A Genetic Algorithm for Maximum Independent Set Problems

Xingzhao LIU[†] liuyt@hitnet.hit.edu.cn
Akio SAKAMOTO[††] sakamoto@ee.tokushima-u.ac.jp
Takashi SHIMAMOTO[††] simamoto@ee.tokushima-u.ac.jp

[†]Institute of Electronic Eng. and Technology, Harbin Institute of Technology, P. R. China.
[††]Dept. of Electrical and Electronic Eng., Faculty of Eng., The University of Tokushima, Japan.

ABSTRACT

Genetic algorithms have been shown to be very useful in a variety of search and optimization problems. In this paper we present a genetic algorithm for maximum independent set problem. We adopt a permutation encoding with a greedy decoding to solve the problem. The well known DIMACS benchmark graphs are used to test our algorithm. For most graphs solutions found by our algorithm are optimal, and there are also a few exceptions that solutions found by the algorithm are almost as large as maximum clique sizes. We also compare our algorithm with a hybrid genetic algorithm, called GMCA, and one of the best existing maximum clique algorithms, called CBH. The experimental results show that our algorithm outperformed two of the best approaches by GMCA and CBH in not only final solutions, but also computation time.

keywords: *genetic algorithm, maximum independent set problem, maximum clique problem, heuristic algorithms.*

1. INTRODUCTION

Consider an undirected graph $G = (V, E)$ where V is the set of vertices and E is the set of edges in G. An *independent set* is a set of vertices of G so that no two vertices of the set are adjacent; that is, no two vertices are joined by an edge of G. An independent set is said to be *maximal* if the set is not a proper subset of any other independent set. Furthermore, an independent set is said to be *maximum* if there is no independent set with a greater number of vertices. The opposite concept of an independent set is a *clique*. That is, a clique of G is a subgraph induced by a set S of vertices which is complete. The maximum independent set problem (MISP) is to find the number of vertices in a maximum independent set for a given graph. In a way analogous to the definition of MISP, we can also define the maximum clique problem (MCP). It is quite obvious that an independent set of a graph G corresponds to a clique of the graph $\overline{G}$ and vice versa, where $\overline{G}$ is the graph complementary to G.

MISP and MCP have a variety of applications such as coding theory, geometry, VLSI design, classification theory, etc. They are computationally very difficult and NP-hard [1]. Pardalos and Xue [2] gave a complete overview on MCP and related problems. There are some algorithms to solve it exactly [1]. For example, Friden *et al.* [3] proposed the algorithm based on an implicit enumeration method combined with tabu search techniques and it did achieve some good results on random graphs. Besides these exact algorithms, there are also some algorithms based on heuristics, for example, Kubo *et al.* [4] presented an approximate algorithm for MISP by using life span method.

Recently genetic algorithms have been applied to a variety of hard combinatorial optimization problems, however, little work has been done on MCP and MISP. The reported results of genetic approach by Murthy *et al.* [5] are limited to rather small graphs so that it is difficult to compare with traditional algorithms for MCP. A hybrid genetic algorithm, called GMCA, proposed by Bui and Eppley [6] was successfully applied to MCP, and it did get some good results on DIMACS benchmarks, which arise from various practical applications with known maximum clique size. GMCA can be comparable to one of the best existing max-

0-7803-3280-6/96/$5.00

imum clique algorithms, called CBH, provided by Gibbons *et al.* [7].

In this paper we present a genetic algorithm for MISP. We adopt a permutation encoding with a greedy decoding to solve the problem, whereas GMCA adopted a direct encoding method, that is, the loci correspond to index number of vertices in a graph and alleles are limited to either 1 or 0. The DIMACS benchmark graphs are used to test our algorithm. For most graphs solutions found by our algorithm are optimal, and there are also a few exceptions that solutions found by the algorithm are almost as large as maximum clique sizes. We also compare our algorithm with GMCA and CBH. The experimental results show that our algorithm outperformed two of the best approaches by GMCA and CBH in not only final solutions, but also computation time.

2. GENETIC ALGORITHMS FOR MISP

Genetic algorithms (GAs) are search algorithms based on the mechanics of natural selection and genetics [8].

