtained.
24:            $fitness[p] \leftarrow fitness[p] + envelope(G_i, p)$
25:        end for
26:    end for
27:    Perform selection.
28:    Create a new generation of individual programs.
29: until the maximum number of generations is reached.
30: return the best program tree.
```

Since the key component of GHH3 is the specialised GP system introduced in Chapter 4, obviously the methods employed in GHH3 for initial population generation, fitness evaluation, selection, genetic operations and termination are the same as in that GP system.

Note that in GHH3, during the process of generating the initial population, we assign each program tree in the population a floating-point random constant c_p in the interval $[-1.0, +1.0]$ (Algorithm 6, step 2). In fact, c_p is designed to finally form the updating value of the algorithm evolved by the GHH3. Note also that we mutate the c_p assigned to each program tree. This is simply performed by replacing the value of c_p with another floating-point random constant in the interval $[-1.0, +1.0]$. We applied this type of mutation randomly to 25% of the new individuals produced by the reproduction, crossover and mutation operations.

As in GHH1 and GHH2 presented in Chapter 5, GHH3 computes the fitness of all individuals in a new generation incrementally, by testing the whole population on a problem in the training set before moving to the next (Algorithm 6, step 24). GHH3 also uses the *Tarpeian* method described in Sec. 5.3 for limiting *bloat*.

6.3.1 General and Specialised Primitives in the Primitive Set

The primitive set used in GHH3 together with the definition of each primitive is indicated in Table 6.1. In this set, the general primitives comprise $+$, $-$, $\times$ and uniformly-distributed random constants, which are very common in a GP system. The specialised primitives include `D`, `DFEN` and `FVL`. These are specifically designed for addressing the ERP.

The degree of a vertex reflects the number of vertices connected to that vertex. There is no doubt that the degree of a vertex is of fundamental importance to graph theory and plays a key role in nodal numbering schemes. In GHH3, the primitive

Table 6.1: Primitive set used in GHH3.

Primitive set	Arity	Definition
+	2	Adds two inputs
-	2	Subtracts second input from first input
$*$	2	Multiplies two inputs
D	0	Returns the degree of each vertex
FVL	0	Returns the component of the Fiedler vector of Laplacian matrix associated with each vertex
DFEN	0	Returns the distance from end vertex *e* for each vertex
Constants	$\varnothing$	Uniformly-distributed random constants with floating-point values in the interval $[-1.0, +1.0]$

D returns the degree of each vertex.

The notion of *distance* is an important concept in level structures (see Sec. 2.5.4). In nodal numbering schemes, it is typically used for finding a pseudo-diameter. Sloan [156, 157] employed the distance quantity for determining the endpoints of a pseudo-diameter, and included it in the priority function of his algorithm. In GHH3, the primitive DFEN, which is motivated by the Sloan algorithm returns the distance from end vertex *e* for each vertex. We incorporated DFEN into our system as a primitive with the aim of taking into account the global structure of the graph associated with a given symmetric sparse matrix.

The inclusion of the primitive FVL in the GHH3 is motivated by the *algebraic connectivity of graphs* [15, 56, 125]. Barnard et al. [15] discovered that by sorting the components of the *Fiedler vector* (see Sec. 2.5.4) in monotonically non-decreasing (or non-increasing) order, a permutation vector is generated by which the envelope size of a sparse matrix could be reduced. This means that these components contain valuable information regarding the vertices of a given graph. Therefore, we designed the primitive FVL and included it in our system with the aim of enhancing the performance of the priority function evolved by the GHH3 and also reducing the number of ties.

As shown earlier in Table 6.1, FVL simply returns the component of the Fiedler vector of Laplacian matrix associated with each vertex. The GHH3 produces a real-valued priority function, while the Sloan algorithm and the hybrid algorithm proposed by Kumfert and Pothen [107] use an integer priority function. This is because the primitive `FVL` in our algorithm employs the Fiedler vector which is a real-valued vector.

6.3.2 Vertex Labelling

In this section, we analyse the GHH3 from a vertex labelling point of view as follows:

1. First, the endpoints of a pseudo-diameter, vertices s and e are found (Algorithm 6, step 8), and then a level structure rooted at e is constructed (Algorithm 6, step 9).

2. For each vertex v which is an element of the set of vertices of graph G_i, first, an inactive status is assigned to each vertex (Algorithm 6, step 12), then the interpreter of the GP system is called (N calls in total, where N is equal to the number of vertices in G_i). Each call of the interpreter executes the selected program tree with respect to the different values returned by primitives `D`, `FVL` and `DFEN`. The outputs obtained from each execution of the given program are then stored in a one dimensional array P (Algorithm 6, step 13). In fact, at this stage, we store the initial priority value of each vertex in P.

3. A pre-active status is assigned to s (start vertex), and it is then inserted into an empty list l. In addition, a node count variable $j \in \{1,2,...,N\}$ is initialised to 1 (Algorithm 6, step 15).

4. The main loop of the vertex labelling procedure is a *repeat...until* loop (Algorithm 6, step 16), in which the following steps are performed until l is not empty (termination condition) (Algorithm 6, step 21).

5. The vertices in l are scanned, and vertex $top \in l$ which has the highest priority value is selected (Algorithm 6, step 17).

6. The top is removed from l. If top is not pre-active, then go to the next step. Else, for each vertex k, which is adjacent to top, $P[k]$ is updated (incrementing the priority value of k by c_p). If vertex k is inactive, then it is added to l with a pre-active status (Algorithm 6, step 18).

7. Vertex top is inserted into array $perm$, and its status is changed to post-active. Next, the vertex count j is incremented by 1 (Algorithm 6, step 19). In this step, we actually label the vertex top.

8. For each vertex n, which is adjacent to top, if n is pre-active, then its status is changed to active, and $P[n]$ is updated. Next, for each vertex r, which is adjacent to n, if r is not post-active, then $P[r]$ is updated. Else if r is inactive, then it is added to l with a pre-active status (Algorithm 6, step 20).

Note that after the termination of the process described earlier, a permutation is produced. This permutation is, in fact, stored in array *perm*. The permutation array is then applied to the adjacency list of the initial graph or the rows and columns of the initial sparse matrix (Algorithm 6, step 22). These processes result in the generation of a new adjacency list or a reordered matrix, which is then used for computing the envelope size (Algorithm 6, step 23).

6.3.3 Training Set of Sparse Matrices

In this work, we used a training set comprising 24 benchmark matrices selected from 6 different sets in the Harwell-Boeing sparse matrix collection, 5 of which were the same as in our previous work (see Sec. 5.3.3 for details). The new set employed in the training set was JAGMESH containing finite-element model problems. The benchmark instances of the training set selected for the present work were slightly larger than those reported previously. Overall, sizes ranged from 61×61 to 1089×1089.

6.4 Augmenting $S^{\star}$ with a Local Search Algorithm

We also present a hybrid algorithm for the ERP. This algorithm is a compound of the $S^{\star}$ and our local search algorithm. We call it $S^{\star}$+.

6.4.1 Local Search Algorithm

Algorithm 7 shows the structure of our local search algorithm. The input of this algorithm is an undirected graph $G(V,E)$ with vertex set V, edge set E and a labelling of vertices $f : V \to \{1,\ldots,n\}$, where $n = |V|$. In Algorithm 7, for each vertex v which is an element of the set V, first, a set of suitable swap vertices X is constructed.

Marti *et al.* [121] in their Tabu Search approach designed for addressing the BRP introduced three quantities for generating the set X as follows: the largest vertex label (l) in the set of vertices adjacent to vertex v, the smallest vertex label (s) in the set of vertices adjacent to vertex v, and the best label (m) for v in the current labeling f which is equal to $\lfloor (l+s)/2 \rfloor$. The set X is then defined as $X = \{x : |m - f(x)| < |m - f(v)|\}$. We have also adopted this strategy in our algo-

Algorithm 7 Our Local Search Algorithm.

```
1: repeat
2:     for each vertex $v \in V(G)$ do
3:         Find quantities $l$ and $s \in \{Adjacent(v)\}$.
4:         $m \leftarrow \lfloor (l+s)/2 \rfloor$
5:         Find a set of suitable swap vertices $X$.
6:         for each vertex $x \in X$ do
7:             $swap(f(v), f(x))$.
8:             Compute the envelope size.
9:             if $envelope\_after\_swap \leq current\_envelope$ then
10:                $current\_envelope \leftarrow envelope\_after\_swap$
11:                $break$
12:            else
13:                $swap(f(v), f(x))$.
14:            end if
15:        end for
16:    end for
17: until the termination condition is met.
```

rithm for finding a set of suitable swap vertices (Algorithm 7, steps 3-5).

After the determination of set X, for each vertex x which is an element of X, we swap the label of vertex v with the label of vertex x in the adjacency list of graph G, and then compute the envelope size for the updated adjacency list obtained (Algorithm 7, steps 7-8).

If the envelope size obtained after swapping is less than or equal to the current envelope size (considering a pass rate of 25% for equal cases), then the current envelope size is assigned the new envelope size, and the related loop is terminated. Else, the label of vertex v is swapped with the label of vertex x in order to reverse the change made to the adjacency list of graph G in step 7 and restore the previous state (Algorithm 7, steps 9-14).

This whole process is repeated until the termination condition is met (Algorithm 7, step 17). The termination condition could be a fixed number of iterations, a number of non-improving iterations, CPU time etc.

The motivation behind this local search method is to refine the ordering produced by $S^{\star}$. However, as our method is general, it could be integrated with any of other ERP solvers in order to further improve the quality of the solutions.

It is useful to mention here that in addition to the point-swap operator used in our local search algorithm, there are more sophisticated operators such as 2-opt [40] and PMX [71], which can be generally employed to deal with permutations in evolutionary algorithms.

6.4.2 Speeding up the Search

Although our local search algorithm has a simple structure, its application to larger problem instances might be very time-consuming. This is mainly because each swap operation should be performed on the adjacency list of a given graph, and this calls for a full scan of the whole adjacency list. In addition, after each swap, the envelope size for the updated adjacency list should be computed in order to evaluate the effect of that swap. It is also obvious that by increasing the dimension of a graph, the number of swaps performed increases.

