Finding Maximum Cliques with Distributed Ants

Thang N. Bui and Joseph R. Rizzo, Jr.

Department of Computer Science
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
tbui@psu.edu, jrr200@cs.hbg.psu.edu

Abstract. In this paper we describe an ant system algorithm (ASMC) for the problem of finding the maximum clique in a given graph. In the algorithm each ant has only local knowledge of the graph. Working together the ants induce a candidate set of vertices from which a clique can be constructed. The algorithm was designed so that it can be easily implemented in a distributed system. One such implementation is also described in the paper. For 22 of the 30 graphs tested ASMC found the optimal solution. For the remaining graphs ASMC produced solutions that are within 16% of the optimal, with most being within 8% of the optimal. The performance of ASMC is comparable to existing algorithms.

1 Introduction

Let $G = (V, E)$ be a graph with vertex set V and edge set E. A *clique* in G is a complete subgraph, i.e., a subgraph of G in which there is an edge between any two vertices in the subgraph. The *size* of a clique is the number of vertices in the clique. The MAXCLIQUE problem is the problem of finding the largest clique in a given graph. MAXCLIQUE arises in a variety of problems such as finding good codes, identifying faulty processors in multiprocessor systems and finding counterexamples to Keller's conjecture in geometry [2][24][25][15][16]. However, it is well known that MAXCLIQUE is $\mathcal{NP}$-hard [11], hence it is not expected to have a polynomial time algorithm. The next best thing to have would be a good and efficient approximation algorithm. But it has been shown under various complexity assumptions that finding a good approximation to an instance of MAXCLIQUE is just as hard as finding an optimal solution. For example, it is known that unless $\mathcal{NP} = co-\mathcal{RP}$ no polynomial time algorithm can achieve an approximation factor of $n^{1-\epsilon}$ for MAXCLIQUE for arbitrarily small ϵ [13].

In practice heuristics are used to solve MAXCLIQUE. One of the simplest such heuristics is the greedy heuristic, a version of which is described in Section 3. Other heuristic approaches to MAXCLIQUE include tabu search, continuous-based heuristics, genetic algorithms and ant colony optimization [26][12][4][10]. A good review of MAXCLIQUE and its algorithms is given in [7].

In this paper we give an ant system algorithm, called ASMC, for MAXCLIQUE. Our algorithm differs from ant colony optimization (ACO) algorithms in that each ant in ASMC does not solve the entire problem, and each ant has only local knowledge of the graph. As in ACO, ants in ASMC use pheromone to help guide the search. Also included in ASMC is a local optimization step where a clique is constructed based on the positions of the ants in the graph. A major impetus for the development of ASMC was to facilitate a variety of distributed implementations. This paper describes one such implementation. The paper also gives a sequential implementation of ASMC. Experimental results on a set of test graphs from DIMACS [14] show that ASMC is comparable to existing algorithms for MAXCLIQUE.

The rest of the paper is organized as follows. In Section 2 we give some preliminaries. In Section 3 we describe our ant system algorithm for the MAXCLIQUE problem. Section 4 describes a distributed implementation of this algorithm. We compare the performance of our algorithm against some existing algorithms in Section 5. The conclusion is given in Section 6.

2 Preliminaries

Ant colony optimization (ACO) is a meta heuristic inspired by the behavior of foraging ants [9]. By using pheromone, ants are able to help each other find short paths to food sources. The main idea in ACO is to have a collection of ants each of which takes its turn solving the problem. For each solution found by an ant an amount of pheromone proportionate to the quality of the solution is placed in the appropriate places in the search space. The pheromone serves as a means of guiding later ants in their search for a solution. This technique has been successfully applied to a number of problems including MAXCLIQUE, e.g., see [9][10][21]. In an ACO algorithm, each individual ant has the full knowledge of the problem and the placement of pheromone is done after an ant has solved the problem, not while it is searching for the solution. In contrast, ants in our ant system have only local knowledge. Here the placement of pheromone is done by each ant as it is moving about the search space. Each individual ant follows the same set of rules and none solves the problem by itself. It is from their collective behavior that we obtain a solution to the problem. We can have more than one species of ants in the system. Ants in different species can behave differently or can follow the same sets of rules. The species may not interact directly except perhaps through the pheromone. Species can be either collaborative or competitive. This technique has been used successfully in solving other problems, e.g., see [5][6].

