将 2-OPT、3-OPT 和 4-OPT 与 K-SWAP-KICK 相结合
旅行推销员问题的扰动

安德鲁斯·布拉津斯卡斯、阿方萨斯·米塞维丘斯

考纳斯理工大学多媒体工程系,Studentu St. 50401, 416a,考纳斯,立陶宛,andrius.blazinskas@ktu.lt,
alfonsas.misevicius@ktu.lt

抽象的。旅行商问题 (TSP) 是一个著名的 NP 难题,通常使用各种启发式方法来解决。流行的启发式类别之一是k-opt 本地搜索。尽管这些启发式方法非常简单,但与迭代本地搜索 (ILS) 框架中的其他技术相结合,它们显示出有希望的前景

结果。在本文中,我们建议将 2-opt、3-opt 和 4-opt 局部搜索算法与所谓的k-swap-kick扰动结合起来,这是众所周知的双桥（随机 4-

选择)移动。为了减少 2-opt、3-opt 和 4-opt 实现的运行时间,我们应用了一些增强功能,例如候选列表（基于kd树）、搜索切割、贪婪启动、两级树数据结构等。我们提供了使用 TSPLIB 实例实现的算法的实验结果。

关键词:旅行商问题、2-opt、3-opt、4-opt、k-swap-kick。

1 简介

旅行商问题（TSP)[6]可以表述如下。给定一组城市和它们之间的距离,任务是找到仅访问每个城市一次的最短旅行（汉密尔顿循环）。两个城市之间的距离不依赖于方向的问题称为对称 TSP (STSP) 和非对称 TSP (ATSP)。 TSP 被认为是 NP 难问题。 STSP 可能的旅行总数等于(n－1)!/2（其中n是问题规模 - 城市（节点)的数量)[7]。对于这样的搜索空间,在可接受的时间内找到好的旅行是一项相对复杂的任务,特别是当n很大时。

解决 TSP 的方法主要有两种:精确法和启发式。对于较大的 n 来说,精确的方法太耗时,因此通常使用启发式方法。 Lin-Kernighan是 STSP 的主要启发法之一[9]。它的有效实现是存在的（例如,参见Concorde* [2]、Helsgaun 代码** (LKH) [7]）。

在本文中,我们仅考虑 STSP。在实验中,我们使用k-opt 启发式算法,它与更强大的Lin-Kernighan启发式算法密切相关,因为两者都使用k-opt 子移动[8]。在第 2 节中,我们描述了如何优化 2-opt、3-opt 和 4-opt 启发式算法以使其运行得更快的几种方法。在第 3 节中， k-swap-kick

描述了扰动,并在第 4 节中介绍了将 2-opt、3-opt 和 4-opt 启发法与k-swap-kick扰动相结合的实验结果。我们还在第 5 节中提供了结论和未来的研究思路。

2 提高2-opt、3-opt和4-opt速度

虽然 2-opt 算法很简单,其简单形式涉及重复断开两条边并以其他（成本降低)方式重新连接它们（参见图 3b）,直到无法进行积极的增益 2-opt 移动,但在实践中,有更多的细节需要考虑,不同的作者有不同的实现方式。

例如,一些作者检查所有可能的翻转并仅执行最好的翻转[11],而其他作者则简单地进行第一个发现的正增益翻转（有关变体,请参阅STSP*** 的 DIMACS 实施挑战）。 [3] 中还指出了 2-opt 启发式的其他几种可能的实现选择。此类选择通常会显着影响成本和时间。大多数考虑因素也适用于 3-opt 和 4-opt,但需要评估更多的案例和具体情况。

朴素 2-opt、3-opt 和 4-opt 的时间复杂度为O(n ^{2}) ,在 ^{3}) 和O(n ^{4}) ,但是通过使用各种加速技术可以大大改善这一点。这些技术通常会稍微牺牲真正的局部最优性（最终的旅行并不是真正的k 最优）,但在某些情况下,将它们组合起来可以达到接近O(n) 的复杂度[1]。

