A Genetic Programming Approach for Evolving Highly-Competitive General Algorithms for Envelope Reduction in Sparse Matrices

Behrooz Koohestani and Riccardo Poli

School of Computer Science and Electronic Engineering,
University of Essex, CO4 3SQ, UK
{bkoohe,rpoli}@essex.ac.uk

Abstract. Sparse matrices emerge in a number of problems in science and engineering. Typically the efficiency of solvers for such problems depends crucially on the distances between the first non-zero element in each row and the main diagonal of the problem's matrix — a property assessed by a quantity called the *size of the envelope* of the matrix. This depends on the ordering of the variables (i.e., the order of the rows and columns in the matrix). So, some permutations of the variables may reduce the envelope size which in turn makes a problem easier to solve. However, finding the permutation that minimises the envelope size is an NP-complete problem. In this paper, we introduce a hyper-heuristic approach based on genetic programming for evolving envelope reduction algorithms. We evaluate the best of such evolved algorithms on a large set of standard benchmarks against two state-of-the-art algorithms from the literature and the best algorithm produced by a modified version of a previous hyper-heuristic introduced for a related problem. The new algorithm outperforms these methods by a wide margin, and it is also extremely efficient.

Keywords: Hyper-Heuristic, Genetic Programming, Envelope Reduction Problem, Graph Labelling, Sparse Matrices.

1 Background

A substantial number of problems in science and engineering require the solution of large systems of linear equations. The effectiveness of methods designed to handle such systems depends critically on finding an ordering for the variables for which the distances between the first non-zero element in each row and the main diagonal of the problem's matrix is small [15]. This property is typically assessed by a quantity called the *size of the envelope* of the matrix. Let us start by providing a formal definition of it.

Let A be an $N \times N$ symmetric matrix with entries a_{ij}. The *row bandwidth* of the i^{th} row of A is defined as follows: $b_i(A) = i - \min\{j : a_{ij} \neq 0\}$. In other words, the row bandwidth is the distance (in columns) from the first non-zero entry in a row to the diagonal [7]. The *envelope* of matrix A is directly related

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part II, LNCS 7492, pp. 287–296, 2012.

to its row bandwidths, and can be thought of as a function $e(i, A) = b_i(A) + 1$, which returns the number of elements between the first non-zero entry in a row and the main diagonal (inclusive). Then the *size of the envelope* of the matrix is defined as [20]:

$$|Env(A)| = \sum_{i=1}^{N} e(i, A)\,.$$

Finding a permutation of rows and columns of A which minimises the envelope size $|Env(A)|$ — a problem known as the *Envelope Reduction Problem* (ERP) — is the focus of this paper. Since there are $N!$ possible permutations for an $N \times N$ matrix, the ERP is considered, in general, a very difficult combinatorial optimisation problem. Indeed, ERP was shown to be NP-complete [3].

1.1 Envelope Reduction Algorithms

A variety of methods have been proposed in order to address the ERP. One of the earliest heuristic approaches for reducing the bandwidth and envelope size of sparse matrices was introduced by Cuthill and McKee [6]. Their algorithm (CM) is still one of the most widely used algorithms to (approximately) solve these problems. 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. In CM, the root node for the level structure is normally chosen from the nodes of minimum degree in the graph. George [8] observed that renumbering the CM ordering in a reverse way (RCM) often yielded a result superior to the original ordering. The GPS algorithm, introduced by Gibbs, Poole and Stockmeyer [10], also uses level structures, and it is comparable with RCM in terms of solution quality, while being several times faster. The GK (Gibbs-King) algorithm [9], which is a variation of GPS, provides considerably better reduction of the envelope in comparison with the original GPS, but it is often much slower in execution. The Sloan algorithm [20] offered a significant improvement over the methods mentioned earlier by introducing a new step in which the ordering obtained from a variant of the GPS algorithm was locally refined. Adopting a very different approach Barnard *et al.* [2] 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 sparse matrix. Recently, also a new variation of the GPS algorithm has been presented [21].

