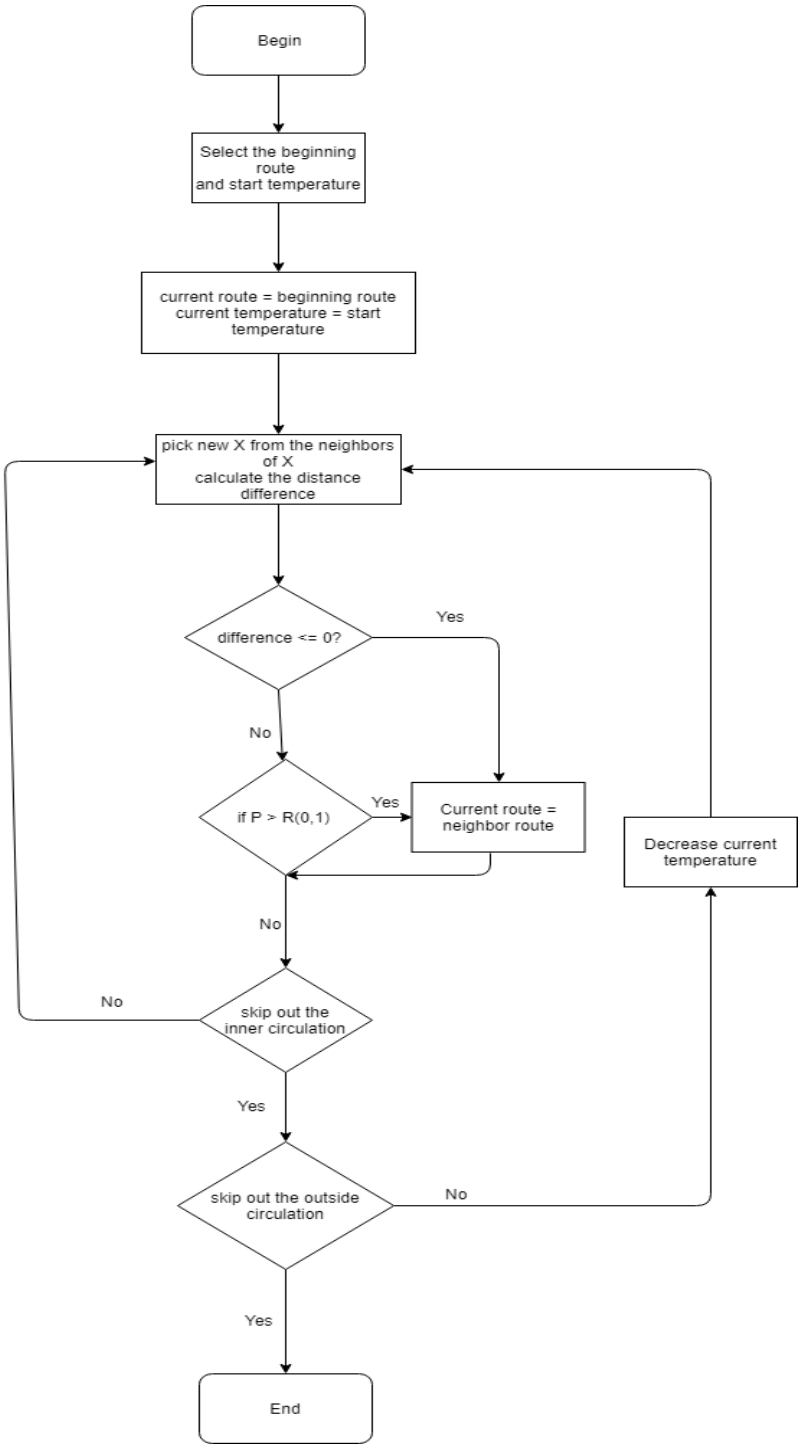
# Introduction

## Simulated Annealing

- Flowchart

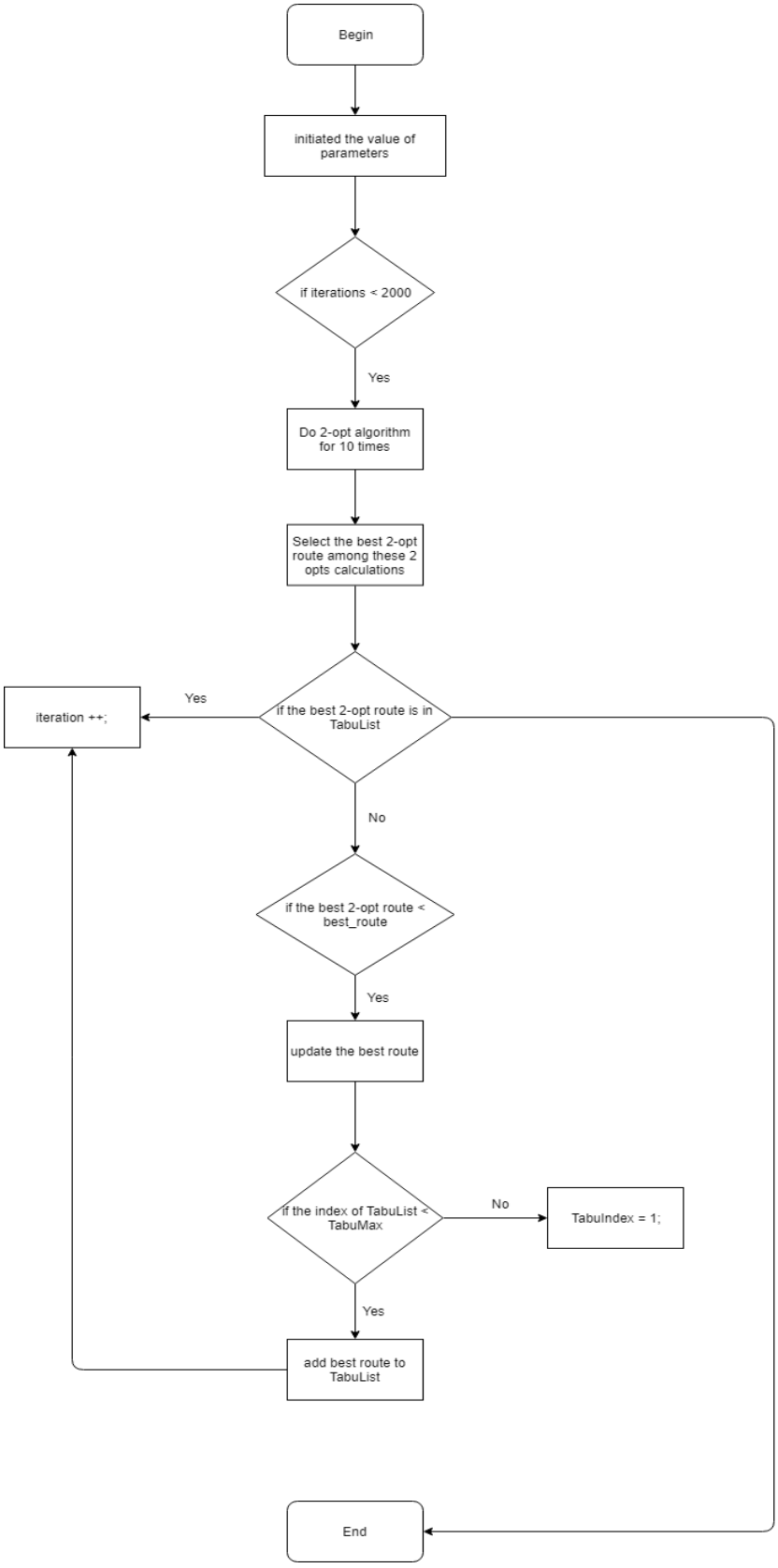


- Sudo Code

```
x := x0; e := f(x)
xbest := x; xbest := x
k := 0
while (k < kmax)
        while (ind1 == ind2)
           ind1 = ceil(rand.*num_cities);
           ind2 = ceil(rand.*num_cities);
        end
        if ind1 < ind2
           route_new (1:ind1-1) = best_route(1:ind1-1);
           route_new (ind1:ind2) = fliplr(best_route(ind1:ind2));
           route_new (ind2+1:num_cities) = best_route(ind2+1:num_cities);
        else
           route_new (1:ind2-1) = best_route(1:ind2-1);
           route_new (ind2:ind1) = fliplr(best_route(ind2:ind1));
           route_new (ind1+1:num_cities) = best_route(ind1+1:num_cities);
        end
   T := temperature(t0)
   xnew := neighbour(x)
   enew := f(xnew)
   if P(e,enew,T) > R(0,1) then
   x := xnew; e := enew
   if enew < ebest then
   xbest := xnew; ebest := enew.
   k := k + 1
Output xbest
```

## Tabu Search algorithms

- Flowchart


Begin
initiated the value of parameters
if iterations < 2000
Yes
Do 2-opt algorithm for 10 times
Select the best 2-opt route among these 2 opts calculations
if the best 2-opt route is in TabuList
Yes
iteration ++;
No
if the best 2-opt route < best_route
Yes
update the best route
if the index of TabuList < TabuMax