我们对快速 2-opt、3-opt 和 4-opt 修改（2-opt-f、3-opt-f 和 4-opt-f)。我们将在随后的小节中简要概述它们。

---

* Concorde TSP 求解器， http://www.tsp.gatech.edu/concorde/

** Helsgaun 的 Lin-Kernighan 实现， http://www.akira.ruc.dk/~keld/research/LKH/

*** STSP 的 DIMACS 实施挑战， http://www2.research.att.com/~dsj/chtsp/

2.1基于Kd树的候选列表

最近节点候选列表(CL)广泛用于改进k-opt 启发法 [1]。使用Kd树进行CL识别比朴素CL方法要有效得多。特别是,朴素CL可以在O(n

^{2}logn)时间 [11],而kd树 — O(Kn + nlogn) [4]（其中K是维度数,在我们的例子中
总是K = 2)。一旦生成,naive CL不需要任何额外的处理来提取列表（它只是将它们包含在数组中）,而每次需要列表时都需要搜索kd树。当然,这可以通过使用缓存来解决,但是对于大型 TSP 实例,缓存需要大量内存,这些内存会以O(mn) 的形式增长（其中m – CL大小）,并且对于存储kd树,这个数字大约为O(n)（高效的Concorde kd树实现使用 52n字节的内存）。在kd树中搜索（自下而上技术)几乎可以在O(1) 时间内完成[4]。因此,高效的kd树实现实际上在所有情况下都击败了朴素的CL 。使用kd树进行CL 的好处是显而易见的,例如,在我们的测试机上（请参阅第 4 节了解规范）,以朴素方式构建CL 的pla85900 TSP 问题实例大约需要 15 分钟,而通过搜索和缓存构建kd树需要大约 15 分钟。 CL – 大约 3 秒（两种情况下 CL大小都等于 80）。必须注意的是， kd 树的使用将 TSP 求解器限制为欧几里德实例。

2.2 如果b1 = a2则停止搜索

如果a1最近的候选者b1已经是当前游览中a1的下一个节点（即a2 = b1） ,我们建议使用搜索切割,而不是流行的不看位想法 [1]（参见图 2 的伪代码））。 [3] 中的2-opt固定半径搜索使用了类似的切割。当b2 = c1和c2 = d1 时,我们也对 3-opt 和 4-opt 使用此类切割。我们的实验表明,这些改进大大缩短了运行时间,而不会显着增加成本。

2.3 对于 3-opt 和 4-opt 分别考虑仅交换 3 条和 4 条边的情况

当在游览中打破k 个边时,有(k－1)!2k－1 种方法重新连接它（包括初始游览)以形成有效的游览 [6],通常不会考虑所有这些情况 [15]。在我们的实现中,我们仅考虑所有k 个边都是新边的情况。例如,在 3-opt 中断开 3 条边,总共有 8 种重新连接的情况（图 1） ,但其中只有 4 种（e,f,g,h)实际上引入了所有新边（所有其他边,除了最初的边,是简单的 2-opt 动作）。类似地,对于 4-opt,断开 4 个边,我们有 48 种重新连接的方法,但只有 25 个案例提供所有 4 个新边。这不会影响 2-opt 启发式,因为只有一种方法可以重新连接到有效的游览。

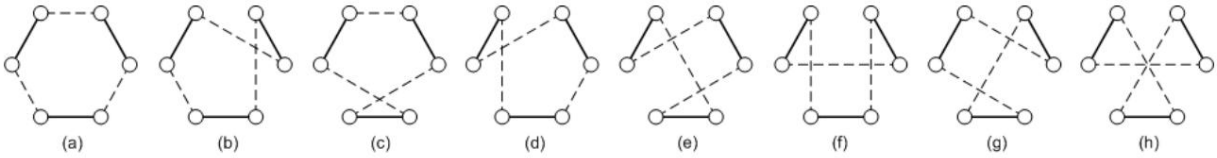

图 1. 所有可能的 3-opt 重新连接情况

2.4 级联 3-opt 和 4-opt我们注意到级联k-opt

启发式方法减少了整体运行时间。我们以这种方式利用这个功能:对于 3-opt-f,我们另外使用 2-opt-f 预处理旅行,对于 4-opt-f 预处理则使用 2-opt-f 完成。
opt-f 和 3-opt-f 按此顺序。每次优化都是从随机节点开始。

