

模拟退火和遗传算法求解TSP问题

模拟退火和遗传算法求解TSP问题

摘要

1 导言

1.1 问题重述

1.2 TSP问题选择

1.3 思路设计

1.4 结果简览

2 实验过程

2.1 TSPbase

2.2 LocalSearch

 2.2.1 流程图

 2.2.2 满意度、活跃度机制

 2.2.3 邻域操作

 2.2.4 代码分析

2.3 SimulatedAnnealing

 2.3.1 流程图

 2.3.2 代码分析

2.4 GeneticAlgorithm

 2.4.1 流程图

 2.4.2 交叉与变异算子

 2.4.3 评分机制及精英保留策略

 2.4.4 代码分析

3 结果分析

3.1 实验环境

 3.1.1 系统信息

 3.1.2 开发工具

3.2 编译运行

 3.2.1 初始状态

 3.2.2 编译

 3.2.3 运行及自定参数

 3.2.4 结果可视化

3.3 搜索结果及比较

 3.3.1 实验数据对比

 3.3.2 实验数据可视化

 3.3.3 路线对比

3.4 性能分析

3.5 探索与展望

4 结论

5 主要参考文献

摘要

本实验使用局部搜索、模拟退火、遗传算法（C/C++实现），对TSP问题进行搜索求解。

在实现算法、适当调参后，在选用的5个TSP问题中皆搜索得到误差小于10%的路线。

1 导言

1.1 问题重述

选一个大于100个城市数的TSP问题：

1. 局部搜索与模拟退火：

1. 采用多种邻域操作的局部搜索local search策略求解；
2. 在局部搜索策略的基础上，加入模拟退火simulated annealing策略，并比较两者的效果；
3. 要求求得的解不要超过最值的10%，并能够提供可视化图形界面，观察路径的变化和交叉程度。

2. 用遗传算法求解TSP问题（问题规模等和模拟退火求解TSP实验同），要求：

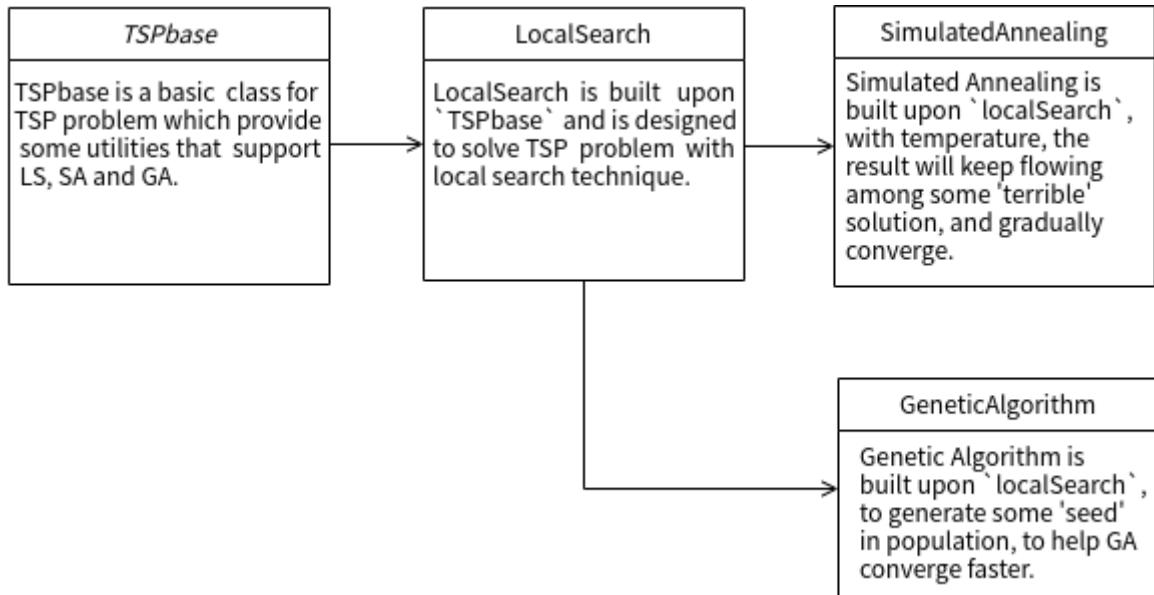
1. 设计较好的交叉操作，并且引入多种局部搜索操作（可替换通常遗传算法的变异操作）
2. 和之前的模拟退火算法（采用相同的局部搜索操作）进行比较
3. 得出设计高效遗传算法的一些经验，并比较单点搜索和多点搜索的优缺点。

1.2 TSP问题选择

本实验从TSPLIB中选择了5个不同的TSP问题，分别为：

TSP Name	Comment	DIMENSION	Optimal Solution
kroC100	100-city problem C (Krolak/Felts/Nelson)	100	20749
ch150	150 city Problem (churritz)	150	6528
tsp225	A TSP problem (Reinelt)	225	3919
a280	drilling problem (Ludwig)	280	2579
pcb442	Drilling problem (Groetschel/Juenger/Reinelt)	442	50778

1.3 思路设计



主要思想即上图所示，详细算法内容将在之后的部分说明。

1.4 结果简览

以下实验结果以5次实验取平均值，详细结果分析将在之后的部分说明。

Summary

TSP Name	LocalSearch	Simulated Annealing	Genetic Algorithm
kroC100	22792.88 (Loss 9.85%)	20851.4 (Loss 0.49%)	21903.34(Loss 5.56%)
ch150	7260.656 (Loss 11.22%)	6560.936 (Loss 0.50%)	7124.162 (Loss 9.13%)
tsp225	4308.558 (Loss 9.94%)	3968.232 (Loss 1.26%)	4236.946 (Loss 8.11%)
a280	2906.766 (Loss 12.7%)	2612.91 (Loss 1.31%)	2922.06 (Loss 13.3%)
pcb442	57557.38(Loss 13.35%)	51877.96 (Loss 2.17%)	57649.48 (Loss 13.5%)

在本次实验实现的三种算法中，可以看到模拟退火算法的表现极佳，遗传算法稍逊，局部搜索算法较次。

实验中的数据比较也进行了可视化处理，将在之后的实验结果中详细分析。

2 实验过程

2.1 TSPbase

TSPbase 是一个用于预处理TSP问题数据、为高级搜索方法提供API的基类。

这里给出头部文件的实例，详细实现可参见 `src/TSPbase.cpp`。

```
/*****************************************************************************  
 * FileName : TSPbase.h  
 * Author : Karl  
 * Created Date : June 5, 2020  
 * Updated : June 8, 2020 - Add Function  
 *           June 9, 2020 - Fix Bug  
 *           June 9, 2020 - Modify Perfomance  
 *           June 22, 2020 - Last Check  
 *=====*  
 * @Functions:  
 *   TSPbase::TSPbase(MAP_TYPE& origin) - Constructor.  
 *   TSPbase::distance(COORD city1, COORD city2)  
 *       - calculate EUC2D distance between city1 and city2.  
 *   TSPbase::currentCost() - calculate cost of `private` path.  
 *   TSPbase::currentCost(vector<int> path)  
 *       - calculate cost of path.  
 *   TSPbase::roulette() - generate a random decimal in (0, 1).  
 *   TSPbase::generateRandomPath() - generate a random path.  
 *   TSPbase::getOptCost() - return the best(least) cost so far.  
 *   TSPbase::updateOptCost(double new_dist)  
 *       - update current best cost with new_dist.  
 *   TSPbase::backUp() - back up current path.  
 *   TSPbase::recover() - recover current path with backup.  
 *=====*/  
class TSPbase {  
public:  
    ...  
protected:  
    int N;                                // Dimension  
    MAP_TYPE citys;                         // Citys map  
    double* dist_map;                       // Dists map  
    double opt_cost;                        // Optimal cost(least distance)  
    int* path;                             // Current path  
    int* city2path;                        // Map a city to its position in path  
    int* path_bak;                          // Path backup  
    int* city2path_bak;                     // Map backup  
};
```

首先，对于一个TSP问题，分析需要考虑的内容：

1. 城市总数，即问题的维度： `N`
2. 每个城市编号以及对应坐标： `citys`
3. 路径： `path`

在读入问题文件后，分析需要记录、预计算的内容：

1. 城市间距离： `dist_map`

在搜索过程中，会非常频繁地计算整条路径的长度（欧氏距离），如果每次都要进行运算，将大大影响我们程序的效率，因此，预计算城市之间距离，静态存到数组中，在计算路径长度时以查表代替计算，能有效提高效率。

2. 最优解： `opt_cost`

3. 城市编号到路径的映射： `city2path`

在搜索过程中，可能涉及城市权重等需要按城市顺序存储的数据，因此，在选中一个城市后，想在O(1)时间内定位到它在路径中的位置，就需要建立一个从城市编号到路径位置的对应关系。

对每一次搜索进行备份：

类似于常见的搜索算法，当当前搜索结果并不如意，我们需要回溯到上一状态，这就要求TSPbase能够提供备份、恢复的功能。

1. 路径的备份： `path_bak`

2. 城市编号到路径的映射的备份： `city2path_bak`

注：

TSPbase提供的方法API在文件头部注释中有详细介绍，且命名是具有可读性的，在此不展开篇幅进行介绍。

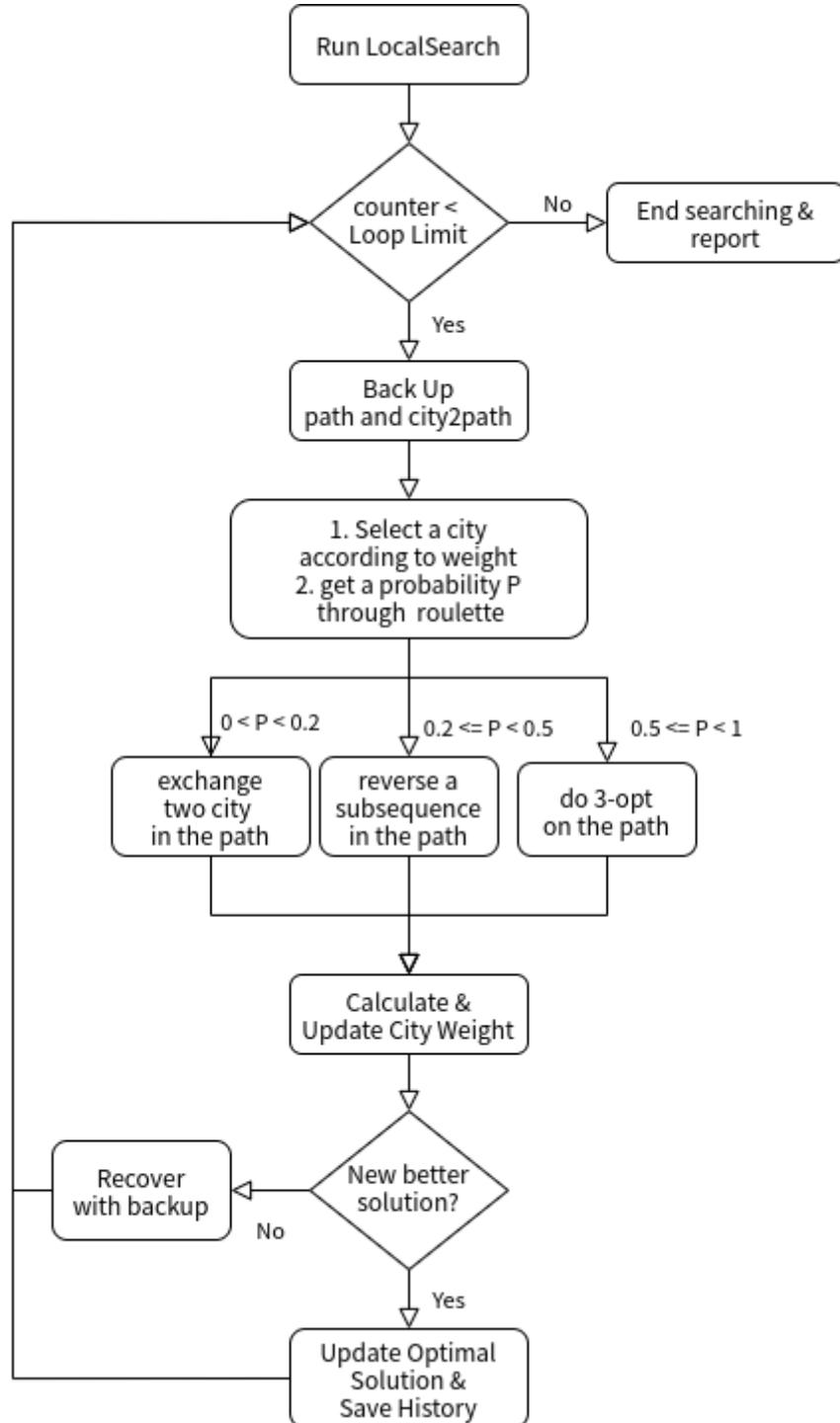
2.2 LocalSearch

LocalSearch 是基于TSPbase实现的局部搜索算法，为优化效果，加入了以下特性：

1. 加入了满意度、活跃度机制
2. 提前终止

以下对局部搜索算法进行详细介绍：

2.2.1 流程图



2.2.2 满意度、活跃度机制

该机制参考于论文《求解TSP问题的自适应领域搜索法及其扩展》

满意度

直观地，对每个城市而言，最佳位置是该城市和与之相距最近的两个城市相连。

在选择城市的时候，我们自然希望所有城市尽可能在自己的最佳位置。为了判断一个城市所在位置是否足够令人满意，加入了满意度 (f_i , 第*i*个城市的满意度) 的定义：

