ACCELERATING THE ANT COLONY OPTIMIZATION BY SMART ANTS, USING GENETIC OPERATOR

Hassan Ismkhan

Department of Computer Engineering, University of Bonab, Bonab, East Azerbaijan, Iran
H.Ismkhan@bonabu.ac.ir

ABSTRACT

This paper research review Ant colony optimization (ACO) and Genetic Algorithm (GA), both are two powerful meta-heuristics. This paper explains some major defects of these two algorithm at first then proposes a new model for ACO in which, artificial ants use a quick genetic operator and accelerate their actions in selecting next state.

Experimental results show that proposed hybrid algorithm is effective and its performance including speed and accuracy beats other version.

KEYWORDS

Ant Colony Optimization, Genetic Algorithm, Genetic Operator, Speeding up

1. INTRODUCTION

The versions of ACO have been successfully applied to type of combinatorial optimization problem same as scheduling problems [5], routing problems [6] and knapsack problem [9]. The ACO is also successful in dealing with continuous optimization problems [11] [12]. However time complexity of ACO is \( \Omega (N^2) \) which is too high [10] and prevents applicability of ACO for large scale problems. In other hand, versions of Genetic Algorithm (GA) which is one of the most important type of meta-heuristics, have been also applied to combinatorial and continues optimization problems. The types of GA have solved scheduling problems [1], knapsack problems [14] and routing problems [15] which all are combinatorial optimization problems. GAs also have been successfully applied to continuous optimization problems such as function optimization [16] [17].

Although GA and ACO are successful meta-heuristics, both have their own bottlenecks. Considering defect points of GA and ACO will help us to design effective frameworks.

GA has powerful population-based search engine which can explore new regions of answer area, instead its exploitation ability dramatically comes down when appropriate operators is not used in body of GA. It should be considered that designing new operators for GA needs more invention. Operators should have enough speed and should increase exploitation ability of GA, where population of GA grant exploration ability of GA.

ACO usually uses small group of artificial ants to make solutions. In addition, ACO uses additional information in its baseline algorithm which is known as pheromone trails. Pheromone update by ants and help ants to find better solutions. Although using pheromone causes appropriate level of exploitation, these additional information decrease exploration ability of algorithm and led to local trap problem.
There are some research attempts to use and merge benefits of both GA and ACO [7]. This paper also proposes an effective model of merging ACO and GA which has appropriate speed and accuracy. Therefore, the rest of the paper is scheduled as follows: next section reviews GA and ACO briefly, section 3 proposes our method, section 4 presents forward experimental results, and finally, section 5 summarizes the paper.

2. GA and ACO, Brief Review of Their Algorithms

The case study of this research is the Travelling Salesman Problem (TSP). TSP is finding the minimum Hamilton tour in a graph. Also, in this section, both GA and ACO are explained by using TSP.

2.1. ACO

Ant System (AS) is the first version of ACO. In AS, in each step, m artificial ants locate on m different start nodes and begin to construct their tours in n sub-steps, where n is the number of nodes in the graph. Each ant to select the next node uses a rule which is named transition rule and is defined as below:

\[ p_k(r,s) = \left\{ \begin{array}{ll} \frac{[\tau(r,s)\alpha][\eta(r,s)\beta]}{\sum_{u\in J(r)} [\tau(r,u)\alpha][\eta(r,u)\beta]} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{array} \right. \quad (1) \]

Where \( p_k(r,s) \) is the probability of choosing node s after node r by ant k. \( J_k(r) \) is the set of unvisited nodes of ant k. \( \eta(r,u) = 1/distance(r,u) \) and \( \tau(r,s) \) is the pheromone amount of rs. The \( \beta \) and \( \alpha \) are parameters. When each ant completes its tour, it tries to update the pheromone value by the following equation which is named pheromone update rule.

\[ \tau(r,s) \leftarrow (1 - \alpha) \tau(r,s) + \sum_{k=1}^{m} \Delta \tau_k(r,s) \quad (2) \]

Where \( \Delta \tau_k(r,s) = \left\{ \begin{array}{ll} \frac{1}{\text{tour length of ant } k} & \text{if edge } (r,s) \text{is in ant } k \text{ tour} \\ 0 & \text{otherwise} \end{array} \right. \],

Ant Colony System (ACS) is another version of ACO which adds another rule in which ants after each sub-step update pheromone locally when selecting their next node. This rule is named local pheromone update rule and implemented by the following equation.

\[ \tau(r,s) \leftarrow (1 - \rho) \tau(r,s) + \rho \tau_0 \quad (3) \]

In ACS transition rule uses the following equation:

\[ S = \left\{ \begin{array}{ll} \arg \max_u \{ [\tau(r,u)]^\alpha [\eta(r,u)]^\beta \} & \text{if } q \leq q_0 \\ \text{use (1)} & \text{otherwise} \end{array} \right. \quad (4) \]

