PAGE NUMBER PROBLEM: AN EVALUATION OF HEURISTICS AND ITS SOLUTION USING A HYBRID EVOLUTIONARY ALGORITHM

Dharna Satsangi[1], Kamal Srivastava[2] and Gursaran[3]

Department of Mathematics, Dayalbagh Educational Institute,
Agra, India
[1]dharna.satsangi@gmail.com
[2]kamalsrivast@gmail.com
[3]gursaran.db@gmail.com

ABSTRACT

The page number problem is to determine the minimum number of pages in a book in which a graph G can be embedded with the vertices placed in a sequence along the spine and the edges on the pages of the book such that no two edges cross each other in any drawing. In this paper we have (a) statistically evaluated five heuristics for ordering vertices on the spine for minimum number of edge crossings with all the edges placed in a single page, (b) statistically evaluated four heuristics for distributing edges on a minimum number of pages with no crossings for a fixed ordering of vertices on the spine and (c) implemented and experimentally evaluated a hybrid evolutionary algorithm (HEA) for solving the pagenumber problem. In accordance with the results of (a) and (b) above, in HEA, placement of vertices on the spine is decided using a random depth first search of the graph and an edge embedding heuristic adapted from Chung et al. is used to distribute the edges on a minimal number of pages. The results of experiments with HEA on selected standard and random graphs show that the algorithm achieves the optimal pagenumber for the standard graphs. HEA performance is also compared with the Genetic Algorithm described by Kapoor et al. It is observed that HEA gives a better solution for most of the graph instances.

KEYWORDS

Graph layout, Page number problem, Book embedding of a graph, Hybrid Evolutionary Algorithm

1. INTRODUCTION

The book embedding problem consists of embedding a graph in a book with its vertices placed on a line along the spine of the book and its edges placed on the pages of the book in such a way that edges residing on the same page do not cross. The *pagenumber* of a graph *G* is the minimum number of pages of the book into which *G* can be embedded. This problem abstracts layout problems arising in direct interconnection networks, fault-tolerant processor arrays, fault tolerant VLSI design, sorting with parallel stacks and single row routing [1]. The problem of obtaining the pagenumber is NP-complete for general graphs [1]. However, this problem has been studied for some standard graphs and their pagenumbers have been found. Pagenumbers of FFT (Fast Fourier Transforms), Benes, Barrel Shifter networks [2], complete graphs, grids, hypercubes, trees, X-trees, pinwheel graphs [3], planar graphs, deBruijn and shuffle-exchange graphs [4,5] are known. Shahrokhi et al. [6] have given polynomial time algorithms to generate near optimal drawing of graphs on books. Berhart et al. [7] showed that the pagenumber is less than or equal to one if and

DOI : 10.5121/ijcseit.2012.2212

only if the graph is outerplanar. Swaminathan et al. [8] have shown that bandwidth-*k* graphs can be embedded in a book of *k*-1 pages, though it is not a tight bound when considered on standard graphs with known bandwidth. Upper and lower bounds on the pagenumber of *k*-ary hypercubes are given by Bettayeb et al. [9]. In order to deal with the problem of obtaining the pagenumber for general graphs, a genetic algorithm (GA) for finding the pagenumber has been proposed by Kapoor et al. [1].

The pagenumber problem (PNP) is formally defined as follows. Let *G*=(*V*, *E*) be an undirected graph with vertex set V and edge set E. A labeling of the vertices (called *vertex labelling* or *vertex ordering* or *vertex layout*) of an undirected graph with *n*=|*V*| vertices, is a bijective function $\varphi: V \rightarrow [n] = \{1,2, \dots, n\}$. The pagenumber is the minimum number of pages used in placing all the edges of edge set E of the graph G for the vertex ordering φ such that there is no crossing between the edges on any page in any drawing. We will refer to this as the optimal pagenumber of *G*. A pair of edges uv and pq cross in a drawing iff $1 \leq \varphi(u) < \varphi(p) < \varphi(v) < \varphi(q) \leq n$ and both lie on the same page.

The length of an edge uv for a labeling φ is defined as $\lambda(uv, \varphi, G) = |\varphi(u) - \varphi(v)|, uv \in E$. Let $E' = \{uv \in E(G)/1 < \lambda(uv, \varphi, G) < n\}$. We store the edges of E' in a matrix $EM = \{e_{ij}\}_{|E'\times 2|}$ where e_{i1} and e_{i2} are the end vertices of the i^{th} edge. The following formula counts the number of crossings for labeling φ when all the edges are embedded on a single page.

$$\sum_{i=1}^{|EM|} \sum_{j=i+1}^{|EM|} (e_{i,1}, e_{i,2}) \Delta (e_{j,1}, e_{j,2})$$

where,

$$(u, v)\Delta(p, q) = \begin{cases} 1, & \varphi(u) < \varphi(p) < \varphi(v) < \varphi(q) \\ 0, & otherwise \end{cases}$$

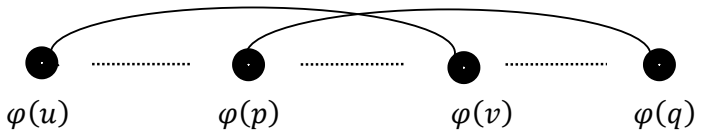


Figure 1. Edge crossing condition $\varphi(u) < \varphi(p) < \varphi(v) < \varphi(q)$

In devising an embedding for minimizing the page number two main factors are important: 1) placement of vertices on the spine and 2) distribution of edges over the pages of the book. Once the order of the vertices on the spine is fixed, the pages required to embed the graph (without any crossings) depend on the manner in which the edges are assigned a page. This paper addresses the two factors and the meta-heuristic generation of solutions to the page number problem and is divided in three parts:

a) In the first part, described in Section 2 and Section3, five heuristics for vertex ordering are statistically evaluated on a suite of test graphs that includes standard as well as random graphs. A vertex ordering *x* is considered to be better than vertex ordering *y* if the number of edge crossings with *x* is smaller than that with *y* when all the edges are placed, or embedded, on *one* page. The results of the experiments were statistically analyzed and the best technique was used for experiments carried out in the second part of the work. Results show that the vertex ordering obtained by a depth first search of the graph provides significantly fewer crossings than the other heuristics.

b) In the second part, described in Section 4 and Section 5, four heuristics for edge distribution are statistically evaluated on the suite of test graphs, described in part (a) above, for fixed vertex orderings determined by depth first search. Experiments reveal that an edge distribution heuristic based on the strategy proposed by Chung et al. [3], which was proposed to distribute edges of a complete graph across a minimal number of pages, is highly effective. This heuristic outperforms the remaining three heuristics evaluated in this work.
c) In the third part, described in Section 6, 7 and 8, a Hybrid Evolutionary Algorithm (HEA) has been implemented and experimentally evaluated for the pagenumber problem (PNP). Experimental results show that HEA outperforms genetic algorithm proposed by Kapoor et al. [1] for PNP on the graphs on which the two were compared.

Section 9 presents the conclusions of the work described in the paper.

