Optimization System in Networks using ACO

P. Cardoso[1], M. Jesus[2] and A. Márquez[3]

Abstract

The Ant Colony Optimization Method (ACO) concept is used as a support to the development of a system to model/simulate network problems. Its goal is the capability to estimate solutions even when some dynamism is inherent to problem.

The ACO system is based on an agent's population with the goal of finding a suitable solution (random-heuristic), considering the paths between origin and target nodes.

To improve the performance and the quality of the solutions, the system must have a collection of capabilities, namely:

- memory (to pass situations aiming the prevision of future occurrences);
- global *vision* over the network (capability to recognize critical points in the network and find solution taking those in account)
- capability to communicate.

1 Introduction

The challenge of solving hard optimization problems, many times leads to the use of classical methods that often come across great difficulty to find the optimal solution, although theoretically they guarantee it.

Evolutionary and adaptive computing have emerged in the last decades, taking advantage of the advent of the computer age and furnishing us the capability to solve some hard problems in a large domain of fields [1, 8]. Many of this systems, usually based on random-heuristic searches, can be well described by analogy to various natural systems [1, 8, 9, 12] and proved to be very attractive since they can be adapted to a wide range of problems.

In this paper, the global optimization system Ant Colony Optimization Algorithm (ACO) was used to set up a system, capable of finding the k shortest paths between two given nodes in a network. The optimization process launches a set of agents that, taking in account information leaved by other agents (pheromone trail) and some heurist (distance to the terminal node), randomly constructs a path between those nodes. The best k solutions from the ones constructed during the search are kept.

In Section 2 we will describe the network model used, the k Shortest Path Problem and the ACO Algorithm. In Section 3 the formulated system is explained along width some tests and results, finishing with conclusions and further work.

2 Preliminares

2.1 Network Model and k-Shortest Path

In this paper, let us consider a network as

$$\mathcal{N} = (\mathcal{V}, \mathcal{E}, l)$$

where, $\mathcal{N}$ is the set of nodes (points in the plane), $\mathcal{E}$ is the set of edges (each one defined by a pair of nodes) and $l : \mathcal{E} \rightarrow \mathbb{R}^+$ is a function such that, for $e \in \mathcal{E}$, $l(e)$ is the Euclidean distance between the nodes that define e.

The k Shortest Path Problem consists on the determination of a set $\{p_1, p_2, \ldots, p_k\}$ of paths between a given pair of nodes, when the objective function of Shortest Path Problem is considered. The following classes of the shortest path are usually considered:

- the k Shortest Disjoint Paths (arc or node disjoint);

[1]Universidade do Algarve (EST). Dep. Eng. Electrotécnica. Email: pcardoso@ualg.pt

[2]Universidade do Algarve (EST). Dep. Eng. Civil. Email: mjesus@ualg.pt

[3]Universidad de Sevilla (FIE). Dep. de Matemática Aplicada I. Email: almar@us.es

- the k Shortest Elementary Paths (no node is repeated);
- the General k Shortest Path Problem.

Deterministic algorithms to find the k shortest paths are known with $O(m+n\log n+k)$ time for a digraph with $|\mathcal{V}|=n$ nodes and $|\mathcal{E}|=m$ edges [6].

2.2 ACO Algorithms

In the evolutionary and adaptive algorithms one of the most recent is the Ant Colony Optimization (ACO) computational paradigm introduced by Marco Dorigo [5]. As the name suggests, this algorithms have been inspired in the real ant colonies behavior, in particular by their pillager behavior, meaning that when a good source of food near the nest is found, the colony turn is attention to it, until it's exhaustion or a better one is found.

In this manner, the basic idea of the ACO artificial system, is to resemble the cooperative behavior of the colony using:

- a colony of agents called artificial ants;
- indirect communication between the ants, supported on artificial pheromone trails, that reflect the experience of past agents while solving the problem;
- some heuristic to improve the search.