For MAXCLIQUE, each vertex in the input graph is considered as a location that ants can occupy. In principle, each vertex can hold an arbitrary number of ants. Ants in our system move from vertex to vertex along the edges of the graph. As an ant traverses an edge of the graph it also puts down a certain amount of pheromone on that edge. To allow for more exploration and possible escape from local optima, pheromone evaporates over time. In addition to using pheromone as a means of communication, ants in our system also use their positions to communicate. For example, a vertex that is occupied by more ants of the same species is more attractive to an ant of that species when it decides where to move. As in most search algorithms of this type there is a constant tug-of-war between exploration and exploitation of the search space. It is usually useful to have more exploration in the beginning and more exploitation nearer to the end so that the algorithm will not converge prematurely to a local optimum. To allow for this type of strategy, ants in our system have an adaptive behavior. For example, they rely less on pheromone and more on the structure of the graph in the beginning of the algorithm. As the algorithm progresses ants make more use of pheromone in determining their movement.

3 Ant System Algorithm for MAXCLIQUE (ASMC)

In this section we describe an ant system algorithm for the MAXCLIQUE problem. The main idea of the algorithm is as follows. Ants are distributed on the graph vertices. Each ant follows the same set of rules to move from vertex to vertex. The rules are designed so that ants are encouraged to aggregate on sets of vertices that are highly connected. These highly connected portions of the graph then serve as candidate sets of vertices from which we can construct cliques.

The algorithm starts by distributing ants of different species on to the vertices of the graph according to some predetermined configuration. It then goes through a number of stages. Each stage of the algorithm consists of a number of cycles. In each cycle a fraction of the ants are selected to move. Each ant that is selected to move will move with certain probability. Its destination is determined by various factors such as pheromone and the structure of the neighborhood around the ant. At the end of each stage, the algorithm constructs a clique based on the current configuration of the ants. The algorithm then shuffles the ants around to help them escape from local optima before moving on to the next stage. After finishing all stages, the algorithm returns the largest clique found in all stages. The algorithm is given in Figure 1. Details of the algorithm are given in the remainder of this section.

3.1 Initialization

The algorithm starts by instantiating $6n$ ants for each species, where n is the number of vertices in the graph. The number of species is a parameter to the algorithm. The description that follows applies to each species. The ants are then distributed to the vertices of the graph. To determine how the ants should be distributed, the algorithm first runs a simple greedy algorithm to find a clique. A large fraction (90%) of the ants are then distributed at random on vertices of the clique found by the greedy algorithm. The remaining ants are distributed randomly on the rest of the vertices. By distributing a majority of the ants to the vertices of a clique found by the greedy algorithm we help speed up the search process at the cost of a possible bias introduced by that clique. This potential bias is alleviated somewhat by the random distribution of the remaining ants and by the placement of different species.

```
Initialize ants
Distribute ants on vertices of the graph
for stage=1 to MaxStage
  for cycle=1 to MaxCycle
    Randomly activate 75% of all ants
      if an ant $a$ is activated
         move($a$)
  endfor
  Find cliques through local optimization
  Save solutions
  Shuffle ants
endfor
return the best solution found
```

Fig. 1. Ant System Algorithm for MAXCLIQUE

The greedy algorithm is given in Figure 2. This completes the initialization step. The algorithm then goes through a number of stages. Each stage in turn consists of a number of cycles and at the end of which a clique is constructed. In each cycle 75% of the ants are activated. An activated ant can decide to stay where it is or move to another vertex. In our experiments we found that it is sufficient to have about 25 stages and 10 cycles in each stage. The next subsection describes this process in more detail.

```
$R \leftarrow \emptyset$
for each $v \in V$
   mark $v$ $feasible$
endfor
sort $V$ into decreasing degree order
for each $v \in V$ in the sorted order
  if $v$ is $feasible$
     $R \leftarrow R \cup \{v\}$
     for each $w \in V$ that is not adjacent to $v$
       mark $w$ $infeasible$
     endfor
endfor
return $R$
```

Fig. 2. A Greedy Algorithm for Finding a Clique

3.2 How an Ant Moves

When an ant is activated, i.e., selected to move, it first determines whether it will move or not. This choice is made probabilistically and ants that are older are less likely to move than younger ants. The age of an ant is the number of times that it has moved. This rule enables the ants to explore more in the beginning and move less later on.

