1 引言

社会性动物的群集活动往往能产生惊人的自组织行为，如个体行为显得简单、盲目的蚂蚁组成蚁群以后能够发现从蚁巢到食物源的最短路径。蚂蚁通过一种叫做信息素的化学物质交换信息以找到好的路径,信息素在短的路径上会增加，在长的路径上会挥发。蚁群在觅食过程中总能找到蚁巢和食物源之间的最短路径，蚁群算法模仿的是蚂蚁在巢穴和食物源之间发现最短距离的自然行为。

受这种现象启发，1991 年意大利学者 M. Dorigo, V. Maniezzo 和 A. Colorni 通过模拟蚁群觅食行为提出了一种基于种群的模拟进化算法—蚁群算法。第一个蚁群优化算法是蚂蚁系统(AS)[1]，蚁群算法的进化计算过程，实际上是计算机通过程序不断的迭代过程，由于蚁群算法在构造解的过程中，利用了随机选择策略，导致进化速度变慢，影响算法的性能。因此，研究蚁群算法具有重要意义。AS 算法是第一个蚁群算法的模型，被称为基本蚁群算法。AS 算法首先被用来求解 TSP 问题，并取得了巨大成功。

针对 AS 算法的不足之处，许多学者对其进行了深入的研究，提出了一些改进的蚁群算法，如基于对信息素矩阵进行局部和全局更新的蚁群系统(Ant Colony System，简称 ACS) [2]；带局部信息素更新机制和全局信息素更新中精英策略的蚂蚁系统(Ant System with elitist strategy，简称 AS_{elite})[1]；Sttttzle 和 Hoss[82]还提出了最大一最小蚂蚁系统(MAX-MIN Ant System，简称 MMAS)，该算法的主要特点就是为信息素设置上下限来避免算法过早出现停滞现象；Bullnheimer 等[4]提出了基于排序的蚂蚁系统(Rank-based Version ofAnt System，简称 AS_{rank})，该算法在完成一次迭代后，将蚂蚁所经路径的长度按从小到大的顺序排列，并根据路径长度赋予不同的权重，路径较短的权重加大；鉴于蚂蚁系统搜索效率低和质量差的缺点，0. Cordon 提出了最优一最差蚂蚁系统(Best-Worst Ant System，简称 BWAS)，该算法的主要思想就是对最优解进行更大限度的增强，而对最差解进行削弱，使得属于最优路径边与属于最差路径边之间的信息素量差异进一步增大，从而使蚂蚁的搜索行为更集中于最优解的附近。

国内对蚁群算法的研究直到上世纪末才拉开序幕，目前国内学者对蚁群算法的研究主要是集中在算法的改进和应用上。冀俊忠等人以 TSP 问题出发，给出了一种新的多粒度的问题描述模型，显示了快速求解大规模 TSP 问题的能力[5]。

夏亚梅等人在蚁群算法基础上进行研究和改进，提出一种多信息素动态更新的算法 MPDACO 算法，可以适应服务组合优化过程中发生的服务无效以及服务中 QoS 变化情况[6]。唐禹研究了蚁群算法收敛速度慢和全局搜索能力差的问题，提出一种基于目标启发策略、参数自适应调整策略等策略的改进蚁群算法[7]。吕嘉伟针对蚁群算法在大规模 PPI 网络中进行功能模块检测所暴露的时间性能方面的不足,提出了一种基于多粒度描述和蚁群优化的快速求解算法，设计了融合功能和结构信息的粒度划分方法，粗粒度的蚁群寻优，解的还原与优化 3 个阶段的求解过程[8]。万正宜、彭玉旭[9]针对传统量子蚁群算法在求解 TSP 时容易陷入局部最优以及收敛速度较慢，提出了一种求解旅行商问题的改进型量子蚁群算法(IQACA)。该算法设计了一种新信息素挥发因子的自适应动态更新策略，对信息素进行动态更新；并采用一种新的量子旋转门对量子概率幅值的收敛趋势进行改变。康与云和唐敦兵[10]为解决矩阵算是求解元功能链设计方案过程中缺乏优化工具的问题，提出了一种矩阵算式结合蚁群算法的优化方法。侯梦婷[11]等人从均衡能量消耗、提高数据传输的可靠性出发，提出采用角度因子的蚁群优化多路径路由算法（ACOMP），以实现节能、可靠的路由，均衡整个网络的能量消耗。

本文首先分析蚁群算法的研究现状，从 TSP 问题描述入手，通过对 TSP 问题描述从而引出蚁群算法描述，分析蚁群算法的原理。并通过 ch130 数据集对蚁群算法进行测试，根据多次实验分析蚁群算法各参数对于蚁群算法性能的影响，另外，与遗传算法在 ch130 城市数据上进行比较，加深蚁群算法的理解。

2 蚁群算法

2.1 TSP 问题描述

为了说明蚂蚁系统的模型，首先引入 TSP 问题。一般来说，旅行商问题(Traveling Salesman Problem，简称 TSP 问题)可以描述如下：给定一个城市集合 $V=\{\nu_1,\nu_2,\ldots,\nu_n\}$，求遍历 V 中所有城市各一次且最后回到出发城市（求解约束）的一个最短旅行的连通图 $g\in G(V,A)$，其中 $G(V，A)$为所有，满足上述求解约束的联通集合，$A=\{a_{ij}=(\nu_1,\nu_2)\big|0<i,j\le n,i\ne j\}$是城市间两两相连的一组边。如果把任意一条边的距离记为 $d_{ij}=|a_{ij}|$，则一个解图的数学形式可表示为 $g=\arg\min_{g'\in G(V,A)}\sum a_{ij}\in g' d_{ij}$。

