World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:4, No:7, 2010

Optimization of Unweighted Minimum Vertex Cover

S. Balaji, V. Swaminathan and K. Kannan

Abstract—The Minimum Vertex Cover (MVC) problem is a classic graph optimization NP - complete problem. In this paper a competent algorithm, called Vertex Support Algorithm (VSA), is designed to find the smallest vertex cover of a graph. The VSA is tested on a large number of random graphs and DIMACS benchmark graphs. Comparative study of this algorithm with the other existing methods has been carried out. Extensive simulation results show that the VSA can yield better solutions than other existing algorithms found in the literature for solving the minimum vertex cover problem.

Keywords—vertex cover, vertex support, approximation algorithms, NP - complete problem.

I. Introduction

THE classical minimum vertex cover problem involves graph theory and finite combinatorics and is categorized under the class of NP - complete problems in terms of its computational complexity. In 1972, in a landmark paper Karp has shown that the vertex cover problem is NP - complete [12], meaning that it is exceedingly unlikely that to find an algorithm with polynomial worst - case running time. The minimum vertex cover problem remains NP - complete even for certain restricted graphs, for example, the bounded degree graphs [10]. Minimum vertex cover has attracted researchers and practitioners not only because of the NP - completeness but also because of many difficult real - life problems which can be formulated as instances of the minimum vertex cover. Examples of such areas where the minimum vertex cover problem occurs in real world applications are communications, particularly in wireless telecommunications, civil, electrical engineering, especially in multiple sequence alignments for computational biochemistry [19].

Due to computational intractability of the MVC problem, many researchers have instead focused their attention on the design of approximation algorithm for delivering quality solutions in a reasonable time. Garey and Johnson [9] presented a simple approximation algorithm based on maximal matching gave an approximation ratio 2 for the general graphs. The first fixed parameter tractable algorithm for k - vertex problem (decision version: Given a graph G, deciding if G has a vertex cover of k vertices, k being the parameter), was done by Fellows [8]. Recently, Dehne et al [6] have reported that they used fixed parameter tractable algorithm to solve the minimum vertex cover problem on coarse-grained parallel machines successfully. Khuri et al [14] presented an evolutionary heuristic for the minimum vertex cover problem.For a comprehensive survey on the analysis of approximation algorithms for MVC, the reader is referred to Hochbaum [11], Monien and Speckenmeyer [16], Berman and Fujito [2], Tang et al [20], Shyu, Yin and Lin [18], Xu and Ma [23], Aggarwal et al[1], Bourjolly et al[4] Katayama et al[13]and Pullan[17].

S. Balaji is with the Department of Mathematics, SASTRA University, Thanjavur, India. e-mail: balaji_maths@yahoo.com.
V. Swaminathan is with the Ramanujan Research centre, Saraswathi Narayanan College, Madurai, India. e-mail: sulanesri@yahoo.com.
K. Kannan is with Department of Mathematics, SASTRA University, Thanjavur, India. e-mail: kkannan@maths.sastra.edu.

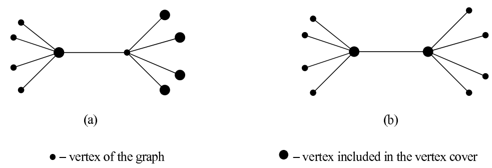

Fig. 1. (a) Possible vertex cover of G (b) Minimum vertex cover of G

In this paper for efficiently solving minimum vertex cover problem, a competent algorithm called Vertex Support Algorithm (VSA) is proposed. The proposed algorithm designed with the term called support of vertices, which involves the sum of the degrees of adjacency vertices, to get a near smallest vertex cover of the graph. Its effectiveness is shown by conducting extensive computational experiments on a large number of random graphs[1][23] and DIMACS benchmark graphs [7]. The simulation results show that the VSA can find the optimum solution.