Algorithm 1 ACO - Ant Colony Optimization Algorithm

```
1: Initialize the pheromone trail.
2: while stopping criterion not is not met do
3:   for all ants do
4:     Construct a new solution using the current
       pheromone trail.
5:     Evaluate the solutions constructed.
6:   end for
7:   Update the pheromone trail.
8: end while
```

The ACO algorithm (Algorithm 1) was first introduced to solve the Traveling Salesman Problem (TSP)[4, 5]. The ACO-TSP comprises a set of cycles. In each cycle, all the (artificial) ants try to find a minimal length closed tour that visits each city once. To do it, each ant, randomly placed in a node, constructs the tour by the ordered addition of feasible (i.e., not yet visited) nodes to the tour, considering two measures:

- the closeness of all feasible nodes to the current one (visibility);
- the (artificial) pheromone trail in the edge between the current node and all the feasible nodes.

Mathematically, if the ant is in node u the probability of moving to the feasible node v is:

$$p_{uv}=\begin{cases}\dfrac{\tau_{uv}^{\alpha}\left(\frac{1}{d_{uv}}\right)^{\beta}}{\sum_{w\notin\mathcal{U}}\tau_{uw}^{\alpha}\left(\frac{1}{d_{uw}}\right)^{\beta}} & \text{if } v\notin\mathcal{U}\\ 0 & \text{otherwise}\end{cases}$$

where $\mathcal{U}$ is the set of already visited nodes, τ_{uv} is the quantity if pheromone in the (u,v) edge, d_{uv} is the Euclidean distance between nodes u and v and $\alpha,\beta\in\mathbb{R}_0^+$ are parameters that emphasis the relative importance of the pheromone trail and the heuristic (visibility), respectably.

After each cycle, i.e., after each ant has determined a tour, the pheromone trail is updated using the founded solutions in the following way:

$$\tau_{uv}=\rho\tau_{uv}+\Delta\tau_{uv} \tag{1}$$

with $0<\rho\leq 1$ the trail persistence ($1-\rho$ is trail evaporation) and

$$\Delta\tau_{uv}=\sum_{k\in\{\text{ant}_1,\text{ant}_2,\ldots,\text{ant}_m\}}\Delta\tau_{uv}^{k} \tag{2}$$

where $\Delta\tau_{uv}^{k}$ is the contribution of the ant_k to the pheromone trail between nodes u and v, $\Delta\tau_{uv}$. Usually,

$$\Delta\tau_{uv}^{k}=\begin{cases}\dfrac{Q}{L_k} & \text{if } (u,v) \text{ belongs to the}\\ & \text{tour contructed by ant}_k\\ 0 & \text{otherwise}\end{cases}, \tag{3}$$

where Q is a constant related to the amount of pheromone laid by ants and L_k the length of the tour constructed by ant_k.

The ACO system allows solving other large combinatorial problems like: the Vehicle Routing Problem (VRP)[7, 3], the Quadratic Assignment Problem (QAP) [5] and the Job-shop Scheduling Problem (JSP) [5].

In conclusion, some of the ACO main features are: versatility, robustness and being population based (which straightforwardly enables parallel implementations) [5]. This desirable properties are counterbalanced by the fact that, for some applications, the ACO can be outperformed by more specialized algorithms.

3 k Shortest Path Problem using ACO

3.1 Applying ACO algorithm to the k Shortest Path Problem

In this section we are going to considerer a network $\mathcal{R} = (\mathcal{N}, \mathcal{E}, l)$ (as defined in Section 2.1). In the ACO algorithm for the k Shortest Path Problem (ACO-KSPP), each cycle comprises the *launching* of a set of ants from the starting node s. Each ant constructs a path between that starting node and a terminal node, t. This construction is guided by the pheromone trail, left by past ants, and by a distance to the terminal node heuristic, i.e., being in a node u and if $(u, v) \in \mathcal{E}$, the probability of moving to node v, is computed considering the pheromone in the (u, v) edge and the Euclidean distance from v to the terminal node t. Mathematically, the probability is given by:

$$p_{uv} = \frac{\tau_{uv}^{\alpha} \left(\frac{1}{d_{vt}}\right)^{\beta}}{\sum_{\substack{(u,w) \in \mathcal{E} \\ w \notin P}} \tau_{uw}^{\alpha} \left(\frac{1}{d_{wt}}\right)^{\beta}} \qquad (4)$$

where, P is a ordered list containing the path being constructed, τ_{uv} is the pheromone present in the (u, v) edge, d_{uv} is the Euclidean distance between the nodes u and v and $\alpha, \beta \in \mathbb{R}_0^+$ are parameters which determine the relative influence of the pheromone trail and the heuristic. After each cycle, the pheromone in the edges is updated using formulas (1) and (2) (in formula (3) L_k is considered as the length of the path constructed by ant_k).