2. Heuristics for Vertex Ordering

Five heuristics were evaluated for generating vertex orderings for a given graph $G=(V,E)$. These are:

1. *Random depth first vertex ordering* (*rdfs*)
   The graph is traversed in depth first order wherein the root vertex and the neighbors of the vertices are visited randomly. The order in which a vertex is visited by the search becomes the label of the vertex.
2. *Random breadth first vertex ordering* (*rbfs*)
   The graph is traversed in the breadth first order wherein the root vertex and the neighbors of the vertices are visited randomly. The order in which a vertex is visited by the search becomes the label of the vertex.
3. *Random vertex ordering* (*rand*)
   The vertices of the graph are labeled randomly.
4. *Vertex cover based vertex ordering* (*Vcover*)
   $i=1$;
   while(*V* is not empty)
   find vertex v in V having highest degree
   $\varphi(v) = i$;
   $i=i+1$;
   $V=V-\{v\}$;
   end
   Vertices with equal degrees are chosen randomly whereas isolated vertices are taken in sequential order.
5. *Max-neighboring vertex ordering* (*maxNbr*)
   At each step find vertex v of G (V, E) having highest degree and label it. Now find neighbors of v and sort them in descending order of their degrees and assign the labels accordingly. Remove v and all its neighbors from the graph. Vertices with equal degrees are chosen randomly whereas isolated vertices are taken in sequential order.

3. Experiments and Results on Heuristics for Vertex Ordering

As defined earlier, vertex ordering x is considered to be better than vertex ordering y if the number of edge crossings with x is smaller than that with y when all the edges are placed, or embedded, on *one* page. In order to determine if there is a significant difference in the number of crossings for vertex orderings generated using the five heuristics identified in Section 2 and also

to possibly identify the best heuristic, a number of experiments were carried out on a suite of standard and random graphs. These experiments together with the results are described in this section.

3.1. Suite of Test Graphs

The test suite consists of standard and random graphs. Standard graphs were chosen whose pagenumber results are known or can be computed. However, for these graphs the minimum crossing number on a single page is not known.

3.1.1. Standard Graphs

- *Complete*: A complete graph with n vertices denoted by K_n, is a simple graph in which every two distinct vertices are joined by exactly one edge. It is regular graph of degree n-1, and total edges equal to $n(n\text{-}1)/2$.
- *Complete bipartite*: A complete bipartite graph, $K_{m,n}$ with $|V|=m+n$ is a simple bipartite graph such that two vertices are adjacent if and only if they are in different partite sets.
- *Cycle*: A cycle C_n is a graph with an equal number of vertices and edges whose vertices can be placed around a circle so that two vertices are adjacent if they appear consecutively along the circle.
- *Hypercubes*: The hypercube, Q_d, *of* dimension d, is a d-regular graph with $2d$ vertices and $d2^{d-1}$ edges. Each vertex is labeled by a distinct d-bit binary string, and two vertices are adjacent if they differ in exactly one bit.
- *Cube-connected cycles*: The cube-connected cycle CC_d, of dimension d is formed from Q_d by replacing each vertex u with a d-cycle of vertices in CC_d and then joining each cycle vertex of the corresponding neighbor of u in Q_d.
- *Complete binary tree*: The binary tree, $B(n)$ is a tree structure in which each node has at most two child nodes. The binary tree is complete if the nodes are well balanced.
- *X-trees*: The depth-d X-tree, $X(d)$ is the edge augmentation of the depth-d complete binary tree that adds edges going across each level of the tree in left-to-right order.
- *Pinwheel*: The depth-n pinwheel graph, $P(n)$ has $2n$ vertices
  $\{a_1,a_2,\ldots,a_n\}$
  and
  $\{b_1,b_2,\ldots,b_n\}$
  and edges connecting each pair of vertices of the form
  $a_i - b_i$ $\quad 1 \le i \le n$,
  $a_i - b_{n-i+1}$ $\quad 1 \le i \le n$,
  $a_i - a_{i+1}$ $\quad 1 \le i < n$,
  $b_i - b_{i+1}$ $\quad 1 \le i < n$.
- *Star:* The star graph S_k is a complete bipartite graph $K_{1,k-1}$, a tree with one internal node and k-1 leaves. It has k vertices and k-1 edges.
- *Triangulated triangle*: The Triangulated triangle graphs, T_l is a graph whose vertices are the triples of non-negative integers summing to l, with an edge connecting two triples if they agree in one coordinate and differ by one in the other two coordinates.
- *Toroidal Mesh*: A 2-dimensional toroidal mesh is defined by $C_m \times C_n$ with $|V|=mn$.
- *Shuffle Exchange* $S(D)$: Its vertices are 0-1 vectors of length D. There is an edge between any vertex $(v_0, \ldots, v_{D-1})$ and vertex $(v_1, \ldots, v_{D-1}, v_0)$. Also, there is an edge between any vertex $(v_0, \ldots, v_{D-1})$ and the vertex $(v_1, \ldots, v_{D-2}, \alpha)$, $\alpha \neq v_{D-1}$.

3.1.2 Random Graphs

Random graphs of 40, 50 and 60 vertices and edge density 30%, where edge density is taken as $2m/n(n-1)$ were generated for the experiment. It was also ensured that the generated graph is connected and undirected.

3.2. Methodology and Results

The following methodology was adopted for the experiments:

1. For each test graph and each heuristic, fifty different vertex orderings were randomly generated
2. For each vertex ordering the crossings on a single page was determined.
3. Basic statistical analysis was then carried out for each graph and each heuristic over the fifty vertex orderings. This included determining the minimum crossing, mean crossing and standard deviation. Further ANOVA was used to determine if there was a significant difference in the means. The reason behind collecting the mean, standard deviation (Sd) and the ANOVA statistics was also to determine if a method can result in vertex orderings with large variation in number of crossings. A method would not be acceptable if the mean is high and the standard deviation is also high and also if the mean is high and the standard deviation is low. Both cases imply that vertex orderings are more likely to be generated that lead to a large number of crossings.

In this section we first report the results on standard graphs and then on random graphs.

3.2.1. Standard Graphs

From Figures 2 to 8, the following observations can be made:

- *rdfs* gives the least value for all the graph instances. In 66 (77.6%) instances it gives the least value among all the techniques. All these instances are small sized graphs.
- The mean value for *rdfs* is a minimum for all the instances.
- In the case of *rdfs* for X-tree, shuffle exchange and triangulated triangle, it can be observed that the Sd increases as the number of vertices increase. This can also be seen in Figure 9 and Figure 10. This implies that as the number of vertices increase, *rdfs* may give a layout for which the number of crossings is large. However, since the mean and minimum value is least for all the selected instances, *rdfs* seems to be an obvious choice for obtaining a layout. This is also supported by Figure 9. In Figure 9 we can see that as the number of vertices increase, *rbfs* and *rand* generate layouts with larger crossing numbers for all fifty layouts.
- *maxNbr* gives a low value of Sd but high mean for most of the graph instances. This is because of the nature of the algorithm which generates almost identical vertex orderings of the graphs in each simulation.
- *rand* results in a large minimum value, large Sd and high mean for most of the instances. *rdfs* as benchmarked against *rand* gives very good results.
- *Vcover* gives highest minimum value for almost (except seven cases) all the graph instances.

