文章编号：1001-0920(2009)08-1132-05

一种基于 GPU 加速的细粒度并行蚁群算法

李建明[a]，胡祥培[b]，庞占龙[a]，钱昆明[a]
（大连理工大学 a.电子与信息工程学院，b.系统工程研究所，辽宁 大连 116024）

摘 要：为改善蚁群算法对大规模旅行商问题的求解性能，提出一种基于图形处理器(GPU)加速的细粒度并行蚁群算法.将并行蚁群算法求解过程转化为统一计算设备架构的线程块并行执行过程，使得蚁群算法在 GPU 中加速执行.实验结果表明，该算法能提高全局搜索能力，增大细粒度并行蚁群算法的蚂蚁规模，从而提高了算法的运算速度.

关键词：蚁群算法；并行处理；图形处理器；细粒度
中图分类号：TP301.6 **文献标识码**：A

A parallel ant colony optimization algorithm based on fine-grained model with GPU-accelerated

LI Jian-ming[a], *HU Xiang-pei*[b], *PANG Zhan-long*[a], *QIAN Kun-min*[a]
(a. School of Electronic and Information Engineering, b. Institute of Systems Engineering, Dalian University of Technology, Dalian 116024, China. Correspondent: LI Jian-ming, E-mail: lijm@dlut.edu.cn)

Abstract: An algorithm of fine-grained parallel ant colony optimization algorithm (ACO) based on graphics process unit(GPU) accelerated is proposed to improve the performance of ACO for application to large-scale TSP problems. The process of parallel ACO is convert into that of parallel compute unified device architecture (CUDA) thread blocks, which makes PACO speed up. The experimental results show that the algorithm improves the ability of global search, increases the ant population in the PACO, speeds up its running and provides ordinary user with a feasible PACO solution.

Key words: Ant colony optimization algorithm; Parallel process; Graphics process unit; Fine-grained

1 引 言

蚁群算法(ACO)是一种启发式搜索算法[1]，用于求解优化问题的近优解.对于中小规模的优化问题，ACO 算法得到了广泛的应用，并取得了良好的优化效果；对于大规模或超大规模的优化问题，ACO 算法往往需要大量的计算时间，因而显得力所不及.并行 ACO 算法能大幅度缩减问题求解的时间，因此成为研究的热点之一[2-4].

细粒度并行蚁群算法(FGACO)是并行蚁群算法的重要模型之一，它具有保持最大并行性、提高全局搜索能力等优势.并行蚁群算法的研究主要是在并行机上运行，或用多线程技术进行模拟.该算法在细粒度并行方面存在以下不足：1)在并行机上实现 FGPACO，需要为每只蚂蚁单独地运行进程，而复杂问题的规模导致了进程间的通信损耗很大，因此当前的并行蚁群算法多采用粗粒度模型[3]；2)多线程技术是在 CPU 上用串行来模拟并行，并不能真正提高运行速度；3)大多数研究人员难以接触到上述并行机，而且并行机的管理和使用也相对复杂.

近年来，图形处理器(GPU)发展迅速，提高了计算机图形处理的速度.GPU 的高速浮点运算能力、并行计算和可编程功能，也为通用计算提供了良好的并行计算平台[5].NVIDIA 公司的统一计算设备架构(CUDA)，为研究人员利用 GPU 进行数据并行处理提供了便捷的手段.通过 GPU 加速并行的优化算法，是解决上述 FGACO 面对问题的可行方法之一.

李建明等[6]提出一种基于 GPU 的细粒度并行遗传算法，将并行遗传算法的求解过程转化为 GPU

收稿日期：2008-09-01；**修回日期**：2008-11-19.
基金项目：国家自然科学基金项目(70571009,70671014)；国家杰出青年基金项目(70725004)；高等学校博士点基金项目(20060141013)；辽宁省高等学校优秀人才支持计划项目([2006]124).
作者简介：李建明(1975—)，男，辽宁葫芦岛人，博士，从事进化计算、图像处理的研究；胡祥培(1962—)，男，安徽绩溪人，教授，博士生导师，从事智能运筹学、最优控制等研究.

纹理渲染过程，通过 GPU 约减技术减少 CPU 与 GPU 的数据交换，取得了较高的加速比. Li 等[7]提出了基于 GPU 的并行鱼群算法，取得了良好的优化效果，但该算法通过 CUDA 架构的每个线程来模拟鱼群的个体，没有充分挖掘 GPU 的并行计算能力.

