文章编号:1007-6735(2014)02-0135-06 DOI:10.13255/j.cnki.jusst.2014.02.006

求解二次分配问题的预处理快速蚂蚁系统

吴果林[1], 李学迁[2]
(1.桂林航天工业学院 理学部,桂林 541004; 2.上海理工大学 管理学院,上海 200093)

摘要:分析了快速蚂蚁系统(FANT)跳出迭代最优解的策略,指出算法易发生停滞现象的原因,并通过改进算法解的构建步,引入一个变动的参数,提出了求解二次分配问题的一种新算法——预处理快速蚂蚁系统(PFANT).新算法改进了FANT算法易发生停滞的现象,拓宽了迭代最优解邻域的搜索范围,提高了二次分配问题解的质量.

关键词:二次分配问题;快速蚂蚁系统;停滞;变参数
中图分类号:TP 301.6 **文献标志码**:A

Preprocessing Fast Ant System for Quadratic Assignment Problem

WU Guo-lin[1], LI Xue-qian[2]
(1. *Faculty of Science, Guilin University of Aerospace Techology, Guilin 541004, China*;
2. *Business School, University of Shanghai for Science & Technology, Shanghai 200093, China*)

Abstract: For a fast ant system (FANT), the strategy for jumping out of local optimal solution was analysed, and the reason that the algorithm is easy to occur stagnation phenomenon was pointed out. Through the improvement of the solution construction step in the algorithm and the introduction of a variable parameter, a new algorithm for quadratic assignment problem-preprocessing fast ant system (PFANT) was proposed. The new algorithm mends FANT's some shortcomings such as stagnation, extends the search area of FANT iterative optimal solution neighbourhood and improves the solution quality.

Key words: *QAP*; *FANT*; *stagnation*; *variable parameter*

二次分配问题(quadratic assignment problem, QAP)是由Koopmans等[1]在1957年提出的组合优化问题.该问题可描述为给定 n 个设施和 n 个节点,两个 $n\times n$ 的矩阵 $\boldsymbol{D}=(d_{rs})$ 和 $\boldsymbol{F}=(f_{ij})$.其中 d_{rs} 是节点 r 和 s 之间的距离,而 f_{ij} 是设施 i 和 j 之间的流量.QAP是寻找一个分配方案,使得每一个设施分配一个节点.由于设施的数量等于节点的数量,故每一个分配方案相当于整数集 $\{1,2,\cdots,n\}$ 中的一个排列.QAP的目标是寻找整数集 $\{1,2,\cdots,n\}$ 中的一个排列,最小化目标函数

$$\min_{\varphi\in\Phi} Z(\varphi)=\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}d_{\varphi_i\varphi_j}$$

收稿日期:2013-03-24
基金项目:国家自然科学基金资助项目(11271088);上海理工大学国家级项目培育基金资助项目(13XGQ05);上海理工大学人文社科基金资助项目(12XSY08)
第一作者:吴果林(1977-),男,讲师.研究方向:智能优化.E-mail:guat_green@163.com

其中，Φ 为整数集$\{1,2,\cdots,n\}$所有排列的集合，φ 为 Φ 中的一个排列(或问题的一个解)，φ_i 给出了设施 i 在当前解 φ 上的节点.

Sahni 等[2]于 1976 年证明 QAP 是一个 NP－难的优化问题，因此当问题的规模超过 30 时，寻找问题的最优解就变得不切实际，一个切实可行的办法是使用元启发式算法以便在合理的时间内找到高质量的解. 当前用于求解 QAP 的元启发式算法有遗传算法[3]、模拟退火算法[4]、禁忌搜索算法[5-6]、蚁群优化算法[7-13]和迭代局部搜索算法[14-15]等.