TSP 问题的求解是一个典型的 NP 难问题，由于该问题的求解模型可广泛应用于多种复杂优化问题，故该问题的求解算法一直以来是国内外学者们关注的一个研究热点.然而，在求解 TSP 问题的过程中，当城市规模 n 大到一定程度后就会面临“组合爆炸”问题，这时传统的一些启发式搜索就会陷入困境。于是人们结合物理学、生物遗传学、仿生学等领域的研究成果，相继提出了多种元启发式的 TSP 问题的求解方法，如模拟退火(simulated annealing) ，禁忌搜索(tabu search)、遗传演化(geneticevolutionary)、蚁群优化(ant colony optimization)等算法。其中，遗传算法(GA)是今年来迅速发展起来的一种全新的随机搜索与优化算法，它是由 J.Holland 教授提出的一类借鉴生物界自然选择和自然遗传机制的随机化搜索算法[12]。遗传算法在理论和应用上发展迅速，效果显著，是一种模拟进化的计算理论。蚁群优化算法是 20 世纪 90 年代初才被提出的一种新的元启发搜索算法，其实现机理是通过模拟人工蚂蚁群体在觅食过程中所体现出的智能行为来完成对 TSP 问题的求解。由于蚁群算法具有良好的鲁棒性、正反馈、分布式及并行计算等特点，所以受到学者们的广泛关注，蚁群算法的理论框架也日渐成熟。

2.2 蚂蚁算法描述

为模拟实际蚂蚁的行为，首先引入如下几号：

m ——蚁群中蚂蚁数量；

$b_i(t)$ ——t 时刻位于城市 i 的蚂蚁数，$m=\sum_{i=1}^{n} b_i \ \ t$ ；

d_{ij}——两城市 i 和 j 之间的距离；

η_{ij}——边(i,j)的能见度，反映由城市移动到城市的启发程度，这个量在蚂蚁的运动中不改变（本文设为两城市间的距离）；

$\tau_{ij} \ \ t$ ——t 时刻边(i,j)的信息素量；

$\triangle\tau_{ij}$——本次迭代中边 ij 上的信息素增量；

$P_{ij}^{k}(t)$——在 t 时刻蚂蚁 k 从城市 i 转移到城市 j 的概率；

每只蚂蚁都具有以下特性：

(1)完成一次循环后，蚂蚁在其访问过的每条边上留下相应的信息素；

(2)蚂蚁依据概率选择下一个将要访问的城市，这个概率是两城市间距离及连接两个城市的边上信息量的函数；

(3)为了满足问题的约束条件，在完成一次循环之前，不允许蚂蚁选择已经访问过的城市，该过程由蚂蚁的路径表来控制。设 $table_k$ 为蚂蚁 k 的禁忌表，则蚂蚁在经过城市 i 以后，就将城市 i 加入到自己的禁忌表 $table_k$ 中，表示下次不能再选择城市 i。用 $table_k$ (s)表示禁忌表中第 s 个元素，也即蚂蚁走过的第 s 个城市。

于是，蚁群算法可以描述如下：在算法的初始时刻，将 m 只蚂蚁分别随机放到 n 个城市中，同时，将每只蚂蚁的路径表中与该蚂蚁所在的城市相应城市号设为 0，其余城市设为 1，表示蚂蚁可经过此城市。初始时刻，各路径上的信息素量相等，设 $\tau_{ij}(0)=C$（C 为较小的常数，本文设为 1.0）。然后每只蚂蚁按照各条路径上的信息素量和启发式信息独立的选择下一座城市，蚂蚁系统所用的状态转移规则称为随机比例规则。在 t 时刻，蚂蚁 k 在城市 i 选择城市 j 的转移概率 $P_{ij}^k(t)$ 为：

$$P_{ij}^k(t)=\begin{cases}\dfrac{\tau_{ij}^{\alpha}(t)\,\eta_{ij}^{\beta}(t)}{\sum_{s\in allowed_k}\tau_{ij}^{\alpha}(t)\,\eta_{ij}^{\beta}(t)}, & j\in allowed_k\\ 0, & otherwise\end{cases} \tag{1}$$

其中，η_{ij} 在本文中取城市 i 和城市 j 之间距离的倒数。α 和 β 分别表示信息素和启发式因子的相对重要度，取 α=1，β=2。$allowed_k$ 表示蚂蚁下一步允许选择的城市，即蚂蚁路径表中值为 1 的城市。当所有城市都被赋值为 0 时，蚂蚁 k 便完成了一次旅行，此时 k 所经过的路径则是 TSP 问题的一个可行解。

当所有蚂蚁完成一次周游后，个路径上的信息素会进行更新：

$$\tau_{ij}(t+n)=\rho.\tau_{ij}(t)+\Delta\tau_{ij} \tag{2}$$

$$\Delta\tau_{ij}=\sum_{k=1}^{m}\Delta\tau_{ij}^k \tag{3}$$

其中 ρ (0< ρ <1)表示信息残留系数，而 1- ρ 表示信息素的挥发系数。另外 $\Delta\tau_{ij}^k$ 的计算方法按照 M.Dorigo 曾给出的 ant-cycle system 模型进行计算，利用蚁群的整体信息，即走完一个循环以后才进行残留信息量的全局调整：

