

1 旅行商问题

平面上给定 n 个点，每两点之间的直线距离是已知的正实数，从某一个起点出发，经过其余点恰好一次，最后回到起点。要求给出一种走法，使得回路的长度最短。

benchmark 测试数据集：<http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/>

1.1 问题描述

旅行商问题是一个典型的 NP 难度问题，易于描述却难于求解。

1. 旅行商问题的类自然语言描述如下：给定 n 个城市，对这 n 个城市中的每两个城市来说，从一个城市到另一个城市所走的路程是已知的正实数（符合三角形三边关系定则），其中 n 是已知的正整数， $n \geq 3$ 。这 n 个城市的全排列共有 $n!$ 个。每一个这 n 个城市的全排列都恰好对应着一种走法：从全排列中的第一个城市走到第二个城市，……，从全排列中的第 $n-1$ 个城市走到第 n 个城市，从全排列中的第 n 个城市回到第一个城市。要求给出一个这 n 个城市的全排列 σ ，使得在 $n!$ 个全排列中，全排列 σ 对应的走法所走的路程是最短的（严格来讲，由于起点任意、顺逆时针等价，问题复杂度为 $\frac{(n-1)!}{2}$ ）。
2. 旅行商问题的形式化描述：给定一个有向完全图 $G = (V, A)$ ，其中集合 $V = v_1, \dots, v_n$ 是顶点集合，每个顶点代表一个城市， n 是顶点数 ($n \geq 3$)，集合 $E = (v_i, v_j) | v_i, v_j \in V, v_i \neq v_j$ 是有向边集合。 c_{ij} 是有向边 (v_i, v_j) 的长度， c_{ij} 是已知的正实数，其中 $(v_i, v_j) \in E$ 。集合 Σ 是顶点全排列的集合，共有 $n!$ 元素。 σ 是所有顶点的一个全排列， $\sigma = (\sigma(1), \dots, \sigma(n))$ ， $\sigma \in \Sigma$ ， $\sigma(i) \in V$ ， $1 \leq i \leq n$ 。 σ 对应着一条遍历所有顶点的回路：从顶点 $\sigma(1)$ 走到顶点 $\sigma(2)$ ，……，从顶点 $\sigma(n-1)$ 走到顶点 $\sigma(n)$ ，从顶点 $\sigma(n)$ 回到顶点 $\sigma(1)$ 。全排列 σ 所对应的回路的长度记为 $L(\sigma)$ ， $L(\sigma) = c_{\sigma(1)\sigma(2)} + \dots + c_{\sigma(n-1)\sigma(n)} + c_{\sigma(n)\sigma(1)}$ 。目标是给出所有顶点的一个全排列 σ^* ，使得 $L(\sigma^*) = \min_{\sigma \in \Sigma} (L(\sigma))$ 。每一对顶点 v_i 和 v_j 来说，都有 $c_{ij} = c_{ji}$ 成立，那么称问题是对称的（Symmetric traveling salesman problem）；否则称问题是不对称的（Asymmetric traveling salesman problem）。

1.2 研究现状

求解旅行商问题的算法可分为两类：确切算法和近似算法。

1. 确切算法保证给出最优解，但计算时间太长，仅可用于计算较小规模实例。
2. 近似算法，或许有可能在短时间内，给出相当接近最优解的近似解。非随机性近似算法包括构建式启发/贪婪算法，克里斯托菲德斯算法；随机性近似算法包括随机局部搜索、模拟退火、遗传算法、粒子群算法等。

1.3 基石算法

1~2 为非随机性近似算法，3~为随机性近似（优化）算法。

- 1) 构建式启发/贪婪算法 (**Constructive heuristics**)：主要是逐步插入点（边），最后得到一个包含所有城市的回路。例如：
 - a. 最近邻点算法：首先选择一个城市作为起点，然后用贪心法，每步均选择距离当前所在城市最近的未访问城市，最后回到起点。
 - b. 用贪心法选择符合要求的长度最短的边加入边集，直至边集构成一个哈密尔顿回路。要求如下：添加该边后，无法形成长度小于城市（顶点）数目的环，也无法形成“某城市（顶点）的度大于 2”的格局。
 - c. “王磊”基本算法：首先生成一个 3 城市回路，然后依照一定次序（可引入随机化因素），用贪心法将未访问城市插入回路，选择部分回路长度最短的动作，最后，得到包含所有城市的回路。
- 2) 克里斯托菲德斯算法 (**Christofides–Serdyukov algorithm**)：可证明 最差情况下，该近似算法所得回路长度也不会超过最途回路长度的 1.5 倍。

