Map Uncertainty Reduction for a Team of Autonomous Drones Using Simulated Annealing and Bayesian Optimization

Jordan Henrio(✉) and Tomoharu Nakashima

Osaka Prefecture University, Sakai, Japan
jordan.henrio@cs.osakafu-u.ac.jp, tomoharu.nakashima@kis.osakafu-u.ac.jp

Abstract. This research focuses on the problem of reducing the uncertainty rate of an environment in the context of surveillance. A human operator designates a set of locations to be checked by a team of autonomous quadcopters. The goal of this work is to minimize the uncertainty rate of the environment while penalizing solutions whose the total travelled distance is large. To cope with this issue, the A* algorithm is employed to plan the shortest path between each pair of points. Then, a simulated annealing algorithm is used to allocate tasks among the team of drones. This paper discusses three different objective functions to solve the problem whose cost-efficient and feasible solutions can be obtained after a few minutes. The presented work also deals with optimization of the simulated algorithms parameters by using Bayesian optimization. It is currently the state-of-the-art approach for the problem of hyper-parameters search, for expensive to evaluate functions, since it allows to save computation time by modeling the cost function. The Bayesian optimizer returns the best parameters within one day, while the use of grid search methods required weeks of computations.

Keywords: Surveillance · Path planning · Task allocation · A* algorithm · Simulated annealing algorithm · Bayesian optimization

1 Introduction

Conventional surveillance systems use fixed-position cameras. Although they have a pan-and-tilt feature in order to record large-plan videos or even 360-degree-view videos, they still suffer from their "fixed geographical position" characteristic. Thus, it is difficult to track a particular object or to record information about blind spots.

Recently, drones have become more and more popular and this trend is still continuing. It exists a large range of different drones, including ones with embedded tools for video recording. Such a system could be used as a moving surveillance camera. In this context, using a drone instead of, or coupled with, usual systems could improve the security level. Drones can perform better object tracking since they can follow the object of interest and they can also fly around this

S. Yamamoto (Ed.): HIMI 2017, Part II, LNCS 10274, pp. 351–370, 2017.
DOI: 10.1007/978-3-319-58524-6_29

object in order to record various views of it. In addition, because of their flying capacity, they can record information on the ground as well as at heights and as they are small they can also record information in narrow locations as well as indoor areas.

Most of the drones have a feature allowing the user to design a flight path. In the case of this research, a human supervisor selects a set of locations that should be visually checked by a team of autonomous drones. These points are called check points. Each of them is associated with a real value that increases as the time elapses after the last visit increases. This value is used as the uncertainty rate of the corresponding point which represents how confident we are that the information about the point is up to date.

The problem of finding a permutation that minimizes the energy consumption of drones as well as the travelled distance have been largely tackled in the literature, often by suggesting to use complex solvers like memetic algorithms [1–3]. This work proposes a path planning method using a simple simulated annealing algorithm to minimize the average of the check points.

A large grid, presented in Sect. 3, is used as a discrete representation of the environment and the path between every possible pair of points is computed by the application of the A* algorithm. A centralized task allocation system using a simulated annealing algorithm searches for the optimal permutation of the check points according to their uncertainty rate. Section 4 discusses three solvers with different objective functions. One is for the case of minimization of the travelled distance, another for the case of minimization of the uncertainty rate and the last solver mixes the two functions. In Sect. 5, we describe the solver's hyper-parameter optimization process by using Bayesian optimization. Such a method is becoming more and more popular to deal with hyper-parameter optimization issues and saved us a large amount of time compared to the use of a grid search method. Section 6 presents the different experiments conducted during this research.

2 Related Work

The problem of optimal path planning for a team of autonomous drones has been largely tackled in the literature by various approaches. A common approach is employing generally mixed linear integer programming as in Richards et al. [4,5]. While this kind of methods provides an optimal solution, the computation time is often unfeasibly long. Other work have proposed to solve the problem of path planning with a reasonable computation time by computing an approximation of the optimal solution as in [6].

Recent works have suggested to focus on the allocation of points to visit, considering the path to join each point is already known. Then, the problem of interest is similar to the well-known Travelling Salesman Problem (TSP). Since this class of problem is NP-complete, there is no known method that can provide the optimal solution in a polynomial time. Therefore, most of the suggested works solve the problem by using meta-heuristic algorithms as the ant colony algorithm

in Jevtic et al. [7], and the more recent firefly algorithm as in Osaba et al. [8,9]. A large number of works rather propose to use memetic algorithms as in [1–3].

The method proposed in this paper is similar to the work of Liu et al. [2] and Jose et al. [3] except that the desired solution is not the one that minimizes the traveled distance by the teams of robots, but minimizes the uncertainty rate of the check points. However, the method also focuses on the reduction of the travelled distance. Furthermore, memetic algorithms are relatively complex because they require to develop several methods such as solutions' reproduction, mutation, crossover and local optimization. This work rather suggests to use a modest simulated annealing algorithm which allows to find nearly optimal solutions.

3 Representation of the Environment

This research aims to plan paths for a team of autonomous drones that monitor the Osaka Prefecture University campus for surveillance. The campus location can be found at (34.545412, 135.506348) in the latitude/longitude system. It is about a 858×624 m^2 area and contains different buildings of different sizes and heights, trees, a pond, a farming area, stables, a large sports ground, etc. Since some of these elements, as buildings and trees, could represent a danger for the drones, they are considered as *non-admissible areas*, in contrast with clear zones which are considered as *admissible areas*.

The environment of interest is shown in Fig. 1. In this figure, the environment is delimited by a large rectangle containing smaller rectangles and triangles representing the non-admissible areas. The black circle represents the starting point of the drones and the grey circles are the check points. It is important to

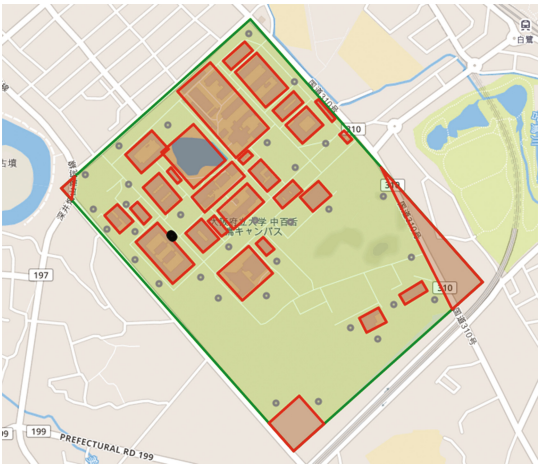

Fig. 1. Map of the environment of interest

note that the map used in this research is not up-to-date and some buildings are missing. We are considering to use a more recent map for future works.