In order to deal with this issue, we have developed a new data structure for the representation of graphs. This data structure is capable of substantially reducing the number of operations, required for swapping a large number of vertices and updating the envelope size after each swap in a dense graph, which leads to a very significant decrease in computational time. Considering the lack of direct relevance of this data structure to the present research, we skip the related details here.

Algorithm 8 $S^{\star}$+.

```
1: Run $S^{\star}$.
2: Apply the permutation returned by $S^{\star}$ to the adjacency list of a given graph.
3: $OrderedAdj \leftarrow$ the adjacency list obtained in step 2.
4: Compute the envelope size.
5: $current\_envelope \leftarrow$ the envelope size obtained in step 4.
6: Run the local search algorithm.
7: $RefinedAdj \leftarrow$ the adjacency list obtained in step 6.
8: $final\_envelope \leftarrow current\_envelope$
9: Return $RefinedAdj$ and $final\_envelope$.
```

6.4.3 Structure of $S^{\star}$+

As stated above, $S^{\star}$+ is a compound of $S^{\star}$ and the local search algorithm. Algorithm 8 indicates how these two methods are integrated.

We first run $S^{\star}$ on a given problem instance G_i. As mentioned earlier, the output of $S^{\star}$ is a permutation, which is then applied to the adjacency list of G_i, and this leads to the creation of a new adjacency list *OrderedAdj*. Next, we compute the envelope size based on *OrderedAdj* and store it in a variable $current\_envelope$, which is then employed by our local search algorithm (Algorithm 8, steps 1-5).

After this stage, we run our local search algorithm on *OrderedAdj* in order to refine the ordering produced by $S^{\star}$. This process leads to the generation of a refined adjacency list *RefinedAdj*, which is then returned by the hybrid algorithm. Note that the final envelope size returned is computed based on *RefinedAdj* (Algorithm 8, steps 6-9).

6.5 Handling Graphs with Multiple Connected Components

There are cases where the graph associated with a given sparse matrix may have multiple connected components (i.e., its nodes can be partitioned into subsets with no edges between the subsets). Recall that in Sec. 5.3.4 we have seen six benchmark matrices of this type included in the DWT set. In this section, a detailed discussion on this issue is given as follows.

A single level structure cannot be constructed for such graphs and, so, in principle most of the envelope reduction algorithms (as well as bandwidth reduction algorithms) discussed in Sec. 2.5 would not be applicable. However, in practice this problem is easily solved. Implementations typically handle the multiple components in two (logically equivalent) ways. Some first do a traversal of the graph associated with the matrix to identify the connected components and then apply their core envelope reduction method to the components. Others identify and deal with the components at run-time. That is, when they find that a level structure has entirely been explored but there are still unlabelled nodes in the graph, they build a new level structure for the next component and restart adding nodes one by one, until all the components are dealt with.

We should note, however, that algorithms that deal with multiple components this way are actually hybrids of a connected component detection algorithm and an envelope reduction algorithm. Of course, the really difficult task (and the focus of this work) is envelope reduction, not connected component detection. Nonetheless, in the presence of multiple components, the connected component detection element of an algorithm typically contributes to a marked reduction of the envelope size of a matrix *on its own*. The reason is that this element orders nodes in the output permutation by component, which corresponds to turning the matrix into

Table 6.2: The influence of block-diagonalisation process on the performance of envelope reduction algorithms.

DWT	Number of Components	Original Envelope Size	Envelope Size after Block-diagonalisation	Envelope Size after RCM
DWT 198	6	5817	1707	1409
DWT 234	7	1999	1939	1596
DWT 346	4	9054	8392	8184
DWT 492	2	34282	18348	7294
DWT 512	32	6530	5471	5449
DWT 607	4	30615	27711	17907

block diagonal form. So, even if one kept the original ordering of the nodes within each component, the envelop size would be reduced.

In order to illustrate this, we report the envelope size of the 6 DWT matrices with multiple components before and after block-diagonalisation (via connected-component detection) as shown in Table 6.2.

To avoid contaminating results with a mix of the performance of the two algorithms, in this study we do not consider benchmark matrices that their corresponding graphs have more than one component. However, our proposed envelope reduction algorithms could trivially be extended with a connected-component detection wrapper and would, thus, have no problems dealing with multiple components.

6.6 Numerical Experiments

In this section, we present the experiments carried out on the GHH3, $S^{\star}$ and $S^{\star}+$. In these experiments, we employed the DWT and BCSPWR sets described in Sec. 5.3.4, as test sets on which to compare the proposed algorithms with six high-performance algorithms. Our algorithms were implemented in C#. With the exception of the quicker runs reported in Section 6.6.2, which were performed on an Intel Xeon(R) X5680 3.33 GHz computer, all the experiments were performed on an AMD Athlon(tm) Dual-core Processor 2.20 GHz. We report the envelope size obtained for each of the benchmark problems as well as the time required to solve each problem instance on this computer. The results associated with the RCM, GK, Sloan, Sloan-MGPS and Hybrid-Perm algorithms on DWT set were taken from [147, 156, 157], and for the Spectral algorithm, we used MATLAB to compute the related envelopes.

Since there were no results reported in the literature on BCSPWR set, MATLAB library and our own implementation of the Sloan algorithm were used in order to produce the results related to the RCM, Spectral and Sloan algorithms respectively. MATLAB is a powerful and reliable tool for scientific computing, and our Sloan algorithm's implementation is also very reliable, such that it can exactly produce all the results reported in Sloan's paper. In terms of the Gibbs-King algorithm (GK), we did not have access to the original code or a trustworthy software package to report the results on BCSPWR.

The algorithms under test were programmed in different languages and run on different machines. Therefore, it was not possible for us to draw meaningful comparisons between their run-times. However, as mentioned earlier, we report all the run-times associated with our algorithms.

6.6.1 Experiments on GHH3 and $S^\star$

Using the training set specified in Sec. 6.3.3 and the parameters listed in Table 6.3, we executed ten independent runs of the GHH3 (see footnote 2 on page 93). Note that the parameters of these runs were originally selected in our previous work (see Sec. 5.3) after conducting experiments on the BRP, considering both the quality of solutions and run times. In general, these parameters produced good-quality solutions in reasonable time also for the ERP. We will further explore the dependency

Table 6.3: Parameters of the runs related to GHH3.

Parameter	Value
Maximum number of generations	100
Maximum depth of initial programs	3
Population size	2500
Tournament size	4
Elitism rate	0.1%
Reproduction rate	0.9%
Crossover rate	80%
Mutation rate	19%
Mutation per node	0.05%
Constant (c_p) mutation rate	25%

of performance on parameter values in Sec. 6.6.2.

Note also that elitism, reproduction, mutation and crossover are mutually exclusive operators, and so their rates of application must add up to 100%. Since we chose a crossover rate of 80%, the remaining 20% is shared (with unequal proportions) between mutation, reproduction and elitism. With the parameters reported in Table 6.3, each run took approximately 9 days of CPU time.

We selected as our overall best evolved heuristic the best program tree (together with its associated constant value c_p) out of the best results obtained in our runs. The simplified version of the best evolved heuristic, h, and its related c_p value are as follows:

$$\begin{aligned}
h(\mathtt{D},\mathtt{DFEN},\mathtt{FVL}) = & \\
& 0.239680968337 \times \mathtt{DFEN} \\
& -0.706902944552 \times \mathtt{FVL} \\
& -0.767697461307 \times \mathtt{D} \\
& -\mathtt{DFEN} \times \mathtt{FVL} \\
& +0.382742287304 \times \mathtt{D} \times \mathtt{DFEN} \times \mathtt{FVL}^2 \\
c_p \quad = \quad & 0.792495691141345
\end{aligned}$$

Simplification was performed using the Maxima symbolic algebra system.

Let us briefly study the function h. As we can see, the function has a linear component (the first three terms) and a non-linear element (term 4 and 5). To understand which components dominate, we need to get an idea of the range of the variables `D`, `DFEN` and `FVL`. Since our benchmark matrices are sparse, typical values of `D` are between a few units to about 30. Values of `DFEN` depend on the size of a matrix, but cannot be smaller than 0 and are very rarely bigger than 60 in our benchmarks. Finally, the Fiedler vector is normalised to be a unit vector and its components, `FVL`, have a zero-mean and are rarely bigger than 0.1 in magnitude (the fact that the norm of the Fiedler vector is 1, implies that, for large matrices, on average its components are very very small).

We plotted the function h for these ranges of values and found that it is almost always planar, except for values of `DFEN` above 30 or 40 when it starts developing a concavity. If we do a Taylor expansion of the function h around $\mathtt{D} = 15$, $\mathtt{DFEN} = 30$

and $\texttt{FVL} = 0.0$, we obtain the linear approximation

$$\begin{aligned} \hat{h}(\texttt{D},\texttt{DFEN},\texttt{FVL}) = \\ 0.239680968337 \times \texttt{DFEN} \\ -30.7069029446 \times \texttt{FVL} \\ -0.767697461307 \times \texttt{D} \end{aligned}$$

If we tabulate h and $\hat{h}$ for values of their arguments within the ranges indicated above, we find that these values are very similar. This is illustrated graphically in Fig. 6.1. As one can see, rarely the ranking of nodes produced by $\hat{h}$ would be very different from the one produced by h.

This implies that GHH3 has approximately evolved a weighted averaging heuristic. High distance from the end vertex, small degree and negative Fiedler components are all elements that vote in favour of the insertion of a vertex in a permutation and *vice versa*. However, not all these criteria have the same importance. Generally, FVL plays a minority role except for a small fraction of the nodes where its magnitude is significantly different from 0 or when D and/or DFEN is very small. For a given FVL, since the weight of D is approximately 3 times bigger than the weight of DFEN, one might think that the former is more important in determining rank than the latter. However, DFEN values can be twice as big as D values in our benchmarks. So, the relative importance of the two criteria is quite similar.

We incorporated the best heuristic evolved by the GHH3, h, into the skeleton of the Sloan algorithm in order to form a new algorithm ($S^{\star}$). In fact, the best heuristic obtained and its associated constant value act as a priority function and its updating value in the $S^{\star}$. In order to assess the performance of the $S^{\star}$, it was evaluated on a large set of standard benchmarks against six of the best envelope reduction algorithms from the literature.