本文在上述方法的基础上，提出一种改进的基于 GPU 加速的细粒度并行蚁群算法，通过使用 CUDA 架构的线程块来模拟蚂蚁个体，充分利用了 GPU 的并行计算能力. 该算法能增大细粒度并行的蚂蚁规模，较好地保持了算法抑制早熟的特性，提高了算法的运行速度.

2　MMAS 求解 TSP 问题

自蚁群算法提出以来，研究人员通过各种方法对其进行改进. 然而，大多数方法都存在容易陷入局部最优值的问题. 为了克服这一缺点，Stutzle 等[8-10]提出了最大-最小蚂蚁算法(MMAS)，有效地解决了搜索停滞的问题. 实验结果表明，该算法比其他改进算法更有效，许多并行 ACO 算法都使用该算法加以实现. 因此本文选择 MMAS 算法对 GPU 进行加速处理.

MMAS 算法求解 TSP 问题的具体步骤如下：

1) 将信息素初始化为最大值 $\tau_{\max}$.

2) 每只蚂蚁都有记录自己已走过区域的禁忌表($\text{tabu}_k(s)$). 蚂蚁必须根据禁忌表和概率函数寻找下一个区域，以保证该蚂蚁从起点出发遍历一次且仅一次其他所有区域，并最终回到起点.

3) 每只蚂蚁依据以区域距离和连接边上信息素为变量的概率函数，决定如何选择下一个区域. 即

$$p=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum\limits_{j\in\text{allowed}}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}, & j\in\text{allowed};\\ 0, & \text{otherwise}.\end{cases}\tag{1}$$

其中：allowed 为该蚂蚁禁忌表中未经过的区域；η_{ij} 为启发式因子，表示蚂蚁从区域 i 转移到区域 j 的期望程度，取区域 i 与 j 之间距离的倒数；$\tau_{ij}(t)$ 为 t 时刻边(i,j) 上信息素的强度；α 和 β 分别为信息素和启发式因子的相对重要程度.

4) 当所有蚂蚁完成一次周游后，使用迭代最优解或全局最优解更新各区域间的信息素，有

$$\tau_{ij}(t+1)=(1-\rho)\tau_{ij}(t)+\Delta\tau_{ij}^{\text{best}}.\tag{2}$$

其中：$\rho(0<\rho<1)$ 表示路径上信息素的挥发程度；$\Delta\tau_{ij}^{\text{best}}=1/L_{\text{best}}$，$L_{\text{best}}$ 为本次迭代的最短路径或迄今为止找到的最短路径. 为了防止迭代中出现停滞现象，总是将区域间的信息素限制在$[\tau_{\min},\tau_{\max}]$ 的范围，超过这个范围的值被强制设为 $\tau_{\min}$ 或 $\tau_{\max}$.

3　基于 GPU 的并行 MMAS 算法

本文通过对 CUDA 的研究，将 MMAS 转化为 GPU 中的 SIMD 处理过程，充分利用 GPU 的高速浮点运算和并行计算的特性，以提高算法的速度. 整个处理过程分为三部分：1) 蚂蚁个体周游区域构造回路；2) 计算路径长度，找出迭代最优解；3) 更新信息素.

采用并行机进行并行计算时，更新信息素矩阵会造成较大的通信延迟，而 GPU 则不存在这种通信开销. 在整个算法执行过程中，信息素矩阵始终保存在显存中，所有蚂蚁共享信息素矩阵和全局最优解，从而加快了寻优过程，提高了全局搜索能力.

在 GPU 中实现并行 MMAS 算法，需要解决以下问题：1) 建立 MMAS 算法的 GPU 并行化模型；2)GPU 转移区域的并行搜索；3)GPU 芯片内存的优化使用；4)FGACO 算法的 GPU 转化.

3.1　MMAS 算法的 GPU 并行化模型

在蚂蚁周游区域的过程中，每只蚂蚁独立地构造自己的回路，因而自然地存在并行性. 根据 CUDA 提供的编程模型，容易让人想到把蚂蚁个体映射为一个线程，但实验结果表明这种方法的效率较低，无法发挥 GPU 并行的特性.