The environment is represented by using a grid of 1716×1248 cells. Thus, one cell represents a 50×50 cm^2 area of the environment. The admissible and non-admissible areas are shown in Fig. 2. The black cells represent non-admissible areas and the white areas admissible ones. For simplification only the buildings and the pond are considered as non-admissible areas. Also the black triangles in the top-right and bottom-left corners as well as the rectangle in the bottom-right corner are not buildings, but only delimitation of the map which considers the boundary of the campus.

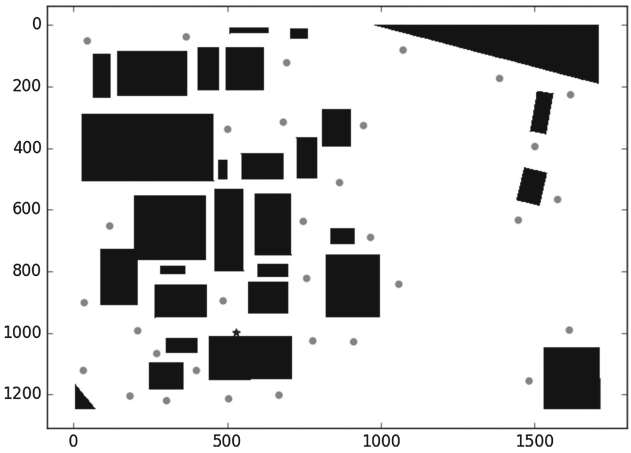

Fig. 2. Discretized representation of the environment with the 32 check points

Once the map has been represented in the above way the path planning module is used to find the shortest path, or at least a short path, between all possible pairs of the check points by applying the A* algorithm. In addition, the path between each check point and the starting point of the drones is also computed. For the implementation of the A* algorithm, the Python library developed by Careaga [10] was used in the computational experiments of this research.

4 Solvers

Once all the paths linking the points designated by the supervisor, as well as the start point, are determined as described in Sect. 3, the task allocation module is called to compute the order in which the points will be visited. The search space can be seen as a graph where the vertices are the locations to visit and the edges are the paths computed by the A* algorithm. Then, the goal is to find the path passing through every vertex, which minimizes the average uncertainty rate as well as travelled distance.

Each check point is associated to a real number u. This value is used to represent the uncertainty about the situation at this particular location in the environment. This uncertainty rate increases according to the elapsed time after the last visit by any drone to this particular point as defined in Acevedo et al. [11]. At the very first time, the uncertainty rate of every check point is initialized to 1, which is equivalent to consider that the situation at these points is unknown. The value of a point i falls directly to 0 when a drone arrives at the point's location and checks the situation. Then, the uncertainty rate will gradually grow up to 1 as the elapsed time increases. This process is mathematically expressed as in (1):

$$u_i(t) = 1 - e^{-\eta t}, \tag{1}$$

where t is the elapsed time after the last visit and η is a decay constant. As depicted in Fig. 3, this function defines an inverted exponential growth of the uncertainty rate. In other words, an event is unlikely to occur right after the situation has been checked, but the environment state can rapidly change after a few minutes. In this work, the uncertainty rate is considered to return to its original state (i.e., 1.0) after one hour since the last visit. In this case, $\eta = 0.001279214$.

The task allocation module's role is to find the permutation which minimizes the estimated average uncertainty rate of the check points at the end of the mission. To do so, we describe three objective functions in the following subsections. The optimization problem with the three objective functions is solved by using a simulated annealing algorithm where the neighbor solutions are obtained by swapping two check points in the permutation. For the implementation of the algorithm, the Python Simanneal library [12] was used.

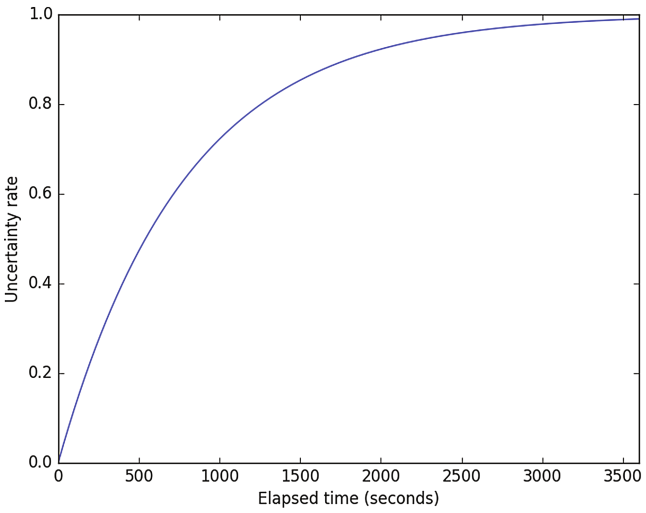

Fig. 3. Uncertainty rate's evolution over one hour

In the following, we search for an approximation of the optimal solution using the check points located as depicted in Fig. 2. In this paper, two drones are involved that monitor the situation of the environment at the 32 check points, starting from the star symbol located in (528,999).

In this paper, the drones are assumed to fly at a constant speed of 1 m/s and their battery is assumed to have 25 min of autonomy. Since the size of one cell in the map is a 50×50 cm^2 area, the drones can travel 3000 cells with one single battery load. In addition, they are assumed to fly at different altitudes to avoid collision issues.

4.1 Tasks Allocation

The problem treated in this section is similar to a TSP. However, in our case the traveling salesman has a limited budget (the drone's energy) and need to return to its start point before running out of money. After he has withdrawn some money (battery replaced), he can resume his travel. Furthermore, in our case, the cities to visit are shared by two salesmen.