对于近似算法求最小值问题，设 Opt 是最优解， x 表示某算法给出的一个解，一般规定， $Opt \leq x \leq \alpha \times Opt$ ， α 记为该算法的近似比，可用于评价算法优劣。拟物仿生万用启发算法（又称元启发算法，metaheuristic），虽然有可能得出比较好的近似解，但往往不涉及在最差情况下的效率证明。基于“最小生成树”的经典非随机性近似算法有两种，分别符合 2 和 1.5 的近似比。

A. 近似比为 2 的算法:

- a. 定义: S 代表一系列边 (允许重边), $c(S)$ 代表各边权重 (长度) 之和。
- b. 定义: H_G^* 为无向多重图 G 上, 长度最短的哈密尔顿回路 (Hamiltonian Cycle), 即途中经过所有点且只经过一次。

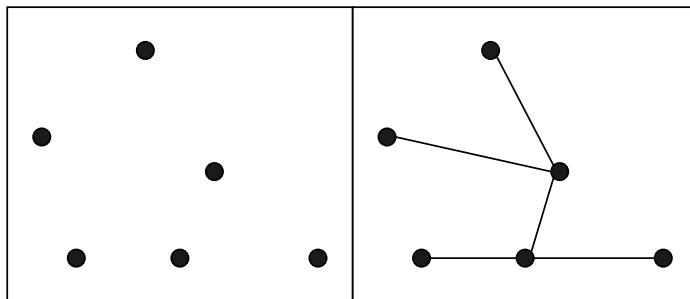


图 1.3.1 最小生成树

- c. 构造最小生成树 T , 根据 最小权生成树定义, $c(H_G^*) \geq c(H_G^* - e) \geq c(T)$ 。
- d. 按深搜次序记录回路 C , 下探一次, 回溯一次, 因此 $c(C) = 2 \times c(T)$ 。

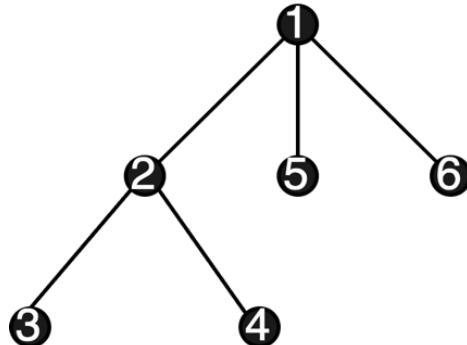


图 1.3.2 最小生成树 (实例)

例如, 1, 2, 3, 2, 4, 2, 1, 5, 1, 6, 1。

- e. 搭桥 (short-cut/bypass) 略过重复访问的点 (起点终点不删) 得到符合问题描述的新回路 C' (最后回到起点), 例如, 1, 2, 3, 4, 5, 6, 1。
- f. 证明:

由 e、三角形三边关系定则, $c(C') \leq c(C)$;

由 c, $c(H_G^*) \geq c(H_G^* - e) \geq c(T)$;

由 d, $c(C) = 2 \times c(T)$;

故 $c(C') \leq 2c(H_G^*)$; 即得证。

B. 近似比为 1.5 的算法（克里斯托菲德斯算法）：

仍基于最小生成树，想方设法减小“每边下探一次，回溯一次”带来的额外开销。

“一笔画”、“不重边”地遍历所有顶点，可以将问题转换成“欧拉回路”问题。无向图存在欧拉回路的充要条件为：该图为连通图，且所有顶点度数均为偶数。倘若，“奇度数”顶点为偶数个，那么可以通过将其两两匹配，为每一个顶点都“附赠”一个度，这样便可以满足“顶点度数均为偶数”条件。

