共享信息素矩阵：一种新的并行 ACO 方法

吕 强 [1, 2] 高彦明 [2] 钱培德 [1, 2]

摘 要 提出并实现了一种新的蚁群优化 (ACO) 并行化策略 SHOP(Sharing one pheromone matrix). 主要思想是基于多蚁群在解的构造过程和信息素更新过程中共享同一个信息素矩阵. 以 ACS 和 MMAS 的 SHOP 并行实现为例, 简要描述了 SHOP 设计思想和实现过程, 尝试了 ACS 和 MMAS 并行混合. 以对称 TSP 测试集为对象, 将 SHOP 的实现与相应串行算法在相同计算环境下的实验结果比较, 以及与现有的并行实现进行比较, 结果表明 SHOP 并行策略相对于串行 ACO 及现有的并行策略具有一定的优势.

关键词 蚁群优化, 并行, 共享信息素矩阵
中图分类号 TP31

Sharing One Pheromone Matrix: A New Approach to Parallel ACO

LU Qiang[1, 2] GAO Yan-Ming[2] QIAN Pei-De[1, 2]

Abstract This paper proposes and implements a new approach to parallel ant colony optimization (ACO) algorithms. The principal idea is to make multiple ant colonies share and utilize only one pheromone matrix. We call our approach sharing one pheromone matrix (SHOP). This paper briefly describes how to parallelize ACS and MMAS by SHOP strategy, and tries to hybridize these two in parallel. By tackling symmetric travelling salesman problems, this paper compares SHOP-ACO implementation with the relevant sequential ACO algorithms under fair computing environment, as well as with the existing parallel ACO algorithms. The experimental results indicate that SHOP strategy is superior to the sequential ACO algorithms and the existing parallel strategies.

Key words Ant colony optimization, parallelization, sharing one pheromone matrix

1 引言

蚁群优化 (ACO) 作为一种元启发 (Meta-heuristic) 式搜索策略, 在解各类组合优化问题时表现出了卓越的性能和效率. ACO 的生态属性及设计思想决定了它的并行研究是自然的方向. 原始的蚁群系统中, 通过群内多个蚂蚁的协作来完成任务. 蚁群算法与生俱来的并行性以及高性能计算平台的发展, 为蚁群算法的并行化研究提供了先天的优势.

元启发的并行策略有很多分类, 大致可划分为细粒度 (Fine-grained) 和粗粒度 (Coarse-grained) 两种. 对于细粒度并行策略, Bolondi 等针对 TSP 提出了一种方法, 运行环境为改进的连接机 (Connection machine) CM-2. 思想是为每只蚂蚁分配一个处理机, 在每次迭代中, 由主控进程生成并行构造路径的蚂蚁, 当所有蚂蚁完成一次搜索时进行同步, 并由主控进程选择此次迭代找到的最短路径来更新信息素. 实验结果表明, 每个蚂蚁的计算量很小, 处理器实际用于计算的时间很短, 所以大部分时间耗在了与主控进程的同步及信息交互方面. 事实上, 这种“并行”策略只是把多个 CPU 当作了提速的单个 CPU 来使用. Bullnheirmer 等[1] 的第一种并行策略显现了同样的负面效果. 因此这种细粒度并行方法对蚁群算法来说不太适用, 相反下面提到的大多数文献表明, 粗粒度并行能取到比较理想的结果.

为了降低处理器间信息交互的频度, Bullnheirmer 等在 [1] 中还提出了 PAPI (Partially asynchronous parallel implementation). 在 PAPI 中, 多个蚁群并行运行固定代数之后才发生信息素的交流, 从而减少了处理器间的通信, 获得了较好的加速比. Middendorf 等[2] 研究了蚁群间信息交流的几种方法并作出比较, 这种交流间隔亦为给定的代数. 方法包括：1) 共享所有蚁群当前找到的最好解；2) 只在局部范围内 (若干个而非所有蚁群) 共享局部当前最好解或者若干代内的最好解或者两者的组合使用. 实验表明几种方法中表现最好的是局部解的共享, 即把信息交互限制在局部的蚁群之间.