Outline of Algorithms

We adopt the following outline of GA, where $P(t)$ is the population at generation t. The population is assumed to be of constant size M, that is, there always exist M chromosomes. We consider a solution of a chromosome α as the size of independent set derived from α. If the value of the best solution has not improved in T_{stop} generations, where T_{stop} is the previously prescribed number, then the procedure stops.

```
procedure GeneticAlgorithms
{
   initialize P(0);
   bestSol = 0;
   count = 0;
   for (t = 0; count < T_stop; t++) {
      evaluate structures in P(t);
      Sol = maximal solution in P(t);
      if (Sol > bestSol) {
         count = 0;
         bestSol = Sol;
      }
      count++;
      select P(t+1) from P(t);
      recombine structures in P(t+1);
   }
}
```

Initialization is to select M chromosomes randomly. *Evaluation* is to evaluate the function value of chromosomes in the population. *Selection* is the process of choosing M chromosomes for the next generation from those in the current generation according to the evaluated fitness such that the expected value of a chromosome to survive in the next generation is proportional to the fitness of its own. *Recombination* typically includes crossover and mutation operators.

Permutation Encoding and Greedy Decoding

The basic idea of our coding method is "permutation encoding with greedy decoding," which is so-called Davis's encoding method, described in [9]. This method encodes a feasible solution as a permutation of the objects concerned and decodes the permutation with a greedy adding heuristics. In MISP the objects are the vertices of G and the greedy heuristics tries to decide all possible vertices which belong to the independent set according to the permutation. Thus a chromosome in a population is a permutation of n vertices, where n is the number of vertices in G.

We denote a permutation of size n, say α, as $\alpha = (\alpha(1), \alpha(2), \cdots, \alpha(n))$, where each $\alpha(k), 1 \leqq k \leqq n$, is a vertex in V.

Since a chromosome denotes an order for vertices to decide which one is within the independent set, the greedy decoding algorithm mainly refers to adjacency relation of vertices to exclude the rest vertices which does not belong to the independent set. This greedy decoding algorithm is like the following function Decode which returns the square of number of vertices of the independent set derived from the chromosome α.

```
function Decode(α)
{
   for (v ∈ V)
      value[v] = -1;
   for (num = 0, i = 1; i <= n; i++) {
      v = α(i);
      if (value[v] == -1) {
         value[v] = 1;
```

```
            num++;
            for (u ∈ adj[v])
               value[u] = 0;
         }
      }
      α = Rearrange(α, value);
      return num;
   }
```

Where a vertex v belongs to independent set if `value[`v`]` equals 1, and it does not if `value[`v`]` is 0, hence, array `value` is initialized to be -1 at first. The set `adj[`v`]` contains vertices that are adjacent to vertex v. It is easily verified that the independent set obtained by this algorithm is always maximal.

```
function Rearrage(α, value)
{
   for (i = j = 1; i <= n; i++)
      if (value[i] == 1)
         β[j++] = α[i];
   for (i = 1; i <= n; i++)
      if (value[i] == 0)
         β[j++] = α[i];
   return β;
}
```

The function `Rearrange(`α`, value)` is used to reorder the vertices in α based on the independent set denoted by `value`. At first vertices belonging to the independent set will be placed one by one from the beginning of rearranged permutation in the order of original α. Next the remaining positions of the permutation will be filled with the vertices which do not belong to the independent set. We expect that the introduction of `Rearrange(`α`, value)` will enhance search ability for the algorithm.

The fitness function form is defined as:

$$f(\alpha) = \texttt{Decode}(\alpha)^2 - D_{min}^2 + 1,$$

where `Decode(`α`)` stands for the number of vertices in derived independent set from chromosome α and D_{min} is the minimum value of `Decode(`β`)` among all chromosomes β at generation `t`. The motivation for adoption of $f(\alpha)$ focuses on difference among chromosomes and we expect the better chromosomes can survive as much as possible by selection process. Hence it will lead a fast convergence on final solution and can also avoid premature convergence.

Crossover and Mutation Operators

After the selection process, all chromosomes are paired, and then *crossover* operator is applied with probability p_c to each pair of chromosomes. The intuition behind the applicability of the crossover operator is information exchange between different potential solutions. When the crossover takes place, these offspring will enter the next step for mutation. Otherwise, the pair of chromosomes will get into next step for mutation directly.

Several crossover operators are intentionally proposed in permutation encoding to solve the traveling salesman problem. For example, a study of three permutation crossover operators, order crossover, partially mapped crossover and cycle crossover, appeared in [10]. In order to find a suitable crossover operators for our problem, not only its original forms of order and partially mapped crossover operators, but also its extension forms are considered in our approach. Its description is neglected here due to space limit, please refer to [11] for details.

Mutation arbitrarily alters one or more genes of a selected chromosome, by a random change with a probability p_m. The intuition behind the mutation operator is the introduction of some extra variability into the population. Here, the mechanics of mutation is the same as that in [11].