The paper is organized as follows. Section 2 briefly describes the minimum vertex cover problem and its theoretical background. Section 3 outlines the VSA. In Section 4 graph models used in the experiments is briefly described. Section 5 provides experiments done and their results. Section 6 summarizes and concludes the paper.

II. Minimum Vertex Cover Problem

Let G = (V, E) be an undirected graph, a set S⊆V is a minimum vertex cover of G if (i) for every edge (u, v)∈E, either u∈S or v∈S or both u,v∈S and (ii) among all covers of E, S has the minimum cardinality, i.e., $\Sigma_{v\in S}v$ is minimum.

To illustrate the minimum vertex cover problem, consider the problem of placing guards with associated costs of guards [21] in a museum where corridors in the museum correspond to edges and task is to place a minimum number of guards so that there is at least one guard at the end of each corridor. Fig. 1 depicts the problem in brief.

Minimum vertex cover problem is a special case of set cover problem[5] which takes as input an arbitrary collection of subsets $S = \{S_1, S_2, ..., S_n\}$ of the universal set V, and the task is to find a smallest subsets from S whose union is V. The minimum vertex cover problem also closely related to many other hard graph problems and so it is of interest to the researchers in the field of design of optimization and approximation algorithms. For instance the independent set problem[12][10] is similar to the minimum vertex cover problem because a minimum vertex cover defines a maximum independent set(MIS) and vice versa. The MIS and MVC problems are related in that the maximum independent set contains all those vertices that are not in the minimum vertex cover of the graph. Another interesting problem that is closely related to the minimum vertex cover is the edge cover which seeks the smallest set of edges such that each vertex is included in one of the edges.

There are two versions of the vertex cover problem: the decision and optimization versions. In the decision version, the task is to verify for a given graph G whether there exists a vertex cover of a specified size but in the optimization version the task is to find a vertex cover of minimum size. In this paper we consider the optimization version of the minimum vertex cover with the goal of obtaining optimum solution. Now the minimum vertex cover problem is formulated as an integer programming problem by using the following conditions: Binary variables a_{ij}(i = 1,2,3,...,n; j = 1,2,3,...,n) form the adjacency matrix of the graph G. Each variable has only two values (1 or 0) according as an edge exists or not. In other words, if an edge (v_i,v_j) is in E, then a_{ij} is 1 else a_{ij} is 0. For example the graph of Fig.1 has the following adjacency matrix

$$A = \begin{bmatrix} 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 1&1&1&1&0&1&0&0&0&0 \\ 0&0&0&0&1&0&1&1&1&1 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \end{bmatrix}$$

The output of the program expresses the vertex v_i is in the vertex cover or not. v_i=1 if it is in the vertex cover otherwise v_i=0. Thus the total number of vertices in the vertex cover can be expressed by Z = $\Sigma v_i,\ 1 \leq i \leq n$. At least one vertex of the edge (v_i, v_j) must be included in the vertex cover, so we have the constrained condition of the minimum vertex cover can be written as $v_i + v_j \geq$1. Thus the problem can be mathematically transformed into the following optimization problem as

Min Z = Σv_i

Subject to

$v_i + v_j \geq 1\ \forall (v_i, v_j) \in$E

$v_i \in \{0, 1\}\ \forall v_i \in$V

III. Terminologies, Algorithm and Computational complexity

Neighborhood of a vertex: Let G = (V, E), V is a vertex set and E is an edge set, be an undirected graph and let $|V|$ = n and $|E|$ = m. Then for each v∈V, the neighborhood of v is defined by $N(v) = \{u \in V/u \ is\ adjacent\ to\ v\}$ and N[v] = v ∪ N(v).

Degree of a vertex: The degree of a vertex v∈V, denoted by d(v) and is defined by the number of neighbors of v.

Support of a vertex: The support of a vertex v∈V is defined by the sum of the degree of the vertices which are adjacent to v, i.e., support(v) = s(v) = $\Sigma_{u\in N(v)} d_G(u)$.