Stützle 在 [3] 中考虑了一种极端情况, 也是一种最简单的并行方法, 即并行的蚁群在求解过程中不进行任何信息交流. 它实现了 MMAS 的并行独立运行, 计算结果也体现了这一策略的高效性.

Talbi 等在 [4,5] 中将主从式并行蚁群算法应用于求解 QAP (Quadratic assignment problem) 问题, 主要思想是在每代循环中, 主处理器收集解以更新信息素矩阵, 并将其广播到所有从处理器, 从处理器构造解并将解送回主处理器. 实验平台为互连网相连的 10 个 SGI Indy 工作站, 编程工具为 PVM.

Randall 等[6] 提出了五种 ACO 的并行策略, 并且对小规模 TSP 实例用其中的一种策略实现了并行 ACS .

Chu 在 [7] 中实现了 ACS 的并行, 它将一个蚁群平均分为 p 个小的蚁群, 对应 p 个处理器, 并行求解. 每个蚁群有各自完全独立的数据结构, 运行完全相同的算法, 即采用数据并行的手段实现. 它除了 ACS 本身全局与局部的信息素更新之外, 每隔给定代数对所有 (或邻居) 蚁群得到的最好解即最短路径予以信息素增量. 文章从解的质量方面对实验结果进行比较, 但实验数据较少, 运行实例也比较小.

在共享存储架构的多处理机系统上实现 ACO 的并行在 [8] 中有所尝试, 采用的程序设计手段为 OpenMP, 应用问题为工业调度.

从以上概述可以总结出本文所关心的几个要点：

1) 粗粒度的并行策略好于细粒度的并行策略.

2) 目前所有的 ACO 并行方法, 每个蚁群都保存独立的私有的数据结构, 特别是无一例外地保存了各自的信息素矩阵.

3) 受到并行化元启发传统研究的影响, 并行化 ACO 的工作主要都侧重于并行搜索线程之间通讯交流模式的研究.

4) 大多数并行 ACO 的实验分析侧重于时间加速比, 忽略解的质量的比较.

5) 除了 Delisle 等的工作[8], 大多数工作都不是在共享存储体系构架上实现.

本文的思想是通过多个蚁群的并行协作来完成任务, 这些蚁群共享同一个信息素矩阵. 我们把本文的方法称为 SHOP(Sharing one pheromone matrix) 策略.

收稿日期 2005-12-26 收修改稿日期 2006-7-15
Received December 26, 2005; in revised form July 15, 2006
江苏省自然科学基金 (2003030) 资助
Supported by the Natural Science Foundation of Jiangsu Province of China (2003030)
1. 苏州大学计算机科学与技术学院 苏州 215006 2. 江苏省计算机信息处理技术重点实验室 苏州 215006
1. School of Computer Science and Technology, Soochow University, Suzhou 215006 2. Jiangsu Provincial Key Laboratory of Computer Information Processing Technology, Suzhou 215006
DOI: 10.1360/aas-007-0418

2 SHOP 的设计

2.1 指导思路

我们认为, 并行进程间的通信模式问题, 是所有并行化工作的共有问题, 不是 ACO 并行化的特有问题. 作为一种元启发, 并行 ACO 更多地应该考虑蚁群并行策略的问题. 信息素矩阵是 ACO 的核心数据结构, 它指导着向最优解的搜索方向. 之所以出现了多种多样的变形算法, 就是因为对信息素的处理没有特定的规则, 不断地构想出更加合理的信息素更新方式和更新时机, 就会不断地使蚁群算法得以改进. SHOP 仍然是以对信息素矩阵的考虑为出发点, 设计了让多个蚁群共享更新同一个信息素矩阵的方法. 正是由于这个特点, 本文基于的并行计算机模型是共享存储的多处理器系统. 针对这样的平台选择的编程环境为 Posix 标准线程库 PThread, 目标问题为典型组合优化问题 TSP.

SHOP 并行策略属于粗粒度结合主从并多搜索线程的并行. 我们设计主线程负责初始化和创建协调蚁群线程的工作, 每个蚁群线程并行地在独立的 CPU 上运行一个蚁群. 通过蚁群内蚂蚁的协作 (ACO 本身的机制) 和蚁群间的协作 (本文的新机制) 来解决 TSP 问题.