The task allocation is done as presented in Algorithm 1. This algorithm cuts the permutation (a single list of several points) in a set of routes (several lists of several points). Here, *permutation* is considered as a queue. The algorithm dequeue the elements while the battery has enough energy to go to the permutation's next point and return to the start point from there. By doing so, every solution returned by this algorithm is feasible. In order to compute the distance between two given points the algorithm takes as input, *distance*, the matrix distance obtained after application of the A* algorithm as explained in Sect. 3. *start* is the start point, *pos* the current position, *battery* the current energy consumption and *limit* is a constant representing the drone autonomy. $list()$, $push(a)$ are functions that, respectively, create an empty list and insert an element a at the tail of the list. $pop()$ is a function that removes the head element of a queue.

It is important to note that for a fixed drone's speed it is possible to estimate the time at which each point is visited since the distance is known. Thus, Algorithm 1 can be easily extended to estimate the points of interest's average uncertainty rate. In addition, the total travelled distance with the solution returned by the algorithm can be easily computed too. As a result, this algorithm can be used as a base for the evaluation function of the solvers described below.

4.2 Minimization of the Travelled Distance

In order to minimize the environment's uncertainty rate, one could intuitively think to search the permutation which minimizes the total travelled distance and thus the duration of the mission. The problem of distance minimization can be defined as in (2):

$$p^* = \arg\min_p distance(p), \tag{2}$$

Algorithm 1. Task Allocation

```
Input: permutation, distance, start, limit
solution, route ← list()
pos ← start
route.push(pos)
battery ← 0
while permutation.length > 0 do
    target ← permutation.head
    if battery + distance_{pos,target} + distance_{target,start} < limit then
        battery ← battery + distance_{pos,target}
        pos ← target
        route.push(pos)
        permutation.pop()
    else
        pos ← start
        route.push(pos)
        solution.push(route)
        route ← list()
        route.push(pos)
        battery ← 0
    end if
end while
return solution
```

$$distance(p) = \sum_{i=2}^{n} d(p_{i-1}, p_i), \tag{3}$$

where p^* is the optimal permutation, n the number of points in the permutation and $d(p_{i-1}, p_i)$ a function returning the number of cells in the path linking points p_{i-1} and p_i, computed by the A* algorithm in Sect. 3.

The simulated annealing algorithm is a meta-heuristic method. Thus, the returned solution is not guaranteed to be optimal. In addition, since the process employs probabilities, solutions returned from one application of the algorithm to another can vary. A typical solution obtained is represented in Fig. 4. The solution is formed by two routes for each drone. Drone 1's routes are represented by a succession of triangles and those of Drone 2 by a succession of crosses. The differences in colors indicate the different routes of each drone. Then, the Drone 1 and Drone 2's black routes are executed in a first time and grey routes are executed after. This solution tries to visit points located on the extremities of the map while visiting points which are on the way.

4.3 Direct Minimization of the Average Uncertainty Rate

The estimated average uncertainty rate at the end of the mission for the solution obtained by the previous solver is about 89.56%. Since this result is relatively high, one can legitimately consider the use of another objective function as described in (4):

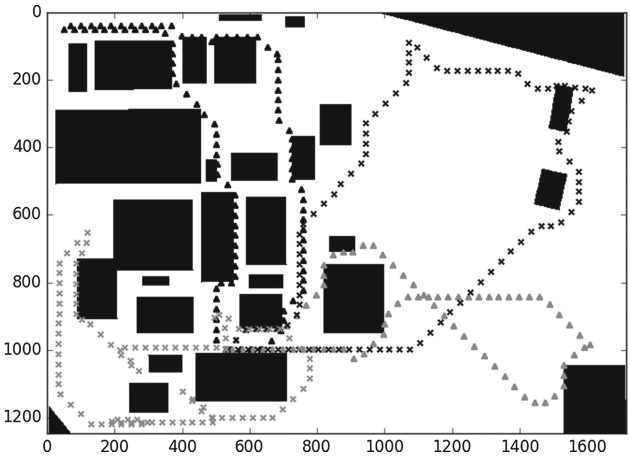

Fig. 4. Solution of the solver minimizing the travelled distance (i.e., (2))

$$p^* = \arg\min_p uncertainty(p), \tag{4}$$

$$uncertainty(p) = \frac{1}{n}\sum_{i=1}^{n} 1 - e^{-\eta(T - t_{p_i})}, \tag{5}$$

where T is the estimated number of elapsed seconds between the beginning and the end of the mission, t_{p_i} is the estimated number of elapsed seconds between the beginning of the mission and the moment when point i will be visited. Thus, this new objective function tries to directly minimize the estimated average uncertainty rate. A typical solution is represented in Fig. 5. The solution tries at first to visit points which are at the extremities of the map without visiting points which are on the way. Since the number of points far away from the start point is lower than the number of points near the start point, this solver "sacrifices" their uncertainty rate in order to maximize the number of points which can be visited quickly just before the end of the mission. This solution has an average uncertainty rate of about 61.61% for a traveled distance of more than 17,000 cells while the first solver finds a solution about two times lower (8392 cells).

4.4 Mixing the Two Objective Functions

By sacrificing the uncertainty rate of points far away from the start point, the previous solver returns solutions which have routes visiting only a single point before returning to the start point like the Drone 1's first route (black triangles path in Fig. 5). This implies that the battery of the drones should be frequently replaced or charged. To cope with this issue, one possible idea is to use an objective function which minimizes the average uncertainty rate while penalizing solutions whose travelled distance is large, by using a positive constant λ

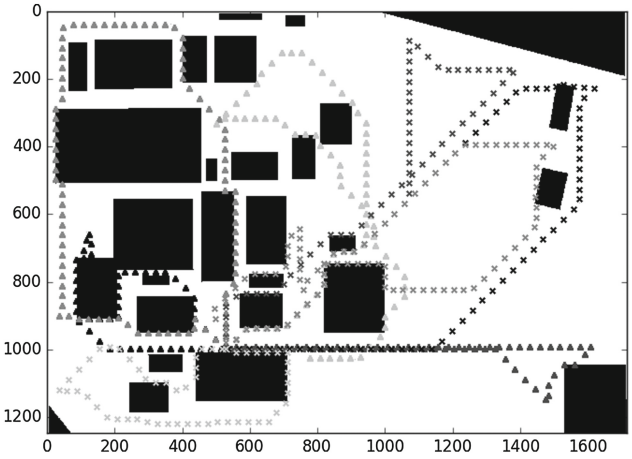

Fig. 5. Solution of the solver minimizing the average uncertainty rate (i.e., (4))

controlling the penalization strength as described in (6):

$$p^* = \arg\min_p \Big\{ uncertainty(p) + \lambda distance(p) \Big\} \tag{6}$$

In our implementation, the average uncertainty rate in (4) and (6) is multiplied by 10000, for two reasons. Firstly, the employed library for the simulated annealing algorithm seemed to have difficulty in optimizing a cost function whose range is between 0 and 1. Secondly, it allows the uncertainty rate to play a more important role in the cost function than the distance. However, this issue could be treated by using a smaller penalization coefficient λ.