对城市*i*以及与之相连的城市*j*、*k*，以及距离城市*i*最近的两个城市*s*和城市*t*：

$$f_i = \exp\left[-\frac{(d_{ij} + d_{ik} - d_{is} - d_{it})^2}{2\delta^2}\right]$$

f_i : 第*i*个城市的满意度

d_{XY} : 城市 X 与城市 Y 之间的距离

δ : 参数

由此可见，满意度是一个在区间 $(0, 1]$ 内变化的数，当满意度为1时，说明该城市已达到最佳状态。

活跃度

为了避免某些满意度低的城市经过多次操作仍然难以提高满意度，从而导致算法始终只操作某些低满意度城市、忽略满意度相对高的城市，陷入局部最优解的情况，引入活跃度进行限制：

对城市*i*:

$$V_i(t+1) = \eta V_i(t) + \Delta V$$

$$h_i(t+1) = 1 - \exp\left[-\frac{(V_i(t+1))^2}{2\sigma^2}\right]$$

$V_i(t)$: t 时刻的信息素

η : 松弛系数，取值 $(0, 1)$ ，未进行操作的城市将缓慢降低活跃度

$h_i(t)$: t 时刻的活跃度

ΔV : 若城市*i*进行了反转操作，则 $\Delta V = \frac{N\sigma(1-\eta)}{K}$ ， K 为反转操作的城市数量

否则， $\Delta V = 0$

城市权重

当某些低满意度的城市被频繁操作后，它们的满意度会大幅上升，此时按照权重函数：

$$P_i(t) = [1 - f_i(t)][1 - h_i(t)]$$

能避免对同一个城市过于频繁操作或根本不对某些城市进行操作的情况。

2.2.3 邻域操作

邻域定义

对每个城市，其邻域定义为：

$$d_{ij} \leq d_{it} + \Delta d$$

$$\Delta d = \frac{3}{N} \sum_{i=1}^N d_i$$

d_i : 城市*i*与相距最近的城市距离

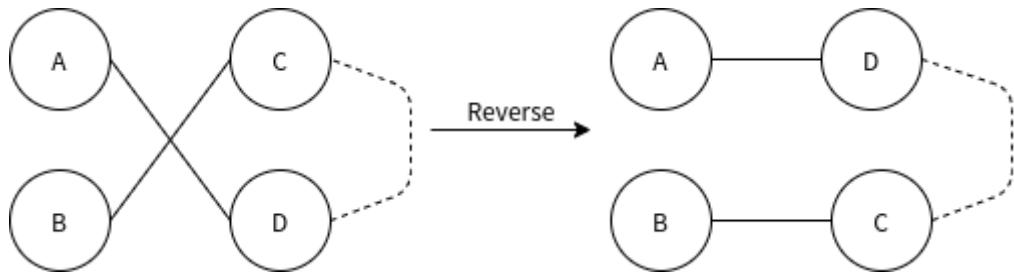
对城市*i*，找到距离它次近的城市*t*，对于其他城市*j*，若满足上述条件，则判定城市*j*在其邻域中。

操作 - 交换两个城市

朴素操作，当变换能够优化路径，则保留。

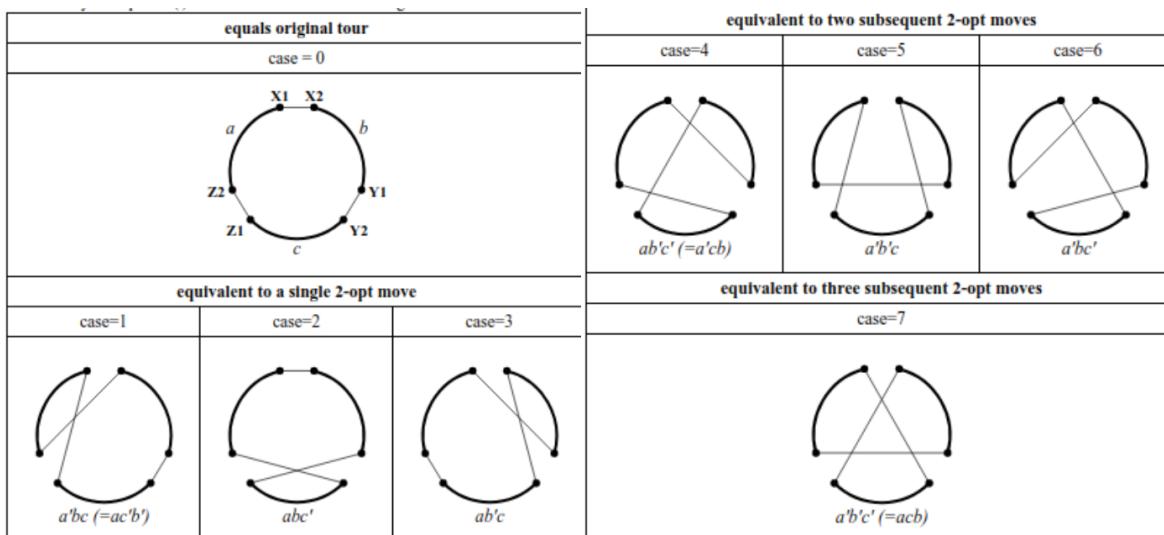
因此容易陷入局部最优解。

操作 - 逆转一段序列



通过逆转序列操作，能够优化上述情况。

操作 - 3-opt



3-opt操作能产生以上7种新情况，从中选择效果最好的一种。

2.2.4 代码分析

这里给出头部文件的实例，详细实现可参见 `src/LocalSearch.cpp`。

```
/*****************************************************************************  
 * FileName : LocalSearch.h  
 * Author : Karl  
 * Created Date : June 5, 2020  
 * Updated : June 8, 2020 - Add Function  
 *           : June 9, 2020 - Fix Bug  
 *           : June 9, 2020 - Modify Perfomance  
 *           : June 12, 2020 - Add satification  
 *           : and activity  
 *=====*  
 * @Functions:  
 * LocalSearch::LocalSearch(MAP_TYPE& origin, int LOOP_LIMITS,*  
 *                         int LOOSE_COEF, int EARLY_STOP, *  
 *                         int VERBOSE) - Constructors.  
 * LocalSearch::chooseNode() *  
 *     - choose a Node according to its weight.  
 * LocalSearch::chooseNode(int base, int range) *  
 *     - choose a Node within (base - range, base + range)  
 * LocalSearch::propagateWeightChange(int node_num, int value)*  
 *     - propagate weight change of node`node_num`. *  
 * LocalSearch::search() *  
 *     - do neighbour operation to search for a new solution. *  
 * LocalSearch::checkPath() - check `private` path's validity.*  
 * LocalSearch::checkPath(vector<int> path) *  
 *     - check path's validity.  
 * LocalSearch::earlyStop() - check whether to early stop.  
 * LocalSearch::run() - start LocalSearch process.  
 * LocalSearch::report() - save result to files.  
 * LocalSearch::exchangeTwoNode(int pos, vector<int>& p) *  
 *     - Neighbour operation: exchange two node(city).  
 * LocalSearch::reverseSequence(int pos, vector<int>& p) *  
 *     - Neighbour operation: reverse a sub-sequence.  
 * LocalSearch::popToFront(int pos, vector<int>& p) *  
 *     - Neighbour operation: pop a node to front.  
 *=====*/  
  
class LocalSearch: public TSPbase {  
public:  
    ...  
protected:  
    int LOOP_LIMITS;           // Maximum Loop Times  
    int EARLY_STOP;           // Early Stop Epoch  
    int VERBOSE;               // Verbose Message  
    int early_stop_counter;    // Early Stop Counter  
    double LOOSE_COEF;         // Coefficient to control loose op.  
    double delta_d;            // parameter  
    double delta_v;            // parameter  
    double* node_weights;      // Weight for selection  
    double* node_satisfaction; // Satisfaction for cities  
    double* node_active_value; // Activy for cities  
    vector<Individual> opt_hist; // History of optimal cost  
    vector<nop> n_ops;          // Vector of neighbour operations  
    ADJ_MAP_TYPE closest_city; // 2 Closest citys  
    ADJ_MAP_TYPE adj_city;     // Adjacent citys  
};
```

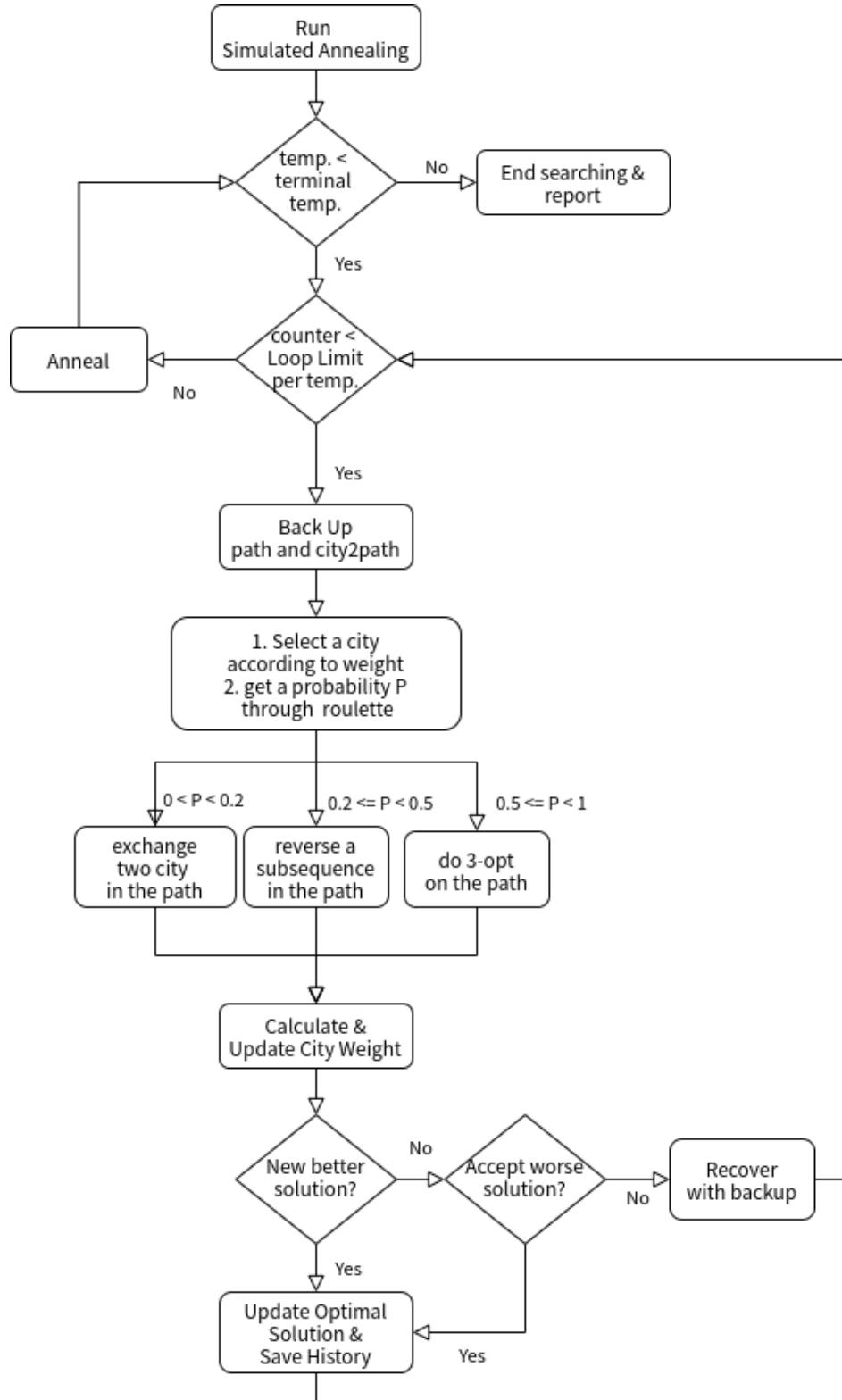
2.3 SimulatedAnnealing

SimulatedAnnealing 是基于 LocalSearch 实现的模拟退火算法。

与局部搜索不同的是：

1. 算法的结束由温度控制
2. 对于比当前解差的解也有机会选用，选用概率与两个解差值以及温度有关，温度越低，接受概率越低
3. 在高温状态下将会剧烈震荡，只有温度降低后才呈现收敛趋势并逐渐收敛。