MMAS 和 ACS 是表现最好的两个 ACO 算法, 我们把对 MMAS 的 SHOP 并行称为 PMMAS, 对 ACS 的 SHOP 并行称为 PACS, 并行蚁群既有 MMAS 又有 ACS 的 SHOP 并行称为 MMACS. 下面简单描述这三者的设计.

2.2 信息素更新规则

对于涉及 MMAS 的蚁群, 共享的信息素下界 $\tau_{min} = \min_{i\in\mathcal{M}}\{\tau_{min}^{(i)}\}$, $\mathcal{M}$ 是并行的 MMAS 蚁群. 同理, 共享的信息素上界 $\tau_{max} = \max_{i\in\mathcal{M}}\{\tau_{max}^{(i)}\}$. 具体分述如下：

对于 PMMAS 来说, 我们对信息素区间 $[\tau_{min}, \tau_{max}]$ 的确定作出一些调整, τ_{min} 不变, 仍由表达式 $\tau_{max}/(2n)$ 决定, 而使 τ_{max}=$1/(\rho L_{cb})$, 其中 L_{cb} 为所有蚁群当前最短路径的长度. 理想情况下, 每个时刻均有 p 处理器只蚂蚁在同时执行上述过程, 而在串行算法中每个时刻只有一只蚂蚁执行解的构造过程. 当然这对 MMAS 来说没什么影响, 因为只要边上的信息量不变, 蚂蚁的行为就不会受到影响, 而 MMAS 中只有所有蚂蚁都求得一条可行路径之后才进行信息素更新. 不过这对于 PACS 的实现而言, 却有很重要的作用.

对于 PACS 来说, 信息素更新的规则不需要任何调整. 这里的方法相对于 [7] 中的并行策略, 蚁群间的宏观反馈信息更加实时有效, 因为一个蚁群中的蚂蚁修改了某条边上的信息素, 其它蚁群中的蚂蚁会立刻感知到这种变化, 因此在前进路线的选择上也会及时作出调整, 甚至有可能改变单蚁群中预期的目标节点. 这种扰动究竟是正面还是负面影响了向最优解的搜索方向, 目前理论上无法给出答案. 但有一点可以肯定的是, 所有蚁群的行为总是朝着好的方向进行的, 因为这符合生物蚁群的自然行为特征. 所以在各种参数及环境相同的情况下, 并行实现的结果不会比串行差. 但这种 PACS 的实现中, 由于局部信息素的更新需要非常频繁地访问信息素矩阵, 因此对共享信息素矩阵会有过多的同步而影响算法的运行效率.

对于 MMACS 来说, 根据 MMAS 的规定, 必有 $\tau_{ij} \in P_1 = [\tau_{min}, \tau_{max}]$, 其中 $\tau_{max} = 1/(\rho L_{bs})$ 和 $\tau_{min} = \tau_{max}/(2n)$. 而初始化值 $\tau_0 = 1/(\rho L_{best})$. 对于 ACS 来说, $\tau_{ij} \in P_2 = [\tau_0, 1/L_{bs}]$, 其中 $\tau_0 = 1/(nL_{best})$, L_{bs} 随着算法的执行会动态改变, 并且由于解是趋优的, 所以 $1/L_{bs}$ 是递增的, 但 L_{bs} 大于等于最短路径的长度 L_{opt}, 所以 P_2 的上界最大为 $1/L_{opt}$, 而 P_1 的上界最大为 $1/(\rho L_{opt})$. 又 ρ 为一个很小的正实数, 即有 $\tau_{min} > \tau_0$ 和 $\tau_{max} > 1/L_{bs}$, 所以我们在 MMACS 中取 $\tau_{ij} \in P_1 \cap P_2 = [\tau_{min}, 1/L_{bs}]$, 并且这里对各边的信息量初始值设置为 τ_{max}, 即 $\tau_0 = \tau_{max} = 1/(\rho L_{bs})$, 其中的 ρ 取 MMAS 中的设置, n 为问题中状态节点的个数. MMAS 与 ACS 中关于蒸发因子 ρ 均采用各自串行算法的设置. 这样我们把 MMAS 和 ACS 不同信息素更新混合起来.

