Algorithm Methods Comparison on TSP

Xian Wu[1,#], Yu Hao[1,#,*], Kaiyu Guo[2], Huyu Wu[2]

[1]Institute of Pittsburgh, Sichuan University, Chengdu, China

[2]Institute of Computer, Sichuan University, Chengdu, China

[#]Xian Wu and Yu Hao contributed equally to this study

*Corresponding author

Keywords: combinatorial optimization, TSP, evolutionary algorithms, reinforcement learning

Abstract: In the previous research, other researchers focus on the improvement and hybrid algorithms that can be used in solving TSP. However, this article indicates eight different algorithms to solve TSP which includes Advantage Actor Critic (A2C), Deep Q-Network (DQN), Proximal Policy Optimization (PPO) and five number of evolutionary algorithms: Particle swarm optimization (PSO), Ant Colony Optimization (ACO), Simulated Annealing (SA), Genetic Algorithm (GA), Tabu Search (TS). The article states that it's significant to do research on TSP, and builds a specific question initially. Next, the article shows the principle of each algorithm and fundamental operation step. Then, author do an internal comparison in the evolutionary algorithms, the step is also known as adjusting parameter. This step compares the specific time and distance-cost among different parameter setting. Obviously, it can make the researcher consider the better performance of the algorithms in different conditions. Finally, the author compares the different behavior between those eight algorithms, illustrating the path figure and converge figure of each algorithm. Path figures and converge figures illustrate how the iteration times affect the converge steps and specific path. All the works done contributes to the research of TSP to some extents.

1. Background

1.1. Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is a combinatorial problem, which elaborates the specific path to reach the list goals of the problem. Importantly, all the goals in the TSP can only be reached once at most. In order to find the shortest path to meet the expectation, many algorithms use different method based on different fundamentals. The problem is very classical and worthwhile to research.

1.2. Specific Introduction

In the traveling sales problem, the authors try to get the final result as quickly as possible, as stable as possible, and as accurate as possible. Then, the author chooses three algorithms from reinforcement learning algorithms, which include A2C, DQN, and PPO. The author also chooses five algorithms from evolutionary and swarm algorithms: PSO, ACO, SA, GA, and Tabu search. All of the algorithms mentioned above are advantageous in solving combinatorial optimization; and are successfully applied in the TSP field. In this article, the authors try to determine these eight algorithms' advantages and disadvantages. Specifically, the author will use Python to run codes in a unique data environment. Then, the author will analyze the result of the output. Finally, compare the convergence and path graphs among different algorithms to conclude.

In the previous research, some authors design many algorithms, including reinforcement learning and evolutionary algorithms. Additionally, all the algorithms perform well in their behavior in solving TSP questions. Some searchers focus on the improvement of the specific algorithms, some searchers vote on the number to be set in the parameters, and others search about generating new practical algorithms like hybrid neuro evolution algorithms.

Although the direction of their research is various, the contribution of the combination optimization is significant. In this research, the author will focus on the new aspect of the question, which is the comparison of different algorithms in the same environment. Every algorithm has its advantages and disadvantages in solving the problems in TSP. Due to the different original principles of algorithms, the author will analyze the different properties. Finally, the author will try to get a general conclusion depending on the data.

2. Building the TSP Model

2.1. Case Study Principle

The Figure 1 below shows a distribution of the cities on Graph G, the map of TSP shows the goal and also the specific destination. These cities must be included in the route T. Additionally, the red points indicate the specific cities and the blue lines illustrate the route T. The goal of authors' project is to minimize the distance of route T.

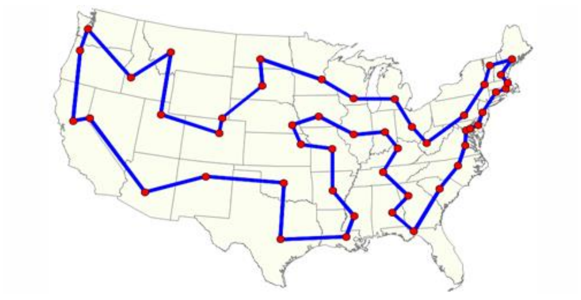


Figure 1 The map of TSP.

Formally, the graph can be described by G= (V, A), where V is the collection of N vertices [1], $A = \{(I,J): I,J \in V\}$ is the set of edges with cost $c: A -> R, c: (i,j) -> c_{i,j}$,

Where $c_{i,j}$ is the distance from node i to node j.

Lastly, X_{ij} is a Boolean variable such that X_{ij} =1 if the edge (i,j) is active. The formulation of this question is:

$$\max - \sum_i \sum_j X_{ij} c_{ij} = f(T) \tag{1}$$

$$\text{s.t} \sum_i X_{ij} = 1 \quad \forall i: (i,j) \in A \tag{2}$$

$$\sum_i X_{ji} = 1 \quad \forall i: (j,i) \in A \tag{3}$$

$$\sum_{i \in S, j \in V \setminus S} X_{ij} \geq 2 \quad \forall\ S \subset V, S \notin \{\emptyset, V\} \tag{4}$$

$$X_{ij} \in \{0,1\}$$

3. Introduction to Each Algorithm

3.1. Advantage Actor Critic (A2C)

Define A2C is one of the members of the actor-critic family. A2C is parallel and is able to support every kind of spaces. An actor-critic algorithm is a policy gradient algorithm which uses function estimation in place of empirical returns in the policy gradient. In addition, the actor-Critic algorithm is a Reinforcement Learning agent that combines value optimization and policy optimization approaches. Specifically, the Q-learning and Policy Gradient algorithms are combined by the the Actor-Critic.

