Optimizing Meta-Heuristics for the Time-Dependent TSP Applied to Air Travels

Diogo Duque[1], José Aleixo Cruz[1][0000−0002−5808−576X], Henrique Lopes Cardoso[1,2][0000−0003−1252−7515], and Eugénio Oliveira[1,2][0000−0001−9271−610X]

[1] Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
[2] Laboratório de Inteligência Artificial e Ciências dos Computadores (LIACC)
{up201406274, up201403526, hlc, eco}@fe.up.pt

Abstract. A travel agency has recently proposed the Traveling Salesman Challenge (TSC), a problem consisting of finding the best flights to visit a set of cities with the least cost. Our approach to this challenge consists on using a meta-optimized Ant Colony Optimization (ACO) strategy which, at the end of each iteration, generates a new "ant" by running Simulated Annealing or applying a mutation operator to the best "ant" of the iteration. Results are compared to variations of this algorithm, as well as to other meta-heuristic methods. They show that the developed approach is a better alternative than regular ACO for the time-dependent TSP class of problems, and that applying a K-Opt optimization will usually improve the results.

Keywords: Traveling Salesman Problem · Air Travel · Ant Colony Optimization · Meta-Optimization.

1 Introduction

We have reached an age where air traveling is easy and accessible. With the expansion of air travel comes a substantial increase in the number of available flights. Travel agencies help people find the best and cheapest flights for a journey. Their search engines chew through massive quantities of data to determine the suitable itinerary for a client. If the traveler only needs to go from a city to another, finding the fittest solution is not that hard. But if the journey consists of more than one destination, obtaining the optimal combination of flights becomes more difficult. Kiwi is a Czech online travel agency that has a feature on their website which allows a user to search for a multi-city itinerary. They have launched a Traveling Salesman Challenge (TSC) [14] in 2017 with the purpose of perfecting this feature. That challenge was the motivation for this work, which proposes a combined algorithm with meta-optimization of parameters.

The TSC consists of solving a seemingly simple problem: given a set of cities to visit and a list of flights with the respective origin, destination, price and day, find the combination of flights with the lowest price that visits each city once and returns to the first city. Participants have 30 seconds to find a solution. For

higher simplicity, Kiwi defined a couple of restrictions: (a) on each day you have to board exactly 1 flight, and (b) flights are immediate, *i.e.*, they take no time.

By such a description, we can immediately associate this problem with the classic Traveling Salesman Problem (TSP). In TSP, we try to determine the path with the least cost that visits a certain amount of cities only once and returns to the origin city. In graph theory, that is the equivalent of finding the Hamiltonian Cycle with the lowest cost. Nonetheless, the challenge's conversion to a TSP is not as simple as it seems, as the price of a plane ticket does not depend only on the origin and destination, but also on the date of the flight. Also, flights may not even exist in some days. The TSC is, in fact, a Time Dependent TSP.

This problem can be addressed more formally as follows. Let there be n cities to visit. The number of days d to visit all cities is equal to n, because of the first restriction. For each day, there is a different set of available flights and respective prices between the cities. The cost of a trip from city i to j in day t is $c_{i,j}^t$, where $1 \leq t \leq d$. A valid path is one that starts at city A, visits all cities, returns to city A, and only consists of existing flights. Let $x_{i,j}^t$ be a binary variable that has a value of 1 if in day t the flight from i to j was taken, or 0 otherwise. The objective function to minimize is defined as $\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{t=1}^{n} c_{ij}^t \cdot x_{ij}^t$. Just as the regular TSP, the proposed challenge is NP-hard. Thus, solving it for a large number of cities using exact algorithms is not feasible. Most approaches use meta-heuristics to find a "good enough" solution in a timely fashion.

The central objective of this work is to implement a meta-heuristic to find the best solution to Kiwi's Traveling Salesman Challenge [14] in 30 seconds. The rest of the paper is organized as follows. Section 2 reviews related work and frames our work within existing literature. Section 3 describes our approach to the TSC, including optimizations. Section 4 focuses on experimental evaluation. Finally, Section 5 concludes the paper and points directions for future work.