A. *Vertex Support Algorithm (VSA)- Proposed*

The following algorithm is designed to find the general minimum vertex cover of a graph G. Adjacency matrix (a_{ij}) of the given graph G of n vertices and m edges are given as the input of the program. The degree d(v) and support s(v) of each vertex v∈V are calculated. Support of the vertex calculated by the relation $\Sigma_{u\in N(v)} d_G(u)$. Add the vertex which has the maximum value of s(v) into the vertex cover V_c . If one or more vertices have equal maximum value of the s(v), in this case if (d(v_i)≥ d(v_j)), add the vertex v_i into the vertex cover V_c otherwise add v_j into V_c. Update the adjacency matrix of G by putting zero in to the row and column entries of the corresponding vertex $v \in V_c$. Proceed the above process until the edge set E has no edges. i.e., up to $a_{ij} \neq 0\ \forall i, j$. The pseudo-code of the proposed algorithm is given below.

Input: G (V, E)
Output: Z = $\Sigma_{v_i \in V_c} v_i$
while E $\neq \phi$ do
step 1:
for i←1 to n
for j ←1 to n
$d_G(v_i) = \Sigma_j a_{ij} = d(v_i)$
step 2:
for i←1 to n
for j ←1 to n
$s_G(v_i) = \Sigma_{v_j \in N(v_i)} d_G(v_j) = s(v_i)$
step 3:
$max = s(v_1)$;
k = 1;
select the vertex which has the maximum value of s(v) in to V_c
for i←2 to n
if$(max < s(v_i))$
$max = s(v_i)$;
t = i;
$V_c \leftarrow V_c \cup v_i$
end **if**
if multiple vertices have equal maximum value of s(v) then follow step 3a
step 3a:
if$((max = s(v_i) \& (d(v_{i-k}) <= d(v_i)))$
$max = s(v_i)$;

t = i;
$V_c \leftarrow V_c \cup v_i$
end **if**
if$((max = s(v_i)\&(d(v_{i-k}) > d(v_i)))$
$max = s(v_{i-k})$;
t = i-k;
$V_c \leftarrow V_c \cup v_{i-k}$
end **if**
k = k+1;
end **for**
step 4:
for i←1 to n
$(a_{ti}) = 0$;
$(a_{it}) = 0$;
end **for**
end **while**.
for i←1 to n
if$(v_i \in V_c)$
$v_i = 1$;
else
$v_i = 0$;
end **for**
end

B. Computational Complexity

The worst case complexity of finding the solution of the minimum vertex cover problem using VSA can be obtained as follows: Assume that there are n vertices and m edges, in the proposed algorithm, calculation of degree of vertices in step 1 and support of vertices in step 2 requires O(n^2)and O(n^2) running time respectively. To pick the vertex which has the maximum value of s(v) in step 3 requires O($n-1$) running time. The procedure of the algorithm goes up to m steps (worst case). So the overall running time of the procedure of SRA can be deduced as follows: m(O(n^2)+ O(n^2)+ O($n-1$)) = O($mn^2 + mn^2 + m(n-1)$) = O(mn^2).

IV. Graph Models

This section outlines the graph models used to assess the effectiveness of the proposed algorithm in previous section. The graph models used are (i) G(n, p) graphs[3] and (ii) G(n, m) graphs[3][22]. The models are standard random graph models from the graph theory and all the graphs are undirected.

A. G(n, p) Model

The G(n, p) model is also called Erdos Renyi random graph model[3], consists of graphs of n vertices for which the probability of an edge between any pair of nodes is given by a constant $p > 0$. To ensure that graphs are almost always connected, p is chosen so that $p >> \frac{log(n)}{n}$. To generate a G(n, p) graph we start with an empty graph. Then we iterate through all pairs of nodes and connect each of these pairs with probability p.

1) Algorithm to generate (G, n, p)graphs: The pseudo code for generating G(n, p) graphs as follows
initialize graph G(V, E)
for i ← 1 to n
for j← i+1 to n
add edge (i, j) to E with probability p
return (G).