Max-Min Ant System is another version of ACO in which pheromone values are bounded between a maximum and minimum value. Both methodologies in ACS and MMAS help ACO to escape from local optima traps [13]. The baseline algorithm for ACO is as figure 1.
For $I = 1$ to MAX-Number-of-Iteration

Locate $m$ ants on $m$ different nodes

For $J = 1$ to $N$ (N is the number of nodes in graph)

For each ant

Select next node according to (4)

Update pheromone locally by (2)

End For

Update pheromone globally by (3)

End For

Figure 1. ACO algorithm

2.2. GA

GA usually starts with population of random solutions. In each step, new solutions are generated by operators. These operators select and take one or more solutions from population and produce new solution according to selected solutions.

There are two types of operator for GA: mutation operator and crossover operator. Mutation operator, takes a solution and changes it. This operator helps GA to prevent increasing similar solutions in population. This is a benefit, because similar solutions in population causes premature convergence and local optima problems. The Double-Bridge is one of the most famous mutation for TSP solver GAs. Figure 2 shows Double-Bridge operator.

Figure 2. The Double-Bridge operator: Part (A) Shows how Double-Bridge works, Part (B) Shows an example for Double-Bridge

In other hand, crossover operator takes two solutions usually are named parent, and produces one or two solutions according to (similar to) parents which are named children or offspring. Paper [3] reviews some of recent crossover operators. There are some accurate crossover operators same as EAX [8] but its time complexity is $o(n^2)$ which is too slow. Here we point to Improved Greedy Crossover which has $o(n)$ time complexity [7]. To explain IGX, please consider a graph with 8 nodes that its edges cost are in matrix of figure 4. Figure 5 shows how IGX constructs child according to the two parents as 1-6-8-4-5-7-3-2 and 6-2-4-8-5-1-7-3.
Figure 4. The cost matrix of a graph with 8 nodes

Use Fig. 2’s father and mother:

**Step 0**: Double-linked list construction

- Dublin's double-linked list
- Mother's double-linked list

**Step 1**: First node selection: It is selected randomly.
- Please suppose it is 4.
- child: 4

Lists updating: Eliminates two pointers that point to 4.

**Step 2**: Select 2\textsuperscript{nd} node

- Among 5, 8, 8 and 2, 5 is closer to 4 so it selected as next node.
- child: 4 5

Lists updating: Eliminates two pointers that point to 5.

**Step 3**: Select 3\textsuperscript{rd} node

- Among 7, 8, 1 and 2, 1 is closer to 5 so it selected as next node.

- child: 4 5 1

Lists updating: Eliminates two pointers that point to 1.

**Step 4**: Select 4\textsuperscript{th} node

- Among 2, 6, 7 and 8, 2 is closer to 1 so it selected as next node.

- child: 4 5 1 2

Lists updating: Eliminates two pointers that point to 2.

Figure 5. IGX [2]
GA, itself has two important types: generational GA and steady state GA. In generational GA, in each generation, new produced solutions (which are produced by mutation or crossover operators) are added to population and after some generations, the size of population normalized by eliminating solutions with low fitness. In steady state GA, in each generation, new solution is replaced by one of solution in population. Figure 6 shows baseline algorithm of both generational and steady state GA.

| Generational GA | Steady state GA |
|------------------|------------------|
| Initial population; | Initial population; |
| **For** gen-num = 1 to Max-Gen | **For** gen-num = 1 to Max-Gen |
| while(some conditions) | select individuals to apply crossover; |
| select individuals to apply crossover; | apply crossover and generate individual(s); |
| apply crossover and generate individual(s); | mutate new individual(s); |
| if(random number < MR) | Replace new individual(s) with some ones from population |
| mutate new individual(s); | | |
| Add new individuals to population; | | |
| **End While** | **End For** |
| Remove extra individuals from population; | |
| **End For** | Steady state GA |

![Figure 6. The two types of GA](image)

3. **PROPOSED SMART ANT**

As you see in previous section, IGX selects one edge with lowest cost among four edges. Proposed smart ant are designed same as IGX, instead probing costs of four edges, probes results of transition rule of ACS which stated in equation 5.

\[
S = \begin{cases} 
\arg \max_{u \in PC} \{[(\tau(r, u))^\alpha, (\eta(r, u))^\beta]\}, & \text{if } q \leq q_0 \\
\text{use (6)}, & \text{otherwise}
\end{cases}
\]  

\[
p_k(r, s) = \begin{cases} 
\frac{[(\tau(r, s))^\alpha, (\eta(r, s))^\beta]}{\sum_{u \in PC} [(\tau(r, u))^\alpha, (\eta(r, u))^\beta]}, & \text{if } s \in PC \\
0, & \text{otherwise}
\end{cases}
\]  

Where all notations in (5) and (6) are same as (1) and (4) and PC is set of all four nodes which are neighbour of current node in both parents.
Please consider that proposed hybrid algorithm consider a population of solutions and in each step, smart ant which operates as IGX, takes two solutions from population, and produce new child. In this process, smart ant considers equation 6 instead of cost of edges.