A typical solution by using $\lambda = 0.1$ is depicted in Fig. 6. As it can be observed, it looks like the solution found by the first solver (i.e., Fig. 4), but it requires two additional routes (then two additional battery changes). The average uncertainty rate is about 61.76% and the total traveled distance is 13426 cells. Thus, this solution is better than the second solver's solution in terms of travelled distance, but slightly lower in terms of uncertainty rate. Table 1 summarizes the results of the optimization with the objective functions that were presented in this section.

Table 1. Summary of typical solutions with by the different objective functions

Objective function	Distance (#cells)	Uncertainty rate (%)	#routes
Distance	8392	89.56	4
Uncertainty rate	17565	61.61	8
Mix	13426	61.76	6

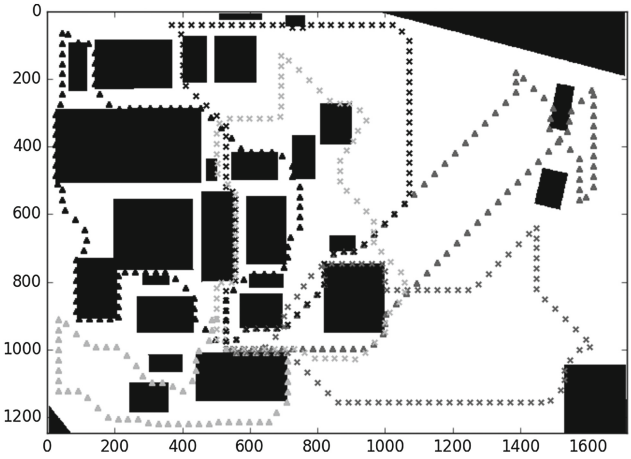

Fig. 6. Solution of the two mixed objective functions (i.e., (6))

5 Hyper-parameter Optimization

The method proposed in Sect. 4 requires the definition of four hyper-parameters. The first one is the penalization coefficient used in the third objective function defined in (6). The three other hyper-parameters come from the simulated annealing algorithm. One is the number of iterations spent by the algorithm to find the best solution. In our implementation, during one iteration, the algorithm swaps two points in the permutation and evaluates the cost of the resulting solution from the new permutation. The simulated annealing as originally defined in Kirkpatrick, Gelatt, Vecchi et al. [13] accepts solutions whose cost is higher than the best solution found with a probability in order to avoid local optima. A new solution so-far with the higher cost is accepted with a probability depending on the current system's temperature, as defined in (7):

$$P(e_{\text{best}}, e_{\text{current}}, T) = \exp\left(-\frac{(e_{\text{current}} - e_{\text{best}})}{T}\right), \tag{7}$$

where e_{best} is the best encountered state's cost, e_{current} is the current state's cost and T is the current system's temperature. This temperature decreases linearly (exponentially) along the iterations, leading to lower and lower exploration. Then, the two other hyper-parameters are the start temperature ($T_{\max}$) and the final temperature ($T_{\min}$) of the system.

In this section, the choice of the penalization coefficient value is presented. Then, we discuss the influence of the number of iterations. Finally, Bayesian optimization is used to optimize the temperature interval.

5.1 Trade-Off Between Uncertainty Rate and Distance

In order to investigate the effect of the penalization coefficient on the cost function, the simulated annealing algorithm was applied several times to the problem defined in Fig. 2 with different values of λ. As explained above, the simulated annealing algorithm induces probabilities in the process and thus solutions resulting from multiple applications of the algorithm are generally slightly different. Therefore, for each tested λ, the algorithm is applied ten times on random starting solutions in order to calculate an average cost.

As depicted in Fig. 7, the traveled distance decreases as the penalization coefficient strength increases. On the other hand, the average uncertainty rate increases. Therefore, there is a trade-off between average uncertainty rate and the traveled distance of solutions.

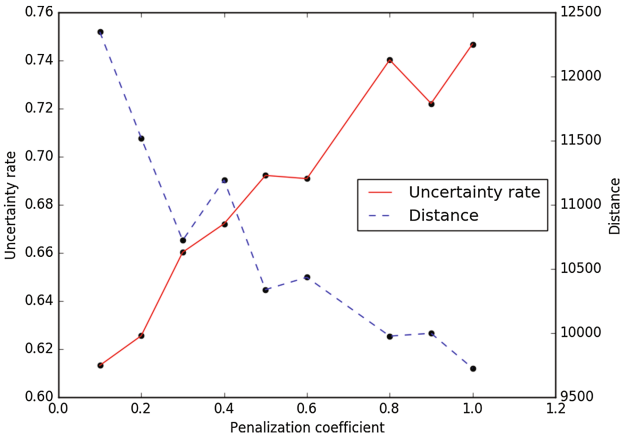

Fig. 7. Influence of the penalization coefficient on the average uncertainty rate and the total traveled distance

Our main goal is to minimize the average uncertainty rate while the distance minimization is a second goal for convenient reasons. Furthermore, the average solution for $\lambda = 0.1$ has a total traveled distance of 12347.5 cells while without penalization the traveled distance was above 17500. Thus, with the drones' characteristics established in Sect. 4, even the lower coefficient in the interval [0.1, 1] saves two routes (i.e. two battery loads) in the plan. Actually, the choice of this hyper-parameter depends on how much the uncertainty rate can be sacrificed in favor of shorter traveled distances, and this choice is arbitrary.