在过去的近 20 年里，蚁群优化算法被用于求解许多组合优化问题[16]. 事实上，对于二次分配问题中结构化的、现实生活例子，蚁群优化算法是已知的求解 QAP 的最好算法之一. 本文调查分析了快速蚂蚁系统(fast ant system，FANT)[10-11]，给出了预处理快速蚂蚁系统(preprocessing fast ant system，PFANT). 该算法在构建问题的解时预先进行了搜索，挑选那些质量较优的解参与局部搜索，并设置动态的算法参数，防止算法陷入局部最优解，出现停滞. 这种对 FANT 算法的改进既保持了其运行前期收敛速度快的优点[11]，拓宽了算法的搜索范围，又改进了解的质量. 数值实验进一步表明，预处理快速蚂蚁系统在求解结构化的、现实生活的例子保持蚁群优化算法的优越性；在求解随机产生无结构、网格距离例子时，其性能与求解这两类例子的最好算法——禁忌搜索算法(robust taboo search，RTS)[6]不相上下.

1 快速蚂蚁系统

受蚂蚁系统算法的启发，Taillard 开发了求解 QAP 的蚁群优化算法，称为快速蚂蚁系统[10-11]. 在 FANT 算法中，解的构建是由人工蚂蚁通过随机地选择还没有分配的设施 i，利用概率选择规则将其分配到空闲的节点 j 上，即

$$p_{ij}(t)=\frac{\tau_{ij}}{\sum_{l\in N_i}\tau_{il}(t)}\quad \text{if } j\in N_i \qquad (1)$$

式中，τ_{ij}为分配设施 i 到节点 j 上的权重；N_i 为设施 i 可分配的节点.

FANT 算法不同于其它蚁群优化算法的地方主要是人工蚂蚁的数量和信息素更新管理. FANT 算法仅使用一个人工蚂蚁，单个人工蚂蚁导致该算法快速寻找到一个质量较好的解归咎于局部搜索算法的引入和相当特殊的参数设置. 另一个不同于其它蚁群优化算法，是信息素更新管理，FANT 算法不使用信息素蒸发. 初始所有的信息素都设置为 1，每一次迭代后信息素更新方式为

$$\tau_{ij}(t+1)=\tau_{ij}(t)+r\Delta\tau_{ij}+R\Delta\tau_{ij}^{ib} \qquad (2)$$

其中 $\Delta\tau_{ij}=1/J_{\varphi}$，$\Delta\tau_{ij}^{ib}=1/J_{\varphi^{ib}}$

式中，J_{φ} 为当前解 φ 的目标函数值；$J_{\varphi^{ib}}$ 为迭代最优解 φ^{ib} 的目标函数值；r 和 R 为两个参数. r 的值在算法运行过程中可以发生改变，初始值设为 1，而 R 为固定常数. 除按式(2)的更新规则更新信息素之外，信息素与参数 r 还将在两种情况下发生改变：a. 如果迭代最优解 φ^{ib} 发生改变，则参数与所有的信息素全都重新设置为 1；b. 当人工蚂蚁构建的解与迭代最优解 φ^{ib} 相同时，参数将增加 1 个单位且所有的信息素设为 r.

设信息素矩阵为 $\boldsymbol{\Gamma}=(\tau_{ij})_{n\times n}$，初始，令 $r=1$，$\tau_{ij}=r(i,j=1,2,\cdots,n)$，迭代最优解为 φ^{ib}，则 FANT 重复执行如下步骤(算法 1)：

步骤 1 利用式(1)构建问题的当前解 φ；

步骤 2 利用局部搜索技术改进当前解 φ，为方便计仍记为 φ；

步骤 3 如果 $Z(\varphi)<Z(\varphi^{ib})$，则 $\varphi^{ib}=\varphi$，并设 $r=1$，$\tau_{ij}=r(i,j=1,2,\cdots,n)$；

步骤 4 按式(2)的更新方式更新信息素矩阵 $\boldsymbol{\Gamma}$.