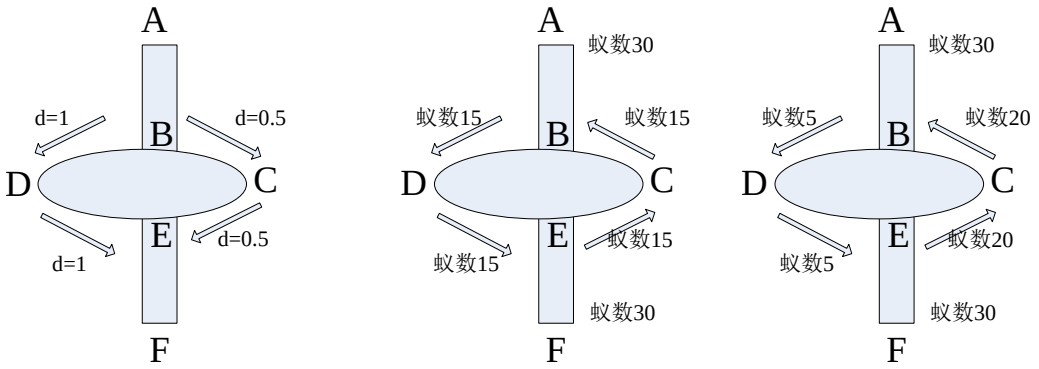
$$\triangle\tau_{ij}^{k}=\begin{cases}\dfrac{Q}{L_k}, \text{若蚂蚁}k\text{在本次循环中经过边}ij\\ 0, \text{否则}\end{cases} \tag{4}$$

本文中为了保持信息素的及时更新，采取二次更新方法，即蚂蚁每走一步（每经过一个城市）都通过公式(4)更新残留信息量，当蚂蚁循环一周后再全局调整依次残留信息量，即通过信息素矩阵进行局部和全局的更新。

3 蚁群算法实现求解 TSP 问题

3.1 蚁群算法原理分析

蚂蚁这种社会性动物，虽然个体行为及其简单，但是由这些简单个体所组成的群体却表现出极其复杂的行为特征。这是因为蚂蚁在寻找食物时，能在其经过的路径上释放一种叫做信息素的物质，使得一定范围内的其他蚂蚁能够感觉到这种物质，且倾向于朝该物质强度高的方向移动。这种觅食行为是蚁群一个重要而有趣的行为。据昆虫学家的观察和研究发现，生物世界中的蚂蚁有能力在没有任何可见提示下找出从蚁穴到食物源的最短路径，并且能随环境的变化而变化的搜索新的路径，产生新的选择。



(a) (b) (c)

图 1 蚁群路径搜索实例

蚁群发现最短路径的原理和机制如图 1 所示，假设 B 和 D 之间、D 和 E 之间以及 B 和 E（通过 C）的距离为 1，C 位于 B 和 E 的中间（图 a 所示），现在考虑等间隔离散时间点（t=0,1,2…）的蚁群搜索情况。为简单起见，假设每单位时间有 30 只蚂蚁从 F 到 E，另 30 只蚂蚁从 A 到 B，其行走速度都为 1（一个单位时间所走距离为 l），在行走时，一只蚂蚁可在时刻 t 留下浓度为 1 的信息素。

设信息素在区间(t+1，t+2)的重点(t+1.5)时刻瞬间完全会发。在 t=0 时刻无任何信息素，但分别有 30 只蚂蚁在 B、30 只蚂蚁在 E 等待出发。它们选择走哪一条路径是完全随机的，因此在两个节点上蚁群可各自一分为二，走两个方向。但在 t=1 时刻，从 F 到 E 的 30 只蚂蚁在通向 C 的路径上（如图(b)）发现一条浓度为 30 的信息素，这是由 15 只走向 EC 的路径的蚂蚁所留下的信息素和从 B 经 C 到达 E 的蚂蚁留下的信息素之和；而在通向 D 的路径上他们可以发现一条浓度为 15 的信息素路径，这是由 15 只从 E 走向 D 的先行蚂蚁留下来的。这是选择路径的概率就有了偏差，向 C 走的蚂蚁数是向 D 走的蚂蚁数的 2 倍。对于从 A 到 B 走的蚂蚁也是如此。这个过程一直会持续到所有的蚂蚁最终都选择了最短的路径为止。

蚁群算法流程如下：

(1) 在初始状态下，蚁群蚂蚁外出，此时没有信息素，每只蚂蚁随机选择一个路径；

(2) 在下一个状态，每只蚂蚁被放在不同的城市，从初始点到这些点之间留下了信息素，蚂蚁继续走，直到已经到达目标城市，与此同时，下一批蚂蚁触动，他们会按照各路径上的信息素的多少选择路线，更倾向于信息素多的路径走（同时也有随机性）；