2.5 贪婪多片段初始旅游多片段启发式（简称贪婪启发式)是

[3]中提出的一种有效的旅游构建启发式。众所周知,它特别适合作为 k-opt 启发式的预处理器,并且比传统的最近邻启发式更有效[1, 3]。在 k-opt 中使用这种贪婪的起始游览,可以观察到游览质量和运行时间都有所改善。在我们的实现中,出于效率原因,多片段游览构建由前面描述的kd树支持。此启发式的复杂度为O(nlogn) [3]。应该注意的是,对于特定问题仅使用一次这种启发式方法是有意义的,因为不存在像最近邻居那样的不同起点,因此最终的游览始终是相同的。

2.6两级树数据结构

两级树 [5] 是 TSP [1, 7] 中最流行的旅游表示数据结构之一。它为反转（翻转)子巡程提供了 O(√) 复杂度 这是 TSP 中最常见的操作之一。

对于n ⩽ 106 的问题很有用 ，否则， Splay树似乎是更好的选择 [5]。应该注意的是,两层树的使用仅在优化的k-opt 启发式中才有帮助,其中翻转操作所需的时间占主导地位。在我们的朴素 2-opt 实验中,仅使用两级树更改数组会导致更高的运行时间:遍历两级树中的节点成本更高,因为它需要访问父节点并执行额外的检查（尽管仍然是O(1) ,常数实际上更大） ,而

进行2-opt移动后，快速 4-opt 加速的想法是在索引的问题上考虑所以2改进 相邻邻居的改进 startNode) begin localyOptimal := FALSE; while (不是localyOptimal) begin 开始localyOptimal := TRUE TRUE; a1 :=起始节点;重复

```
a2 := next( (a1 );
for (∀b1    cl(a1 )) begin if if (a2 ==
        b1 ) break; // 加速b b2 := next(b1 ); for for
        (∀c1    cl( b1 ))
        begin if (b2 == c1 ) break; // 加速
                c2 := next(c1 ); for (∀d1    cl(c1 )) begin if
                (c2 == d1 ) break; //
                加速d2 := next(d1 );
                        findBestReconnection(a1 , a2 , b1 , b2 , c1 ,
                        if (存在更好的重新连
                        接) begin reconnect(a1 , a2 , b1 , b2 , c1 , c2 , d1 , d2 ,    c2 、 d1 、 d2 、成本) ;
                        tour); localyOptimal := false;

                        结尾
                结尾
        结束
结束
a1 :=下一个(a1 1 );
直到(a1 == startNode);
结束
结束fast4Opt
```

图 2. 。用于快速 4-opt 实现的通用伪代码

3 K-swap-kick 扰动最简单的非顺序移动（k-opt 移动）

[8 8]，首先在 [9] 中提到，后来被认为是有效的变异算子[10]。这是众所周知的举动，它的随机版本是它的随机版本，它使用的是最强大的启发法之段的串联，Iterated Lin-Kernighan [11]。这个简单的概念是[14]中引入的 k-swap-kick，考虑改进大小为k ≤ 5的大小移动的 k-opt 移动，因此我们提出了我们应用的 k-swap-kicks对于我们的快速 2-opt 、3-opt 和 4-opt 启发式，我们只使用2-opt移动(k = 2) 的所有相同模式。图 3 提供了k = 3、4 4 时的图形2-opt移动和k-swap-kicks图示。