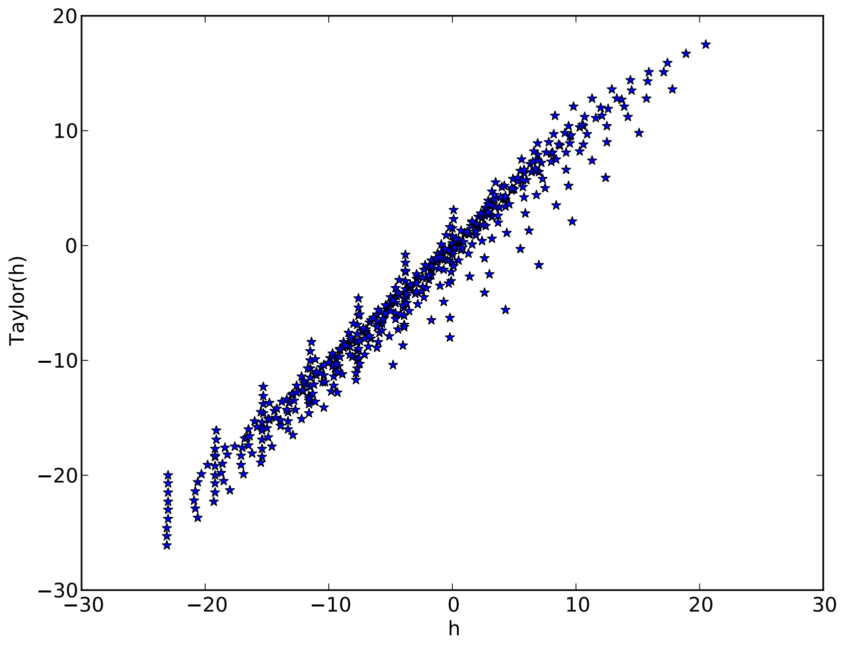


Fig. 6.1: Relationship between the best heuristic function evolved by GHH3, h, and its first order Taylor expansion around $\texttt{D} = 15$, $\texttt{DFEN} = 30$ and $\texttt{FVL} = 0.0$.

Table 6.4 shows a comparison of the $S^\star$ against the RCM, GK, Spectral and Sloan algorithms on DWT and BCSPWR sets. The results reported on both test sets clearly reveal that the $S^\star$ outperforms all four algorithms by a significant margin with respect to the mean of the envelope sizes and the number of the best solutions obtained. In addition to achieving a noticeably low mean envelope size, overall, the $S^\star$ yields the lowest envelope size on 27 benchmark instances out of 34, while the Sloan algorithm produces the lowest envelope size on only 6 benchmark instances. The RCM, GK and Spectral algorithms perform much less favourably than the Sloan and $S^\star$ algorithms. For the reasons mentioned in Sec. 6.6, we do not compare our algorithm with the other algorithms under test in terms of CPU time. However, a simple inspection of the run-times of $S^{\star}$ reported in Table 6.4 reveals that the $S^{\star}$ is not only very effective but also extremely fast.

Table 6.5 demonstrates a comparison of the $S^{\star}$ against the Sloan-MGPS and Hybrid-Perm algorithms on the nine largest Everstine problems. The Sloan-MGPS algorithm is the Sloan algorithm combined with a modified version of the GPS algorithm, and the Hybrid-Perm algorithm is a compound of the Sloan and Spectral algorithms [147] (see Sec. 2.5.2 for more details). The results related to the Sloan-MGPS and Hybrid-Perm algorithms are reported exactly as they appear in the original source, i.e., in thousands. We only report performance on a subset of the test problems because smaller DWT instances were not studied in [147].

Although the envelope sizes produced by these two algorithms are not reported precisely, it is still possible to draw a comparison between the $S^{\star}$ and these methods on the test problems specified. As shown in Table 6.5, the Sloan-MGPS algorithm gives slightly better mean envelope size than the Hybrid-Perm algorithm, while the Hybrid-Perm performs better than the Sloan-MGPS considering the number of the best solutions obtained. On the other hand, the $S^{\star}$ yields a noticeably low mean envelope size in comparison with both algorithms under test, and it also produces the lowest envelope size on 4 benchmark instances (out of a total of 9), which is equal to the number of the best solutions obtained by the Hybrid-Perm algorithm. As it is clear, there is a significant difference between the envelope sizes produced by the $S^{\star}$ and the other two algorithms on the two largest instances of the Everstine problems, i.e., DWT 1242 and DWT 2680.

6.6.2 Sensitivity of GHH3 to Parameter Choices

In order to get some insights as to the extent to which the results obtained by GHH3 are sensitive to the choice of parameters, we performed a series of quicker runs

Table 6.4: A comparison of the $S^{\star}$ against the RCM, GK, Spectral and Sloan algorithms on DWT and BCSPWR sets.

	Envelope size					CPU time
DWT	RCM	GK	Spectral	Sloan	$S^{\star}$	$S^{\star}$
DWT 59	314	314	290	294	**274**	0.0485659
DWT 66	217	**193**	195	**193**	**193**	0.0466174
DWT 72	244	244	245	244	**241**	0.0443534
DWT 87	696	682	622	**525**	549	0.0460990
DWT 162	1641	1579	1751	1554	**1506**	0.0491389
DWT 193	5505	4609	5149	4618	**4487**	0.0506573
DWT 209	3819	4032	3323	3316	**3299**	0.0599485
DWT 221	2225	2154	2030	2052	**1980**	0.0504165
DWT 245	4179	3813	2874	**2676**	3095	0.0564288
DWT 307	8132	8132	7722	7550	**6920**	0.0748792
DWT 310	3006	3006	3102	2982	**2954**	0.0754029
DWT 361	5075	**5060**	5339	5062	5067	0.0628419
DWT 419	8649	8073	8953	7255	**7088**	0.0858158
DWT 503	15319	15042	14814	14436	**13187**	0.1231712
DWT 592	11440	10925	10782	10073	**9691**	0.0868744
DWT 758	8580	8175	7563	7598	**7268**	0.0953332
DWT 869	19293	15728	16261	14909	**14048**	0.1008546
DWT 878	22391	19696	19896	20537	**19639**	0.1304005
DWT 918	23105	20498	18603	**17236**	17289	0.1490151
DWT 992	38128	34068	35748	33928	**33448**	0.2459142
DWT 1005	43068	40141	32613	36396	**31953**	0.2542567
DWT 1007	24703	22465	**21474**	22669	21749	0.1650062
DWT 1242	50052	52952	43661	36822	**35728**	0.2687483
DWT 2680	105663	99271	92513	89686	**87900**	0.5642004
Mean	16893.50	15868.83	14813.46	14275.46	**13731.37**	0.1222891
Wins/Draws	0/0	2/1	1/0	4/1	**19/1**	
BCSPWR01	138	N/A	144	126	**125**	0.0458540
BCSPWR02	263	N/A	197	**166**	174	0.0447663
BCSPWR03	892	N/A	698	575	**571**	0.0449469
BCSPWR04	4605	N/A	3399	2915	**2821**	0.0558754
BCSPWR05	11227	N/A	5238	**3645**	3804	0.0581744
BCSPWR06	64636	N/A	20653	18754	**17473**	0.1637185
BCSPWR07	75956	N/A	21547	18879	**17861**	0.1320252
BCSPWR08	79811	N/A	25014	21621	**19912**	0.1474253
BCSPWR09	80983	N/A	31404	24019	**22014**	0.1637689
BCSPWR10	672545	N/A	177431	194017	**174794**	1.8733869
Mean	99105.60	N/A	28572.50	28471.70	**25954.90**	0.2729941
Wins/Draws	0/0	N/A	0/0	2/0	**8/0**	

N/A = not available
All CPU times are in seconds.
Numbers in bold face are the best results.

Table 6.5: A comparison of the $S^\star$ against the Sloan-MGPS and Hybrid-Perm algorithms on the nine largest Everstine problems.

	Envelope size		
DWT	Sloan-MGPS	Hybrid-Perm	$S^\star$
DWT 758	7.3×10^3	7.5×10^3	**7268**
DWT 869	$\mathbf{13.9} \times 10^3$	15.7×10^3	14048
DWT 878	19.4×10^3	$\mathbf{19.2} \times 10^3$	19639
DWT 918	$\mathbf{17.0} \times 10^3$	17.3×10^3	17289
DWT 992	33.5×10^3	$\mathbf{33.4} \times 10^3$	**33448**
DWT 1005	34.7×10^3	$\mathbf{30.8} \times 10^3$	31953
DWT 1007	22.7×10^3	$\mathbf{20.4} \times 10^3$	21749
DWT 1242	36.5×10^3	39.8×10^3	**35728**
DWT 2680	89.7×10^3	91.4×10^3	**87900**
Mean	30.52×10^3	30.61×10^3	**29891.33**
Wins/Draws	2/0	4/1	4/1

Numbers in bold face are the best results.
The results related to the Sloan-MGPS and Hybrid-Perm algorithms are reported exactly as they appear in the original source, i.e., in thousands. However, for clarity, we have multiplied them by 10^3.

using only one problem instance, the largest instance in the DWT set (DWT 2680), to form a training set. We then tested on the remaining 23 problem instances in the DWT set and on the 10 problem instances in the BCSPWR set. We expected one problem instance to be insufficient to obtain general high-performance solutions. However, to our surprise, as we will see, this was not the case.

In this configuration, we performed 10 independent runs with the parameter settings listed in Table 6.3 and three new sets: one as in Table 6.3 but with a higher mutation rate (39%) and a correspondingly lower crossover rate (60%), one as in Table 6.3 but with a smaller population (of size 500), and, finally, one as in Table 6.3 but with both a higher mutation rate (39%) and smaller population.

Results of our runs are summarised in Table 6.6. For each condition and problem, we report the median, minimum and maximum envelope size recorded when using the best evolved program in each of our 10 runs. It is apparent from this data that varying the parameters within the small range we explored had relatively little influence on the typical end-of-run results.

Also, surprisingly, we see that training on just DWT 2680 was sufficient to produce high quality ERP solvers which generalise to other instances both within and outside the training problem class. We suspect this is due to the fact that DWT 2680 is a finite element model of a Destroyer including a considerable variety of configurations (with its 2680 nodes, 3172 rod elements, 134 triangular plate elements, and 2793 quadrilateral plate elements).

