Chapter 8
Construction Learning

After having studied the four basic principles—modeling, decomposition, construction, and improvement—this chapter introduces the fifth principle of metaheuristics: learning mechanisms. The algorithms seen in the previous chapter rely solely on chance to try to obtain better solutions than would be provided by greedy constructive methods or local searches. This is probably not very satisfactory from the intellectual point of view. Without solely relying upon chance, this chapter studies how to implement learning techniques to build new solutions. Learning processes require three ingredients:

- Repeating experiences and analysing successes and failures: we only learn by making mistakes!
- Memorizing what has been made.
- Forgetting the details. This gives the ability to generalize when in a similar but different situation.

8.1 Artificial Ants

The artificial ant technique provides simple mechanisms to implement these learning ingredients in the context of constructing new solutions.

The social behavior of some animals has always fascinated, especially when a population comes to realizations completely out of reach of an isolated individual. This is the case with bees, termites, or ants: although each individual follows an extremely simple behavior, a colony is able to build complex nests or efficiently supply its population with food.

É. D. Taillard, *Design of Heuristic Algorithms for Hard Optimization*, Graduate Texts in Operations Research, https://doi.org/10.1007/978-3-031-13714-3_8

8.1.1 Real Ant Behavior

Following the work of Deneubourg et al. [2] who described the almost algorithmic behavior of ants, researchers had the idea of simulating this behavior to solve difficult problems.

The typical behavior of an ant is illustrated in Fig. 8.1 with an experience made with a real colony that has been isolated. The latter can only look for food by going out from a single orifice. The last is connected to a tube separated into two branches joining further. The left branch is shorter than the one on the right. As ants initially have no information on this fact, the ants equally distribute in both branches (Fig. 8.1a).

While exploring, each ant drops a chemical substance that it is apt to detect with its antennas, which will assist it when returning to the anthill. Such a chemical substance carrying information is called pheromones. On the way back, an ant deposits a quantity of pheromones depending on the quality of the food source. Naturally, an ant that has discovered a short path is able to return earlier than that which used the bad branch.

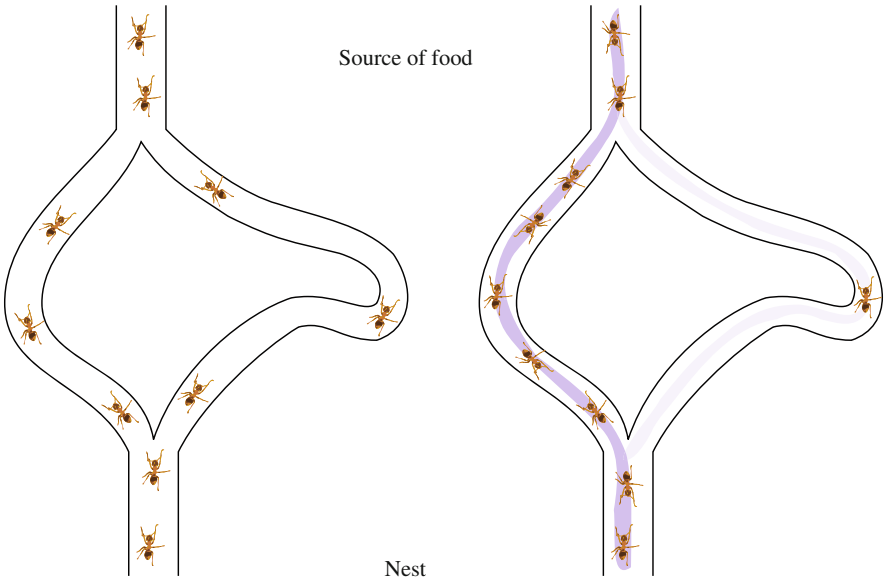

Fig. 8.1 Behavior of an ant colony separated from a food source by a path that is divided. Initially, ants are evenly distributed in both branches (**a**). The ants having selected the shortest path arrive earlier at the food source. Therefore, they faster lay additional pheromones on the way back. The quantity of pheromones deposited on the shortest path grows faster. After a while, virtually all ants will use the shortest branch (**b**)

Therefore, the quantity of pheromones deposited on the shortest path grows faster. Consequently, a new arriving ant has information on the way to take and bias its choice in favour of the shortest branch. After a while, it is observed that virtually all ants use the shortest branch (Fig. 8.1b). Thus, the colony collectively determines an optimal path, while each individual sees no further than the tip of its antennas.

