

A Hyper-Heuristic Approach to Evolving Algorithms for Bandwidth Reduction Based on Genetic Programming

Behrooz Koohestani and Riccardo Poli

Abstract The bandwidth reduction problem is a well-known NP-complete graph-layout problem that consists of labeling the vertices of a graph with integer labels in such a way as to minimize 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 so that its non-zero entries are maximally close to the main diagonal — a problem which presents itself in a large number of domains in science and engineering. A considerable number of methods have been developed to reduce the bandwidth, among which graph-theoretic approaches are typically faster and more effective. In this paper, a hyper-heuristic approach based on genetic programming is presented for evolving graph-theoretic bandwidth reduction algorithms. The algorithms generated from our hyper-heuristic are extremely effective. We test the best of such evolved algorithms on a large set of standard benchmarks from the Harwell-Boeing sparse matrix collection against two state-of-the-art algorithms from the literature. Our algorithm outperforms both algorithms by a significant margin, clearly indicating the promise of the approach.

1 Introduction

Graph layout problems are a special class of combinatorial optimization problems aiming at discovering a linear layout of an input graph such that a certain objective function is optimized [11]. A linear layout is a labeling of the nodes of a graph with unique integers from the set $\{1, \dots, n\}$ where n is the number of nodes in the graph.

Behrooz Koohestani

School of Computer Science and Electronic Engineering, University of Essex, Colchester, CO4 3SQ, UK. e-mail: bkooshe@essex.ac.uk

Riccardo Poli

School of Computer Science and Electronic Engineering, University of Essex, Colchester, CO4 3SQ, UK. e-mail: rpoli@essex.ac.uk

The bandwidth minimisation problem (BMP) is a very well-known graph layout problem, which has connections with a wide range of other problems, such as finite element analysis of mechanical systems, 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 [6]. The BMP can be stated in the context of both graphs and matrices. The bandwidth problem for graphs consists of finding a special labeling of vertices of a graph which minimizes the maximum absolute difference between the (integer) labels of adjacent vertices. In terms of matrices, it consists of finding a permutation of rows and columns of a given matrix which ensures that the non-zero elements are located in a band as close as possible along the main diagonal. In fact, if the non-zero entries of a symmetric matrix and the permutations of rows and columns are identified with the edges of a graph and the flips of the vertex labels, respectively, then the bandwidth of the graph is equal to the bandwidth of the matrix [34].

One of the most common applications of bandwidth minimisation algorithms arises from the need to efficiently solve large systems of equations [35]. In such a scenario, more efficient solutions are obtained if the rows and columns of the matrix representing the set of equations can be permuted in such a way that the bandwidth of the matrix is minimized [35].

Unfortunately, the BMP has been proved to be NP-complete [33]. Hence, 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 the BMP 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 [15].

Due to the existence of strong links between the BMP and a wide range of other problems in scientific and engineering fields, a variety of methods have been proposed for reducing the bandwidth.

The first direct method for the BMP was proposed by Harary [21]. Cuthill and McKee [10] introduced the first heuristic approach to the problem. Their method is still one of the most important and widely used methods to (approximately) solve the 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. This partition is known as *level structure* for the given node. In Cuthill-McKee algorithm, the root node for the level structure is normally chosen from the nodes of minimum degree in the graph. The permutation selected to reduce the bandwidth of the matrix is then simply obtained by visiting the nodes in the level structure in increasing-distance order.

George [17] in the study of envelope reduction algorithms observed that renumbering the Cuthill-McKee ordering in a reverse way often yielded a result superior to the original ordering.¹ This algorithm is known as the Reverse Cuthill-McKee (RCM) algorithm. Experimental evidence confirming the superior performance of

¹ The envelope minimisation problem is a problem strongly related to the BMP which requires the reorganisation of the nodes in a graph or the rows and columns in a matrix, but with a slightly different objective (we will formally define the bandwidth and envelope in Sect. 3).

the RCM over Cuthill-McKee for matrices arising from the finite element method has been reported in [9, 29].

A few years later, Gibbs *et al.* [19] proposed an algorithm, known as GPS (which stands for “Gibbs, Poole and Stockmeyer”), that makes 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 apart) to be used as an appropriate starting node. The algorithm is substantially faster than the Cuthill-McKee algorithm, and it can occasionally outperform it. However, GPS algorithm is more complex to implement.