- a. 定义： S 代表一系列边（允许重边）， $c(S)$ 代表各边权重（长度）之和。
- b. 定义： H_G^* 为无向多重图 G 上，长度最短的哈密尔顿回路（Hamiltonian Cycle），即途中经过所有点且只经过一次。
- c. 定义：假设 S 为无向多重图 G 上的导出子图，在 S 上长度最短的哈密尔顿回路记为 H_S^* 。根据三角形三边关系定则易证， $c(H_S^*) \leq c(H_G^*)$ 。
- d. 构造最小生成树 T ，根据最小权生成树定义， $c(H_G^*) \geq c(H_G^* - e) \geq c(T)$ 。
- e. 分离在 T 上度数为奇数的点，生成导出子图 S （根据握手定理，给定无向图 $G = (V, E)$ ，一条边贡献 2 度，故有 $\sum_{v \in V} \deg G(v) = 2|E|$ ；除开度数为偶数的顶点所贡献的度数，推论可知，度数为奇数顶点数有偶数个）；
- f. 构造 S 的最小权完美匹配 M ，构造多重图 $G' = T \cup M$ （此时每个顶点均为偶数度，故存在欧拉回路）；
- g. 生成 G' 的欧拉回路 C ， $c(C) = c(T) + c(M)$ ；
- h. 搭桥（short-cut/bypass）略过重复访问的点（起点终点不删）得到符合问题描述的新回路 C' （最后回到起点）。
- i. 证明：
 - i. 由 e、三角形三边关系定则， $c(C') \leq c(C)$ ；
 - ii. 由 d， $c(H_G^*) \geq c(H_G^* - e) \geq c(T)$ ；
 - iii. 由 g， $c(C) = c(T) + c(M)$ ；
 - iv. 由 f、c， $c(M) + c(M) \leq c(M1) + c(M2) = c(H_S^*) \leq c(H_G^*)$ ；
 - v. 故 $c(C') \leq c(T) + c(M) \leq c(H_G^*) + 0.5c(H_G^*)$ ；即得证。

