An Ant Colony Optimization approach to solve the Minimum String Cover Problem

S. M. Ferdous[1], Anindya Das[2], M. Sohel Rahman[3], Md. Mustafizur Rahman[4]
Department of Computer Science and Engineering[1,2,3,4]
Bangladesh University of Engineering and Technology[1,2,3,4]
Dhaka, Bangladesh
s.m.ferdous@csebuet.org[1], anindya_088@yahoo.com[2], msrahman@cse.buet.ac.bd[3], mustafiz_rahman@cse.buet.ac.bd[4]

***Abstract*—In this paper, we consider the problem of covering a set of strings *S* with a set of strings *C*. *C* is said to cover *S* if every string in *S* can be written as a concatenation of a set of strings which are elements of *C*. We discuss here three different variants of Ant Colony Optimization (ACO) and propose how we can solve the minimum string cover problem using these techniques. Our simulation results show that ACO based approach gives better solution than the existing approximation algorithm for this problem.**

Keywords - Ant Colony Optimization, String Cover, Metaheuristics, Stringology.

I. INTRODUCTION

String Covering problems are very important and mature topics in the field of Stringology having extensive applications in areas like computational biology [1], formal language theory [2], etc. In this paper, we study the Minimum String Covering Problem (MSCP), one of the hardest problems of Stringology. In particular, we propose an algorithm based on the Ant Colony metaheuristic to solve MSC problem.

We have implemented three different versions of ACO metaheuristics, namely, the MAX-MIN Ant System (MMAS), the Ant Colony System (ACS) and the Hybrid Ant Colony System (HAS) to solve MSCP. We have tested HAS with our generated datasets and the result is much better than the existing Approximation Algorithm [1].

II. PROBLEM DEFINITION

In String Cover problem, there is a Target Set$S$. Let $C(S)$ denote all the substrings of S. We say, $C \subset C(S)$ is an l-*cover* of S if each string in S can be written as a concatenation of at most l strings in C. Let, $S = \{abc, cba, bca\}$. So, $C = \{a, b, c\} \subset S$ is a 3-*Cover* of S.

Given a weight function $w: c(S) \rightarrow \mathbb{N}$, a Minimum String Cover (MSC) is an l-cover of S with minimum possible total weight. The total weight of the cover, $W = \sum_{c \in C} w(c)$. Let, n is the number of strings in S, i.e., $n = |S|$ and m is the maximum length of any string of S. If $w(c)$ is a unitary function *i.e.* $w(c) = 1$ for every $c \in C(S)$, then it minimizes the cardinality of cover. For example: Let $S = \{abc, ab, c\}$ and the cost of each string is 1. Then $C = \{ab, c\}$ is the minimum string cover of S with $l = 2$ and $W = 2$.

III. RELATED WORKS

Covering problems are computational problems that ask whether a certain combinatorial structure covers another. Three of the most well known covering problems are the set cover problem, vertex cover problem and edge cover problem. The decision versions of first two are NP-complete [3] whereas the last one can be solved in polynomial time [4].

The MSCP is perhaps the first problem in stringology that incorporate the idea of covering. It is surprising that the topic did not come into surface until 1990 when Neraud [5] studied the problem of determining whether a given set of words is *elementary*. A set of strings, X, is defined to be *elementary* if there exists no set of strings Y with $|Y| < |X|$ and $X \in Y^*$. For example, $X' = \{ABC, BCA\}$ is elementary and $X'' = \{ABC, BCA, A\}$ is not due to $Y = \{A, BC\}$ and $X'' \in Y$.

The MSCP was formally defined and studied in [1], where the MSCP was proved to be NP-Hard. It was further shown that in general, the problem is hard to be approximated within a factor of $c * lnn$ for some $c > 0$, and within $\lfloor m/2 \rfloor - 1 - \varepsilon$ where $\varepsilon > 0$. They also showed that the problem remains *APX-hard* even when m is a constant, and the given weight function is either the unitary or the length-weighted function.