1.2 Hyper-Heuristics

The term *hyper-heuristic* was first introduced by Cowling *et al.* [5]. 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. Here, a *heuristic* is a rule-of-thumb or "educated guess" that reduces

the search required to find a solution. More generally a hyper-heuristic could be defined as "heuristics to choose other heuristics" [4]. Here, we embrace a slightly different definition and see a *hyper-heuristic as a search algorithm that explores the space of problem solvers*. Genetic Programming (GP) [13,16] has been very successfully used as a hyperheuristic. For example, GP has evolved competitive SAT solvers [1], state-of-the-art or better than state-of-the-art bin packing algorithms [19], particle swarm optimisers [18], evolutionary algorithms [14], and TSP solvers [11].

In this paper, a hyper-heuristic approach based on GP is introduced for evolving graph-theoretic envelope reduction algorithms. Our approach is to adopt the basic ideas of some of the best algorithms for ERP, in particular their use of level structures, but to evolve the strategy the algorithm uses to construct permutations.[1] The paper is organised as follows: in Sec. 2, we describe the proposed hyper-heuristic for the solution of the ERP in detail; in Sec. 3, we report the results of our experiments; and finally, our conclusions are given in Sec. 4.

2 Proposed Hyper-Heuristic

In our method for addressing the ERP, which we call *Genetic Hyper-Heuristic* or GHH for brevity, GP is given a training set of matrices as input, and it produces a novel solver for ERPs as its output. To cope with such a complex task, following the strategy adopted in previous work [19,12], we provide GHH with the "skeleton" of a generic level-structure-based ERP solver and we ask GP to evolve the "brain" of that solver, that is the decision-making element of the system which prioritises nodes for insertion into a permutation.

A description of GHH is given in Algorithm 1. For efficiency, GHH 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 (Step 4). For the same reason, we operate on the graph representation of sparse matrices instead of directly acting on the matrices. Note also that, unlike previous solvers (including our method [12]) which prioritise nodes at each level in a level structure independently, GHH is capable of exploring and sorting vertices located beyond a specific level. More details on Algorithm 1 are provided below.

2.1 Our GP System

We used a tree-based GP system with some additional decoding steps required for the ERP. The initial population was generated randomly using a modified version of the ramped half-and-half method [13,16] using the functions and terminals shown in Table 1 (more on these below). As shown in Algorithm 1, the

[1] To the best of our knowledge, no prior attempt to use a hyper-heuristic to evolve ERP solvers has been reported in the literature. However, we conducted previous research with a hyper-heuristic for the related bandwidth minimisation problem where the objective is to minimise $\max_i b_i(A)$ [12]. We will compare our new envelope reduction approach against an envelope-reduction version of such hyper-heuristic in Sec. 3.