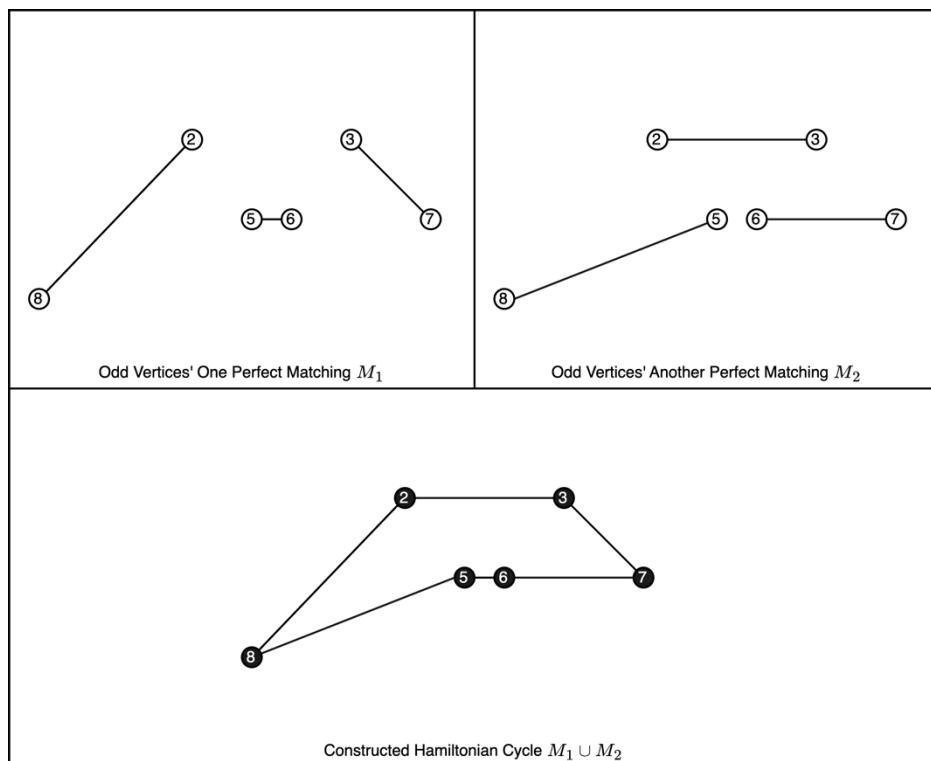


图 1.3.3 证明 iv 图例（匹配与哈密尔顿回路）

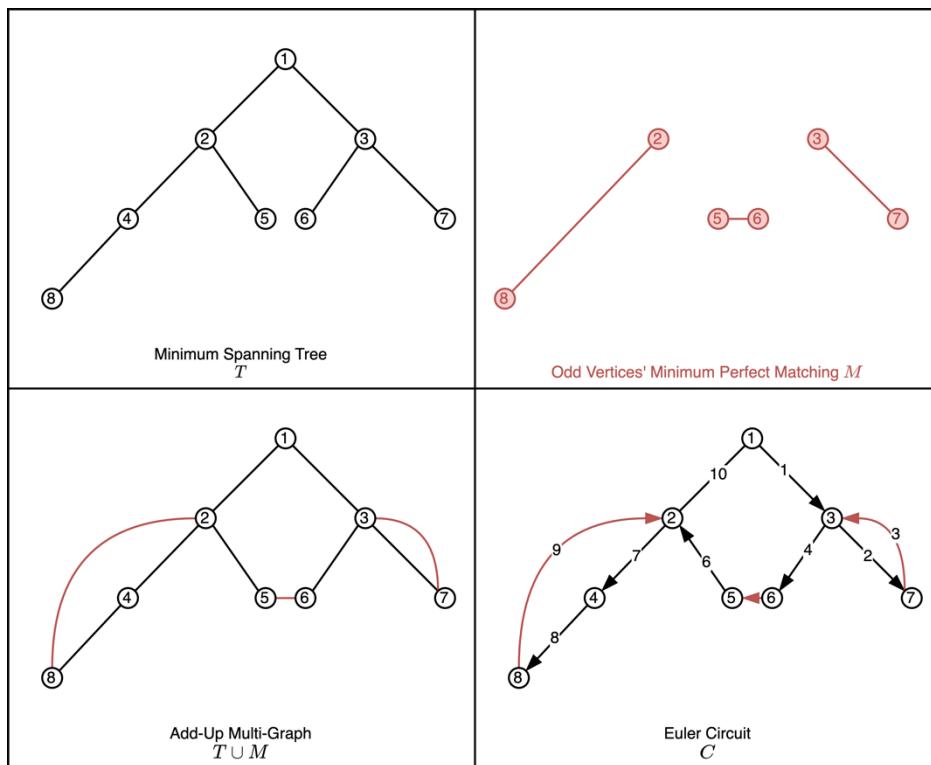


图 1.3.4 克里斯托菲德斯算法

3) 基于邻域跳坑的(元)启发算法: 解空间中的一个巡回旅行路线直接或间接对应一个全排列 $\sigma = (\sigma(1), \dots, \sigma(n))$ 。将其视作 n 维空间中的一个点, 其邻域常常定义为 σ 对换和移动后转化成的 σ' 。此外 Cores 提出的 2-opt 扰动也很经典。

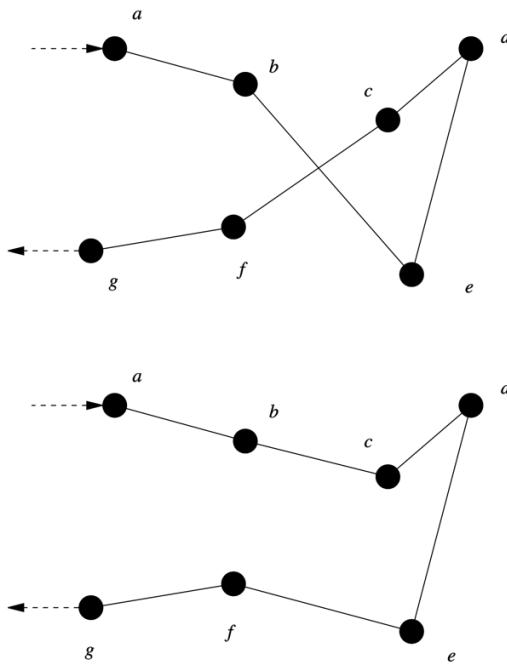


图 1.3.5 2-opt

处理 TSP 问题的常见思路之一是: 先使用非随机性启发算法获得一个相对合理的解, 再进行后处理, 使用万用启发式算法跳出局部最优, 以期待结果有所提升。这仿佛王磊老师经常说的, “如果你期末总评已经满绩了, 还要见贤思齐, 到更有希望的区域继续提高。”

不过, “大智”常常“若愚”, “峰回”方有“路转”, 谁说“错棋”不能“封神”, 谁说“绝处”不能“逢生”; 如果一个人时时刻刻都太精明, 反而让人生烦, 不利于开展工作。因此, 我认为, 更为精明的跳槽策略, 企业家的儿子先苦心志、劳筋骨, 先去稍微比董事长低一点的职位试试水, 甚至瞎猫碰上死耗子, 跌跌撞撞直接去基层工作也不是不可, 带着民间智慧总能回来继承家业, 发扬光大的。也就是说:

- 若新位置明显优于旧位置, 则跳转至新位置;
- 纵使新的位置不如旧位置, 也要以“一定概率”跳转至新位置。

事实上，人们从物理世界状态演化、自然界各种现象、千百年来生存斗争经验获得启发，以仿生拟人拟物途径设计了各种千奇百怪五花八门的算法。模拟退火（Simulated Annealing）就是其中一种，具有自然背景，且实现简单。

模拟退火的超参数包括：初始温度、终止温度、指数衰减系数、运行时间。

模拟退火的通用步骤是：

- a. 在解空间选择一个初始格局 σ ；
- b. 温度以一定系数衰减，若小于终止温度，算法停机；否则，循环执行 c；
- c. 邻域搜索：在当前格局的周围（随机点边对换等）找一个邻近格局 σ' ，根据度量指标计算优劣 $\Delta E = E(\sigma') - E(\sigma)$ （E 越小越好），跳坑策略：
 - i. 如果 $\Delta E < 0$ ，则直接令 $\sigma = \sigma'$ ；
 - ii. 如果 $\Delta E \geq 0$ ，以 $e^{-\frac{\Delta E}{T}}$ 的概率，令 $\sigma = \sigma'$ 。

上述通用步骤重置启动，若干次。其中，E 代表巡回旅行路线长度。

1.4 算法设计

步骤一：使用 1.3.1 最近邻、1.3.2 克里斯托菲德斯算法计算巡回旅行路线。

受《求解二维矩形 Packing 问题的一种优美度枚举算法》的启发，对于最近邻而言，进行如下改进：我们在生成的所有路线中，取前 m 条进行 2-opt 随机化改进（这是一种非随机型优化算法），如果超过指定时间 T，该步骤停机。

步骤二：依照模拟退火算法通用步骤，执行有限次邻域搜索，概率跳坑。

模拟退火的初始排列状态是纯随机，又或是从步骤一得到的相对合理的解轻微扰动获得，本文并没有详细研究。设计实现时，考虑到模拟退火主要是在基础上进行改进，故选择后者。邻域包括插入（将某个城市从全排列中删除再插入任意一个位置）、块插入（将任意连续 k 个城市从全排列删除再在尾部一齐插入）、点对换（将第 i 号和第 j 号交换）、以及类 2-opt 边“对换”（主要是选取两条边，进行重组，可有效消除交叉）。其中，点边对换的设计的比重较小。此外，尝试过模拟退火每次随机后，再额外进行一次 2-opt 优化（如果定义模拟退火的邻域是 2-opt 无法到达的），但往往时间难以接受故没有采用。如果超过指定时间 T，该步骤停机。