2.3.1 流程图



2.3.2 代码分析

这里给出头部文件的实例，详细实现可参见 `src/LocalSearch.cpp`。

```
/****************************************************************************
 *             FileName : SimulatedAnnealing.h
 *             Author : Karl
 *          Created Date : June 6, 2020
 *          Updated : June 11, 2020 - Add Function
 *                      June 11, 2020 - Fix Bug
 *                      June 12, 2020 - Modify Perfomance
 *                      June 22, 2020 - Last Check
 *=====
 * @Functions:
 *   SimulatedAnnealing::SimulatedAnnealing(MAP_TYPE& origin,
 *                                         double LOOSE_COEF, double TEMP_INIT,
 *                                         double TEMP_END, int LOOPS_PER_TEMP,
 *                                         double ANNEALING_COEF, int VERBOSE)
 *   - Constructor.
 *   SimulatedAnnealing::runSA() - start Simulated Annealing.
 *   SimulatedAnnealing::run() - override run() in LocalSearch.
 *   SimulatedAnnealing::report() - save result to files.
 *
 *   SimulatedAnnealing::Metropolis(int pos, vector<int>& p)
 *   - criterion for acceptance of worse solution.
 *=====
 */
class SimulatedAnnealing: public LocalSearch {
public:
    ...
private:
    double TEMP_INIT;           // Initial temperature
    double TEMP_END;            // Terminal temperature
    int LOOPS_PER_TEMP;         // Loop times per temperature do
    double ANNEALING_COEF;      // Coefficient to control annealing speed
};
```

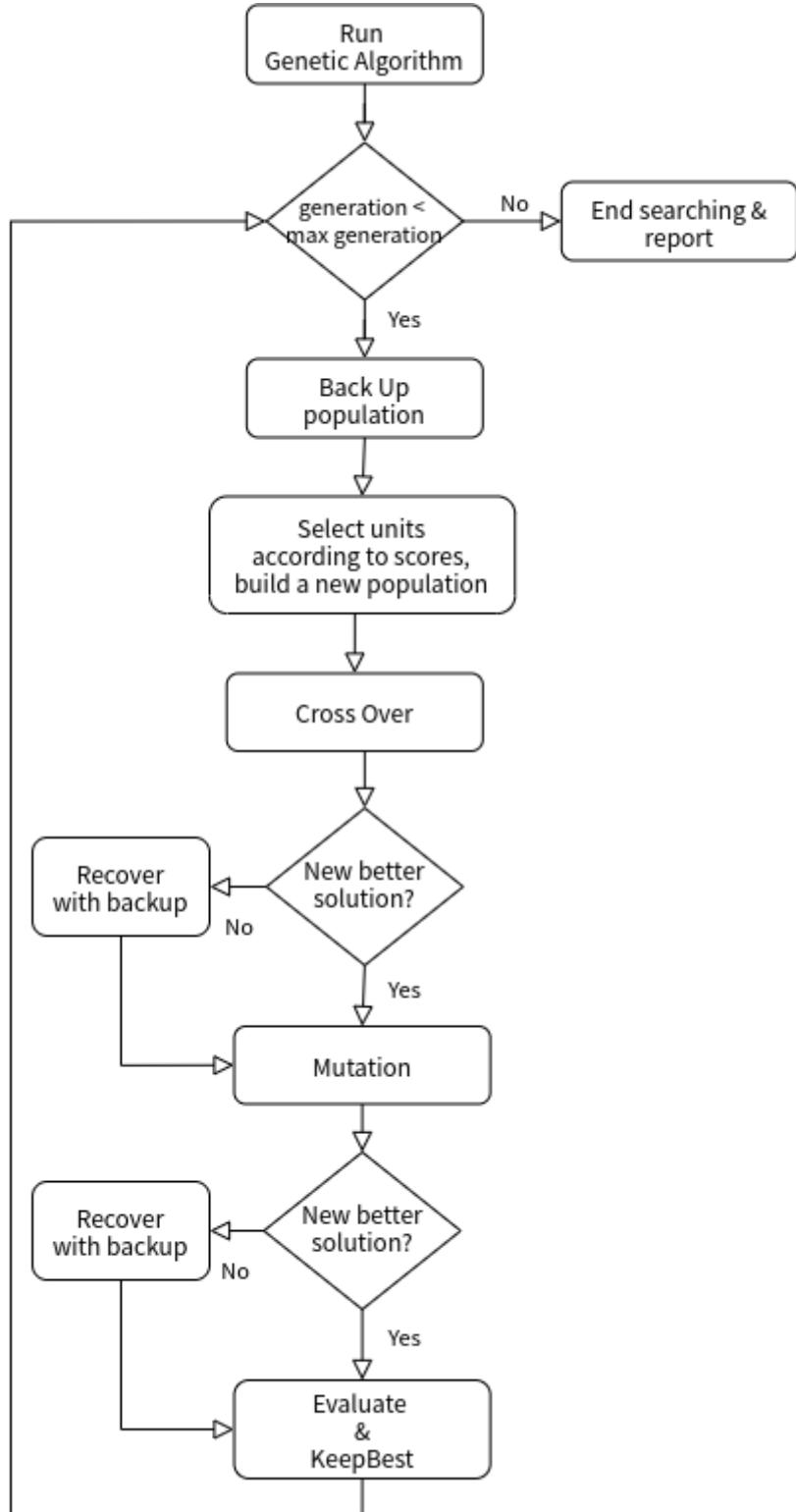
模拟退火的温度控制机制由以下属性控制：

1. 初始温度： `TEMP_INIT`
2. 终止温度： `TEMP_END`
3. 退火系数： `ANNEALING_COEF`

每次循环结束后，`新温度 = 当前温度 * 退火系数`，实现降温。

2.4 GeneticAlgorithm

2.4.1 流程图



2.4.2 交叉与变异算子

遗传算法是模拟生物种群繁衍的一种搜索策略，其中最关键的即为基因的交叉与变异操作。

交叉算子

基因的交叉即，两个个体基因的片段进行交换，在TSP问题中，基因即是一种可能的路径排列，基因交叉的过程，即将两条路径的一部分进行交换。

按以下流程进行交叉：

1. 遍历整个群体，每个个体随机选择一个其他个体进行交叉（不与相邻的个体进行交叉以避免近亲交配）
2. 每个个体按轮盘赌策略，随机产生一个概率，只有该概率超过预定的交叉概率时才允许交叉
3. 交叉后处理冲突
4. 更新新个体的路径代价

冲突处理：

由于一条路径中，城市是唯一的，所以基因的交叉可能导致非法路径的出现：

Path1 - 1 2 3 4 5 6 7 8 9

Path2 - 1 2 9 6 3 4 7 8 5

在位置2到4进行交叉：

Path1 - 1 **2 9 6** 5 6 7 8 9 - 城市9，6冲突

Path2 - 1 **2 3 4** 3 4 7 8 5 - 城市3，4冲突

分别遍历交叉后的两条路径，每条路径在访问到重复城市时，暂停等候另一条路径也访问到重复城市（原路径内城市编号唯一，因此当一条路径出现重复时，另一路径一定也会出现重复）；两条路径都暂停后，交换这两个城市即可。

```
Function crossOver:  
    elite_size:= POPULATION_SIZE * 0.2;  
    i:= elite_size;  
    while i < POPULATION_SIZE - 2  
        get random `prob` by roulette()  
        if prob <= CROSSOVER_PROB:  
            j:= get another unit j randomly  
            cross_point: = choose cross_point randomly  
            cross(population[i], population[j], cross_point);  
        end if  
        solveCrossOverConflict(population[i], population[j]);  
        update cost of i and j  
    end while  
end  
  
Function cross(UNIT i1, UNIT i2, pos):  
    copy i2[pos:] to i1[pos:]  
    copy i1[pos:] to i2[pos:]  
end  
  
Function solveCrossOverConflict(UNIT i1, UNIT i2):  
    i:= 0, j:= 0, N:= Numbers of City;  
    while i < N and j < N:  
        if i1[i] is visited and i2[j] is visited:  
            swap i1[i] and i2[j]  
            i++, j++;  
        end if  
        if i < N and i1[i] is not visited:  
            note i1[i] as visited, i++;
```

```

        end if
        if j < N and i2[j] is not visited:
            note i2[j] as visited, j++;
        end if
    end while
end

```

变异算子

基因的变异即，路径内部发生非正常的变化，本实验中采用局部搜索中邻域操作作为变异的操作。

为提高收敛速度，只有优秀的变异会被采纳。

按以下流程进行变异：

1. 遍历整个群体
2. 每个个体按轮盘赌策略，随机产生一个概率，只有该概率超过预定的变异概率时才允许变异
3. 全部个体完成变异后，只有使个体变优秀的变异被采用

```

Function mutation(last_population):
    elite_size:= POPULATION_SIZE * 0.2;
    for i from elite_size to POPULATION_SIZE-1:
        get random `prob` by roulette();
        if prob <= MUTATION_PROB:
            mutate(population[i]);
        end if
    end for
    for i from elite_size to POPULATION_SIZE-1:
        if newcost[i] is not better than oldcost[i]:
            Unit i is recoverd by old unit;
        end if
    end for
end

Function mutate(UNIT ind):
    get random `prob` by roulette();
    buildCity2Path(city2path, ind.second);
    get random `pos` by roulette();
    if prob < 0.25:
        neighbour operation 0 at `pos`;
    else if prob < 0.5:
        neighbour operation 1 at `pos`;
    else if prob < 1:
        neighbour operation 2 at `pos`;
    end if
    update cost of unit ind;
end

```

2.4.3 评分机制及精英保留策略

评分是基于TSP问题的最优解以及当前路径长度设定的：

$$Score(Unit) = \frac{1}{cur_len - optimal_len}$$

精英保留策略：

在每一代中，将表现最好（路径代价最小、评分最高）的个体优先保留20%，在后续随机选取个体时就不再进行选择。这些精英个体不会主动进行交叉、变异操作，但可以接受其他个体的交叉请求。

2.4.4 代码分析

```
*****  
*          FileName : GeneticAlgorithm.h           *  
*          Author : Karl                         *  
*          Created Date : June 12, 2020           *  
*          Updated : June 13, 2020 - Modify Perfomance*  
*                      June 23, 2020 - Last Check        *  
*=====*  
* @Functions:  
*  GeneticAlgorithm::GeneticAlgorithm(MAP_TYPE& origin,           *  
*          int LOOP_LIMITS, int POPULATION_SIZE,           *  
*          int GENERATIONS, int MUTATION_TIMES = 3,       *  
*          double CROSSOVER_PROB = 0.7,                   *  
*          double MUTATION_PROB = 0.2,                   *  
*          double OPT_COST = 0.0, int VERBOSE = 0)         *  
*          - Construtor.                            *  
*  GeneticAlgorithm::runGA() - start Genetic Algorithm.      *  
*  GeneticAlgorithm::run() - override run() in LocalSearch.   *  
*  GeneticAlgorithm::init() - Initialization.             *  
*  GeneticAlgorithm::keepBest() - Keep the elite, the elite    *  
*          never crossover or mutate.                  *  
*  GeneticAlgorithm::selectUnit() - select from population.    *  
*  GeneticAlgorithm::crossOver() - crossover genetically.     *  
*  GeneticAlgorithm::cross(UNIT& i1, UNIT& i2, int pos)       *  
*          - crossover between i1 and i2.                  *  
*  GeneticAlgorithm::solveCrossOverConflict()            *  
*          - solve conflicts(same city in 1 path)*  
*  GeneticAlgorithm::mutation(vector<UNIT> last_population)  *  
*          - mutate genetically.                     *  
*  GeneticAlgorithm::mutate(UNIT& ind) - unit ind mutates.    *  
*  GeneticAlgorithm::score(vector<int>& p)                 *  
*          - calculate score of path `p`.                *  
*  GeneticAlgorithm::evaluate() - evaluate current population.*  
*  GeneticAlgorithm::checkPath() - check path's validity.     *  
*  GeneticAlgorithm::getElite() - return elite solution.      *  
*  GeneticAlgorithm::report() - save result to files.        *  
*=====*/  
  
class GeneticAlgorithm: public LocalSearch {  
public:  
    ...  
private:  
  
    int POPULATION_SIZE;           // The size of the population  
    int GENERATIONS;              // Iterations the population will do  
    int MUTATION_TIMES;           // Times to try to mutate per loop  
    int GA_VERBOSE;               // GA Verbose Message  
    double CROSSOVER_PROB;         // The probability to crossover  
    double MUTATION_PROB;          // The probability to mutate  
    double OPT_COST;               // The optimal cost of the tsp  
    UNIT ELITE;                  // The best unit in the population  
    vector<UNIT> elite_hist;       // History of elites  
    vector<UNIT> population;        // Population of units  
    vector<double> scores;          // Scores to evaluate unit  
    vector<double> chance_by_score; // Accumulation of scores  
};
```

