In the simulated annealing algorithm section, I also need to address the followings,

* The starting design that I generated here versus any random starting design.
* The accelerated cooling method versus standard cooling method.
* Pair swapping method versus one-to-one swapping method.
* Two-stages swapping method versus standard swapping method.

These four issues will be addressed using one experiment with on specific set of design parameters, which are

Phase 1 experiment - 6 treatments, 3 biological replicates, 2 technical replicates,

Phase 2 experiment – 9 runs and 4 tags.

For the Phase 1 experiment, the 6 treatments are denoted by “a”, “b”, “c”, “d”, “e” and “f”. Since 3 biological replicates are used, this means 3 animals are assigned to each treatment which gives a total of 15 animals. These 15 animals are denoted by upper case letters of “A” to “R”. The theoretical ANOVA of the Phase 1 experiment can be presented as follows,

$ANOVA

DF Ani

Between Ani

Trt 5 1

Residual 12 1

$EF

Trt eff.Trt

Between Ani

Trt 3 1

Note that all treatment information is in the between animals stratum.

1. The starting design that I generated here versus any random starting design.

The pattern (which I have been using) for assigning the animals to the runs and tags is to group pair of animals and allocating them in a quadrant of 2 runs and 2 tags. For this case, the total number of runs needed is 9; hence, the last pair of animals is assigned to the last run. The pair of animas can be Animals “A” and “B”, Animals “C” and “D” to Animals “Q” and “R”. The allocation of the animals to runs and tags can be shown as follows,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run | Tag | | | |
| 114 | 115 | 116 | 117 |
| 1 | A | B | C | D |
| 2 | B | A | D | C |
| 3 | E | F | G | H |
| 4 | F | E | H | G |
| 5 | I | J | K | L |
| 6 | J | I | L | K |
| 7 | M | N | O | P |
| 8 | N | M | P | O |
| 9 | Q | Q | R | R |

The bold box in this design represents each pair of the animals. Note that the animal is confounded with both runs and tags. More specifically, the animal is confounded with a tag contrast of 114, 115 versus 116, 117. For the relationship between runs and animals, the runs can be separated into 5 groups according to the pairs of animals that are assigned. This means 4 DF associated with the animals are confounded with the runs, or we can also say that 4Df associated with the animals should be in the between runs stratum.

The treatment allocation to runs and tags is based on the assignments of treatments to animals of the Phase 1 experiments. The treatment design is shown as follows,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run | Tag | | | |
| 114 | 115 | 116 | 117 |
| 1 | a | b | c | d |
| 2 | b | a | d | c |
| 3 | e | f | a | b |
| 4 | f | e | b | a |
| 5 | c | d | e | f |
| 6 | d | c | f | e |
| 7 | a | b | c | d |
| 8 | b | a | d | c |
| 9 | e | e | f | f |

The bold box in this design represents each pair of treatments from the pair of the animals. The treatment is also confounded with both runs and tags. The treatment is also confounded with the confounded with a tag contrast of 114, 115 versus 116, 117.

> res <- optim(newInit, obj.fun.new, swap, method = "SANN", control = list(maxit = 1e+6,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 100, fnscale = -1))

sann objective function values

initial value -97.617991

iter 100000 value -97.617991

iter 200000 value -97.617991

iter 300000 value -97.617991

iter 400000 value -97.617991

iter 500000 value -97.617991

iter 600000 value -97.617991

iter 700000 value -97.617991

iter 800000 value -97.617991

iter 900000 value -97.617991

iter 999999 value -97.617991

final value -97.617991

sann stopped after 999999 iterations

> set.seed(527)

> init = sample(1:n)

>

> newInit = c(init, init[1])

>

> res <- optim(newInit, obj.fun.new, swap, method = "SANN", control = list(maxit = 1e+6,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 100, fnscale = -1))

sann objective function values

initial value -51.555453

iter 100000 value -72.088654

iter 200000 value -76.271591

iter 300000 value -76.271591

iter 400000 value -76.271591

iter 500000 value -76.271591

iter 600000 value -76.271591

iter 700000 value -76.271591

iter 800000 value -76.271591

iter 900000 value -76.271591

iter 999999 value -76.271591

final value -76.271591

sann stopped after 999999 iterations

With the initial temperature of 100, simulated annealing algorithm was performed using the opitm function for one million iterations. This takes just over 9 minutes.