2 Literature Review

In the 19th century, the problem of finding a Hamiltonian cycle on a graph was mathematically formulated by Hamilton [2]. The Hamiltonian cycle is a well-known reduction of the TSP. Some of the first techniques to be used to solve the TSP include Branch and Bound [15], the Christofides algorithm [6] and the 2-Opt [17].

Gambardella and Dorigo, who wrote the original paper on the Ant Colony Optimization (ACO) meta-heuristic, combined ACO and Q-Learning to calculate the optimal solution for a TSP instance [10]. Results indicated that this procedure was better than some other popular approaches, such as Simulated Annealing. ACO has been compared favorably to other heuristic approaches [19, 20]. More recently, Mohsen [18] used ACO with a particular operator that combines Genetic Algorithms (GA) and Simulated Annealing (SA) to solve the TSP. The operator was used to control ant diversity, a technique that has shown improved performance when compared to a standard ACO implementation.

As meta-heuristics generally involve setting a few parameters that affect their performance, it is necessary to tune them for a particular problem. However, there is no rule for the tuning. *Meta-optimization* consists of using an optimization method to tune another optimization method. It has been used to improve, for example, the performance of genetic algorithms [9].

Inspired by other works, instead of creating a new meta-heuristic, we adopted Mohsen's mixed approach [18] to solve the TSC. We also perform meta-optimizations using genetic algorithms [9] and local search by applying K-Opt moves [12]. With this work, we intend to improve Mohsen's meta-heuristic performance and develop an even faster algorithm for the TSP and for Kiwi's TSC.

3 Methodological approach

A TSP is commonly represented using graph theory. In an air travel context, each node represents a city and each edge a flight. The final solution is a set of ordered flights, expressing the trips to do. In the case of the TSC, however, given the time-dependent nature of trip prices, we need to add a restriction stating that, on a given day, only available flights can be used to travel between cities. We do this by representing a city with different nodes for each day.

We can also formalize the TSP as a scheduling problem [3], by using the $\alpha|\beta|\gamma$ notation [11]: $1|s_{ij}|C_{max}$. There is a single machine, the traveler. A job is a city to visit: job j is characterized by its processing time p_j and by a setup time s_{ij} that represents the amount of time needed to setup job j after job i. The setup time of a job depends on the job that precedes it. The processing time is the cost of remaining in the city. Since it is not relevant to this particular problem, every processing time is equal to a constant that we disregard. The setup time is the cost of going from city i to city j. The objective function is to reduce the overall make-span C_{max}, that is, the total cost of executing every job (which in our case corresponds to the sum of flight prices).

Although this formalization is correct for a regular TSP, it does not fit the TSC because of time-dependencies: going from city i to city j on day x may have a different price than doing so on day y. Thus, the setup time s_{ij} is dependent on the slot t in which the machine executes the job. This t variable corresponds to the day of flight in the original problem proposed by the Kiwi travel agency. A more appropriate formulation is: $1|{s_{ij}}^t|C_{max}$. Instead of a single cost matrix, there are t matrices, one for each day.

3.1 Algorithms

Meta-heuristics can be applied to solving all kinds of optimization problems. We have adopted Simulated Annealing and Ant Colony Optimization as our chosen meta-heuristics. For comparison, we also implemented Greedy search (Basic Hill Climbing) and Backtracking as a deterministic algorithm.

Simulated Annealing Many algorithms are greedy and tend to get stuck in local optima (such as the well-known hill-climbing algorithm). Simulated Annealing (SA) is a well-known probabilistic technique for finding the global optimum. SA comprises a local-search optimization procedure and includes a temperature parameter: the higher it is, the higher the chance of accepting a worse neighboring state for the next iteration of the algorithm. After each iteration, temperature decreases. SA will start to look for solutions in a vast space, but with time it will search in an increasingly small space until the temperature gets close to 0.