8.1.2 Transcription of Ant Behavior to Optimization

If an ant colony manages to optimize the length of a path, even in a dynamic context, we should be able to transcribe the behavior of each individual in a simple process for optimizing intractable problems. This transcript may be obtained as follows:

- An ant represents a process performing a procedure that constructs a solution with a random component. Many of these processes may run in parallel.
- Pheromone trails are τ_e values associated with each element e constituting a solution.
- Traces play the role of a collective memory. After constructing a solution, the values of the elements constituting the latter will be increased by a quantity depending on the solution quality.
- The oblivion phenomenon is simulated by the evaporation of pheromone trails over time.

Next is to clarify how these components can be put in place. The construction process can use a randomized construction technique, almost similar to the GRASP method. However, the random component must be biased not only by the incremental cost function $c(s, e)$, which represents the *a priori* interest of including element e in the partial solution, but also by the value τ_e which is the *a posteriori* interest of this element. The last is solely known after having constructed a multitude of solutions.

The marriage of these two forms of interest is achieved by selecting the next item e to include in the partial solution s with a probability proportional to $\tau_e^{\alpha} \cdot c(s, e)^{\beta}$, where $\alpha > 0$ and $\beta < 0$ are two parameters balancing the respective importance accorded to memory and incremental cost. The update of artificial pheromones is performed in two steps, each requiring a parameter. First, the evaporation of pheromones is simulated by multiplying all the values by $1 - \rho$, where $0 \leqslant \rho \leqslant 1$ represents the evaporation rate. Then, each element e constituting a newly constructed solution has its τ_e value increased by a quantity $1/f(s)$, where $f(s)$ is the solution cost, which is assumed to be minimized and greater than zero.

8.1.3 MAX-MIN Ant System

The first artificial ant colony applications contained only the components described above. The trail update is a positive feedback process. There is a bifurcation point between a completely random process (learning-free) and an almost deterministic one, repeatedly constructing the same solution (too fast learning). Therefore, it is difficult to tune a progressive learning process with the three parameters α, β and ρ.

To remedy this, Stützle and Hoos [5] suggested limiting the trails between two values τ_{min} and τ_{max}. Hence, selecting an element is bounded between a minimum and a maximum probability. This avoids elements possessing an extremely high trail value, implying that all solutions would contain these elements. This leads to the MAX-MIN ant system, which proved much more effective than many other previously proposed frameworks. It is given in Algorithm 8.1.

Algorithm 8.1: MAX-MIN ant system framework

Input: Set E of elements constituting a solution; incremental cost function $c(s,e) > 0$; fitness function f to minimize, parameters $I_{max}, m, \alpha, \beta, \tau_{min}, \tau_{max}, \rho$ and improvement method $a(\cdot)$
Result: Solution s^*

$f^* \leftarrow \infty$
for $\forall e \in E$ **do**
  $\tau_e \leftarrow \tau_{max}$
for I_{max} *iterations* **do**
  for $k = 1 \dots m$ **do**
    Initialize s as a trivial, partial solution
    $R \leftarrow E$ // Elements that can be added to s
    while $R \neq \varnothing$ **do** Build a new solution
      Randomly choose $e \in R$ with a probability proportional to $\tau_e^\alpha \cdot c(s,e)^\beta$ // Ant colony formula
      $s \leftarrow s \cup e$
      From R, remove the elements that cannot be added any more to s
    $s_k \leftarrow a(s)$ // Find the local optimum s_k associated with s
    if $f^* > f(s_k)$ **then** Update the best solution found
      $f^* \leftarrow f(s_k)$
      $s^* \leftarrow s_k$
  for $\forall e \in E$ **do** Pheromone trail evaporation
    $\tau_e \leftarrow (1-\rho) \cdot \tau_e$
  $s_b \leftarrow$ best solution from $\{s_1, \dots, s_m\}$
  for $\forall e \in s_b$ **do** Update trail, maintaining it between the bounds
    $\tau_e \leftarrow max(\tau_{min}, min(\tau_{max}, \tau_e + 1/f(s_b)))$

This framework comprises an improvement method. Indeed, implementations of "pure" artificial ants colonies, based solely on building solutions, have proven

inefficient and difficult to tune. There may be exceptions, especially for the treatment of highly dynamic problems where an optimal situation at a given time is no longer optimum at another one.