Algorithm 1. GHH for ERP

```
1: Randomly generate an initial population of programs from the available primitives.
2: repeat
3:    Initialise the fitness of each program p ∈ population to 0.
4:    for each instance G_i ∈ training set of ERPs do
5:       Select a starting vertex s and construct a level structure rooted at s.
6:       for each program p ∈ population do
7:          l ← empty list
8:          for each vertex v ∈ 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' ∈ l do
12:               Execute p.
13:            end for
14:            Create permutation σ represented by p.
15:            Sort vertices in l in order given by σ.
16:            s ← 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 1. The functions and terminals used in our GP system

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 $[-1.0, +1.0]$

fitness of a program tree (to be minimised) is the sum of the envelopes of the solutions that it creates when run on each problem instance in the training set.

The parameters of our GP runs are given in Table 2.[2] Tournament selection was used. New individuals were created by applying reproduction, sub-tree crossover and point mutation. We also used elitism to preserve the overall best found solution. Also, to control excessive code growth, the Tarpeian method [17] was utilised in the system. The termination criterion used was based on the predetermined maximum number of generations to be run.

2.2 Specialised Primitives

To make it possible for GHH to exploit the new possibilities offered by its ability to explore and prioritise vertices located at different depths in the level structure, we provided two special primitives, `ED` and `DFSV` (see Table 1).

[2] Parameters were selected after conducting a number of preliminary experiments, considering both the quality of solutions and run times.

Table 2. Parameters used for our runs

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%

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 ERP solvers. The degree of a vertex is the number of vertices connected to that vertex. There is no doubt that this is of fundamental importance in node ordering algorithms for ERP. However, we found that prioritising using 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, all vertices in a level are located at the same distances from the root vertex. Since in traditional ERP solvers 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 GHH, 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. This may help prioritise vertices and break ties.[3]

2.3 Vertex Selection

Let us analyse Algorithm 1 from the 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*) 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. 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 a permutation associated with the

[3] Sloan [20] also uses a distance quantity in his algorithm, but he computes distances from the end node of a pseudo-diameter.

original program (Step 14). The vertices located in l are then ordered based on the permutation generated (Step 15).

In Step 16, the first element of l is then removed and considered as a new 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), a new adjacency list is generated (Step 18), and its envelope is computed (Step 19).

2.4 Training and Test Sets

We used a training set of 25 benchmark instances G_i from the Harwell-Boeing sparse matrix collection. This is a collection of standard test matrices arising from problems in FEM 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, BCSSTRUC3, CANNES, LANPRO and LSHAPE with sizes ranging from 24×24 to 960×960. This training set was used only to evolve the heuristics. The performance of the evolved heuristics was then evaluated using a completely separate test set of 30 matrices taken from Everstine's collection (DWT) and BCSPWR, both included in the Harwell-Boeing database. 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 the BCSPWR set, which is in a totally different class compared to the training set used. We did this to assess how well the generated heuristics generalised in unseen situations.

3 Results

Ten independent runs of GHH with the training set specified above were performed, and the corresponding best-of-run individual in each was recorded. We then selected as our overall best evolved heuristic the best program tree from these ten best-of-run results.[4] The simplified version of the best heuristic evolved by GHH 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 shown graphically in Figure 1. The function is monotonic in both `ED` and `DFSV`. For small values of `ED`, nodes closer to the root are preferred

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

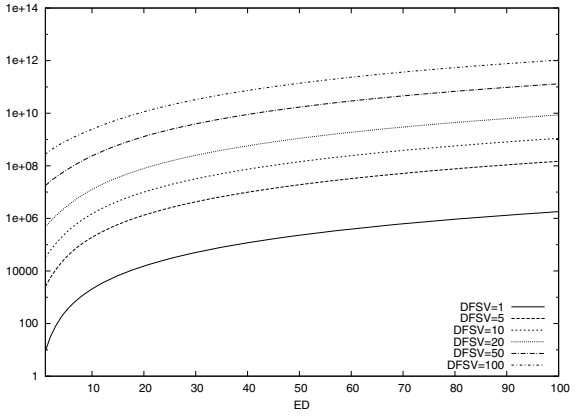

Fig. 1. Plot of the best GHH heuristic

over nodes further away. So, in a highly sparse matrix the algorithm behaves similarly to CM. 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.

We incorporated this heuristic into a level structure system and carried out experiments with the test set. In order to assess the performance of the heuristic generated, we compared it against two well-known and high-performance algorithms: RCM and GK. In practice, both algorithms are still among the best and most widely used methods for envelope reduction. We also tested this heuristic against GP-HH, which is the best algorithm produced by an envelope-minimising version of our previous hyper-heuristic method for evolving *bandwidth* reduction heuristics [12]. Unlike GHH's heuristics, GP-HH is constrained to operating at only one level of a level structure at a time.