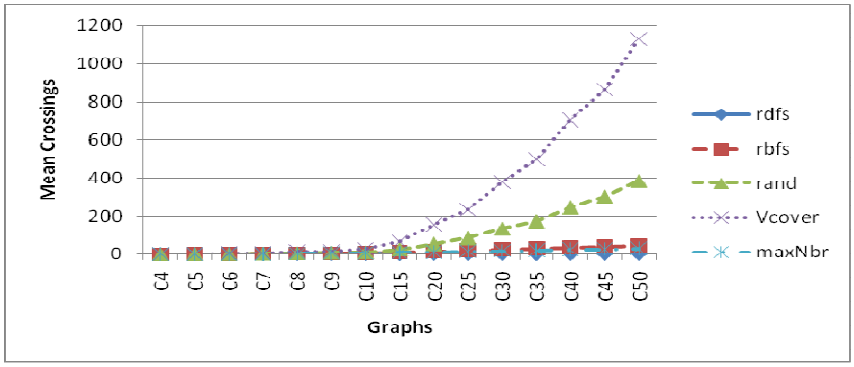


Figure 2. Mean crossings for cycle graphs

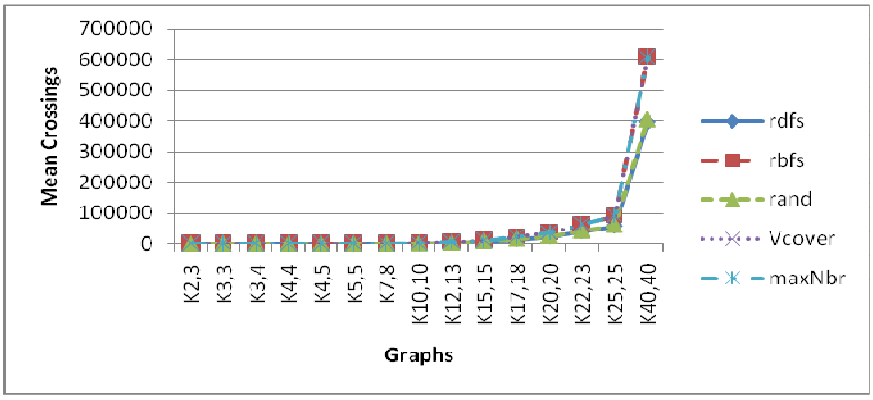


Figure 3. Mean crossings for complete bipartite graphs

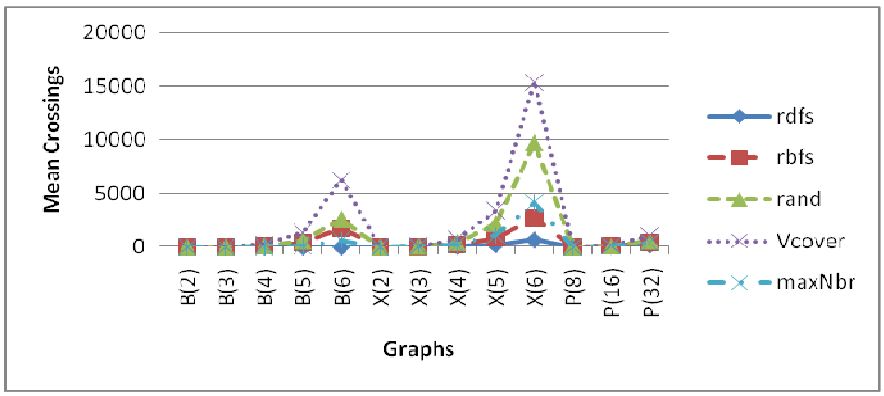


Figure 4. Mean crossings for binary trees, X-trees and pinwheel graphs

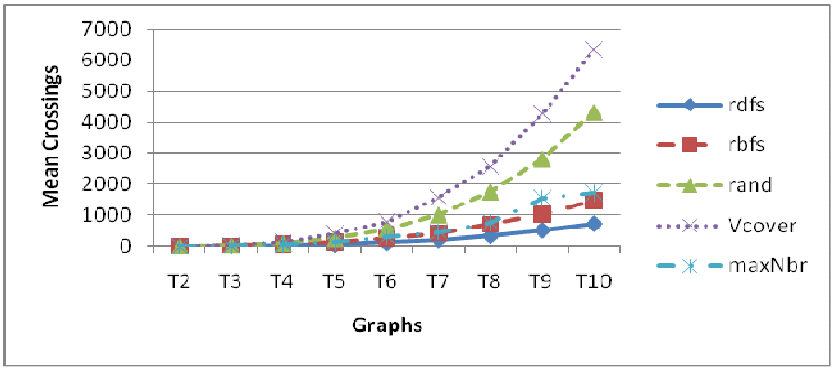


Figure 5. Mean crossings for triangulated triangle graphs

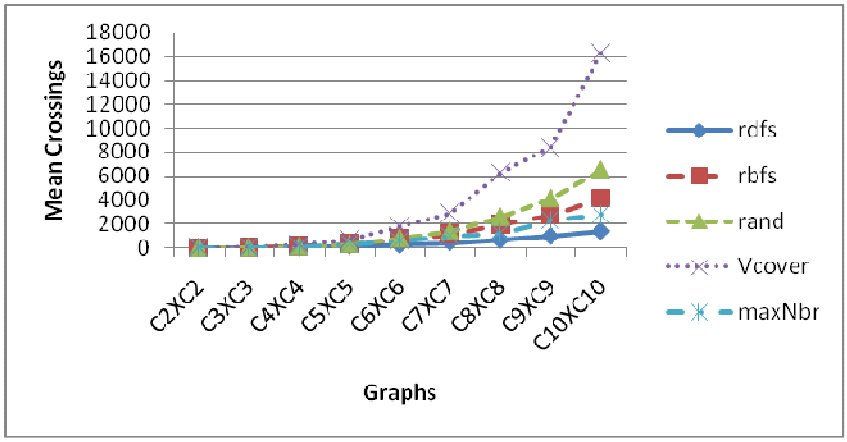
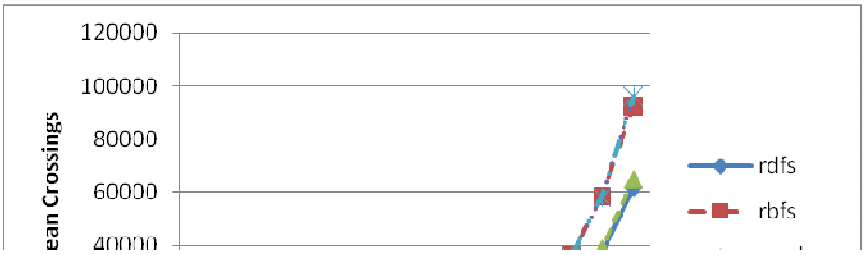


Figure 6. Mean crossings for toroidal meshes



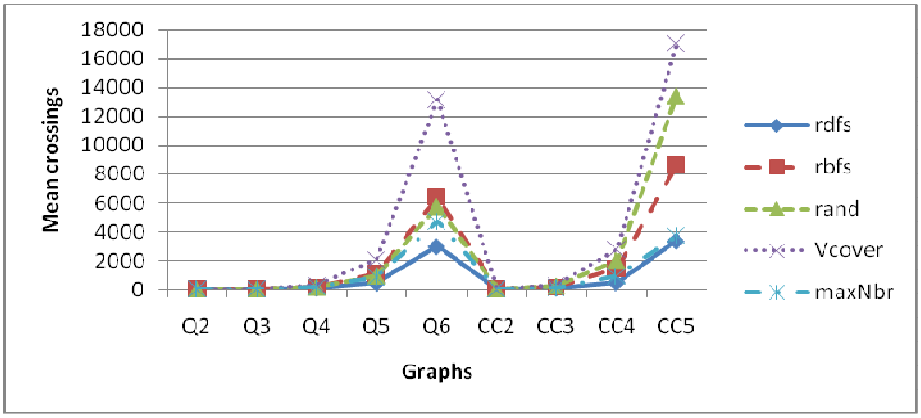


Figure 8. Mean crossings for hyper cubes and cube-connected cycle graphs

To test whether the difference in mean number of crossings for different vertex ordering techniques is significant, we ran one way ANOVA to compute F and p values. From these ANOVA values it can be concluded that the difference is significant. This together with the mean and minimum values gives another reason to choose *rdfs* over all the other techniques.