2.3 对信息素矩阵更新的同步

对于共享信息素矩阵的读访问, 不需要任何同步操作, 而对于写更新, SHOP 必须实现同步机制. 事实上, 并行蚁群的全局或局部更新, 都可以用 $\tau_{ij} = (1-\rho)\tau'_{ij} + \rho\cdot\Delta\tau_{ij}$ 来表示. 从设计的角度, 可以采用以下同步方案.

1) 最简单的粗粒度的同步：每当进行更新操作时, 用同一个互斥信号量上锁, 从而完成对整个信息素矩阵互斥访问.

2) 最简单的细粒度的同步：对信息素矩阵 $[\tau_{ij}]$ 的每一个元素使用一个信号量, 当进行更新操作时, 锁住相应的信号量.

3) 综合的同步：当局部更新时, 采用细粒度的同步方案; 当全局更新时采用粗粒度的同步方案.

需要说明的是, 并行蚁群对信息素的更新时刻, 是可以被有意地设计成错开的. 首先由于局部优化的引入, 使得并行蚁群每一代的完成时间不可预计, 所以, 信息素全局更新的时刻是无意间被打散了. 其次, 对于信息素局部更新的时刻, 由于蚂蚁初始位置设置的随机性, 再加上可以通过有意错开蚂蚁的出发时间的手段, 使得每个蚂蚁需要进行局部更新的时刻可以被错开.

因为多个蚁群共享同一个信息素矩阵, 因此多个线程对其访问时必须采取安全的互斥机制, 本文中使用传统的加锁方法来满足这种需求. 具体到 PThread 平台, 我们设置一个锁变量, 通过调用 pthread_mutext_lock 和 pthread _mutex _unlock 函数对, 来实现多蚁群对信息素矩阵的互斥访问. 从这一点来看, SHOP 在协调诸线程方面花费了许多额外的代价, 但是具体的定量分析有待于进一步的研究.

2.4 SHOP 的创新

与现有的并行蚁群拥有独立信息素矩阵的信息交流方式相比, SHOP 策略其实是一种隐式的信息交流方式, 对宏观蚁群间的反馈信息利用得更加充分. 因为任何一只蚂蚁选择下一个城市节点的动态依据只有两个节点间的信息量 (启发函数值是固定的), 所以当一个蚁群把自己的进化结果反映到某条边上的信息量变化时, 即修改了信息素矩阵之后, 其它蚁群马上就会感知到, 从而影响其它蚁群中蚂蚁的前进路线. 而这种影响通常情况下都是积极的, 并且更加实时有效. 总的来说, 也就是把蚁群间的知识交流融入到对共享信息素矩阵的更新之中. 从某种角度来说, SHOP 为只提供全局信息素更新的 ACO 算法 (例如 MMAS) 增加了局部更新机制.

信息素的重新初始化在 MMAS 是一个突破性的尝试, 思想是当超过足够多迭代次数, 例如 250 次, 仍未发现更好解, 并且收敛性估计变量分支因子 (Branching-factor) 小于某个设定值时, 则认为算法已经收敛, 其中分支因子反映了从实际问题抽象出来的图中各边上的信息素集中情况, 其值越小说明信息素越集中到少量的边上. 此时记录下当前最好解, 并将所有边上的信息量重新初始化, 不过现在的初始值不再是 τ_0, 尽管仍然为 τ_{max}, 但当前最好解的改变使得 τ_{max}

的值变大了, 它相当于算法从更有利的初始状态重新开始运行, 以损失小概率找到更好解的代价来换取一次新的求解过程, 期望能找到更优质的解, 这样就增大了对解空间的搜索覆盖度, 使算法有更多的机会找到更好的解, 从而增强了算法整体上发现最优的能力.