3 结果分析

3.1 实验环境

3.1.1 系统信息

OS	Ubuntu 18.04.4 LTS
CPU	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz

3.1.2 开发工具

- Vscode + make
- gcc/g++ 7
- python 3.7

3.2 编译运行

3.2.1 初始状态

```
$ tree ./LocalSearch
.
├── bin
└── src
    ├── LocalSearch.cpp
    ├── LocalSearch.h
    ├── localsearch_main.cpp
    ├── Makefile
    ├── TSPbase.cpp
    └── TSPbase.h
└── testcases
    └── ...
9 directories, 21 files

...
$ tree ./SimulatedAnnealing
.
├── bin
└── src
    ├── LocalSearch.cpp
    ├── LocalSearch.h
    ├── Makefile
    ├── SimulatedAnnealing.cpp
    ├── SimulatedAnnealing.h
    ├── simulatedannealing_main.cpp
    ├── TSPbase.cpp
    └── TSPbase.h
└── testcases
    └── ...
7 directories, 26 files

...
$ tree ./GeneticAlgorithm
.
└── bin
```

```
|   └── src
|       ├── GeneticAlgorithm.cpp
|       ├── GeneticAlgorithm.h
|       ├── geneticalgorithm_main.cpp
|       ├── LocalSearch.cpp
|       ├── LocalSearch.h
|       ├── Makefile
|       ├── TSPbase.cpp
|       └── TSPbase.h
└── testcases
    └── ...
```

5 directories, 32 files

3.2.2 编译

在三种算法各自文件夹内：

```
$ cd src/ && make
```

示例：LocalSearch的Makefile

```
CXX = g++-7
CXXFLAGS = -Wall -Werror -Wextra -pedantic -std=c++17 -g -fsanitize=address
LDFLAGS = -fsanitize=address

SRC = localsearch_main.cpp TSPbase.cpp LocalSearch.cpp
OBJ = $(SRC:.cpp=.o)
EXEC = ./bin/local_search

all: $(EXEC)

$(EXEC): $(OBJ)
    $(CXX) $(LDFLAGS) -o $@ $(OBJ) $(LBLIBS)

clean:
    rm -rf $(OBJ) $(EXEC)
```

```
g++-7 -Wall -Werror -Wextra -pedantic -std=c++17 -g -fsanitize=address -c -o
localsearch_main.o localsearch_main.cpp
g++-7 -Wall -Werror -Wextra -pedantic -std=c++17 -g -fsanitize=address -c -o
TSPbase.o TSPbase.cpp
g++-7 -Wall -Werror -Wextra -pedantic -std=c++17 -g -fsanitize=address -c -o
LocalSearch.o LocalSearch.cpp
g++-7 -fsanitize=address -o ./bin/local_search localsearch_main.o TSPbase.o
LocalSearch.o
```

3.2.3 运行及自定参数

1. 切换到 bin/ 文件夹

```
$ ./local_search  
Invalid Usage.  
>> ./local_search [TSP_FILE_NAME]  
e.g.:  
>> ./local_search a280
```

2. 按照使用格式调用，此处以a280问题为例

需要进行搜索的问题需要提前将文件 `[tsp_name].tsp` 以及 `[tsp_name].opt.tour` 复制到 `testcases/` 中。

```
$ ./local_search a280
```

```
# karl @ karl in ~/Coding/AI-algorithms/GASA/LocalSearch/bin [18:39:10]  
$ ./local_search a280  
=====  
          Local Search  
          a technique to approach TSP's optimal solution  
          Author : Karl  
          Date : June 9, 2020  
=====  
To get started, please ensure that your dataset is in  
TSPLIB style, with EUC2D EDGE_WEIGHT_TYPE.  
@Parameters:  
(1) LOOP LIMITS: the maximum times that local search run  
... (Default: 1000000)  
(2) LOOSE COEF: the coefficient that controls the degree  
... those active cities will slow down at.  
... (Default: 0.99, [0.9-0.99] suggested)  
(3) EARLY STOP: the loops after which when no better  
... solutions have been found  
... (Default: 500000)  
(4) VERBOSE: whether to print out debug message  
... (Default: 1)  
NOTE: Press [ENTER] to use default value.  
=====  
>> LOOP LIMITS[1000000]:  
>> LOOSE COEF[0.99]:  
>> EARLY STOP[500000]:  
>> VERBOSE[1]:  
=====  
User Input:  
LOOP LIMITS : 1e+06  
LOOSE COEF : 0.99  
EARLY STOP : 500000  
VERBOSE : 1  
=====  
>> Press [Enter] to run on file: ../testcases/a280.tsp:
```



```
>> Current Optimal Solution Updated:  
>> From 2931.07 to 2931.07.  
>> Current Optimal Solution Updated:  
>> From 2931.07 to 2930.52.  
>> Current epoch: 100000 / 1000000 in 2.99964 s.  
>> Current Optimal Solution Updated:  
>> From 2930.52 to 2927.26.  
>> Current Optimal Solution Updated:  
>> From 2927.26 to 2925.47.  
>> Current Optimal Solution Updated:  
>> From 2925.47 to 2925.24.  
>> Current Optimal Solution Updated:  
>> From 2925.24 to 2921.15.  
>> Current epoch: 200000 / 1000000 in 3.209 s.  
>> Current epoch: 300000 / 1000000 in 2.99923 s.  
>> Current epoch: 400000 / 1000000 in 2.97206 s.  
>> Current epoch: 500000 / 1000000 in 2.95564 s.  
>> Current epoch: 600000 / 1000000 in 3.06401 s.  
>> Search is done in 19.2844s.  
>> Best solution found: 2921.15.
```

在程序运行后，接受参数或回车使用参数缺省值，再次回车确认将开始搜索。

相关参数已在命令行中说明。

```
$ ./simulated_annealing a280  
$ ./genetic_algorithm a280
```

```

$ ./simulated_annealing a280
=====
Simulated Annealing
a technique simulating annealing process,
to approach TSP's optimal solution.

Author : Karl
Date : June 11, 2020
=====

To get started, please ensure that your dataset is in
TSPLIB style, with EUC2D EDGE_WEIGHT_TYPE.

@Parameters:
(1) TEMP_INIT: the initial temperature
... (Default: 50000)
(2) TEMP_END: the end temperature
... (Default: 0.00001, [0.01-0.00001] suggested)
(3) LOOPS_PER_TEMP: the loop times under
... a certain temperature
... (Default: 10000)
(4) LOOSE_COEF: the coefficient that controls the degree
... those active citys will slow down at.
... (Default: 0.99, [0.9-0.99] suggested)
(5) ANNEALING_COEF: the coefficient that controls the
... speed the temperature cools down at.
... (Default: 0.99)
(6) VERBOSE: whether to print out debug message
... (Default: 1)

NOTE: Press [ENTER] to use default value.

=====
>> TEMP_INIT[50000]:
>> TEMP_END[0.00001]:
>> LOOPS_PER_TEMP[10000]:
>> LOOSE_COEF[0.99]:
>> ANNEALING_COEF[0.99]:
>> VERBOSE[1]:
=====
User Input:
TEMP_INIT : 50000
TEMP_END : 1e-05
LOOPS_PER_TEMP : 10000
LOOSE_COEF : 0.99
ANNEALING_COEF : 0.99
VERBOSE : 1
=====
>> Press [Enter] to run on file: ../testcases/a280.tsp: |
```

```

$ ./genetic_algorithm a280
=====
Genetic Algorithm
a technique simulating biological reproduction,
to approach TSP's optimal solution.

Author : Karl
Date : June 20, 2020
=====

To get started, please ensure that your dataset is in
TSPLIB style, with EUC2D EDGE_WEIGHT_TYPE.

@Parameters:
(1) POPULATION_SIZE: the size of population in GA.
... (Default: 500)
(2) GENERATIONS: the iterations GA will do.
... (Default: 10000)
(3) LOOPS_INIT: the loops of local search as seeds for GA.
... (Default: 1000)
(4) MUTATION_TIMES: the times each generation will take.
... (Default: 2)
(5) CROSSOVER_PROB: probability of crossover operator.
... (Default: 0.7)
(6) MUTATION_PROB: probability of mutation operator.
... (Default: 0.1)
(7) OPT_COST: optimal solution of current tsp problem.
... (Default: 0.0)
(8) VERBOSE: whether to print out debug message
... (Default: 1)

NOTE: Press [ENTER] to use default value.

=====
>> POPULATION_SIZE[500]:
>> GENERATIONS[10000]:
>> LOOPS_INIT[1000]:
>> MUTATION_TIMES[2]:
>> CROSSOVER_PROB[0.7]:
>> MUTATION_PROB[0.1]:
>> OPT_COST[0.0]:
>> VERBOSE[1]:
>> Initialized the population with localsearch, as seeds.
>> New Best UNIT Found: Score From 0 to 9.14825e-05
=====
```

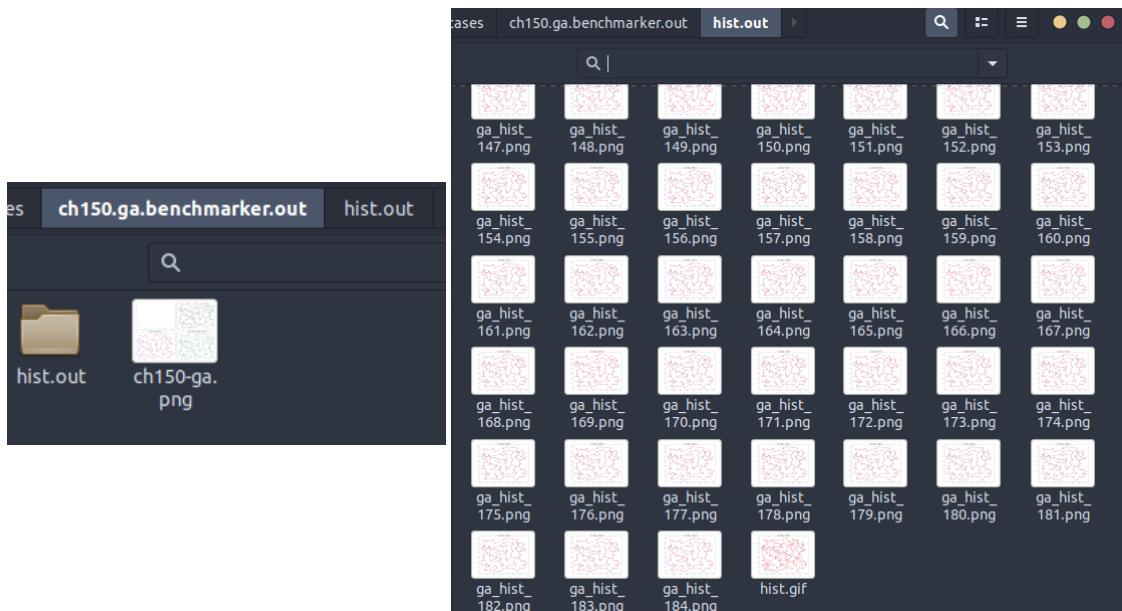
3.3.4 结果可视化

每次运行完程序，结果将保存在 `testcases/result.[method].tour` (搜索得到的最佳路径) 以及 `testcases/result.[method].hist` (搜索记录) 中。

本实验采用Python + matplotlib进行数据可视化：

```
$ cd testcases/
$ python Benchmark.py [Method(ls/sa/ga)] [TSP_FILE_NAME]
# e.g. python Benchmark.py ls a280
```

Python脚本会创建文件夹： `[TSP_FILE_NAME].[method].benchmark.out/`，其中包括历史记录图示、路线变化GIF，以及最优解对比图等。



3.3 搜索结果及比较

3.3.1 实验数据对比

Local Search

TSP	1st	2nd	3rd	4rd	5rd	AVE
kroC100	22523.3	22667.5	24123.4	22938.9	21711.3	22792.88 (Loss 9.85%)
ch150	6929.42	7696.9	7076.38	6996.5	7604.08	7260.656 (Loss 11.22%)
tsp225	4276.71	4404.6	4344.26	4229.38	4287.84	4308.558 (Loss 9.94%)
a280	2951.92	2888.39	2902.9	2918.46	2872.16	2906.766 (Loss 12.7%)
pcb442	58725.1	57542.1	57856.8	56568.1	57094.8	57557.38(Loss 13.35%)