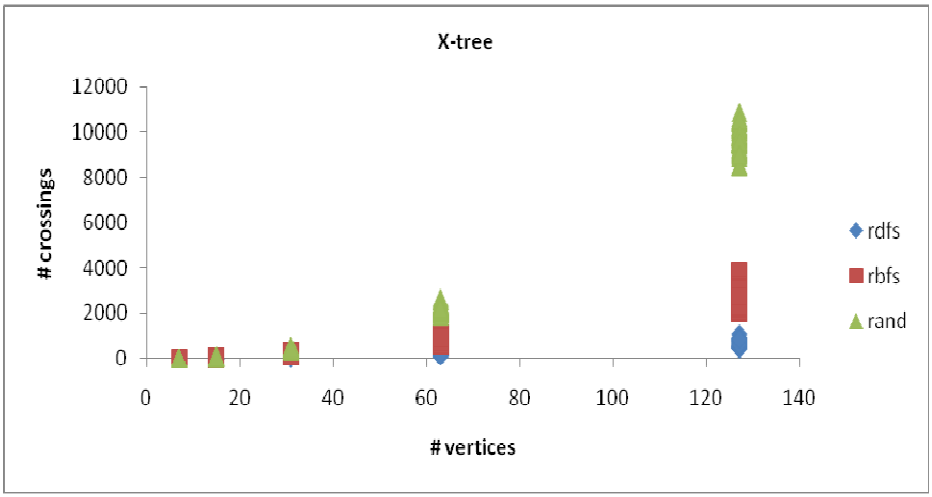


Figure 9. Scatter plot for X-tree (d=2, … , 6) for *rdfs, rbfs* and *rand*

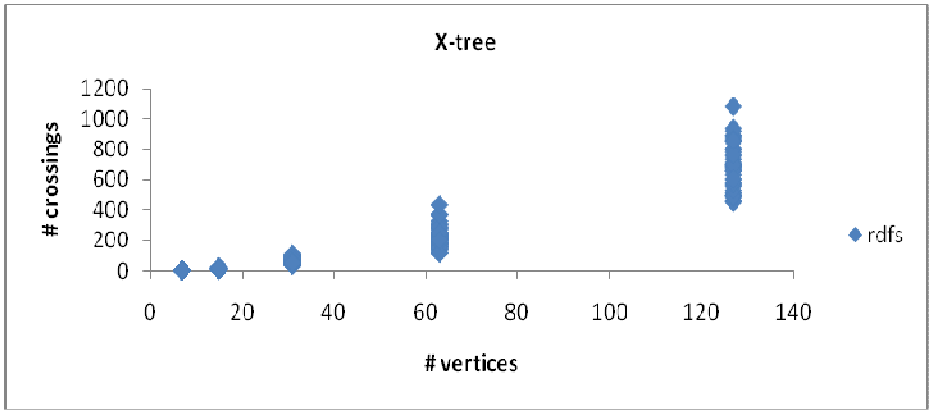


Figure 10. Scatter plot for X-tree (d=2, … , 6) for *rdfs*

3.2.2. Random Graphs

To further test the performance of the vertex ordering techniques mentioned above, we generated three random connected graphs, one each of 40, 50 and 60 vertices and edge density 30%. For each graph fifty layouts were generated with each of the vertex ordering techniques and a statistical analysis was carried out. The results of this analysis are shown in Table 1. From the table it is clear that the minimum and the mean value is the lowest for *rdfs*. ANOVA also shows that there is a significant difference in the means obtained by various techniques. Thus *rdfs* is chosen as the technique for generating layouts for further experiments on PNP.

Table 1. Results on random graphs for different heuristics for vertex ordering

***n*=|*V*|**		***rdfs***	***rbfs***	***rand***	***Vcover***	***maxNbr***	***F-value***	***p-value***
40	Mean	6694	8520	8220	10889	8695	514.513	0.00
	Sd	334.4334	400.1506	435.5663	20.54105893	15.00097		
	Min	6009	7804	7101	10779	8599		
50	Mean	17430	21506	20819	26441	20780	17407.723	0.00
	Sd	699.9343	723.1122	863.5149	731.591	686.8343067		
	Min	16261	20132	19228	24759	19331		
60	Mean	38298	45197	43673	54837	42419	45766.676	0.00
	Sd	1290.958	1202.054	1595.302	14.89122	25.56647		
	Min	35345	42858	39852	54800	42332		

4. Edge distribution heuristics

We now describe four edge distribution heuristics. Using these heuristics, an attempt is made to distribute edges on a minimal number of pages such that there are no crossings on any page for any drawing.

1. *Greedy edge distribution* (*gr*)
   One edge of the graph at a time is picked in a random order and is placed on a page where it does not cross with any of the already placed edges. If no such page exists then a new page is taken for placing the edge.
2. *Edge-length based edge distribution* (*eg*)
   This technique is motivated by edge-length heuristic proposed by Cimikowski et al. [10] for 2-page crossing number problem. The edges of the graph are taken in the decreasing order of their length and are placed one at a time on the page where it does not cause any crossing. A new page is added whenever required.
3. *Ceil-floor edge distribution* (*cf*)
   This edge embedding strategy is proposed by Kapoor et al. [1]. The sequence in which the edges are considered for edge embedding depends on their lengths which are given as:
   $\lfloor n/2 \rfloor, \lceil n/2 \rceil, \lfloor n/2 \rfloor - 1, \lceil n/2 \rceil + 1, \lfloor n/2 \rfloor - 2, \lceil n/2 \rceil + 2, \ldots$
4. *Circular edge distribution* (*circ*)
   This heuristic is motivated by the edge distribution strategy for pagenumber [3] of complete graphs.
   Steps:
   - Consider a labeling $\varphi = 1, 2, \ldots, n$ of vertices in graph G.
   - Now for a complete graph with vertex labels 1, 2, …, n, place the edges ($x \leftrightarrow y$ denotes an edge with endpoints x and y)of K_n in a sequence S according to the procedure given below:
     Initialize S to the empty sequence;
     for v=1 to $\lceil n/2 \rceil$
     S+=$v \leftrightarrow (v+1) \leftrightarrow (v-1) \leftrightarrow (v+2) \leftrightarrow (v-2) \leftrightarrow \ldots \leftrightarrow (v+\lceil n/2 \rceil - 1)$
     $\leftrightarrow (v-\lceil n/2 \rceil + 1) \leftrightarrow (v+\lceil n/2 \rceil)$
     endfor
     where += means concatenate with the sequence obtained from the previous step and addition and subtraction are modulo operators assuming labels are placed in a circle. Note that S includes all the edges in a complete graph of n vertices with some edges possibly included more than once.
   - Now $E(G) \subseteq E(K_n)$. Insert edges in $E(G)$ in the same sequence as they appear in S in edge matrix Q, without duplicating the edges.
   - Place the edges in Q one by one, in sequence, starting on page one using the following rule: Assign page one to the first edge. For each e_i in Q_i, $2 \leq i \leq |E|$, assign the lowest numbered page to e_i if adding e_i to the page does not result in a crossing. Otherwise assign a new page to e_i.
   - Pagenumber = total number of pages assigned.