1.5 算法测试

上述算法是用 `Python` 编程实现的；测试编码同步进行，在实现基石算法过程中，对于问题有了新的认识：

- 1) 最近邻贪心策略没有想象中那么差劲，尤其是配合 2-opt 最优邻域搜索策略。

在最近邻基础上，半分钟不到就可得到右图结果，近似解相较最优解提升 10%。

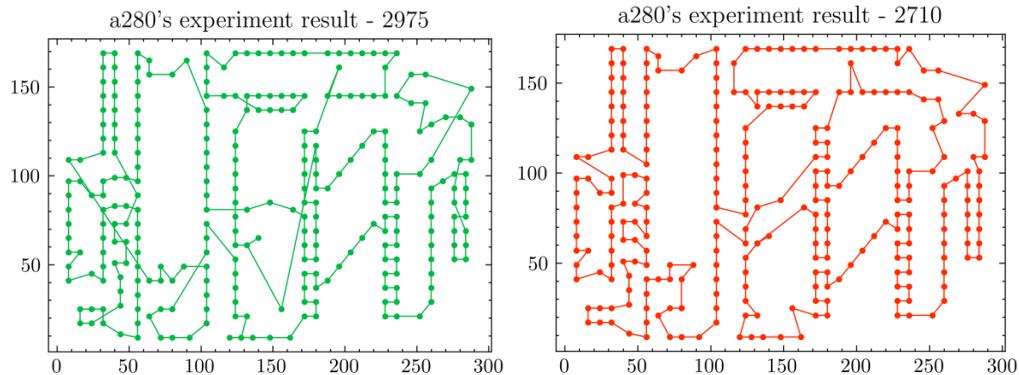


图 1.5.1 最近邻（左）与 2-opt 优化的最近邻（右）

- 2) 克里斯托菲德斯算法，饱含人类智慧。

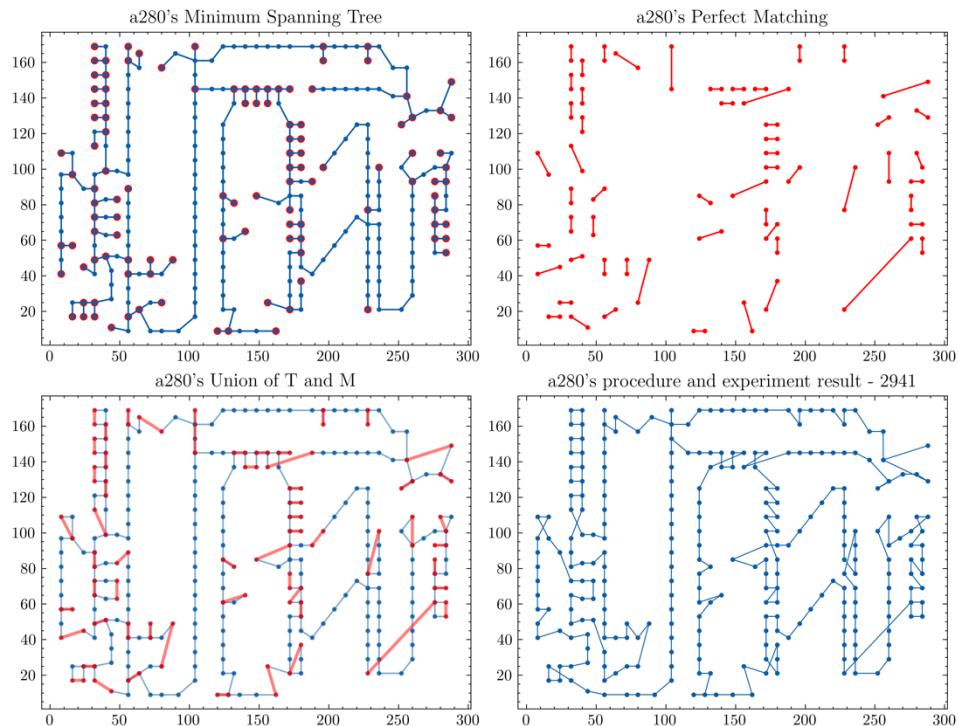


图 1.5.2 克里斯托菲德斯算法原始步骤

加以 2-opt 优化，几乎一眨眼的功夫，就可以获得不错的结果（2699）。

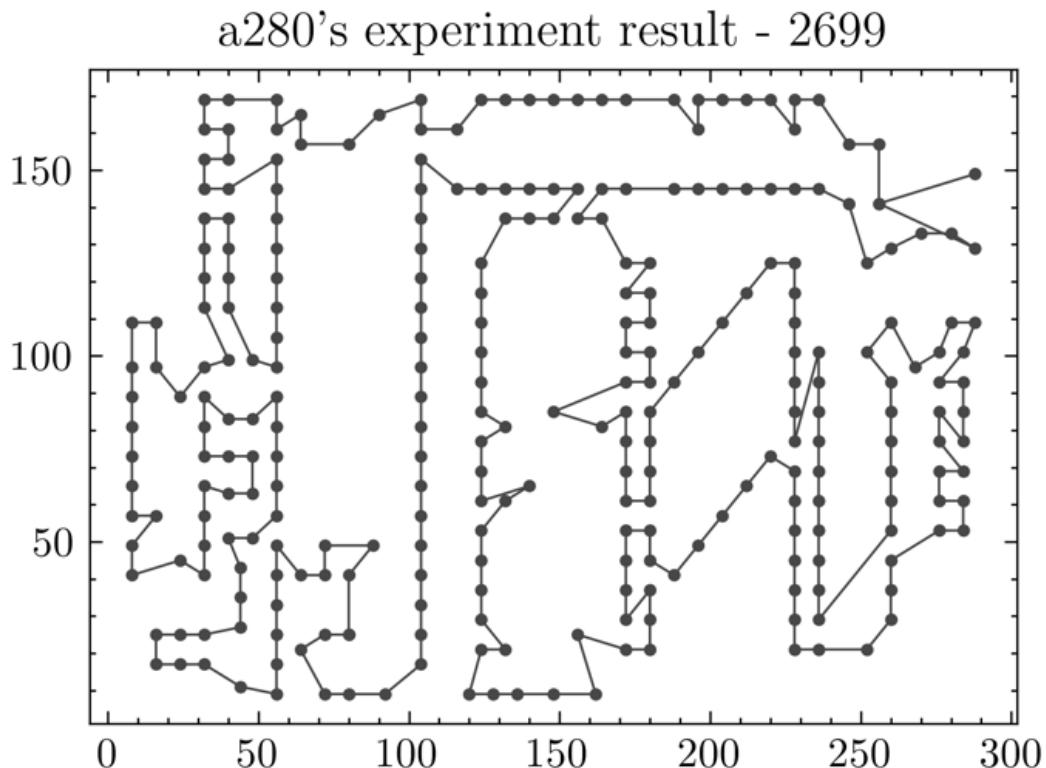


图 1.5.3 经过 2-opt 优化克里斯托菲德斯算法的解

3) 步骤二中，随机扰动与模拟退火是否如预期一般有神力？在 1 分钟内，a280 数次实验表明，模拟退火貌似无法再对结果做出任何改进。难怪有学者认为：“淬火模拟的是个一成不变因而往往不贴切的物理世界，我们应当向比晶体有更高智慧的人学习”。一种可能的原因是，初始状态选择不当，导致最后没有跳出步骤一提供的“局部”最优解。但是，步骤一的“局部”和步骤二的“局部”并不一致。需要进一步验证，超参数、邻域、运行时间也是影响因素。

```
Sat Jun 10 10:01:21 2023 - a280 -> opt-nearest-neighbor: 2710
Sat Jun 10 10:01:38 2023 2709
Sat Jun 10 10:01:39 2023 2702
Sat Jun 10 10:01:40 2023 2699
Sat Jun 10 10:01:49 2023 - a280 -> opt-christofides: 2699
Sat Jun 10 10:02:19 2023 - a280 -> opt-stimulated_annealing: 2699
```

图 1.5.4 短时间内，模拟退火在非随机启发式算法后没有提升