If an ant decides to move then it can either move to a randomly selected vertex or to a vertex determined by the *vertex attractiveness (VA) heuristic*. The former choice is made with a fixed probability. The availability of this choice offers the ants a chance to escape from local optima. The latter choice, when selected, uses the VA heuristic to help determine the ant's destination. The VA heuristics computes, for each vertex adjacent to the vertex that the ant is currently on, the probability that the ant will move to that vertex. The ant then selects its destination based on this probability. More specifically, if an ant is currently on vertex i and j is a vertex adjacent to i then the probability that the ant will move to j, called $p_{i,j}$, is defined as follows.

$$p_{i,j} = \frac{\kappa \cdot \tau_{i,j}(t) + \lambda \cdot \nu_j(t) + \mu \cdot \sigma_j^d}{\sum_{j \in L(i)} [\kappa \cdot \tau_{i,j}(t) + \lambda \cdot \nu_j(t) + \mu \cdot \sigma_j^d]}$$

where κ, λ, and μ are nonnegative weights, $L(i)$ is a list of vertices that are adjacent to i, and $\tau_{i,j}(t), \nu_j(t), \sigma_j^d$ are the pheromone score, the population score and the connectivity score, respectively. The pheromone score, $\tau_{i,j}(t)$, is the total amount of pheromone on the edge (i,j) at time t. Each time an ant traverses an edge of the graph, it lays down a certain amount of pheromone on that edge. Under the assumption that frequently traversed edges most likely link groups of highly connected vertices, the pheromone score, acting as a form of memory that records the aggregate behavior of the ants, helps encourage ants to follow these edges. The population score, $\nu_j(t)$, is the number of same-species ants on vertex j at time t. This score helps to cluster ants of the same species. This score is helpful as the final clique is extracted from groups of vertices that are occupied by ants of the same species. Finally, the connectivity score, σ_j^d, measures how well the vertices in the neighborhood of j with radius d connect to each other. More formally, let $V_j^d = \{u \in V \mid u \text{ is reachable from } j \text{ by a path of length at most } d\}$ and E_j^d be the set of edges of the graph that have both endpoints in V_j^d. Then,

$$\sigma_j^d = \frac{2|E_j^d|}{|V_j^d|(|V_j^d| - 1)}.$$

The inclusion of the connectivity score in the computation of $p_{i,j}$ helps ants to discover well-connected regions of the graph.

The nonnegative weights κ and λ are constants whereas μ decreases over time. In the beginning μ is set to a value that is higher than both κ and λ so that the structure of the graph plays a much larger role in the exploration of the ants. μ is decreased linearly over time to allow the aggregate behavior of the ants, expressed through pheromone scores and population scores, to help in guiding the movement of an ant. We stop decreasing the value of μ when its value is comparable to that of κ and λ.

The connectivity score depends on the neighborhood radius d, which increases as the ant's age increases. Initially, d is set to 1. Overall, as an ant gets older it moves less often but when it does it looks at a larger neighborhood before deciding where to move. This works well since, as time passes, more information are stored in the pheromone deposited on the edges and ant positions.

At the end of a stage, the algorithm extracts a clique from the current ant configuration. It also attempts to help the ants move away from a possible local optimum by randomly moving a small fraction of the ants to different locations on the graph.

3.3 Local Optimization

At the end of each stage, the algorithm takes the following actions for each species. It first extracts a candidate set of vertices C based on the positions of the ants in that species and the placement of pheromone on the graph. Specifically, each vertex v in the graph is given a *threshold score* by the following formula

$$\text{thresholdScore}(v) = \alpha n_v^S + \beta p_v,$$

where n_v^S is the number of ants of species S occupying vertex v, p_v is the total amount of pheromone on all edges incident to v, and α and β are weights that vary after each stage, from 10 to 5 and from 0.1 to 1, respectively. In this fashion, the threshold scores emphasize the number of ants in the earlier stages, when exploration is important, and emphasize the pheromone in the later stages when exploitation is more important. The candidate set C is obtained by taking those vertices whose threshold scores are in the top γ percent, where γ varies from 10% to 25% over the course of the algorithm. In the beginning stages it is not expected that ant configurations would reflect the highly connected regions of the graph very well as ants may not have enough time to explore the graph yet. In later stages, ant configurations would be more accurate and hence it is reasonable to increase γ.