The simulated annealing algorithm applied on the constructed starting design does not improve the result since the results from the objective function was already very high with 97.62. This value is however still higher than the using a random starting design and applied simulated annealing method with 1000000 iterations, which gives 76.28.

1. The accelerated cooling method described by John and Whitaker (1993) versus standard cooling method.

The standard simulated annealing is believe to be converge slowly and the solution maybe far from optimal. This is due to the slow cooling schedules.

Here a modified version of simulated annealing was used, namely “nested simulated annealing”.

Start with a high temperature of 100, then the temperature is divided by 2 at each level; hence, the initial temperature is drop from 100 to 50, then 25, 12.5 and so on for each standard simulated annealing algorithm.

The divisor of 4, as mentioned in the paper, has been used, but the performance in the value from the objective function is worse than using the divisor of 2.

This approach to simulated annealing produce good solution via the accelerated cooling achieved by moving down levels, but it also carries out standard simulated annealing at each temperature level.

Nested annealing starts at the same high temperature, but it more quickly down the temperature, where at least good local optimum solutions are found.

Then it returns to normal nested algorithm. Further excursion into accelerate cooling giving frequent improve solution with more opportunities to find a global optimum.

This Nesting means the switching of standard cooling and accelerated cooling. Random walk across a surface which initially is high temperature with any depress that are found being fully investigated via accelerated cooling as well as depressions within depressions. Gradually the random walks become more confined following the contours of the surface, with fewer excursions into depressions.

The improve can be show in an example as follows,

> y2 = 100

> res <- optim(newInit, obj.fun.new, swap, method = "SANN", control = list(maxit = 1e+5,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 100, fnscale = -1))

sann objective function values

initial value -97.617991

iter 99999 value -97.617991

final value -97.617991

sann stopped after 99999 iterations

>

> for (i in 1:5) {

+ y2 = y2/2

+ print(y2)

+ res <- optim(res$par, obj.fun.new, swap, method = "SANN", control = list(maxit = 1e+5,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 100, fnscale = -1))

+

+ }

[1] 50

sann objective function values

initial value -97.617991

iter 99999 value -97.617991

final value -97.617991

sann stopped after 99999 iterations

[1] 25

sann objective function values

initial value -97.617991

iter 99999 value -97.617991

final value -97.617991

sann stopped after 99999 iterations

[1] 12.5

sann objective function values

initial value -97.617991

iter 99999 value -97.617991

final value -97.617991

sann stopped after 99999 iterations

[1] 6.25

sann objective function values

initial value -97.617991

iter 99999 value -97.617991

final value -97.617991

sann stopped after 99999 iterations

[1] 3.125

sann objective function values

initial value -97.617991

iter 99999 value -97.617991

final value -97.617991

sann stopped after 99999 iterations

> y2 = 100

> set.seed(527)

> init = 1:n #sample(1:n)

> init = sample(1:n)

>

> newInit = c(init, init[1])

>

> y2 = 100

> res <- optim(newInit, obj.fun.new, swap, method = "SANN", control = list(maxit = 1e+5,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 100, fnscale = -1))

sann objective function values

initial value -51.555453

iter 99999 value -72.088654

final value -72.088654

sann stopped after 99999 iterations

>

> for (i in 1:9) {

+ y2 = y2/2

+ print(y2)

+ res <- optim(res$par, obj.fun.new, swap, method = "SANN", control = list(maxit = 1e+5,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 100, fnscale = -1))