(3) 下一批蚂蚁进入第 2 个步骤，通过信息素的增强进行迭代，多次迭代过后，就会有某一条路径上的信息素明显多于其他路径，这通常就是一条最优路径。

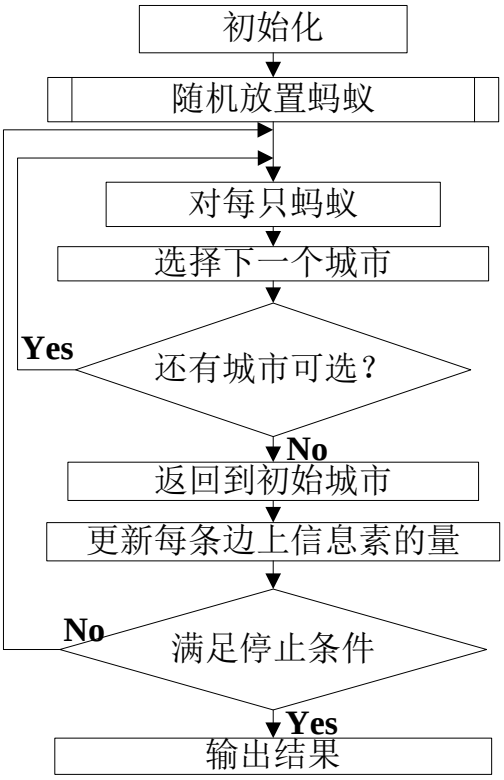


图 2 蚁群算法流程图

3.2 实验结果

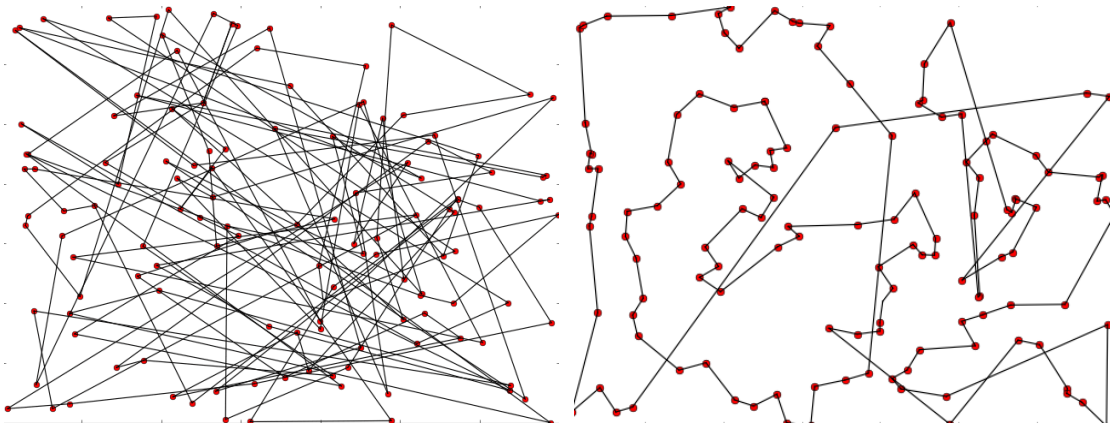
本文采用数据集 ch130 数据进行测试，实验的测试城市数据来源于 http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95 网站，实验数据包含 130 个城市，每个城市。实验所用程序为 Python 语言编写，在 Sublime 开发环境上运行，测试结果四舍五入取整数。

在本文采用的 ant-cycle system 模型算法中，由式(4)可见，从城市 i 到城市 j 的蚂蚁在路径上残留的信息量为 Q/L_{ij}，由于 L_{ij} 为第 k 只蚂蚁所走路径总长度，因而残留信息量随着蚂蚁的运动而变化，值得注意的是，信息素的变化与常数 Q 无关。由于本文采用二次更新，因此，残留信息素的强度与蚂蚁每次循环中所获路径以及循环结束所获路径优劣有关，更新规则会让短路径（较优的解）上对应信息素迭代增加。

3.2.1 参数的选择测试

(1)蚂蚁的数量

通过取不同蚂蚁数量 *AntCount*，试图找出蚂蚁数量对蚁群算法性能的影响。实验选取 α=1.0， β=2.0， ρ=0.5，Q=100，迭代次数 1000。



(a)初始随机路径（48563）　　(b)蚁群算法计算路径（6690）

图 3 蚁群爬行结果图

图 3(a)展示了数据集初始随机路径，其结果为 48563 (b)展示了取 α=1.0， β=2.0， ρ=0.5，Q=100，迭代次数 1000，*AntCount* 取 4 时的结果图，其值为 6690。

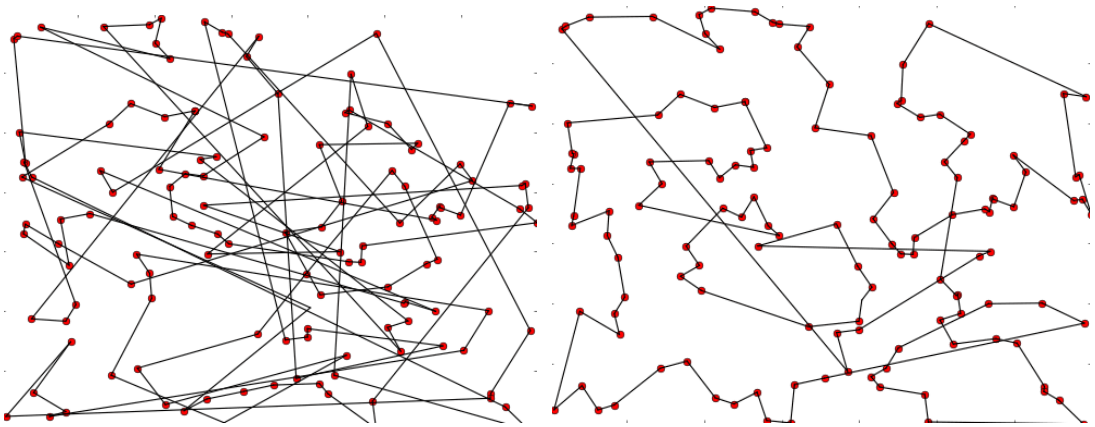
表 1 *AntCount* 测试结果

实验次数	蚂蚁数量	运行结果	运行时间(S)
1	1	8675	13
2	2	7180	22.2
3	3	6807	30.2
4	4	6847	40
5	5	6468	49.3
6	10	6663	93.8
7	15	6733	138.6
8	20	6835	180.4
9	25	6564	225.1
10	30	6661	273.6