Actor-Critic has two function approximations (two neural networks): The parameter of policy function (Actor) is θ: $\pi_\theta(s, a, \theta)$ [2]. The Actor is a PG algorithm that decides on an action to take. The parameter of value function (critic) is ω: $\widehat{q_w}(s, a, \omega)$.

A2C uses the Advantage function as Critic, and it does not use Action value function, which is an advantage. This method has an idea that the Advantage function calculates how better taking that action at a state is compared to the average value of the state. It is subtracting the mean value of the state from the state action pair:

$$A(s,a) = Q(s,a) - V(s)$$

Q(s,a) is Q value for action *a* in state *s*, and V(s) refers to the average value of that state.

The problem with implementing this advantage function is that it requires two value functions — Q(s,a) and V(s) and TD error is a good estimator of the advantage function.

$$A(s,a) = r + \gamma V(s') - V(s) \quad (5)$$

```
Algorithm 1 Advantage actor-critic pseudocode
Input:
Output: optimized
Initialize: ; s from environment
Repeat for each step
Perform action according to policy  $\pi(a|s;\theta)$
Get s', reward from the environment
        $\delta = reward + \gamma V_\omega(s') - V_\omega(s)$
        $\theta = \theta + \alpha \nabla_\omega \log\pi_\theta(s,a)\delta$
        $\omega = \omega + \beta\delta\nabla V_\omega(s)$
        $s \leftarrow s'$
Until num of steps reaches the limit
```

3.2. Deep Q-Network (DQN)

Q(s,a) is built to store Q-values for the combination of s and an in Q-Learning [3]. However, when the memory and computation required for the Q-value algorithm are high, the Q-Learning function should be instead by an approximator -- Deep Q-Network (DQN): $Q(s,a;\theta)$with parameters θ. DQN is a neural network used to learn a Q-function. As most reinforcement learning is associated with complex (typically visual) inputs, the initial layers of a DQN usually are convolutional.

The optimized loss functions at iteration i, which is to estimate this network is that:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(y_i{}^{DQN} - Q(s,a;\theta_i))^2] \quad (6)$$

With

$$y_i{}^{DQN} = r + \gamma \max_{a'} Q(s',a';\theta^-) \quad (7)$$

Where θ^- is the parameters of a fixed and separate target network. Also, there is a crucial innovation that to make the parameters of the target network $Q(s',a';\theta^-)$ to stay for a fixed number of iterations while updating the online network $Q(s,a;\theta_i)$ by gradient descent and the update is:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(y_i{}^{DQN} - Q(s,a;\theta_i)\nabla_\theta, Q((s,a;\theta_i)] \quad (8)$$

This approach is model-free because this environment produces the states and rewards. It is also off-policy because these states and rewards can be obtained with a behavior policy (epsilon greedy in DQN) and it is different from the online policy that is learned.

Another important ingredient that make the DQN to proceed is the experience replay. During learning, the agent accumulates a dataset$\mathcal{D}_t = \{e_1, e_2, \dots, e_t\}$of experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ from many episodes. The sequence of the losses can be illustrated by the function below:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim u(D)}[(y_i{}^{DQN} - Q(s,a;\theta_i))^2] \quad (9)$$

3.3. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an architecture and it can avoid over much policy updates and it makes the agent's training more stable. The PPO algorithm combines ideas from A2C which is an algorithm has multiple workers and TRPO that uses a trust region to improve the Actor. The main method is that after an update, the new policy would not be too different with the old or original policy [4]. Thus, PPO uses clipping to avoid the over much update.

```
Algorithm 2 Deep Q-Learning with experience replay
Initialize: replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights $\theta$
Initialize target action-value function $\widehat{Q}$ with weights $\theta^{-} = \theta$
For episode = q, M do
Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\emptyset_1 = \emptyset(s_1)$
For t=1,T do
With probability $\varepsilon$ select a random action $a_t$
Otherwise select $a_t = argmax_a Q(\emptyset(s_t), a; \theta)$
Execute action $a_t$in emulator and observe reward $r_t$and image $x_{t+1}$
Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\emptyset_{t+1}$=$\emptyset(s_{t+1})$
Store transition $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ in D
Sample random minibatch of transitions $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ from D
Set $y_j = \begin{cases} r_j & if\ episode\ terminates\ at\ step\ j+1 \\ r_j = \gamma max_{a'} \hat{Q}(\emptyset_{j+1}, a'; \theta^{-}) & otherwise \end{cases}$
Perform a gradient descent step on $(y_j - Q(\emptyset_j, a_j, \theta))^2$ with respect to the network parameters $\theta$
Every C steps reset $\hat{Q} = Q$
End for
End for
```

It should to use a loss function which combines the policy surrogate and a value function error term if the neural networks architecture shares parameters bewteen the policy and value function. The following objective which is approximately every iteration is illustrated below:

$$\mathrm{L_t^{CLIP+VF+S}(\theta) = \widehat{\mathbb{E}_t}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]} \tag{10}$$

Where $\mathrm{c_1}$ and $\mathrm{c_2}$ are coefficients, S denotes an entropy bonus, and $\mathrm{L_t^{VF}}$ is a squared-error loss $\mathrm{(V_\theta(s_t) - L_t^{targ})^2}$.