The algorithm for the ACO-KSPP is Algorithm 2 and for the construction of the path by the ants is Algorithm 3.

Algorithm 2 k Shortest Path Problem using ACO algorithm

Require: $kpaths$ {number of short paths to calculate},
$\mathcal{N} = (\mathcal{V}, \mathcal{E}, l)$ {network}
$s, t \in \mathcal{V}$.
Ensure: k shortest paths between s and t.
1: Initialize the pheromone trail.
2: $S = \emptyset$ {set of calculated paths.}
3: $npaths = 0$ {number of calculated paths.}
4: **while** stopping criterion not met **do**
5: **for all** ants **do**
6: Construct a new path, p_{st}, between nodes s and t using algorithm 3.
7: **if** $p_{st} \notin S$ **then**
8: **if** $kpaths > npaths$ **then**
9: $S = S \cup \{p_{st}\}$
10: **else**
11: Find $q_{st} \in S : ||q_{st}|| = \max_{r_{st} \in S} ||r_{st}||$
12: **if** $||q_{st}|| > ||p_{st}||$ **then**
13: $S = (S \cup \{p_{st}\}) - \{q_{st}\}$ {replace q_{st} by p_{st} in S}
14: **end if**
15: **end if**
16: **end if**
17: **end for**
18: Update the pheromone trail using the paths in S and the paths constructed by the ants.
19: **end while**

Algorithm 3 Ant Algorithm - Construction of a path between nodes s and t

Require: Network $\mathcal{N} = (\mathcal{V}, \mathcal{E}, l)$
$s, t \in \mathcal{V}$.
Ensure: P { path between s and t}.
1: $P = \{s\}$ {initialization of the path (LIFO).}
2: $r = s$.
3: **while** $r \neq t$ **do**
4: **if** exists q such that $(r, q) \in \mathcal{E} \wedge q \notin P$ **then**
5: randomly chose $q \in \mathcal{N} \cap \overline{P}$ according to the probability $p_{rq} = \frac{\tau_{rq}^{\alpha} \frac{1}{d_{qt}^{\beta}}}{\sum_{o \notin P \wedge (r,o) \in \mathcal{E}} \tau_{ro}^{\alpha} \frac{1}{d_{ot}^{\beta}}}$
6: insert q in the top of P
7: **else** {impossible to continue from the current node.}
8: removes de top element of the P.
9: set r as the top element of the P.
10: **end if**
11: **end while**

3.2 Test & Results

In this subsection, we present some of the results obtained with an implementation, in *C++*, of the ACO-KSPP[1]. To exemplify the performance we have selected three networks (Figure 1). Each network has 100 nodes and unitary length edges.

For all combinations of the parameters $\rho \in \{0.25, 0.5, 0.75, 1.0\}$ (persistence of the trail) and $\alpha, \beta \in \{1, 5, 10\}$ (relative importance of the trail and heuristic, respectibly) the program was run 10 times, with 100 cycles per run and 100 ants per cycle. We also considered the problem of finding the 10 shortest paths ($k = 10$), starting at the left-downer node and terminating in the right-upper node of the presented networks. Optimum was found for all runs.