2 预处理快速蚂蚁系统

从前面的叙述可以看出，FANT 试图设计一种能够整合重点搜索迭代最优解和解构建时探索性的算法. 也就是说，一方面通过信息素更新，FANT 不断地增加迭代最优解的权重，使得算法在迭代最优解邻域选择问题的解；另一方面，当算法出现停滞时(FANT 以构建解等于迭代最优解作为判断标准)，重新设置信息素，并令 $r=r+1$，降低迭代最优解与算法构建解所对应节点信息素增加的比重，减少迭代最优解所对应节点的权重，拓宽解的搜索范围，增加算法构建解的探索性. 然而，从实验的结果来看，FANT 算法解的质量并没有达到预期效果，只是在算法运行的初期，FANT 总体上比其它算法要好一些，在算法运行的后期，其解的质量比其它算法要劣[11]. 即 FANT 算法初期收敛速度快，后期易出现停滞. 出现这种现象，与 FANT 算法信息素更新策略有关. 由于 FANT 算法在每次出现新的迭代最优解时，信息素重新设置为 1，迭代最优解对应的节点

信息素设为 R,算法始终在迭代最优解邻域寻找问题的解,故算法运行初期收敛速度快.易见,这样处理的结果是算法容易陷于局部最优解,产生停滞.尽管 FANT 算法设计了跳出局部最优解的策略(当人工蚂蚁构建的解与迭代最优解 φ^{ib} 相同时,参数将增加 1 个单位且所有的信息素设为 r),但这种方式所需迭代次数太多,代价太高.关于这一点,可从下面的分析看出.

设 FANT 在 t 次迭代产生的迭代最优解 φ^{ib},信息素矩阵为 $\boldsymbol{\Gamma}=(\tau_{ij})_{n\times n}$,$\tau_{ij}=r$ 且 $r=1$,进一步假设此解 φ^{ib} 为局部最优解,算法出现停滞.下面计算 FANT 算法跳出局部最优解所需迭代的次数.由于 FANT 算法采用当迭代产生的解与迭代最优解 φ^{ib} 相同时,重新设置信息素 r,降低迭代最优解权重的方法,改变停滞现象.为此,先计算第一次当前解与迭代最优解 φ^{ib} 相同(即第 $t+k_1$ 次迭代产生的解等于迭代最优解 φ^{ib})时,FANT 算法所需迭代次数,记为 k_1.根据 FANT 算法,每次迭代时,当前解对应节点的信息素增加 1;迭代最优解 φ^{ib} 对应节点的信息素增加 R.故第 $t+k_1$ 次迭代时,信息素矩阵 $\boldsymbol{\Gamma}$ 每一行元素和都为 $n+k_1(R+1)$,迭代最优解 φ^{ib} 对应的每一个节点的信息素应小于等于 $k_1(R+1)+1$(此值为每次迭代时当前解等于迭代最优解时信息素的值).设在第 $t+k_1$ 次迭代时,迭代最优解作为当前解的概率 P,由概率乘法公式

$$P(A_1A_2\cdots A_n)=P(A_1)P(A_2\mid A_1)\cdots P(A_n\mid A_1A_2\cdots A_{n-1})$$

则 P 应满足

$$P\leqslant\frac{k_1(R+1)+1}{n+k_1(R+1)}\frac{k_1(R+1)+1}{n+k_1(R+1)-1}\frac{k_1(R+1)+1}{n+k_1(R+1)-2}\cdots\frac{k_1(R+1)+1}{n+k_1(R+1)-(n-1)}\leqslant$$

$$\frac{k_1(R+1)+1}{n+k_1(R+1)}\frac{k_1(R+1)+1+1}{n+k_1(R+1)-1+1}\frac{k_1(R+1)+1+2}{n+k_1(R+1)-2+2}\cdots\cdot$$

$$\frac{k_1(R+1)+1+(n-1)}{n+k_1(R+1)-(n-1)+(n-1)}=\frac{k_1(R+1)+1}{n+k_1(R+1)}\frac{k_1(R+1)+2}{n+k_1(R+1)}\frac{k_1(R+1)+3}{n+k_1(R+1)}\cdots\cdot$$

$$\frac{k_1(R+1)+n}{n+k_1(R+1)}\leqslant\left(\frac{1}{n+k_1(R+1)}\right)^n\left\{\frac{[k_1(R+1)+1]+[k_1(R+1)+2]+\cdots+[k_1(R+1)+n]}{n}\right\}^n=$$

$$\left(\frac{1}{n+k_1(R+1)}\right)^n\left[k_1(R+1)+\frac{n+1}{2}\right]^n=\left[(k_1(R+1)+\frac{n+1}{2})/(n+k_1(R+1))\right]^n$$

