文章编号: 1001-0920(2008)07-0762-05

一种求解 TSP 问题的 ACO &SS 算法设计

张晓霞, 唐立新
(东北大学 信息科学与工程学院, 沈阳 110004)

摘 要: 提出一种求解旅行商(TSP)问题的新型分散搜索算法. 将蚁群算法(ACO)的构解方法引入分散搜索(SS)算法,在搜索过程中既考虑解的质量,又考虑解的分散性. 采用一种将蚁群算法的信息素更新技术与分散搜索的组合机制相结合的新型子集组合成新解的构解机制,同时采用动态更新参考集与临界准则策略来加快收敛速度. 实验结果表明,该算法优于其他现有的方法,获得了较好的结果.

关键词: 旅行商; 蚁群算法; 分散搜索
中图分类号: TP18 **文献标识码**: A

An ACO &SS algorithm for traveling salesman problem

ZHANG Xiao-xia, TANG Li-xin
(College of Information Science and Engineering, Northeastern University, Shenyang 110004, China. Correspondent: ZHANG Xiao-xia, E-mail: syzhangxx@163.com)

Abstract: A scatter search algorithm is presented. The solution construction mechanism of ant colony optimization (ACO) is introduced into scatter search (SS). Both solution quality and diversification are considered. A new mechanism of the subset combination method is applied simultaneity, which hybridizes the mechanism of the pheromone trail updating with the combination mechanism of scatter search to generate new solutions. The dynamic updating strategy and the criterion of threshold are adopted to accelerate the convergence. The experimental results show that the method is more efficient and competitive compared with the existing methods in terms of solution quality.

Key words: Traveling salesman problem; Ant colony optimization; Scatter search

1 引 言

旅行商问题(TSP)是一个经典组合优化问题,其模型在计算机网络、集成电路布线等优化问题中有着广泛的应用. 求解算法可分为精确型和启发式两大类,其中精确型算法仅能解决一些小规模问题. 另一类是近几年发展起来的智能优化算法,如禁忌搜索法[1]、模拟退火算法[2]、遗传算法及蚁群算法[3]等,亦被称为超启发式算法. 计算机技术的迅速发展,为超启发式算法提供了快速的计算能力,也为求解大规模复杂 TSP 问题提供了方便.

本文针对蚁群算法易出现早熟、停滞现象,设计一种新型分散搜索算法. 该算法是将蚁群算法(ACO)的构解方法引入分散搜索(SS)算法中,简称 ACO &SS. ACO &SS 算法在搜索过程中,既考虑解的质量,又考虑解的分散性. 在分散搜索算法中,子集组合产生新解的过程比较复杂,耗时也较多. 为加快新解构解速度,ACO &SS 算法采用一种新型子集组合成新解的构解机制. 该机制是将蚁群算法的信息素更新技术与分散搜索的组合机制相结合而生成新的解. 信息素更新策略是影响算法收敛速度的关键因素之一,除采用信息素局部和全局更新外,还采用了组合解公共边临时信息素更新机制,同时采用动态更新参考集和临界准则来加快收敛速度.

2 算法设计

2.1 蚁群算法的基本原理

蚁群优化(ACO)算法是由意大利学者 Dorigo 等提出的一种进化算法[4]. 其基本原理是:模拟蚂蚁在搜索食物源时释放一种信息激素,蚁群通过信息

收稿日期: 2007-08-01; **修回日期**: 2007-10-23.
基金项目: 国家自然科学基金项目(60674084); 国家杰出青年科学基金项目(70425003); 国家 863 计划项目(2006AA04Z174).
作者简介: 张晓霞(1967—),女,辽宁鞍山人,博士生,从事物流优化、智能优化算法等研究; 唐立新(1966—),男,黑龙江绥化人,教授,博士生导师,从事物流优化、生产调度等研究.

素交流来选择最短路径并达到搜索食物的目的. 蚂蚁选择城市时采用确定与随机选择方式[4],随机选择方式可增强搜索的多样性,避免过早地陷于搜索停滞.