The next step in the FindClique algorithm is to expand the size of the candidate set C. This is done by adding vertices to C until C has grown by δ%. The added vertices are selected in order of how well they are connected to C. That is, the more neighbors a vertex has in C the higher it is on the list of vertices to be selected. This step is done to ensure that any nearby local optima that the ant configuration missed are included in the candidate set. FindClique seems to work well when δ varies over time from 0% to 1.3%.

Finally, a clique is extracted from this candidate set in a greedy manner similar to the greedy algorithm described earlier. The full FindClique algorithm is given in Figure 3.

The clique R obtained by the FindClique algorithm is then improved in a simple manner by examining each vertex that is not in R and see if adding it to

```
FindClique ($G = (V, E)$, $S$) // $S$: species name
// build candidate set
for each $v \in V$
  thresholdScore($v$) = $\alpha \cdot n_v^S + \beta \cdot p_v$
endfor
Let $C$ be the set of vertices whose threshold
 scores are in the the top $\gamma$%
// expand candidate set
for each $v \in V$
  solutionDegree($v$) = $|\{u \in C \mid (v,u) \in E\}|$
endfor
Select vertices, in decreasing order of solution
 degree, to add to $C$ until $C$ has grown by $\delta$%
// identify clique
$R \leftarrow \emptyset$
while $C \neq \emptyset$
  Update solutionDegree of vertices in $C$
  $v \leftarrow$ highest solutionDegree vertex in $C$
  $R \leftarrow R \cup \{v\}$
  $C \leftarrow C - (\{v\} \cup \{u \in C \mid (u,v) \notin E\})$
endwhile
return $R$
```

Fig. 3. The FindClique algorithm

R still gives us a clique. If so, the vertex is added to R. For each vertex that is added to R in this manner, the algorithm also adds more ants of the appropriate species to that vertex and adds pheromone to the edges incident to that vertex. This is done to enforce the fact that the new R is a clique which can be utilized by the ants in the next stage.

The last operation the algorithm performs before going to the next stage is to perturb the ant configuration enough to allow the ants a chance to break out of a possible local optimum. The perturbation is, however, not too large so that all previous information gathered by the ants are destroyed. Specifically, the algorithm selects 40% of the vertices at random. The ants on the selected vertices are swapped randomly among these vertices. Furthermore, the pheromone on 10% of the edges incident to each of these selected vertices is reduced by a small amount. Experimental results found that this perturbation produced desirable effects.

The algorithm is now ready to start another stage. When all stages have finished, the algorithm returns the largest clique found at the end of each stage.

4 The Distributed Implementation

ASMC is especially suited to distributed implementation since ants contribute partial solutions based on local information. Distribution makes it possible to optimize the algorithm for speed, due to parallel processing, or for space, due to partitioning ants and large graphs over several machines. Though the benefits gained depend on the specific implementation, ASMC itself does not preclude any of the benefits.

The distributed implementation built for this paper is a simple proof of concept and, as such, perfoms synchronous interprocess communication through a central server. Ants are distributed across four machines, each of which has a complete copy of the graph. The idea is for ants to move from processor to processor. One machine is designated the server and the others clients 1, 2 and 3. The server coordinates four synchronous transactions: starting, ant transfer, local optimization and ending.

- To **start**, the server first waits for each client to connect. It then partitions the vertices into four roughly equal sets and assigns each set to a processor, including itself. It sends each client a list of which vertices are “owned” by which processors.
- **Ant transfer** occurs at the end of each stage. The server asks each client for a list of ants that are moving to other processors. It sorts these ants according to their destinations and sends each client a list of incoming ants. Each processor instantiates the ants on its copy of the graph. It also updates cached data regarding vertices and edges connected to its “owned” vertices. The cached data makes it possible for ants to make informed decisions about whether to move to another processor in the future.
- **Local optimization** occurs after each ant transfer. Each client sends the server vertex and edge data, and the server performs the same operations as in the sequential implementation of ASMC. In principle, this step could be redesigned to be less centralized.
- The server **ends** by letting each client know that the run has ended. Synchonized starting and ending makes it simpler to invoke the program from a script.