Algorithm 8.1 has a theoretical advantage: it can be proved that if the number of iterations $I_{max} \rightarrow \infty$ and if $\tau_{min} > 0$, then it finds a globally optimal solution with probability tending to one. The demonstration is based on the fact that $\tau_{min} > 0$ implies that the probability of building a globally optimal solution is not zero. In practice, however, this theoretical result is not tremendously useful.

8.1.4 Fast Ant System

One of the disadvantages of numerous frameworks based on artificial ants is their large number of parameters and the difficulty of tuning them. This is the reason why we have not presented *Ant systems* (AS [1]) or *Ant Colony System* (ACO [3]) in detail. In addition, it can be challenging to design an incremental cost function providing pertinent results. An example is the quadratic assignment problem. Since any pair of elements contributes to the fitness function, the ultimate element to include can contribute significantly to the quality of the solution. Conversely, the first item placed does not incur any cost. This is why a simplified framework called *FANT* (for Fast Ant System) has been proposed.

In addition to the number of iterations, I_{max}, the user of this framework must only specify another parameter, τ_b. It corresponds to the reinforcement of the artificial pheromone trails. This reinforcement is systematically applied to the elements of the best solution found so far at each iteration. The reinforcement of the traces associated with the elements of the solution constructed at the current iteration, τ_c, is a self-adaptive parameter. Initially, this parameter is set to 1. When over-learning is detected (the best solution is again generated), τ_c is incremented, and all trails are reset to τ_c. This implements the oblivion process and increases the diversity of the solutions generated.

If the best solution has been improved, then τ_c is reset to 1 to give more weight to the elements constituting this improved solution. Ultimately, FANT incorporates a local search method. As mentioned above, it has indeed been noticed that the sole construction mechanism often produces bad quality solutions. Algorithm 8.2 provides the FANT framework.

Figure 8.2 illustrates the FANT behavior on a TSP instance with 225 cities. In this experiment, the value of τ_b was fixed to 50. This figure provides the number of edges different from the best solution found so far, before and after calling the improvement procedure.

A natural implementation of the trails for the TSP is to use a matrix τ rather than a vector. Indeed, an element e of a solution is an edge $[i, j]$, defined by its two incidents vertices. Therefore the value τ_{ij} is the a posteriori interest to have the edge $[i, j]$ in a solution. The initialization of this trail matrix and its update may therefore be implemented with the procedures described by Code 8.2.

Algorithm 8.2: FANT framework. Most of the lines of code are about automatically adjusting the weight τ_c assigned to the newly built solution against the τ_b weight of the best solution achieved so far. If the latter is improved or if over-learning is detected, the trails are reset

Input: Set E of elements constituting a solution; fitness function f to minimize, parameters I_{max}, τ_b and improvement method $a(\cdot)$
Result: Solution s^*

$f^* \leftarrow$
$\tau_c \leftarrow 1$
for $\forall e \in E$ **do**
    $\tau_e \leftarrow \tau_c$
for I_{max} *iterations* **do**
    Initialize s to a partial, trivial solution
    $R \leftarrow E$ // Elements that can be added to s
    while $R \neq \varnothing$ **do**
        Randomly choose $e \in R$ with a probability proportionnal to τ_e
        $s \leftarrow s \cup e$
        From R, remove the elements that cannot be added any more to s
    $s' \leftarrow a(s)$ // Find the local optimum s' associated with s
    if $s' = s^*$ **then** manage over-learning
        $\tau_c \leftarrow \tau_c + 1$ // More weight to the newly constructed solutions
        for $\forall e \in E$ **do** Erase all trails
            $\tau_e \leftarrow \tau_c$
    if $f^* > f(s_k)$ **then** manage best solution improvement
        $f^* \leftarrow f(s_k)$
        $s^* \leftarrow s_k$ // Update best solution
        $\tau_c \leftarrow 1$ // Give minimum weight to the newly constructed solutions
        for $\forall e \in E$ **do** Erase all trails
            $\tau_e \leftarrow \tau_c$
    for $\forall e \in s'$ **do** reinforce the trails associated with the current solution
        $\tau_e \leftarrow \tau_e + \tau_c$
    for $\forall e \in s^*$ **do** reinforce the trails associated with the best solution
        $\tau_e \leftarrow \tau_e + \tau_b$