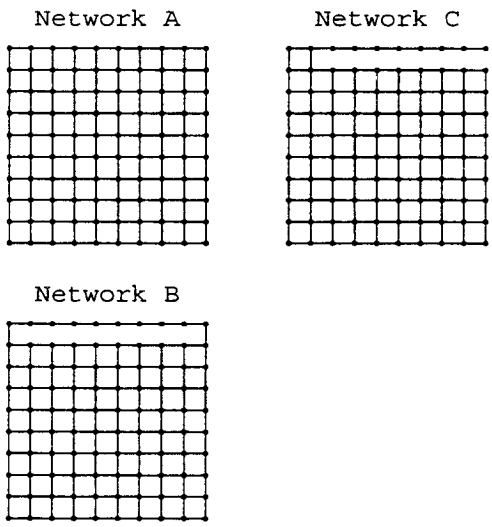

Figure 1: Example Networks

The results obtained for *Networks A*, *B* and *C* are in Tables 1, 2 and 3, respectively. Each table entry has two elements: first element is the medium number of cycles to reach optimum and second element is the medium time took to run the 100 cycles (for the 10 runs of the program).

Figures 2, 3 and 4 illustrate, respectively for *Networks A*, *B* and *C*, the typical progress of the medium length of the constructed paths over the 100 cycles considering $\alpha = 5$, $\beta = 10$ and $\rho \in \{0.25, 0.5, 0.75, 1\}$ (from bottom line to upper line we have $\rho = 1$, $\rho = 0.75$, $\rho = 0.5$ and $\rho = 0.25$ respectively).

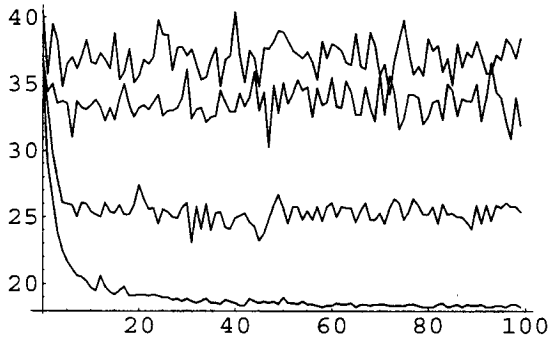

Figure 2: Medium length (ordinates) per cycle (abscissas) typical progress over the 100 cycles for *Network A* with $\alpha = 5$, $\beta = 10$ and $\rho \in \{0.25, 0.5, 0.75, 1\}$ (from bottom line to upper line we have $\rho = 1$, $\rho = 0.75$, $\rho = 0.5$ and $\rho = 0.25$ respectively).

$\rho = 0.25$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(13.8,2.3)	(10.0,2.1)	(9.0,2.1)
$\alpha = 5.00$	(12.4,2.3)	(9.7,2.2)	(8.4,2.1)
$\alpha = 10.00$	(12.4,2.3)	(10.2,2.2)	(9.5,2.2)

$\rho = 0.50$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(7.2,2.2)	(5.6,2.0)	(4.8,2.0)
$\alpha = 5.00$	(7.0,2.2)	(6.2,2.0)	(5.2, 2.0)
$\alpha = 10.00$	(7.8,2.2)	(5.8,2.0)	(5.8, 2.0)

$\rho = 0.75$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(4.1,1.9)	(3.9,1.7)	(3.8,1.7)
$\alpha = 5.00$	(4.4,1.9)	(3.9,1.7)	(3.6,1.7)
$\alpha = 10.00$	(4.3,1.9)	(3.7,1.7)	(3.4,1.7)

$\rho = 1.00$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(3.1,1.6)	(3.0,1.5)	(3.0,1.4)
$\alpha = 5.00$	(3.4,1.6)	(3.2,1.5)	(2.8,1.4)
$\alpha = 10.00$	(3.6,1.6)	(3.2,1.5)	(2.9,1.4)

Table 1: Results obtained for *Network A* - table entries: (medium number of cycles to reach optimum, medium time to run 100 cycles).

For *Networks A* and *B* the best results (i.e., medium number of cycles to reach minimum and best time) were obtained when $\rho = 1$ and $\beta = 10$. By the other side, the number of runs don't let us

[1]This tests were run in a *LINUX-PC* with an *AMD-K6-350Mhz* processor and *64Mb* of *RAM*.

infer the optimum value for α.