算法使用的状态转移规则称为随机比例规则，要求计算出所有可走区域的转移概率. 如果每个线程完成一只蚂蚁的区域选择，则每个线程首先要对所有可走的区域计算$[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$，然后计算$\sum\limits_{j\in\text{allowed}}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$，求得每个区域的选择概率$[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}\Big/\sum\limits_{j\in\text{allowed}}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}$. 这便造成了 3 次复杂度为 $O(n)$ 的循环，严重地降低了效率.

本文将蚂蚁个体转化为线程块，每个线程块完成一只蚂蚁的区域选择运算. 此时，线程块中的线程并行累加和扫描，运算的复杂度由 $O(n)$ 降为 $O(\log_2^n)$. 算法的 GPU 并行化模型如图 1 所示.

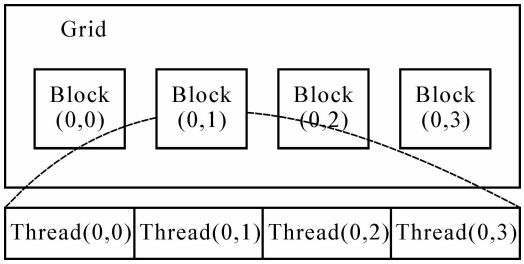


图 1　MMAS 算法的 GPU 并行化模型

3.2　转移区域的 GPU 并行搜索

转移区域的搜索过程如下：

首先计算所有可走区域的$[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$，然后

求得 $\sum_{j\in \text{allowed}}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$，最后利用求得的 $[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$ 计算转移概率 $[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}/\sum_{j\in \text{allowed}}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$.

根据转移概率选择下一个区域时，顺序地累加区域的转移概率. 如果累加结果大于一个随机值，则此时的区域便作为转移区域. GPU 是把一个函数应用于一组数据产生一组结果的并行过程. 在 GPU 中，直接进行数据求和或根据转移概率选择转移区域，这类串行运算的效率很低. 本文将这两种串行计算转化为并行计算，以适应 GPU 的 SIMD 特性.

利用二分树模型的GPU并行消减方法，并行计算 $\sum_{j\in \text{allowed}}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$；对转移概率并行计算前缀和，并用二分树模型的 GPU 并行消减方法，找到前缀和与随机值之差绝对值最小的区域，该区域即为转移区域.

3.3 GPU 芯片内存的优化使用

与设备内存相比，GPU 每个多处理器上的芯片内存空间很小，但访问速度却比设备内存快. 设备内存的读写操作需要大约 400 ～ 600 个时钟周期，而芯片内存只需 4 个时钟周期. 因此利用芯片内存可有效降低设备内存的访问延迟.

为了提高算法效率，在算法设计过程中优化了设备内存和芯片内存的使用. 对于需要多次读写数据的处理过程，充分利用GPU的芯片内存降低内存读取的延迟. 以蚂蚁选择下一个转移区域为例，对算法作如下设计处理：

设 m 为蚁群规模，n 为区域规模，用 $n\times n$ 的二维数组存储任意两个区域之间的距离(distance[n][n]) 和信息素浓度(trial[n][n])，用 $m\times n$ 的二维数组存储蚂蚁的禁忌表(tabu[m][n]) 和允许区域表(AllowedCity[m][n]). 每个线程块从设备内存中读取蚂蚁当前所在区域与其他区域之间的距离和信息素值

$\text{distance}_{\text{bid}}[\text{curcity}][j]$, $\text{trial}_{\text{bid}}[\text{curcity}][j]$.

该蚂蚁的允许区域为

$\text{AllowedCity}_{\text{bid}}[j]$, $j=0,1,\cdots,n-1$.

其中 bid 是 Block 的索引. 这些数据写入芯片内存中的临时数组 s_distance[n][n]，s_trial[n][n] 和 s_allowed[n][n]. 每个线程利用芯片内存中的数据计算到区域 tid(线程索引) 的转移概率，写入芯片内存中的 s_prob[tid]. 根据 s_prob 中的数据计算出下一步要走的区域，将结果写回设备内存中的禁忌表 tabu 和允许区域表 AllowedCity. 计算过程如图 2 所示.