While the results thus obtained are not in the same league as the best program evolved with our larger training set (compare Table 6.6 with Table 6.4), they are not far behind. In particular, they are far superior to Sloan's algorithm. As a result of a smaller training set, runs with populations of 2500 individuals required approximately 17 hours of CPU on our faster computer, while runs with the smaller population typically lasted 3.5 hours.

6.6.3 Experiments on $S^{\star}$+

Table 6.7 presents a comparison of the $S^{\star}$+ against the Sloan and $S^{\star}$ algorithms on DWT and BCSPWR sets. In order to take the non-deterministic nature of the $S^{\star}$+ into consideration, 20 independent runs were executed for each of the selected benchmark instances, and the best, worst and mean envelope sizes obtained were

Table 6.6: The influence of parameter variations on the performance of GHH3.

	population size 2500, 19% mutation, 80% crossover			population size 2500, 39% mutation, 60% crossover			population size 500, 19% mutation, 80% crossover			population size 500, 39% mutation, 60% crossover		
Instance	Median	Best	Worst	Median	Best	Worst	Median	Best	Worst	Median	Best	Worst
DWT 59	284.0	279	290	284.5	274	290	284.5	276	290	283.0	279	289
DWT 66	193.0	193	193	193.0	193	194	193.0	193	197	193.0	193	193
DWT 72	242.5	238	244	238.5	230	247	240.0	228	247	244.0	237	252
DWT 87	534.5	526	557	534.5	528	560	541.0	524	556	534.5	524	560
DWT 162	1523.0	1458	1576	1528.0	1460	1607	1520.0	1456	1613	1526.0	1460	1600
DWT 193	4543.5	4491	4701	4517.5	4493	4556	4515.5	4481	4684	4509.5	4481	4557
DWT 209	3349.0	3099	3404	3287.5	3035	3410	3297.5	3100	3439	3188.0	3114	3404
DWT 221	2007.0	1965	2025	1991.0	1968	2012	1987.0	1956	2028	1998.0	1963	2019
DWT 245	2813.5	2640	2945	2857.5	2688	3575	2863.0	2740	3471	2791.0	2688	3220
DWT 307	7040.0	6935	7380	7039.0	6950	7426	6999.0	6905	7462	7020.5	6928	7206
DWT 310	2952.0	2951	2966	2952.0	2951	2996	2952.0	2951	2992	2952.5	2951	2967
DWT 361	5067.0	5060	5096	5063.5	5060	5089	5063.5	5055	5114	5063.5	5052	5080
DWT 419	7222.0	7081	7337	7224.5	6902	7333	7206.0	6994	7304	7185.5	7017	7320
DWT 503	12920.5	12736	13874	12943.0	12740	13752	13071.0	12700	14072	13029.5	12679	13866
DWT 592	9848.0	9684	10080	9811.5	9684	10045	9854.0	9677	10256	9857.0	9684	10153
DWT 758	7600.0	7343	7749	7455.5	7280	7689	7514.0	7254	8428	7507.5	7281	7700
DWT 869	14455.0	14242	14915	14363.0	13784	14657	14417.0	14095	15480	14514.5	14022	14787
DWT 878	19944.0	19378	20235	19942.5	19099	20169	19778.5	19112	20169	20070.5	19174	20169
DWT 918	17189.5	16987	17891	17220.5	16941	17906	17251.5	16998	18429	17140.5	17004	17598
DWT 992	33686.0	33448	33948	33612.0	33508	33888	33676.0	33448	34044	33888.0	33620	34248
DWT 1005	32852.0	31728	33931	32735.0	31790	34169	32337.0	31480	34082	32867.0	31861	35391
DWT 1007	22446.0	21875	22725	22457.5	21246	22703	22252.0	21288	22703	22571.0	21542	22743
DWT 1242	37412.5	36018	42039	37021.5	35979	40283	36935.0	35775	43861	37204.0	36575	40638
DWT 2680	88616.0	88109	88890	88354.0	88058	88748	88682.0	87986	89258	88758.0	88064	88914
BCSPWR01	126.0	125	128	126.0	122	136	126.0	124	133	126.5	125	135
BCSPWR02	169.0	166	188	168.5	166	186	171.0	166	205	169.5	166	179
BCSPWR03	590.5	578	605	585.0	572	603	593.0	570	618	595.5	572	599
BCSPWR04	2839.0	2677	3965	2817.5	2496	3739	2778.5	2601	5812	2821.0	2531	3938
BCSPWR05	3803.0	3639	4041	3796.5	3617	4029	3836.5	3685	4701	3823.5	3703	3896
BCSPWR06	17944.5	17673	19321	17924.5	17509	18531	18050.5	17467	19516	18196.0	17309	18917
BCSPWR07	18648.5	17591	19212	18718.5	17457	19355	18941.5	18032	21454	18863.5	17366	19499
BCSPWR08	21471.0	20450	22851	20951.5	19352	22185	21231.0	19246	23439	21037.5	19260	22671
BCSPWR09	23561.0	23026	24801	23589.5	21683	24710	23749.0	21428	26129	23999.0	21204	24756
BCSPWR10	189730.0	174727	199343	180866.5	164871	198898	183580.5	168598	218014	185497.0	169147	206049
Mean	18047.74	17326.94	18807.24	17740.32	16902.53	18696.35	17837.87	17017.32	19711.76	17941.93	17052.24	18985.68

recorded. We used the mean envelope size related to each benchmark problem in order to compare $S^{\star}+$ with the Sloan algorithm. As it is clear from Table 6.7, the $S^{\star}+$ is far superior to the Sloan algorithm considering both the mean of the results and the number of the best solutions obtained. The worst results of $S^{\star}+$ are also quite good in comparison with the Sloan's results. This was expected, since the $S^{\star}+$ benefits from a high-performance envelope reduction algorithm ($S^{\star}$) and an effective local search algorithm at the same time. The small standard errors obtained for the means also reflect that the process is very reliable.

In Table 6.7, the quantity Δ refers to the difference between the best envelope size obtained by the $S^{\star}+$ and the envelope size produced by the $S^{\star}$ for each benchmark problem. In other words, the Δ shows the improvement achieved by the $S^{\star}+$ over the $S^{\star}$ on each problem instance. We report the Δ values within 1000 seconds (the termination condition of our local search algorithm). Note that by increasing the time period specified as the termination condition, much better solutions could be obtained by the $S^{\star}+$. For instance, we experimented with the largest benchmark problem in our test sets (BCSPWR10, 5300×5300) and obtained $\Delta = -9038$, which is significantly less than that reported in Table 6.7. However, there is no doubt that this can increase the cost of our algorithm, which might not be desirable especially in a situation, where the execution-time is of critical importance.

6.6.4 Statistical Analysis of the Results

In order to investigate whether or not the relative performance differences observed in our experiments are statistically significant, we carried out non-parametric statistical tests. First, the results obtained by the $S^{\star}$ and $S^{\star}+$ and the results produced by each of the algorithms under test (RCM, GK, Spectral and Sloan) were paired ($S^{\star}$/RCM, $S^{\star}$/GK, etc) on a problem-by-problem basis. Note that as mentioned

Table 6.7: A comparison of the $S^{\star}$+ against the Sloan and $S^{\star}$ algorithms on DWT and BCSPWR sets.

		$S^{\star}$+					
DWT	Sloan	Best	Worst	Mean	SD	SEM	Δ
DWT 59	294	273	273	**273**	0	0	-1
DWT 66	**193**	193	193	**193**	0	0	0
DWT 72	244	236	238	**236.5**	0.67	0.21	-5
DWT 87	**525**	548	548	548	0	0	-1
DWT 162	1554	1506	1506	**1506**	0	0	0
DWT 193	4618	4485	4486	**4485.2**	0.40	0.12	-2
DWT 209	3316	3218	3222	**3219.1**	1.51	0.47	-81
DWT 221	2052	1957	1958	**1957.3**	0.45	0.14	-23
DWT 245	**2676**	3055	3058	3056.2	0.97	0.30	-40
DWT 307	7550	6857	6859	**6858.4**	0.91	0.28	-63
DWT 310	2982	2950	2950	**2950**	0	0	-4
DWT 361	5062	5060	5065	**5061.7**	1.79	0.56	-7
DWT 419	7255	7038	7041	**7039.7**	1.10	0.34	-50
DWT 503	14436	13047	13055	**13048.8**	2.44	0.77	-140
DWT 592	10073	9638	9640	**9639**	0.89	0.28	-53
DWT 758	7598	7250	7258	**7251.6**	2.45	0.77	-18
DWT 869	14909	14031	14035	**14033.2**	1.66	0.52	-17
DWT 878	20537	19585	19592	**19587.7**	2.68	0.84	-54
DWT 918	17236	17107	17123	**17110.6**	5.18	1.63	-182
DWT 992	33928	33448	33448	**33448**	0	0	0
DWT 1005	36396	31493	31500	**31497.6**	2.76	0.87	-460
DWT 1007	22669	21472	21485	**21476.5**	4.15	1.31	-277
DWT 1242	36822	35343	35382	**35359.4**	15.58	4.92	-385
DWT 2680	89686	87534	87587	**87546**	15.49	4.89	-366
BCSPWR01	126	122	122	**122**	0	0	-3
BCSPWR02	**166**	174	174	174	0	0	0
BCSPWR03	575	561	561	**561**	0	0	-10
BCSPWR04	2915	2749	2750	**2749.1**	0.3	0.29	-72
BCSPWR05	**3645**	3727	3732	3729.1	2.21	0.69	-77
BCSPWR06	18754	17104	17122	**17109.5**	6.65	2.10	-369
BCSPWR07	18879	17256	17293	**17270.7**	12.93	4.08	-605
BCSPWR08	21621	19480	19493	**19485.2**	4.48	1.41	-432
BCSPWR09	24019	21468	21529	**21488.7**	22.29	7.05	-546
BCSPWR10	194017	173970	174139	**174015.7**	60.51	19.13	-824
Mean	18450.82			**17179.04**			
Wins/Draws	5/1			**30/1**			

SD = Standard Deviation

SEM = Standard Error of the Mean

$\Delta = S^{\star}+ \text{(Best)} - S^{\star}$

Numbers in bold face are the best results.