根据表 1 所示，对于 ch130 数据来说，当蚂蚁数量达到 3 时，其解就可接近整个算法最优解；另外，从图中可看出，蚂蚁数量与时间成线性关系增长。

(2) α 和 β 的相对重要度

取 Q=100，蚂蚁数量 *AntCount* 取 4，ρ =0.5，迭代次数 1000。为方便观察结果，分别使 α 和 β 值其中一个值不变，另外一个值变化。试验 5 次，取平均值，结果如下：



(a) α 和 β 选择不恰当所得结果（11123）　　(b) α 和 β 选择恰当所得结果（6772）

图 4　α 和 β 选择结果对比图

表 2　α 和 β 测试结果

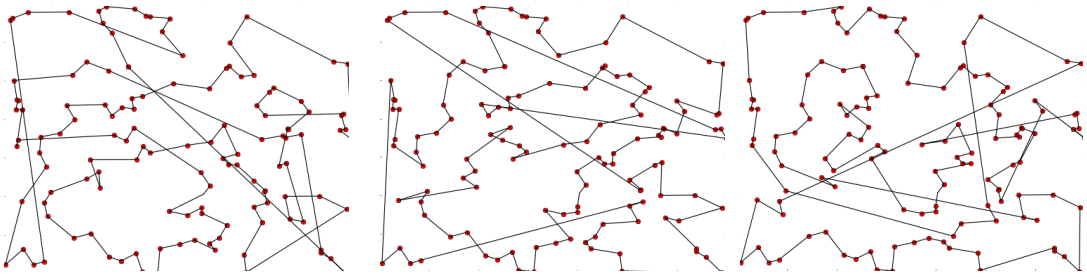
试验次数	α=1.0 β=0.50	α=1.0 β=1.0	α=1.0 β=1.5	α=1.0 β=2.0	α=1.0 β=2.5	α=1.0 β=3.0	α=1.0 β=3.5
1	12683	8187	7135	6526	6810	6835	6768
2	13162	7544	7387	6719	6558	6638	6736
3	12528	8245	7239	6606	6381	6615	6612
4	12476	7935	6392	7062	6749	6903	6484
5	12628	8009	7083	6595	6816	6503	6694
平均值	**12695**	**7984**	**7047**	**6701**	**6663**	**6698**	**6659**
试验次数	α=0.5 β=2.0	α=1.0 β=2.0	α=1.5 β=2.0	α=2.0 β=2.0	α=2.5 β=2.0	α=3.0 β=2.0	α=3.5 β=2.0
1	7665	6526	6877	7654	7671	8409	8125
2	7686	6719	6998	7404	7960	8347	8024
3	7660	6606	7331	7147	7952	8292	8801
4	7819	7062	6948	7322	7549	8173	7813
5	7661	6595	7490	7290	7803	7631	8110
平均值	**7698**	**6701**	**7129**	**7363**	**7787**	**8170**	**8175**

从表 2 和图 4 可看出，启发因子相对 β 过小时，出现差的搜索结果，相当于未给予路径上的能见度（本文指城市 ij 之间距离）充分的重视，蚁群算法易陷入纯粹的、无休止的随机搜索中，这种情况很难找到最优解。当给予 β 合适的重

视时，路径能见度信息反馈作用增强，蚁群算法能获得较好的搜索结果。因此，适当选择 α 和 β 的范围，能够大大提高蚁群算法的性能。本文中，α 的最优值在 1 左右，而 β 在 2~5 之间最优。

(3) 残留系数的选择

实验除 ρ 外其他参数选取合适的范围，蚂蚁数量 *AntCount* 取 4，α=1.0，β=2.5，*Q*=100，迭代次数 1000。信息残留系数 ρ 的选择结果如图 5 和表 3 所示。