信息素随时间变化需不断更新,通常有局部和全局更新方式[4]. 信息素局部更新是为避免因频繁选择一条路径而陷入局部最优,每只蚂蚁选择一个城市后,就更新该边上的信息素. 信息素全局更新使搜索向最短路径边进行,从而获得全局最好解. 当所有蚂蚁走完全部城市后,仅对最佳路径上的信息素更新. 蚁群算法虽采用确定性与随机选择相结合的策略,但迭代到一定代数后,较短路径因信息素不断增强,使其成为最优路径的可能性加大,易陷入局部最优.

2.2　分散搜索算法

分散搜索算法(SS)是一种基于种群的进化算法,是 Glover 于 1977 提出的,近几年在组合最优化问题上取得很大进展. Glover[5]给出了 SS 算法详细的框架. SS 原理如下:首先产生初始种群,在初始种群中选择一些高质量解和比较分散的解作为参考集;将这些解通过组合机制产生新解集合,通过局域搜索算法对新解进行改进,获得局部最优解;更新参考集,以进行新的搜索.

SS 算法与 GA 算法最明显的区别是 GA 算法初始种群数目较多(100 个组成),随机抽样选取而产生新的组合;而 SS 算法参考集中解的数目相对较少(20 个左右),用系统方法选择两个或多个解进行组合产生新解. 因此,SS 算法既考虑了解的质量,同时考虑了解的分散性.

2.3　ACO &SS 分散搜索算法

SS 算法主要由初始解的生成、参考集合的生成、组合生成新解、解的改进以及参考集更新 5 部分组成. 图 1 为 SS 算法的结构示意图,白色与黑色小圆分别代表初始解和改进解. ACO &SS 算法主要是将 ACO 构解机制嵌入 SS 算法框架结构. 由于 SS 保持搜索在大范围内进行,增强了 ACO 算法跳出局部最优的能力. 为描述方便,给出下面定义:

定义 1　参考集 RefSet 是一些可行解的集合,由 RefSet_1 集合与 RefSet_2 集合组成. RefSet_1 中有 b_1 个高质量的解,RefSet_2 由 b_2 个分散性好的解组成.

定义 2　假设 $s^*(v_i^*)$ 为参考集合 RefSet 中的一个解,$s_j(v_i)$ 为非参考集合 non-RefSet 中的一个解,设 $\mathrm{NUM}(s_j, s^*)$ 代表这两个解相同边的数目,表示这两个解的分散性. $\mathrm{NUM}(s_j, s^*)$ 值越大,这两个解越相似;反之,$\mathrm{NUM}(s_j, s^*)$ 值越小,这两个解的分散性越大.

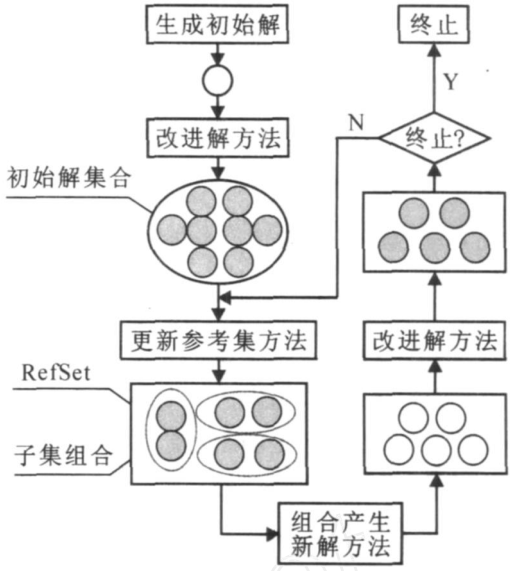

图 1　分散搜索算法的结构示意图

定义 3　对于非参考集中的任意一个解 s_j,$\mathrm{NUM}_{\max}(s_j)$ 表示该解在 RefSet 中的分散度,定义为

$$\mathrm{NUM}_{\max}(s_j) = \max_{s^* \in \mathrm{RefSet}} \{\mathrm{NUM}(s_j, s^*)\},$$

$$s_j \in \text{non-RefSet}.$$

$\mathrm{NUM}_{\max}(s_j)$ 值最小的解就是非参考集合中分散性最好的解.