One of the methods of policy gradient execution, popularized in [Mni+16] and well-suited for use in current neural networks, and it runs the policy for T timesteps and T is extremely shorter than the episode length, and uses the samples that are collected for an update. This style requires an advantage estimator that does not look beyond timestep T. The estimator used by [Mni+16] is

$$\widehat{A_t} = -V(s_t) + r_t + \gamma r_{t+1} \widehat{+\ldots+} (\gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \tag{11}$$

Where t specifies the time index in [0, T], within a given length-T trajectory segment.

In general, synthesize these choices, a truncated version of generalized advantage estimation can be used, which reduces to the above Equation when $\lambda = 1$:

$$\widehat{A_t} = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{12}$$

Where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

A proximal policy optimization (PPO) algorithm that uses fixed-length trajectory segments is shown below.

```
Algorithm 3 PPO, Actor-Critic Style
For iteration=1, 2,... do
For actor=1, 2, ...,N do
Run policy $\pi_{\theta_{old}}$ in the environment for T timesteps
Compute advantage estimates $\widehat{A_1}, \ldots, \widehat{A_T}$
End for
Optimize surrogate L wrt $\theta$, with K epochs and minibatch size $M \le NT$
$\theta_{old} \leftarrow \theta$
End for
```

3.4. Particle Swarm Optimisation (PSO)

Particle swarm optimization (PSO) is a type of the bio-inspired algorithms, and it is easy to search for an optimal solution in the solution space. It is much different from other optimization algorithms, because it only has few hyperparameters and it only needs the objective function, meanwhile, it is not dependent on any differential form or the gradient of the objective.

Because there is no better way to know if p- or g-increment should be larger. Therefore, these terms were also stripped out of the algorithm [5]. The current simplified PSO now adjusts velocities by this formula

$$vx[][] = vx[][] + \\ 2 * rand() * (pbestx[][] - presentx[][]) + \\ 2 * rand() * (pbestx[][gbest] - presentx[][] \tag{13}$$

Algorithm 4 PSO algorithm

```
Procedure PSO
For each particle i
Initialize velocity Vi and position Xi for particle i
Evaluate particle i and set pBesti = Xi
End for
gBest = min {pBesti}
while not stop
For i=1 to N
Update the velocity and position of particle i
Evaluate particle i
If fit(pBesti)< fit(gBest)
gBest= pBesti;
end for
end while
print gBest
end procedure
```

3.5. Ant Colony Optimization (ACO)

ACO, also known as ant colony optimization, is a model with parallel computing support and continuous optimization ability. The general idea of creating this kind of algorithm is foraging behaviors from the real ants. When the ants search for food at the beginning, they randomly explore the nest. As soon as the ants find the food, they return to the nest. When the ants move from place to place, they will leave a pheromone, leading the ants to find the route as well as possible [6]. ACO, as an algorithm, is developed to be a metaheuristic for combinatorial optimization. It is a good choice for solving TSP questions.

The ACO metaheuristic consists of three algorithmic components that are gathered in the Schedule Activities construct.

Algorithm 5 Ant Colony Optimization metaheuristic

```
While termination conditions are not met, do
Schedule Activities
Ant Based Solution Construction ( )
Pheromone Update ( )
Daemon Actions ( ) {optional}
end Schedule Activities
end while
```

Ant Based Solution Construction () is Artificial ants construct solutions from sequences of solution components taken from a finite set of n available solution components C={c_{ij}}.

Pheromone Update ( ) is The aim of pheromone update is to increase the pheromone values associated with good or promising solutions and decrease those that are associated with bad ones.

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho)\tau_{ij} + \rho\Delta\tau & \text{if } \tau_{ij} \in s_{ch} \\ (1-\rho)\tau_{ij} & \text{otherwise} \end{cases} \quad (14)$$

Where $\rho \in (0,1]$ is the evaporation rate. Pheromone evaporation is needed to avoid too rapid a convergence of the algorithm.

Daemon Actions ( ) is Daemon actions that can be used to implement centralized actions which cannot be performed by single ants.

3.6. Simulated Annealing (SA)

SA is the abbreviation of simulated annealing, which is also a module to solve combinational optimization. SA is widely used in evolutionary and swarm algorithms, and it can be treated like a synchronous approach with occasional enforcement of the best solution. It is different from the traditional hill-climbing algorithm. SA avoids falling into the local optimum. Just like annealing at a high temperature, the algorithm accepts many solutions. Then, the algorithm refuses to accept the non-optimal solution when the temperature gets lower. In this case, the probability of finding the best solution increases.

The basic SA approach is shown below.