No
TabuIndex = 1;
Yes
add best route to TabuList
End

- Sudo Code

```
while iterations < 2000

    while twoopt_index < 10

        ind1 = 0; ind2 = 0;
        while (ind1 == ind2 || ind1 == 1 || ind2 == 1)
            ind1 = ceil(rand.*num_cities);
            ind2 = ceil(rand.*num_cities);
        end
       if ind1 < ind2
            route_new (2:ind1-1) = best_route(2:ind1-1);
            route_new (ind1:ind2) = fliplr(best_route(ind1:ind2));
            route_new (ind2+1:num_cities) = best_route(ind2+1:num_cities);
        else
            route_new (2:ind2-1) = best_route(2:ind2-1);
            route_new (ind2:ind1) = fliplr(best_route(ind2:ind1));
            route_new (ind1+1:num_cities) = best_route(ind1+1:num_cities);
        end
        twoopt_temp = cities(best_route,:);
        current_length = sumdistance(twoopt_temp);
        if current_length < first_length
            first_length = current_length
%              disp('fuzhi')
            first_route = route_new;
        end
        twoopt_index = twoopt_index + 1;
    end

    if   ismember(first_route, TabuList, 'rows')
        iterations = iterations + 1
    else
            if first_length < best_length
                best_route = first_route
                best_length = first_length
            end

            if tabu_index <= TabuNum
                TabuList(tabu_index,:) = best_route;
                tabu_index = tabu_index+1;
            else
                tabu_index = 1;                    % 重新开始录入 TabuList
            end
            temp_cities = cities(best_route,:);
            candidate_length= sumdistance(temp_cities);
```

```
        end
        iterations = iterations + 1;
end
```

# Tune Parameters

## Simulated Annealing

The main parameters in Simulated Annealing are in the process of Temperature Calculation. And in this process, the main factors are:

- Parameters of Temperature cooling Function:
  First through searching online, get the approximate range of this parameter. Then, select several values of this parameters and do the experiment. Using the same iteration times 500 and the same start/end temperature, and different parameters of Temperature cooling Function, to compare the result. Each parameter runs for 5 times and calculate the average Minimum distance which shows below.

| Parameters of Temperature cooling Function | Average Minimum distance |
|---|---|
| 0.7 | 8552 |
| 0.8 | 7539 |
| 0.9 | 7382 |
| 0.95 | 7112 |
| 0.93 | 7140 |
| 0.97 | 7112 |
| 0.98 | 7100 |
| 0.99 | 7052 |

  We could see 0.99 are the relatively better parameters. So Finally select 0.99 as the Parameters of Temperature cooling Function.