Initial population:

For \text{gen-num} = 1 \text{ to Max-Gen}

Assign two solution for each ant

For each ant

For \text{I = 1 \text{ to N}}

Select next node according to 5

Update pheromone locally

End For

//Solution of this ant is now completed
Apply 2-Opt [13] local search on solution of this ant
Update pheromone globally by equation 3

End For

Figure 6. Proposed hybrid ACO with smart ants

4. EXPERIMENTAL RESULTS

To probe performance of Hybrid-ACO-with-Smart-Ant (proposed method), call it HACO-SA, we set up an experiment in which HACO-SA competed with MMAS to solve instance selected from TSPLIB [5]. We ran this experiment on 2.40 GHz Intel CPU with 1 GB of random access memory. All algorithm were implemented by c++ programming language. Experimental results show that accuracy of HACO-SA is as well as MMAS, however speed of HACO-SA is more than MMAS. Table 1 shows average (the number of runs was 30) cost of solutions gained by each algorithm. As can been seen in table 1, HACO-SA even has better solutions in average and for last four instances completely overcomes MMAS.

Table 1. Average costs, computed by each algorithm per each instance.

| Instance | MMAS     | HACO-SA   |
|----------|----------|-----------|
| eil51    | 426      | 426       |
| eil76    | 538      | 538       |
| kroA100  | 21282    | 21283.5   |
| lin105   | 14379    | 14410.29  |
| d198     | 15883.12 | 15983.5   |
| lin318   | 42267.5  | 42219.33  |
| pcb442   | 51010.233| 50907.15  |
| att532   | 27837.11 | 27761.3   |
| rat783   | 8899.3   | 8851.21   |
Table 2 shows speed of HACO-SA is better than MMAS for last four instances. Also for lin105, HACO-SA is quick than MMAS.

| Average time | MMAS   | HACO-SA |
|--------------|--------|---------|
| eil51        | 1.4887767 | 1.759   |
| eil76        | 1.8801767 | 2.27605 |
| kroA100      | 3.859565  | 4.1837  |
| lin105       | 4.4618017 | 3.16685 |
| d198         | 9.13653   | 10.13775|
| lin318       | 19.974383 | 15.9019 |
| pcb442       | 16.971528 | 16.4665 |
| att532       | 46.208823 | 34.2132 |
| rat783       | 46.455887 | 36.1391 |

5. Conclusions

This paper reviews GA and ACO algorithms which both are two important types of meta-heuristics. Although these meta-heuristics are successful in dealing with optimization problems, both face with some critical defects. These paper lists some of these problems and explains how can eliminate these by utilizing good properties of these two methods in a hybrid way.

According to these, this research propose a hybrid version of ACO in which ants act as greedy genetic operator. Experimental results show that proposed method hybrid method with these smart ants has better performance in both factors of speed and accuracy.

References

[1] Lei, Wang & Dun-bing, Tang, (2011) “An Improved Adaptive Genetic Algorithm Based on Hormone Modulation Mechanism for Job-shop Scheduling Problem”, Expert Systems with Applications, Vol. 38, pp 7243-7250.

[2] Hassan, Ismkhan & Kamran Zamanifar, (2012) “Developing Improved Greedy Crossover to Solve Symmetric Traveling Salesman Problem”, International Journal of Computer Science Issues, Vol. 9, No. 5, pp121-126.

[3] Hassan, Ismkhan & Kamran Zamanifar, (2013) “Study of Some Recent Crossovers Effects on Speed and Accuracy of Genetic Algorithm”, using Symmetric Travelling Salesman Problem”, International Journal of Computer Applications, Vol. 80, No. 6, pp 0975 – 8887.

[4] TSPLIB,http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/.

[5] C, Blum & M. Sampels, (2004) “An ant colony optimization algorithm for shop scheduling problems”, Journal of mathematical modelling and algorithms, Vol. 3, No. 3 pp 285-308.

[6] C.Y. Lee & Z. Lee & S.W. Lin & K. C. Ying, (2010) “An enhanced ant colony optimization (eaco) applied to capacitated vehicle routing problem”, Applied intelligence Vol. 32, pp 88-95.

[7] Hassan, Ismkhan & Kamran Zamanifar, (2010) “Using Ants as a Genetic Crossover Operator in GLS to Solve STSP”, International Conference of Soft Computing and Pattern Recognition, pp 344-348.

[8] Yuichi, Nagata & Shigenobu, Kobayashi, (2012) “A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem”
Authors
Hassan Ismkhan is instructor at the University of Bonab.