即 $P^{\frac{1}{n}}\leqslant(k_1(R+1)+\frac{n+1}{2})/(n+k_1(R+1))$,$\Rightarrow$

$$k_1\geqslant(nP^{\frac{1}{n}}-\frac{n+1}{2})/((R+1)(1-P^{\frac{1}{n}}))$$

同理,第 s 次重新设置信息素时,FANT 算法所需迭代次数 $k_s\geqslant s(nP^{\frac{1}{n}}-\frac{n+1}{2})/(R+s)(1-P^{\frac{1}{n}})$.假定算法在第 s 次重新设置信息素后找到新的迭代最优解,则算法跳出局部最优解所需要迭代的次数 $k\geqslant\sum_{i=1}^{s}k_i$.考虑到 FANT 算法每次迭代所需要的时间花费为 $O(n^3)$[11],算法跳出局部最优解的代价非常高.

表 1 给出了 QAP 规模为 25,50,100,选择概率为 0.1,0.5,0.9,重新设置信息素为 1,2,…,6 时,FANT 算法所需迭代的次数,R 的取值均为 6.

表 1　跳出局部最优解的迭代次数

Tab.1　Iterations times of jumping out of local optimal solution

信息素	25			50			100		
	0.1	0.5	0.9	0.1	0.5	0.9	0.1	0.5	0.9
1	16	60	405	71	248	1 656	297	1 010	6 701
2	44	164	1 113	195	681	4 554	816	2 777	18 428
3	82	301	2 056	360	1 258	8 417	1 508	5 133	34 063
4	127	467	3 188	558	1 950	13 053	2 338	7 960	52 826
5	178	656	4 474	783	2 737	18 321	3 281	11 172	74 147
6	234	863	5 889	1 031	3 602	24 116	4 319	14 706	97 600

从表 1 可以看出,要以较高的概率跳出局部最优解,算法所需的迭代次数非常多.以 QAP 规模 50 为例,若以 0.9 的概率取得局部最优解,算法至少需要迭代 1 656 次.由于算法每次迭代所需的时间为 $O(50^3)$,故算法跳出局部最优解的时间花费至少为 $1\,656\times O(50^3)$,这个数值已经超过 RTS[6] 算法求

解 QAP 规模为 50 的迭代 1 000×50 次所需的时间(RTS 每次迭代所需的时间为 $O(50^2)$),而 RTS 算法迭代 1 000×50 次得到问题的解常常作为其它算法求 QAP 时比较的参照文献[8,14-15]. 也就是说,对于规模为 50 的 QAP 实例而言,FANT 算法以 0.9 的概率跳出局部最优解的时间大于 RTS 算法 1 000×50 次迭代的时间. 由表 1 不难得出,QAP 其它规模的例子同样也有类似的结果. 综上所述,FANT 算法设计跳出局部最优解的策略所需迭代次数太多,算法后期很少改进算法解的质量.

尽管有上述不足,但 FANT 算法仍然是求解 QAP 的非常有竞争力的算法,文献[12-13]给出了两种改进的 FANT 算法. 事实上,仔细分析算法 1 的流程不难发现,步骤 2 是对步骤 1 产生的解利用局部搜索进行改进,而改进后解质量的好坏很大程度取决于步骤 1 产生解的质量. 因此,可以对步骤 1 进行改进,如对步骤 1 产生的解进行预处理,以便较高质量的解参与步骤 2 的搜索改进. 由于步骤 1 构建解是不断通过随机选择一个设施由概率选择式(1)分配一个节点得到,故对于每一个设施 i 分配的节点 φ_i,可以比较与前次迭代产生的解所对应的设施 i 的节点 φ'_i 是否相同. 若 $\varphi_i \neq \varphi'_i$,将前次迭代解对应的两个节点 φ_i,φ'_i 的位置进行交换,再比较交换后解的质量;若质量优于交换前的,则设施 i 分配节点 φ_i,否则设施 i 还是分配原来的节点 φ'_i. 上述方法通过不断地重复,最终得到改进的迭代解. 易见,改进后的迭代解剔除了那些迭代过程中产生的质量较劣的解,使得每次迭代产生的解始终以较好的解参与步骤 2 的局部搜索,减少一些无为的迭代与局部搜索,保证更好质量的解产生. 当然,上述改进也易使算法快速陷入局部最优解,产生停滞. 为防止或减少该现象的发生,在算法每次得到一个新的迭代最优解时重新设置一个新的 r 值,记为 $r(t)$,区别于原算法每次重新设置 r 都等于 1.