3. EXPERIMENTAL RESULTS

To illustrate the validity of the algorithm described in the previous sections, the algorithm is coded as C language on Sun SPARC LX station.

The DIMACS Benchmark Graphs

The DIMACS benchmark graphs are originally considered for MCP. These graphs come from the Second DIMACS Implementation Challenge at the NSF Science and Technology Center in Discrete Mathematics and Theoretical Computer Science in 1993. The benchmark graphs consist of random graphs with known maximum clique size as well as graphs arise from various areas of applications such as coding theory, geometry, and fault diagnosis. There are 9 different classes of graphs and a total of 66 graphs

ranging in size from 28 vertices to 3,361 vertices and up to 5,506,380 edges.

Results and Discussion

In order to do a complete comparison with CBH and GMCA, we choose graphs in the DIMACS as CBH and GMCA did. The evaluation of our simulation results is complicated by the fact that we are dealing with *randomized* algorithms, that is, algorithms do not always yield the same answer on the same graph due to seeds of random numbers. Moreover, results can differ substantially from run to run, making comparisons between algorithms less straightforward. For example, in contrast, CBH is a deterministic algorithm, which has only one trial available for each graph. Therefore we always try ten trials for each graph and report best answer as well as average running time in second, which is also the same way adopted by [6].

Table 1 is the experimental results for some instances in the DIMACS to compare performance of the algorithm with and without function `Rearrange(`α`, value)`, where $M = 50$, $p_c = 0.8$, $p_m = 0.7$, $T_{stop} = 50$, and two-point-inside partially mapped crossover are adopted. The detail description of crossover operators is in [11]. In Table 1, method-1 and method-2 stand for the algorithm with and without function `Rearrange(`α`, value)`, respectively. Numbers at columns Sol are maximum clique sizes obtained by the algorithms, where bold numbers show obtained clique sizes equal to optimal ones designated at column Opt. It is obvious that performance of method-1 is much better than that of method-2, since optimal solutions can be obtained for every instances by method-1, while only near optimal solutions obtained by method-2. Therefore we adopt method-1 at the following experiments.

Table 1: Experiments for `Rearrange`.

Data	Opt.	method-1		method-2	
		Sol.	Time	Sol.	Time
san400-0.5-1	13	**13**	43.1	9	33.7
sanr400-0.5	13	**13**	45.8	12	32.7
p-hat700-1	11	**11**	117.7	9	91.0
p-hat1000-1	10	**10**	247.6	9	183.2

Tables 2 is the experimental results with the same parameters as Table 1 for DIMACS. We also quote the results of GMCA [6] and CBH [7], where the running time of CBH has been converted to equivalent ones run under Sun SPARC LX station. Moreover the running time of our algorithm includes time of converting original graphs to its complementary graphs and calculating its maximum independent set. Therefore the comparison with GMCA and CBH can be considered to be fair.

In Table 2 bold numbers corresponding to columns Sol show that answers obtained are the same as that of optimal solutions. Since only 24 graphs in DIMACS were tested by GMCA, the corresponding space for the rest graphs has to be remained blank. *DNR* in column Sol of CBH means that the corresponding graphs were not run because it required too much memory. We found the bold numbers for our algorithm and CBH were 36 and 25, respectively, out of 54 graphs whose optimal solutions are known. If corresponding to 24 graphs tested by GMCA, then these bold numbers will be 18 for our algorithm, 10 for GMCA, and 11 for CBH. In general, our algorithm did very well on the **c-fat**, **johnson**, **keller**, **hamming** and **sanr** graphs, where it can find maximum clique sets for almost every graphs, especially for **keller5** the algorithm succeeded finding its optimal size 27, while the sizes of clique sets found by GMCA and CBH are 18 and 21, respectively.

Corresponding to the column headed with Opt in Table 2, there are some numbers preceded by symbol "$\geq$," which means maximum clique sizes for these instances are not known. However these numbers stand for the largest clique size found so far by some algorithms in the literature [7]. Here we assume these numbers to be the *lower bound* of maximum clique size. There are also some numbers preceded by symbol "=" in the column, which mean the solution is equal to the lower bound. For example for **c-fat500-10**, both our algorithm and CBH reached its lower bound. Also for **hamming10-4** and **sanr400-0.7** our algorithm performs better than CBH.