定义 4　设 s^{new} 是当前迭代产生的一个新解,线路 s^{best} 代表当前参考集合的最好解,$f(s^{\mathrm{new}})$ 和 $f(s^{\mathrm{best}})$ 分别为解 s^{new} 和解 s^{best} 的目标函数值. 设

$$\delta = \delta_0 (f(s^{\mathrm{new}}) - f(s^{\mathrm{best}}))/f(s^{\mathrm{best}}),$$

$$\delta^* = \delta_0^* (1 - N/\mathrm{CN}_{\mathrm{num}}).$$

其中:δ_0 和 δ_0^* 为初始参数,N 和 $\mathrm{CN}_{\mathrm{num}}$ 分别为当前迭代次数和最大迭代代数. 则当 $\delta \leq \delta^*$ 时,称为满足临界准则.

只有满足临界准则,才能调用 2-opt 和 3-opt 来改进当前解. 采用临界准则,可限制搜索不必要的空间,加快搜索速度. 在初始时 δ^* 值较大,满足临界准则的解较多,搜索空间也较大. 随迭代次数的增加,δ^* 值变小,搜索空间也较小,最后收敛到较为优化的解.

分散搜索算法每部分根据具体情况都可作出灵活的设计,下面重点介绍 ACO &SS 算法中的子集组合生成新解和参考集合更新两部分,它们是 ACO &SS 算法的核心,也是本文的创新部分.

2.3.1　子集组合生成新解

子集组合新解的方法很多,如边的重组合方法(ER)[6]和线路连接(PR)[7]等,这些方法都很复杂. 本文通过更新组合子集公共弧的信息素浓度,用蚁群构解方法快速构成新解,新解具有组合子集中解

的公用特点. 参考集中的解采用两种组合策略:一种策略是两个解都来自 RefSet_1 集合;另一种策略是两个解分别来自 RefSet_1 集合和 RefSet_2 集合.

设 s_k 和 s_l 为任意两个解的弧集合,A^{kl} 是这两个解的公共弧集合,$A^{kl}=\{(i,j),i,j\in s_k\cap s_l\}$. 计算公共弧数目,如果满足

$$\mathrm{NUM}(s_k,s_l)\geq\varepsilon \tag{1}$$

的规定值 ε,则按

$$\tau_{ij}=(1-\rho)\tau_{ij}+\rho L_0,\ i,j\in A^{kl} \tag{2}$$

更新公共弧信息素. 式(2) 中 L_0 和 ρ 为参数. 如果公用边数目太少,则所产生的新解质量不是很高. 为加快搜索速度,需放弃建立新的解,继续求下个组合产生的新解,直到产生 N_{num} 新解为止. 为使每个组合产生的解不影响其他组合产生的解,每个组合公共边的信息素均用临时变量存储.

2.3.2 参考集合的更新

参考集合更新分为静态和动态更新策略[8]. 静态更新策略是每次所有新解生成后,根据一些准则确定这些解是否加入到参考集中. 动态更新则不必等待所有新解都产生,而是每产生一个新解就动态检查该解是否更新参考集中的解. 本文采用动态更新方法. 设 s_j 为一个新解,s_i 是 RefSet_1 中的一个解,如果 $f(s_j)<\max\limits_{i\in\mathrm{RefSet}_1}\{f(s_i)\}$,则新解 s_j 就更新 RefSet_1 集合中目标函数值最大的解. 如果新解的 $\mathrm{NUM}_{\max}(s_j)$ 值比较小,而且比 RefSet_2 集合中的 $\mathrm{NUM}_{\max}(s_i)$ 最大值还小,则将该解添加到 RefSet_2 中. 这种动态更新策略能随时快速获得较好的解,从而加快算法的收敛速度.

2.3.3 ACO &SS 算法步骤

Step1: 产生初始解,初始化参数 q_0,β,ρ,$|\mathrm{RefSet}|$,τ_0,$N_{\max}$,$\mathrm{CN}_{\max}$,k,count. 将 m 只蚂蚁随机分配到 n 个城市,出发点放置到禁忌表 M_k 中. 用 ACO 产生不同初始解,并对这些解进行改进,生成初始解集合 P.

Step2: 建立 RefSet,从 P 中选择 b_1 个较好的解加入 RefSet_1 中,b_2 个较分散的解加入 RefSet_2 中,$|\mathrm{RefSet}|=b_1+b_2$.