另一方面,由算法 1 可知,FANT 算法采用固定参数 R 的策略,这种固定参数的策略对于规模较小的 QAP 例子容易过早陷入局部最优解. 而对于规模较大的 QAP 例子由于迭代最优解的权重较轻,算法收敛速度较慢. 为此,文献[13]给出了一种变参数的 FANT 算法,实验证明,改进了的算法求解 QAP 的质量有较大的提高. 借鉴文献[13]的思想,为进一步简化运算,针对不同规模 QAP 采用不同的参数 R 的策略,记为 $R(n)$. 因此,改进后的算法信息素更新方式为

$$\tau_{ij}(t+1) = \tau_{ij}(t) + r(t)\Delta\tau_{ij} + R(n)\Delta\tau_{ij}^{ib} \quad (3)$$

综上所述,可以对 FANT 算法作如下两点改进:其一,在 FANT 算法解的构建步,对每一个设施分配的节点进行预处理,判断该分配是否改进上一次迭代解的质量,若改善,则分配该节点,否则,保留原迭代解分配的节点;其二,在 FANT 算法信息素更新步,由式(3)取代式(2)作为算法的信息素更新策略. 改进后的 FANT 算法称为预处理快速蚂蚁系统(PFANT),其算法流程为:

设信息素矩阵为 $\boldsymbol{\Gamma} = (\tau_{ij})_{n\times n}$,初始令 $r(0) = 1$,$\tau_{ij} = r(0)(i,j = 1,2,\cdots,n)$,初始解为 φ,迭代最优解为 φ^{ib},则 PFANT 重复执行步骤(算法 2)为:

步骤 1 考查由式(1)分配的每一个节点的能否改进上一次迭代解 φ 的质量. 若改善,则分配该节点;否则,保留原迭代解分配的节点. 为方便计构建后的解记为 φ;

步骤 2 利用局部搜索技术改进当前解 φ,仍记为 φ;

步骤 3 如果 $Z(\varphi) < Z(\varphi^{ib})$,则 $\varphi^{ib} = \varphi$,重新设置信息素 $\tau_{ij} = r(t)(i,j = 1,2,\cdots,n,t = 1,2,\cdots)$;

步骤 4 按式(3)的更新方式更新信息素矩阵 $\boldsymbol{\Gamma}$.

3 数值实验

这一部分选择 QAPLIB 中的例子运行 PFANT 和 FANT 算法. 实验的硬件环境为 Pentium(R)4 2.5GHz 处理器,80 GB 硬盘,256 MB 内存,Microsoft Windows SP3 操作系统,开发工具为VC++ 6.0. 对于 FANT 算法,参数 $R = 6$;PFANT 算法参数 $R = [2n]$,其中[]为取整运算,参数 r 随着迭代最优解 φ^{ib} 的改变在 1 与 2 之间不断地取值. 实验以 RTS 算法迭代 1 000n 次所需的时间作为迭代终止条件. 实验采用 QAPLIB 中的例子按文献[17]分为 4 类. 第一类,随机产生无结构问题,这类问题是根据均匀分布随机产生距离流量矩阵的问题. 第二类,随机产生网格距离问题,这类问题的距离来源于一个网格,节点与节点的距离定义为网格上节点之间的曼哈顿距离. 第三类,现实问题,这类问题是从实际问题抽象出来的. 第四类,模拟现实问题,因为第三类现实问题规模较小,所以这类问题是根据现实问题的特点随机产生的较大规模的问题. 另外,考虑到规模小于 20 的问题容易求解,这里只测试规模大于等于 20 的问题,数值实验结果见表 2. 表中给出的结果是超出已知最优解的百分比,最好的结果用斜粗体表示.