The expected number of edges of G(n, p) graph is $pn(n-1)/2$ and expected degree is np. Graphs are generated for different p and n values.

B. G(n, m) Model

The G(n, m) model consists of all graphs with n vertices and m edges. The number of vertices n and the number of edges m are related by m = nc, where c>0 is constant. To generate a random G(n, m) graph, we start with a graph with no edges. Then, cn edges are generated randomly using uniform distribution over all possible graphs with cn edges. Each node is thus expected to connect to 2c other nodes on average. The pseudo-code for the random graph generation is shown in the following algorithm.

1) Algorithm to generate (G, n, c)graphs: The pseudo code for generating G(n, m) graphs as follows
initialize graph G(V, E)
$m \leftarrow n * c$
for i ← 1 to m
repeat
e ← random edge
until e not present in E
E ← E $\cup \{e\}$
return (G).

V. Experimental Results and analysis

All the procedures of VSA have been coded in C++ language. The experiments were carried out on an Intel Pentium Core2 Duo 1.6 GHz CPU and 1 GB of RAM. The effectiveness of the VSA heuristic was evaluated using 136 instances. These instances are divided into 3 sets as shown in the TABLE I. Simulations are carried out on three types of graphs: the randomly generated small size, moderate and large scale graphs for the minimum vertex cover problem.

TABLE I
MVC Instances

Problem set	No. of Instances	Scale	Graph Model	Optimal Solution
1	36	small-large	G(n, p)	Unknown
2	80	small-large	DIMACS	Known
3	20	moderate	G(n, m)	Unknown

A. Results for random graphs

We first tested the VSA on 36 random graphs generated based on the concept explained in Section 4.1. The result we recorded for each test graph and their information are shown in the TABLE II and these results are compared with the theoretical evaluation of expected MIS(MVC) for G(n, p) random graphs, shown in [15], and it is guaranteed that the proposed algorithm estimations are quite well to the expected size of the minimum vertex cover. In the 36 instances tested the maximum time taken of 29 seconds, (3000, 0.8; 4000, 0.9 & 5000, 0.8), is an encouraging one but also it is comparatively very less time for finding the MVC of random graphs of large number of vertices with high density. So, it is interest to see the performance of the proposed algorithm on benchmark graphs with known optimal (best known) solutions.

TABLE II
SIMULATION RESULTS FOR THE 1 SET OF INSTANCES

Graph		VSA		Graph		VSA	
n	p	V_c	Time(s)	n	p	V_c	Time(s)
100	0.7	85	<1	700	0.7	654	6
	0.8	80	<1		0.8	648	3
	0.9	69	<1		0.9	628	12
150	0.8	127	<1	1000	0.7	917	17
	0.9	113	3		0.8	893	8
	0.95	96	2		0.9	888	28
200	0.7	181	<1	2000	0.7	1871	23
	0.8	174	3		0.8	1858	18
	0.9	157	5		0.9	1841	27
300	0.7	279	2	3000	0.7	2857	15
	0.8	271	<1		0.8	2833	29
	0.9	249	5		0.9	2811	17
400	0.7	377	<1	4000	0.7	3827	28
	0.8	369	2		0.8	3794	27
	0.9	347	4		0.9	3764	29
500	0.7	468	<1	5000	0.7	4773	23
	0.8	459	5		0.8	4751	29
	0.9	441	3		0.9	4717	24