Simulated Annealing

TSP	1st	2nd	3rd	4rd	5rd	AVE
kroC100	20871.4	20871.4	20750.8	20993.5	20769.9	20851.4 (Loss 0.49%)
ch150	6553.66	6563.02	6563.02	6559.6	6565.38	6560.936 (Loss 0.50%)
tsp225	3973.63	3948.07	3964.74	3972.09	3973.63	3968.232 (Loss 1.26%)
a280	2590.18	2634.79	2590.18	2597.54	2651.87	2612.91 (Loss 1.31%)
pcb442	51468.7	52130.3	51981.3	52061.9	51747.6	51877.96 (Loss 2.17%)

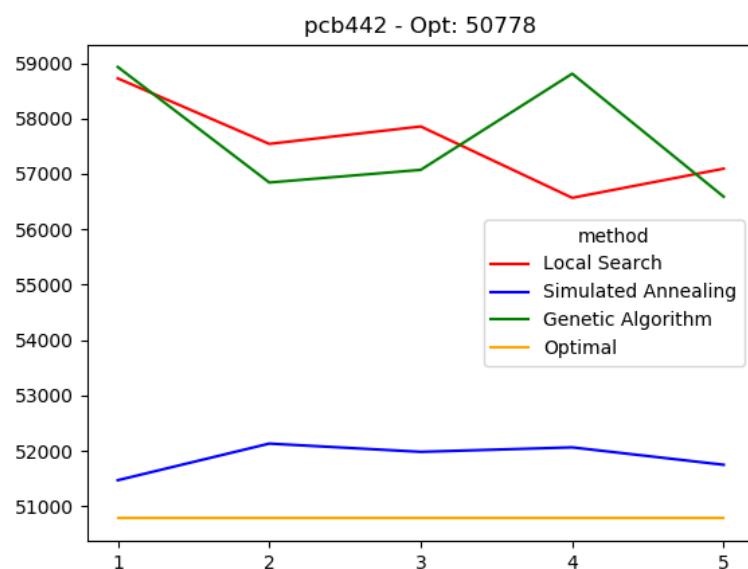
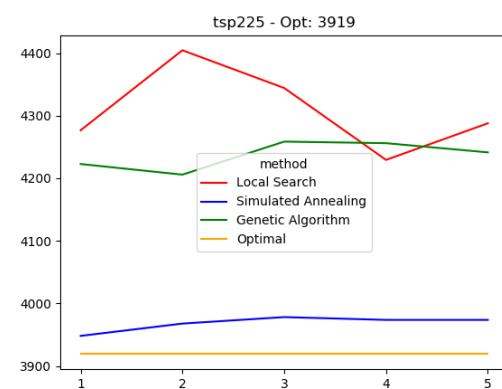
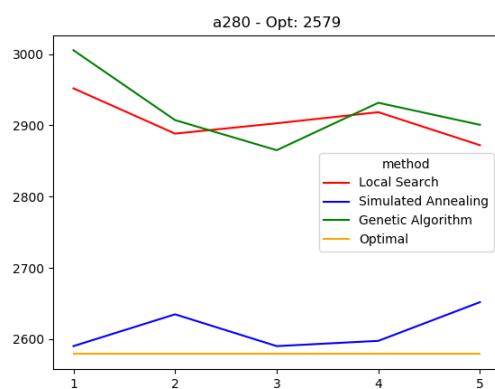
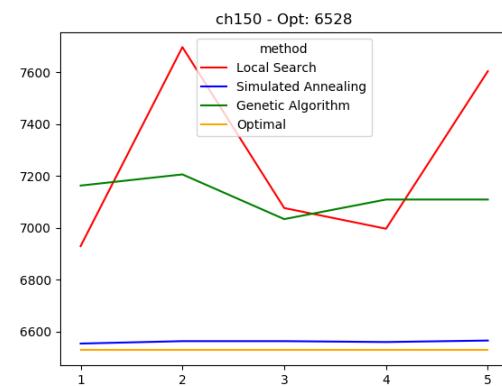
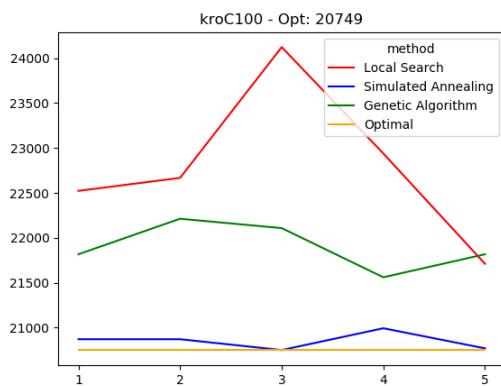
Genetic Algorithm

TSP	1st	2nd	3rd	4rd	5rd	AVE
kroC100	21817.3	22212.8	22108.4	21560.9	21817.3	21903.34(Loss 5.56%)
ch150	7162.96	7205.97	7033.58	7109.15	7109.15	7124.162 (Loss 9.13%)
tsp225	4222.78	4205.8	4258.63	4256.1	4241.42	4236.946 (Loss 8.11%)
a280	3005.36	2907.37	2865.11	2931.69	2900.76	2922.06 (Loss 13.3%)
pcb442	58931	56844.7	57073.4	58810	56588.3	57649.48 (Loss 13.5%)

summary

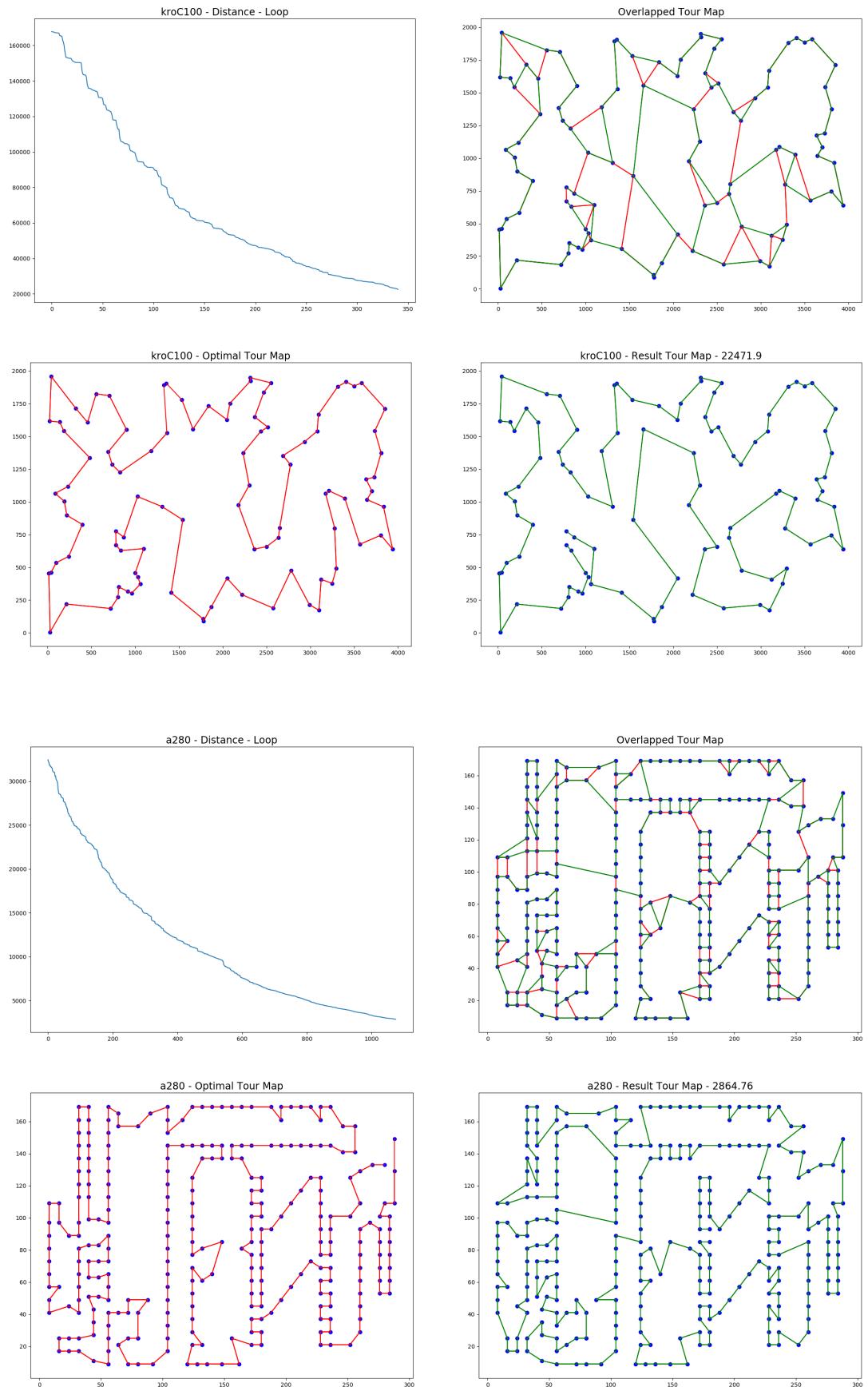
TSP Name	LocalSearch	Simulated Annealing	Genetic Algorithm
kroC100	22792.88 (Loss 9.85%)	20851.4 (Loss 0.49%)	21903.34(Loss 5.56%)
ch150	7260.656 (Loss 11.22%)	6560.936 (Loss 0.50%)	7124.162 (Loss 9.13%)
tsp225	4308.558 (Loss 9.94%)	3968.232 (Loss 1.26%)	4236.946 (Loss 8.11%)
a280	2906.766 (Loss 12.7%)	2612.91 (Loss 1.31%)	2922.06 (Loss 13.3%)
pcb442	57557.38(Loss 13.35%)	51877.96 (Loss 2.17%)	57649.48 (Loss 13.5%)