Ant Colony Optimization Ant Colony Optimization (ACO) is a rather common approach to solving the TSP [7, 16], based on the concept of collective intelligence of an ant colony. Every ant tries to find the best path to the food, and the better the path, the more pheromones will be laid on it. These make certain trails more "attractive", causing a convergence to better paths. While ACO can be seen as a probabilistic approach, it explores the search space using a set of individuals. It has some similarities with genetic algorithms, but while the latter combines individuals through crossover and mutation operators, ACO explores synergies between individuals through environment-based communication.

Backtracking Backtracking [1] is a general deterministic algorithm whose strength is that it does not need to explore the entire search space. Whenever Backtracking finds itself in a branch that does not lead to an improvement on the current best solution, it moves on to the different branch.

3.2 Optimizations

Discovering new valuable meta-heuristics is very difficult. For this work, we focus on optimizing a meta-heuristic's behavior in order to improve its performance.

Parallelized Ant Colony Parallelization is often a good solution to speed execution time, if the program has parts that can be parallelized, that is, executed independently. In ACO, the ant's path generation (line 8 of Algorithm 1) can be parallelized, as each ant's path is generated on its own, needing only the graph and the pheromone trails. Pheromone updates occur on a single thread. The computer we ran the simulation on had a dual-core CPU with hyper-threading; as such, we defined a thread pool of 4, so as to match the maximum number of threads available on the CPU. A larger thread pool would mean that one or more *CPU threads* would need to handle the extra *software thread*, thus limiting the total speed of the algorithm.

Ant Colony with Simulated Annealing Many so-called hybrid metaheuristics [5] have been proposed in the past, and the optimization described here can be seen as one such attempt. This approach [18] is a variation of the original ACO, with a twist at the end of each iteration: if the population *diversity* is

Algorithm 1: Ant Colony Optimization with Simulated Annealing

```
1  function antColonySA (g, n, ph_w, vis_w, ev_f, q, ini_ph_lvl, temp_dec,
     iter_per_temp);
   Input  : graph g, number of ants n = 30, pheromone weight ph_w = 0.5,
            visibility weight vis_w = 0.5, evaporation factor ev_f = 0.8, q = 1000,
            initial pheromone level ini_ph_lvl = 20, temperature decrease
            temp_dec = 0.05, initial temperature init_temp = 1 and number of
            iterations per temperature iter_per_temp = 1
   Output: the path with the least cost obtained
2  startTimer();
3  best_cost ← +∞;
4  best_path ← null;
5  initializePheromones(ini_ph_lvl);
6  while not timerFinished() do
7      createAnts(b);
8      moveAnts(g, ph_w, vis_w);
9      new_paths ← getAntsPaths();
10     updatePathPheromones(new_paths, ev_f, q);
11     diversity = (averageCost − bestCost) / (worstCost − bestCost);
12     if diversity > 0.7 then
13         simulatedAnnealing(getBestAnt(), temp_dec, iter_per_temp, init_temp);
14     end
15     else
16         mutate(getBestAnt());
17     end
18     updatePathPheromones(getBestAnt().getPath(), ev_f, q);
19     if getBestAnt().getCost() < best_cost then
20         best_cost ← getBestAnt().getCost();
21         best_path ← getBestAnt().getPath();
22     end
23 end
24 return best_path;
```

high, simulated annealing is applied to the best-performing ant and then its pheromones are applied to the paths as usual; if the population diversity is low, a mutation is performed on that ant (swapping the order of two random adjacent cities) and the pheromones are also applied. By intensifying the pheromone trail of a globally good solution through the application of SA to the best ant, attention is concentrated on that search area, reducing dispersion. The mutation operation brings a reinforcement of random paths, increasing the level of exploration by opening up the search space. Algorithm 1 shows our implementation.