Step3: 计算每个组合子集公共弧集合,如果满足式(1) 的约束,则按式(2) 更新这些公共弧的信息素. 调用 ACO 程序,生成新解 s_j.

Step4: 计算目标函数 $f(s_j)$,$f(s^{\mathrm{best}})$,δ和δ^* 的值,当满足临界准则条件 $\delta\leq\delta^*$ 时,调用 2-opt 或 3-opt 对解进行改进,$L_{\mathrm{best}}=f(s^{\mathrm{best}})$;否则,转 Step5.

Step5: 计算 $\mathrm{NUM}_{\max}(s_j)$,当 $f(s_j)<\max\limits_{i\in\mathrm{RefSet}_1}\{f(s_i)\}$ 时,用 s_j 更新参考集中的 RefSet_1 集合 s_i 解,$b=\mathrm{true}$;如果 $\mathrm{NUM}_{\max}(s_j)<\max\limits_{i\in\mathrm{RefSet}_2}\{\mathrm{NUM}_{\max}(s_i)\}$,则用 s_j 更新参考集中的 RefSet_2 集合 s_i 解,$b=\mathrm{true}$. 如果 $b=\mathrm{true}$,则转 Step6;否则,调用 ACO 程序,产生一个新解,转 Step4.

Step6: 计算和更新参考集合的当前最好解 L_{current},$b=\mathrm{false}$,$\mathrm{count}=\mathrm{count}+1$,当 $L_{\mathrm{current}}<L_{\mathrm{best}}$ 时,$L_{\mathrm{best}}=L_{\mathrm{current}}$,转 Step7.

Step7: 重复 Step3～Step7,直到 $\mathrm{count}>N_{\max}$,转 Step8.

Step8: $k=k+1$,若 $k\leq\mathrm{NC}_{\max}$,则重复 Step3～Step8;否则,转 Step9.

Step9: 输出最终结果.

3 实例求解与分析

本文算法采用 C++ 编程,在 512 RAM 的 IBM 机器上运行. 测试实例采用国际通用的 TSPLIB 中典型例子. 蚁群算法中存在一些参数,这些参数值直接或间接影响算法性能. 蚁群算法通常采用的参数是一个固定值,而固定值易陷入局部最优或搜索时间长. 为避免这些不足,ACO &SS 算法采用参数动态自动调节方式. 对于 TSP 较小规模的实例,已证明是最优解,所以本文以 eil51 为测试例子.

为测试 β 值对解的影响,测试 β 取不同值,经 500 次迭代后求其最好解. 图 2 显示 $\beta=4$ 比 $\beta=1$ 收敛快,也就是说,在同样或较少的迭代次数时能获得更好的解. β 取相同值时,分散算法收敛速度比蚁群算法快,而且收敛值比蚁群算法小. 同时测试迭代 500 次时不同 β 值所获得最好解的数目,实验测定 $\beta\in$

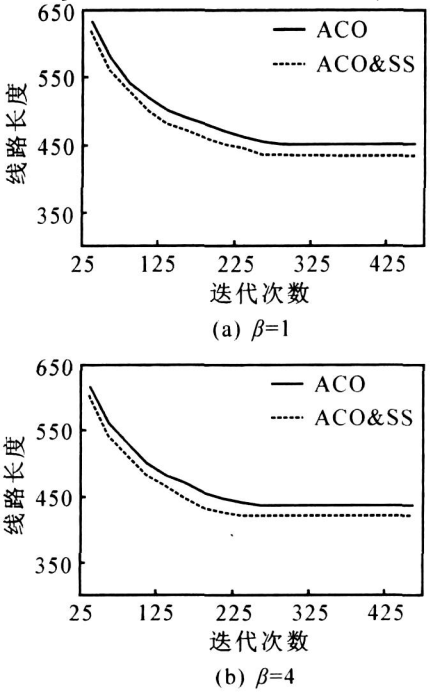

图 2 在 eil151 实例上参数 β 的收敛曲线

{3,4,5,6} 能获得较好的结果.