(a) ρ=0.2，运行值：7141　(b) ρ=0.2，运行值：6968　(c) ρ=0.2，运行值：6565

图 5　ρ 选择结果对比图

表 3　ρ 测试结果

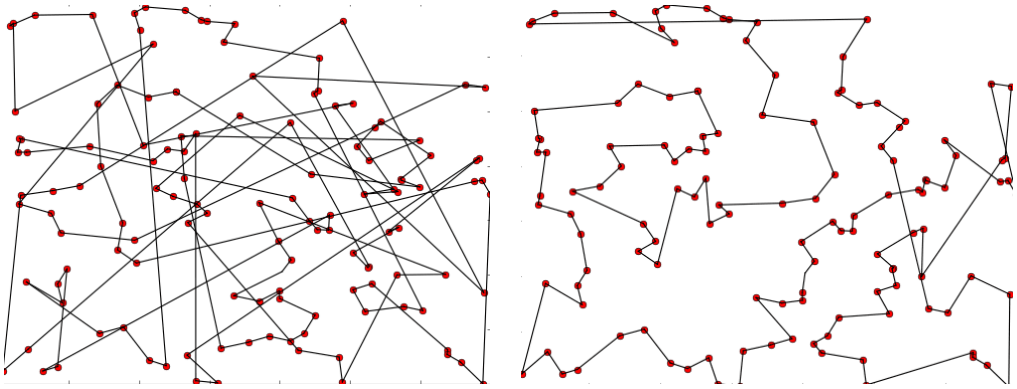
实验次数	ρ=0.2	ρ=0.3	ρ=0.4	ρ=0.5	ρ=0.6	ρ=0.7
1	6850	6688	6704	6364	6573	6749
2	6818	6569	6773	6552	6785	6558
3	6907	6889	6846	6381	6846	6488
4	7099	6664	6819	6749	6536	6753
5	6817	6767	6805	6816	6840	7259
平均值	**6898**	**6715**	**6789**	**6572**	**6716**	**6761**

根据表 3（其中阴影值为本次实验最优解）和图 5 对比图可分析信息残留系数 ρ 对蚁群算法性能的影响。当 ρ 比较小时，由于搜索的随机性增强，搜索结果不佳，容易陷入局部最优。上述结果表明，当 ρ 范围在 0.5-0.7 范围内，算法的性能比较一致。

3.2.2 蚁群算法与遗传算法的比较

蚁群算法参数设置为：*AntCount*=4，α=1.0，β=2.5，ρ=0.5，*Q*=100，蚂

蚁数迭代次数 1000。遗传算法参数设置为：交叉概率 P_x=0.7，变异概率 P_m=0.005，生命个数 LifeCount=50。蚁群算法和遗传算法运行结果图如图 6 所示。



(a)遗传算法（9918）　　(b)蚁群算法（6603）

图 6 蚁群算法与遗传算法对比图

如图 6 所示，遗传算法得到最优解为 9918，而蚁群算法能得到最优解值更小，则说明蚁群算法能达到比遗传算法更优的解。

表 4 算法比较

实验次数	蚁群算法	运行时间(s)	遗传算法	运行时间(s)
1	6670	40.7	9006	1263.5
2	6560	41.9	9817	747.3
3	6532	40	11595	1545.1
4	6979	39.5	8524	2092.6
5	6722	39.3	7836	2436.7
平均值	**6693**	**40.28**	**9356**	**16170.4**

从表 4 可看出，蚁群算法在平均时间 40.28 秒获得平均值为 6693，而遗传算法在平均时间 16170.4 秒获得平均值为 9356。显然，蚁群算法比遗传算法具有更快的收敛速度，能在更短的时间内获得更优的解。本文通过用遗传算法与蚁群算法进行比较，蚁群算法与遗传算法的相似之处有两点：首先，两种算法均采用群体表示问题的解；其次，新群体通过包含在群体中与问题相关的知识来生成。两者的主要区别在于进化计算中所有问题的的知识都包含在当前群体中，而蚁群算法中代表过去所学的知识保存在信息素的踪迹中。

4 总结

蚁群的觅食行为实际上是一种分布式的协同优化机制，在寻找最短路径的过程中，蚁群使用了一种间接的通信方式，即通过向所经过的路径上释放一定的信息素，其他蚂蚁通过感知这种物质的强弱来选择下一步要走的路。换句话说，信息素在蚁群的协作和通讯中起到一种间接媒介的作用。总的来说，人工蚁群的算法主要特点可以概括为以下几点：

(1)采用分布式控制，不存在中心控制，具有自组织性，即群体的复杂行为是通过个体的交互过程显现出来的智能；

(2)每个个体（蚂蚁）只能感知局部的信息，不能直接使用全局信息，这是一种多主体的智能算法，各个体之间通过相互协作来更好的适应环境；

(3)蚁群算法是一类概率性的全局搜索方法，这种非确定性是算法能够有更多的机会求得全局最优解。

参考文献

[1] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a colony of Cooperating agents[J].IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41，1996.

[2] M. Dorigo and L.M. Gambardel la. Ant colony system: Acooperative learning approach to the traveling saleaman problem[J].IEEE Transactions on Evolutionary Computation, 1(1):53-66, 1997.

[3] T. Stutzle and H. Hoos. The MAX-MIN ant system and local search for the traveling salesman problem[C].In T.Baeck, Z. Michalewicz, and X. Yao, editors Proceedings of IEEE-ICEC-EPS 97, IEEE Intenational Conference on Evolutionary Computation and Evolutionary Programming Conference, pages 309-314. IEEE Press,1997.

[4] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank-based version of the ant system: A computational study[J].Central Research Europeon Journal for Operations and Economics,7(1):25-38,1999.

[5]冀俊忠,黄振,刘椿年,代启国. 基于多粒度的旅行商问题描述及其蚁群优化算法[J]. 计算机研究与发展,2010,03:434-444.

[6]夏亚梅,程渤,陈俊亮,孟祥武,刘栋. 基于改进蚁群算法的服务组合优化[J]. 计算机学报,2012,02:2270-2281.

[7]唐禹. 一种路径规划问题的蚁群算法研究[D].哈尔滨工程大学,2013.

[8]吕嘉伟. 面向大规模和动态 PPI 网络功能模块检测的蚁群算法研究[D].北京工业大学,2015.