Barnard *et al.* [3] 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 of a graph. In particular, the method permutes the rows and columns of a matrix based on the eigenvector associated with the first non-zero eigenvalue of the Laplacian matrix. While the envelope is only indirectly related to the bandwidth of a matrix, this algorithm is very effective at reducing it. Further information on these and other classic methods for the BMP can be found in [7, 20].

Recently, meta-heuristic approaches have been tested to see if they can be viable alternatives to solve the BMP. For example, Tabu search was employed by Marti *et al.* [30], while Lim *et al.* [27] used a hybrid between genetic algorithm and hill-climbing to solve this problem. Lim *et al.* also introduced two other hybrid algorithms to solve the BMP: one combining ant colony optimization with hill-climbing [25] and one combining particle swarm optimization with hill-climbing [26]. More recently, simulated annealing has also been used to attack the problem [41].

In this paper, a genetic programming hyper-heuristic is presented for evolving bandwidth reduction algorithms. A hyper-heuristic is a higher-level search algorithm specialised in the production of search algorithms, heuristics, optimisers or problem solvers [5]. The algorithms generated from our hyper-heuristic are heuristic in nature and based on the level structures. In order to evaluate the performance of the generated heuristics, we test them against a high-performance version of the RCM algorithm contained in the MATLAB library and the spectral algorithm (also implemented in MATLAB) described earlier. To the best of our knowledge, no prior attempt to employ a hyper-heuristic in order to evolve BMP heuristics has been reported in the literature.

The paper is organised as follows. Sect. 2 provides some background information in relation to hyper-heuristics and genetic programming. Sect. 3 briefly describes graph and matrix definitions of the BMP. In Sect. 4, the proposed hyper-heuristic for the solution of the BMP is presented in detail. In Sect. 5, experimental results and related statistical analyses are reported. Finally, our conclusions are presented in Sect. 6.

2 Genetic Programming and Hyper-heuristics

Genetic Programming (GP) [24, 36] is an evolutionary algorithm inspired by biological evolution. GP has the potential to solve problems automatically without the need for the user to know or specify the form or structure of the solution in advance. It is actually a specialization of Genetic Algorithms (GAs) in which the evolving individuals are computer programs rather than a set of fixed length strings from a limited alphabet of symbols. GP has a random nature similar to GA and is able to stochastically transform populations of programs into new and, hopefully better populations of programs. In this study, GP is utilised as a hyper-heuristic.

The term *hyper-heuristic* was first introduced by Cowling *et al.* [8]. According to their definition, a hyper-heuristic manages the choice of which lower-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. However, a *hyper-heuristic* could more simply and generally be defined as “heuristics to choose other heuristics” [5]. Here, a *heuristic* is considered as a rule-of-thumb or “educated guess” that reduces the search required to find a solution.

The difference between meta-heuristics and hyper-heuristics is that the former operate directly on the problem search space with the goal of finding optimal or near-optimal solutions. The latter, instead, operate on the heuristics search space (which consists of the heuristics used to solve the target problem). The goal then is finding or generating high-quality heuristics for a problem, for a certain class of instances of a problem, or even for a particular instance.

GP has been very successfully used as a hyperheuristic. For example, GP has evolved competitive SAT solvers [1, 2, 14, 23], state-of-the-art or better than state-of-the-art bin packing algorithms [4, 40], particle swarm optimisers [39], evolutionary algorithms [31], and TSP solvers [22, 32].

3 Bandwidth, Envelope and Level Structures

Let $G = (V, E)$ be a finite undirected graph, such that V is the set of vertices, E is the set of edges and $f : V \rightarrow \{1, \dots, n\}$ is a labeling of its nodes where $n = |V|$, then the *bandwidth* of G under f can be defined as:

$$B_f(G) = \max_{(u,v) \in E} |f(u) - f(v)| \quad , \quad (1)$$

i.e., as the maximum absolute difference between the labels of the adjacent nodes (i.e., nodes connected by an edge). The BMP consists of finding a labeling f which minimises $B_f(G)$ while the easier *bandwidth reduction problem* requires finding any labeling which reduces $B_f(G)$. Since there are $n!$ possible labellings for a graph with n vertices, it stands to reason that the BMP is, in general, a very difficult combinatorial optimisation problem.