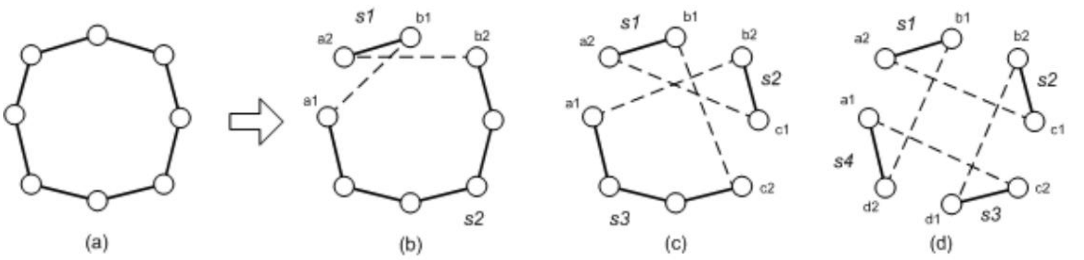

图 3. 2-opt 移动和k-swap-kick 踢扰动：a)初始游览，b)2-选择移动，c)3-选择移动 (k=3) 和 d)双桥或4-选择移动（k = 4）

所有k-swap-kicks都是k-opt移动的特殊情况，已经执行，条件

我们将k-swap-kick 视为随机移动，不考虑移动期间对旅行成本的影响并且通常可能导致更糟糕的旅行。这与 k-opt 启发式中使用的移动相反，k-opt 启发式通常仅执行游览，仅执行游览成本减少的移动。

随机双桥移动先前也与3-opt启发式相结合，形成迭代3-opt [1]。在某些情况下，它甚至产生了比Lin-Kernighan 更好的结果，并且比Iterated Lin-Kernighan稍差。我们继续扩展这个想法。我们进行了一个实验，结合了我们的 2-opt-f、3-opt-f 和

4-opt-f 实现,具有随机 2-opt 移动和随机k-swap-kicks (k = 3...15)。图 4b 提供了如何实现此组合的伪代码。 z* :=无穷大;

```
重复
    生成贪婪随机解t;
    从t开始进行局部搜索,找到局部最优tl ； if (z(tl ) < z*)开始
        z* := z(tl );
        t* := tl ;
    结尾
直到满足停止标准;
```

(A)

```
生成贪心解t;
从t开始进行局部搜索,找到局部最优tl ； z* := z(tl );重复
    使用随机k-swap-kick形成tlp扰乱tl巡演;
    从tlp开始进行局部搜索,找到局部最优tlpl ； if (z(tlpl) < z*)开始
        z* := z(tlpl);
        t* := tl := tlpl;
    结尾
直到满足停止标准;
```

(二)

图 4. GRASP (a) 和k-swap-kick扰动 (b) 框架的通用伪代码（t* –找到的最佳解决方案， z* = z(t*),其中z –目标函数）

应该指出的是， k-swap-kick ILS 框架与其他众所周知的程序 - GRASP 非常相似（参见图 4a,另请参见[12]）。它们之间的主要区别在于，GRASP 在每次迭代中都从完全不同的初始巡演开始,而不是扰乱现有的巡演。

4 实验结果

所有算法均完全使用 Java (JDK 6) 实现。两级树、 kd树和多片段启发式的实现基于Concorde C 代码。实验在配备 Intel Core i7-860 处理器和 8GB RAM 的机器上进行。

下面提供的表格和图表总结了我们快速 2-opt、3-opt 和 4-opt 实现（2-opt-f、3-opt-f 和 4-opt-f)的结果。为了强调扰动的优势,我们还在不使用扰动的情况下进行了两项额外的实验:一项是随机启动,另一项是贪婪启动。在这些实验中,我们使用了类似于GRASP程序,但它并不完全符合基本的GRASP方案（见图4a）:随机启动实验没有使用贪婪随机启发式作为初始解决方案,并且贪婪启动实验没有随机化（因为我们使用原始的多片段启发式）。后面情况中唯一的随机化是快速k-opt 局部搜索中的不同起点（节点） 。所有其他连续列提供随机 2-opt (k = 2) 和具有固定k 的k-swap-kick扰动的结果。在这两个框架中,迭代次数均设置为 1000（停止标准）。使用以下公式估计与最佳游览长度的偏差（最佳游览长度可以在 TSPLIB [13] 中找到）： %][其中z* – 获得的目标函数的最佳值， zopt表示可证明的最佳目标函数值。所有实验重复10次并计算平均值。