5. Experiments and results for edge distribution heuristics

The results obtained by these heuristics for standard graphs with their upper bounds/ optimal values are shown in the Table 2. These upper bounds/ optimal values for the pagenumber are given in [1, 3, 4, 5, and 8].

5.1. Standard Graphs

Bold values in the table indicate lowest value of pagenumber obtained by the experiments. From Table 2 it is clear that the minimum pagenumber is obtained with the circular edge distribution heuristic of Chung et al. [3], except for $P(16)$, $T4$ and $T5$.

Table 2. Results on standard graphs for different edge distribution heuristics

Graphs	***cf***	***circ***	***eg***	***gr***	**Upper bound/ Optimal**
S_6	1	1	1	1	2
S_{10}	1	1	1	1	4
S_{30}	1	1	1	1	14
T_3	2	2	2	2	3
T_4	3	3	2	2	4
T_5	4	3	2	2	5
T_6	4	4	4	4	6
T_7	4	3	3	4	7
T_8	7	**6**	7	8	8
T_{10}	6	5	5	6	10
$K_{25,25}$	30	**25**	29	40	18
$K_{40,40}$	47	**40**	47	67	28
C_{10}	1	1	1	1	1
C_{20}	1	1	1	1	1
C_{50}	1	1	1	1	1
$B(3)$	1	1	1	1	1
$B(6)$	1	1	1	1	1
K_8	6	**4**	6	5	4
K_{10}	7	**5**	7	8	5
K_{16}	13	**8**	13	13	8
K_{20}	15	**10**	14	17	10
K_{40}	34	**20**	35	37	20
K_{60}	48	**30**	54	59	30
K_{100}	86	**50**	85	102	50
K_{150}	135	**75**	137	154	75
Q_3	2	2	2	2	2
Q_4	4	4	4	6	3
Q_5	7	7	8	9	4
$X(5)$	5	5	5	5	2
$X(7)$	8	6	6	7	2
$S(4)$	1	1	1	1	3
$S(7)$	3	3	4	3	3
$S(10)$	7	5	5	6	3
$S(20)$	13	**10**	13	15	3
$S(30)$	19	**15**	19	22	3
CC_3	3	3	3	3	5
CC_4	8	8	8	8	5
CC_5	16	15	15	18	5
$P(8)$	2	2	2	2	3

P(16)	3	4	3	4	3
P(32)	6	5	5	5	3

5.2. Random Graphs

One random connected graph for each of 40, 50, 60, 70 vertices having an edge density of 30% was generated. For each graph fifty different vertex orderings were generated and each edge distribution heuristic was run on each vertex ordering. Since each edge distribution heuristic was run on the same set of vertex orderings, p*aired t-test* was carried out to test significance in difference of means. The results of the analysis are shown in Table 3. In *paired t-test* the confidence level is taken as 99% and Dunn-Sidak adjusted probabilities are calculated using SYSTAT 9. The mean pagenumber by these techniques is used to compare the pagenumber of graphs.

Table 3. Results of paired t-test on random graphs for edge distribution heuristics

Techniques	**t-value**	**Dunn-Sidak Adjusted Probability**
cf and *circ*	30.147	0.0
cf and *eg*	-6.149	0.0
eg and *circ*	-31.347	0.0
cf and *gr*	-33.297	0.0
circ and *gr*	-37.561	0.0
eg and *gr*	-31.244	0.0

It is clear from Table 3 that the difference in means is significant for all heuristics. Considering the plot in Figure 11, it is evident that circular edge distribution heuristic (*circ*) gives the best results for mean and Figure 12 shows that it gives the smallest value for page number. Figure 13 shows the complete statistics for circular edge distribution heuristic. *circ*, as described earlier, is based on the strategy proposed by Chung et al. [3] for complete graphs.