B. Results for DIMACS benchmark graphs

To test the performance of VSA approach, further we have tested the proposed algorithm on benchmark graphs with known results, they have been extracted from DIMACS[7] challenge suite. That suite structured from the perspective of finding maximum cliques, so we considered the benchmark graphs as $\overline{G}$. We compare the heuristic performance with implementation of the algorithms KLS[13], OCH[1] and the results were shown in the TABLES III & IV. The first two columns reports the type of the instances such as name, cardinality of the instances; the third gives the best results obtained in the challenge, the forth,fifth and sixth gives the minimum vertex cover found by corresponding algorithms. Sixth column reports the optimality achieved by proposed algorithm, in which * indicates the instances where proposed algorithm fail to reach the optimality, mostly in MANN type of instances. In TABLES V and VI, we listed the CPU time (in seconds) and success rate to find the MVC of the DIMACS instances. TABLES III, IV, V & VI shows that proposed algorithm could find the optimal solution for most of the DIMACS benchmark graphs i.e., out of 80 instances tested the proposed algorithm reaches the optimum value for 73 instances.

TABLE III
SIMULATION RESULTS FOR DIMACS BENCHMARK GRAPHS

$\overline{G}$	$\|V\|$	Optimum V_c	KLS V_c	OCH V_c	VSA V_c
brock200_1	200	179	181	-	179
brock200_2	200	188	190	188	188
brock200_3	200	185	187	-	185
brock200_4	200	183	186	183	183
brock400_1	400	373	380	373	373
brock400_2	400	371	377	371	371
brock400_3	400	369	377	369	369
brock400_4	400	367	377	367	367
brock800_1	800	777	777	780	777
brock800_2	800	776	776	776	776
brock800_3	800	775	775	775	775
brock800_4	800	774	774	777	774
C125.9	125	91	-	91	91
C250.9	250	206	-	206	206
C500.9	500	$\leq$443	-	443	443
C1000.9	1000	$\leq$932	-	932	932
C2000.5	2000	$\leq$1984	-	1984	1984
C2000.9	2000	$\leq$1923	-	1923	1923
C4000.5	4000	$\leq$3982	-	-	3982
c-fat200-1	200	188	188	-	188
c-fat200-2	200	176	176	-	176
c-fat200-5	200	142	-	-	144*
c-fat500-1	500	486	486	-	486
c-fat500-2	500	474	474	-	474
c-fat500-5	500	446	448	-	446
c-fat500-10	500	126	127	126	126
DSJC500.5	500	$\leq$487	487	487	487
DSJC1000.5	1000	$\leq$985	985	985	985
gen200_p0.9_44	200	156	-	156	156
gen200_p0.9_55	200	145	-	145	145
gen400_p0.9_55	400	345	-	347	345
gen400_p0.9_65	400	335	-	335	335
gen400_p0.9_75	400	325	-	325	325
Hamming6-2	64	32	30	32	32
Hamming6-4	64	60	60	60	60
Hamming8-2	256	128	128	128	128
Hamming8-4	256	240	240	240	240
Hamming10-2	1024	512	512	512	512
Hamming10-4	1024	984	-	984	984
Johnson8-2-4	28	24	24	24	24

Since we know the optimal solution value for each instance we tested, we can measure the quality of the solution derived by an algorithm by computing ratio between them. That is, we define the quality measure ratio as value/optimum, where value is the value of a solution found by an algorithm and optimum is the optimal solution value. We note that smaller the ratio indicates that the performance of an algorithm is guaranteed one. In TABLE VII we sum up the information concerning the ratios.

C. Results for G(n, m) random graphs

In this experiment the parameter set opted like small-large scale problems, that is V varied from 50 to 1000. Here we used