Meta-optimizing using Genetic Algorithms The amount of adjustable parameters resulting from the combination of ACO and SA is too high to manually test all possible configurations. Optimizing these values is, in itself, a new problem that we tackle using a genetic algorithm. Chromosomes have one gene for

Algorithm 2: Meta-optimization using Genetic Algorithms

```
1  function metaOptimization (g, n);
   Input  : the graph g and the number of generations n
   Output: the meta-optimized set of parameters
2  randomly initialize population p;
3  for i = 1 to n do
4  |  while processing time < maximum processing time do
5  |  |  for each individual i in population p do
6  |  |  |  fitness(i) = antColonySA(i.getParameters());
7  |  |  end
8  |  end
9  |  generate new population based on fitness values;
10 end
11 return best individual's parameters;
```

each parameter. The fitness function indicates the performance of the ACO-SA algorithm when using the values of the chromosome. We use the average result of ten runs for the fitness function. The lower the average cost obtained, the greater the value of fitness. The procedure we use for this genetic algorithm is listed in Algorithm 2. Running meta-optimization is costly, but it only needs to be performed once. Since Kiwi's challenge is mostly about getting the best result in a short period of time, by already having good parameters we are increasing our chances of finding the optimal solution.

Solution optimization with K-Opt K-Opt [12] is a local search algorithm involving swap moves over a previously-obtained path. K is the number of edges to delete, creating $K + 1$ sub-tours. Apart from the first and last sub-tour, all others are reordered and/or reversed to try to find a better global tour. The algorithm attempts all possible combinations of removed edges (that still preserve K+1 sub-tours) to find the best combination of reordering and/or reversing. We used $K \in \{2, 3\}$ since, even though the algorithm runs in polynomial time $T(n^K)$, the running time for higher K in heavier datasets is too high. 3-Opt usually gets within 3-4% of an optimal tour [12], so it ensures an already rather good proximity to the optimal solution.

4 Experimental evaluation

In order to assess the merits of each optimization approach (described in Section 3.2) in enhancing the performance of the heuristics, we compare the results obtained with and without such optimizations. The dataset used was the one provided by Kiwi for the challenge [13], which is a comma-separated values (CSV) file. The first line contains the string corresponding to the origin city. The following lines contain the available flights, according to the syntax $origin_city, destination_city, day, price$.

We tested the heuristics for several dataset sizes of up to 100 cities. Results shown in Table 1 are all relative to one single documented run. However, in most of the undocumented runs we executed, the scenario was the same. According to Kiwi's challenge, the output route of an execution is the best route found in 30 seconds. After this time, the execution of the algorithm terminates.

Table 1. Results of the algorithms in one run with 15, 60 or 100 cities.

	15 cities		**60 cities**		**100 cities**	
Algorithm	**Lowest cost**	**Iterations**	**Lowest cost**	**Iterations**	**Lowest cost**	**Iterations**
Backtrack	5268	-	29387	-	53202	-
Greedy	4801	15	12073	60	17893	100
SA	4344	5596770	15450	319760	20116	66810
ACO	4464	19218	10396	652	17002	150
ACO (p)	4500	23623	10340	1213	16614	237
ACO-SA	4385	22189	10213	696	16156	165
ACO-SA (p)	4385	20740	10551	630	16452	145
ACO-SA (m)	4385	15538	10007	665	16206	150
ACO-SA (p) (m)	4385	16660	10374	670	16037	162

We start by comparing the performance of a regular version of Ant Colony Optimization with other heuristics, such as Simulated Annealing and Hill Climbing (Greedy), but also exact algorithms like Backtrack. These algorithms gave us a control group to compare the performance of our approach. Figure 1 demonstrates how the ACO variations behave for different dataset sizes.

For datasets with a number of cities smaller than 15, the results achieved by brute-force and heuristic algorithms were identical. Table 1 documents the differences that arise as the search space grows larger. As expected, with the increase in the number of cities, Backtrack and Hill Climbing struggled to find satisfactory solutions in the 30s time window (in fact, the running time needed for Backtracking to run with 15 cities is of 7m16s, compared to 155ms for 10 cities and 1ms for 5 cities). Ant Colony Optimization proved better than Simulated Annealing in all sizes. The parallelization of the Ant Colony Optimization proved valuable in increasing the number of iterations the algorithm could perform before running out of time, although it did not bring a much better result, leading us to conclude that convergence was already achieved most of the times.

The Simulated Annealing tweak makes an iteration take longer to execute. Because of the 30 seconds time limit, ACO-SA makes less iterations than the regular ACO. However, ACO-SA consistently leads to a better solution than ACO, as seen in Figure 1. The control of the ants diversity proved to have a great effect in the speed of convergence. ACO-SA found better intermediate solutions than ACO faster, so its performance is effectively better than ACO.