5.2 Effect of Iterations on the Search Performance

The next step consists in analyzing the gain in cost by increasing the number of iterations. While the simulated annealing algorithm could find the optimal

solution for an infinite computation time and a temperature range large enough such that the acceptance rate is not null, we can accept an approximation of the best solution to reduce computation time. Thus, one can legitimately wonder how many iterations it is necessary until the satisfactory solutions are obtained. To do so, the simulated annealing algorithm is applied to the problem defined in Fig. 2 by testing different numbers of iterations in the interval [100000, 20000000]. For each test the algorithm is applied ten times in order to calculate an average solution.

The results of this experiment are depicted in Fig. 8. This figure represents the evolution of the cost function according to the number of iterations. In addition, the gain over the additional computation time implied by the increasing number of iterations is also depicted. The gain changes quickly during the stage of the search process, but tends towards 0 after around 10M iterations (represented by the horizontal dotted line). In other words, the improvement in the cost is smaller than the additional computation time for large numbers of iterations. Also, some peaks can be observed at 8M, 12M and 16M iterations. The lowest cost is found at 16M iterations for an average cost of 7359.24. The two other peaks at 8M and 12M iterations have a cost of 7408.33 and 7395.97 respectively. These abrupt changes come from the variance induced by the simulated annealing which is applied with non optimized temperatures ($T_{\max} = 100$, $T_{\min} = 1$).

To simply rely on the number of iterations is equivalent to perform a brute-force search. In order to refine the algorithm solution cost and reach an approximation of the optimal solution without simply increasing the number of iterations, the temperature range of the system should be setup appropriately according to the problem of interest. The next section focuses on the simulated annealing algorithm's temperature range optimization for a number of iterations fixed to 3M.

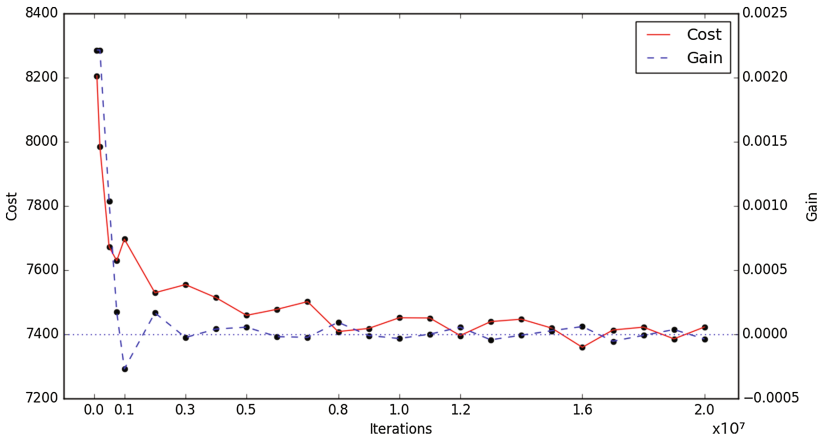

Fig. 8. Cost evolution according to the number of iterations

5.3 Temperature Optimization by Using Bayesian Optimization

As explained above, the system's temperature range is an important setup since it controls the algorithm exploration behavior. If the temperature is too high, most of the new states are accepted and the simulated annealing algorithm acts in a similar way to random search method. On the other hand, if the temperature is too low the algorithm behavior becomes similar to a greedy search method.

Our first attempts for the system's temperatures optimization were done by using a grid search technique. However, since the algorithm takes about 4 min to perform 3M iterations on a core-i7 computer, it is relatively time expensive to test it several times with different hyper-parameters. Thus, the grid search application took several weeks of computations. In addition, the grid search algorithm should be avoided since the optimal values of the hyper-parameters could be outside the grid definition.

Recent researches, especially in the field of machine learning, suggested the use of Bayesian optimization to cope with the issue of function optimization in an intelligent way. This method is efficient for time-consuming task as the simulated annealing algorithm since it takes benefit from inferential probabilities to reduce the number of evaluations at the cost of a relatively expensive sample preparation. The core idea of Bayesian optimization is to first model the cost function, generally with a Gaussian process as defined in (8):

$$f(x) \sim GP(\mu(x), k(x, x')), \quad \forall x, x' \in X^2, \tag{8}$$

given some prior sample data $x \in X$ where X is the search space, $\mu(x)$ the mean function and $k(x, x')$ a covariance function. Then, the optimization process involves sampling data as defined in (9):

$$x_{\text{next}} = \arg\max_{x \in X} a(x) \tag{9}$$

where x_{next} is the next iteration to try and $a(x)$ is an acquisition function whose role is to evaluate the search space according to some criteria given the model of $f(x)$. After sampling data where the acquisition function is maximized, the model defined in (8) is updated with the observed results and the search space is reevaluated by the acquisition function. This process is repeated until convergence or any other termination criteria.

For our implementation we used the Python library GpyOpt [14]. In our case $f(x)$ is the function returning the average cost obtained after five applications of the simulated annealing algorithm with 3M iterations to the problem defined in Fig. 2. x is a vector containing the temperature boundaries, $T_{\max}$ and $T_{\min}$. The search space ranges in [0.01, 400] for both temperatures and is constrained by (10):

$$T_{\min} < T_{\max}, \quad \forall\, T_{\min}, T_{\max} \in [0.01, 400] \tag{10}$$

The function $f(x)$ is modeled by using a Gaussian process whose covariance function is the square exponential kernel. The acquisition function used is the *expected improvement* acquisition function, as defined in Snoek et al. [15] (11):

$$a(x; \{x_n, y_n\}, \theta) = \sigma(x; \{x_n, y_n\}, \theta)(\gamma(x)\phi(\gamma(x)) + \mathcal{N}(\gamma(x); 0, 1)), \tag{11}$$

$$\gamma(x) = \frac{f(x_{\text{best}}) - \mu(x; \{x_n, y_n\}, \theta)}{\sigma(x; \{x_n, y_n\}, \theta)}, \tag{12}$$

where $x \in X$, $\{x_n, y_n\}$ is the pair of prior sample point x_n and its outcome y_n, θ represents the Gaussian process parameters, $\phi(\cdot)$ is the cumulative distribution function of the standard normal distribution, $\mathcal{N}(\cdot)$ is the standard normal distribution and x_{best} designates the best iteration observed during the optimization process. $\mu(\cdot)$ and $\sigma(\cdot)$ are the predictive mean and the predictive variance functions of the Gaussian process respectively.