TABLE IV
SIMULATION RESULTS FOR DIMACS BENCHMARK GRAPHS

$\overline{G}$	$\|V\|$	Optimum V_c	KLS V_c	OCH V_c	VSA V_c
Johnson8-4-4	70	56	56	56	56
Johnson16-2-4	120	112	112	112	112
Johnson32-2-4	496	480	-	481	480
keller4	171	160	164	160	160
keller5	776	749	750	749	749
keller6	3361	≤3302	-	3303	3307*
MANN_a9	45	29	29	29	29
MANN_a27	378	252	261	258	253*
MANN_a45	1035	690	-	697	692*
MANN_a81	3321	≤2221	-	2228	2237*
p_hat300-1	300	292	292	292	292
p_hat300-2	300	275	275	275	275
p_hat300-3	300	274	274	274	274
p_hat500-1	500	491	491	491	491
p_hat500-2	500	464	464	464	464
p_hat500-3	500	450	453	453	450
p_hat700-1	700	689	693	689	689
p_hat700-2	700	656	656	657	656
p_hat700-3	700	638	641	640	639*
p_hat1000-1	1000	900	900	900	900
p_hat1000-2	1000	954	956	955	954
p_hat1000-3	1000	934	938	937	935*
p_hat1500-1	1500	1488	1490	1488	1488
p_hat1500-1	1500	≤1435	1436	1436	1435
p_hat1500-1	1500	≤1406	1409	1409	1406
san200-0.7.1	200	170	185	170	170
san200-0.7.2	200	182	188	188	188
san200-0.9.1	200	130	155	135	130
san200-0.9.2	200	140	161	143	140
san200-0.9.3	200	156	169	156	156
san400-0.5.1	400	387	393	387	387
san400-0.7.1	400	360	380	360	360
san400-0.7.2	400	370	385	370	370
san400-0.7.3	400	378	388	378	378
san400-0.9.1	400	300	350	304	300
san1000	1000	900	-	900	900
sanr200-0.7	200	282	284	282	282
sanr200-0.9	200	158	159	158	158
sanr400-0.5	400	387	387	387	387
sanr400-0.7	400	379	379	379	379

the G(n, m) graph model to generate the random graphs. For most of the test instances the optimal solutions are unknown, we obtained the time (in sec.) taken by the VSA for finding the minimum vertex cover of the graph. These results are shown in the Fig. 2 where the major axis represents the size (in terms of number of vertices) of the 20 test instance's and for each test instances the time taken by VSA were plotted as points and for each instances their points are linked by a line. It is clear from the Fig. 2 that the time taken by the VSA to find the optimum value of each of the MVC instances increases steadily when the size of the problem increases and the maximum time taken is 7.41 sec. With this figure we show that the proposed algorithm took very less time to produce a minimum vertex cover for each of the test instances of G(n, m) graph model also.

TABLE V
TIME TAKEN (SEC.) AND SUCCESS RATE FOR DIMACS INSATNCES

$\overline{G}$	Density	Time(s)	Success(%)
brock200_1	0.745	<1	100
brock200_2	0.496	<1	100
brock200_3	0.605	<1	100
brock200_4	0.658	<1	100
brock400_1	0.748	<1	100
brock400_2	0.749	<1	100
brock400_3	0.748	<1	100
brock400_4	0.749	<1	100
brock800_1	0.649	2	100
brock800_2	0.651	2	100
brock800_3	0.649	5	100
brock800_4	0.65	4	100
C125.9	0.898	<1	100
C250.9	0.899	<1	100
C500.9	0.9	6	100
C1000.9	0.901	13	100
C2000.5	0.5	18	100
C2000.9	0.9	26	100
C4000.5	0.5	30	100
c-fat200-1	0.077	<1	100
c-fat200-2	0.163	<1	100
c-fat200-5	0.426	<1	96
c-fat500-1	0.036	<1	100
c-fat500-2	0.073	<1	100
c-fat500-5	0.186	<1	100
c-fat500-10	0.374	<1	100
DSJC500.5	0.5	13	100
DSJC1000.5	0.5	20	100
gen200_p0.9_44	0.9	<1	100
gen200_p0.9_55	0.9	<1	100
gen400_p0.9_55	0.9	6	100
gen400_p0.9_65	0.9	9	100
gen400_p0.9_75	0.9	8	100
Hamming6-2	0.905	<1	100
Hamming6-4	0.349	<1	100
Hamming8-2	0.969	3	100
Hamming8-4	0.639	2	100
Hamming10-2	0.99	9	100
Hamming10-4	0.829	23	100
Johnson8-2-4	0.556	<1	100