为了确定参数 q_0 值,每个 q_0 值测试 20 次,记录每次找到最好解的平均时间. 实验表明, q_0 对最终结果影响不大, 只是消耗时间有些差别, 当 $q_0 \in [0.65, 0.90]$ 时,消耗时间较少. 所以 ACO &SS 分散算法采用 $q_0 \in [0.65, 0.90]$. 用同样方法测试 $\rho \in [0.04, 0.06]$,参数默认值为 $q_0 = 0.90$, $\beta = 6$, $\rho = 0.04$. 实验观测到,有些参数在迭代若干代后,一些边信息素浓度高,使蚂蚁选择其他路径的可能性较小. 为避免陷入局部最优,当算法连续几次选择同样线路时,这些参数是可以动态调整的.

为测试 RefSet 大小对算法性能的影响,选取几种规模实例测试,测试每个实例最好解的平均值和平均运行时间. 实验表明,参考集合较大时,搜索空间较大,解的质量也较好,但耗时稍多;而参考集较小时,因搜索空间小,解的质量相对差一些. 同时注意到, b_1 与 b_2 相等或相近时效果好一些. 所以测试采用参考集 $|\text{RefSet}| = 10$, $b_1 = b_2 = 5$,其他参数设置为 $\tau_0 = 0.01$,迭代次数 $CN_{max} = 2\,000$.

表 1 给出了 ACO &SS 算法与 ACO 算法运行结果比较. 实验结果表明,ACO &SS 分散算法运行效果好,平均改进值为 2.26 %. 但是,由于分散算法组合解的生成和参考集更新等消耗了一些时间,运行时间比蚁群算法要长.

表 2 给出了 kroa100 实例在不同计算机平台、操作系统和编程语言下的运行时间[9]. ACO &SS 算法获当前最好解仅需 1.66 s,比其他算法要快.

表 1　蚁群算法与 ACO &SS 分散算法寻优效果对比

实例	ACO		ACO &SS		改进值 / %
	最短路程	时间 / s	最短路程	时间 / s	
eil51	435	1.1	426	2.9	2.11
st70	691	2.3	675	3.8	2.37
eil76	555	2.8	538	4.3	3.16
rd100	8104	3.5	7930	7.6	2.19
eil101	646	3.7	631	7.9	2.38
lin105	14756	4.1	14541	8.2	1.48
pr107	45391	4.5	44810	8.5	1.30
pr124	59992	5.6	59050	10.6	1.60
bier127	122038	5.8	119850	13.1	1.82
pr136	99296	8.4	96800	14.4	2.57
pr152	75532	9.6	73780	14.8	2.37
rat195	2379	15.4	2330	26.7	2.10
kroa200	30525	17.5	29398	38.7	3.83
lin318	43460	30.6	42473	67.3	2.32

表 2　kroa100 实例运行时间对比

算　法	计算机平台	编程语言	时间 / s
Meta-RaPS (Depuy et al,2005)	AMD 900 MHz Athlon PC	C ++	50
Neural net (Modares et al,1999)	Sun Sparcstation 10	C	55
SA (Voudouris et al,1999)	DEC Alpha 3000	N/A	37
TA (Voudouris et al,1999)	DEC Alpha 3000	N/A	21
ACO &SS	IBM 512 RAM	C ++	1.66

表 3　旅行商问题的启发式算法与最优解的偏差

实例	城市数	目前最优解	平均计算偏差 / %						
			KniesG	KniesL	SA	Budinich	ESom	ACO	ACO &SS
eil51	51	426	2.86	2.86	2.33	3.1	2.1	2.11	0
st70	70	675	2.33	1.51	2.14	1.7	2.09	2.37	0
eil76	76	538	5.48	4.98	5.54	5.32	3.89	3.15	0
rd100	100	7910	2.62	2.09	3.26	3.16	1.96	2.45	0.25
eil101	101	629	5.63	4.66	5.74	5.24	3.43	2.70	0.32
lin105	105	14383	1.29	1.98	1.87	1.71	0.25	2.59	1.09
pr107	107	44303	0.42	0.73	1.54	1.32	1.48	2.45	1.14
pr124	124	59030	0.49	0.08	1.26	1.62	0.67	1.62	0.04
bier127	127	118282	3.08	2.76	3.52	3.61	1.70	3.17	1.32
pr136	136	96772	5.15	4.53	4.90	5.20	4.31	2.61	0.03
pr152	152	73682	1.29	0.97	2.64	2.04	0.89	2.51	0.13
rat195	195	2323	11.92	12.24	13.29	11.48	7.13	2.41	0.30
kroa200	200	29368	6.57	5.72	5.61	6.13	2.91	3.93	0.10
lin318	318	42029	NC	NC	7.56	8.19	4.11	3.40	1.05
总时间 / s			NC	NC	3715	353	369	147	265