The core of an ant heuristic is the construction of a new solution exploiting artificial pheromones. Code 8.1 provides a procedure not exploiting the a priori interest (an incremental cost function) of the elements constituting a solution. In this implementation, the departure city is the first of a random permutation p. At iteration i, the i first cities are definitively chosen. At that time, the next city is selected with a probability proportional to the trail values of the remaining elements.

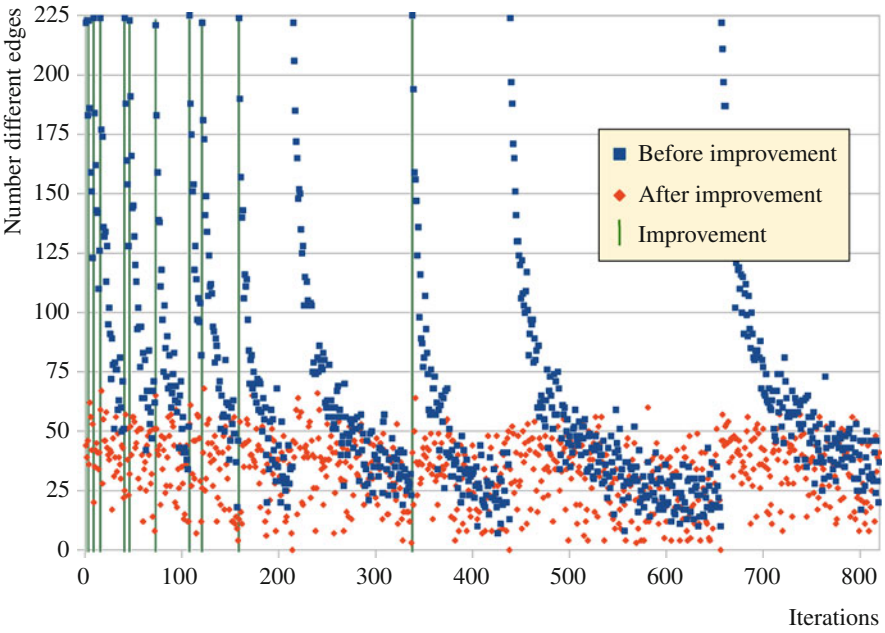

Fig. 8.2 FANT behaviour on a TSP instance with 225 cities. For each iteration, the diagram provides the number of edges different from the best solution found by the algorithm, before and after calling the ejection chain local search. Vertical lines indicate improvements in the best solution found. In this experiment, the last of these improvements corresponds to the optimal solution

Code 8.1 generate_solution_trail.py Implantation of the generation of a permutation only exploiting the information contained in the pheromone trails

```
from random_generators import unif                                    # Listing 12.1
from tsp_utilities import tsp_length                                  # Listing 12.2

########## Building a solution using artificial pheromone trails
def generate_solution_trail(d,                                        # Distance matrix
                            tour,                                     # Tour produced by the ant
                            trail):                                   # Pheromone trails
    n = len(tour)
    for i in range(1, n - 1):
        total = 0
        for j in range(i + 1, n):
            total += trail[tour[i - 1]][tour[j]]
        target = unif(0, total - 1)
        j = i
        total = trail[tour[i - 1]][tour[j]]
        while total < target:
            total += trail[tour[i - 1]][tour[j + 1]]
            j += 1
        tour[j], tour[i] = tour[i], tour[j]
    return tour, tsp_length(d, tour)
```