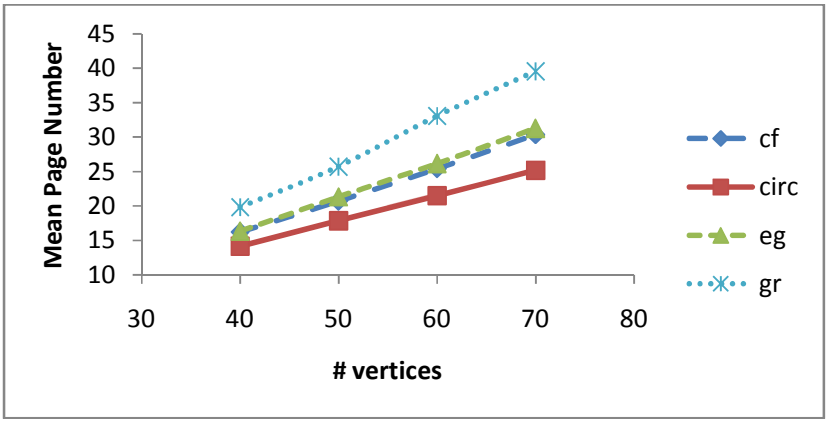


Figure 11. Mean Page number of Random Graphs

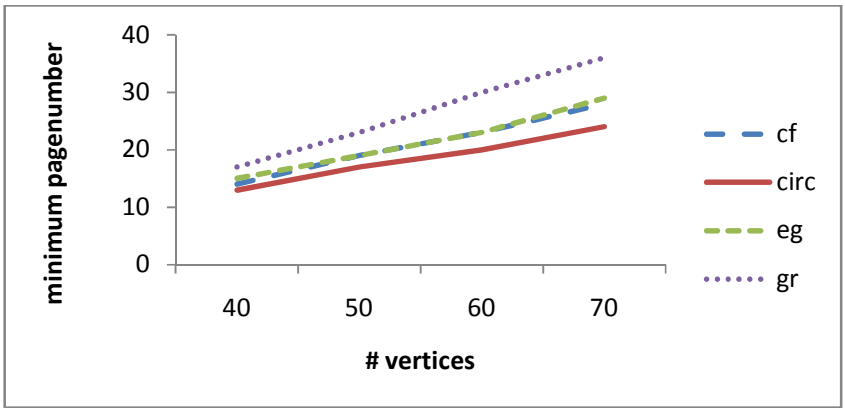


Figure 12. Minimum Page number of Random Graphs

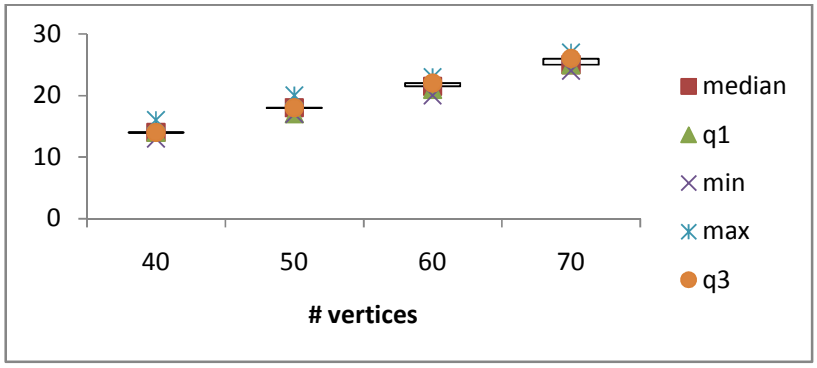


Figure 13. Box Plot of random graphs for circular edge distribution heuristic

6. Hybrid Evolutionary Algorithm for PNP

In this section we describe a hybrid evolutionary algorithm for PNP, followed by its implementation details in Section 7. Evolutionary algorithms (EAs) are the approximate optimization algorithms that are based on evolutionary principles. They are inspired by nature's capability to evolve living beings well adapted to their environment. At the core of EA is a population of individuals. In each iteration in the algorithm, *reproduction* operators are applied to the individuals of the current population to generate the individuals of the next generation. *Recombination* operators may also be used to recombine two or more individuals to produce new individuals. The algorithm may also use *mutation* operators that cause self-adaptation of individuals. The driving force in the EAs is the *selection* of individuals based on their fitness (which might be based on the objective function or some kind of simulation experiment). Individuals with higher fitness have a higher probability to be chosen as members of the next generation (parents) which corresponds to the principle of *survival of fittest* in natural selection. Simulated annealing (SA) proposed by Kirkpatrik et al. [11] is another powerful technique used to solve optimization problems. The algorithm is of sequential nature and converges slowly. The robustness of EAs is usually enhanced when they are hybridized with other meta heuristics such as tabu search [12], SA etc. Yip and Pao [13] proposed to incorporate the process of SA into the selection process of EA and to utilize its power to guide local search. A detailed outline of the algorithm involving the hybridization of EA with SA in the context of PNP is given in Section 6.2 followed by its pseudo code.

6.1. Solution Representation

In HEA, for a graph $G(V, E)$, $V=\{V_1, V_2, ..., V_n\}$, each solution, $\zeta=(R, Q)$ is represented in two components:

1) An array R of length n that represents a labeling φ. Vertices are placed on the spine in the label sequence 1,2, …, n.
2) An $|E|\times 3$ array Q, where each row contains the end vertices of an edge in the first two columns and the third column stores the page index in which the edge is placed.

The page number of a solution ζ is $max_{\forall i}\, Q[i][3]$.

6.2. The Algorithm

HEA starts with building an initial population of parents P_0 consisting of *pop_size* number of solutions produced by *rdfs* and then applying *circ* to distribute the edges on the pages. The SA parameters namely, initial temperature T_i, final temperature T_f, and cooling ratio α are initialized. Also a non-zero positive integer k is assigned to $ch\_num[i]$, $1 \le i \le pop\_size$, where $ch\_num[i]$ indicates the number of children to be produced by parent $P_t[i]$ in the t^{th} generation. It means initially each parent is given an equal chance to produce children. Clearly, the total number of children to be produced by the parents P_0 is $k \times pop\_size$. After generating the initial population of parents P_0 the iterative process (steps 7 to 23) starts in which each parent $P_t[i]$ generates $ch\_num[i]$ number of children using *idfs* (see section 3.3) and *circ*. We refer to the population of children as C_t, where $C_t[i][j]$ denotes the j^{th} child of parent $P_t[i]$. Now for each i, $1 \le i \le pop\_size$, child with lowest pagenumber (best_$C_t[i]$) is determined for finding the parents for the next generation. This child becomes the parent for the next generation if its pagenumber is less than that of parent; otherwise there is a probability that the child will be selected to become a parent (Steps 12 to 20). Next important step is to decide $ch\_num[i]$ for the next generation of parents P_{t+1}. This decision is based on the competition between parents wherein the performance of a parent is measured by comparing the pagenumber of each of its children with the *best_pagenumber,* where *best_pagenumber* is the lowest pagenumber obtained so far in this process. Pseudo code for computing $ch\_num[i]$ is given in the procedure *find_ch_num* at the end of this section. The iterative process is repeated until no improvement in the *best_pagenumber* is observed for a specified count (*cnt*) of generations or when final temperature is reached.
The pseudo code for HEA

1. Initialize *pop_size*, number of children to be produced by each parent of the initial generation k, cooling ratio α, initial temperature T_i, final temperature T_f.
2. Set $t \leftarrow 0$; $T \leftarrow T_i$;
3. **for** i=1 to *pop_size*
4. $ch\_num[i] \leftarrow k$;
5. **endfor**
6. Generate initial population of parents $P_t[i]$ using *rdfs* and then apply *circ* to distribute the edges on the pages
7. **Repeat**
8. **for** i=1 to *pop_size*
9. Apply *idfs* on $P_t[i]$, $ch\_num[i]$ times to create children $C_t[i][j]$, $1 \le j \le ch\_num[i]$ and then apply *circ* to distribute the edges on the pages
10. **endfor**
11. Apply mutation on $k \times pop\_size \times r_m$ number of children from C_t.
12. **for** i=1 to *pop_size*
13. Find child with lowest pagenumber i.e. *best_* $C_t[i]$

```
14.          β ←pg_no (G, P_t[i]) – pg_no (best_ C_t [i]);
15.          if  β >0 or exp (β/T)>ρ // ρ is a random number lying between 0 and 1
16.             P_{t+1}[i] ← best_ C_t [i];
17.         else
18.               P_{t+1}[i] ← P_t[i];
19.         endif
20.       endfor
21.       Update best_pagenumber
22.       T ← αT ; t ←t+1;
23. Until (stopping criteria is satisfied)
```

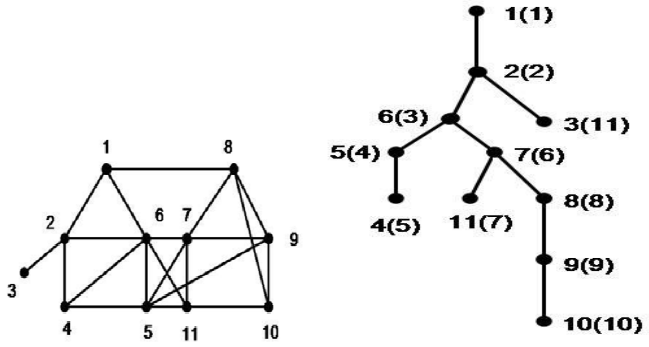
Procedure *find_ch_num*()

```
1.  sum←0;
2.  for i=1 to pop_size
3.          count[i]←0
4.          for  j=1 to ch_num[i]
5.                   β ← best_pagenumber – pg_no (G, C_t [i][j]);
6.              if  β >0 or exp (β/T)>ρ
7.                   count[i]← count[i]+1
8.              endif
9.       endfor
10.         sum←sum+ count[i]
11.  endfor
12.  for i=1 to pop_size
13.         ch_num[i] ←k×pop_size×count[i]/sum
14.  endfor
```

7. Implementation of Hybrid Evolutionary Algorithm

7.1. Initial Population

An initial population $P_0[i]$, i=1, ..., *pop_size*, consists of *pop_size* solutions. For each $P_0[i]$= (R_i, Q_i), R_i is generated using *rdfs* and *circ* is used to generate Q_i. In this section we take an example, shown in Figure 14a, to illustrate the components of a solution.