The BMP can also be stated in the context of matrices. If $A = [a_{ij}]_{n \times n}$ is a sparse matrix, its *bandwidth* is defined as

$$B(A) = \max_{(i,j):a_{ij} \neq 0} |i - j| . \quad (2)$$

The *matrix bandwidth minimisation problem* consists of finding a permutation of rows and columns which brings all non-zero elements of A into the smallest possible band around the diagonal. More formally, if σ is a permutation of $(1, 2, \dots, n)$, and A_σ is the matrix obtained by permuting the rows and columns of A according to σ (i.e., $A_\sigma = [a_{\sigma_i \sigma_j}]$), then the problem can be formulated as

$$\min_{\sigma} B(A_\sigma) . \quad (3)$$

An important concept related to the *bandwidth* is the notion of *envelope*. Given a matrix A , its *envelope* is:

$$P(A) = \sum_{i=1}^n \max_{j: j < i, a_{ij} \neq 0} (i - j) . \quad (4)$$

Naturally, the *envelope* is also influenced by the permutations of A .

One of the most important concepts in many graph-theoretic bandwidth and envelope reduction algorithms is that of *level structure* [19]. A level structure, $L(G)$, of a graph G is a partition of the set $V(G)$ into levels L_1, L_2, \dots, 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 either 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.

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 [19].

In a nodal numbering scheme based on the level structures such as [10, 17, 19], 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* [18] in a graph. 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 by the RCM algorithm.

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

Algorithm 1 GHH for BMP

```

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 speci-
  fied probabilities.
26: until the maximum number of generations is reached.
27: return the best program tree appearing in the last generation.

```

4 Proposed Hyper-heuristic for BMP

We propose to use GP as a hyper-heuristic for the BMP. In other words, in our system, which we will call GHH (for “Genetic Hyper-Heuristic”), GP is given a training set of matrices as input and produces a novel solver for BMPs as its output.

Naturally, the task is of a colossal difficulty. So, following the strategy adopted in [40] and to some extent also in [1], to make this feasible we provide GHH with the “skeleton” of a good solver (an enhanced version of the RCM graph-based BMP solver, mentioned in Sect. 1) and we ask GP to evolve the “brain” of that solver, that is the strategy by which it prioritises nodes in a level structure.

A description of the operations of the system is given in Algorithm 1. Below, the elements of the system are described in detail.

4.1 GP Setup

We used a tree-based GP system with some additional decoding steps required for the BMP. The system is implemented in C#. Individuals in our GP system are tree-like expressions which, for efficiency reasons, are internally stored as linear arrays using a flattened representation for trees. The primitive set used is shown in Table 1.

The initial population was generated randomly using a modified version of the ramped half-and-half method (which produces a mixture of full trees of different depths and randomly-grown trees) [24, 36]. In our system, during the process of tree initialisation, we artificially ensure that each program tree contains at least one instance of the primitive *SDV*. We will explain the motivation for this in Sect. 4.3.

In nodal numbering algorithms based on the level structures, the vertices in each level are numbered in order of increasing degree. We should note, however, that our primitive set 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 RCM algorithm uses for prioritising nodes within a level. We believe evolution found particularly difficult to use such a primitive due to the problem of ties.

In a level, there are often nodes with the same degree. As mentioned in Sect. 3, the normal strategy for dealing with this issue is to break ties arbitrarily. 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 the *SDV*, which is effectively equivalent to the product of a vertex degree and the mean degree of the vertex's children.

While the BMP can be stated both in terms of graphs (see Eq. (1)) and in terms of matrices (see Eq. (2)), in order to develop fast algorithms for sparse matrices, one really needs to calculate such quantities from graphs associated with the matrices. Therefore, we utilised the bandwidth as defined in Eq. (1) in our GP system as the fitness contribution of a problem. Naturally, the fitness of a program tree is then the total of the bandwidths of the solutions that it creates when run on each problem instance in the training set. Fitness, rather obviously, needs to be minimised by the GP system.

The parameters of the runs are presented in Table 2. We employed tournament selection to choose individual program(s) from the population based on fitness to participate in genetic operations. New individual programs were created by applying the genetic operations of reproduction, sub-tree crossover and point mutation with specified probabilities. We also used elitism to ensure that the best individual in one generation was transferred unaltered to the next. In addition, in order to control excessive code growth or bloat, the *Tarpeian method* (which artificially weeds out a

Table 1. Primitive set used in our GP system

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 $[-1.0, +1.0]$

Table 2. Parameters used in our experiments

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%

proportion of above-average-size trees) [37, 38] was utilised in the GP system. The termination criterion used was based on the predetermined maximum number of generations to be run. Finally, the best program tree appearing in the last generation was designated as the final result of a run.