Since the evaluation function is expensive to compute, one could want to take advantage from processor architectures and performs several evaluations in parallel. Some works suggested different methods for batch Bayesian optimization as in Snoek et al. [15] as well as González et al. [16]. As stated in [16], Batch Bayesian optimization is very helpful since the acquisition can be multi-modal especially during the first steps of the optimization process when the model is quite rough. In our case we used the local penalization method defined in [16]. This method consists in building, at each iteration, a batch of sample points by penalizing the acquisition function for each element in the batch as in (13):

$$x_k = \arg\min_{x \in X}(a(x) \prod_{i=1}^{k-1} \varphi(x; x_i)), \tag{13}$$

where x_k is the k-th element in the batch and $\varphi(x; x_i)$ ranges in [0, 1]. This function tends towards 0 when x_i is far from x. Thus, it reduces the acquisition function in a neighborhood where $f(x)$ will be tested by a batch's element already prepared. As a result, the several acquisition function's modes are explored since after being penalized the best mode could become lower than the second best. In our case, we used batches of six elements distributed over six cores on a single machine.

The resulting Gaussian process is depicted in the two first subplots of Fig. 9. The first contour plot represents the Gaussian process mean function and the second its standard deviation function. The x-axis corresponds to $T_{\max}$ and the

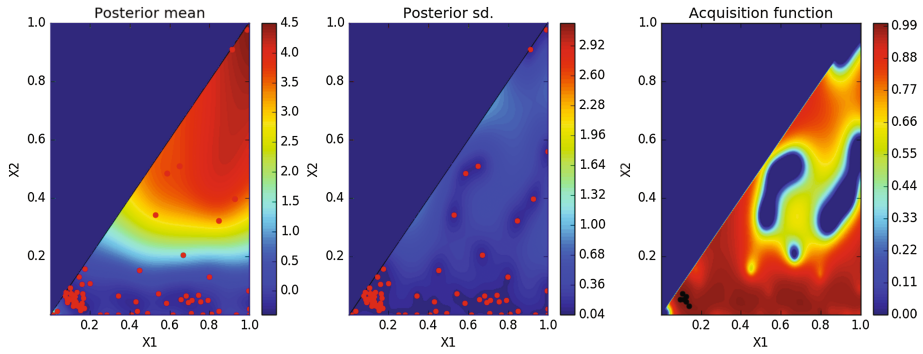

Fig. 9. Gaussian process, sample locations and acquisition function

y-axis to $T_{\min}$. During our first applications of the Bayesian optimization method we encountered difficulties by using directly the domain $X = [0.01, 400]$. However, scaling the search space between 0 and 1 produced smoother models. Also, the simulated annealing algorithm's outcomes are automatically scaled by the Bayesian optimization algorithm, according to the outcomes encountered during the optimization process. Since the aim is to find solutions that minimize the objective function, we are interested in low values of the mean function's contour plot. The dots in the contour plots represent the different observed iterations during the optimization process. It started by sampling 12 points in X and building a prior Gaussian process according to the observed simulated annealing outcomes. Then, by applying the local penalization algorithm the optimization converges around $T_{\max} = 45.58$ and $T_{\min} = 21.56$, as it can be seen in Fig. 9 where a large number of iterations are gathered in the corresponding region in the scaled search space ($T_{\max} = 0.1139$, $T_{\min} = 0.0539$). Thus, it is interesting to note that even by penalizing five times the acquisition function around $T_{\max} = 0.1139$ and $T_{\min} = 0.0539$, the optimizer still samples in this region of the search space.

The Gaussian process representation indicates that large values of $T_{\min}$ produce poor outcomes. In addition, the optimization process spent a considerable number of iterations in sampling values for $T_{\max}$ around 300. Such values give an acceptance rate of nearly 50% during the first iterations of the simulated annealing. Actually, such values of $T_{\max}$ seem to be a waste of iterations in the case of our problem since the optimizer concluded that an acceptance rate of 5% ($T_{\max} \sim 50$) is much more efficient. On the other hand, too low values of $T_{\max}$ yield relatively poor outcomes. Thus, it is important to have a non null acceptance rate during the first steps of the simulated annealing algorithm. The third subplot represents the acquisition function. While probabilities of sampling large $T_{\max}$ for low $T_{\min}$ are high, the next iterations (represented by dots in the left bottom corner) still gather around $T_{\max} = 0.1139$ and $T_{\min} = 0.0539$.

The first subplot in Fig. 10 depicts the distance, in the search space, between the different iterations over the optimization process. From 0 to 12, the optimizer did a random sampling in order to build a prior model. The 60 next iterations alternate between short and long distances. Since the optimizer evaluated iterations six by six, the differences in distances are due to the exploration of the different modes of the acquisition function. Actually, from the 70th iteration the distance between iterations remains low (excepted one peak around 80). Thus, it can be interpreted that the optimizer converged to the optimal solution. The second subplot indicates the best outcome encountered over the optimization process. During the last iterations, $f(x_{\text{best}}) = 7344.57$ which is lower than the average cost for 16M iterations in Fig. 8. Therefore, by using Bayesian optimization, we obtained a temperature configuration helping the simulated annealing algorithm to find cost-efficient solutions within a reasonable number of iterations. In addition, the optimization process took 15 h to find the best setup, saving us a considerable amount of time compared to the application of grid search methods.