VI. CONCLUSION

A new VSA for MVC of graphs using vertex cover has been proposed and its effectiveness has been shown by simulation experiments. The terminology support of a vertex introduced in the new model, with that, the new model can find the minimum vertex cover effectively. Experimental result shows that this approach greatly reduce the execution time and in addition, the simulation results show that the new VSA can yield better solutions than KLS and OCH heuristics found in the literature. At the same time, our approach gives best solutions for DIMACS benchmark graph instances and also for random graphs. The proposed algorithm has led to give near optimal solutions for most of the test instances where we know the optimal solutions. Furthermore attractiveness of this heuristic is its outstanding performance in the optimization of MVC.

TABLE VI
TIME TAKEN (SEC.) AND SUCCESS RATE FOR DIMACS INSTANCES

$\overline{G}$	Density	Time(s)	Success(%)
keller6	0.818	25	91
MANN_a9	0.927	<1	100
MANN_a27	0.99	12	99
MANN_a45	0.996	33	99
MANN_a81	0.999	43	98
p_hat300-1	0.244	<1	100
p_hat300-2	0.489	<1	100
p_hat300-3	0.744	<1	100
p_hat500-1	0.253	2	100
p_hat500-2	0.505	5	100
p_hat500-3	0.752	3	100
p_hat700-1	0.249	<1	100
p_hat700-2	0.498	15	100
p_hat700-3	0.748	18	98
p_hat1000-1	0.245	8	100
p_hat1000-2	0.49	23	100
p_hat1000-3	0.744	30	98
p_hat1500-1	0.253	23	100
p_hat1500-1	0.506	26	100
p_hat1500-1	0.754	24	100
san200-0.7.1	0.7	<1	100
san200-0.7.2	0.7	<1	100
san200-0.9.1	0.9	5	100
san200-0.9.2	0.9	17	100
san200-0.9.3	0.9	23	100
san400-0.5.1	0.5	6	100
san400-0.7.1	0.7	<1	100
san400-0.7.2	0.7	<1	100
san400-0.7.3	0.7	19	100
san400-0.9.1	0.9	8	100
san1000	0.502	<1	100
sanr200-0.7	0.697	<1	100
sanr200-0.9	0.898	<1	100
sanr400-0.5	0.501	<1	100
sanr400-0.7	0.7	<1	100

TABLE VII
AVERAGES AND STANDARD DEVIATIONS OF THE RATIO VALUES

Algorithm	Min.	Average	Max.	Std. Dev.
VSA	1.00	1.06	1.18	0.06
OCH	1.00	1.26	1.45	0.13
KLS	1.15	1.40	1.70	0.17