[9]万正宜,彭玉旭. 求解旅行商问题的改进型量子蚁群算法[J]. 计算机工程与应用. 2015.

[10]康与云,唐敦兵. 基于矩阵算式和蚁群算法的元功能链设计方案优化方法[J]. 机械工程学报,2016:1-9.

[11]侯梦婷,赵作鹏,高萌,张娜娜. 采用角度因子的蚁群优化多路径路由算法[J]. 计算机工程与应用,2016:1-6.

[12] 王银年. 遗传算法的研究与应用[D].江南大学,2009.

附录 A　TSP 测试数据集

NAME: ch130
TYPE: TSP
COMMENT: 130 city problem (Churritz)
DIMENSION: 130
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 334.5909245845 161.7809319139
2 397.6446634067 262.8165330708
3 503.8741827107 172.8741151168
4 444.0479403502 384.6491809647
5 311.6137146746 2.0091699828
6 662.8551011379 549.2301263653
7 40.0979030612 187.2375430791
8 526.8941409181 215.7079092185
9 209.1887938487 691.0262291948
10 683.2674131973 414.2096286906
11 280.7494438748 5.9206392047
12 252.7493090080 535.7430385019
13 698.7850451923 348.4413729766
14 678.7574678104 410.7256424438
15 220.0041131179 409.1225812873
16 355.1528556851 76.3912076444
17 296.9724227786 313.1312792361
18 504.5154071733 240.8866564499
19 224.1079496785 358.4872228907
20 470.6801296968 309.6259188406
21 554.2530513223 279.4242466521
22 567.6332684419 352.7162027273
23 599.0532671093 361.0948690386
24 240.5232959211 430.6036007844
25 32.0825972787 345.8551009775
26 91.0538736891 148.7213270256
27 248.2179894723 343.9528017384
28 488.8909044347 3.6122311393
29 206.0467939820 437.7639406167
30 575.8409415632 141.9670960195
31 282.6089948164 329.4183805862
32 27.6581484868 424.7684581747
33 568.5737309870 287.0975660546
34 269.4638933331 295.9464636385
35 417.8004856811 341.2596589955

36 32.1680938737 448.8998721172
37 561.4775136009 357.3543930067
38 342.9482167470 492.3321423839
39 399.6752075383 156.8435035519
40 571.7371050025 375.7575350833
41 370.7559842751 151.9060751898
42 509.7093253204 435.7975189314
43 177.0206999750 295.6044772584
44 526.1674198605 409.4859418161
45 316.5725171854 65.6400108214
46 469.2908100279 281.9891445025
47 572.7630641427 373.3208821255
48 29.5176994283 330.0382309000
49 454.0082936692 537.2178547659
50 416.1546762271 227.6133100741
51 535.2514330806 471.0648643744
52 265.4455533675 684.9987192464
53 478.0542110167 509.6452028741
54 370.4781203413 332.5390063041
55 598.3479202004 446.8693279856
56 201.1521139175 649.0260268945
57 193.6925360026 680.2322840744
58 448.5792598859 532.7934059740
59 603.2853485624 134.4006473609
60 543.0102490781 481.5168231148
61 214.5750793346 43.6460117543
62 426.3501451825 61.7285415996
63 89.0447037063 277.1158385868
64 84.4920100219 31.8474816424
65 220.0468614154 623.0778103080
66 688.4613313444 0.4702312726
67 687.2857531630 373.5346236130
68 75.4934933967 312.9175377486
69 63.4170993511 23.7039309674
70 97.9363495877 211.0910930878
71 399.5255884970 170.8221968365
72 456.3167017346 597.1937161677
73 319.8855102422 626.8396604886
74 295.9250894897 664.6291554845
75 288.4868857235 667.7284070537
76 268.3951858954 52.9010181645
77 140.4709056068 513.5566720960
78 689.8079027159 167.5947003748
79 280.5784506848 458.7533546925

80 453.3884433554 282.9082328989
81 213.5704943432 525.8681817779
82 133.6953004520 677.1757808026
83 521.1658690522 132.8617086506
84 30.2657946347 450.0754502986
85 657.0199585283 39.7772908299
86 6.9252241961 23.8749241575
87 252.4286967767 535.1659364856
88 42.8551682504 63.8232081774
89 145.8999393902 399.5255884970
90 638.4885715591 62.6262558472
91 489.2756391122 665.3131282446
92 361.2231139311 564.2347787901
93 519.9475425732 347.9711417040
94 129.3349741063 435.6692740389
95 259.7172815016 454.6495181318
96 676.3421890013 371.0979706551
97 84.5133841706 183.3260738572
98 77.7164048671 354.3833863300
99 335.9802442534 660.6321896676
100 264.3554717810 377.5743377274
101 51.6826916855 676.0429509187
102 692.1376849300 543.8010925819
103 169.2191356800 547.8194325476
104 194.0131482339 263.4791316822
105 415.1928395332 78.9133571973
106 415.0432204919 479.0801701569
107 169.8389859939 245.6103433244
108 525.0987124228 213.5063718969
109 238.6851191283 33.4932910965
110 116.2112467718 363.5742702940
111 16.9283258126 656.5711014044
112 434.3440768162 92.6996831431
113 40.5253860363 424.6829615797
114 530.4849979086 183.8390534273
115 484.3595848990 49.2460387276
116 263.6501248722 426.5852608187
117 450.2891917862 126.3853415784
118 441.7822805823 299.7724362653
119 24.2169105375 500.3474481664
120 503.7886861157 514.6895019799
121 635.5389390312 200.9811207275
122 614.5922732529 418.8691931188
123 21.7161351334 660.9741760476