d = z 100选择 选择 / ,

实验的问题数据取自 TSP 库 – TSPLIB [13]。

表 1. 4-opt-f 的实验结果:巡演质量（10 次运行与最优值的平均偏差,%） 4-opt-f 随机

问题名称			4-opt-f 带有贪婪启动和k-swap-kicks							
			k							最佳找到(k) 0
	开始	贪婪的开始	2	3	4	5	6	10	15	
eil51	0.00	1.17	0.02	0.07	0.05	0.02	0.12	0.00	0.02	（全部） 0
st70	0.00	3.26	0.00	0.00	0.00	0.00	0.00	0.00	0.00	（全
kroE100	0.03	0.00	0.00	0.00	0.03	0.02	0.00	0.00	0.02	部） 0
kroB150	0.19	2.11	0.11	0.02	0.04	0.04	0.02	0.03	0.03	（全
TS225	0.21	1.35	0.02	0.00	0.00	0.00	0.00	0.00	0.00	部） 0
gil262	1.73	2.11	0.40	0.46	0.13	0.14	0.23	0.20	0.27	（全部）
a280	1.81	1.98	0.64	0.48	0.02	0.05	0.20	0.03	0.04	0 （全
lin318	1.66	2.09	0.54	0.49	0.36	0.42	0.35	0.37	0.54	部） 0 (4,
rd400	2.42	2.10	0.57	0.33	0.24	0.33	0.29	0.51	0.54	6) 0 （全部）
	2.98	2.50	1.18	0.53	0.62	0.63	0.69	0.79	1.06	0 (2, 4)
u574rat783	3.96	2.58	1.43	1.00	1.11	1.13	1.19	1.55	1.78	0.07 (4, 6)
虚拟机1084	3.05	1.92	0.87	0.47	0.47	0.60	0.50	0.77	0.98	0.13 (3) 0.48
PCB1173	4.92	3.78	2.10	1.76	1.45	1.77	1.82	2.15	2.54	(3) 0.15 (3,
VM1748	3.68	2.89	1.24	0.81	0.85	0.96	1.08	1.23	1.52	4) 1.17 (4)
D2103	6.33	2.46	1.20	0.91	0.82	1.22	1.22	1.53	1.80	0.50 (4)
FNL4461	4.96	2.57	2.21	2.19	2.17	2.44	2.47	2.76	2.74	0.27 (4)
RL5934	5.71	2.95	2.06	1.82	1.81	2.21	2.08	2.69	3.07	1.92 (2) 1.47 (3)

问题名称	4-选择-f		4-opt-f 带有贪婪启动和k-swap-kicks							
	随机开始	贪心开始	k							最佳发现(k)
							6	10	15	
pla7397	4.85	2.94	2 2.01	3 1.74	4 1.62	5 2.04	1.86	2.28	2.71	1.34 (4)
rl11849	6.30	2.85	2.52	2.58	2.52	2.82	2.88	3.14	3.13	2.23 (4)
美国13509	5.11	3.07	2.61	2.72	2.59	2.98	2.93	3.21	3.19	2.47 (2)
BRD14051	5.30	3.03	2.77	2.89	2.94	3.07	3.09	3.06	3.27	2.66 (2)
D15112	5.25	2.92	2.71	2.82	2.87	2.94	3.01	3.10	3.16	2.58 (2)
D18512	5.37	2.84	2.68	2.82	2.84	2.94	2.94	3.04	3.07	2.54 (2)