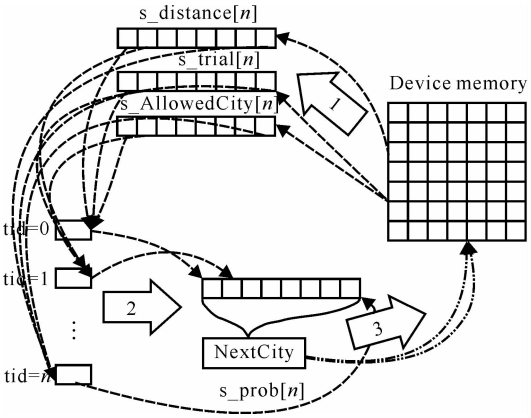


图 2 利用芯片内存计算转移概率

整个过程只需读写设备内存一次，寻找转移区域的计算利用载入芯片内存中的数据完成，显著提高了算法的效率. 处理的问题规模越大，数据读写操作越密集，使用芯片内存取得的加速效果越显著.

其他计算过程(如找出迭代最优值、更新信息素等) 均采用类似的处理方法.

3.4 FGACO 的 GPU 转换(GPUACO)

3.4.1 GPUACO 算法模型

本文将 FGACO 算法转化为 GPU 的 CUDA 并行计算过程. 变量定义如下：

设区域数为 n，蚁群规模为 m. 定义数组 distance[n][n] 存储区域之间的距离；trial[n][n] 存储信息素；inctrial[n][n] 存储信息素增量；tabu[m][m] 存储蚂蚁经过的区域；AllowedCity[m][n] 存储可选择区域；第 i 只蚂蚁第 k 次寻找转移区域时，当前所在区域为 curcity = tabu[i][$k-1$]；length 存储一次迭代中得到的所有解的回路长度；GlbBestTour 存储全局最优解；IterBestTour 存储迭代最优解.

程序执行过程如下：

Step1：初始化参数. 在 CPU 中，对 distance[n][n]，trial[n][n]，Rate，tabu[m][n] 和 AllowedCity[m][n] 等参数进行初始化.

Step2：将数据传送到 GPU 中.

Step3：激活前 $n-2$ 次选择转移区域的内核，激活选择最后一个转移区域的内核，完成蚂蚁周游区域.

Step4：计算 m 条回路的长度，并写入记录回路长度的数组 length.

Step5：对 GPU 进行消减操作，从 length 中找到迭代最优解，将其写入数组 IterBestTour，并与全局最优解 GlbBestTour 进行比较. 如果优于全局最优解，则修改全局最优解.

Step6：使用迭代最优解更新信息素矩阵. 当迭

代次数超过一定值时，交替使用迭代最优解和全局最优解更新信息素矩阵.

Step7：清理禁忌表数组和允许区域数组，使蚂蚁回到出发的区域.

Step8：如果满足停止条件，则输出结果，程序终止；否则，转 Step3.

3.4.2　针对 GPU 的优化处理

为充分利用GPU的硬件优势，对程序流程进行以下改进处理：

1）采用多个内核完成算法的流程，每个内核执行算法的不同部分. 根据内核所处理的问题来设置其线程块网格维度和线程块维度. 比如更新信息素的内核只需一个线程块，从而可充分利用GPU的硬件资源，得到最大的加速比.

2）周游区域时，前 $n-2$ 次选择的内核与最后一次选择的内核不同，因为最后一个转移区域是确定的，不需要计算转移概率，降低了计算量.

3）更新信息素时，首先将信息素增量写入 $n\times n$ 的二维数组，然后执行网格维度为 n 的内核，每个线程块更新二维数组的一行，从而实现了设备内存读写的连续性.

4　实验与分析

本文采用 TSPLIB 中的算例. 实验环境为 Pentium Ⅳ 2.6 GHz CPU，768 M RAM，NVDIA GeForce 8600 GT 显卡的 PC 机，操作系统为 Windows XP. 实验数据和相关参数如表 1 所示.

表 1　TSP 问题及相关参数

问 题	m	n	α	β	ρ	迭代次数
pr76	50	76	1	2	0.98	1000
ch150	120	150	1	2	0.98	1000
tsp225	210	225	1	2	0.98	1000

分别采用 GPU 和 CPU 算法，对每个问题进行 100 次实验. 实验数据表明本文算法具有以下特点：

1）获得了较高的加速比. 表 2 数据显示了对不同问题进行 1000 次迭代时，本文算法对标准 ACO 算法的加速比. 当问题规模范围为 76 到 225 时，加速范围为 4.2 到 13.5 倍. 数据显示，问题规模越大，蚁群个体规模越大，加速效果越好. 对于要求更大规模蚁群的复杂问题，本文方法将取得更高的加速比.