表 2　PFANT 算法的测试结果

Tab.2　Testing result of the PFANT algorithm

	问题实例	已知最优解	RTS	PFANT	FANT
随机产生无结构问题	Tai25a	1 167 256	0.008 2	***0.004 0***	0.010 5
	Tai30a	1 818 146	***0.004 0***	0.005 9	0.020 3
	Tai35a	2 422 002	***0.008 2***	0.016 8	0.015 6
	Tai40a	3 139 370	0.012 6	***0.006 9***	0.012 1
	Tai50a	4 938 796	0.014 3	***0.013 5***	0.019 6
	Tai60a	7 205 962	0.013 3	***0.013 1***	0.019 4
	Tai80a	13 499 184	***0.017 8***	0.023 2	0.017 8
模拟现实问题	Tai20b	122 455 319	***0.000 0***	***0.000 0***	***0.000 0***
	Tai25b	344 355 646	***0.000 0***	***0.000 0***	***0.000 0***
	Tai30b	637 117 113	0.000 2	***0.000 0***	***0.000 0***
	Tai35b	283 315 445	***0.000 0***	***0.000 0***	0.001 9
	Tai40b	637 250 948	***0.000 0***	***0.000 0***	0.000 1
	Tai50b	458 821 517	0.001 9	0.004 1	***0.000 0***
	Tai60b	608 215 054	0.000 6	***0.000 0***	0.000 2
	Tai80b	818 415 043	***0.000 7***	0.009 1	0.006 0
	Tai100b	1 185 996 137	0.002 6	***0.000 2***	0.002 7
随机产生网格距离问题	nug30	6 124	***0.000 0***	0.000 7	0.000 7
	sko42	15 812	***0.000 4***	***0.000 4***	0.003 5
	sko49	23 386	0.001 0	***0.000 7***	0.000 9
	sko56	34 458	***0.001 7***	0.002 1	0.003 1
	sko64	48 498	0.000 9	***0.000 1***	0.003 1
	sko72	66 256	0.000 8	***0.000 7***	0.002 6
	sko81	90 998	0.001 6	***0.001 1***	0.002 9
	sko90	115 534	***0.001 3***	0.003 3	0.002 8
现实问题	kra30a	88 900	***0.000 0***	***0.000 0***	***0.000 0***
	kra30b	91 420	***0.000 0***	***0.000 0***	***0.000 0***
	ste36a	9 526	0.002 5	***0.000 0***	0.020 8
	ste36b	15 852	0.008 7	***0.000 0***	***0.000 0***

尽管算法解的质量很大程度依赖所求例子[17]，但从表 2 仍然不难发现，PFANT 算法解的质量较 FANT 算法有较大幅度的提升，总体上要优于 FANT 算法，例外的只有 Tai35a，Tai80a，Tai50b，Tai80b. 对比 RTS 算法，在 RTS 算法求解有较大优势的随机产生无结构与网格距离例子中，PFANT 算法总体上质量也稍优于 RTS 算法；至于现实问题及模拟现实问题，PFANT 算法全面优于 RTS 算法，例外的只有 Tai50b，Tai80b. 相对 MMAS[8]，ILS[14]，HILS[15]算法在现实问题及模拟现实问题优于 RTS 算法，在随机产生无结构及网格距离问题逊于 RTS 算法而言，PFANT 算法求解 QAP 上有较大提高.

其次，考察 FANT 与 PFANT 算法跳出局部最优解时重新设置信息素的次数，表 3（见下页）给出了两算法求解上述 4 类问题的结果. 从表 3 明显可以看出，上述 4 类问题的每一个例子，PFANT 算法重新设置的次数都小于或等于 FANT 算法. 出现这种情况的原因是由于 PFANT 算法在每次迭代构建问题的解时都预先进行了搜索，剔除了那些质量较劣的解参与局部搜索. 在相同的迭代时间内，算法能在局部最优解更广的邻域内搜索，增加发现更优的局部最优解的机会，减少了重新设置信息素的次数.