Once the three procedures given by the Codes 8.1 and 8.2 as well as an improvement procedure are available, the implementation of FANT is very simple. Such an implantation, using an ejection chain local search, is given by Code 8.3

Code 8.2 init_update_trail.py Implementation of the trail matrix initialization and update for the FANT method applied to a permutation problem. If the solution just generated is the best previously found, trails are reset. Otherwise, the trails are reinforced both with the current solution and the best one

```
######## (Re-)initialize all trails
def init_trail(initial_value,                    # Initial value for all trails
               trail):                                    # Pheromone trails

    n = len(trail[0])
    for i in range(n):
        for j in range(n):
            trail[i][j] = initial_value
    for i in range(n):
        trail[i][i] = 0
    return trail

######### Updating trail values
def update_trail(tour,                        # Last solution generated by an ant
                 global_best,                          # Global best solution
                 exploration,                 # Reinforcement of last solution
                 exploitation,          # Reinforcement of global best solution
                 trail):                                    # Pheromone trails

    if tour == global_best:
        exploration += 1                      # Give more weight to exploration
        trail = init_trail(exploration, trail)
    else:
        for i in tour:
            n = len(trail[0])
            trail[tour[i]][tour[(i + 1) % n]] += exploration
            trail[global_best[i]][global_best[(i + 1) % n]] += exploitation
    return trail, exploration
```

Code 8.3 tsp_FANT.py FANT for the TSP. The improvement procedure is given by Code 12.3

```
from random_generators import rand_permutation                    # Listing 12.1
from generate_solution_trail import *                             # Listing  8.1
from init_update_trail import *                                   # Listing  8.2
from tsp_LK import tsp_LK                                         # Listing 12.3

######### Fast Ant System for the TSP
def tsp_FANT(d,                                                   # Distance matrix
             exploitation,              # FANT Parameters: global reinforcement
             iterations):                      # number of solution to generate

    n = len(d[0])
    best_cost = float('inf')
    exploration = 1
    trail = [[-1] * n for _ in range(n)]
    trail = init_trail(exploration, trail)
    tour = rand_permutation(n)
    for i in range(iterations):
        # build solution
        tour, cost = generate_solution_trail(d, tour, trail)
        # improve built solution witho a local search
        tour, cost = tsp_LK(d, tour, cost)
        if cost < best_cost:
            best_cost = cost
            print('FANT {:d} {:d}'.format(i+1, cost))
            best_sol = list(tour)
            exploration = 1                 # Reset exploration to lowest value
            trail = init_trail(exploration, trail)
        else:
            # pheromone trace reinforcement - increase memory
            trail, exploration = update_trail(tour, best_sol,
                                              exploration, exploitation, trail)
    return best_sol, best_cost
```

8.2 Vocabulary Building

Vocabulary building is a more global learning method than artificial ant colonies. The idea is to memorize fragments of solutions, which are called words, and to construct new solutions from these fragments. Put differently, one has a dictionary used to build a sentence attempt in a randomized way. A repair/improvement procedure makes this solution attempt feasible and increases its quality. Finally, this new solution sentence is fragmented into new words that enrich the dictionary.

This method has been proposed in [4] and is not yet widely used in practice, although it has proved efficient for a number of problems. For instance, the method can be naturally adapted to the vehicle routing problem. Indeed, it is relatively easy to construct solutions with tours similar to those of the most efficient solution known. This is illustrated in Fig. 8.3.

By building numerous solutions using randomized methods, the first dictionary of solution fragments can be acquired. This is illustrated in Fig. 8.4.