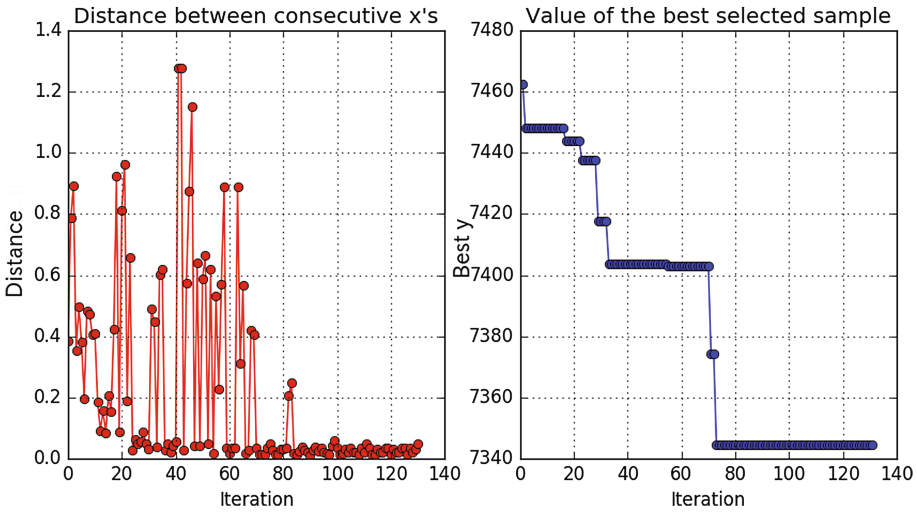

Fig. 10. Optimization convergence

6 Experiments

This section first investigates the effect of drones team's size on the average uncertainty rate by using the method presented in Sect. 4 as well as the hyper-parameters optimization in Sect. 5. The second experiment focuses on the system's abilities to generalize for other random problems' configuration. The third and the last experiment verifies the Bayesian optimizer convergence speed remains unchanged even for problems requiring more check points.

6.1 Team Size Effect

Table 2 summarizes the effect of team size on the cost, uncertainty rate and distance. The results are the average outcomes of ten applications of the simulated annealing algorithm by using the hyper-parameters determined in Sect. 5. Increasing the number of drones largely decreases the uncertainty rate since more locations can be checked in parallel, however, the uncertainty rate stagnates around 46.5% from a team of five drones. On the other hand, the distance explodes as the team's size increases. Since the uncertainty rate plays a larger part in the objective function than the distance, $\lambda = 0.1$, the simulated annealing returns obvious solutions having a low cost but a large distance. Increasing the value of the penalization coefficient λ fixed this problem. For example, by using three drones and $\lambda = 0.2$ the simulated annealing returned an average solution of cost 15788.49 with an uncertainty rate of 51.26% and a distance of 12552 cells. Thus, the value of λ plays an important role and should be chosen according to the problem.

Table 2. Cost evolution according to the team size

#drones	Cost	Uncertainty rate (%)	Distance (#cells)
1	8416.4	73.93	10229.8
2	7374.95	61.42	12323.2
3	6674.15	53.46	13280
4	6460.39	48.6	15998.5
5	6578.21	46.55	19222.4
6	6573.17	46.5	19224

6.2 Setup Limits

Similarly to the experiment in Sect. 5, this second experiment focuses on the influence of the number of iterations in the simulated annealing on the search performance. In the experiment of this subsection, we do not use the problem depicted in Fig. 2 anymore. The problem of interest here is for a team of two drones to visit 50 check points that are randomly selected.

Figure 11 represents the cost evolution over the number of iterations and the gain in improvement according the number of iterations. Since the number of locations to check is larger than the problem represented in Fig. 2, solutions are more expensive. A peak at 15M iterations indicates it exists solutions with a cost at least equal to a cost of 8057. The gain drops to 0 quickly since the improvement in cost has become smaller and smaller compared to the incremental computation time. On the other hand, there is a considerable gap between outcomes from simulated annealing algorithm applied with few iterations (1M∼3M) and applications with further iterations. While outcomes could be improved by optimizing the temperature range, more complex problems require more iterations.

6.3 Efficiency of Bayesian Optimization

One could wonder if the Bayesian optimizer in Sect. 5 requires more iterations to optimize the temperature range according to more complex problems. To check this ability, this experiment runs a new Bayesian optimization on a simulated annealing algorithm spending 3M iterations to find the best permutation on the 50 random check points problem used in the last section.

The first subplot of Fig. 12 represents the distance between two consecutive iterations during the optimization process. The first 12 iterations were selected randomly to explore the search space and build a prior Gaussian process. Until the 53th iteration the distance between them is large due to the exploration of four different modes in the acquisition function. However, from the 54th iteration and until the end of the optimization process the distance is nearly equal to zero. The optimizer focused every iteration in $T_{\min} \in [6, 29]$ and $T_{\max} \in [29, 52]$ during 13 epochs (78 iterations). The second plot represents the minimum cost encountered at each iteration. At the moment of the prior Gaussian process

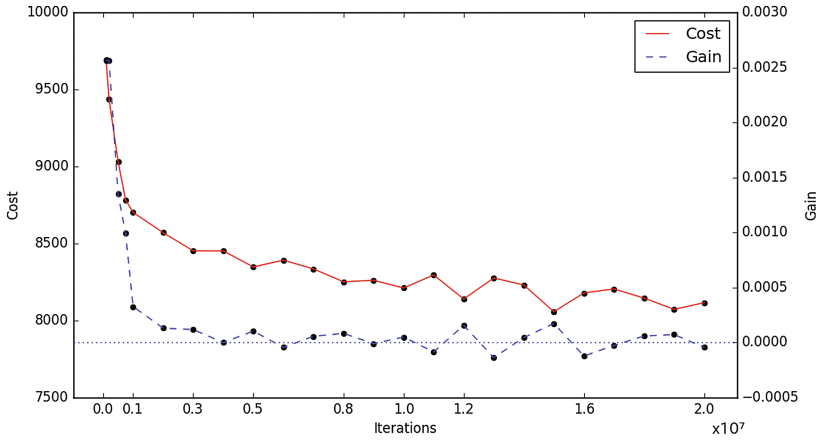

Fig. 11. Cost evolution according to the number of iterations for 50 random check points