(a) (b)

Figure 14. (a) A graph and (b) its *dfs* tree

Starting from vertex 1, *rdfs* of the graph (*dfs* tree is shown in Figure 14(b)) provides the following labeling:

R=

1	2	11	5	4	3	6	8	9	10	7

In Figure 14b, for each vertex, the number in regular font are the vertex identifiers and the number in the parenthesis corresponds to its depth first index, i.e., the label of the vertex.
circ gives the distribution of edges in the array *Q* as

Q=

Edges	1	2	5	7	7	2	1	7	5	2	10	8	8	4	4	6	1	9	5	5	6
	2	3	9	8	11	6	6	9	6	4	11	10	9	5	6	7	8	10	7	11	11
Page #	1	1	1	1	1	1	2	1	1	3	2	2	1	1	2	2	3	1	1	4	2

The page number for this solution is 4.

7.2. Generating Children

We use a unary reproduction operator, intermediate depth first search (*idfs*) to produce children from the parents, which is explained as follows:

Let the parent *par_pop*[*i*] and the child child_*pop*[*i*] are denoted by arrays *S_par* and *S_child* respectively. A vertex *v* ($1 \leq v \leq n$) is selected randomly. The label of this vertex is *S_par*[*v*]. Labels of vertices with labels less than or equal to *S_par*[*v*] are copied from *S_par* to *S_child* at the respective positions. These vertices are referred to as *visited* and remaining ones are referred to as *unvisited*. This process is shown in pseudo code below:

1. **for** $\forall u \in V$
2. **if** *S_par*[*u*] $\leq$ *S_par*[*v*]
3. *S_child* [*u*] $\leftarrow$ *S_par*[u]
4. **endif**
5. **endfor**

Now starting from the vertex *v*, *dfs* of the *unvisited* vertices of the graph is carried out. The labeling of the *unvisited* vertices starts from *S_par*[*v*] +1 and the labels are stored at the respective positions of array *S_child*. This ensures that a new obtained child solution also has a *dfs* based labeling. Thus *idfs* helps to explore various possible *dfs* trees of the graph starting with the same root vertex.

Figure 15 shows a *dfs* tree obtained from that shown in Figure 14(b) when the *rdfs* is carried out after the randomly chosen vertex 6.

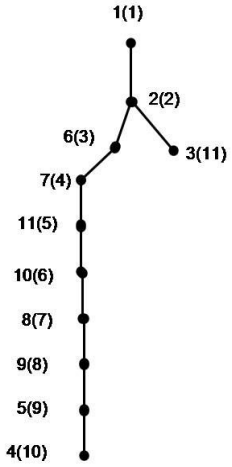


Figure 15. A *dfs* tree obtained by applying *idfs* on the tree of Figure 14.

The labelling obtained is

R=	1	2	11	10	9	3	4	7	8	6	5

Application of *circ* on this layout provides the following edge distribution over the pages. The page number for this solution is 3.

Q=	Edges	1	2	4	5	5	9	8	2	1	8	6	5	7	6	1	4	10	7	7	5	2
		2	3	6	7	11	10	10	6	6	9	7	9	11	11	8	5	11	8	9	6	4
	Page #	1	1	1	1	1	1	1	1	2	1	1	1	1	2	2	1	1	3	3	1	1

7.3. Mutation

In HEA, *insert mutation* is used to perturb the population of children C_t. In this, a vertex is selected randomly and is placed at another randomly chosen position on the spine. This facilitates the entry of non-*dfs* solutions in the population. Mutation is applied on $k \times pop\_size \times r_m$ number of randomly selected children, where r_m is mutation rate.

8. Experiments and Results for HEA

In this section, we present the results of our experiments for HEA and GA [1] which is preceded by the results of parameter tuning and GA settings.

8.1 GA Settings and Parameter Tuning

To compare HEA with the GA of Kapoor et al [1], we coded the GA with cyclic crossover and insert mutation operator as no specific operator is mentioned in their paper. Furthermore, as other GA parameters such as population size, mutation rate etc. are also not specified, the various parameters involved in HEA and GA are set after performing preliminary experiments on the randomly generated graphs. The size of initial population (*pop_size*) is set as $n=|V|$ and the mutation rate (r_m) of 0.5 is used in both the algorithms. The number of children generated

initially (i.e. k) is fixed at 3. The SA parameters, namely, initial temperature (T_i), final temperature (T_f), and cooling ratio (α) are chosen as 1, .01 and 0.99 respectively. In both the algorithms, the iterative process is repeated until no improvement is observed in *best_page number* for 50 consecutive iterations.

8.2 Results

The results of HEA and GA on various graphs are shown in Table 4 through Table 7. The results of standard graphs with known optimal page numbers are given in Table 4. HEA performs better than GA in most of the cases. The results in bold indicates the cases for which an algorithm has better performance than the other. HEA attains optimal page number for most of the graphs. The page number for complete graphs are all optimal as expected since *circ* is based on edge distribution strategy for the optimal page number of these graphs. GA does not perform well, for the huper cubes as the page numbers obtained by it are substantially higher than the optimal page number which is d-1. HEA attains these values easily when tested for $d \leq 6$. Both the algorithms attain the optimal page number 1 for the complete binary trees and star graphs which are obvious as these graphs are outer planar. HEA is able to obtain optimal page numbers of pinwheel graphs $P(n)$ tested for n = 8, 16 and 32.

Table 5 contains the results of those classes of standard graphs for which optimal page numbers are not known and for which some upper bounds are available for them. The results for shuffle exchange and triangulated triangles are much less than their known upper bounds. HEA and GA both give large page numbers of complete bipartite graphs. The reason for this may be attributed to the fact that the *optimal* page number is obtained with a non-*dfs* layout. Mean values of *page numbe*r for random graphs of sizes 40 and 50 obtained by GA and HEA are shown in Table 6 and Table 7 respectively. Results (Figure 16) clearly demonstrate that HEA outperforms GA in terms of solution quality though the elapsed time of HEA is more than that of GA. The difference in the page numbers obtained by HEA and GA is more significant in the case of higher density graphs.

Table 4. Comparison of GA and HEA for the standard graphs with known optimal page numbers

Graphs	GA	HEA	Optimal
K_{12}	7	**6**	6
K_{13}	7	**7**	7
K_{14}	8	**7**	7
K_{15}	9	**8**	8
K_{20}	13	**10**	10
K_{25}	18	**13**	13
K_{30}	22	**15**	15
K_{35}	26	**18**	18
K_{40}	31	**20**	20
K_{100}	86	**50**	50
K_{150}	135	**75**	75
$P(8)$	2	2	3
$P(16)$	3	3	3
$P(32)$	6	**3**	3
Q_3	2	2	2
Q_4	3	3	3
Q_5	7	**4**	4
Q_6	13	**5**	5

Table 5. Comparison of GA and HEA for the standard graphs with the upper bounds on the page numbers