4.2 Training Set

We used a training set of 25 benchmark instances from the Harwell-Boeing sparse matrix collection (available at <http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/dwt/dwt.html>). 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), BCSTRUC3 (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 .

This training set was used only to evolve the heuristics. Performance of the evolved heuristics was then evaluated using a completely separate test set of 30 matrices taken from Everstine's sparse matrix collection (DWT) and BCSPWR, both included in the Harwell-Boeing database.

4.3 Numbering Vertices Located in Each Level

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

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. 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 (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) (Step 20).

In Sect. 4.1, we mentioned that each program tree in our GP system contains at least one SDV primitive. Now, by considering our ordering method explained above, the motivation behind this type of tree initialisation can be easily understood. Trees without an SDV generate a constant output (the problem size, N , is constant for each problem) and, thus, they represent a fixed permutation $\sigma = (1, 2, \dots, n)$. Such a permutation is useless, since numbering the vertices located in each level in the order given by σ produces no change in the labels of vertices, and as a result, the bandwidth remains unchanged.

5 Experimental Results

We performed ten independent runs of our GP system with the training set specified above, recording the corresponding best-of-run individual in each. 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:

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

which is also shown graphically in Figure 1. What is interesting about this re-

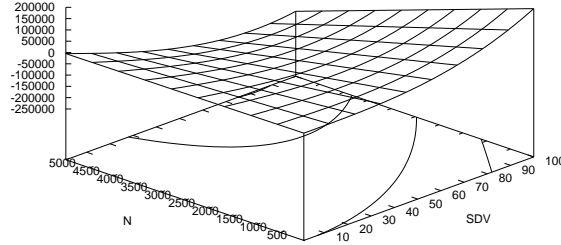


Fig. 1. Plot of the best heuristic evolved for BMP by GHH.

sult is that the function evolved is *not* monotonically increasing in SDV (for any given N). Indeed, the minimum of the function as N varies is given by $SDV = \sqrt{1.5863264966 \times N + 4.16677290311} / 1.07695756903 - 0.543846560629$. 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 is done in RCM. However, as N increases, the heuristic function evolved by GHH 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 Sect. 3, and carried out a substantial number of experiments. In the experiments conducted in this study, we employed Everstine's sparse matrix collection (DWT) [13] as test problems. This collection consists of 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 actually a subset of the well-known 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 reduction algorithms. DWT set is closely related to CANNES and LSHAPE sets used in our training set in terms of its discipline and the class of problems.

We also picked the six largest instances from BCSPWR set (power network patterns), which is totally in a different class compared to the training set used. We did this to observe how well the generated heuristic generalises in unseen situations.

In order to assess the performance of the heuristic generated, we compared it against two well-known and high-performance algorithms, i.e., the RCM contained

in the MATLAB library (RCMM), and spectral. RCMM is a highly enhanced version of the RCM algorithm, which is based closely on the SPARSPAK implementation described by George and Liu [16]. This algorithm is still one of the best and most widely used methods for bandwidth reduction. Indeed, the results reported in [12] and [28] indicate that RCMM is also superior to the well-known GPS [19] algorithm in terms of the quality of the solutions obtained.

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 in each of the benchmark problems, we also measured the time required to solve the problem on this computer.

Table 3 shows a performance comparison of the algorithms under test. The results of the tests reveal that the heuristic generated by our GHH 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/Draw” rows). A Wilcoxon signed-rank test, which is a reliable and widely used nonparametric test for paired data, revealed that the performance differences between the heuristic generated by GHH and these algorithms are highly statistically significant ($p = 0.000$).

6 Conclusions

In this paper, GHH — a hyper-heuristic approach based on genetic programming — has been proposed for evolving bandwidth reduction algorithms. To help GP in this difficult task, we constrained its search by providing it with the scaffolding of a good graph-based solver and asking it to only evolve the key element of the algorithm: its brain, i.e., the sorting strategy for nodes in a level structure.

The best solver produced by GHH is a very interesting and unconventional brain, which essentially goes against the accepted practice of ordering nodes by degree, particularly for large BMP instances. We compared this solver against two high-performance algorithms, the RCM contained in the MATLAB library and spectral, on a large set of standard benchmarks from the Harwell-Boeing sparse matrix collection. The best heuristic evolved performed extremely well both on benchmark instances from the same class as the training set and also (and perhaps even better) on large problem instances from a totally different class, confirming the efficacy of our approach.