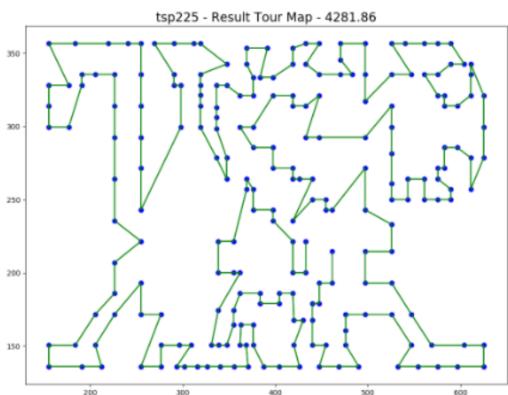
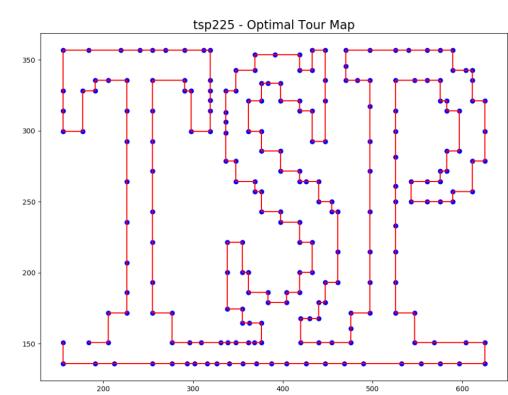
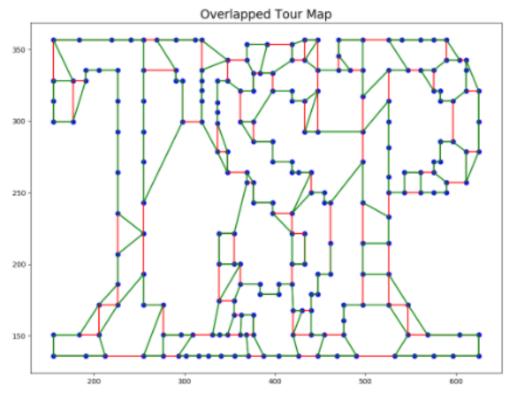
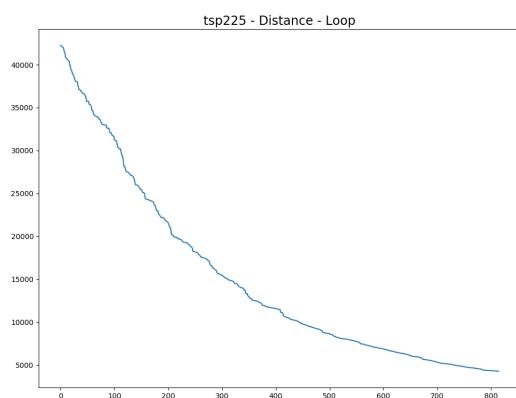
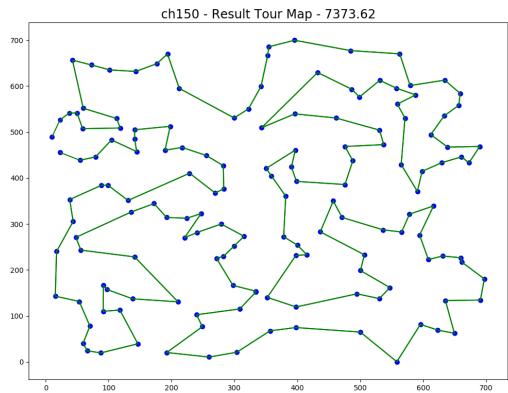
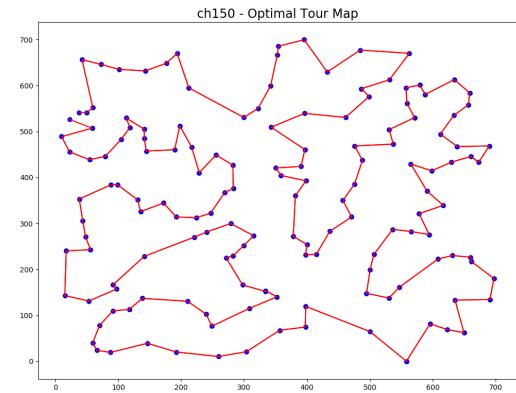
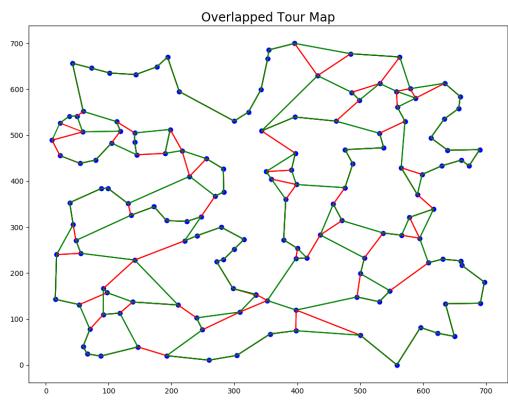
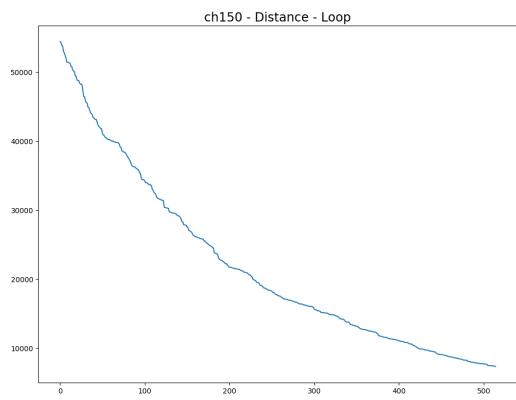
3.3.2 实验数据可视化

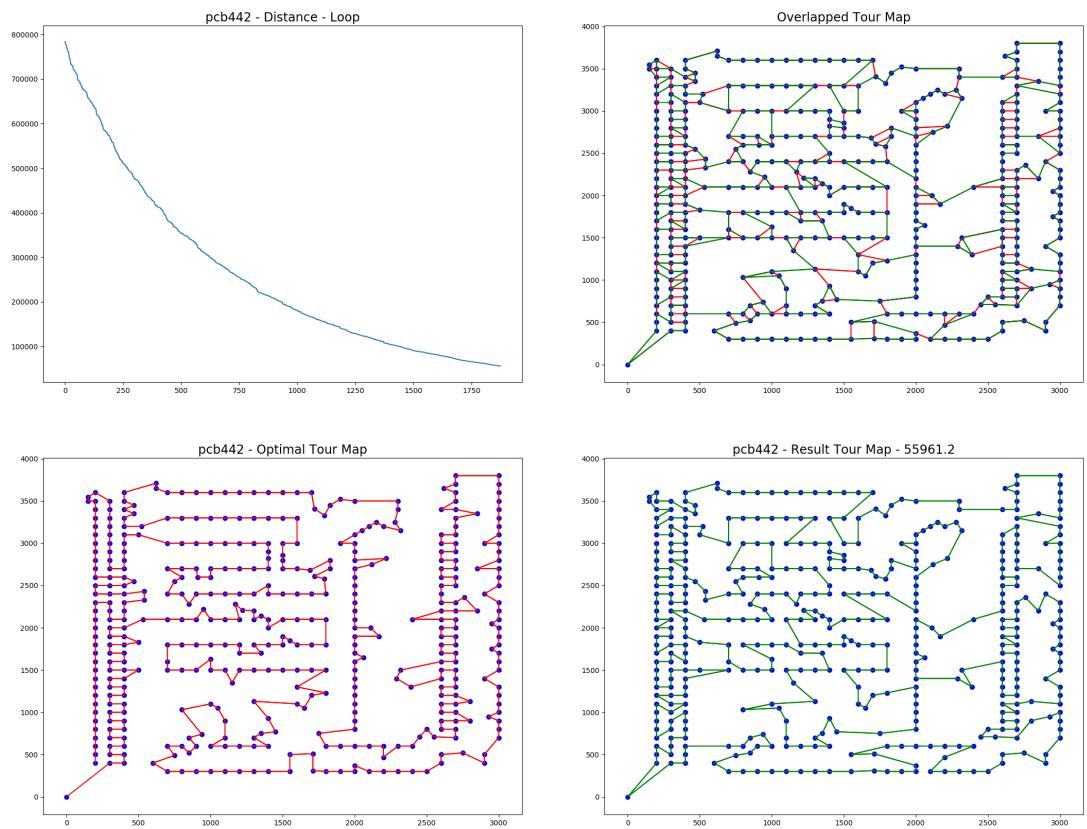


3.3.3 路线对比

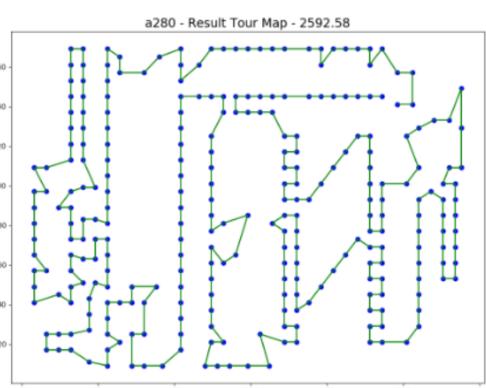
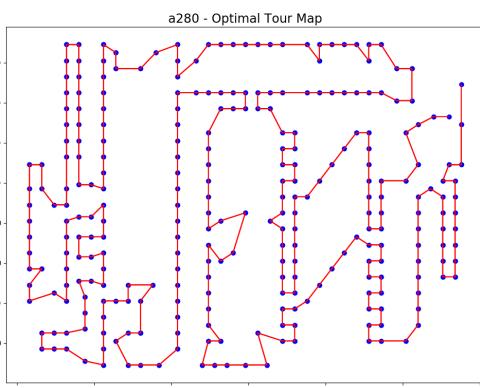
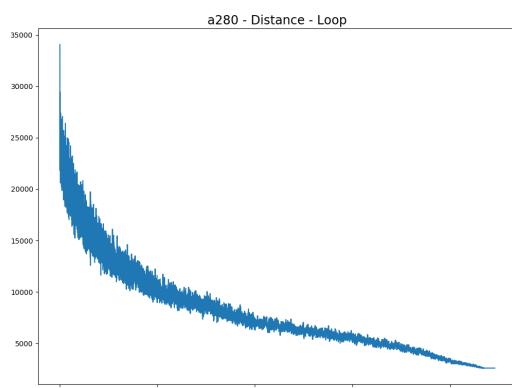
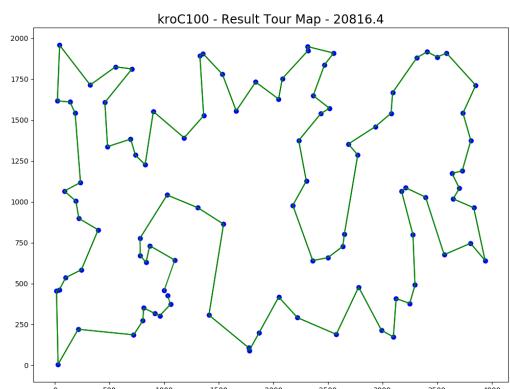
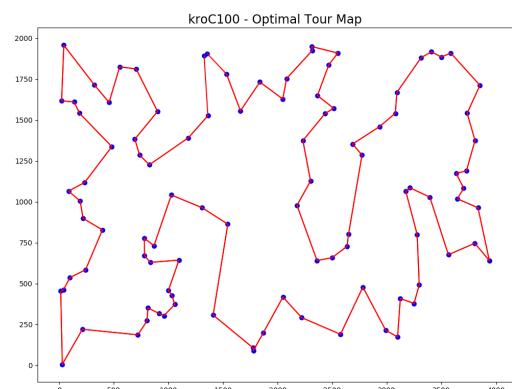
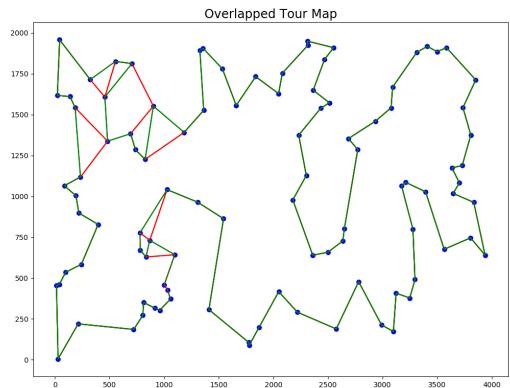
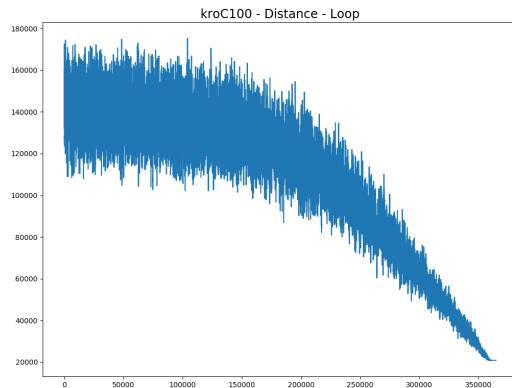
Local Search