An interesting and recent trend is to solve many hard combinatorial optimization problems using metaheuristic techniques like Ant Colony Optimization (ACO) [9]-[10]. The basic idea of ACO was introduced by Dorigo [6] and successively extended by Dorigo et al. [7]. This metaheuristic was inspired by the foraging behavior of real ants. This behavior—as described by Deneubourg et al. [8]—enables ants to find the shortest paths between food sources and their nest. Initially, ants explore the area surrounding their nest in a random manner. As soon as an ant finds a source of food, it evaluates quantity and quality of the food and carries some of this food to the nest. During the return trip, the ant deposits a pheromone trail on the ground. This pheromone trail guides other ants to the food source and eventually to find the shortest path between their nest and food sources. This functionality of real ant colonies is exploited in artificial ant colonies in order to solve Combinatorial Optimization (CO) problems.

IV. PRELIMINARIES

In ACO algorithm the pheromone trails are simulated via a parameterized probabilistic model, i.e., the pheromone model consisting of a set of parameters whose values are called the pheromone values. The basic ingredient of the ACO algorithm is a constructive heuristic that is used to probabilistically construct solutions using the pheromone values.

In general, the ACO approach attempts to solve a CO problem by iterating the following two steps. At first, solutions are constructed using a pheromone model, i.e. a parameterized probability distribution over the solution space. Then, the solutions that were constructed in earlier iterations are used to modify the pheromone values in a way that is deemed to bias the search towards the high quality solutions.

Ant Based Solution Construction: As mentioned above, the basic ingredient of ACO algorithm is a constructive heuristic to probabilistically construct solutions. A constructive heuristic assembles solutions as sequences of solution components taken from a finite set of solution components $C = \{c1, c2, \ldots cn\}$. A solution construction starts with an empty partial solution $\boldsymbol{s^p} = <>$. Then at each construction step the current partial solution s is extended by adding a feasible solution component from the solution space C. The process of constructing solutions can be regarded as a walk (or a path) on the so-called *construction graph* $G_c = (C, L)$ whose vertexes are the solution components C and the set L are the connections (i.e. edges).

Heuristic Information: In most ACO algorithms the probabilities for choosing the next solution component—also called the transition probabilities—are defined as follows:

$$p(c_i|s^p) = \frac{\tau_i * \eta(c_i)^\beta}{\sum_{c_j \epsilon N(s^p)} \tau^\alpha * \eta(c_j)^\beta}, \forall c_i \in N(s^p) \quad (1)$$

Here, η is a weighting function. Sometimes depending on the current partial solution, it assigns a heuristic value $\eta(c)$ at each construction step to each feasible solution component $c_j \in N(s)$. The values that are given by the weighting function are commonly called the *heuristic information*. Furthermore, α and β are positive parameters whose values determine the relation between pheromone information and heuristic information.

Pheromone Update: In ACO algorithms we can find different types of pheromone updates. First, we outline a pheromone update that is used by basically every ACO algorithm. This pheromone update consists of two parts. First, a pheromone evaporation, that uniformly decreases all the pheromone values, is performed. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region. It implements a useful form of forgetting, favoring the exploration of new areas in the search space. Then, one or more solutions from the current or from earlier iterations are used to increase the values of pheromone trail parameters on solution components that are part of these solutions. As a prominent example, we outline in the following the pheromone update rule that was used in Ant System (AS) [6], which was the first ACO algorithm proposed. This update rule is defined by

$$\tau_i \leftarrow (1 - \varepsilon) * \tau_i + \tau_i * \sum_{s \epsilon G_{iter} | c_i \epsilon s} F(s) \quad (2)$$

for $i = 1,2,\ldots,n$, where G_{iter} is the set of solutions that were generated in the current iteration Furthermore, $\varepsilon \in (0,1]$ is a parameter called the *evaporation rate*, and $F: G \rightarrow \mathbb{R}+$ is a function such that $f(s) < f(s') \Rightarrow F(s) \geq F(s'), s \neq s'$ *for all* $s \in G * F(.)$ is commonly called Fitness Function.