In future work, we will investigate the possibility of further extending the primitive set used by GHH, we will test evolved solvers over even more diverse and large benchmark sets and we will also attack the envelope minimisation problem.

Table 3. Comparison of our GHH against the RCMM and Spectral algorithms. Numbers in bold face are best results.

Instance	Dimension	Bandwidth			Run time		
		RCMM	Spectral	GHH	RCMM	Spectral	GHH
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
<i>Mean</i>		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
<i>Mean</i>		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

References

1. Bader-El-Den, M., Poli, R.: Generating SAT local-search heuristics using a GP hyper-heuristic framework. In: Monmarché, N., Talbi, E.G., Collet, P., Schoenauer, M., Lutton, E. (eds.) Evolution Artificielle, 8th International Conference. Lecture Notes in Computer Science, vol. 4926, pp. 37–49. Springer, Tours, France (29-31 Oct 2007)
2. Bader-El-Den, M.B., Poli, R.: A GP-based hyper-heuristic framework for evolving 3-SAT heuristics. In: Thierens, D., Beyer, H.G., Bongard, J., Branke, J., Clark, J.A., Cliff, D., Congdon, C.B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K.O., Stutzle, T., Watson, R.A., Wegener, I. (eds.)

- GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. vol. 2, pp. 1749–1749. ACM Press, London (7-11 Jul 2007)
3. Barnard, S.T., Pothen, A., Simon, H.D.: A spectral algorithm for envelope reduction of sparse matrices. In: *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing*. pp. 493–502. ACM, New York, NY, USA (1993)
 4. Burke, E.K., Hyde, M.R., Kendall, G.: Evolving bin packing heuristics with genetic programming. In: Runarsson, T.P., Beyer, H.G., Burke, E., Merelo-Guervos, J.J., Whitley, L.D., Yao, X. (eds.) *Parallel Problem Solving from Nature - PPSN IX*. LNCS, vol. 4193, pp. 860–869. Springer-Verlag, Reykjavik, Iceland (9-13 Sep 2006)
 5. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In: *Handbook of Metaheuristics*, chap. 16, pp. 457–474. International Series in Operations Research & Management Science (2003)
 6. Chinn, P.Z., Chvátalová, J., Dewdney, A.K., Gibbs, N.E.: The bandwidth problem for graphs and matrices — a survey. *Journal of Graph Theory* 6(3), 223–254 (1982)
 7. Corso, G.D., Manzini, G.: Finding exact solutions to the bandwidth minimization problem. *Computing* 62(3), 189–203 (1999)
 8. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (eds.) *Practice and Theory of Automated Timetabling III*, Lecture Notes in Computer Science, vol. 2079, pp. 176–190. Springer Berlin / Heidelberg (2001)
 9. Cuthill, E.: Several Strategies for Reducing the Bandwidth of Matrices, pp. 157–166. Plenum Press, New York (1972)
 10. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: *ACM National Conference*. pp. 157–172. Association for Computing Machinery, New York (1969)
 11. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. *Computing Surveys* 34, 313–356 (2002)
 12. Esposito, A., Malucelli, F., Tarricone, L.: Bandwidth and profile reduction of sparse matrices: An experimental comparison of new heuristics. In: *ALEX'98*. pp. 19–26. Trento, Italy (1998)
 13. Everstine, G. C.: 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)
 14. Fukunaga, A.: Automated discovery of composite SAT variable selection heuristics. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. pp. 641–648 (2002)
 15. Garey, M., Graham, R., Johnson, D., Knuth, D.: Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics* 34(3), 477–495 (1978)
 16. George, J.A., Liu, J.W.H.: *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall (1981)
 17. George, J.A.: *Computer implementation of the finite element method*. Ph.D. thesis, Stanford, CA, USA (1971)
 18. George, A., Liu, J. W. H.: An implementation of a pseudoperipheral node finder. *ACM Transactions on Mathematical Software* 5(3), 284–295 (1979)
 19. Gibbs, N.E., Poole, W.G., Stockmeyer, P.K.: An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis* 13(2), 236–250 (1976)
 20. Gurari, E., Sudborough, I.: Improved dynamic programming algorithms for bandwidth minimization and the min-cut linear arrangement problem. *Journal of Algorithms* 5, 531–546 (1984)
 21. Harary, F.: *Graph Theory*. Addison-Wesley, Reading, Mass (1969)
 22. Keller, R.E., Poli, R.: Linear genetic programming of parsimonious metaheuristics. In: Srinivasan, D., Wang, L. (eds.) *2007 IEEE Congress on Evolutionary Computation*. pp. 4508–4515. IEEE Computational Intelligence Society, IEEE Press, Singapore (25–28 Sep 2007)
 23. Kibria, R.H., Li, Y.: Optimizing the initialization of dynamic decision heuristics in DPLL SAT solvers using genetic programming. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) *Proceedings of the 9th European Conference on Genetic Programming*. Lecture Notes in Computer Science, vol. 3905, pp. 331–340. Springer, Budapest, Hungary (10 - 12 Apr 2006)