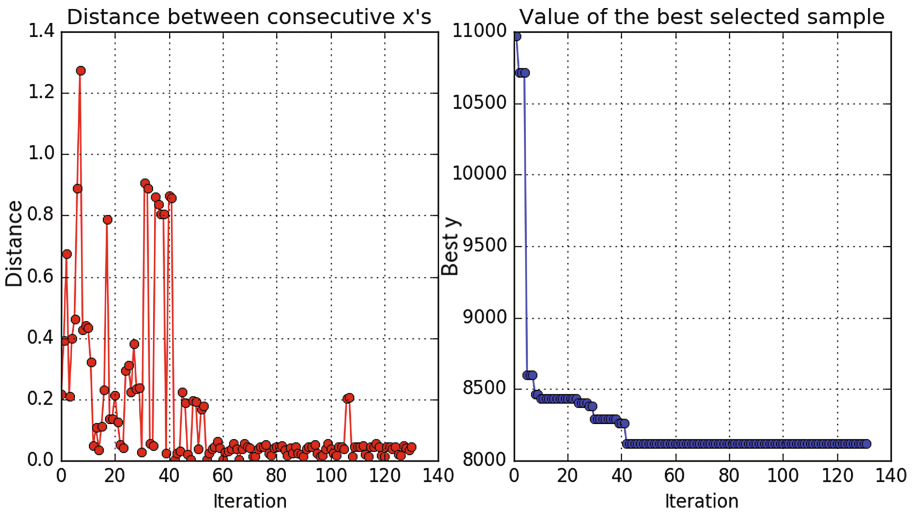

Fig. 12. Optimization convergence for a more complex problem

construction the minimum encountered cost is slightly lower than the one for the same number of iterations in Fig. 11. Actually, the best outcomes obtained at the end of the optimization process is equal to 8119.65 and then equivalent to the cost obtained for 12M iteration in Fig. 11. However, this best cost is larger than 8057 (best outcomes obtained in the previous experiment). Therefore, even by optimizing the temperature range 3M iterations is not enough to get the global optimum. On the other hand, the convergence speed is equivalent than the one in Sect. 5. Thus, the Bayesian optimization convergence speed seems independent from the permutation size treated by the simulated annealing algorithm.

7 Conclusion

This paper proposed a method representing the environment as a graph whose vertices are locations for which the situation has to be checked by a team of autonomous drones and edges are built by the A* algorithm. Thus, the problem of finding a path in the graph passing through every node is similar to a TSP. A simulated annealing algorithm is used to find the path which minimizes the environment's average uncertainty rate while considering the total travelled distance.

Whereas the simulated annealing algorithm's number of iterations plays an important role on the solution's quality, the temperature adjustment is also very important since it avoids getting blocked in a local optimum. In addition, the temperature range optimization allows to speed up convergence and then to reduce the number of iterations.

The temperature optimization problem was addressed by using Bayesian optimization. This method has shown capacities to find the optimum in a relatively low number of iterations. Furthermore, it saved us weeks of computations compared to the use of grid search methods. On the other hand, in its current definition the optimization process requires to fix the simulated algorithm's number of iterations, but this number depends on the problem. Further work would consist in making the optimization process more general. To do so, the Bayesian optimizer should consider both the number of iteration and the temperature range. However, to simply search the optimal number of iterations would always results on the search space upper bound. To cope with this issue, a possible idea is the optimization of the *expected improvement per second* as proposed in Snoek et al. [15]. By doing so, the optimizer looks for a setup which provides good outcomes of the function to optimize (in our case the simulated annealing algorithm) while requiring its computation time to be as low as possible.

Finally, the algorithm used for the task allocation is too simple and reconsidering this step would certainly improve the solution quality. However, the proposed method is adjustable. Thus, it is independent from the used task allocation algorithm as well as the A* algorithm or the simulated annealing algorithm. Further work would consist in proposing a task allocation algorithm which considers different drones' start points, different drone's characteristics (autonomy, speed, ...) and which allocates tasks in a less greedy way.

References

1. Castro, M., SöRensen, K., Vansteenwegen, P., Goos, P.: A memetic algorithm for the travelling salesperson problem with hotel selection. Comput. Oper. Res. **40**(7), 1716–1728 (2013)
2. Liu, C., Kroll, A.: Memetic algorithms for optimal task allocation in multi-robot systems for inspection problems with cooperative tasks. Soft Comput. **19**(3), 567–584 (2015)
3. Jose, K., Pratihar, D.K.: Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods. Robot. Auton. Syst. **80**, 34–42 (2016)

4. Richards, A., How, J.P.: Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301), vol. 3, pp. 1936–1941. IEEE (2002)
5. Richards, A., How, J., Schouwenaars, T., Feron, E.: Plume avoidance maneuver planning using mixed integer linear programming. In: Proceedings of the AIAA Guidance, Navigation, and Control Conference, pp. 6–9 (2001)
6. Richards, A., Bellingham, J., Tillerson, M., How, J.: Coordination and control of multiple UAVs. In: AIAA guidance, navigation, and control conference, Monterey, CA (2002)
7. Jevtić, A., Andina, D., Jaimes, A., Gomez, J., Jamshidi, M.: Unmanned aerial vehicle route optimization using ant system algorithm. In: 2010 5th International Conference on System of Systems Engineering (SoSE), pp. 1–6. IEEE (2010)
8. Osaba, E., Carballedo, R., Yang, X.-S., Diaz, F.: An evolutionary discrete firefly algorithm with novel operators for solving the vehicle routing problem with time windows. In: Yang, X.-S. (ed.) Nature-Inspired Computation in Engineering. SCI, vol. 637, pp. 21–41. Springer, Cham (2016). doi:10.1007/978-3-319-30235-5_2
9. Osaba, E., Yang, X.S., Diaz, F., Onieva, E., Masegosa, A.D., Perallos, A.: A discrete firefly algorithm to solve a rich vehicle routing problem modelling a newspaper distribution system with recycling policy. Soft Comput. 1–14 (2016)
10. Careaga, C.: Python A* pathfinding (with binary heap). ActiveState Code. http://code.activestate.com/recipes/578919-python-a-pathfinding-with-binary-heap/
11. Acevedo, J.J., Arrue, B.C., Maza, I., Ollero, A.: Distributed approach for coverage and patrolling missions with a team of heterogeneous aerial robots under communication constraints. Int. J. Adv. Robot. Syst. **10**, 1–13 (2013)
12. Perry, M.T.: simanneal. https://github.com/perrygeo/simanneal
13. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., et al.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
14. The GPyOpt authors: GPyOpt: a Bayesian optimization framework in python (2016). http://github.com/SheffieldML/GPyOpt
15. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, pp. 2951–2959 (2012)
16. Gonzalez, J., Dai, Z., Hennig, P., Lawrence, N.: Batch Bayesian optimization via local penalization. In: Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, pp. 648–657 (2016)