德国海德堡大学 (Heidelberg University) 教授 Gerhard Reinelt 维护的网站 TSPLIB (<http://comopt.if.uni-heidelberg.de/software/TSPLIB95/>) 中包含了 TSP 问题的 benchmark 数据。本文选取在 EUC_2D 类型 (两点间距离须四舍五入取整) 且城市数小于等于 1000 中全部 48 个 benchmark 测试用例。在 Apple M1 Chip 个人微型计算机上, 对每个实例计算 10 次, 表给出了统计结果。最近邻的 2-opt 优化停机时间设为 **0.5 秒** (不包括非优化部分的时间), 模拟退火停机时间设为 **5 秒 (10 次共计 50 秒)**, 初始温度 $10e5$, 终止温度 $10e-5$, 衰减系数 0.97。

代码详见: <https://github.com/DURUII/Homework-TSPLIB95>。

表 1.5.5 算法 10 次计算的统计结果

测试用例	城市数	最短回路长度	最小值	平均值	RE最小相对误差%
<i>a280</i>	280	2579	2699	2699	4.652966266
<i>berlin52</i>	52	7542	7542	7542	0
<i>bier127</i>	127	118282	127211	127211	7.548908541
<i>ch130</i>	130	6110	6269	6269	2.602291326
<i>ch150</i>	150	6528	6744	6744	3.308823529
<i>d198</i>	198	15780	16046	16046	1.685678074
<i>d493</i>	493	35002	36993	36993	5.688246386
<i>d657</i>	657	48912	51766	51766	5.834968924
<i>eil51</i>	51	426	434	434.9	1.877934272
<i>eil76</i>	76	538	562	562	4.460966543
<i>eil101</i>	101	629	659	659	4.769475358
<i>fl417</i>	417	11861	12449	12449	4.957423489
<i>gil262</i>	262	2378	2576	2576	8.326324643
<i>kroA100</i>	100	21282	21748	21748	2.18964383
<i>kroB100</i>	100	22141	23152	23152	4.566189422
<i>kroC100</i>	100	20749	21825	21825	5.185792086