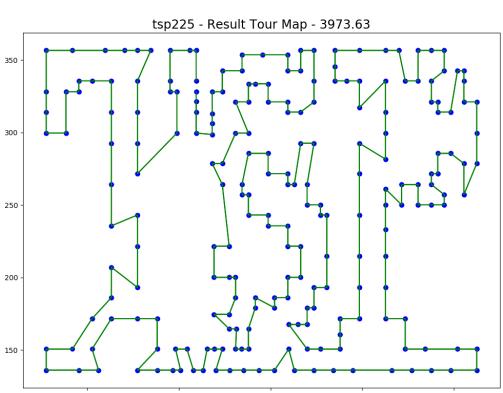
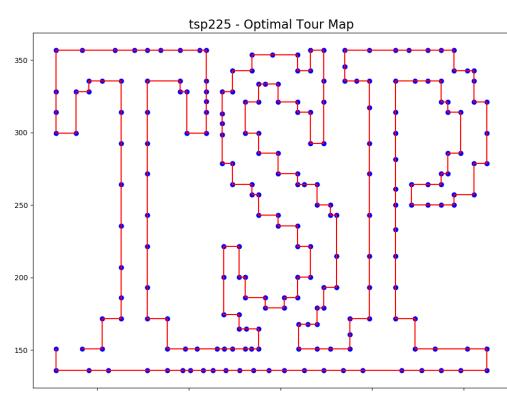
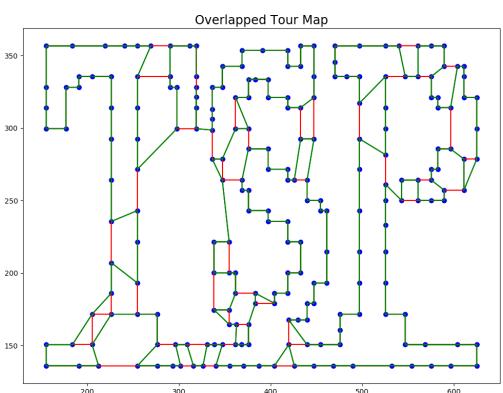
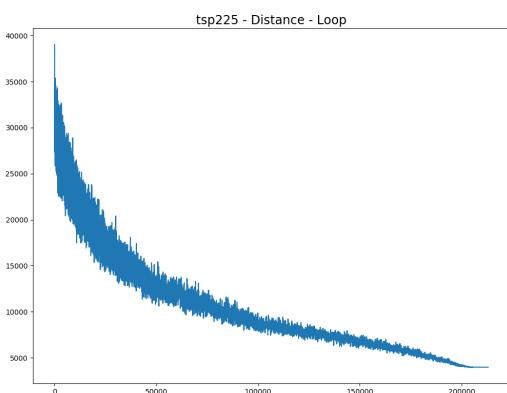
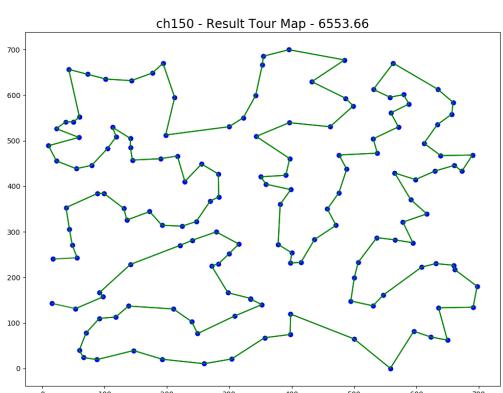
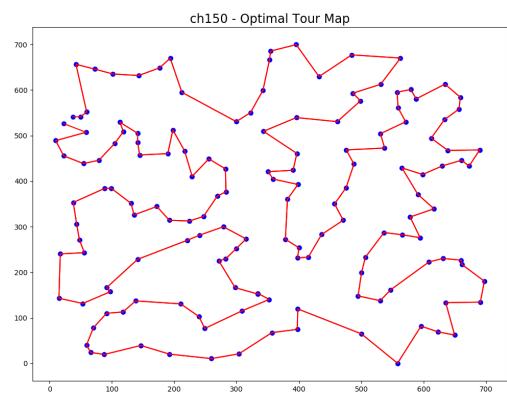
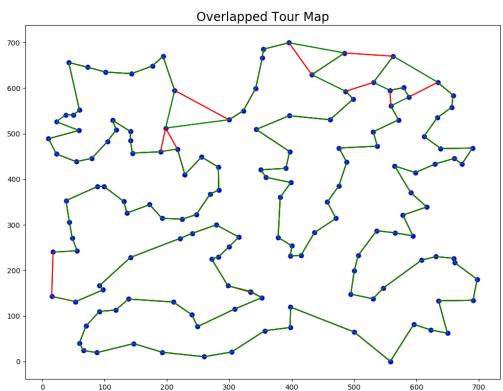
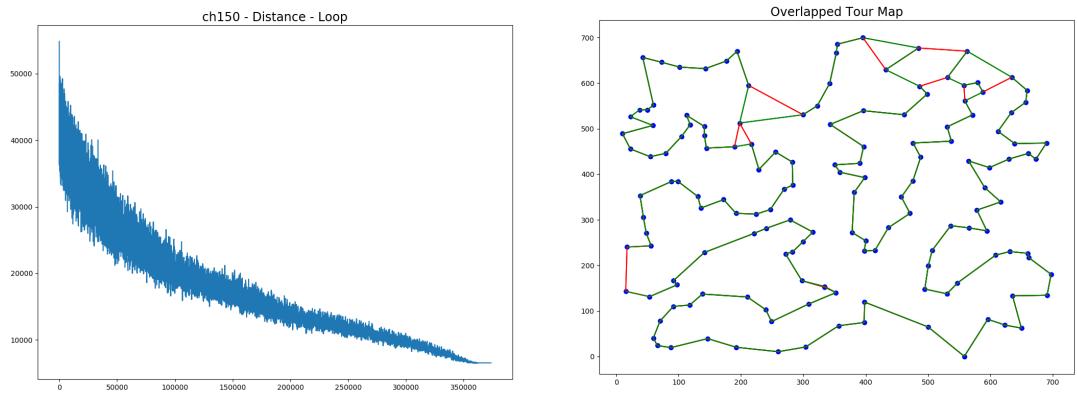


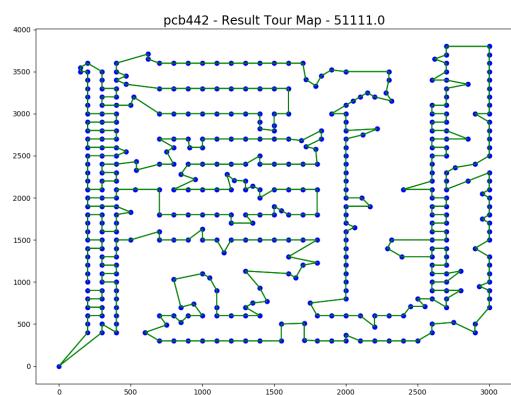
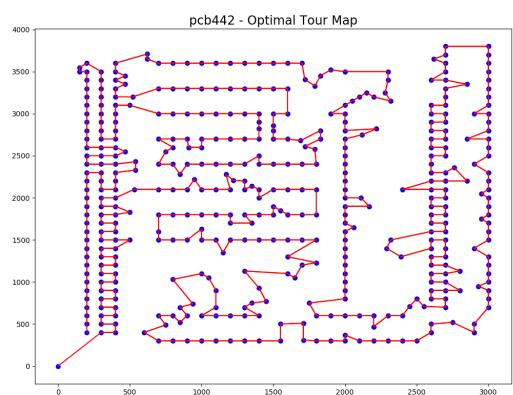
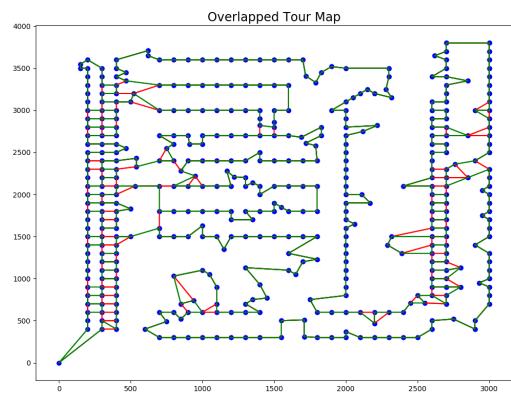
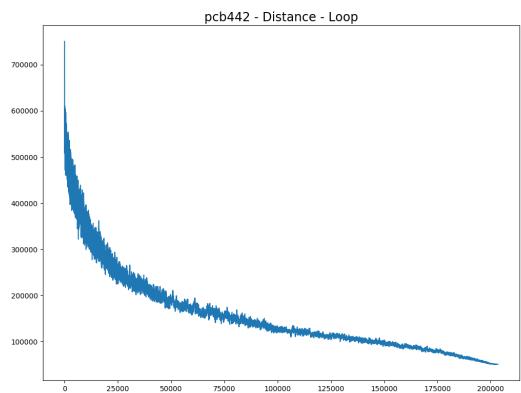