Parameter optimization through genetic algorithms was done by targeting a dataset with 60 cities. For datasets with this approximate size, meta-optimization did improve the results of the algorithm. In datasets of significantly different sizes, however, it did not behave as well as we thought, being worse than the man-

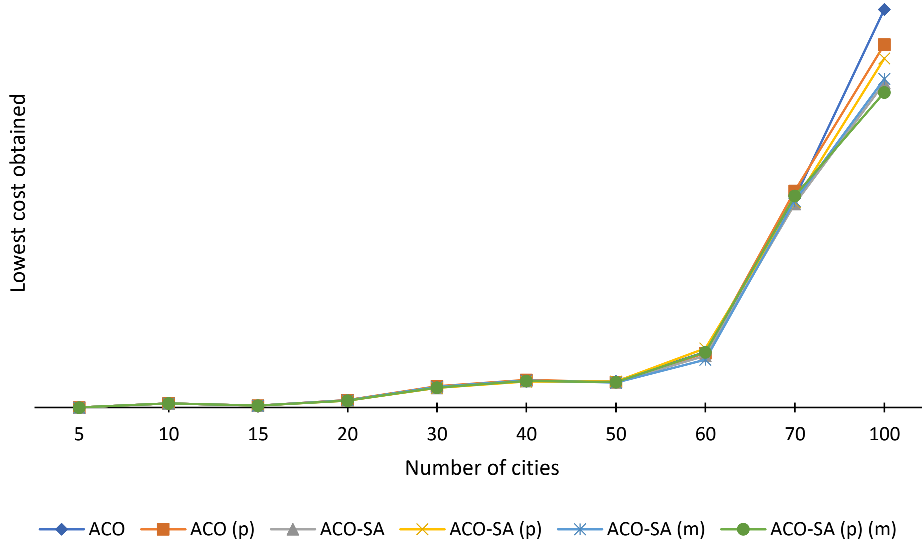

Fig. 1. Lowest cost obtained for different dataset sizes. (p) means that the algorithm has been parallelized. (m) means that meta-optimization was applied. The values have been raised to the power of 4 to better demonstrate how algorithms behave as the search space gets larger.

ually defined parameters. We believe that by redefining the meta-optimization strategy to take into account datasets of different sizes the results would have been better.

Applying K-Opt on the final solution allows us to find even better solutions. The actual value of using the K-Opt technique is better for a number of cities between 10 and 80, as can be observed by crossing information in Figure 2. In smaller datasets, the difference K-Opt makes is not so noticeable because there is not much room for improvement. In larger datasets, and given the fact that the time complexity of a K-Opt execution for n cities is n^K, 3-Opt reveals itself as inappropriate due to the time it takes to run. Thus, we can conclude that the usage of the K-Opt technique should be targeted mainly to medium-sized datasets, and only 2-Opt should be used in larger datasets so as to avoid a significant increase in execution time.

K-Opt's running time does not count to the 30 seconds total of the challenge. As their runtime is constant (for a constant dataset size), we assume that anyone planning on using it could reduce the main algorithm's running time by a calculated margin that would allow for the optimization to be executed. As such, when comparing our main algorithms in all datasets, we chose to keep the 30s total so as to not affect the comparison between every result.

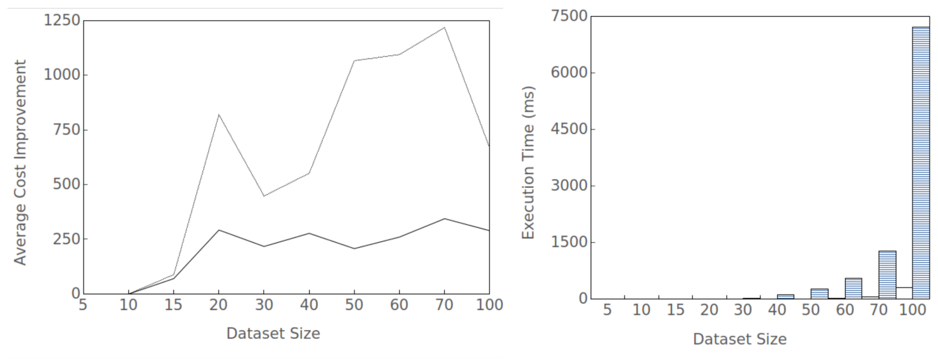

Fig. 2. Average cost improvement by applying K-Opt (left) and execution time of 2-Opt and 3-Opt on the final solution of ACO-SA (right).