+

+ }

[1] 50

sann objective function values

initial value -72.088654

iter 99999 value -72.953904

final value -72.953904

sann stopped after 99999 iterations

[1] 25

sann objective function values

initial value -72.953904

iter 99999 value -74.463678

final value -74.463678

sann stopped after 99999 iterations

[1] 12.5

sann objective function values

initial value -74.463678

iter 99999 value -76.540454

final value -76.540454

sann stopped after 99999 iterations

[1] 6.25

sann objective function values

initial value -76.540454

iter 99999 value -76.540454

final value -76.540454

sann stopped after 99999 iterations

[1] 3.125

sann objective function values

initial value -76.540454

iter 99999 value -76.540454

final value -76.540454

sann stopped after 99999 iterations

[1] 1.5625

sann objective function values

initial value -76.540454

iter 99999 value -78.647182

final value -78.647182

sann stopped after 99999 iterations

[1] 0.78125

sann objective function values

initial value -78.647182

iter 99999 value -78.647182

final value -78.647182

sann stopped after 99999 iterations

[1] 0.390625

sann objective function values

initial value -78.647182

iter 99999 value -78.647182

final value -78.647182

sann stopped after 99999 iterations

[1] 0.1953125

sann objective function values

initial value -78.647182

iter 99999 value -78.647182

final value -78.647182

sann stopped after 99999 iterations

The accelerated cooling method made no difference on the modified starting design. However, it does improve the result from the objective function with a random starting design.

1. Pair swapping method versus one-to-one swapping method.

> y2 = 100

> res <- optim(newInit, obj.fun.new, swap.new, method = "SANN", control = list(maxit = 1e+5,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

sann objective function values

initial value -97.617991

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 50000 value -98.189248

iter 60000 value -98.189248

iter 70000 value -98.189248

iter 80000 value -98.189248

iter 90000 value -98.189248

iter 99999 value -98.189248

final value -98.189248

sann stopped after 99999 iterations

>

> for (i in 1:5) {

+ y2 = y2/2

+ print(y2)

+ res <- optim(res$par, obj.fun.new, swap.new, method = "SANN", control = list(maxit = 1e+5,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

+

+ }

[1] 50

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 50000 value -98.189248

iter 60000 value -98.189248

iter 70000 value -98.189248

iter 80000 value -98.189248

iter 90000 value -98.189248

iter 99999 value -98.189248

final value -98.189248

sann stopped after 99999 iterations

[1] 25

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 50000 value -98.189248

iter 60000 value -98.189248

iter 70000 value -98.189248

iter 80000 value -98.189248

iter 90000 value -98.189248

iter 99999 value -98.189248

final value -98.189248

sann stopped after 99999 iterations

[1] 12.5

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 50000 value -98.189248

iter 60000 value -98.189248

iter 70000 value -98.189248

iter 80000 value -98.189248

iter 90000 value -98.189248

iter 99999 value -98.189248

final value -98.189248

sann stopped after 99999 iterations

[1] 6.25

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 50000 value -98.189248

iter 60000 value -98.189248

iter 70000 value -98.189248

iter 80000 value -98.189248

iter 90000 value -98.189248

iter 99999 value -98.189248

final value -98.189248

sann stopped after 99999 iterations

[1] 3.125

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 50000 value -98.189248

iter 60000 value -98.189248

iter 70000 value -98.189248

iter 80000 value -98.189248

iter 90000 value -98.189248

iter 99999 value -98.189248

final value -98.189248

sann stopped after 99999 iterations

Pair swapping is only applicable with the modified starting designs, because the swapping are taking place over the samples of same technical replicates. It shows, with the pair swapping method, that the optimal design has found just after 10000 iterations.

1. Two-stages swapping method versus standard swapping method.

Two-stage swapping is only applicable for row-column designs.

For this experiment, the runs and tags are considered as the rows and columns.

In the first stage of swapping, the swapping only take place within runs, that means when the swapping of two observations, it has to be in the same runs.

The second stage of swapping is swapping within tags that means when the swapping of two observations, it has to be in the same tags.

This method reduces the search space of the simulated annealing algorithm; hence, it has ability to find a better result from the objective function quickly.

> y2 = 100

> res <- optim(newInit, obj.fun.new, swap.stage1.new, method = "SANN", control = list(maxit = 5e+4,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

sann objective function values

initial value -97.617991

iter 10000 value -97.672376

iter 20000 value -97.672376

iter 30000 value -97.672376

iter 40000 value -97.672376

iter 49999 value -97.672376

final value -97.672376

sann stopped after 49999 iterations

>

> y2 = 100

> res1 <- optim(res$par, obj.fun.new, swap.stage2.new, method = "SANN", control = list(maxit = 5e+4,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

sann objective function values

initial value -97.672376

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

>

> for (i in 1:5) {

+ y2 = y2/2

+ print(y2)

+ res1 <- optim(res1$par, obj.fun.new, swap.stage1.new, method = "SANN", control = list(maxit = 5e+4,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