对于改进的蚁群算法也进行了 GPU 加速的仿真实验. 表 3 数据显示了对于两种改进的蚁群算法，GPU 加速方法取得了较好的实验效果. 其中：DYNACO 是具有动态调整选择策略的改进型蚁群算法[11]，ADPACO 是自适应调节参数 ρ 的改进型蚁群算法[12].

表 2　GPU 算法相对于 CPU 算法的加速倍数

问 题	CPU 最优解	GPU 最优解	CPU 时间 /s	GPU 时间 /s	加速比
pr76	118917	118483	83.39	19.76	4.2
ch150	6722.72	6736.82	771.26	95.01	8.1
tsp225	4278.80	4227.39	2844.15	209.98	13.5

表 3　GPU 加速改进蚁群算法的实验数据

问 题	算　法	CPU 最优解	GPU 最优解
pr76	DYNACO	116472	116244
	ADPACO	116483	116153
pr150	DYNACO	6573.46	6594.31
	ADPACO	6536.57	6537.79
tsp225	DYNACO	4073.29	4068.62
	ADPACO	4033.40	4047.35
问 题	CPU 时间 /s	GPU 时间 /s	加速比
pr76	90.48	21.54	4.22
	90.57	21.46	4.20
ch150	815.70	103.47	7.88
	821.33	102.50	8.01
tsp225	3248.46	212.62	15.27
	3265.80	214.43	15.23

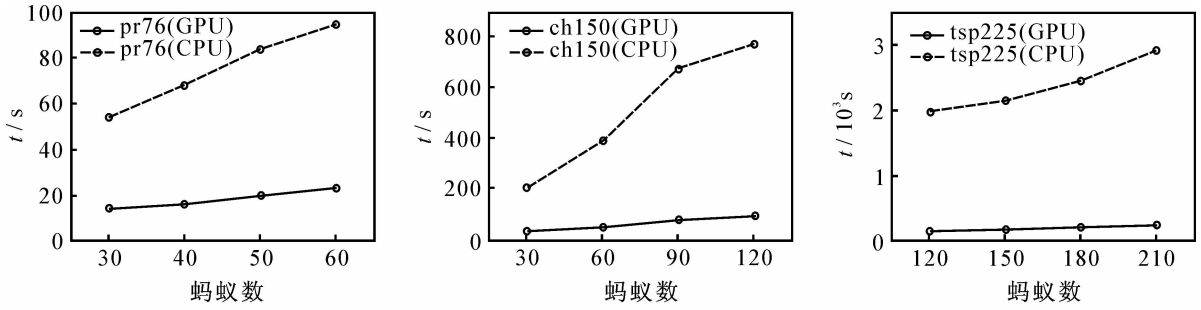
2）增大了细粒度并行的蚂蚁数量. 并行蚁群算法中蚂蚁的数量越多，全局搜索能力越强，但算法的计算量与蚂蚁数量 m 成正比. 对于一个规模为 n 的问题，$m=2n/3$ 是比较合适的. 当问题规模较大时，蚂蚁数量往往达到数百，启动如此多的进程，进程间的通信和管理将是相当大的消耗. 对于并行蚁群算法，目前多采用粗粒度并行的方法. 本文使用线程块来为蚂蚁构造解，即每个线程块相当于一个处理机，用于实现完全的细粒度并行. 当前 GPU 至少可承受几百个线程块，能满足较大规模问题求解的要求.

图 3 数据显示了进行 1000 次迭代时，蚂蚁规模对时间复杂度的影响. CPU 算法运行时间与蚂蚁数量近似为一种线性关系，而 GPU 算法运行时间与蚂蚁数最近似为一种次线性关系，运行时间随蚂蚁增长的速度远远小于 CPU 算法. 随着蚂蚁规模的增大，本文算法保持了较快的运算速度.