表3 FANT与PFANT算法重新设置信息素的次数

Tab.3 The times of resetting pheromone in the FANT and PFANT algorithm

算法	问题实例							
	Tai25a	Tai30a	Tai35a	Tai40a	Tai50a	Tai60a	Tai80a	Tai20b
FANT	3	2	1	3	2	1	0	1
PFANT	1	0	1	1	1	0	0	0
	Tai25b	Tai30b	Tai35b	Tai40b	Tai50b	Tai60b	Tai80b	Tai100b
FANT	3	0	0	5	1	1	0	0
PFANT	0	0	0	0	0	1	0	0
	nug30	sko42	sko49	sko56	sko64	sko72	sko81	sko90
FANT	0	2	2	2	1	1	0	0
PFANT	0	2	1	1	0	1	0	0
	kra30a	kra30b	ste36a	ste36b				
FANT	0	3	2	1				
PFANT	1	0	2	0				

4 结 论

尽管FANT算法在信息素更新管理与解的构建方面较其它蚁群优化算法作了较大的改进，为求解不规则QAP例子最有竞争性的算法之一[9]，但也有明显的缺点(如算法前期收敛速度较快，后期速度较慢，易发生停滞现象等)．通过分析FANT算法跳出迭代最优解的策略，指出算法易发生停滞现象的原因，并提出了改进的方法，得到一种求解QAP的新算法——预处理快速蚂蚁系统(PFANT)．理论与实验分析表明，PFANT算法较大程度地改进FANT算法的缺点，拓宽了算法迭代最优解邻域的搜索范围，减少了重新设置信息素的次数，改善了QAP解的质量．

参考文献：

[1] Koopmans T C, Berkmann M J. Assignment problems and the location of economic activities [J]. Econometrica, 1957, 25(1): 53－76.

[2] Sahni S, Gonzalez T. P-Complete approximation problems[J]. Journal of the Association of Computing Machinery, 1976, 23(3): 555－565.

[3] Ahuja R K, Orlin J B, Tiwari A. A greedy genetic algorithm for the quadratic assignment problem[J]. Computers and Operations Research, 2000, 27(10): 917－934.

[4] Connolly D T. An improved annealing scheme for the QAP[J]. European Journal of Operational Research, 1990, 46(1): 93－100.

[5] Skorin-Kapov J. Tabu search applied to the quadratic assignment problem[J]. ORSA Journal on Computing, 1990, 2(1): 33－45.

[6] Taillard E D. Robust taboo search for the quadratic assignment problem[J]. Parallel Computing, 1991, 17(4/5): 443－455.

[7] Maniezzo V, Colorni A. The ant system applied to the quadratic assignment problem[J]. IEEE Transactions on Knowledge and Data Engineering, 1999, 11(5): 769－778.

[8] Stützle T, Hoos H H. MAX－MIN ant system[J]. Future Generation Computer Systems, 2000, 16(8): 889－914.

[9] Gambardella L M, Taillard E D, Dorigo M. Ant colonies for the quadratic assignment problem[J]. Journal of the Operational Research Society, 1999, 50(2): 167－176.

[10] Taillard E D. FANT: fast ant system. Technical report IDSIA－46－98[R]. Lugano: IDSIA, 1998.

[11] Taillard E D, Gambardella L M. Adaptive memories for the quadratic assignment problems. Technical ReportIDSIA－87－97[R]. Lugano: IDSIA, 1997.

[12] 吴果林，刘登峰．改进的快速蚁群系统求解二次分配问题[J]．桂林航天工业学院学报，2012，17(4)：424－427.

[13] 吴果林．变参数的快速蚂蚁系统求解二次分配问题[J]．科学技术与工程，2013，13(7)：324－328.

[14] Stützle T. Iterated local search for the quadratic assignment problem [J]. European Journal of Operational Research, 2006, 174(3): 1519－1539.

[15] Hussin M S, Stützle T. Hierarchical iterated local search for the quadratic assignment problem [J]. Lecture Notes in Computer Science, 2009, 5818: 115－129.

[16] 马良，项培军．蚂蚁算法在组合优化中的应用[J]．管理科学学报，2004，4(2)：32－37.

[17] Taillard E D. Comparison of iterative searches for the quadratic assignment problem[J]. Location Science, 1995, 3(2): 87－105.

(编辑：金 虹)