Table 6.8: Statistical Analysis of the Results.

Table	Algorithm	Z	Asymp. Sig. (2-tailed)	Exact Sig. (2-tailed)	Exact Sig. (1-tailed)
Table 6.4 (DWT)	$S^{\star}$ — RCM	-4.286	.000	.000	.000
	$S^{\star}$ — GK	-4.136	.000	.000	.000
	$S^{\star}$ — Spectral	-3.714	.000	.000	.000
	$S^{\star}$ — Sloan	-3.315	.001	.000	.000
Table 6.4 (BCSPWR)	$S^{\star}$ — RCM	-2.803	.005	.002	.001
	$S^{\star}$ — Spectral	-2.803	.005	.002	.001
	$S^{\star}$ — Sloan	-1.988	.047	.049	.024
Table 6.7	$S^{\star}+$ — Sloan	-4.315	.000	.000	.000
	$S^{\star}+$ — $S^{\star}$	-4.782	.000	.000	.000

earlier, the results related to the Sloan-MGPS and Hybrid-Perm algorithms are not reported accurately (i.e., divided by 1,000 and probably rounded) in the original source and are only available for a subset of our test set. Therefore, we did not consider them in our statistical analysis.

Then, we ran the Wilcoxon signed-rank test, which is a reliable and widely used non-parametric test for paired data. The tests were performed using the SPSS software package (ver. 16.0). The results of these tests are summarised in Table 6.8. We used the *Exact* method for computing the significance levels of the statistics, since the exact significance is always reliable, regardless of the size, distribution, sparseness, or balance of the data. Note that the Z values reported in Table 6.8 are based on the positive ranks.

The statistical analysis performed on the results confirms the superiority of the $S^{\star}$ and $S^{\star}+$ over the other methods tested by providing the related p-values. A simple inspection of the p-values presented in Table 6.8 clearly demonstrates that the performance differences between our algorithms and these methods are statistically significant.

6.7 Chapter Summary

In this chapter, we have presented a genetic hyper-heuristic approach (GHH3) for automatically evolving a highly enhanced version of the Sloan algorithm, which is a very powerful technique for addressing the envelope reduction problem (ERP). The GHH3 is a search algorithm that explores the space of ERP solvers, and it is based on the general framework introduced in the previous chapter. The new version of the Sloan algorithm produced by the GHH3 benefits from a sophisticated decision-making element as its priority function, which incorporates new ideas inspired by the algebraic connectivity of graphs in order to guide the algorithm towards better solutions.

We have also presented a local search algorithm and integrated it with the best algorithm evolved by the GHH3 in order to further improve the quality of the solutions obtained. Our local search algorithm is a general method. Therefore, it could be integrated with any other envelope reduction solver in order to refine the ordering produced by them. The new version of the Sloan algorithm ($S^\star$) and the hybrid algorithm ($S^\star$+) were evaluated against six of the best envelope reduction algorithms from the literature, namely the RCM, GK, Spectral, Sloan, Sloan-MGPS and Hybrid-Perm on a wide-ranging set of standard benchmark matrices from the Harwell-Boeing sparse matrix collection. Both our algorithms showed remarkable performance on benchmark matrices from the same class as the training set as well as problem instances from a completely different class. This confirms the ability of our algorithms to generalising to unseen situations. The $S^\star$ algorithm is also computationally very fast.

Chapter 7

Conclusions

This chapter presents some conclusions on the work described in this thesis and suggests directions for future research.

7.1 Motivation and Key Result of the Thesis

As we have seen, graph layout problems belong to an important class of combinatorial optimisation problems with a large number of applications in various domains. Therefore, many algorithms have been proposed in the literature for the solution of this class of problems among which only graph-theoretic heuristics are appropriate for practical use. These heuristic algorithms are generally fast and reliable. However, they are very hard and time-consuming to design, and there is usually plenty of room for improvement in the solutions that they produce.

This motivated us to develop algorithms, termed *Genetic Hyper-Heuristics*, capable of automatically generating more effective, accurate and generally applicable graph theoretic heuristics for addressing graph layout problems. Two very important graph layout problems, namely *bandwidth* and *envelope* reduction problems

(BRP and ERP) were employed as testbeds, and a substantial number of experiments were then carried out in order to determine the effectiveness of the heuristics generated. The results obtained showed that the evolved heuristic algorithms could outperform state-of-the-art human-designed algorithms. This confirms that the main objective of this study has been achieved.

In the following section we will review the contributions of this thesis in more detail, and in Sec. 7.3 we will describe possible avenues for future work.

7.2 Detailed Contributions of the Thesis

7.2.1 Specialised GP System for Graph Layout Problems

In Chapter 4, we have introduced a specialised genetic programming system, which is in fact the first GP system designed to solve graph layout problems. It is a tree-based GP system capable of transforming program trees into permutations, which are then applied to the adjacency list of the initial graph (or the initial matrix) in order to address the target problem. The proposed GP system can be utilised both as a meta-heuristic and as a hyper-heuristic. The aim of developing this GP system was to form the central organising element of genetic hyper-heuristics (the major contribution of this study).

7.2.1.1 GP as a Meta-heuristic for the BRP and ERP

We initially applied the GP system (as a meta-heuristic) to the BRP. In order to test its performance, we compared it with two high-performance BRP solvers, i.e., the RCMM and Spectral. The results obtained were very encouraging. For further analysis, and to check its applicability to other graph layout problems, we then applied it to the ERP, but this time we used much larger and more diverse benchmark problem instances. We evaluated our method against four of the best performing ERP solvers, i.e., the RCM, GK, Spectral and Sloan. In terms of reducing the envelope size, the proposed GP approach outperformed the four reference algorithms in most cases. However, it performed poorly compared to the algorithms under test (which are graph-theoretic heuristics) in terms of execution speed. This is not surprising, given that our meta-heuristic approach requires runs of a GP system. Note that previous studies on similar problems have generally reported that successful graph-theoretic heuristic algorithms can be thousands to hundreds of thousands of times faster than corresponding meta-heuristics.

7.2.2 Genetic Hyper-Heuristics

In Chapter 1, we have defined a Genetic Hyper-Heuristic (GHH) as a heuristic search algorithm which explores the space of problem solvers. The key component of GHHs is the specialised GP system described earlier. In Chapter 5, we have presented a general framework for genetic hyper-heuristics. This framework is actually the first hyper-heuristic framework that has the ability to automatically generate heuristic algorithms for graph layout problems. The proposed framework is an offline learning methodology that produces new reusable heuristics by acquiring knowledge from a set of training instances.

More specifically, we first applied a GHH approach to an independent training set of graphs/matrices (learning phase), and we then employed the best evolved heuristic in order to directly solve the target problem instance(s), which would obviously be from a totally different dataset. As mentioned above, GHHs were originally designed to produce reusable heuristics, since having such heuristics increases the speed of solving new unseen instances of graph layout problems. However, a GHH can also easily be used as an online learning method to produce disposable heuristics.

In this thesis, we have presented three hyper-heuristic approaches, namely GHH1 (Chapter 5), GHH2 (Chapter 5) and GHH3 (Chapter 6). These are, in fact, specialisations of the proposed genetic hyper-heuristic framework.

GHH1 was designed to generate graph-theoretic heuristic algorithms based on the level structures for the BRP. The common components of level-structure-based solvers (i.e., a level structure system plus a method for finding a suitable starting vertex) were incorporated into the GHH1. The algorithm was then asked to evolve the key element of such solvers (i.e., a numbering scheme) using a set of primitives compatible with the algorithm. More specifically, in GHH1 we used a small set of standard primitives (addition, subtraction, multiplication and random constants) and two *problem-specific specialised* primitives, i.e., `SDV` and `N`. The best heuristic generated by GHH1 is very interesting and unconventional, since it goes against the accepted practice of ordering vertices by degree, especially for large problem instances.

Following the strategy adopted in GHH1, we designed GHH2 to produce graph-theoretic heuristic algorithms for the ERP. Unlike GHH1, GHH2 is capable of exploring and prioritising vertices located at different depths in a level structure. Therefore, although GHH2 uses the notion of level structures, the algorithms generated by GHH2 are entirely different from conventional level-structure-based nodal numbering schemes. In GHH2, we used the same set of standard primitives as in GHH1 as well as two specialised primitives, i.e., `ED` and `DFSV`.

GHH3 was designed with the aim of automatically generating an enhanced version of Sloan's algorithm (a very powerful ERP solver). We provided GHH3 with the skeleton of the Sloan algorithm, and then we asked GHH3 to evolve the decision-making element of this algorithm or, in other words, its priority function and updating value. Three specialised primitives, i.e., `D`, `DFEN` and `FVL` were employed for this purpose. The new version of the Sloan algorithm evolved by the GHH3 ($S^{\star}$) benefits from a sophisticated decision-making element, which incorporates new ideas inspired by the algebraic connectivity of graphs in order to guide the algorithm towards better solutions.

We evaluated the best heuristic algorithms generated by GHH1, GHH2 and GHH3 against state-of-the-art human-designed BRP and ERP solvers, namely the RCM, RCMM, GK, Spectral, Sloan, Sloan-MGPS and Hybrid-Perm on a wide-ranging set of standard benchmark matrices from the DWT and BCSPWR collections. Our algorithms showed remarkable performance on benchmark matrices from the same class as the training set as well as problem instances from an entirely different class. This clearly confirms that our algorithms are capable of generalising to unseen situations. The results obtained also indicated that the evolved heuristics were highly efficient, having very short average run times.

Our experiments with the GHH3 have also led to the unexpected conclusion that training on just one problem instance may be sufficient to generate high quality reusable heuristics, which generalise to unseen problems, provided that the selected problem instance includes a variety of configurations. This is interesting, since it contradicts conventional expectations. This is also important, because by adopting this strategy one can significantly reduce the CPU time required for running hyper-heuristic algorithms.

7.2.3 Meta-heuristic GP versus Hyper-heuristic GP

As we mentioned before, in this thesis we have used genetic programming (GP) as a meta-heuristic (in Chapter 4) and also as a hyper-heuristic (in Chapter 5 and Chapter 6). In order to draw a direct comparison between these two methodologies, we selected two of our experiments, which were performed under exactly the same conditions. Table 7.1 illustrates the results associated with the experiments chosen. In this table, GPb represents the best result obtained by our specialised GP system (meta-heuristic GP) for each benchmark problem in 20 independent runs, and $S^{\star}$ represents the best envelope reduction algorithm created by our genetic hyper-heuristic approach, i.e., GHH3.