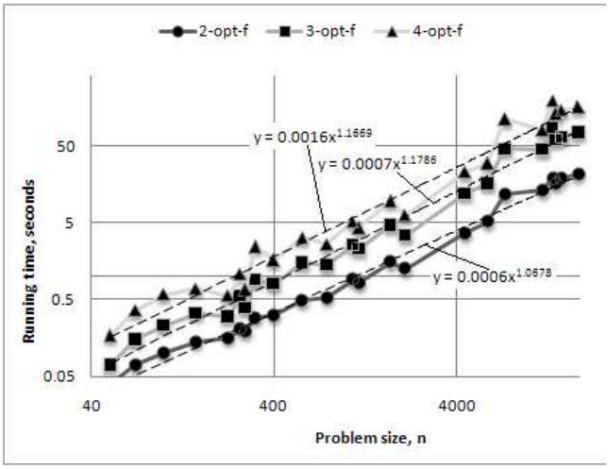

图 5. 运行时间对问题大小的依赖性（k-swap-kick大小 = 4,对数刻度）

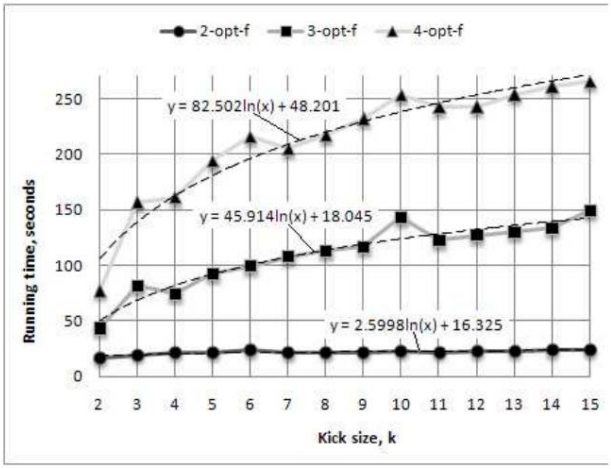

图 6. 运行时间对k-swap-kick大小的依赖性（问题 d18512）

表 2. 4-opt-f 的实验结果:运行时间（10 次运行的平均值,秒） 4-opt-f 随机

问题			4-opt-f 带有贪婪启动和k-swap-kicks						
姓名		贪心开始	k						
	开始	0.33					6	10	15
51号	0.51		2 0.07	3 0.16	4 0.17	5 0.2	0.21	0.29	0.32
ST70	1.04	0.36	0.19	0.3	0.36	0.35	0.41	0.53	0.67
克罗E100	1.98	0.86	0.27	0.46	0.58	0.6	0.76	0.93	1.26
克罗B150	2.79	1.53	0.37	0.58	0.69	0.71	0.87	1.08	1.45
TS225	3.09	0.73	0.32	0.56	0.56	0.66	0.78	1.05	1.39
吉尔262	5.74	1.73	0.57	1.01	1.06	1.18	1.47	1.8	2.54
A280	5.15	1.73	0.37	0.65	0.67	0.73	0.98	1.33	1.82
林318	10.46	4.52	1.3	2.26	2.45	2.71	3.17	4.63	5.58
RD400	9.52	3.37	0.87	1.65	1.63	1.89	2.28	3.35	4.07
u574	17.71	9.28	1.74	3.12	3.19	3.55	4.35	6.59	7.74
大鼠783	18.21	6.32	1.31	2.49	2.61	2.97	3.59 5.46 6.56 7.07 10.72		
vm1084	31.18	16.08	2.76	5.26	5.2	5.98	12.57 6.39 9.8 12.19 13.78		
PCB1173	34.78	13.31	2.38	4.5	4.31	5.32	20.37 21.04		
vm1748	61.29	33.66	5.51	10.04	9.65	11.34			
d2103	65.52	16.4	3.58	6.35	6.29	7.24	8.69	12.12	14.63
民族解放阵线4461	175.72	59.5	11.44	22.77	22.74	26.38	32.31	38.78 44.42	
rl5934	238.15	88.53	16.11	31.66	29.63	37.66	42.11	56.06 67.38	
pla7397	964.15	377.13	50.73	118.11	111.6	143.28 170.82 222.82 278.1 101.43 119.85			
rl11849 美	716.1	241.73	39.93	86.49	80.5	149.98 172.37 222.92 269.8 318.19 356.73			
国13509	1197.66	619.04	89.98	186.62	195.33	143.25 176.97 204.42 228.78			
brd14051	894.35	347.39	62.32	132.13	130.7				
d15112	1018.65	474.92	70.16	144.26	144.54	163.4	195.59 233.7 264.27		
d18512	1159.27	510.4	77.3	156.8	160.93	193.64 215.89 253.28 265.59			