Though this implementation is rather simple, other distributed ASMC designs are possible. For example, one could build a less centralized, peer-to-peer model. The ant transfers could occur asynchronously throughout each stage. The graph partitions could adapt over time to minimize the number of shared edges. The graph itself could be distributed so that each client is completely unaware of vertices it does not “own,” thereby enabling runs on extremely large graphs in a reasonable time.

5 Experimental Results

In this section we describe the results obtained from testing the sequential implementation and the distributed implementation of ASMC.

Sequential Implementation. The algorithm was implemented in C++ and run on an Intel Pentium IV 2.4GHz machine. The algorithm was tested on a set of 30 graphs selected from the benchmark graphs of the Second DIMACS Implementation Challenge [14]. The graphs that we used have up to 1,500 vertices and over 500,000 edges. There are 9 classes of graphs. The C-fat graphs are from fault diagnosis problems [2], the Johnson and Hamming graphs are from problems in coding theory [24][25]. The Keller graphs are from tiling problems in geometry [15][16]. The remaining families of graphs: San, Sanr, Brock, P-hat and Mann are various types of random graphs with known optimal cliques. In our implementation of ASMC only one species was used.

For each graph, ASMC is run 100 times. Table 1 summarizes the results produced by ASMC. Overall, ASMC did very well for graphs in the classes C-fat, Keller, Johnson and Hamming. It always found the optimal solution. For all but two of the tested graphs in the San and Sanr families ASMC found the optimal solution. For the Mann graphs the solutions returned by ASMC were within 1% of the optimal solution. The most difficult families of graphs for ASMC were the Brock and P-hat graphs. For the P-hat graphs ASMC produced solutions that were within 10% of the optimal solution, with some being optimal. However, for the Brock graphs solutions given by ASMC were as bad as 18% from the optimal. These results seem to be consistent with those obtained by other algorithms. It should be noted that in developing ASMC, the parameters used were derived from experiments based only on three graphs from this set: keller4, san200_0.7_1 and p_hat300_1.

We compare the results given by ASMC against the following three algorithms for MAXCLIQUE: (i) GMCA – a hybrid genetic algorithm [4], (ii) CBH – a global optimization algorithm that uses a continuous formulation of MAXCLIQUE [12], and (iii) IHN – a neural approximation algorithm based on discrete Hopfield networks [3]. These algorithms are chosen to represent different approaches to MAXCLIQUE as well as for the availability of their test results. Table 1 summarizes the results of ASMC together with these three algorithms. It is clear that ASMC is comparable to the other algorithms. We provided no running time comparisons since the algorithms were tested on different machines and there were not sufficient information for us to derive reasonable conversion factors for the running times. We believe, however, that ASMC might be slower than some of these algorithms.

There are other more recent algorithms for MAXCLIQUE [1][10][19]. Their results are not included in Table 1 since there are not enough overlapped test results [1][10] or the problem solved is slightly different from MAXCLIQUE (the size of the largest known clique is required as an input to the algorithm) [19]. Based on the available data ASMC is also comparable to the algorithms of [1] and [10].

Distributed Implementation. The distributed implementation of ASMC was run on four Sun Blade 100 450MHz workstations. Due to time limitations we were able to run the algorithm only on a subset of the 30 test graphs. Also, we ran the algorithm 100 times for each graph. The results are comparable to that of the sequential implementation. Table 2 summarizes the results.

Table 1. ASMC solution quality (sequential implementation)