3）提高了算法的全局搜索能力，较好地保持了 FGACO 抑制早熟的特性. 当使用并行机实现时，子蚁群的规模较小，全局搜索能力比细粒度并行有所降低. 由于要在计算与通信开销之间寻找平衡，局部迭代若干次后才交换信息，使得早熟的概率增大. GPU 中线程拥有一致的地址空间，不需要显式的通信，每次迭代所有蚂蚁使用的是全局正反馈之后的信息素，从而降低了早熟的概率.

5　结　　论

本文提出一种基于 GPU 加速细粒度并行蚁群



(a) pr76 问题 (b) ch150 问题 (c) tsp225 问题

图 3 算法运行时间与蚂蚁规模的关系

算法. 该算法具有以下特点:1)利用 GPU 的高速浮点计算和并行特性,提高了算法的运算速度,取得了较高的加速比;2)增大了细粒度并行 ACO 的个体规模,算法执行时间与个体规模为次线性关系;3)较好地维持了群体多样性,保持了 FGACO 抑制早熟的特性;4)当前普通 PC 机显卡中大多配有 GPU 芯片,研究人员可用并行 ACO 算法来解决实际问题,避免了并行机等硬件环境对算法应用的限制.

进一步的研究工作包括:1)将 GPU 应用于其他改进的蚁群算法;2)使用 GPU 对其他并行优化算法进行处理.

参考文献(References)

[1] Colorni A , Dorigo M , Maniezzo V , et al. Distributed optimization by ant colonies[C]. Proc of 1st European Conf on Artificial Life. Paris, 1991: 134-142.

[2] Marcus Randall, Andrew Lewis. A parallel implementation of ant colony optimization [J]. J of Parallel and Distributed Computing, 2002, 62(9): 1421-1432.

[3] 于滨, 程春田, 杨忠振. 一种改进的粗粒度并行蚁群算法[J]. 系统工程与电子技术, 2006, 28(4): 626-629.
(Yu B, Cheng C T, Yang Z Z. Coarse-grain parallel ant colony optimization algorithm[J]. Systems Engineering and Electronics, 2006, 28(4): 626-629.)

[4] 萧蕴诗, 李炳宇, 吴启迪. 求解 TSP 问题的模式学习并行蚁群算法[J]. 控制与决策, 2004, 19(8): 885-888.
(Xiao Y S, Li B Y, Wu Q D. Parallel model-learning ant colony optimization algorithm for TSP[J]. Control and Decision, 2004, 19(8): 885-888.)

[5] Jowens J D, Luebke D, Govindaraju N. A survey of general purpose computation on graphics hardware[C]. Euro-Graphics 2005. Dublin, 2005: 21-51.

[6] 李建明, 迟忠先, 万单领. 一种基于 GPU 加速细粒度并行遗传算法的实现方法[J]. 控制与决策, 2008, 23(6): 697-704.
(Li J M, Chi Z X, Wan D L. A parallel genetic algorithm based on fine-grained model with GPU-accelerated[J]. Control and Decision, 2008, 23(6): 697-704.)

[7] Hong Li, Allison M. Parallel simulation for a fish schooling model on a general-purpose graphics processing unit [J]. Concurrency and Computation: Practice and Experience, 2009, 21(6): 725-735.

[8] Stutzle T, Hoos H. Max-Min ant system and local search for the traveling salesman problem[C]. Proc of IEEE Int Conf on Evolutionary Computation. Indianapolis: IEEE Press, 1997: 309-314.

[9] Thomas Stuzle, Holger H Hoos. Max-Min ant system [J]. Future Generation Computer Systems, 2000, 16(8): 889-914.

[10] Marco Dorigo, Eric Bonabeau. Ant algorithms and stigmergy[J]. Future Generation Computer Systems, 2000, 16(8): 851-871.

[11] 郑松, 侯迪波, 周泽魁. 动态调整选择策略的改进蚁群算法[J]. 控制与决策, 2008, 23(2): 225-228.
(Zheng S, Hou D B, Zhou Z K. Ant colony algorithm with dynamic transition probability[J]. Control and Decision , 2008, 23(2): 225-228.)

[12] 王颖, 谢剑英. 一种自适应蚁群算法及其仿真研究[J]. 系统仿真学报, 2002, 14(1): 31-33.
(Wang Y, Xie J Y. An adaptive ant colony optimization algorithm and simulation[J]. J of System Simulation, 2002, 14(1): 31-33.)