表 3 给出了 ACO &SS 分散算法与其他启发式算法的运行结果. 表 3 中 KniesG, KniesL, SA, Budinich 和 ESom 算法的数据均来源于文献[10], ACO 和 ACO &SS 分别为蚁群算法和分散算法,NC 表示没有相关数据报道,总时间表示运行所有测试实例需要的时间[11]. 平均计算偏差表示算法的最好解与该实例当前的最好解之间的平均偏差,偏差为 0 表示算法能找到当前最好解. 实验表明,ACO &SS 算法结果优于其他现有较好的方法,获得了较好的结果,平均偏差为 0.41 %. 蚁群算法平均偏差比其他算法小, 但个别实例平均偏差较大. ACO 和 ACO &SS 算法运行总时间比其他算法少,但 ACO 运行速度比 ACO &SS 算法快.

根据实验仿真结果,可得出以下结论:

1) ACO &SS 算法能提高全局搜索能力. 从算法的最终结果看,ACO &SS算法在解决 TSP问题时具有一定的优越性, 与最优解的平均偏差只有 0.41 %.

2) 参考集的大小对算法性能有直接影响. 参考集太大,搜索时间长;参考集太小,不能达到分散的目的. 因此,选择合适的参考集是分散算法的关键因素之一.

3) 随着问题规模的增大,邻域规模也增大,搜索消耗的时间会越来越长.

4 结 论

本文以求解 TSP 问题为例,提出一种新型 ACO &SS 混合分散搜索算法. 该算法将蚁群的构解机制与分散搜索算法组合机制相结合. 采用这种分散算法,可避免陷入局部最优,提高算法全局寻优能力. 实验结果表明了所提出算法的实用性和有效性,为 TSP 问题的研究提供了一种新的有效方法,同时分散搜索算法也为解决组合最优化问题提供了新的思路.

参考文献(References)

[1] Liu G, He Y, Fang Y, et al. A novel adaptive search strategy of intensification and diversification in tabu search [C]. Proc of Neural Networks and Signal Processing. Nanjing, 2003: 428-431.

[2] Budinich M. A self-organizing neural network for the traveling salesman problem that is competitive with simulated annealing[J]. Neural Comput, 1996, 8(2): 416-424.

[3] Dorigo M, Maniezzo V, Colorni A. The ant system: Optimization by a colony of cooperating agents [J]. IEEE Trans on Systems, Man and Cybernetics — Part B, 1996, 26(1): 1-13.

[4] Dorigo M, Gambardella L M. Ant colonies for the traveling salesman problem[J]. BioSystems, 1997, 43(2): 73-81.

[5] Glover F. A template for scatter search and path relinking[J]. Artificial Evolution, Lecture Notes in Computer Science, 1998, 1363(1):13-54.

[6] Liu Y H. A hybrid scatter search for the probabilistic traveling salesman problem [J]. Computers & Operations Research, 2007, 34(10): 2949-2963.

[7] Ho S C, Gendreau M. Path relinking for the vehicle routing problem[J]. J of Heuristics, 2006, 12(1/2): 55-72.

[8] Cotta C. Scatter search with path relinking for phylogenetic inference[J]. European J of Operational Research, 2006, 169: 520-532.

[9] DePuy G W, Moraga R J, Whitehouse G E. Meta-RaPS: A simple and effective approach for solving the traveling salesman problem [J]. Transportation Research Part E, 2005, 41(2): 115-130.

[10] Siqueira P H, Steiner M T A, Scheer S. A new approach to solve the traveling salesman problem[J]. Neurocomputing, 2007, 70(4-6): 1013-1021.

[11] Leunga K S, Jinb H D, Xu Z B. An expanding self-organizing neural network for the traveling salesman problem[J]. Neurocomputing, 2004, 62: 267-292.