The analysis of the results reported on the DWT set shows that the $S^{\star}$ is far superior to the GPb configuration, taking into consideration the mean of the envelope sizes, the number of the best solutions obtained, and the run times. It should be emphasized that the results associated with the GPb configuration and the $S^{\star}$ (both the envelope sizes and their corresponding CPU times) for each benchmark problem were obtained in *20 runs* (the best of which was selected for this comparison) and *one single run*, respectively.

In Sec. 2.4 and Sec. 3.2, the main disadvantages of employing meta-heuristic approaches for addressing real-world problems in general and graph layout problems in particular were described. Here, we have provided evidence that supports those arguments. With respect to the results obtained in this study, we encourage the use of GP as a hyper-heuristic for the solution of graph layout problems as well as other similar problems.

7.2.4 Proposed Local Search Algorithm

In Chapter 6, we have proposed a local search algorithm. Unlike many previous studies on graph layout problems, we did not employ our local search method as a stand-alone solver. The reason was that previous attempts (including ours) had not met with a great deal of success. The motivation for the development of this local search algorithm was to refine the ordering produced by $S^{\star}$ (the enhanced version of the Sloan algorithm generated by GHH3). Therefore, we integrated it with the

Table 7.1: A comparison of the $S^{\star}$ against the GPb configuration on DWT set.

	Envelope size		CPU time	
DWT	GPb	$S^{\star}$	GPb	$S^{\star}$
DWT 59	278	**274**	12.01	**0.0485659**
DWT 66	**193**	**193**	12.65	**0.0466174**
DWT 72	**233**	241	13.23	**0.0443534**
DWT 87	**520**	549	22.18	**0.0460990**
DWT 162	**1506**	**1506**	30.22	**0.0491389**
DWT 193	4604	**4487**	54.22	**0.0506573**
DWT 209	**3125**	3299	59.32	**0.0599485**
DWT 221	**1976**	1980	69.21	**0.0504165**
DWT 245	**2626**	3095	68.97	**0.0564288**
DWT 307	7275	**6920**	107.69	**0.0748792**
DWT 310	2976	**2954**	44.58	**0.0754029**
DWT 361	**5060**	5067	74.92	**0.0628419**
DWT 419	7577	**7088**	155.82	**0.0858158**
DWT 503	13581	**13187**	207.36	**0.1231712**
DWT 592	10193	**9691**	278.20	**0.0868744**
DWT 758	7369	**7268**	332.28	**0.0953332**
DWT 869	14260	**14048**	417.18	**0.1008546**
DWT 878	**19194**	19639	484.76	**0.1304005**
DWT 918	17885	**17289**	610.15	**0.1490151**
DWT 992	**33134**	33448	540.37	**0.2459142**
DWT 1005	**31494**	31953	604.85	**0.2542567**
DWT 1007	**20650**	21749	745.94	**0.1650062**
DWT 1242	41220	**35728**	881.73	**0.2687483**
DWT 2680	91083	**87900**	3900.50	**0.5642004**
Mean	14083.83	**13731.37**	405.3475	**0.1222891**
Wins/Draws	10/2	**12/2**	0/0	**24/0**

GPb = the best result obtained by our specialised GP system in 20 runs
$S^{\star}$ = the best envelope reduction algorithm generated by GHH3
Numbers in bold face are the best results.

$S^\star$ to form a hybrid algorithm for the ERP (i.e., $S^\star$+).

In order to test the effectiveness of the proposed local search algorithm, $S^\star$+ was compared against the $S^\star$ and Sloan algorithms. Even the worst results obtained by $S^\star$+ were extremely good compared to the Sloan's results. The improvement achieved by the $S^\star$+ over the $S^\star$ on each problem instance was also very significant, confirming the efficacy of the local search algorithm. Note that our local search algorithm is a general method, and that it could be integrated with any other nodal numbering scheme for further improving the quality of its solutions.

7.2.5 Some Insights into Nodal Numbering Schemes

This study has also led to some insights regarding the performance of nodal numbering schemes (e.g., BRP and ERP solvers):

1. Undoubtedly, the degree of a vertex is of fundamental importance to nodal numbering schemes. However, we found that this quantity might fail to provide accurate guidance in the process of prioritising nodes for the construction of permutations. We have introduced two specialised primitives (`SDV` and `ED`) to deal with this issue.

2. Taking into consideration the global structure of a graph could have a very beneficial influence on the performance of a numbering scheme. We have presented two specialised primitives (`DFSV` and `FVL`) for this purpose.

3. The effect of ties (vertices with the same degree) on the performance of nodal numbering schemes might be much worse than expected, especially for large problem instances. By incorporating the specialised primitives presented in this thesis into the decision-making elements of such schemes, this effect could be reduced considerably.

4. We have shown the possibility and promise of utilising a level structure system without its conventional nodal numbering scheme.

5. We have shown that in graphs with multiple components, the connected component detection element of an ERP solver could contribute to a significant reduction in the envelope size of a given matrix *on its own*. We believe that the same could be true for a BRP solver and the bandwidth.

7.3 Future Work

In this section, I suggest some research directions for extending the work presented in this thesis.

Specialised GP System

- Building a parallel implementation of the current GP system for improving its execution speed.
- Investigating if the performance could be further enhanced by mutating constants, using non-random populations and hybridising the GP system by incorporating a local optimiser.
- Employing machine learning techniques in order to better guide the generation of new individuals.

Genetic Hyper-Heuristics

- Applying the methodology presented in this thesis to other graph layout problems with the aim of automatically creating more effective heuristic algorithms for addressing them.

- Further automation of the design process in order to reduce the cost associated with identifying suitable heuristic components.

- Further extending the primitive set used and designing new primitives which contain more general information.

- Investigating the effect of training set on the performance of the generated algorithms in detail.

- Investigating the possibility of speeding up the heuristic generation process by adopting different strategies, such as estimating the fitness value of individual program trees in the population without the need for executing them, implementing a distributed version of the proposed GP system and intelligently selecting smaller but more effective training sets.

Local Search Algorithm

- Minimising the number of swap operations by predicting the effect of a swap on the layout cost before performing it.

- Replacing the set X (the set of suitable swap vertices) with a more effective one, which may be generated by a genetic programming approach.

Nodal Numbering Schemes

- Investigating the influence of block-diagonalisation process on the performance of bandwidth reduction algorithms.

References

[1] D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973.

[2] D. L. Adolphson. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing*, 6(1):40–54, 1977.

[3] S. Allen, E. K. Burke, M. Hyde, and G. Kendall. Evolving reusable 3D packing heuristics with genetic programming. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 931–938, New York, NY, USA, 2009. ACM.

[4] B. A. Armstrong. Near minimal matrix profiles and wavefronts for testing nodal resequencing algorithms. *International Journal for Numerical Methods in Engineering.*, 21:1785–1790, 1985.

[5] M. Bader-El-Den and R. Poli. Generating SAT local-search heuristics using a GP hyper-heuristic framework. In N. Monmarché, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, editors, *Evolution Artificielle, 8th International Conference*, volume 4926 of *Lecture Notes in Computer Science*, pages 37–49, Tours, France, 29-31 Oct. 2007. Springer.

[6] M. Bader-El-Den and R. Poli. Evolving effective incremental solvers for SAT with a hyper-heuristic framework based on genetic programming. In R. L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice VI*, Genetic and Evolutionary Computation, chapter 11, pages 163–179. Springer, Ann Arbor, 15-17May 2008.

[7] M. Bader-El-Den and R. Poli. Evolving heuristics with genetic programming. In M. Keijzer, G. Antoniol, C. B. Congdon, K. Deb, B. Doerr, N. Hansen, J. H. Holmes, G. S. Hornby, D. Howard, J. Kennedy, S. Kumar, F. G. Lobo, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, J. Pollack, K. Sastry, K. Stanley, A. Stoica, E.-G. Talbi, and I. Wegener, editors, *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 601–602, Atlanta, GA, USA, 12-16 July 2008. ACM.

[8] M. Bader El Den and R. Poli. Grammar-based genetic programming for timetabling. In A. Tyrrell, editor, *2009 IEEE Congress on Evolutionary Computation*, pages 2532–2539, Trondheim, Norway, 18-21 May 2009. IEEE Computational Intelligence Society, IEEE Press.

[9] M. B. Bader-El-Den and R. Poli. A GP-based hyper-heuristic framework for evolving 3-SAT heuristics. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1749–1749, London, 7-11 July 2007. ACM Press.

[10] M. B. Bader-El-Den, R. Poli, and S. Fatima. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3):205–219, 2009.

[11] R. Bai, E. Burke, M. Gendreau, G. Kendall, and B. McCollum. Memory length in hyper-heuristics: An empirical study. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (SCIS '07)*, pages 173 –178, april 2007.

[12] R. Bai, E. K. Burke, and G. Kendall. Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. *Journal of the Operational Research Society*, 59(10):1387–1397, 2008.

[13] W. Banzhaf, F. D. Francone, R. E. Keller, and P. Nordin. *Genetic Programming – An introduction; On the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[14] S. T. Barnard, A. Pothen, and H. Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numerical Linear Algebra with Applications*, 2(4):317–334, 1995.

[15] S. T. Barnard, A. Pothen, and H. D. Simon. A spectral algorithm for envelope reduction of sparse matrices. In *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 493–502, New York, NY, USA, 1993. ACM.

[16] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300 – 343, 1984.

[17] R. A. Botafogo. Cluster analysis for hypertext systems. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '93, pages 116–125, New York, NY, USA, 1993. ACM.

[18] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Number XVI in Genetic and Evolutionary Computation. Springer, 2007.

[19] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward. A classification of hyper-heuristic approaches. *Handbook of Metaheuristics*, pages 449–468, 2010.

[20] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, chapter 16, pages 457–474. 2003.

[21] E. Burke, B. MacCarthy, S. Petrovic, and R. Qu. Knowledge discovery in a hyper-heuristic for course timetabling using case-based reasoning. In E. Burke and P. Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 276–287. Springer Berlin Heidelberg, 2003.