5 结论和未来研究

在本文中,我们提出了 2-opt、3-opt 和 4-opt 启发式方法的有效实现来解决旅行商问题。我们还关注将这些启发法与随机 2-opt 移动和k-swap-kick扰动相结合。我们还提供了这些启发式变体的实验结果。为了显示基于扰动的方法的效率,此外还使用了简化的类似 GRASP 的程序来比较启发式方法。

从实验结果可以看出,问题规模越大,扰动的影响（以成本而言)就越小,而简单地增加扰动规模只会增加成本和运行时间。总体扰动算法处理时间对问题大小的依赖性约为O(n 1.2),而对冲击大小的依赖性是对数的（见图 5 和图 6）。由于多片段缺乏随机性

启发式,这种贪婪的 k-opt 启发式开始,不使用扰动,对于较小的问题规模不是那么有效。这可以在n < 400的实例中看到,其中简单的非改进随机开始给出更好的结果,但是超出该限制,贪婪开始会超越。

尽管正如预期的那样,尺寸 4 的k-swap-kick （即双桥移动)被证明是最有效的扰动之一,但通常也可以通过使用简单的随机 2-opt 移动和 3- 找到总体最佳解决方案。
选择移动扰动（参见表 1,最后一列）。这为进一步的实验提出了想法,其中双桥移动可以以某种合理的方式与 2-opt 和 3-opt 移动相结合。另一个有趣的扩展是使用象限最近邻 CL [6] 而不是典型的CL。然而,这对于本文（第 2.2 节)中描述的切割优化可能无效,因此可能需要进一步改进的方法。

参考

[1] Aarts E.,Lenstra JK组合优化中的局部搜索。约翰·威利父子公司， 1997 年。

[2] Applegate DL、Bixby RE、Chvátal V.、Cook WJ旅行商问题:计算研究,普林斯顿大学出版社， 2007 年。

[3] Bentley JL几何旅行商问题的快速算法， ORSA J. Comput.， 1992 年,卷。 4-4、387-411。

[4]用于半动态点集的Bentley JL Kd树。第六届 ACM 计算几何年度研讨会论文集， 1990 年,187-197。

[5] Fredman ML、Johnson DS、Mcgeoch LA、Ostheimer G。旅行推销员的数据结构。算法杂志， 1995,卷。 18-3、432-479。

[6] Gutin G.,Punnen AP旅行商问题及其变体， Kluwer， 2002 年。

[7] Helsgaun K. Lin-Kernighan 旅行推销员启发式的有效实施。欧洲杂志
运筹学， 2000 年,第 126 卷， 106-130。

[8] Helsgaun K. Lin-Kernighan TSP 启发式的通用 k-opt 子移动。数学规划计算，
2009 年,119-163。

[9] Lin S., Kernighan BW旅行商问题的有效启发式算法。行动调查，
1973 年,卷。 21、498-516。

[10] Martin O.、Otto SW、Felten EW用于旅行商问题的大步马尔可夫链。复杂的
系统， 1991。

[11] Misevi ius A.、Ostreika A.、 imaitis A.、 ilevi ius V.改进旅行商问题的本地搜索。信息技术与控制,考纳斯,技术， 2007 年,卷。 36、没有。 2、
187-195。

[12] Pitsoulis LS 和 Resende MGC贪婪随机自适应搜索过程。 PM Pardalos 和 MGC
Resende,编辑,《应用优化手册》,牛津大学出版社， 2002 年,178-183。

[13] Reinelt G. TSPLIB 旅行商问题库。 ORSA 计算杂志， 1991 年,第 3-4 卷,376-385。
通过互联网访问:<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>。]

[14] Richter D.、Goldengorin B.、Jager G.、Molitor P.提高对称 TSP 的 Helsgaun Lin-Kernighan 启发式的效率。第四届网络组合和算法方面会议论文集,2007 年。

[15] Sierksma G. Hamiltonicity 和旅行商问题的 3-Opt 程序。应用数学， 1994 年,卷。 22-3, 351–358。