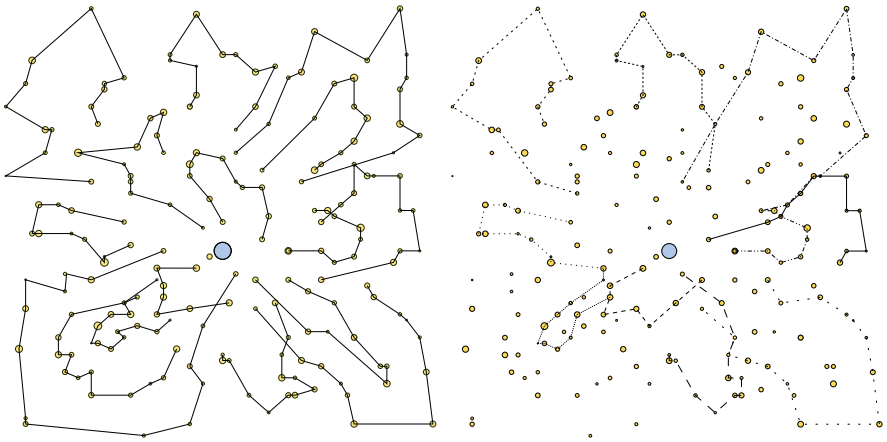

Fig. 8.3 (**a**) The optimal solution to a VRP instance. (**b**) A few tours quickly obtained with a taboo search. We notice great similarities between the latter and those of the optimal solution

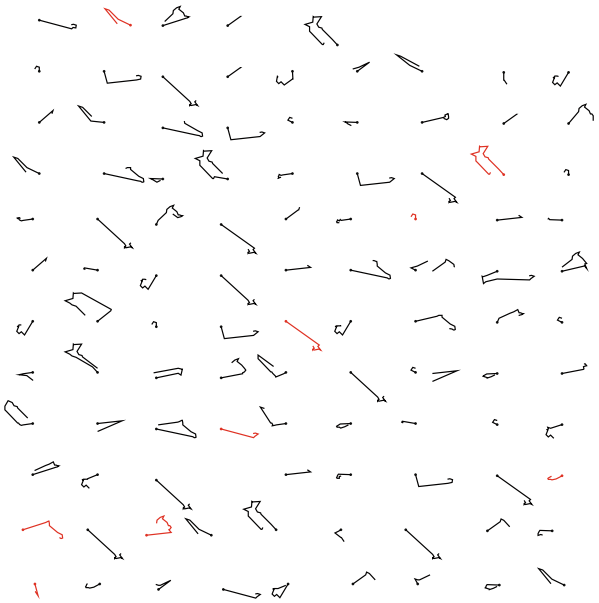

Fig. 8.4 Fragments of solutions (vehicle routing tours) constituting the dictionary. A partial solution is built by randomly selecting a few of these fragments (indicated in color)

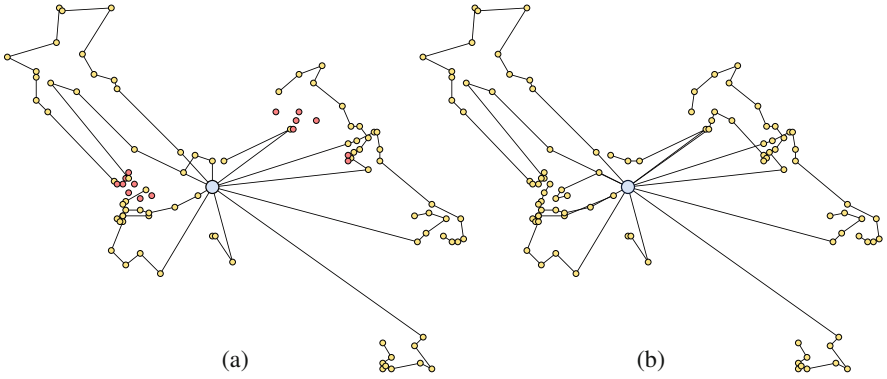

Fig. 8.5 (**a**) A sentence attempt is constructed by randomly selecting a few words from dictionary (**b**). This attempt is completed and improved

Once an initial dictionary has been constructed, solution attempts are built, for instance, by selecting a subset of tours that do not contain common customers. This solution is not necessarily feasible. Indeed, during the construction process, the dictionary might not include tours only containing customers not yet covered. Therefore, it is necessary to repair this solution attempt, for instance, by means of a method similar to that used to produce the first dictionary but starting with the solution attempt. This phase of the method is illustrated in Fig. 8.5. The improved solution is likely to contain tours that are not yet in the dictionary. These are included to enrich it for subsequent iterations.