The initial solution is generated randomly by assigning a value to each dimension i, i=1... N, in the interval [lb_i,ub_i] and $lb_i$is the lower bound and the ub_i is higher bound for variable i.

The neighbor solution, x^{k+1}, is created below:

$$x_i^{k+1} = x_i^k + u(ub_i - x_i^k), \operatorname{sgn}(u - 0.5) < 0 \quad (15)$$

And

$$x_i^{k+1} = x_i^k - u(x_i^k - ub_i), \operatorname{sgn}(u - 0.5) \geq 0 \quad (16)$$

Else,

$$x_i^{k+1} = x_i^k,$$

where $u \in U[0,1]$

The new function value is calculated, and the solution is accepted if the move improves the current solution.

$$P\,A(\Delta, t^k) = \exp(-(\frac{\Delta}{f(x^k)})t^k) \quad (17)$$

Where PA is the probability of acceptance, $\Delta$is the difference between $f(x^{k+1})$ and $f(x^k)$, and t^k is the temperature in iteration k.

Algorithm 6 Template of simulated Annealing Algorithm
Input: Cooling schedule. S=s_0; Repeat Repeat Generate a random neighbor s' $\Delta E = f(s') - f(s)$ If $\Delta E \leq 0$ Then s=s' Else Accept s' with a probability $e^{\frac{-\Delta E}{T}}$; Until Equilibrium condition T=g(T) Until Stopping criteria satisfied Output: Best solution found

3.7. Genetic Algorithm (GA)

The idea of generating this algorithm comes from the evolution of the genetics of human beings. Initially, the data of the algorithm is chaotic. However, the algorithm gives the data several evolutionary steps to be a feasible solution. GA will treat the data like each specific gene in the chromosomes. Moreover, GA will also use the method of genetic recombination and genetic mutation to get the new chromosomes. Judging from the fitness value of chromosomes, GA will show us the optimal solution to the problems.

```
Algorithm 7 Genetic Algorithm
Generate initial population of size nP
Evaluate the initial population according to fitness criteria
While ( current_generation< nG ) {
Breed rC*nP new solutions from current population
From child solution via crossover
If( RandomRange(0.0, 1.0) < rM    )
Mutate the child solution
Evaluate the child's solution according to fitness criteria
Add child to population ( MaxPop = nP* (1+ rC)
Remove the rC8np least-fit solutions from the population
}
Return [most fit member of population
```

Where np is the base population size, nG is the number of generations, rC is the crossover rate, and rM is the mutation rate.

3.8. Tabu Search (TS)

Tabu search is an advantageous algorithm for solving combinatorial optimization problems. Due to its unique algorithm, it will also build a tabu list to prevent redundant data changes, which benefits the accuracy of the result.

There are two vital elements in a simple form of tabu search. That is constraining the search by classifying sure of its moves as forbidden (i.e., Tabu) and freeing the search by a short-term memory function that provides "strategic forgetting."

The main procedure of simple Tabu Search is shown below:

Select an initial $x \in X$ and let x* : = x. Set the iteration counter k = 0 and begin with T empty. Then, if S(x) - T is empty, go to the Step 4.

Else, set k := k + 1 and select $s_k \in S(x) - T$ such that $s_k(x) = \mathrm{OPTIMUM}(s(x){:}\, s \in S(x) - T)$. Let x:=$s_k$(x).

If c(x) < c(x*), where x* denotes the best solution currently found, let x* := x.

If a number which is chosen of iterations has elapsed either in total or since x* was last improved, or if S(x)-T = $\emptyset$ upon reaching this step directly from Step 2, stop. Else, update T and return to Step 2.

4. Set Up and Parameter Comparing

4.1. The Set for Reinforcement Study Algorithms

Those algorithms include A2C, DQN, and PPO. There is no parameter set for these algorithms, which indicates they do not have to be set.

4.2. The Set for Evolutionary Algorithms

According to the different principle used in the different evolutionary, author try to find the fundamental for those different algorithms. Importantly, collect the experiment result set by the different number of parameters. By analyzing the cost and time used in different conditions, therefore, the authors can get the trend of the data.

4.2.1. Set Up for ACO

ACO algorithm can be used in the TSP question; it can be treated as ants travel through different cities. When the ants travel, they will choose the closest distance between the two cities. Here comes an equation:$p_{ij}^{k} = \frac{(\tau_{ij}^{\alpha})(\vartheta_{ij}^{\beta})}{\sum(\tau_{ij}^{\alpha})(\vartheta_{ij}^{\beta})}$, it shows the probability of coming to the next city. τ_{ij} is the concentration of the pheromone; ϑ_{ij} is the inverse of the distance between the two cities. Additionally, α and β are the parameters of the Equation. Therefore, in order to solve the question, the authors definite a distance function and the primary data. Then, the authors adjust the parameter of the number of ants and the α and β to get the optimization solution for this specific question.