In general, different versions of ACO algorithms differ in the way they update the pheromone values. This also holds for the two currently best-performing ACO variants in practice, namely, the Ant Colony System (ACS) [9] and the MAX-MIN Ant System (MMAS) [11]. In the following we briefly outline the peculiarities of these two algorithms.

Ant Colony System (ACS): The ACS algorithm was introduced to improve over the performance of Ant System (AS). ACS is based on AS but shows some important differences. First, after each iteration, a pheromone update is done using the best solution only (i.e., the best solution found so far). The pheromone evaporation is only applied to the solutions components that are in the best-so-far solution. Second, the transition probabilities are defined by a rule that is called *pseudo-random-proportional* rule. With this rule, some construction steps are performed in a deterministic manner, whereas in others the transition probabilities are defined. Third, during the solution construction the pheromone value of each added solution component is slightly decreased.

MAX-MIN Ant System: MMAS algorithms are characterized as follows. First, the pheromone values are limited to an interval $[\tau_{MIN}, \tau_{MAX}]$ with $0 < \tau_{MIN}, < \tau_{MAX}$. Explicit limits on the pheromone values prevent that the probability for constructing a solution falls below a certain value greater than 0. This means that the chance of finding a global optimum never vanishes. Second, in case the algorithm detects that the search is too much confined to a certain area in the search space, a restart is performed. This is done by reinitializing all the pheromone values. Third, the pheromone update is always performed with either the

iteration-best solution, the restart-best solution (i.e. the best solution found since the last restart was performed), or the best-so-far solution.

V. OUR APPROACH

We have applied ant colony optimization technique to solve minimum string cover problem. We have already discussed that MSCP is a combinatorial optimization problem. ACOs are specially designed for COPs [6].

The whole optimization process can be divided in five (5) steps as follows:

1. Initialization and Configuration
2. Construction of candidate solution
3. Local search (optional)
4. Fitness assessment
5. Pheromone update

We have used 3 variants of ACO, namely, **MAX-MIN Ant System (MMAS)**, **Ant Colony System (ACS)** and a hybrid of MMAS and ACS (**HAS**).

A. MAX-MIN Ant System (MMAS) for MSCP

Initialization and Configuration: The solution components of MSCP are the substrings of S, $C(S)$ generated from the target string set S. In initial step, we set all pheromone to the maximum pheromone value, τ_{MAX}. All other constants like evaporating constant *(ε)*, α , β are set up.

If static heuristic information is used then the total desirability is also calculated. In our problem we have used static heuristic information. It is calculated as the sharing of a component in all target string set. Mathematically,

$$\eta(C_i) = \frac{\sum_j^{|S|} H(C_i, j)}{cost(C_i) * |S|} \qquad (3)$$

Here, $\eta(C_i)$ stands for desirability and $H(C_i, j)$ is defined by,

$$H(C_i, j) = \begin{cases} 1 & if\ C_i\ is\ a\ substring\ of\ S_j \\ 0 & Otherwise \end{cases} \quad (4)$$

The total heuristic of a component is:

$$d_t(C_i) = \rho(C_i)^{\alpha} \eta(C_i)^{\beta} \qquad (5)$$

Here, α and β are tuning parameters that control the effect of pheromone over heuristic information and $\rho(C_i)$ represents the pheromone value of that component.

Construction of candidate solution: Ant colony optimization techniques build solution iteratively. Basically it takes one of the components from candidate components and adds it to the partial solution list. Then it checks whether it is a complete solution or not. It goes on until the solution is complete or no more components are left.