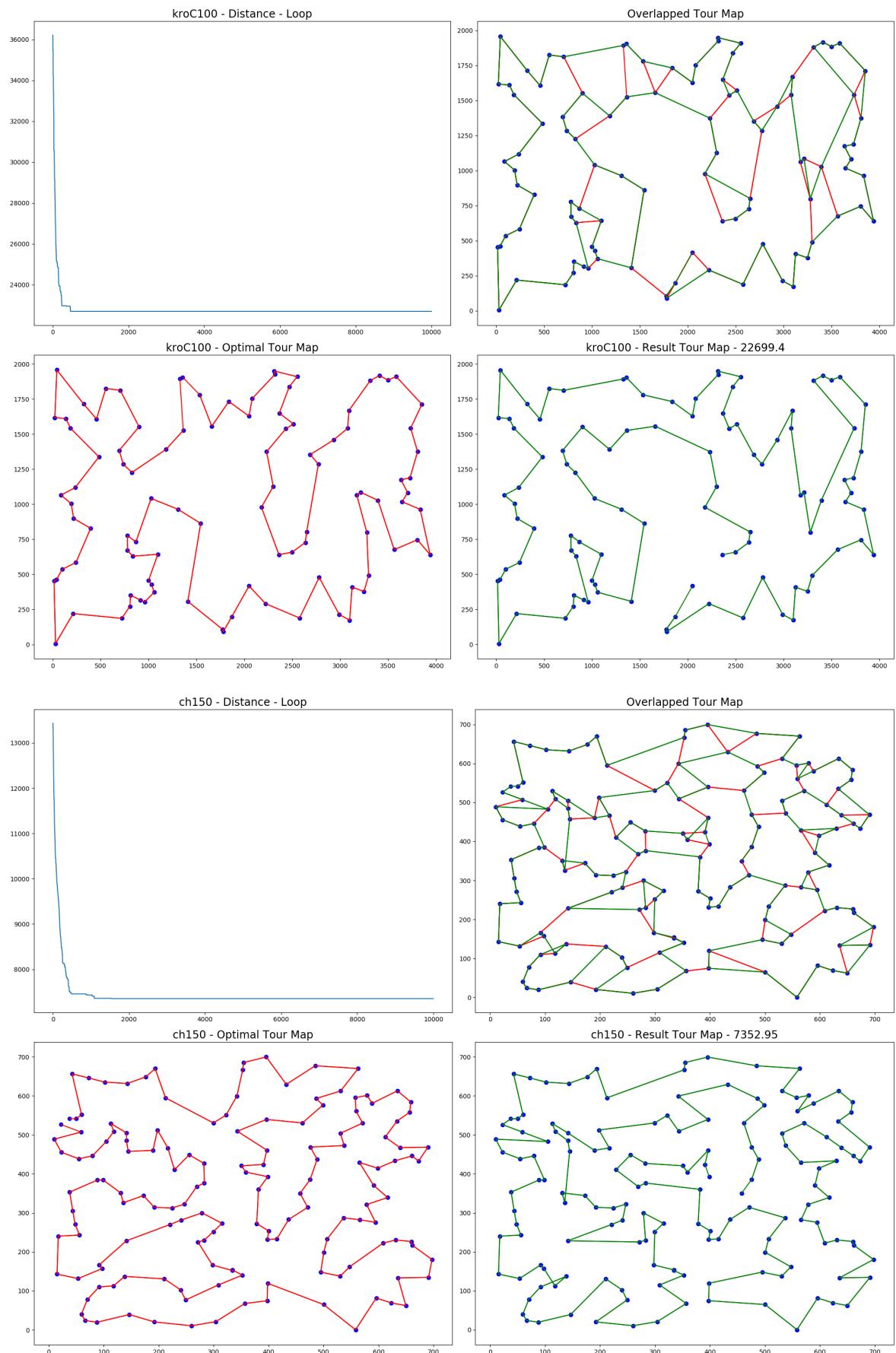
Simulated Annealing

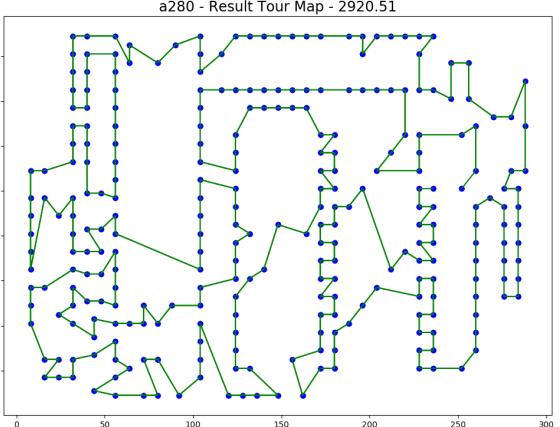
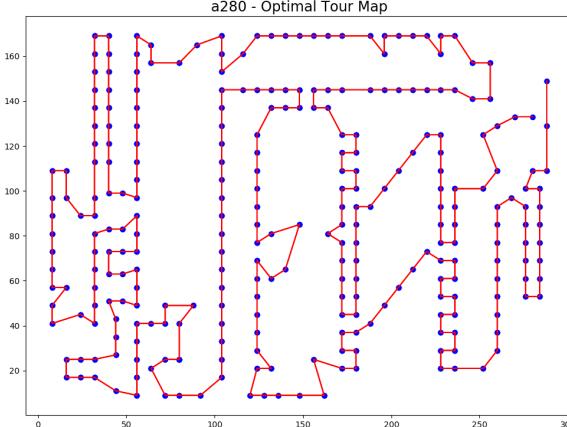
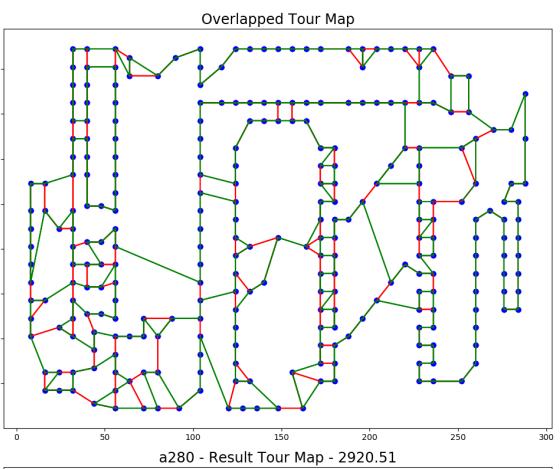
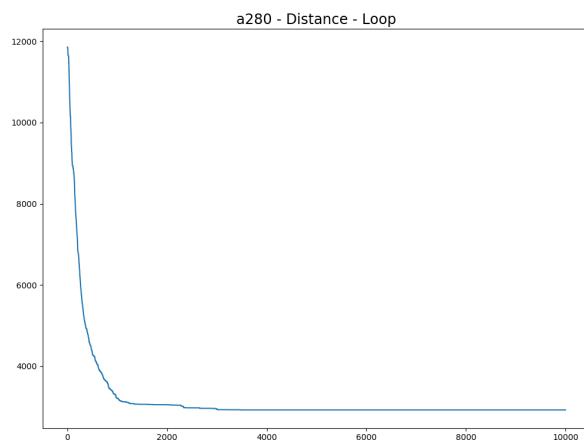
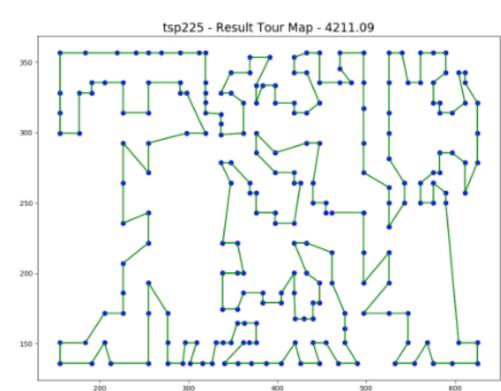
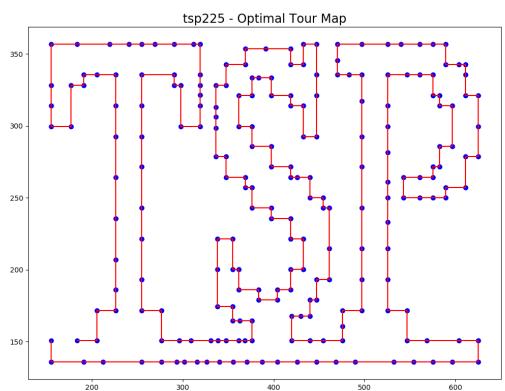
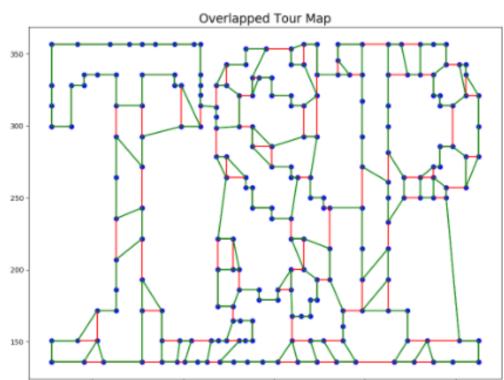
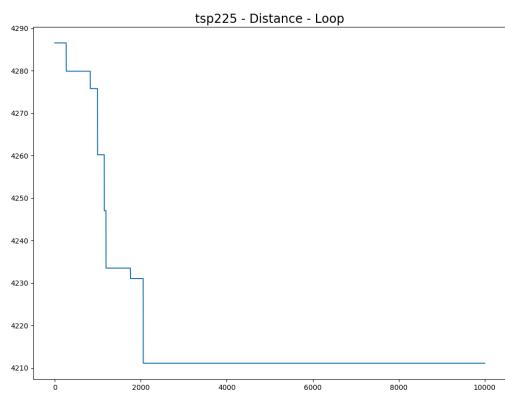


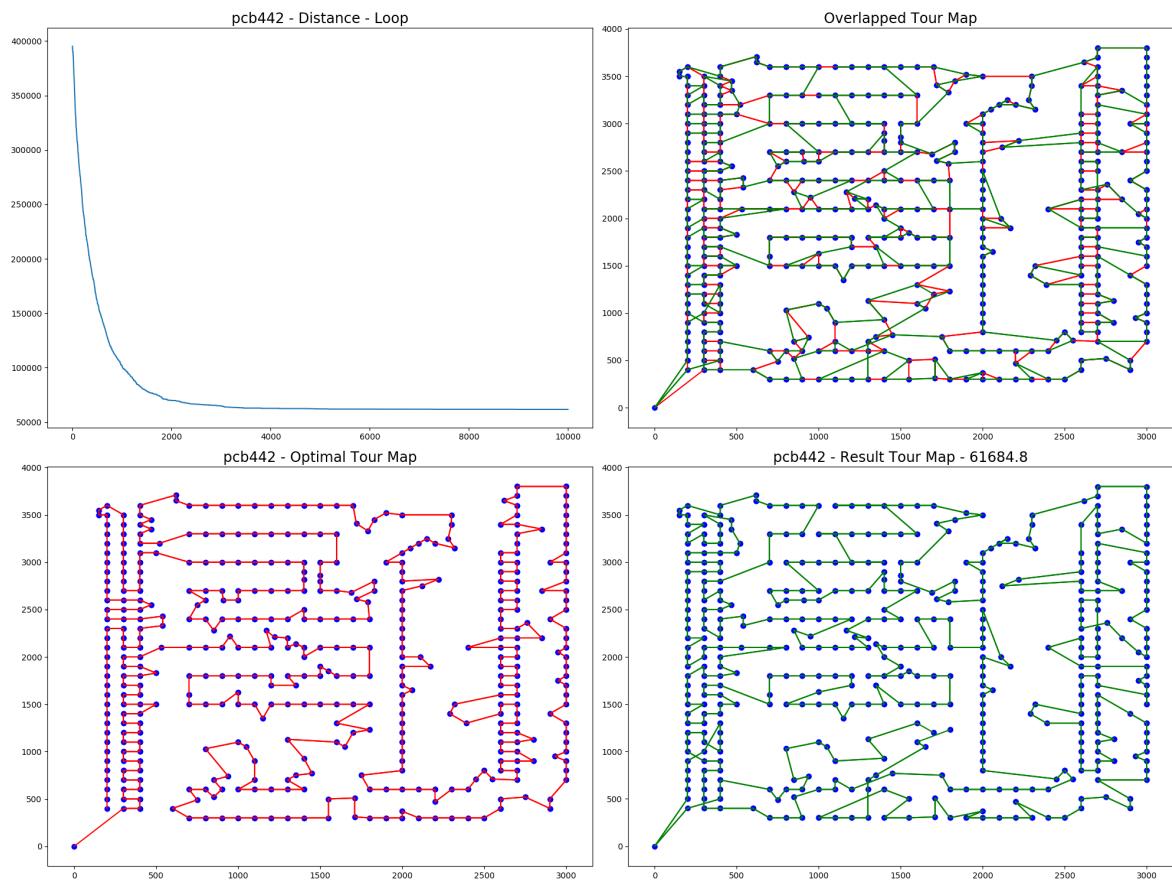




Genetic Algorithm







3.4 性能分析

TSP Name	LocalSearch	Simulated Annealing	Genetic Algorithm
kroC100	22792.88 (Loss 9.85%)	20851.4 (Loss 0.49%)	21903.34(Loss 5.56%)
ch150	7260.656 (Loss 11.22%)	6560.936 (Loss 0.50%)	7124.162 (Loss 9.13%)
tsp225	4308.558 (Loss 9.94%)	3968.232 (Loss 1.26%)	4236.946 (Loss 8.11%)
a280	2906.766 (Loss 12.7%)	2612.91 (Loss 1.31%)	2922.06 (Loss 13.3%)
pcb442	57557.38(Loss 13.35%)	51877.96 (Loss 2.17%)	57649.48 (Loss 13.5%)
Time	around 10s	150 ~ 300s	300s

[回顾实验数据](#)

算法精度

再次参考实验结果表，可以看到，**模拟退火算法的效果最为突出**，误差都在**3%**以内，另外两个算法表现一般，误差在**10%**上下波动。

算法效率

同时，除去精度，三个算法的速度也各有区别，模拟退火以及遗传算法的耗时较高，局部搜索耗时较低；

这是由于算法性质导致的，存在优化空间。

路径的变化与交叉

首先，随机初始化的路径往往是充满大量交叉路径，使得路径代价大大提高，各种领域操作搜索的原理实质上是在尽量消除路径代价。

为了充分观察在搜索过程中路径的变化，我将实验过程中的路径变化取样并且压在GIF图中，并建立了网页示例，可访问[DEMO](#)中查看(GIF文件较大，需要时间加载)。

3.5 探索与展望

1. 尝试更多邻域操作。
2. 尝试不同的参数而不是设定的默认参数（默认参数已经过调试、修订）。
3. 尝试将搜索并行化，遗传算法显然是可并行的。
4. 探究遗传算法效果不佳的原因（概率的设定？种群的大小？迭代次数？交叉变异操作？精英策略以外的策略？如何避免陷入局部最优？）
5. 提高模拟退火算法的效率（如何选择一个更合适的退火系数？）

对于默认初始温度50000度，默认终止温度1e-5，以0.99的退火系数需要约2200次降温，如何设定退火系数值得推敲。

4 结论

模拟退火算法、遗传算法，对于计算机专业的学生来说并不陌生，只是在实验前都是纸上谈兵。

在本次实验中，从逻辑设计开始，完成了三个搜索算法的C/C++实现，虽然细节手法仍然稚嫩，性能、效率上都沒有做到最优，但对了解算法、了解TSP问题帮助极大。

实验要求中相关问题

1. (模拟退火) 求得的解不要超过最优化的10%，并能够提供可视化图形界面，观察路径的变化和交叉程度。

求得的解精度在3%以内，效果很好。

可视化图形界面由python实现，路径的变化和交叉程度在上文已经进行了介绍。

2. (遗传算法) 和之前的模拟退火算法（采用相同的局部搜索操作）进行比较

效果不如模拟退火算法好，可能出现的问题已在上文进行了介绍。

3. (遗传算法) 得出设计高效遗传算法的一些经验，并比较单点搜索和多点搜索的优缺点。

在概率的设定、种群的大小、迭代次数、交叉变异操作、精英策略以外的策略、避免陷入局部最优上需要多下功夫。

单点搜索优点：速度快、易收敛、实现简单。

单点搜索缺点：容易陷入局部最优解。

多点搜索优点：模拟自然规律，直观上更可信，能更好避免局部最优解。

多点搜索缺点：速度慢、实现复杂、鲁棒性不足（不同TSP问题应该对应的自然模型可能不同）。

关于实验评分

1. 所有实验成果必须原创。允许提交半成品、失败品但绝不允许提交复制品。

本实验代码原创，基本完成功能，仍有改进空间（并行化、算法改进等）。

2. 实验程序对正确的输入有正确的输出。

程序要求tsp问题文件置于 testcases/ 中，并在调用程序时使用正确的tsp问题名，如：

```
$ ls testcases/  
a280.tsp a280.opt.tour  
$ ./local_search a280
```

3. 提交实验结果，完整齐全。

实验结果已在前文展示。

4. 源代码编写符合编码规范，可读性高。

源代码符合C/C++ google style，函数方法在文件头有说明，且部分函数是self-explainable的。

5. 源代码逻辑清晰。

逻辑已在前文说明。

6. 程序具有鲁棒性。

对于任意挑出的五个tsp问题，程序都能很好处理，具有足够的鲁棒性。

同时，对于一些可能的错误输入、文件格式问题，都在程序内进行了提示性报错。

7. 界面友好。

未单独绘制GUI界面，而是以CMD形式提供交互界面；对于输出的结果可视化提供了python脚本，无需另外处理实验结果。

5 主要参考文献

1. 赵雪梅. 遗传算法及其在TSP问题求解中的应用[J]. 四川兵工学报(11):22-27.
 2. 王银年. 遗传算法的研究与应用[D]. 江南大学.
 3. 范展, 梁国龙, 林旺生,等. 求解TSP问题的自适应邻域搜索法及其扩展[J]. 计算机工程与应用, 2008(12):75-78.
 4. Song, Chi-Hwa, Kyunghee Lee, and Won Don Lee. "Extended simulated annealing for augmented TSP and multi-salesmen TSP." *Proceedings of the International Joint Conference on Neural Networks, 2003.*. Vol. 3. IEEE, 2003.
 5. Chi-Hwa Song, Kyunghee Lee and Won Don Lee, "Extended simulated annealing for augmented TSP and multi-salesmen TSP," *Proceedings of the International Joint Conference on Neural Networks, 2003.*, Portland, OR, 2003, pp. 2340-2343 vol.3, doi: 10.1109/IJCNN.2003.1223777.
-

Karl 2020/6

[Back To Top](#)