Our algorithm also did well with **san** and **p-hat** graphs, where the algorithm is the best among these 3 approaches. It is worth mentioning that for **san400-0.9-1** our algorithm got

the optimal clique set with size 100, while only half of its optimal size obtained by GMCA and CBH. Another important finding is that for **p-hat1000-3** and **p-hat1500-2** the clique sizes found by our algorithm are larger than the lower bounds known so far, where these answers are preceded by symbol "•" in Table 2. So these lower bounds should be modified. The results for **brock** graphs are almost the same among these 3 approaches where CBH and our algorithm perform better than GMCA.

Our algorithm performs better for **MANN** graphs, where the optimal solution is obtained for **MANN-a9** and **MANN-a27**. The solutions by the algorithm are the nearest to the maximum among these approaches for **MANN-a45** and **MANN-a81**.

Looking at Table 2 again, we can find that our algorithm is much faster than GMCA and CBH for some large dense graphs, however, for some sparse graphs its running time is at the same order of magnitude with GMCA and CBH, sometimes slower than those two approaches. In average our algorithm is faster than GMCA and CBH.

4. CONCLUSIONS

In this paper we have presented a genetic approach for MISP. We adopt a permutation encoding with a greedy decoding algorithm to solve MISP. The notable feature of this algorithm is that its concept and implementation are very simple. The algorithm was tested on DIMACS benchmark graphs, and obtained optimal solutions for 36 out of 54 graphs. Moreover two new lower bounds were found for **p-hat1000-3** and **p-hat1500-2**. Such results are very encouraging compared with one of best existing algorithms CBH [7] and a hybrid genetic approach GMCA [6]. Besides final clique sets obtained, the overall running time for one trial for our algorithm in average is also faster than CBH and GMCA.

5. REFERENCES

[1] H. S. Wilf, *Algorithms and Complexity*, Prentice-Hall, 1986.

[2] P. M. Pardalos, and J. Xue, "The maximum clique problems," *Journal of Global Optimization*, vol. 4, pp. 301–328, 1994.

[3] C. Friden, A. Hertz, and D. De Werra, "TABARIS: An exact algorithm based on tabu search for finding a maximum independent set in a graph," *Computers Opns Res.*, vol. 17, no. 5, pp. 437–445, 1990.

[4] M. Kubo, K. Fujisawa, A. Yoshikawa, and S. Morito, "An approximate algorithm for the maximum stable set problem," *1993 Japan Oper. Res. Fall Conf.*, 2-B-6, pp. 162–163, 1993.

[5] A. S. Murthy, G. Parthasarathy, and V. U. K. Sastry, "Clique finding — A genetic approach," *Proc. of the First IEEE Conf. on Evolutionary Computation*, pp. 18–21, 1994.

[6] T. N. Bui, and P. H. Eppley, "A hybrid genetic algorithm for the maximum clique problem," *Proc. 6th Intl. Conf. on GAs*, pp. 478–484, 1995.

[7] L. E. Gibbons, D. W. Hearn, and P. M. Pardalos, "A continuous based heuristic for the maximum clique problem," *Research Report 94-9*, Department of Industrial & Systems Engineering, University of Florida, 1994.

[8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[9] D. R. Jones, and M. A. Beltramo, "Solving partitioning problems with genetic algorithms," *Proc. 4th Intl. Conf. on GAs*, pp. 442–449, 1991.

[10] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the traveling salesman problem," *Proc. 2nd Intl. Conf. on GAs*, pp. 224–230, 1987.

[11] X. Liu, A. Sakamoto, and T. Shimamoto, "Genetic channel router," *IEICE Trans. Fundamentals*, vol. E77-A, no. 3, pp. 492–501, 1994.