For our problem, we have calculated a discrete probability distribution on the solution space. At each iteration, we build a discrete cumulative probability distribution function over the candidate components, i.e., the components that have not been yet taken in the (partial) solution. An ant k chooses a component j with probability,

$$p_{C_j}^k = \frac{d_t(C_j)}{\sum_{C_i \notin PS^k} d_t(C_i)} \qquad (6)$$

Here, PS^k represents partial solution of ant ***k.***

The construction step continues until it finds a complete solution. In our case after adding a component in the partial solution list it checks whether the partial solution is an *l-cover* of $\boldsymbol{S}$. If the answer is yes, the construction process ends, otherwise it takes another component according to (6).

Fitness Assessment: Fitness (F) is assessed with respect to cost. Each complete solution (CS) are assigned a fitness value which is calculated as follows:

$$F(CS) \propto \frac{1}{\sum_i^{|CS|} cost(CS_i)} \qquad (7)$$

Pheromone Update: In MMAS pheromone of each component is bound to a maximum and minimum value of pheromone $\boldsymbol{\tau_{MIN}}$ and $\boldsymbol{\tau_{MAX}}$, respectively.

After each iteration, the components of iteration best (IB) or global best (GB) solution are updated. First, a bit of pheromone is evaporated and then a fraction of fitness is spread to the components of IB or GB as follows:

$$\rho(C_i) = (1 - \epsilon)\rho(C_i) + \epsilon * F(B), C_i \in B \quad (8)$$

Here, B represents *IB* or *GB*. Now, we limit $\rho(C_i)$ within the range between τ_{MAX} and τ_{MIN}.

$$\rho(C_i) = \begin{cases} \tau_{MIN}, if\ \rho(C_i) \le \tau_{MIN} \\ \tau_{MAX}, if\ \rho(C_i) \ge \tau_{MAX} \\ \rho(C_i), \quad otherwise \end{cases} \qquad (9)$$

B. Ant Colony System (ACS) for MSCP

The initialization and configurations are the same as MAX-MIN Ant System. The only difference is that the pheromone is initialized to τ_{MIN} . The fitness assessments are exactly same as that of MMAS. ACS differs from MMAS in construction and pheromone update steps.

Construction of candidate solution: ACS exploits the so called pseudo-random proportional action choice rule in the solution construction. Like MMAS, ACS also builds the probability distribution function. But, sometimes it just ignores the probabilistic approach and draws the best components according to the total desirability function *(d_t)*. So, if the next component is *j* we have,

$$j = \begin{cases} argmax\ C_i \notin\ PS^K\ d_t(C_i) & if\ q \le q0 \\ drawProb, & otherwise \end{cases} \quad (10)$$

Here, q is a random number taken from a uniform distribution, q_0 is the threshold value and *drawProb* represents the probabilistic approach used in MMAS.

Pheromone Update: In ACS pheromones are updated quite differently than MMAS. There are two types of updates: Local update and global update. Ants modify the pheromone trails also while constructing a solution. Immediately after ant k'-s has added a component C_j to its partial solution PS^k, it modifies $\rho(C_j)$ as follows:

$$\rho(C_j) = (1-\xi)\rho(C_j) + \xi * \tau_0 \qquad (11)$$

Here, $0 \leq \xi \leq 1$, is a parameter and τ_0 is the initial value of the pheromone trail.

C. Hybrid of MMAS and ACS for MSCP

In our hybrid approach we have taken the construction step like ACS but pheromone update step like MMAS [10].

D. Psuedo Code

The detailed pseudo code of our approach is given below:

Algorithm 1: ***ACO for MSCP***

```
Initialize_all_parameter( )
for test 1 to n        //n is the number of tests
  for run 1 to 10
    repeat
      for iteration 1 to ∞
        for ant 1 to 10
          construction(ant)
          if local_search =true
            apply_local_search ()
          end
          update_solution()
          if acs_flag = true
            local_update_pheromone()
          end
        end
        update_pheromone()
      end
    until Best Solution is found or Time is finished or
  no update for 300 iterations
  end
end
```

VI. EXPERIMENTAL SETUP

A. Testbed preparation

We have used 10 workstations of following configurations:

1. Processor: intel Core 2 duo 3.0 GHz
2. RAM: 2 GB
3. OS: Windows XP

We have developed 8 datasets. Each set contains 6 different test cases. Each of the test cases has been run for 10 times. The maximum time limit is allowed for each case is 7200 seconds.