- Start Temperature and end Temperature:
  First through searching online, get the approximate range of start Temperature and end Temperature. Then, select several values of this parameters and do the experiment. With the same Parameters of Temperature cooling Function and same iteration times 500, and different start/end time. Each parameter runs for 5 times and calculate the average Minimum distance, choosing some representative values which show below.

| Parameters of Start Temperature | Average Minimum distance |
|---|---|
| 100 | 7110 |
| 90 | 7100 |
| 95 | 7083 |
| 97 | 7013 |

We could see 97 are the relatively better start temperature. So Finally select 97 as the start temperature. Using the similar method, select the end temperature 3.

## Tabu Search algorithms

The main parameters in Tabu Search algorithms are the times of 2-opt algorithms in one iteration and the length of Tabu List.
To tune the parameters, through searching online, I first got the approximate range of this parameter. Then, select several values of this parameters and do the experiment. Using the same iteration times 500 and other initialized values, and different the times of 2-opt algorithms in one iteration and the length of Tabu List., to compare the result. Each parameter runs for 5 times and calculate the average Minimum distance which shows below.

- Times of 2-opt algorithms in one iteration

| Times of 2-opt algorithms in one iteration | Average Minimum distance |
|---|---|
| 4 | 7179 |
| 10 | 7132 |
| 20 | 7214 |
| 5 | 7175 |
| 6 | 7185 |

We could see 10 are the relatively better start temperature. So Finally select 10 as the start temperature.

- the length of Tabu List
  According the calculus TabuLength=round((num_cities*(num_cities-1)/2)^0.5), we could get the round `number of Tabu List's length is 115. After using the similar method to tune the parameters and record the results, the change rate of distance decreasing trend is considered comprehensively. Finally choose the length of Tabu List 100.

# Average Result & Standard Deviations

## Simulated Annealing

| | |
|---|---|
| 1 | 7039 |
| 2 | 7013 |
| 3 | 7019 |

| | |
|---|---|
| 4 | 7013 |
| 5 | 7013 |
| 6 | 7092 |
| 7 | 7013 |
| 8 | 7013 |
| 9 | 7097 |
| 10 | 7013 |
| 11 | 7013 |
| 12 | 7013 |
| 13 | 7087 |
| 14 | 7013 |
| 15 | 7091 |
| 16 | 7087 |
| 17 | 7019 |
| 18 | 7092 |
| 19 | 7013 |
| 20 | 7013 |
| 21 | 7013 |
| 22 | 7013 |
| 23 | 7100 |
| 24 | 7066 |
| 25 | 7029 |
| 26 | 7013 |
| 27 | 7083 |
| 28 | 7019 |
| 29 | 7087 |
| 30 | 7013 |
| 31 | 7039 |

Average Result: 7040
Standard Deviations: 35.4235

## Tabu Search algorithms

| | |
|---|---|
| 1 | 7105 |
| 2 | 7015 |
| 3 | 7013 |
| 4 | 7195 |
| 5 | 7114 |
| 6 | 7239 |
| 7 | 7092 |
| 8 | 7218 |

| | |
|---|---|
| 9 | 7100 |
| 10 | 7113 |
| 11 | 7174 |
| 12 | 7114 |
| 13 | 7210 |
| 14 | 7013 |
| 15 | 7116 |
| 16 | 7190 |
| 17 | 7092 |
| 18 | 7087 |
| 19 | 7195 |
| 20 | 7013 |
| 21 | 7195 |
| 22 | 7092 |
| 23 | 7287 |
| 24 | 7159 |
| 25 | 7143 |
| 26 | 7013 |
| 27 | 7227 |
| 28 | 7116 |
| 29 | 7105 |
| 30 | 7219 |
| 31 | 7143 |

Average Result: 7132

Standard Deviations: 74.0963

# Compare the Results

Using the statistic method: Wilcoxon signed-rank test to analyze the 31 results generated by Simulated Annealing and Tabu Search algorithms, which shows below:

Wilcoxon signed-rank test

| Name | Sample Size | Median | Statistical Magnitude | P |
|---|---|---|---|---|
| Simulated Annealing | 31 | 7019 | 4.928 | 0.000** |
| Tabu Search algorithmss | 31 | 7116 | 4.863 | 0.000** |

* p<0.05 ** p<0.01