<i>kroD100</i>	100	21294	22357	22368.7	4.99201653
<i>kroE100</i>	100	22068	22689	22689	2.814029364
<i>kroA150</i>	150	26524	28112	28112	5.987030614
<i>kroB150</i>	150	26130	27404	27404	4.875621891
<i>kroA200</i>	200	29368	30817	30817	4.933941705
<i>kroB200</i>	200	29437	31182	31182	5.92791385
<i>lin105</i>	105	14379	15222	15222	5.862716462
<i>lin318</i>	318	42029	44317	44317	5.443860192
<i>p654</i>	654	34643	35933	35933	3.723695985
<i>pcb442</i>	442	50778	52858	52858	4.096262161
<i>pr76</i>	76	108159	111972	111972	3.525365434
<i>pr107</i>	107	44303	44613	44613	0.699726881
<i>pr124</i>	124	59030	59246	59246	0.365915636
<i>pr136</i>	136	96772	100324	100324	3.670483198
<i>pr144</i>	144	58537	60754	60754	3.787348173
<i>pr152</i>	152	73682	75809	75809	2.886729459
<i>pr226</i>	226	80369	82801	82801	3.02604238
<i>pr264</i>	264	49135	52408	52408	6.661239442
<i>pr299</i>	299	48191	49859	49859	3.4612272
<i>pr439</i>	439	107217	110141	110141	2.727179458
<i>rat99</i>	99	1211	1280	1280	5.697770438
<i>rat195</i>	195	2323	2470	2470	6.328024107
<i>rat575</i>	575	6773	7293	7293	7.677543186
<i>rat783</i>	783	8806	9439	9439	7.188280718
<i>rd100</i>	100	7910	8241	8241	4.184576485
<i>rd400</i>	400	15281	16008	16008	4.757542046
<i>st70</i>	70	675	692	692	2.518518519

<i>ts225</i>	225	126643	129007	129007	1.866664561
<i>tsp225</i>	225	3919	4065	4065	3.725440163
<i>u159</i>	159	42080	44658	44658	6.126425856
<i>u574</i>	574	36905	39046	39046	5.801381927
<i>u724</i>	724	41910	44945	44945	7.241708423

1.6 结果反思

当不知道问题的最优解时，可仅使用单位时间内的最后结果衡量算法优劣；相反，可以等到获得相对误差低于某特定数值时的时间长短衡量。本次实验采用前者。

从实验结果来看，48个实例中，仅给出1个实例最优解，相对误差平均4.38%。

从实验日志来看，最小值和平均值几乎一模一样，总体没有发挥模拟退火随机型近似算法的优势（除标框线外，10次结果非常稳定），徒增计算时间（可以说，步骤一和步骤二的结合是相对失败的设计）。可以说，贡献最大的还是非随机性近似算法。

精妙的通用元启发式算法和人类智慧设计的启发式算法一直相互推动着彼此的边界。从实验结果来看，期待；当然，设计邻域结构、精调超参数、延长停机时间或许一个可能结果改进方向。初始状态的选择与超参数配合，或许也是具有一定影响力的因素之一。此外，符合直觉的邻域，应在全排列对应的路线的长度、图形上差异不能太大，点对换、插入是否高效，也值得探讨，应当分析现有解与最优解的细微差距。

1.7 参考资料

- a) 王磊-求解旅行商问题的拟物拟人算法研究计算结果, 2023 年 6 月
- b) 王磊.求解工件车间调度问题的一种高效近似算法.2006.华中科技大学,PhD dissertation.
- c) 黄文奇, 许如初. 近世计算理论导引:NP 难度问题的背景、前景及其求解算法研究. 科学出版社, 2004.
- d) 王磊, 尹爱华."求解二维矩形 Packing 问题的一种优美度枚举算法." 中国科学:信息科学 45.09(2015):1127-1140.
- e) MIT6.046, 2015 Approximation Algorithms: Traveling Salesman Problem
- f) <https://youtu.be/GiDsJlBOVoA>
- g) https://en.wikipedia.org/wiki/Travelling_salesman_problem
- h) <https://networkx.org/documentation/stable/reference/introduction.html>
- i) <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>

1.8 附录一

以下是实验图示，与表 1.5.5 一致。