对 SHOP 并行蚁群而言, 这种机制同样奏效. 我们规定当所有蚁群均满足上述条件, 例如迭代次数达到 250 次和分支因子小于某个值, 记录下每个蚁群和所有蚁群中的当前最好解, 将信息素矩阵重新初始化为 τ_{max}. 初始化之后, 各个蚁群又重新开始按前面的步骤构造解及更新信息素. 对于多个蚁群而言, 任一蚁群找到更好解都会使最终结果得到改善, 因此最终解的质量得到改善的可能性就会更大一些, 并且在给定的计算时间内, 这种重新初始化一直是允许的.

3 实验结果及分析

3.1 总体说明

对蚁群算法来说, 不能只用加速比来衡量并行 ACO 算法与串行 ACO 算法孰优孰劣, 因为蚁群算法作为一种启发式算法, 串行等价性都不能保证获得, 所以还要同时考察解的平均质量等诸多因素. 我们这里在比较并行 ACO 与串行 ACO 时, 取它们各自运行 $k = 10$ 次后的平均解的质量, 运行时间分别为 t_{max}/p 和 t_{max}, 其中 p 为蚁群数即 CPU 数, t_{max} 为给定的串行算法最大运行时间. 这样, 我们把并行算法和串行算法的时间代价统一到公平的标准. 同步策略采用的是最简单的粗粒度同步. 从花费计算资源的角度, 如果加上同步的代价, 按照本文的实验设计, 串行算法比并行算法获得了更多的包括 CPU 和内存在内的其它计算资源.

本文选择的运行环境为 IBM pServer, 带有 2 个 power550 1.5GHz 处理器, 6.0GB 内存, 操作系统为 AIX5.3, 程序设计工具为 PThread+gcc. 所以并行蚁群数目 $p = 2$. 我们对从 TSPLIB[9] 中随机抽取的 15 个典型实例进行了计算, 实例规模从 318(lin318) 到 5915(rl5915) 个城市. 除非特殊说明, 所有计算使用到的参数如下: $\alpha = 1$, $\beta = 2$, $\rho = 0.2$, $q_0 = 0.98$, $\lambda_{branch} = 1$.

很多研究表明对于 TSP, 最有效的局部优化方案是 3-opt, 下面的计算对于并行和串行 ACO 算法都使用了 3-opt. 由于 SHOP 的要点在于对蚁群行为的并行, 所以下面的结论, 对于是否使用了局部优化方案, 或者使用哪一种局部优化方案, 都没有影响. 简单起见, 下面的计算结果都是基于粗粒度的同步控制方案.

于是, 在这样的实现中, SHOP-ACO 算法求解对称 TSP 问题时, 空间复杂度为 $O(n^2)$, 时间复杂度仍为 $O(n^3)$, n 是城市数目.

3.2 与串行 ACO 算法的比较

我们用 Best, Worst, Sol_{avg} 和 Sol_{std} 分别表示所有 k 次运行中返回的值中的最好值、最差值、平均值和标准差; e_b, e_{avg}, e_{wst} 分别表示 Best、Sol_{avg} 和 Worst 指标超过最优解的比例; Iter_{avg} 和 Iter_{std} 分别表示获得 Best 时的平均进化代数及其标准差; t_{avg} 和 t_{std} 分别表示获得 Best 时的平均运行时间及其标准差; t_{tol} 和 t_{stdtol} 分别表示所有 k 次运行结束的平均运行时间及其标准差. 由于篇幅限制, 我们在这里选择分析一个典型实例 pr2392 的计算结果 (串行蚁群蚂蚁个数 50 个, 并行蚁群蚂蚁个数 25 个), 计算结果见表 1.

从表 1 中可以看到, PMMAS, PACS 和 MMACS 都获得了最优解, 而相应的串行算法 SMMAS 和 SACS 都没有获得. 注意到 SMMAS1, SACS1 的最大允许运行时间与并行算法一样, 而 SMMAS2 和 SACS2 的最大允许运行时间是相应并行算法的两倍. 串行算法还是没有能够找到最优解, 说明串行和并行的搜索行为不一样了. 从 e_{avg} 和 e_{wst} 的指标来看, 说明并行的搜索空间比串行的大. 而从 t_{tol} 的指标来看, 并行并未全部用完指定的时间, 而串行都用完了指定的时间, 所以, SHOP 的搜索效率提高了.