5 Conclusions

Combining Ant Colony Optimization with Simulated Annealing proved to have better performance than the regular ACO version to solve Kiwi's multi-city air travel challenge. Applying the K-Opt technique to an already good solution is useful mostly for medium sized datasets, because of the trade-offs between solution improvement and computation time.

Our experiments in applying meta-optimization to the ACO-SA did not improve the results much, but we believe that by slightly altering our approach we could achieve better results. Since the optimization was made using a fixed dataset size, it was not so effective on the other sizes, which means that if we had used datasets with different sizes, the result might have been better. We also found that Backtracking could be useful in a real-life context (which uses few cities), but clearly is not enough in the TSC.

In the future, an improvement that could be made would be to use racing conditions in the meta-optimization [4]. Instead of spending resources evaluating every single generated individual, we would drop the least promising candidates during the evaluation process, focusing on the best individuals. Another improvement could be the parallelization of 3-Opt so that it becomes faster to execute and thus less cumbersome to use with larger datasets. The code used for the project is hosted in a GitHub repository. [8].

References

1.
2. N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, New York, NY, USA, 1986.
3. L.-P. Bigras, M. Gamache, and G. Savard. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, 5(4):685 – 699, 2008.

4. M. Birattari. The problem of tuning metaheuristics: as seen from the machine learning perspective. Master's thesis, Université libre de Bruxelles, 2005.
5. C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels. *Hybrid Metaheuristics: An Emerging Approach to Optimization*. Springer, 1st edition, 2008.
6. N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. page 10, 02 1976.
7. M. Dorigo and G. Di Caro. Ant colony optimization: A new meta-heuristic. In *Procs. Congress on Evolutionary Computation*, pages 1470–1477. IEEE Press, 1999.
8. Diogo Duque and Jose Aleixo Cruz. https://github.com/jazzchipc/MPE-FEUP, 2018.
9. J.A. Fernandez-Prieto, J. Canada-Bago, M.A. Gadeo-Martos, and J. R. Velasco. Optimisation of control parameters for genetic algorithms to test computer networks under realistic traffic loads. *Applied Soft Computing*, 12(7):1875–1883, 2012.
10. L. M. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *Procs. 12th Int. Conf. on Machine Learning*, ICML'95, pages 252–260. Morgan Kaufmann Publishers Inc., 1995.
11. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979.
12. D. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*. Wiley, 1997.
13. Kiwi.com. Rules for kiwi.com travelling salesman competition. https://github.com/kiwicom/travelling-salesman, 2018. [Accessed 29-June-2018].
14. Kiwi.com. Travelling salesman challenge. https://travellingsalesman.cz, 2018. [Accessed 29-June-2018].
15. P. J. Kolesar. A branch and bound algorithm for the knapsack problem. *Management Science*, 13(9):723–735, 1967.
16. B. Li, L. Wang, and W. Song. Ant colony optimization for the traveling salesman problem based on ants with memory. In *2008 Fourth International Conference on Natural Computation(ICNC)*, pages 496–501, 2008.
17. S. Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, Dec 1965.
18. A. M. Mohsen. Annealing ant colony optimization with mutation operator for solving tsp. *Computational Intelligence and Neuroscience*, 2016:13, 2016.
19. H. Mukhairez and A. Maghari. Performance Comparison of Simulated Annealing, GA and ACO Applied to TSP. *Int. Journal of Intelligent Computing Research*, 6(4):647–654, 2015.
20. V. Selvi and R. Umarani. Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques. *Int. Journal of Computer Applications*, 5(4):1–6, 2010.