Table 2: Experimental Results for DIMACS benchmark graphs

Data	Nodes	Edges	Opt.	GMCA		CBH		our method	
				Sol.	Time	Sol.	Time	Sol.	Time
c-fat200-1	200	1534	12	**12**	8.2	**12**	7.6	**12**	12.3
c-fat200-2	200	3235	24			**24**	5.4	**24**	16.4
c-fat200-5	200	8473	58			**58**	4.3	**58**	21.1
c-fat500-1	500	4459	14	**14**	33.2	**14**	64.6	**14**	60.7
c-fat500-2	500	9139	26			**26**	45.4	**26**	72.9
c-fat500-5	500	23191	64			**64**	55.4	**64**	97.7
c-fat500-10	500	46627	≧ 126			= **126**	94.1	= **126**	125.8
johnson8-2-4	28	210	4			**4**	0.3	**4**	0.7
johnson8-4-4	70	1855	14			**14**	0.4	**14**	2.0
johnson16-2-4	120	5460	8	**8**	6.0	**8**	2.0	**8**	4.5
johnson32-2-4	496	107880	16	**16**	187.4	**16**	70.8	**16**	63.2
keller4	171	9435	11	**11**	13.3	10	6.8	**11**	9.1
keller5	776	225990	27	18	438.1	21	156.9	**27**	256.7
keller6	3361	4619898	≧ 59			*DNR*		50	4798.6
hamming6-2	64	1824	32			**32**	0.2	**32**	1.4
hamming6-4	64	704	4			**4**	0.0	**4**	1.0
hamming8-2	256	31616	128	**128**	53.0	**128**	7.6	**128**	21.2
hamming8-4	256	20864	16			**16**	9.7	**16**	19.9
hamming10-2	1024	518656	512	**512**	886.6	**512**	798.7	**512**	351.3
hamming10-4	1024	434176	≧ 40			35	163.8	36	422.9
sanr200-0.7	200	13868	18	17	21.5	**18**	8.9	**18**	14.6
sanr200-0.9	200	17863	≧ 42			41	10.2	41	17.5
sanr400-0.5	400	39984	13	12	69.6	12	29.6	**13**	45.8
sanr400-0.7	400	55869	≧ 21			20	31.6	= **21**	56.7
san200-0.7-1	200	13930	30	**30**	51.7	15	10.1	**30**	16.6
san200-0.7-2	200	13930	18			12	15.5	**18**	15.5
san200-0.9-1	200	17910	70			46	9.7	**70**	18.8
san200-0.9-2	200	17910	60			36	11.8	**60**	16.5
san200-0.9-3	200	17910	44			30	11.3	37	17.6
san400-0.5-1	400	39900	13	7	411.2	8	45.7	**13**	43.1
san400-0.7-1	400	55860	40			20	106.8	**40**	59.4
san400-0.7-2	400	55860	30			15	65.4	**30**	50.6
san400-0.7-3	400	55860	22			14	74.3	17	48.1
san400-0.9-1	400	71820	100	50	128.6	50	99.5	**100**	70.5
san1000	1000	250500	15	8	704.3	8	675.4	10	242.8
brock200-1	200	14834	21	20	27.9	20	7.4	20	14.9
brock200-2	200	9876	12			**12**	10.3	10	12.1
brock200-3	200	12048	15			14	30.3	14	13.1
brock200-4	200	13089	17			16	6.9	16	14.0
brock400-1	400	59723	27	20	118.8	23	28.6	23	50.9
brock400-2	400	59786	29			24	117.8	24	52.5
brock400-3	400	59681	31			23	40.9	23	51.4
brock400-4	400	59765	33			24	62.2	23	56.8
brock800-1	800	207505	23	18	460.8	20	407.7	18	172.1
brock800-2	800	208166	24			19	351.5	19	188.8
brock800-3	800	207333	25			20	411.5	18	190.9
brock800-4	800	207643	26			19	355.0	20	171.8
p-hat300-1	300	10933	8	**8**	20.0	**8**	27.0	**8**	24.1
p-hat300-2	300	21928	25			**25**	23.2	**25**	30.4
p-hat300-3	300	33390	36			**36**	43.8	35	36.1
p-hat500-1	500	31569	9	**9**	49.1	**9**	118.3	**9**	62.0
p-hat500-2	500	62946	36			35	101.1	**36**	85.9
p-hat500-3	500	93800	≧ 49			= **49**	113.9	= **49**	101.0
p-hat700-1	700	60999	11	**11**	310.1	**11**	355.3	**11**	117.7
p-hat700-2	700	121728	44			**44**	241.9	**44**	165.0
p-hat700-3	700	183010	≧ 62			60	308.1	61	192.7
p-hat1000-1	1000	122253	10	8	671.0	**10**	831.9	**10**	247.6
p-hat1000-2	1000	244799	≧ 46			= **46**	629.4	= **46**	351.5
p-hat1000-3	1000	371746	≧ 65			= **65**	1000.8	• **68**	427.9
p-hat1500-1	1500	284923	12	10	1580.3	11	2661.5	11	573.2
p-hat1500-2	1500	568960	≧ 63			= **63**	1252.1	• **64**	952.0
p-hat1500-3	1500	847244	≧ 94			= **94**	4699.9	92	1079.5
MANN-a9	45	918	16			**16**	0.3	**16**	1.0
MANN-a27	378	70551	126	125	121.8	121	66.6	**126**	52.4
MANN-a45	1035	533115	345	337	916.9	336	1148.0	341	459.3
MANN-a81	3321	5506380	≧ 1100			*DNR*		1094	6249.0