结论是不管 SHOP 的并行蚁群是同质的 (都是 MMAS 或都是 ACS), 还是异质的 (一个 MMAS , 另一个 ACS), SHOP-ACO 有着与串行 ACO 不同的行为, 从而导致不同的性能.

3.3 并行 ACO 的横向比较

本节将 SHOP-ACO 算法与 IMMAS[3], 以及 ChuPACS[7] 相比较, 目标数据全部采自相关的论文.

IMMAS 将 MMAS 算法独立运行在多个处理器上. 而 PMMAS 则是对 MMAS 采用了 SHOP 并行策略, 所以, 它们都是对同一种 ACO 算法的不同并行策略. 除了并行蚁群的个数外 (PMMAS 的设计是让所有并行蚁群的个数总和等于串行蚁群; 而 IMMAS 则是让每个并行蚁群的蚂蚁个数等于串行蚁群), 我们对 PMMAS 的运行参数设置为与 IMMAS 给出的完全相同, 比较结果如表 2 所示.

表 1 SHOP-ACO 与串行 ACO 在 pr2392 的实验比较

Table 1 Comparison between SHOP-ACO and sequential ACO algorithms on pr2392

Algorithm	Iter_{avg}	Sol_{std}	Iter_{std}	t_{avg}	t_{std}	t_{tol}	t_{stdtol}	e_b	e_{avg}	e_{wst}
PMMAS	474.2	1256.42	167.02	447.49	155.34	450.78	156.42	0	0.005411	0.012417
SMMAS1	722.6	599.85	154.33	271.68	44.69	500.19	0.14	0.000251	0.003131	0.005116
SMMAS2	1656.1	281.49	578.5	572.27	189.66	1000.18	0.12	0.000286	0.001878	0.003002
PACS	288	2190.87	135.25	313.82	142.39	450.79	157.33	0	0.012776	0.019816
SACS1	1143.4	518.74	272.06	391.93	93.11	500.16	0.1	0.000635	0.002961	0.005277
SACS2	1175.7	449.75	411.39	627.93	218.69	1000.18	0.11	0.022890	0.024859	0.026527
MMACS	521.3	545.66	219.51	346.47	140.93	450.56	156.99	0	0.002434	0.0041

表 2 PMMAS 与 IMMAS 实验比较
Table 2 Comparison between PMMAS and IMMAS

Instance	Algorithm	Sol_{avg}	$Iter_{avg}$	Best	Worst
d198	PMMAS	15780.6	412.4	15780	15781
	IMMAS	15780.3	236.1	15780	15780.3
lin318	PMMAS	42029	435.1	42029	42029
	IMMAS	42029	494.2	42029	42029
pcb442	PMMAS	50859.3	900	50778	50912
	IMMAS	50886.5	1164.5	50785	50912
att532	PMMAS	27701.8	267.1	27686	27706
	IMMAS	27707.4	561.2	27703	27728
rat783	PMMAS	8811	509.2	8806	8817
	IMMAS	8811.5	878.2	8806	8821
pcb1173	PMMAS	56904.1	1685.4	56892	56930
	IMMAS	56960.3	1837.4	56892	57091
d1291	PMMAS	50812.4	1015	50801	50820
	IMMAS	50845.8	1054.1	50801	50909

在表 2 中, 由于计算平台的差异, 绝对的计算时间比较没有意义. 但是我们让 PMMAS 的平均进化代数 $Iter_{avg}$ 小于 IMMAS 的进化代数, 这样, PMMAS 没有比 IMMAS 获得更多的计算资源. 即使如此, 由表 2 可知, PMMAS 的各项指标都优于 IMMAS.

我们接着比较 PACS 和 ChuPACS. ChuPACS 的目标问题规模太小, 以至于 PACS 都能够在 2 秒钟之内给出相同答案, 表 3 就是相应的计算结果比较, 其中所有关于 ACS 的参数, PACS 都与 ChuACS 取相同设置.