+

+ res1 <- optim(res1$par, obj.fun.new, swap.stage2.new, method = "SANN", control = list(maxit = 5e+4,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

+ }

[1] 50

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

[1] 25

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

[1] 12.5

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

[1] 6.25

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

[1] 3.125

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

sann objective function values

initial value -98.189248

iter 10000 value -98.189248

iter 20000 value -98.189248

iter 30000 value -98.189248

iter 40000 value -98.189248

iter 49999 value -98.189248

final value -98.189248

sann stopped after 49999 iterations

For this set of design parameters, the pair swapping with two-stages swapping method on the modified starting design shown to be slower than the using pair swapping method alone, because it requires two-stages. However, the optimal designs can still be generated within 60000 iterations.

> set.seed(527)

> init = 1:n #sample(1:n)

> init = sample(1:n)

>

> newInit = c(init, init[1])

>

> y2 = 100

> res <- optim(newInit, obj.fun.new, swap.stage1, method = "SANN", control = list(maxit = 5e+4,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

sann objective function values

initial value -51.555453

iter 10000 value -61.965492

iter 20000 value -61.965492

iter 30000 value -61.965492

iter 40000 value -61.965492

iter 49999 value -61.965492

final value -61.965492

sann stopped after 49999 iterations

>

> y2 = 100

> res1 <- optim(res$par, obj.fun.new, swap.stage2, method = "SANN", control = list(maxit = 5e+4,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

sann objective function values

initial value -61.965492

iter 10000 value -72.821139

iter 20000 value -75.045529

iter 30000 value -75.045529

iter 40000 value -75.272277

iter 49999 value -75.272277

final value -75.272277

sann stopped after 49999 iterations

>

> for (i in 1:9) {

+ y2 = y2/2

+ print(y2)

+ res1 <- optim(res1$par, obj.fun.new, swap.stage1, method = "SANN", control = list(maxit = 5e+4,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

+

+ res1 <- optim(res1$par, obj.fun.new, swap.stage2, method = "SANN", control = list(maxit = 5e+4,

+ temp = y2, tmax = 1000, trace = TRUE, REPORT = 10, fnscale = -1))