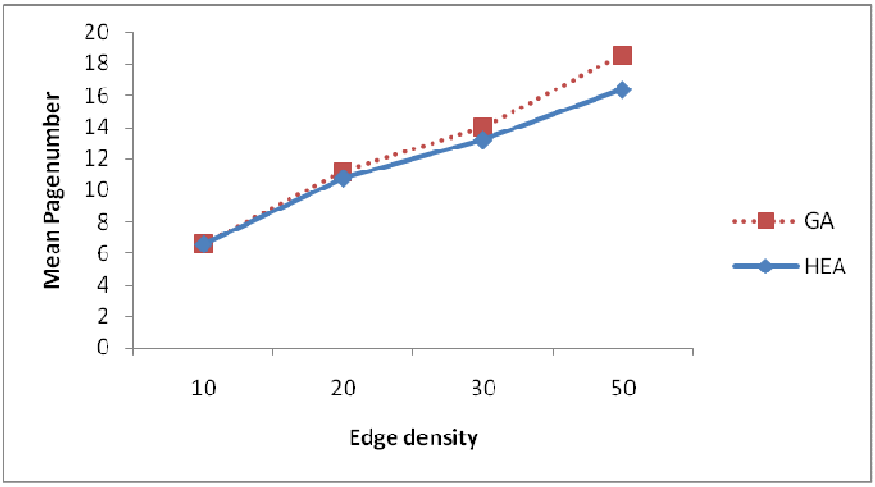
Graphs	GA	HEA	Upper Bound
CC_3	4	**3**	5
CC_4	7	**5**	5
$S(4)$	1	1	3
$S(5)$	2	2	3
$S(6)$	3	3	3
$S(7)$	3	3	3
T_4	3	3	4
T_5	3	3	5
T_6	3	**2**	6
T_7	4	**3**	7
T_8	4	4	8
T_9	4	4	9
T_{10}	4	4	10
T_{11}	5	5	11
$K_{4,4}$	3	4	3
$K_{5,5}$	4	5	4
$K_{6,6}$	5	6	5
$K_{7,7}$	6	7	5
$K_{8,8}$	7	8	6
$K_{9,9}$	8	9	7
$K_{10,10}$	9	10	7

Table 6. Comparison of GA and HEA for the random graphs ($|V|$=40)

Edge density (%)	Pagenumber		Elapsed Time (in sec.)	
	GA	HEA	GA	HEA
10	6.6	6.6	147.278	202.65
20	11.2	**10.8**	244.8094	347.581
30	14	**13.2**	391.4998	492.0376
50	18.6	**16.4**	736.2366	795.784

Table 7. Comparison of GA and HEA for the random graphs ($|V|$=50)

Edge density (%)	Pagenumber		Elapsed Time (in sec.)	
	GA	HEA	GA	HEA
10	9.4	**9.2**	220.4186	490.2936
20	14.8	**14**	455.9404	845.9158
30	19	**16.8**	1.0004e+003	1.20776e+003
50	24.2	**20.8**	1.86306e+003	1.9547e+003



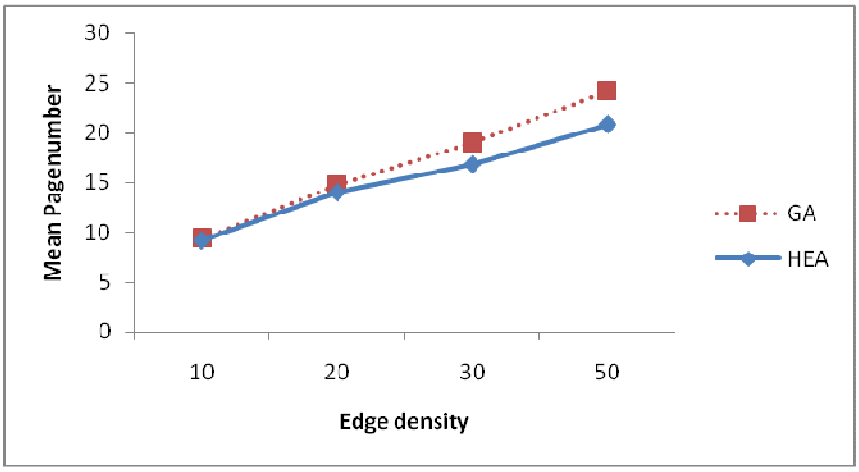


Figure 16. Edge density vs Pagenumber graph for HEA and GA of sizes (a) 40 and (b) 50

9 Conclusions

In this paper we have (a) statistically evaluated five heuristics for ordering vertices on the spine for minimum number of edge crossings with all the edges placed in a single page, (b) statistically evaluated four heuristics for distributing edges on a minimum number of pages with no crossings for a fixed ordering of vertices on the spine and (c) implemented and experimentally evaluated a hybrid evolutionary algorithm (HEA) for solving the page number problem.

The results described in Section 3 indicate that a depth first search (*dfs*) based technique for placing the vertices on the spine seems to be helpful in minimizing the number of crossings when edges are placed in a single page as it helps in generating vertex orderings in which there are edges between adjacent nodes.

The performance of edge distribution heuristics is presented in section 5. These experiments show that *circ* outperforms the other heuristics. Although, Chung et al. [3] have given an edge embedding strategy that provides the optimal page number for complete graphs results show that this strategy can be adapted to give good results for other graphs also.

We have presented a hybrid evolutionary algorithm (HEA) for the page number problem in which a *rdfs* is used to determine vertex ordering and *circ* is employed for edge distribution. Results show that the algorithm achieves the optimal page number for most of the standard graphs tested by us. HEA also outperforms the GA described by Kapoor et al. [1] in terms of solution quality for most of the instances tested.

Acknowledgement

This work is supported by University Grants Commission (UGC), New Delhi vide letter no 36-66/2008 (SR).

References

[1] Kapoor N, Russell M, Stojmenovic I, (2002) A Genetic Algorithm for finding the Page number of Interconnection Networks. *Journal of Parallel and Distributed Computing*, Vol. 62, pp. 267-283.

[2] Games RA, (1986) Optimal Book Embeddings of the FFT, Benes, and Barrel Shifter Networks. *Algorithmica*, Vol. 1, pp. 233–250.

[3] Chung FRK, (1987) Leighton FT and Rosenberg AL. Embedding Graphs In Books: A Layout Problem With Applications To VLSI Design. *Siam Journal on Algebraic and Discrete Methods*, Vol. 8, No. 1, pp. 33-58.

[4] Hasunuma T, Shibata Y, (1997) Embedding de Brujin, kautz and shuffle-exchange networks in books. *Discrete Applied Mathematics*, Vol. 78, pp. 103-116.

[5] Obrenic B, (1991) Embedding deBrujin and shuffle-exchange graphs in five pages. *SPAA'91 Proceedings of the third annual ACM symposium on Parallel algorithms and architectures*.

[6] Shahrokhi F, Shi W, (2000) On Crossing Sets, Disjoint Sets and the Page number. *Journal of Algorithms*, Vol. 34, pp. 40–53.

[7] Bernhart F, Kainen P C, (1979) The book thickness of a graph, *Journal of Combinatorial Theory*, Vol. 27, No. 3, pp. 320–331.

[8] Swaminathan RP, Giriraj D, Bhatia DK, (1995) The page number of the class of bandwidth-*k* graphs is *k*-1. *Information Processing Letters*, Vol. 55, pp. 71-74.

[9] Bettayeb S, Hoelzeman DA, (2009) Upper and Lower Bound on the Page number of the Book Embedding of the *k*-ary Hypercube. *Journal of Digital Information Management*, Vol. 7, No. 1, pp. 31-35.

[10] Cimikowski R, (2002) Algorithms for the Fixed Linear Crossing Number Problem. *Discrete Applied Mathematics*, Vol. 122, Issues 1-3,pp. 93-115.

[11] Kirkpatrick, S., Gelatt, C.D., Vechhi, M.P. (1983), Optimization by Simulated Annealing. *Science*, Vol. 220, No. 4598, pp. 671-680.

[12] Glover, F., laguna M., (1997) Tabu Search. Second ed., Kluwer, Boston.

[13] Percy, P.C., Yip, Pao, Yoh-Han. (1995) Combinatorial Optimization with use of Guided Evolutionary Simulated Annealing, *IEEE Transactions on Neural Networks*, Vol. 6, No. 2.