表 3 PACS 和 ChuPACS 实验比较
Table 3 Comparison between PACS & ChuPACS

Instance	Algorithm	Sol_{avg}	Best	Worst
st70	PACS	675	675	675
	ChuPACS		677/677/678	
eil101	PACS	629	629	629
	ChuPACS		646/648/645	
tsp225	PACS	3916	3916	3916
	ChuPACS		3939/3934/3916	

表 3 中 ChuPACS 的三个计算结果是相应于 ChuPACS 的三种不同信息素交流模型, 而 PACS 都无一例外地迅速收敛到最优解. 表 3 表明, PACS 全面优于 ChuPACS.

4 结语

SHOP 提出了一种简单有效的并行 ACO 的策略. 由于共享信息素矩阵导致的同步开销将随着并行处理器数目的增大而增大, 所以对于更多的并行处理器, SHOP 的性能如何, 将是一个新的问题. 但是 SHOP 并行策略在定性分析和实验中都表现出了能够改善串行搜索能力, 这一点对于元启发算法来说, 总是有价值的. 我们需要更加多的数学分析和实验来研究平衡同步开销和搜索能力的增强等问题. 另外, 我们也将 SHOP-ACO 策略实现和应用到更多的组合优化问题中, 从而验证其更加广泛的适用性.

致谢

本文关于 MMAS 和 ACS 的计算结果来自于 Thomas Stützle 的软件包 ACOTSP[10]. PMMAS、PACS 和 MMACS 均以该软件包为基础修改. 在此表示感谢.

References

1 Bullnheimer B, Kotsis G, Strauss C. Parallelization strategies for the ant system. *Applied Optimization*, 1998, **24**: 87∼100

2 Middendorf M, Reischle F, Schmeck H. Multi colony ant algorithms. *Journal of Heuristics*, 2002, **8**(3): 305∼320

3 Stützle T. Parallelization strategies for ant colony optimization. In: Proceedings of 5th International Conference on Parallel Problem Solving for Nature. 1998, 722∼731

4 Talbi E G, Roux O, Fonlupt C, Robillard D. Parallel ant colonies for combinatorial optimization problems. *Lecture Notes in Computer Science*, 1999, Springer, **1586**: 239∼247

5 Talbi E G, Roux O, Fonlupt C, Robillard D. Parallel ant colonies for the quadratic assignment problem. *Future Generation Computer Systems*, 2001, **17**(4): 441∼449

6 Randall M, Lewis A. A parallel implementation of ant colony optimization. *Journal of Parallel and Distributed Computing*, 2002, **62**(9): 1421∼1432

7 Chu S C, Roddick J F, Pan J S, Su C J. Parallel ant colony systems. International Symposium on Methodologies for Intelligent Systems. *Lecture Notes in Computer Science*, 2003, Springer, **2871**: 279∼284

8 Delisle P, Krajecki M, Gravel M, Gagné C. Parallel implementation of an ant colony optimization metaheuristic with openmp. In: Proceedings of the 3rd European Workshop on OpenMP. 2001, Barcelone, Espagne, IEEE, 2001. 79∼84

9 Reinelt G. TSPLIB[Online], available: http://www.iwr. uniheidelberg.de /groups /comopt /software/TSPLIB95, Dec. 1, 2005

10 Stützle T. ACOTSP[Online], available: http:// iridia . ulb. ac. be /mdorigo /ACO /downloads / ACOTSP.V1.0. tar. gz, Dec. 1, 2005

吕 强 教授, 研究方向包括并行分布计算, 信息系统, 中文信息处理. 本文通信作者. Email: qiang@suda.edu.cn
(**LU Qiang** Professor in School of Computer Science and Technology, Soochow University. His research interest covers distributed/parallel computing, information systems, and Chinese information processing. Corresponding author of this paper.)

高彦明 客座助研, 研究领域为并行分布计算.
E-mail: ymgao@zhhz.org
(**GAO Yan-Ming** Visiting research assistant in Jiangsu Provincial Key Lab of Computer Information Processing Technology. His research interest covers distributed/parallel computing.)

钱培德 教授, 研究方向包括并行分布计算, 操作系统, 中文信息处理.
Email: pdqian@suda.edu.cn
(**Qian Pei-De** Professor in School of Computer Science and Technology, Soochow University. His research interest covers distributed/parallel computing, operating system, and Chinese information processing.)