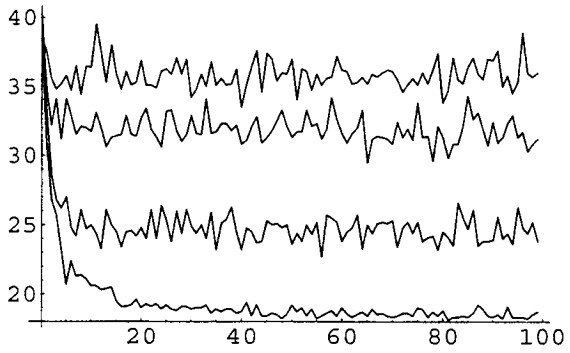

Figure 3: Medium length (ordinates) per cycle (abscissas) typical progress over the 100 cycles for *Network B* with $\alpha = 5$, $\beta = 10$ and $\rho \in \{0.25, 0.5, 0.75, 1\}$ (from bottom line to upper line we have $\rho = 1$, $\rho = 0.75$, $\rho = 0.5$ and $\rho = 0.25$ respectively).

$\rho = 0.25$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(10.1,2.1)	(10.2,2.1)	(9.0,2.0)
$\alpha = 5.00$	(9.7,2.0)	(8.4,2.0)	(9.6,2.0)
$\alpha = 10.00$	(10.7,2.0)	(10.7,2.0)	(8.7,2.0)

$\rho = 0.50$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(6.8,1.9)	(5.4,1.8)	(4.8,1.8)
$\alpha = 5.00$	(6.5,1.9)	(5.9,1.8)	(5.4,1.8)
$\alpha = 10.00$	(5.5,1.9)	(5.2,1.8)	(4.8,1.9)

$\rho = 0.75$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(4.0,1.6)	(4.2,1.6)	(3.9,1.6)
$\alpha = 5.00$	(4.0,1.6)	(3.9,1.6)	(3.4,1.6)
$\alpha = 10.00$	(3.9,1.6)	(3.7,1.6)	(3.7,1.6)

$\rho = 1.00$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(3.1,1.4)	(3.1,1.4)	(3.0,1.4)
$\alpha = 5.00$	(3.1,1.4)	(3.1,1.4)	(3.0,1.4)
$\alpha = 10.00$	(3.1,1.4)	(2.9,1.4)	(3.0,1.4)

Table 2: Results obtained for *Network B* - table entries: (medium number of cycles to reach optimum, medium time to run 100 cycles).

For *Network C* the best results were similar when $\rho = 0.75$ and $\rho = 1$, although for $\rho = 0.75$ better time were obtained. Here, considering $\beta = 10$ can be penalizing since, that would guide the search straight forward to the terminal node.

In conclusion, setting a low evaporation of the pheromone trail (higher values of ρ) seems to lead

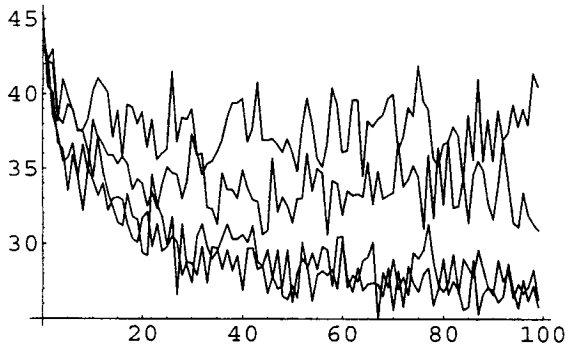

Figure 4: Medium length (ordinates) per cycle (abscissas) typical progress over the 100 cycles for *Network C* with $\alpha = 5$, $\beta = 10$ and $\rho \in \{0.25, 0.5, 0.75, 1\}$ (from bottom line to upper line we have $\rho = 1$, $\rho = 0.75$, $\rho = 0.5$ and $\rho = 0.25$ respectively).