B. Dataset Generation

To the best of our knowledge, there has not been any attempt to apply metaheuristic techniques to solve MSCP in the literature. As a result, we could not find any standard benchmark dataset to test the performance of our algorithm. So, we have generated our dataset as follows. We have chosen a set of sentences as input strings. Each input set contains a set of strings generated from the sentences. For example, let us consider the following sentence (without any punctuation marks): *"Dont go around saying the world owes you a living the world owes you nothing it was here first"*. Our instance generation algorithm takes this kind of sentences as input. Then we use the space as delimiter and find out the words from the sentences. For this example, the *word set (W)* is :*{ Dont, go, around, saying, the, world, owes, you, a, living, nothing, it, was, here, first}*.Then we shuffle the words randomly and append them to make new strings. Such as:

S={
owesgoyoutheherenothingworldDontlivingowesitaround thesayingworldyouawasfirst,
wastheworldarounditherelivinggoyoufirstnothingDonto wesworldyouowessayingthea,
sayingnothingyouowesfirstaherethearoundDontyouowes worlditworldthewasgoliving,
wasarounditworldfirstsayingherelivingtheowesDontnot hingyouthegoayouowesworld,
worldasayingowesnothingitDontherefirstgoyouaroundli vingworldwasyouowesthethe
}

It is expected that, if we intelligently choose l and number of individual letters and use the unitary cost function, the minimum string cover of above *Target Set (S)* would be the cardinality of *Word set (W)*. We have selected about hundreds of sentences from different websites and literature and shuffle them to generate different *Target Sets (S)*.

Table 1: Dataset characteristics

Test No	No. of Test cases	Average No. of words	Average length of a target string	Average cardinality of target sets	Average total length of each target set
1	6	6.67	30.17	6.67	201.24
2	6	11	50.8	7	355.6
3	6	12.3	61.5	8	492
4	6	13.3	67.3	7	471.1
5	6	15.3	77.8	6	466.8
6	6	16.7	76.2	6	457.2
7	6	16.8	97.2	6	583.2
8	6	20.2	95.2	6	571.2

C. Parameters:

The parameters used in our simulation are shown in Table 2. The values of α, β and l are fixed based on the

performance of the algorithm in some preliminary experiments. The value of ρ is kept small to prefer exploration to exploitation.

Table 2: Parameters

Parameter	Value
α	3
β	1 to 4
Cover l	15 to 24
Rate of Evaporation ρ	0.08

VII. EXPERIMENTAL RESULTS

We tested both the Hybrid Ant System and the Approximation algorithm [1] on the generated dataset discussed above. The table shows the results obtained from both the algorithms. Here $cost(expected)$ represents *expected cost* and $cost(simulated)$ represents simulated cost. Here *expected cost* of the input dataset is the cardinality of *Word set (W)*. The error rate is defined as:

$$\frac{|cost(expected) - cost(simulated)|}{cost(expected)} * 100\%$$

Fig. 1 and 2 show the average error rate for both the ACO based approach and the approximation algorithm.

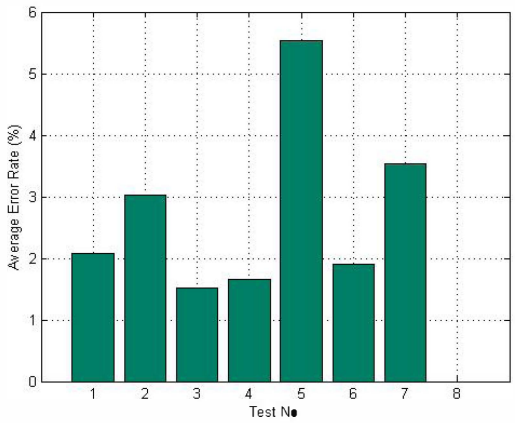

Fig 1: Average Error Rate of HAS

Fig. 3 shows a comparison in terms of solution cost between the two algorithms. It shows that our approach (the last 6 slots along x axis) gives solutions that are almost equal to the expected (middle 6 slots) solutions. From Table 3, we observe that for our algorithm, the standard deviation of the solutions of 10 runs is within 1.1785 which is very small. It reflects the nonvolatile nature of the solutions generated by our approach. Due to space constraint, we can only provide a subset of our experimental findings.