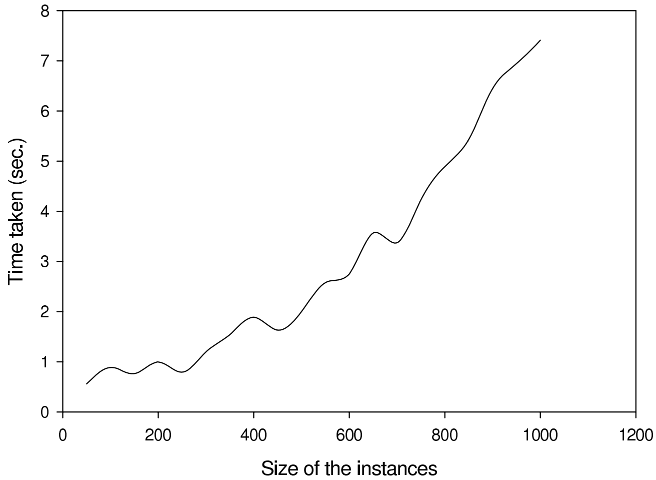

Fig. 2. Time taken (in sec.) by VSA for 3rd set of instances

REFERENCES

[1] C. Aggarwal , J. B Orlin and R. P Tai, *Optimized cross cover for the independent set problem*, Operations Research, Vol. 45, (1997), 226-234.
[2] P. Berman and T. Fujito, *On approximation properties of the independent set problem for low degree graphs*, Theory of Computing Syst., Vol. 32, (1999), 115 - 132.
[3] B. Bollobas, *Random graphs*, 2nd Ed., Cambridge, UK: Cambridge University press (2001).
[4] J. M. Bourjolly , P. Gill , G. Laporte and H. Mercure, *An exact quadratic 0-1 algorithm for the stable set problem*, American Mathematical Society Providence, RI. 1996, pp. 53-73.
[5] T. H. C. E. Cormen, R. L. R. Leiserson, and C. Stein, *Introduction to algorithms*, 2nd ed., McGraw - Hill, New York (2001).
[6] F. Dehne et. al, *Solving large FPT problems on coarse grained parallel machines*, Available: http://www.scs.carleton.ca/fpt/papers/index.htm.
[7] *DIMACS clique benchmarks*, Benchmark instances made available by electronic transfer at dimacs.rutgers.edu, Rutgers Univ., Piscataway. NJ. (1993).
[8] M. R. Fellows, *On the complexity of vertex cover problems*, Technical report, Computer science department, University of New Mexico (1988).
[9] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the theory NP - completeness*, San Francisco: Freeman (1979).
[10] M. R. Garey, D. S. Johnson and L. Stock Meyer, *Some simplified NP - complete graph problems*, Theoretical computer science, Vol. 1563, (1999), 561 - 570.
[11] D. S. Hochbaum *Efficient bounds for the stable set, vertex cover and set packing problems*, Discrete Appl. Mathematics, Vol. 6, (1983), 243 - 254.
[12] R. M. Karp, *Reducibility among combinatorial problems*, Plenum Press, New York, (1972), pp 85 - 103.
[13] K, Katayama, A. Hamamoto and H. Narihisa, *An effective local search for the maximum clique problem*, Information Processing Letters, Vol. 95, (2005), 503-511.
[14] S. Khuri and T. Back, *An evolutionary heuristic for the minimum vertex cover problem*, J. Kunze and H. Stoyan, editors, KI - 94 workshops (Extended Abstracts), Bonn (1994), pp. 83 - 84..
[15] D. Matula, *On the complete subgraph of a random graph*, Combinatory mathematics and its Applications, (1970), pp. 356-369.
[16] B. Monien and E. Speckenmeyer *Ramsey numbers and an approximation algorithm for the vertex cover problems*, Acta Informatica, Vol. 22, (1985), 115 - 123.
[17] W. Pullan, *Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers*, Discrete Optimization, Vol. 6, (2009), 214-219.
[18] S.J. Shyu, P.Y. Yin and B.M.T. Lin, *An ant colony optimization algorithm for the minimum weight vertex cover problem*, Annals of Operations Research, Vol. 131, (2004), 283 - 304.
[19] U. Stege, *Resolving conflicts from problems in computational Biology*, Ph.D thesis, No.13364, ETH Zurich (2000).
[20] C. Z. Tang, X. Xu et al., *An algorithm based on Hopfield network learning for minimum vertex cover problem*, Lecture Notes in computer science, Vol. 3173, (2004), 430 - 435,.
[21] M. Weight and A. K. Hartmann, *The number of guards needed by a museum - a phase transition in vertex covering of random graphs*, Phys - Rev. Lett., 84, 6118 (2000b).
[22] M. Weight and A. K. Hartmann, *Minimal vertex covers on finite-connectivity random graphs - A hard-sphere lattice-gas picture*, Phys. Rev. E, 63, 056127.
[23] X. Xu and J. Ma, *An efficient simulated annealing algorithm for the minimum vertex cover problem*, Nerocomputing, Vol. 69, Issues 7-9, (2006), 613 - 616.