[22] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. A survey of hyper-heuristics. Technical report, School of Computer Science, University of Nottingham, 2009.

[23] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942–958, Dec. 2010.

[24] E. K. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervos, L. D. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *LNCS*, pages 860–869, Reykjavik, Iceland, 9-13 Sept. 2006. Springer-Verlag.

[25] E. K. Burke, M. R. Hyde, and G. Kendall. Grammatical evolution of local search heuristics. *Evolutionary Computation, IEEE Transactions on*, 16(3):406 –417, june 2012.

[26] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In J. Kacprzyk, L. C. Jain, C. L. Mumford, and L. C. Jain, editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg, 2009.

[27] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1559–1565. ACM, 2007.

[28] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, 20(1):63–89, Spring 2012.

[29] E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward. The scalability of evolved on line bin packing heuristics. In D. Srinivasan and L. Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 2530–2537, Singapore, 25-28 Sept. 2007. IEEE Computational Intelligence Society, IEEE Press.

[30] E. K. Burke, G. Kendall, D. L. Silva, R. O'Brien, and E. Soubeiga. An ant algorithm hyperheuristic for the project presentation scheduling problem. In *IEEE Congress on Evolutionary Computation*, volume 3, pages 2263–2270. IEEE, sept. 2005.

[31] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[32] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177 – 192, 2007.

[33] E. K. Burke, S. Petrovic, and R. Qu. Case-based heuristic selection for timetabling problems. *J. of Scheduling*, 9(2):115–132, 2006.

[34] P. Z. Chinn, J. Chvátalová, A. K. Dewdney, and N. E. Gibbs. The bandwidth problem for graphs and matrices — a survey. *Journal of Graph Theory*, 6(3):223–254, 1982.

[35] G. D. Corso and G. Manzini. Finding exact solutions to the bandwidth minimization problem. *Computing*, 62(3):189–203, 1999.

[36] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002*

Congress - Volume 02, CEC ’02, pages 1185–1190, Washington, DC, USA, 2002. IEEE Computer Society.

[37] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin / Heidelberg, 2001.

[38] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristic International Conference, MIC 2001*, pages 127–131, 2001.

[39] P. I. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, pages 1–10, London,UK, 2002. Springer-Verlag.

[40] G. Croes. A method for solving Traveling Salesman Problems. *Operations Research*, 6(6):791–812, 1958.

[41] P. Cross. A fill-in storage technique for geodetic normal equations. *Journal of Geodesy*, 54(4):503–509, 1980.

[42] A. Cuesta-Cañada, L. Garrido, and H. Terashima-Marín. Building hyper-heuristics through ant colony optimization for the 2D bin packing problem. In R. Khosla, R. Howlett, and L. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 3684 of *Lecture Notes in Computer Science*, pages 907–907. Springer Berlin / Heidelberg, 2005.

[43] E. Cuthill. *Several Strategies for Reducing the Bandwidth of Matrices*, pages 157–166. Plenum Press, New York, 1972.

[44] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *ACM National Conference*, pages 157–172, New York, 1969. Association for Computing Machinery.

[45] T. A. Davis. *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms. SIAM, Philadelphia, 2006.

[46] N. M. M. de Abreu. Old and new results on algebraic connectivity of graphs. *Linear Algebra and its Applications*, 423(1):53 – 73, 2007.

[47] N. Deo. *Graph theory with applications to engineering and computer science*. PHI Learning, Delhi, India, 2004.

[48] J. Díaz. The δ-operator. In L. Budach, editor, *Fundamentals of Computation Theory*, pages 105–111. Akademie-Verlag, 1979.

[49] J. Díaz. Graph layout problems. In I. Havel and V. Koubek, editors, *Mathematical Foundations of Computer Science 1992*, volume 629 of *Lecture Notes in Computer Science*, pages 14–23. Springer Berlin / Heidelberg, 1992.

[50] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *Computing Surveys*, 34:313–356, 2002.

[51] K. A. Dowsland, E. Soubeiga, and E. Burke. A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179(3):759 – 774, 2007.

[52] G. Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104:86–92, 1993.

[53] I. S. Duff, J. K. Reid, and J. A. Scott. The use of profile reduction algorithms with a frontal code. *International Journal for Numerical Methods in Engineering.*, 28:2555–2568, 1989.

[54] A. Esposito, F. Malucelli, and L. Tarricone. Bandwidth and profile reduction of sparse matrices: An experimental comparison of new heuristics. In *ALEX'98*, pages 19–26, Trento, Italy, 1998.

[55] G. C. Everstine. A comparison of three resequencing algorithms for the reduction of matrix profile and wavefront. *International Journal for Numerical Methods in Engineering.*, 14:837–853, 1979.

[56] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak. Math. J.*, 23:298–305, 1973.

[57] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference*. Carnegie Institue of Technology, 1961.

[58] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice-Hall, Englewood Cliffs, 1963.

[59] A. Fukunaga. Automated discovery of composite SAT variable-selection heuristics. In *Eighteenth national conference on Artificial intelligence*, pages 641–648, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[60] A. Fukunaga. Evolving local search heuristics for SAT using genetic programming. In K. Deb, editor, *Genetic and Evolutionary Computation (GECCO 2004)*, volume 3103 of *Lecture Notes in Computer Science*, pages 483–494. Springer Berlin Heidelberg, 2004.

[61] M. Garey, R. Graham, D. Johnson, and D. Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34(3):477–495, 1978.

[62] I. P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the eleventh national conference on Artificial intelligence*, AAAI'93, pages 28–33. AAAI Press, 1993.

[63] A. George and J. W. H. Liu. An implementation of a pseudoperipheral node finder. *ACM Transactions on Mathematical Software*, 5(3):284–295, 1979.

[64] J. A. George. *Computer implementation of the finite element method*. PhD thesis, Stanford, CA, USA, 1971.

[65] J. A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. 1981. Prentice-Hall.

[66] N. E. Gibbs. A hybrid profile reduction algorithm. *ACM Transactions on Mathematical Software.*, 2(4):378–387, 1976.

[67] N. E. Gibbs, W. G. Poole, and P. K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13(2):236–250, 1976.

[68] F. Glover. Tabu search–part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

[69] F. Glover. Tabu search–part II. *ORSA Journal on computing*, 2(1):4–32, 1990.

[70] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[71] D. E. Goldberg and R. Lingle, Jr. Alleles, Loci and the Traveling Salesman Problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

[72] M. Golumbic and I. Hartman. *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications*. Operations Research/Computer Science Interfaces Series. Springer, 2005.

[73] E. Gurari and I. Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the min-cut linear arrangement problem. *Journal of Algorithms*, 5:531–546, 1984.

[74] L. Han and G. Kendall. Guided operators for a hyper-heuristic genetic algorithm. In T. Gedeon and L. Fung, editors, *AI 2003: Advances in Artificial Intelligence*, volume 2903 of *Lecture Notes in Computer Science*, pages 807–820. Springer Berlin Heidelberg, 2003.

[75] L. Han and G. Kendall. Investigation of a tabu assisted hyper-heuristic genetic algorithm. In *Proceedings of Congress on Evolutionary Computation (CEC2003)*, pages 2230–2237, Canberra, 2003. IEEE Press.

[76] L. Han, G. Kendall, and P. Cowling. An adaptive length chromosome hyper-heuristic genetic algorithm for a trainer scheduling problem. In *Proceedings*

of the fourth Asia-Pacific Conference on Simulated Evolution And Learning, (SEAL'02), pages 267–271, Orchid Country Club, Singapore, November 2002.

[77] F. Harary. *Graph Theory*. Addison-Wesley, Reading, Mass, 1969.

[78] L. H. Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964.

[79] L. H. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1(3):385 – 393, 1966.

[80] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[81] Y. F. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM Journal on Scientific Computing.*, 23(4):1352–1375, 2001.

[82] M. Hyde. *A genetic programming hyper-heuristic approach to automated packing*. PhD thesis, University of Nottingham, July 20 2010.

[83] M. Hyde, E. K. Burke, and G. Kendall. Evolving human-competitive reusable 2D strip packing heuristics. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 2189–2192, New York, NY, USA, 2009. ACM.

[84] D. R. Karger. A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pages 11–17, New York, NY, USA, 1995. ACM.

[85] R. M. Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 278–285, New York, USA, 1993. ACM.

[86] R. E. Keller and R. Poli. Linear genetic programming of parsimonious metaheuristics. In D. Srinivasan and L. Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 4508–4515, Singapore, 25-28 September 2007. IEEE Press.

[87] D. G. Kendall. Incidence matrices, interval graphs, and seriation in archeology. *Pacific Journal of Mathematics*, 28:565–570, 1969.

[88] G. Kendall and N. Hussin. An investigation of a tabu-search-based hyperheuristic for examination timetabling. In G. Kendall, E. Burke, S. Petrovic, and M. Gendreau, editors, *Multidisciplinary Scheduling: Theory and Applications*, pages 309–328. Springer US, 2005.

[89] G. Kendall and N. Hussin. A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology. In E. Burke and M. Trick, editors, *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 270–293. Springer Berlin / Heidelberg, 2005.

[90] G. Kendall and M. Mohamad. Channel assignment optimisation using a hyper-heuristic. In *Proceedings of IEEE Conference on Cybernetics and Intelligent Systems*, volume 2, pages 791 –796, 2004.

[91] C. Kenyon. Best-fit bin-packing with random order. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 359–364, 1996.

[92] R. H. Kibria and Y. Li. Optimizing the initialization of dynamic decision heuristics in DPLL SAT solvers using genetic programming. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 331–340, Budapest, Hungary, 10 - 12 Apr. 2006. Springer.

[93] I. P. King. An automatic reordering scheme for simultaneous equations derived from network systems. *International Journal for Numerical Methods in Engineering*, 2(4):523–533, 1970.

[94] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[95] R. Klasing. The relationship between the gossip complexity in vertex-disjoint paths mode and the vertex bisection width. *Discrete Applied Mathematics*, 83(1-3):229 – 246, 1998.