From Table 3 and Fig. 1 to 3, it can be easily observed that our approach is much more superior to the existing approximation algorithm.

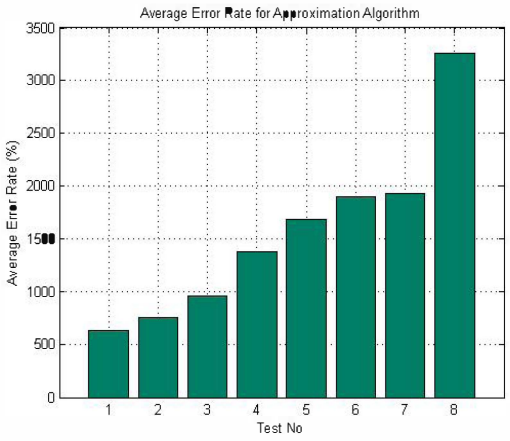

Fig 2: Average Error Rate for Approximation Algorithm

VIII. CONCLUSION

Minimum string cover problem has important applications both in computational biology and formal language theory. In this paper, we have presented metaheuristic solutions to solve the problem. We have used static heuristic information. Adding dynamism in heuristic information can be a good future direction. Calibration is also a great issue in metaheuristic techniques. It is hard to find the optimal values for all the relevant parameters. We have developed 3 variants of ACO. Many other existing variants can be applied to find better solutions.

REFERENCES

[1] Hermelin, Danny and Rawitz, Dror and Rizzi, Romeo and Vialette, "The minimum substring cover problem", in *Proc. of the 5th international conference on Approximation and online algorithms,* 2007, pp. 170-183.

[2] H. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett and H. T. Wareham, "Parameterized Complexity Analysis in Computational Biology", *J. of Comput. Appl. Biosci*, vol.11,1995,pp.49-57.

[3] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations,* Plenum Press, R. Miller and J. Thatcher Ed. 1972, pp.85-103.

[4] Garey, R. Michael and Johnson and S. David, "Computers and Intractability; A Guide to the Theory of NP-Completeness", New York, NY, USA, W. H. Freeman & Co.,1990.

[5] J. Neraud, "Elementariness of a finite set of words is co-NP-complete", *J. of ITA,* vol.24, 1990, pp.459-470.

[6] M. Dorigo, "Ottimizzazione, apprendimento automatico, ed algoritmi basatisu metafora naturale.", PhD thesis, Politecnico di Milano, Milan, Italy, 1992.

[7] Dorigo, Marco and Di Caro, Gianni and Gambardella, M. Luca, "Ant algorithms for discrete optimization", *J. of Artificial Life,* vol.5,issue.2, April,1999, pp.137-172.

[8] Deneubourg, Jean-Louis and Aron, Serge and Goss, Simon and Pasteels, M . Jacques, "The self-organizing exploratory pattern of the Argentine ant", *J. of Insect Behavior,* vol.3:2,1990, pp.159-168.

[9] Marco Dorigo and Luca Maria Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", *J. of IEEE Transactions on Evolutionary Computation,* vol.1,1996, pp.53-66.

[10] Lessing, Lucas and Dumitrescu, Irina and Stützle, Thomas, "A Comparison Between ACO Algorithms for the Set Covering Problem", Ant Colony Optimization and Swarm Intelligence, Springer Berlin / Heidelberg, vol.3172, pp. 105-122.