Graph	Vertices	Edges	Opt	ASMC Best*	ASMC Avg*	(StdDev)	Avg Time*†	GMCA Best	CBH Best	IHN Best
c-fat200-1	200	1534	12	12	12.00	(0.00)	0.46	12	12	12
c-fat500-1	500	4459	14	14	14.00	(0.00)	1.55	14	14	14
johnson16-2-4	120	5460	8	8	8.00	(0.00)	1.58	8	8	8
johnson32-2-4	496	107880	16	16	16.00	(0.00)	48.97	16	16	16
keller4	171	9435	11	11	10.44	(0.76)	3.16	11	10	–
keller5	776	225990	27	26	21.90	(1.20)	110.34	18	21	–
hamming10-2	1024	518656	512	512	512.00	(0.00)	281.04	512	512	512
hamming8-2	256	31616	128	128	128.00	(0.00)	11.20	128	128	128
san200_0.7_1	200	13930	30	30	18.81	(5.35)	4.85	30	15	30
san200_0.9_1	200	17910	70	70	47.72	(4.18)	5.75	–	–	70
san200_0.9_2	200	17910	60	60	40.80	(5.73)	5.76	–	–	41
san200_0.9_3	200	17910	44	37	32.72	(1.12)	6.02	–	–	–
san400_0.5_1	400	39900	13	13	8.35	(0.91)	16.61	7	8	–
san400_0.9_1	400	71820	100	100	55.74	(11.67)	29.83	50	50	–
sanr200_0.7	200	13868	18	18	15.32	(0.83)	4.80	17	18	17
sanr400_0.5	400	39984	13	13	10.58	(0.57)	16.56	12	12	12
san1000	1000	250500	15	15	9.66	(0.74)	135.42	8	8	10
brock200_1	200	14834	21	20	17.98	(0.93)	5.12	20	20	–
brock400_1	400	59723	27	25	20.14	(0.80)	25.23	20	23	–
brock800_1	800	207505	23	20	16.65	(0.77)	102.46	18	20	–
p_hat300_1	300	10933	8	8	7.17	(0.38)	4.26	8	8	8
p_hat300_2	300	21928	25	25	23.93	(0.81)	8.84	–	–	25
p_hat300_3	300	33390	36	36	31.82	(1.04)	12.83	–	–	36
p_hat500_1	500	31569	9	9	8.19	(0.47)	14.19	9	9	9
p_hat500_2	500	62946	36	36	32.03	(1.45)	29.40	–	–	36
p_hat700_1	700	60999	11	11	8.42	(0.54)	30.87	8	11	11
p_hat1000_1	1000	122253	10	10	8.73	(0.57)	68.00	8	10	10
p_hat1500_1	1500	284923	12	11	9.54	(0.66)	177.70	10	11	–
MANN_a27	378	70551	126	125	124.64	(0.52)	27.89	125	121	–
MANN_a45	1035	533115	345	341	338.93	(1.01)	282.15	337	336	–

*Results for 100 runs per graph. †All times are in seconds.

6 Conclusion

This paper describes an ant system algorithm for MAXCLIQUE which seems to perfom comparably with the current best known algorithms for this problem. One possible improvement of ASMC is to increase the diversity of the initial ant configuration. This can be done by generating several cliques instead of just one using the same greedy algorithm. Another direction is to make the distributed implementation more scalable, more network efficient and less centralized.

Table 2. ASMC solution quality (distributed implementation)

Graph	Vertices	Edges	Opt	ASMC Best*	ASMC Avg*	(StdDev)	Avg Time*†
c-fat200-1	200	1534	12	12	12.00	(0.00)	0.72
c-fat500-1	500	4459	14	14	14.00	(0.00)	2.61
johnson16-2-4	120	5460	8	8	8.00	(0.00)	1.37
keller4	171	9435	11	11	10.49	(0.64)	3.21
hamming8-2	256	31616	128	128	127.26	(2.78)	32.70
san200_0.7_1	200	13930	30	30	17.16	(1.93)	6.29
san200_0.9_1	200	17910	70	48	47.09	(0.40)	10.64
sanr200_0.7	200	13868	18	17	15.44	(0.69)	6.22
brock200_1	200	14834	21	20	18.49	(0.64)	7.21
p_hat300_1	300	10933	8	8	8.00	(0.00)	4.52
p_hat500_1	500	31569	9	9	8.11	(0.31)	33.60

*Results for 100 runs per graph. †All times are in seconds.

Acknowledgements. The authors would like to thank Sue Rizzo and Linda Null for useful discussions.