+ }

[1] 50

sann objective function values

initial value -75.272277

iter 10000 value -75.554581

iter 20000 value -75.554581

iter 30000 value -75.554581

iter 40000 value -75.554581

iter 49999 value -75.554581

final value -75.554581

sann stopped after 49999 iterations

sann objective function values

initial value -75.554581

iter 10000 value -75.554581

iter 20000 value -75.554581

iter 30000 value -76.310063

iter 40000 value -76.310063

iter 49999 value -76.310063

final value -76.310063

sann stopped after 49999 iterations

[1] 25

sann objective function values

initial value -76.310063

iter 10000 value -77.911739

iter 20000 value -78.220483

iter 30000 value -78.220483

iter 40000 value -78.220483

iter 49999 value -78.220483

final value -78.220483

sann stopped after 49999 iterations

sann objective function values

initial value -78.220483

iter 10000 value -78.220483

iter 20000 value -78.220483

iter 30000 value -78.220483

iter 40000 value -79.050471

iter 49999 value -82.481516

final value -82.481516

sann stopped after 49999 iterations

[1] 12.5

sann objective function values

initial value -82.481516

iter 10000 value -83.076766

iter 20000 value -83.135141

iter 30000 value -83.135141

iter 40000 value -83.135141

iter 49999 value -83.135141

final value -83.135141

sann stopped after 49999 iterations

sann objective function values

initial value -83.135141

iter 10000 value -83.135141

iter 20000 value -83.135141

iter 30000 value -83.135141

iter 40000 value -83.135141

iter 49999 value -83.135141

final value -83.135141

sann stopped after 49999 iterations

[1] 6.25

sann objective function values

initial value -83.135141

iter 10000 value -83.135141

iter 20000 value -83.210350

iter 30000 value -83.210350

iter 40000 value -83.210350

iter 49999 value -83.210350

final value -83.210350

sann stopped after 49999 iterations

sann objective function values

initial value -83.210350

iter 10000 value -83.210350

iter 20000 value -83.210350

iter 30000 value -83.210350

iter 40000 value -83.210350

iter 49999 value -83.210350

final value -83.210350

sann stopped after 49999 iterations

[1] 3.125

sann objective function values

initial value -83.210350

iter 10000 value -83.210350

iter 20000 value -83.210350

iter 30000 value -83.210350

iter 40000 value -83.216117

iter 49999 value -83.216117

final value -83.216117

sann stopped after 49999 iterations

sann objective function values

initial value -83.216117

iter 10000 value -83.216117

iter 20000 value -83.216117

iter 30000 value -83.216117

iter 40000 value -83.216117

iter 49999 value -83.216117

final value -83.216117

sann stopped after 49999 iterations

[1] 1.5625

sann objective function values

initial value -83.216117

iter 10000 value -83.216117

iter 20000 value -83.216117

iter 30000 value -83.216117

iter 40000 value -83.216117

iter 49999 value -83.222822

final value -83.222822

sann stopped after 49999 iterations

sann objective function values

initial value -83.222822

iter 10000 value -84.439363

iter 20000 value -84.439363

iter 30000 value -84.439363

iter 40000 value -84.439363

iter 49999 value -84.439363

final value -84.439363

sann stopped after 49999 iterations

[1] 0.78125

sann objective function values

initial value -84.439363

iter 10000 value -84.969996

iter 20000 value -84.969996

iter 30000 value -84.969996

iter 40000 value -84.969996

iter 49999 value -84.969996

final value -84.969996

sann stopped after 49999 iterations

sann objective function values

initial value -84.969996

iter 10000 value -84.969996

iter 20000 value -84.969996

iter 30000 value -84.969996

iter 40000 value -84.969996

iter 49999 value -84.969996

final value -84.969996

sann stopped after 49999 iterations

[1] 0.390625

sann objective function values

initial value -84.969996

iter 10000 value -84.969996

iter 20000 value -84.969996

iter 30000 value -84.969996

iter 40000 value -84.969996

iter 49999 value -84.969996

final value -84.969996

sann stopped after 49999 iterations

sann objective function values

initial value -84.969996

iter 10000 value -84.969996

iter 20000 value -84.969996

iter 30000 value -84.969996

iter 40000 value -84.969996

iter 49999 value -84.969996

final value -84.969996

sann stopped after 49999 iterations

[1] 0.1953125

sann objective function values

initial value -84.969996

iter 10000 value -84.970056

iter 20000 value -84.970056

iter 30000 value -84.970056

iter 40000 value -84.970056

iter 49999 value -84.970056

final value -84.970056

sann stopped after 49999 iterations

sann objective function values

initial value -84.970056

iter 10000 value -84.970056

iter 20000 value -84.970056

iter 30000 value -84.970056

iter 40000 value -84.970056

iter 49999 value -84.970056

final value -84.970056

sann stopped after 49999 iterations

The comparison between the two-stage swapping method with the standard swapping method on the random starting design. It shows that two-stage swapping method can generate an more optimal design than the standard swapping method based on the results from the objective function.

Summary of table for comparing different method in simulated annealing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Methods | Modified Starting design | Accelerated cooling method | Pair swapping | Two-stage swapping | Objective function | | Iteration |
| Before | After |
| SA |  |  |  |  | 51.555453 | 76.271591 | 1e6 |
| SA on modified starting design | ˅ |  |  |  | 97.617991 | 97.617991 | 1e6 |
| Accelerated cooling method |  | ˅ |  |  | 51.555453 | 78.647182 | 1e6 |
| Accelerated cooling method on modified design | ˅ | ˅ |  |  | 97.617991 | 97.617991 | 1e6  ((10)1e5) |
| Pair swapping method with accelerated cooling method on modified starting design | ˅ | - | ˅ |  | 97.617991 | 98.189248 | 1e4 |
| Two-stages swapping method with Pair swapping method with accelerated cooling method on modified starting design | ˅ | - | ˅ | ˅ | 97.617991 | 98.189248 | 6e4 |
| Two-stages swapping method |  |  |  | ˅ | 51.555453 | 71.125898 | 1e6 (5e5) |
| Two-stages swapping method with accelerated cooling method |  | ˅ |  | ˅ | 51.555453 | 84.970056 | 1e6  ((10)1e5  (5e4)) |

In conclusion, based on these results, I believe that using the modified starting design with accelerate cooling, pair swapping and two-stage swapping will allow user to find the optimal design more quickly and efficiently.