[11] Utzle, Thomas and Hoos, H Holger, "MAX-MIN Ant System", *J. of Future Gener. Comput. System,* vol.16:9, 2000, pp. 889-914.

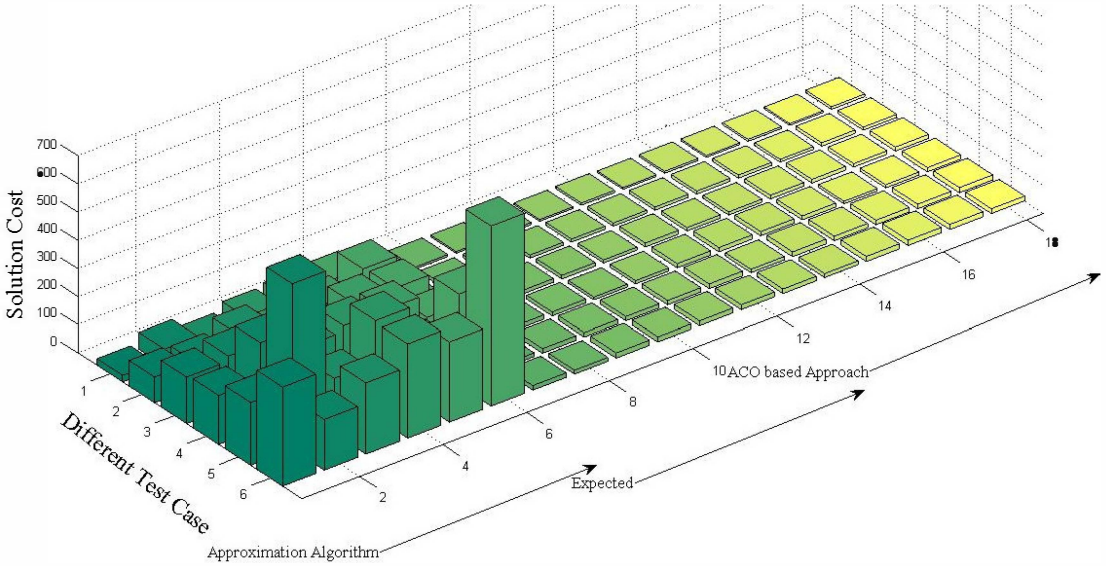

Fig 3: Comparison between approximation algorithm and ACO based approach

Table 3: Summary of Experimental Results

Test	Case	Expected Solution Cost	Minimum Solution Cost		Mean of Solution Cost of HAS	Standard Deviation of Solution Cost of HAS	Number of Iteration	
			Approximation Algorithm	HAS			Approximation Algorithm	HAS
A	1	6	23	6	6.8	0.63246	6	1
	2	7	59	7	8.4	0.84327	7	3
	3	6	24	6	6.4	0.5164	6	3
	4	7	53	7	8.5	1.1785	7	3
B	1	11	91	11	11.9	0.31623	11	9
	2	11	85	11	11.5	0.52705	11	3
	3	11	128	11	11.7	0.48305	11	1
	4	11	85	11	10.9	0.56765	11	5
C	1	12	165	12	12.5	0.52705	12	4
	2	11	183	11	11.2	0.42164	11	2
	3	13	180	13	13.8	0.42164	13	11
	4	13	159	13	13.7	0.48305	13	1
D	1	15	174	15	15.7	0.48305	15	1
	2	15	304	15	15.8	0.42164	15	2
	3	14	230	14	14.8	0.42164	14	2
	4	10	250	10	9.1	0.31623	10	2
E	1	13	223	12	12.5	0.70711	13	26
	2	16	586	17	17.6	0.5164	16	18
	3	16	195	16	17.2	0.63246	16	26
	4	15	342	16	16.4	0.5164	15	11
F	1	16	352	16	16.1	0.31623	16	15
	2	14	180	14	14.8	0.42164	14	3
	3	18	253	18	18.9	0.56765	18	25
	4	18	334	19	19.5	0.52705	18	6