[96] N. Knight Jr, R. Gillian, S. Mccleary, C. Lotts, E. Poole, A. Overman, and S. Macy. CSM testbed development and large-scale structural applications. Technical report, National Aeronautics and Space Administration (NASA), 1989.

[97] B. Koohestani and D. Corne. An improved fitness function and mutation operator for metaheuristic approaches to the bandwidth minimization problem. In *Proceedings of the 1st International Conference on Bio-Inspired Compu-*

tational (BICS), volume 1117, pages 21–28. AIP Conference Proceedings, 2009.

[98] B. Koohestani and R. Poli. A genetic programming approach to the matrix bandwidth-minimization problem. In R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6239 of *Lecture Notes in Computer Science*, pages 482–491. Springer Berlin / Heidelberg, 2010.

[99] B. Koohestani and R. Poli. A hyper-heuristic approach to evolving algorithms for bandwidth reduction based on genetic programming. In M. Bramer, M. Petridis, and L. Nolle, editors, *Research and Development in Intelligent Systems XXVIII*, pages 93–106. Springer London, 2011.

[100] B. Koohestani and R. Poli. A genetic programming approach for evolving highly-competitive general algorithms for envelope reduction in sparse matrices. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 287–296. Springer Berlin / Heidelberg, 2012.

[101] B. Koohestani and R. Poli. On the application of genetic programming to the envelope reduction problem. In *Computer Science and Electronic Engineering Conference (CEEC), 2012 4th*, pages 53–58. IEEE, 2012.

[102] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[103] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[104] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, G. Lanza, and J. Yu. *Genetic programming IV: Routine human-competitive machine intelligence*, volume 5. Springer Science+Business Media, 2007.

[105] R. Kumar, B. K. Bal, and P. I. Rockett. Multiobjective genetic programming approach to evolving heuristics for the bounded diameter minimum spanning tree problem: MOGP for BDMST. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 309–316, New York, NY, USA, 2009. ACM.

[106] R. Kumar, A. H. Joshi, K. K. Banka, and P. I. Rockett. Evolution of hyper-heuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In M. Keijzer, G. Antoniol, C. B. Congdon, K. Deb, B. Doerr, N. Hansen, J. H. Holmes, G. S. Hornby, D. Howard, J. Kennedy, S. Kumar, F. G. Lobo, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, J. Pollack, K. Sastry, K. Stanley, A. Stoica, E.-G. Talbi, and I. Wegener, editors, *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1227–1234, Atlanta, GA, USA, 12-16 July 2008. ACM.

[107] G. Kumfert and A. Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT Numerical Mathematics.*, 37:559–590, 1997.

[108] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, 1991.

[109] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Foundations of Computer Science, 21st Annual Symposium on*, pages 270–281. IEEE Computer Society Press, 1980.

[110] T. Lengauer. Upper and lower bounds on the complexity of the min-cut linear arrangement problem on trees. *SIAM Journal on Algebraic and Discrete Methods*, 3:99 – 113, 1982.

[111] J. G. Lewis. Implementation of the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms. *ACM Transactions on Mathematical Software.*, 8(2):180–189, 1982.

[112] A. Lim, J. Lin, B. Rodrigues, and F. Xiao. Ant colony optimization with hill climbing for the bandwidth minimization problem. *Applied Soft Computing*, 6(2):180–188, 2006.

[113] A. Lim, J. Lin, and F. Xiao. Particle swarm optimization and hill climbing for the bandwidth minimization problem. *Applied Intelligence*, 26(3):175–182, 2007.

[114] A. Lim, B. Rodrigues, and F. Xiao. Integrated genetic algorithm with hill climbing for bandwidth minimization problem. In E. Cantú-Paz and *et al.*, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, LNCS, vol. 2724, pages 1594–1595. Springer, Heidelberg, 2003.

[115] A. Lim, B. Rodrigues, and F. Xiao. A centroid-based approach to solve the bandwidth minimization problem. In *37th Hawaii international conference on system sciences (HICSS)*, page 30075a, Big Island, Hawaii, 2004.

[116] A. Lim, B. Rodrigues, and F. Xiao. Heuristics for matrix bandwidth reduction. *European Journal of Operational Research.*, 174(1):69 – 91, 2006.

[117] Y. Lin and J. Yuan. Profile minimization problem for matrices and graphs. *Acta Mathematicae Applicatae Sinica (English Series).*, 10:107–112, 1994.

[118] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[119] W. H. Liu and A. H. Sherman. Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis.*, 13(2):198–213, 1976.

[120] F. Makedon and I. H. Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243 – 265, 1989.

[121] R. Marti, M. Laguna, F. Glover, and V. Campos. Reducing the bandwidth of a sparse matrix with tabu search. *European Journal of Operational Research*, 135(2):450–459, 2001.

[122] B. McKay, S.-H. Chen, and X. H. Nguyen. Genetic programming: An emerging engineering tool. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 12(1):1–2, 2008. Guest Editorial.

[123] G. Mitchison and R. Durbin. Optimal numberings of an n×n array. *SIAM J. Algebraic Discrete Methods*, 7(4):571–582, 1986.

[124] N. Mladenovic, D. Urosevic, D. Pérez-Brito, and C. G. García-González. Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research*, 200(1):14–27, 2010.

[125] B. Mohar. The Laplacian spectrum of graphs. *Graph Theory, Combinatorics and Applications*, 2:871–898, 1991.

[126] K. Nakano. Linear layouts of generalized hypercubes. In J. van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science*, volume 790 of *Lecture Notes in Computer Science*, pages 364–375. Springer Berlin / Heidelberg, 1994.

[127] M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, Fall 2005.

[128] M. Oltean and D. Dumitrescu. Evolving TSP heuristics using multi expression programming. In M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, editors, *Computational Science - ICCS 2004: 4th International Conference, Part II*, volume 3037 of *Lecture Notes in Computer Science*, pages 670–673, Krakow, Poland, 6-9 June 2004. Springer-Verlag.

[129] I. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:511–623, 1996.

[130] C. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[131] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[132] G. H. Paulino, I. F. M. Menezes, M. Gattass, and S. Mukherjee. A new algorithm for finding a pseudoperipheral vertex or the endpoints of a pseudodiameter in a graph. *Communications in Numerical Methods in Engineering*, 10(11):913–926, 1994.

[133] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[134] T. Perkis. Stack-based genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[135] E. Piñana, I. Plana, V. Campos, and R. Martí. GRASP and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, 153(1):200–210, 2004.

[136] S. Pissanetskey. *Sparse Matrix Technology*. Academic Press, London, 1984.

[137] E. Polak. *Optimization: Algorithms and Consistent Approximation*. Applied Mathematical Sciences Series. Springer-Verlag GmbH, 1997.

[138] R. Poli. Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. In G. D. Smith, N. C. Steele, and R. F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference, ICANNGA97*, University of East Anglia, Norwich, UK, 1997. Springer-Verlag. published in 1998.

[139] R. Poli. Evolution of graph-like programs with parallel distributed genetic programming. In T. Back, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 346–353, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann.

[140] R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 204–217. Springer-Verlag, 14-16 Apr. 2003.

[141] R. Poli. Covariant tarpeian method for bloat control in genetic programming. In R. Riolo, T. McConaghy, and E. Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, chapter 5, pages 71–90. Springer, Ann Arbor, USA, 20-22 May 2010.

[142] R. Poli, W. B. Langdon, and O. Holland. Extending particle swarm optimisation via genetic programming. In M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 291–300, Lausanne, Switzerland, 30 Mar. - 1 Apr. 2005. Springer.

[143] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. 2008. Published via `http://lulu.com`, with contributions by J. R. Koza.

[144] R. Poli, J. Woodward, and E. K. Burke. A histogram-matching approach to the evolution of bin-packing strategies. In D. Srinivasan and L. Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 3500–3507, Singapore, 25-28 Sept. 2007. IEEE Computational Intelligence Society, IEEE Press.

[145] S. S. Rao. *Engineering Optimization: Theory and Practice, 3rd Edition*. John Wiley & Sons, 1996.

[146] R. Ravi, A. Agrawal, and P. Klein. Ordering problems approximated: single-processor scheduling and interval graph completion. In J. Albert, B. Monien, and M. Artalejo, editors, *Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 751–762. Springer Berlin Heidelberg, 1991.

[147] J. K. Reid and J. A. Scott. Ordering symmetric sparse matrices for small profile and wavefront. *International Journal for Numerical Methods in Engineering.*, 45:1737–1755, 1999.

[148] E. Rodriguez-Tello, H. Jin-Kao, and J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research*, 185(3):1319–1335, 2008.

[149] P. Ross. Hyper-heuristics. In E. K. Burke and G. Kendall, editors, *Search Methodologies*, pages 529–556. Springer US, 2005.

[150] C. Ryan. Book review: Genetic programming 3: Darwinian invention and problem solving. *Genetic Programming and Evolvable Machines*, 1(4), Oct. 2000.

[151] B. Selman and H. Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. In *Proceedings of the 13th international joint conference on Artifical intelligence - Volume 1*, IJCAI'93, pages 290–295, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[152] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, AAAI '94, pages 337–343, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[153] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the tenth national conference on Artificial intelligence*, AAAI'92, pages 440–446. AAAI Press, 1992.

[154] P. Siarry and Z. Michalewicz. *Advances in metaheuristics for hard optimization*. Springer, 2007.

[155] J. P. i. Silvestre. *Layout Problems*. PhD thesis, Universitat Politècnica de Catalunya, 2001.

[156] S. W. Sloan. A FORTRAN program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering.*, 28(11):2651–2679, 1989.

[157] S. W. Sloan and W. S. Ng. A direct comparison of three algorithms for reducing profile and wavefront. *Computers & Structures.*, 33(2):411–419, 1989.

[158] J. Tavares, P. Machado, A. Cardoso, F. B. Pereira, and E. Costa. On the evolution of evolutionary algorithms. In M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 389–398, Coimbra, Portugal, 5-7 Apr. 2004. Springer-Verlag.

[159] Q. Wang and X.-W. Shi. An improved algorithm for matrix bandwidth and profile reduction in finite element analysis. *Progress In Electromagnetics Research Letters*, 9:29–38, 2009.