124 143.8266469611 92.6996831431
125 637.7191022040 54.2048412384
126 566.5645610042 199.9551615873
127 196.6849168280 221.8209157619
128 384.9270448985 87.4630166986
129 178.1107815614 104.6905805938
130 403.2874386776 205.8971749407
EOF

附录 B　蚁群算法核心代码

```
import random
import copy
import time
import sys
(ALPHA, BETA, RHO, Q) = (1.0, 2.5, 0.7, 100.0)
(city_num, ant_num, iter_max) = (130, 4, 1000)
distance_x = []
distance_y = []
for line in open('testSet.txt'):
    items = line.strip('\n').split(' ')
    distance_x.append(float(items[1]))
    distance_y.append(float(items[2]))
distance_graph = [[0.0 for col in xrange(city_num)]
                  for raw in xrange(city_num)]
pheromone_graph = [[1.0 for col in xrange(
    city_num)] for raw in xrange(city_num)]

class Ant(object):
    def __init__(self, ID):
        self.ID = ID
        self.__clean_data()
    def __lt__(self, other):
        return self.total_distance < other.total_distance
    def __clean_data(self):
        self.path = []
        self.total_distance = 0.0
        self.move_count = 0
        self.current_city = -1
        self.open_table_city = [True for i in xrange(city_num)]
        city_index = random.randint(1, city_num - 1)
        self.current_city = city_index
        self.path.append(city_index)
        self.open_table_city[city_index] = False
        self.move_count = 1
    def __choice_next_city(self):
        next_city = -1
        select_citys_prob = [0.0 for i in xrange(city_num)]
        total_prob = 0.0
        for i in xrange(city_num):
            if self.open_table_city[i]:
                try:
                    select_citys_prob[i] = pow( pheromone_graph[self.current_city][i],
```

```
ALPHA ) *\
                                pow((1.0 / distance_graph[self.current_city][i]), BETA)
                            total_prob += select_citys_prob[i]
                        except ZeroDivisionError, e:
                            print 'Ant ID: {ID}, current city: {current}, target city: {target}'.format(
                                ID=self.ID, current=self.current_city, target=i)
                            sys.exit(1)
            # select city by way of roulette
            if total_prob > 0.0:
                temp_prob = random.uniform(0.0, total_prob)
                for i in xrange(city_num):
                    if self.open_table_city[i]:
                        temp_prob -= select_citys_prob[i]
                        if temp_prob < 0.0:
                            next_city = i
                            break
            if next_city == -1:
                for i in xrange(city_num):
                    if self.open_table_city[i]:
                        next_city = i
                        break
            return next_city
        def __cal_total_distance(self):
            temp_distance = 0.0
            for i in xrange(1, city_num):
                start, end = self.path[i], self.path[i - 1]
                temp_distance += distance_graph[start][end]
            end = self.path[0]
            temp_distance += distance_graph[start][end]
            self.total_distance = temp_distance
            # print "best_path:", self.path
            Path = self.path
            return Path
        def __move(self, next_city):
            self.path.append(next_city)
            self.open_table_city[next_city] = False
            self.total_distance += distance_graph[self.current_city][next_city]
            self.current_city = next_city
            self.move_count += 1
        def search_path(self):
            self.__clean_data()
            while self.move_count < city_num:
                next_city = self.__choice_next_city()
                self.__move(next_city)
```

```
            self.__cal_total_distance()

class TSP(object):
    def __init__(self):
        self.ants = [Ant(ID) for ID in xrange(ant_num)]
        self.best_ant = Ant(-1)
        self.best_ant.total_distance = 1 << 31
        for i in xrange(city_num):
            for j in xrange(city_num):
                temp_distance = pow(
                    (distance_x[i] - distance_x[j]), 2) + pow((distance_y[i] - distance_y[j]),
2)
                temp_distance = pow(temp_distance, 0.5)
                distance_graph[i][j] = float(int(temp_distance + 0.5))
                pheromone_graph[i][j] = 1.0
    def search_path(self):
        for i in xrange(iter_max):
            for ant in self.ants:
                ant.search_path()
                if ant < self.best_ant:
                    self.best_ant = copy.deepcopy(ant)
            self.__update_pheromone_graph()
    def __update_pheromone_graph(self):
        temp_pheromone = [[0.0 for col in xrange(
            city_num)] for raw in xrange(city_num)]
        for ant in self.ants:
            for i in xrange(1, city_num):
                start, end = ant.path[i - 1], ant.path[i]
                temp_pheromone[start][end] += Q / ant.total_distance
                temp_pheromone[end][start] = temp_pheromone[start][end]
            end = ant.path[0]
            temp_pheromone[start][end] += Q / ant.total_distance
            temp_pheromone[end][start] = temp_pheromone[start][end]
        for i in xrange(city_num):
            for j in xrange(city_num):
                pheromone_graph[i][j] = pheromone_graph[
                    i][j] * RHO + temp_pheromone[i][j]
if __name__ == '__main__':
    test = TSP()
    test.search_path()
    print "Best Distance: ", test.best_ant.total_distance
```