Using the scientific method of controlling parameters. The authors change the value of a parameter. Meanwhile, the authors also keep the other parameter the same. The reason is to find the vital parameter; additionally, the authors can find a rough range to get the optimal solution by adapting the parameter. In order to eliminate the occasional error made by Python code, the authors will do the same value five times to get the average value. The following Table 1 shows us the details of the ant comparison:

Table 1 Internal comparison in Ant-count

Ant-count	5	25	45	65
Totally cost	469.69	458.72	457.70	454.60
Totally time	5.66s	28.36s	51.77s	72.63s

Analyzing the data shown in the Table 1 tells us that with the increase of the number of Ant-count, the totally cost of the time will increase. One more thing to notice is that there is a significant change in totally cost during the range from 5 to 25. However, it can also be found that the cost is indeed reducing to a slight degree.

Table 2 Internal comparison in pheromone

Pheromone	1	1.5	2	2.5
Totally cost	466.71	497.11	479.80	480.95
Totally time	5.71s	5.61s	6.21s	5.63s

From the result of Table 2, the time cost for each pheromone is almost identical; from the concentration of 1 to the concentration of 1.5, the totally time decreases only at a small number. From 1.5 to 2, the totally time increases instead. Finally, the totally cost of time will decrease again.

As for the totally cost, it is wired to find the pheromone increase from 1 to 1.5. Then from 1.5 to 2, the cost reduces. Last, the cost is kept at a range from 2 to 2.5.

Table 3 Internal comparison in Beta

Beta	2	3	4	5
Totally cost	467.25	465.82	461.63	462.54
Totally time	5.70s	5.69s	5.62s	5.63s

For parameter beta, authors choose four different numbers. From the Table 3 data author did, authors realized there is no significant difference between the four different totally time costs. It can also be easily found that the cost is similar to each other. Therefore, maybe the parameter Beta did not play an essential role in the algorithms.

4.2.2. Set Up for SA

In the SA algorithms, to solve the problem of TSP, the authors set the parameter for the initial temperature, the initial distance, the cycle index, and the maximum of the cycle index. The first step is to exchange the position of two cities in a random manner; next, there will be a new distance. The SA will compare the number of the initial distance and the new distance [7]. Here comes the equation $\text{delta D} = D_n - D_i$. If the delta D is smaller than zero, it will return the number 1; if the delta D is more extensive than zero, it will return the number $1/\exp\left(\frac{\text{delta D}}{T}\right)$. Additionally, the return number will also be compared to evenly distributed random values from zero to one. Finally, if the number is more significant than the random values, it will be the new solution. As the temperature falls and the cycle of index increases, it will get the combination optimization solution.

Since the essential parameters for SA are the initial temperature and the alpha in lower the temperature. The authors try to experiment with different performances in different parameters.

Table 4 Adjusting parameter of Initial T

Initial T	50000	100000	150000	200000	500000
Totally cost	463.49	461.67	457.588	448.09	457.00
Totally time	3.038	3.00	2.97	2.98	3.03

As Table 4 shown above, the temperature should not be as high as possible. Instead, according to the experiments' data, the temperature increase leads to the total cost decrease from 50000 to 200000. However, it seems to have a limitation; in the range from 200000 to 500000, the totally cost increases instead.

Table 5 Adjusting parameter of alpha

Alpha	0.95	0.98	0.99
Totally cost	449.04	459.54	456.18
Totally time	3.00	2.98	2.996

Some research said the best alpha set in SA should be 0.99 or 0.98. However, this Table 5 indicates that the alpha 0.95, 0.98, and 0.99 have a slight difference in the time cost. It can almost be ignored by the effect of time. In the aspect of cost, the number will increase firstly and decreases at the Alpha 0.98. Finally, the parameter gets to 0.99; the cost reduces a little.

4.2.3. Set Up for GA

As mentioned above, GA uses genetic evolution to solve problems. It is advantageous to use in figuring out the TSP question. The authors need to set the iterations of the chromosomes and the probability of mutation and recombination. Each city in the TSP can be represented like a single gene in chromosomes. GA changes the position of the genes in chromosomes to generate new chromosomes. With the increase of the iterations, the fitness value will decrease because GA chooses the chromosomes with the low fitness value as the optimal solution to the problem.

The important stuff to do is to adapt the particular value of the parameter. For GA, there are two parameters that play an essential role in the total cost and time of TSP. The authors will also do two experiments to check which values of mutation and iteration are optimal.

Table 6 checking SA iterations number

Iteration	100	500	1000	2000
Totally cost	696.91	482.14	472.59	459.618
Totally time	1.19	5.93	11.9	23.61

There will be no optimal solution when the iteration value is small. Instead, the local maximum will exist. In the other aspects, the iterations value is very high, and there will cost much time to solve it. From the Table 6 authors can see with the number of iterations increases, the time will increase for sure. The authors can also notice that the cost will keep decreasing with the rising iteration times.

Table 7 checking the probability of mutation value

Probability of mutation	0.001	0.01	0.05	0.1
Totally cost	827.1	552.48	458.30	468.08
Totally time	11.76	12.06	12.47	11.86

After checking the result of Table 7, the author figured out that when the probability is small, there will not be many new genes generated by the mutation. Meanwhile, when the probability is high, it will be a random search, which harms accurate cost and time. The total time used by the increasing probability of mutation, which wound increase firstly. However, it wound reduce when the probability is from 0.05 to 0.1. As for the cost, it reduces from 0.001 to 0.05. The cost will rise again when the probability increases from 0.05 to 0.1. The probability at 0.05 may be the inflection point.