References

1. R. Battiti and M. Protasi, "Reactive Local Search for Maximum Clique," Proceedings of the Workshop on Algorithm Engineering (WAE'97), Venice, G. F. Italiano and S. Orlando, Eds., Venice, Italy, 1997, pp. 74–82.
2. P. Berman and A. Pelc, "Distributed Fault Diagnosis For Multiprocessor Systems," Proceedings of the 20th Annual International Symposium on Fault-Tolerant Computing, pp. 340–346, Newcastle, UK, 1990.
3. A. Bertoni, P. Campadelli and G. Grossi, "A Discrete Neural Algorithm for the Maximum Clique Problem: Analysis and Circuit Implementation," Proceedings of the Workshop on Algorithm Engineering (WAE'97), Venice, G. F. Italiano and S. Orlando, Eds., Venice, Italy, 1997.
4. T. N. Bui and P. H. Eppley, "A Hybrid Genetic Algorithm for the Maximum Clique Problem," Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA), L. Eshelman (Ed.), pp. 748–484, Morgan Kauffman Publishers, 1995.
5. T. N. Bui and C. M. Patel, "An Ant System Algorithm for Coloring Graphs," Computational Symposium on Graph Coloring and Generalizations (COLOR02), Ithaca, NY, September 2002.
6. T. N. Bui and L. C. Strite, "An Ant System Algorithm for Graph Bisection," GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, W. B. Langdon et al. (Eds.), pp. 43–51, Morgan Kauffman Publishers, 2002.
7. I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The Maximum Clique Problem," D.-Z. Du and P. M. Pardalos (Eds.), *Handbook of Combinatorial Optimization*, 4, Kluwer Academic Publishers, Boston, MA, 1999.

8. K. Corradi and S. Szabo, "A Combinatorial Approach for Keller's Conjecture," Periodica Mathematica Hungarica, 21, pp. 95–100, 1990.
9. M. Dorigo and G. Di Caro, "The Ant Colony Optimization Meta-Heuristic," *New Ideas In Optimization*, D.Corne, M. Dorigo and F. Glover (Eds.), McGraw-Hill, London, pp. 11–32, 1999.
10. S. Fenet and C. Solnon "Searching for Maximum Cliques with Ant Colony Optimization," in Applications of Evolutionary Computing, Proceedings of the EvoWorkshops 2003, April 2003 Lecture Notes in Computer Science, No. 2611, Springer-Verlag, pp. 236-245
11. M. Garey and D. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
12. L. E. Gibbons, D. W. Hearn and P. M. Pardalos, "A Continuous Based Heuristic for the Maximum Clique Problem," in [14], pp. 103–124, 1996.
13. J. Hastad, "Clique Is Hard to Approximate within $n^{1-\epsilon}$," Acta Mathematica, 182, pp. 105–142, 1999.
14. D. S. Johnson and M. A. Trick (Editors), *Cliques, Coloring and Satisfiability – Second DIMACS Implementation Challenge 1993,* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Volume 26 (1996).
15. O. H. Keller, "Über die lückenlose Erfüllung des Raumes mit Würfen," Journal für die reine und angewandte Mathematik, 163, pp. 231–238, 1930.
16. J. C. Lagarias and P. W. Shor, "Keller's Cube-Tiling Conjecture Is False In High Dimensions," Bulletin of the American Mathematical Society, 27(2), pp. 279–283, 1992.
17. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1979.
18. C. Mannino and A. Sassano, "Solving Hard Set Covering Problems," Operations Research Letters, 18, pp. 1–5, 1995.
19. E. Marchiori, "Genetic, Iterated and Multistart Local Search for the Maximum Clique Problem," Applications of Evolutionary Computing, Proceedings of the EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN, Kinsale, Ireland, S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, G.R. Raidl (Eds.), pp. 112–121, April 3-4, 2002.
20. H. Minkowski, *Diophantische Approximationen*, Teubner, Leipzig, 1907.
21. G. Navarro Varela and M.C. Sinclair, "Ant Colony Optimisation for Virtual-Wavelength-Path Routing and Wavelength Allocation," Proceedings of the Congress on Evolutionary Computation (CEC'99), Washington DC, July 1999.
22. O. Perron, "Über lückenlose Ausfüllung des n-dimensionalen Raumes durch kongruente Würfel," Mathematische Zeitschrift, 46, pp. 1–26, 161–180, 1940.
23. L. Sanchis and A. Jagota, "Some Experimental and Theoretical Results on Test Case Generators for the Maximum Clique Problem," INFORMS Journal on Computing, 8(2), pp. 87–102, Spring 1996.
24. N. J. A. Sloane, "Unsolved Problems in Graph Theory Arising from the Study of Codes," Graph Theory Notes of New York, XVIII, pp. 11–20, 1989.
25. N. J. A. Sloane and F. J. MacWilliams, *The Theory of Correcting Codes,* North Holland, Amsterdam, 1979.
26. P. Soriano and M. Gendreau, "Tabu Search Algorithms for the Maximum Clique Problem," in [14], pp. 221–244, 1996.