$\rho = 0.25$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(28.4,2.5)	(23.2,2.5)	(28.8,2.5)
$\alpha = 5.00$	(29.4,2.5)	(26.2,2.5)	(18.9,2.5)
$\alpha = 10.00$	(18.4,2.5)	(20.5,2.5)	(18.2,2.5)

$\rho = 0.50$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(8.5,2.3)	(11.4,2.3)	(12.0,2.3)
$\alpha = 5.00$	(8.9,2.3)	(9.3,2.3)	(18.1,2.3)
$\alpha = 10.00$	(18.8,2.3)	(11.1,2.3)	(10.0,2.3)

$\rho = 0.75$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(8.2,2.3)	(9.6,2.3)	(9.3,2.3)
$\alpha = 5.00$	(8.9,2.2)	(8.3,2.3)	(7.4,2.3)
$\alpha = 10.00$	(9.1,2.3)	(9.0,2.3)	(9.7,2.3)

$\rho = 1.00$

	$\beta = 1.00$	$\beta = 5.00$	$\beta = 10.00$
$\alpha = 1.00$	(9.1,2.8)	(9.0,2.8)	(7.9,2.8)
$\alpha = 5.00$	(8.6,2.8)	(9.4,2.8)	(8.3,2.8)
$\alpha = 10.00$	(8.4,2.8)	(8.7,2.8)	(8.8,2.8)

Table 3: Results obtained for *Network C* - table entries: (medium number of cycles to reach optimum, medium time to run 100 cycles).

to good results, although the premature convergence to local optimums, will restrain the exploration of the search space. To avoid this, a dynamic evolution of the parameters, over and in the cycles, seems a good idea. We can also conclude that, for good sets of parameters, a small number of cycles were needed to achieve optimum, with each cycle taking few hundredths of second.

3.3 Conclusions & Further Work

In this paper, we showed another application of the ACO algorithms. The final objective wasn't to *compete* with the known determinist algorithms for the k Shortest Path Problems but, to formulate a support to solve the problem of model/simulate dynamic networks (as, for example, those from traffic simulation problems). Here, the responses are improved as system evolves, due to the establishment of a memory (pheromone trails). That memory resembles the memory of the real individuals that usually travel in a network, which allows them to dynamically make choices as they move from one point to another, accordingly to the network conditions (like the jamming of a given edge or the time of the day).

Taking this in consideration there is a set of factors to introduce or improve, namely:

- introduction of a origin/destiny flow matrix.
- develop and introduce in the algorithm a characterization of the network, that allows a better simulation. Some of this obvious characteristics are:
  - capacity and flows in the edges.
  - capacity of individuals going through a node.
  - time slices.
- make a hybrid system by the introduction of some local search.
- introduce distributed/parallel computation.

References

[1] Emile Aarts and Jan Karel Lenstra. *Local Search in combinatorial Optimization.* Wiley&Sons, 1997.

[2] Mikhail Atallah. *Algorithms and Theory of Computation Handbook.* CRC Press, 1999.

[3] Bernd Bullnheimer, Richard Hartl, and Christine Strauss. An improved ant system algorithm for the vehicle routing problem. Technical Report POM-10/97, Institute of Management Science, University of Vienna, Austria, 1997.

[4] Marco Dorigo and Luca Gambardella. Ant colonies for the traveling salesman problem. Technical Report TR/IRIDIA/1996-3, Université Libre de Bruxelles, Belgium, 1996.

[5] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transaction on Systems, Man and Cybernetics*, 26(1):29–41, 1996.

[6] David Eppstein. Finding the k shortest paths. Technical Report 94-26, Department of Information and Computer Science, University of California, Irvine, USA, 1994.

[7] Luca Gambardella, Éric Taillard, and Giovanni Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. Technical Report IDSIA-06-99, Lugano, Switzerland, 1999.

[8] Ian Parmee. *Evolutionary and Adaptative Computing in Engineering Design.* Springer-Verlag, 2001.

[9] Sadiq Sait and Habib Youseff. *Iterative Computer Algorithms with Applications in Engineering.* IEEE - Computer Society, 1997.

[10] Tetsuo Shibuya, Takahiro Ikeda, Hiroshi Imai, Shigeki Nishimura, Hiroshi Shimoura, and Kenji Tenmoku. Finding a realistic detour by ai search techniques.

[11] Thomas Stutzle and Marco Dorigo. Aco algorithms for the traveling salesman problem.

[12] Michael Vose. *The Simple Genetic Algoritm: Foundations and Theory.* MIT Press, 1999.