The technique can be adapted to other problems, like the TSP. In this case, the dictionary words can be edges appearing in a tour. Figure 8.6 shows all the edges present in more than two-thirds of 100 tours obtained by applying a local search starting with a random solution. The optimal solution to this problem is known. Hence, it is possible to highlight the few edges frequently obtained that are not part of the optimal solution. Interestingly, nearly 80% of the edges of the optimal solution have been identified by initializing the dictionary with a basic improvement method.

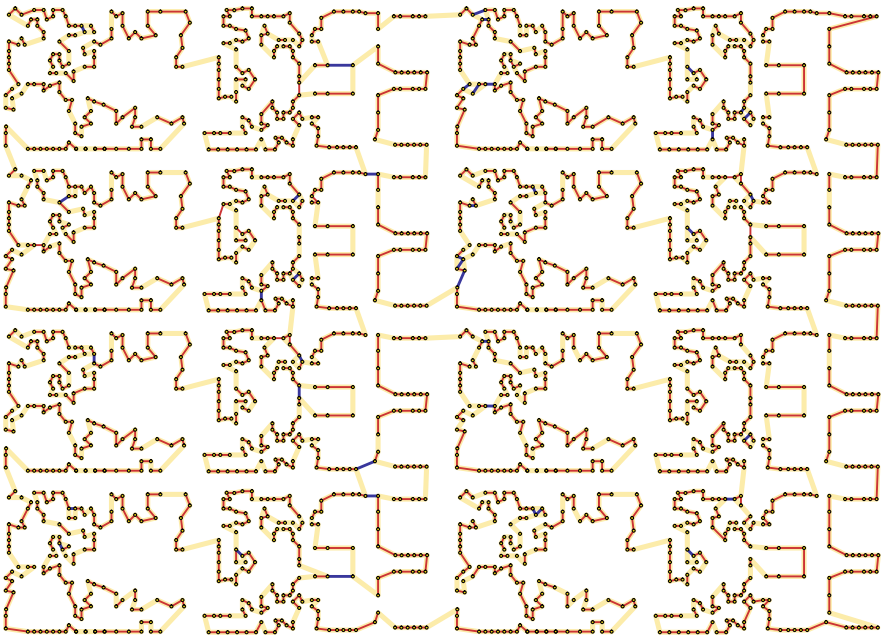

Fig. 8.6 An optimal solution (light color) and fragments of tours constituting an initial dictionary for the TSP instance pr2392. The fragments are obtained by repeating 100 local searches starting with random solutions and only retaining the edges appearing in more than 2/3 of the local optima. Interestingly, almost all these edges belong to an optimal solution. The few edges that are not part of it are highlighted (darkest color)

Problems

8.1 Artificial Ants for Steiner Tree

For the Steiner tree problem, how to define the trails of an artificial ant colony? Describe how these trails are exploited.

8.2 Tuning the FANT Parameter

Determine good values for the parameter τ_b of the `tsp_FANT` method provided by Code 8.3 when the latter performs 300 iterations. Consider the *TSPLIB* instance tsp225.

8.3 Vocabulary Building for Graph Coloring

Describe how vocabulary construction can be adapted to the problem of coloring the vertices of a graph.

References

1. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: Actes de la première conférence européenne sur la vie artificielle, pp. 134–142. Elsevier, Paris (1991)
2. Deneubourg, J., Goss, S., Pasteels, J., Fresneau, D., Lachaud, J.: Self-organization mechanisms in ant societies (II): Learning in foraging and division of labor. In: Pasteels, J., et al. (eds.) From Individual to Collective Behavior in Social Insects, Experientia supplementum, vol. 54, pp. 177–196. Birkhäuser, Basel (1987)
3. Dorigo M., Gambardella L.M.: Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997). https://doi.org/10.1109/4235.585892
4. Glover, F.: Tabu search and adaptive memory programming — advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kennington, J.L. (eds.) Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies, pp. 1–75. Springer, Boston (1997). https://doi.org/10.1007/978-1-4615-4102-8_1
5. Stützle, T., Hoos, H.H.: MAX MIN Ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000). https://doi.org/10.1016/S0167-739X(00)00043-1