Table 3 shows a performance comparison of the algorithms under test. All results associated with RCM and GK on the DWT set were taken from [20]. Because there were no results available in the literature for the BCSPWR set, we used the highly enhanced version of the RCM algorithm contained in the MATLAB library to compute the related envelopes. We do not report the results of GK on the BCSPWR problems as we did not have access to the original code, or a reliable software package.

As shown in the table, the results of GHH are extremely encouraging with respect to the mean of the envelope values and the number of the best results obtained (shown in the "Wins/Draws" rows). GHH's best evolved program outperforms RCM, GK and GP-HH's best evolved program by a significant margin, and produces extremely good results for the BCSPWR set.

Our system was implemented in C#, and all the experiments were performed on an AMD Athlon(tm) Dual-core Processor 2.20 GHz. We measured the time required for our method to solve each problem instance on this computer. The running times for DWT 59 (the smallest instance) and BCSPWR10 (the largest instance) were 0.0307 and 1.0094 seconds, respectively, while the average running

Table 3. Comparison of GHH's best evolved program against the RCM and GK algorithms as well as GP-HH's best evolved program

		Envelope			
Instance	Dimension	RCM	GK	GP-HH	GHH
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.

time was 0.1605 seconds. This reveals that our evolved algorithm is not only very effective but also extremely efficient.

4 Conclusions

We have proposed a hyper-heuristic approach (GHH) based on genetic programming for evolving envelope reduction algorithms. The algorithm is novel not only from the point of view of being the first to use GP on this problem but also because it incorporates new ideas for using a level structure system without its conventional constraints. Also, we have employed two novel features in the process of prioritising nodes for the construction of permutations.

The best heuristic generated by GHH were compared against two well-known and high-performance algorithms, i.e., the RCM and GK, as well as the best heuristic evolved by a hyper-heuristic method we previously developed, on a large set of standard benchmarks from the Harwell-Boeing sparse matrix collection. GHH's best evolved heuristic showed remarkable performance, both on benchmark instances from the same class as the training set and also on large problem instances from a totally different class, confirming the efficacy of our approach. The evolved heuristic was also extremely efficient.

References

1. 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 2007: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1749–1749. ACM Press, London (2007)
2. Barnard, S.T., Pothen, A., Simon, H.D.: A spectral algorithm for envelope reduction of sparse matrices. In: Supercomputing 1993: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, pp. 493–502. ACM, New York (1993)
3. Barnard, S.T., Pothen, A., Simon, H.D.: A spectral algorithm for envelope reduction of sparse matrices, dedicated to william kahan and beresford parlett (1993), `http://citeseer.ist.psu.edu/64928.html`
4. 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. International Series in Operations Research & Management Science, ch. 16, pp. 457–474 (2003)
5. Cowling, P.I., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
6. 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)
7. 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)

8. George, J.A.: Computer implementation of the finite element method. Ph.D. thesis, Stanford, CA, USA (1971)
9. Gibbs, N.E.: A hybrid profile reduction algorithm. ACM Transactions on Mathematical Software 2(4), 378–387 (1976)
10. 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)
11. Keller, R.E., Poli, R.: Linear genetic programming of parsimonious metaheuristics. In: Srinivasan, D., Wang, L. (eds.) 2007 IEEE Congress on Evolutionary Computation, September 25-28, pp. 4508–4515. IEEE Computational Intelligence Society, IEEE Press, Singapore (2007)
12. Koohestani, B., Poli, R.: A hyper-heuristic approach to evolving algorithms for bandwidth reduction based on genetic programming. In: Bramer, M., Petridis, M., Nolle, L. (eds.) Research and Development in Intelligent Systems XXVIII, pp. 93–106. Springer, London (2011)
13. Koza, J.R.G.P.: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
14. Oltean, M.: Evolving evolutionary algorithms using linear genetic programming. Evolutionary Computation 13(3), 387–410 (Fall 2005)
15. Pissanetskey, S.: Sparse Matrix Technology. Academic Press, London (1984)
16. 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
17. 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, May 20-22, vol. 8, ch.5, pp. 71–90. Springer, Ann Arbor (2010)
18. Poli, R., Langdon, W.B., Holland, O.: Extending Particle Swarm Optimisation via Genetic Programming. In: Keijzer, M., Tettamanzi, A.G.B., Collet, P., van Hemert, J., Tomassini, M. (eds.) EuroGP 2005. LNCS, vol. 3447, pp. 291–300. Springer, Heidelberg (2005)
19. Poli, R., Woodward, J., Burke, E.K.: A histogram-matching approach to the evolution of bin-packing strategies. In: Srinivasan, D., Wang, L. (eds.) IEEE Congress on Evolutionary Computation, September 25-28, pp. 3500–3507. IEEE Computational Intelligence Society, IEEE Press, Singapore (2007)
20. Sloan, S.W.: A FORTRAN program for profile and wavefront reduction. International Journal for Numerical Methods in Engineering 28(11), 2651–2679 (1989)
21. Wang, Q., Shi, X.W.: An improved algorithm for matrix bandwidth and profile reduction in finite element analysis. Progress In Electromagnetics Research Letters 9, 29–38 (2009)