4.2.4. Set Up for Tabu

In the combinatorial optimization of TSP, there is a way to show the result: $D = \{x = (i_1, i_{2....}i_n) | n \in N\}$, defining its field mapping 2-opt, the field structure will be $C_n^2 = n(n-1)/2$. Therefore, it comes to lots of probability [8]. Tabu changes two cities' positions each time to compare the distance between the new one and the original. The Tabu will mark the path which is not the best solution. In this case, it can escape from the local optimization. It is also significant for the Tabu search to build a list that prevents the algorithm from running in the same cycle [9]. All the paths that were marked before will not allow us to get into the new cycle.

In tabu search, the tabu list is the most basic stuff. Therefore, the parameter of tabu-long is also significant for the authors to set. The authors will build an experiment to find the best value among the domains in this case.

Table 8 considering the effect of Tabu long in different number

Tabu long	10	100	1000
Totally cost	481.92	451.05	484.99
Totally time	40.14	40.36	40.50

Compared to other algorithms, the result score of cost is stable. In Table 8, the authors get the same result by five times. Due to the existence of the tabu list, the algorithm can have the "memory." It can explain the reason why the result is stable. After seeing this result data collected, the author finds time spent on finding the path in TSP is similar. The total cost decreased initially, and it increased with the Tabu list's length.

4.2.5. Set Up for PSO

As for the PSO algorithms, the authors follow the previous research other authors did. In this algorithm, the authors imitated the behavior of birds. The optimal solution will be the food source that needs to be found by birds. When running the algorithms, the birds must change the information they get to find the optimal solution. The renew velocity in several dimensions is shown [10]:

$$V_{id} = \omega V_{id} + C_1 \text{random}(0,1)(P_{id} - X_{id}) + C_2 \text{random}(0,1)(P_{gd} - X_{id})$$

C_1 and C_2 are the velocities constant, and the $\text{random}(0,1)$ indicates the number of it will be randomly chosen from 0 to 1. P_{id} shows us the dimension, and the P_{gd} presents the optimal solution that the algorithms can find [11]. Using this concept, the author defined the distance sum function. Then initialize the parameter of this Python program, and the author chooses the different iteration times of PSO. The result is below:

Table 9 the comparison of iteration in PSO

Iteration	5	10	50	100	1000
Totally Cost	478.62	459.62	443.82	428.98	430.75
Totally Time	8.48	15.61	80.85	163.25	1910.50

As all authors can see from the Table 9, the total time wound increase with the iteration rose. As for the cost in the path to find the optimal solution, it would decrease ranged from 5 to 100. Additionally, in the range from 100 to 1000, the cost will increase instead. The authors also realized it might change from 100 to 1000. In other words, the change was not significant.

5. Comparing Different Algorithms and Discussions

5.1. The Optimal Setting for the Case Study

First of all, the case has an optimal target solution shown with the optimal cost of 426 with a number of 51 cities in the below Figure 2:

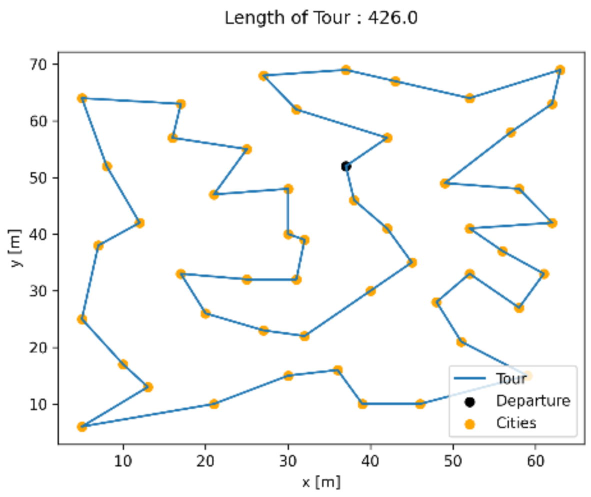
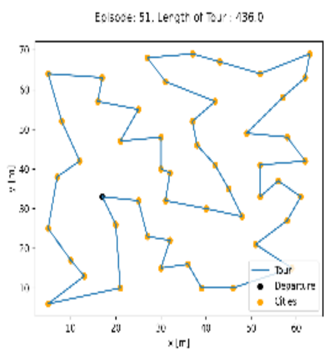
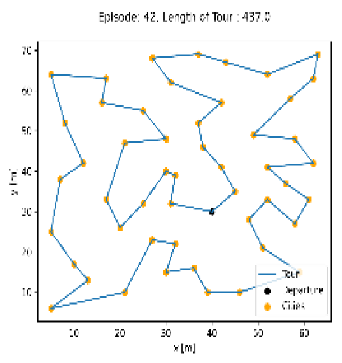
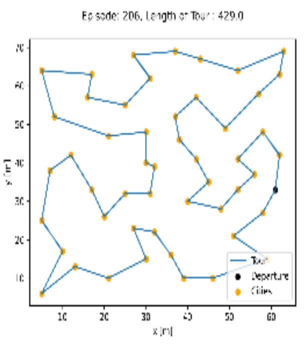


Figure 2 The figure of optimal solution

5.2. Comparing Reinforcing Algorithms

5.2.1. The Path, Cost, and Time Used to Compare

Table 10 Comparison of Totally cost and time

	A2C	DQN	PPO
Totally cost	436	437	429
Totally time	201.15s	279.36s	527.46s
Figure			

Judging from the cost of each algorithm. It was seen from Table 10 that PPO is the most closed to the optimal solution. In reinforcement algorithms, all of the algorithms above performed well and were all approached to the optimal solution.