24. Koza, J.R.G.P.: *On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA (1992)
25. Lim, A., Lin, J., Rodrigues, B., Xiao, F.: Ant colony optimization with hill climbing for the bandwidth minimization problem. *Applied Soft Computing* 6(2), 180–188 (2006)
26. Lim, A., Lin, J., Xiao, F.: Particle swarm optimization and hill climbing for the bandwidth minimization problem. *Applied Intelligence* 26(3), 175–182 (2007)
27. Lim, A., Rodrigues, B., Xiao, F.: Integrated genetic algorithm with hill climbing for bandwidth minimization problem. In: Cantú-Paz, E., *et al.* (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. pp. 1594–1595. LNCS, vol. 2724, Springer, Heidelberg (2003)
28. Lim, A., Rodrigues, B., Xiao, F.: A centroid-based approach to solve the bandwidth minimization problem. In: *37th Hawaii international conference on system sciences (HICSS)*. p. 30075a. Big Island, Hawaii (2004)
29. Liu, W.H., Sherman, A.H.: 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)
30. Marti, R., Laguna, M., Glover, F., Campos, V.: Reducing the bandwidth of a sparse matrix with tabu search. *European Journal of Operational Research* 135(2), 450–459 (2001)
31. Oltean, M.: Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation* 13(3), 387–410 (Fall 2005)
32. Oltean, M., Dumitrescu, D.: Evolving TSP heuristics using multi expression programming. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) *Computational Science - ICCS 2004: 4th International Conference, Part II. Lecture Notes in Computer Science*, vol. 3037, pp. 670–673. Springer-Verlag, Krakow, Poland (6-9 Jun 2004)
33. Papadimitriou, C.H.: The NP-completeness of the bandwidth minimization problem. *Computing* 16(3), 263–270 (1976)
34. Papadimitriou, C. H., Steiglitz, K.: *Combinatorial Optimization : Algorithms and Complexity*. Prentice-Hall (1982)
35. Pissanetsky, S.: *Sparse Matrix Technology*. Academic Press, London (1984)
36. Poli, R., Langdon, W.B., McPhee, N.F.: *A Field Guide to Genetic Programming* (2008), published via <http://lulu.com>, with contributions by J. R. Koza
37. Poli, R.: A simple but theoretically-motivated method to control bloat in genetic programming. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., Costa, E. (eds.) *Genetic Programming, Proceedings of EuroGP'2003*. LNCS, vol. 2610, pp. 204–217. Springer-Verlag (14-16 Apr 2003)
38. Poli, R.: Covariant tarpeian method for bloat control in genetic programming. In: Riolo, R., McConaghy, T., Vladislavleva, E. (eds.) *Genetic Programming Theory and Practice VIII, Genetic and Evolutionary Computation*, vol. 8, chap. 5, pp. 71–90. Springer, Ann Arbor, USA (20-22 May 2010)
39. Poli, R., Langdon, W.B., Holland, O.: Extending particle swarm optimisation via genetic programming. In: Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J.I., Tomassini, M. (eds.) *Proceedings of the 8th European Conference on Genetic Programming. Lecture Notes in Computer Science*, vol. 3447, pp. 291–300. Springer, Lausanne, Switzerland (30 Mar - 1 Apr 2005)
40. Poli, R., Woodward, J., Burke, E.K.: A histogram-matching approach to the evolution of bin-packing strategies. In: Srinivasan, D., Wang, L. (eds.) *2007 IEEE Congress on Evolutionary Computation*. pp. 3500–3507. IEEE Computational Intelligence Society, IEEE Press, Singapore (25-28 Sep 2007)
41. Rodriguez-Tello, E., Jin-Kao, H., Torres-Jimenez, J.: An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research* 185(3), 1319–1335 (2008)