5.2.2. The Coverage Figure

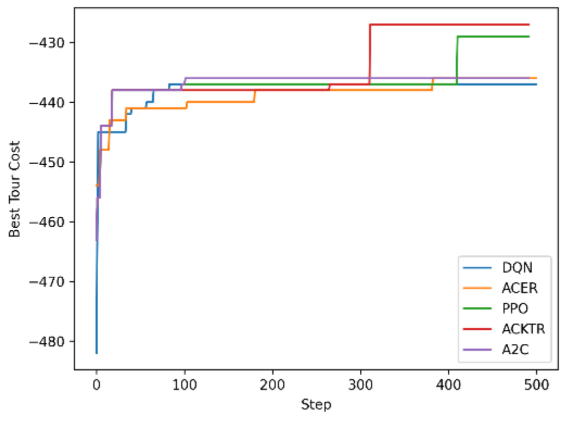


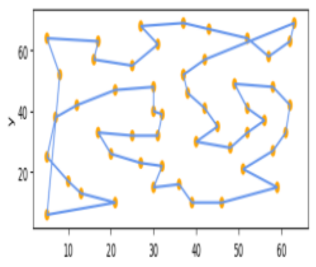
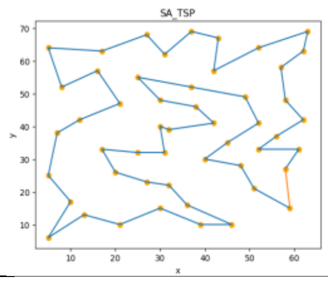
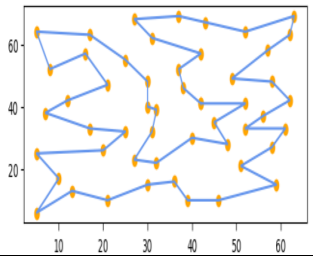
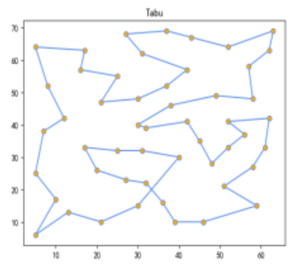
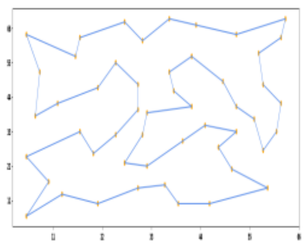
Figure 3 The coverage figure

The figure 3 above shows us the specific coverage step in the reinforces algorithms; the authors realize the first few steps for all three algorithms are similar [12]. In contrast, the PPO has a significant change in step about 400, allowing it to reduce the cost and be closer to the optimal solution.

5.3. Comparing the Different Evolutionary Algorithm

Like 5.2, authors were curious about the behavior and path did by different evolutionary algorithms. Additionally, choosing the specific parameters which are compared in part 4 to make sure the preciseness:

Table 11 Comparison of cost and time

	Totally cost	Totally time	Figure
ACO	458.72	28.36s	
SA	442.96	2.91s	
GA	453.73	11.84s	
Tabu	451.04	39.75	
PSO	449.93	66.18	

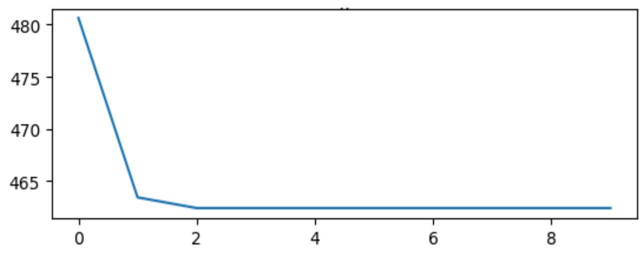
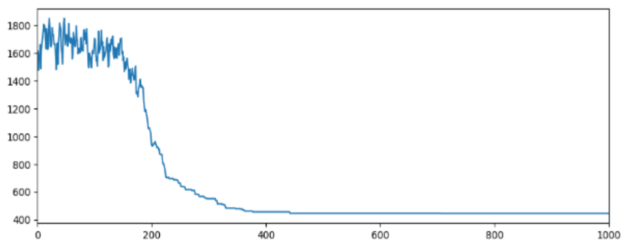
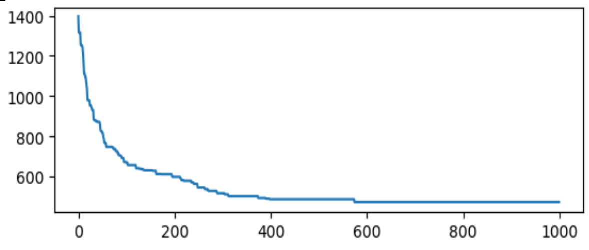
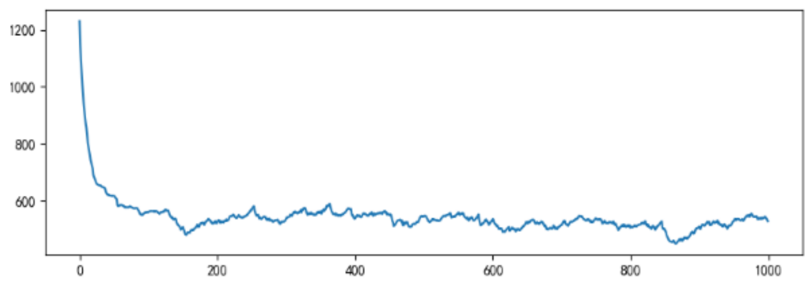
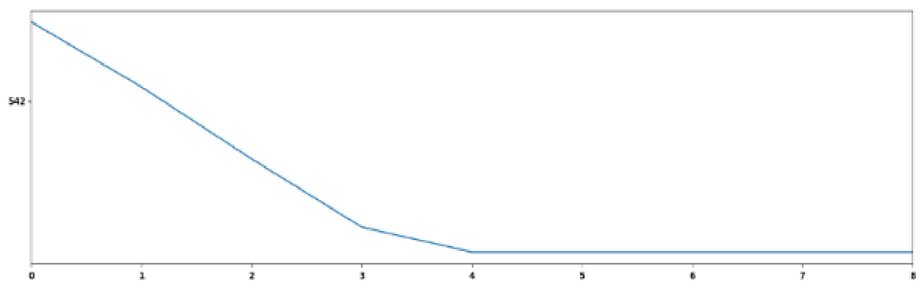
5.3.1. The Different Time, Cost, and Figure Paths

According to the experiment result from Table 11, the solution in SA with a total cost of 442.96 is better among all five evolutionary algorithms. Notably, the different fundamental principle of each algorithm shows the different path in solving the TSP.

5.3.2. Compared of Converged Figure

Knowing that the iteration time is the particularly significant parameter in evolutionary algorithms, and the converge figure plays a massive role in comparing different algorithms, the below Table 12 shows the specific converge steps:

Table 12 Comparison of converged figure

Algorithms	Figure
ACO	
SA	
GA	
Tabu	
PSO	

All the figure above shows the concrete steps of converge. As the authors see from the table, ACO can converge to the optimal solution found at a short iteration time. When the iteration times get to 2, the line is smooth. Although the iteration time still increased, the value would not change [13].

In the aspect of SA, the converge step is hard; the line may get into the local maximum for the first iteration. However, when the iteration time approaches 200, there is a significant change in the cost for TSP. Then it keeping reduce the total cost until the iteration gets to 400. Finally, the line becomes smooth after iteration time at 420.

Looking at the figure about the GA, authors can see it initially decreases the cost. Then, the decreasing rate is lower and becomes smooth when the iteration time is 400. At the time of 600, the cost reduces slightly and remains to converge after that.

The Tabu is very different from other algorithms, and the figure shows the authors that the Tabu algorithms cannot get to the converge condition. The curve is constantly changing with the iteration time changes at a reasonable range.

PSO is more like the converge step of ACO. The converge step is clear and straightforward; the cost reduces until the iteration time reaches 4. The curve then becomes smooth, and the cost becomes stable.

6. Conclusion

Generally speaking, PPO shows a stable and close approach to the problem. SA shows the fast running time of this problem, and PSO and ACO indicate the stable converge curve for solving TSP. Among the authors' research, all the algorithms have advantages and disadvantages. When facing the TSP, the choice for us to solve it is multiple. We can select the best method depending on the specific problem condition. The control of variables still has some limitations.

Furthermore, we can focus on how exactly the number of parameters changes; for the different parameters, the converge condition may be various. Moreover, we can find which algorithm is better following the advantage listed above; our case study is limited, and there will exist more cases to be explored.

References

[1] https://neorl.readthedocs.io/en/latest/examples/ex1.html

[2] https://arxiv.org/abs/1602.01783

[3] https://arxiv.org/abs/1312.5602

[4] https://arxiv.org/abs/1707.06347

[5] Kennedy, J., Eberhart, R. (1995). Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks (Vol. 4, pp. 1942-1948), IEEE.

[6] Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. European journal of operational research, 185(3), 1155-1173.

[7] Onbaşoğlu, E., Özdamar, L. (2001). Parallel simulated annealing algorithms in global optimization. Journal of global optimization, 19(1), 27-50.

[8] Glover, F. (1989). Tabu search—part I. ORSA Journal on computing, 1 (3), 190-206.

[9] Glover, F. (1990). Tabu search—part II. ORSA Journal on computing, 2 (1), 4-32.

[10] Kennedy, J., & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In: 1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation (Vol. 5, pp. 4104-4108), IEEE.

[11] Clerc, M., Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE transactions on Evolutionary Computation, 6(1), 58-73.

[12] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, New York (1985).

[13] Zhang, H., Gao, Y.F. (2021). Solving TSP based on an Improved Ant Colony Optimization Algorithm. J. Phys.: Conf. ser. 1982 012061