文章编号: 1671-6833(2021)06-0034-08

求解加权 MTSP 问题的 CUDA 并行群智能方法

苏守宝[1,2], 赵 威[1,2], 李 智[1,2]

(1.江苏科技大学 计算机学院,江苏 镇江 212003; 2.金陵科技学院 数据科学与智慧软件江苏省重点实验室,江苏 南京 211169)

摘 要: 针对混合迭代算法执行时间长的问题,根据粒子群优化(PSO)算法和蚁群优化(ACO)算法的并行特点,结合其在 GPU 上并行化实现技术和编程优化技巧,提出一种基于 CUDA 的粒子群聚类蚁群的并行群智能混合方法 GPSO-AC。该算法利用 GPU 的多个流处理器(SM)和单指令多线程(SIMT)的指令架构,将 GPSO-AC 算法在运行中的独立个体的搜索过程同时并行执行,在保证算法精度的基础上,加快混合迭代法的执行速度。考虑到实际场景中旅行商在每个路段上各项开销不同,可以抽象为每段路程区间上都有一个与之对应的代价,将路程代价考虑到 MTSP 问题中。采用 TSPLIB 库中 6 个测试数据集,将 GPSO-AC 与 PSO-AC、TPHA、*K*-means-AC 等算法进行比较,并进一步探讨了加入代价均衡约束后对加权 MTSP 问题最优解收敛性能的影响。使用 chn31 数据集上不同旅行商数时,GPSO-AC 在不考虑代价均衡、代价均衡约束、加权代价均衡的情况下的代价标准差分别为 1 165.26、54.97、6.74。结果表明: 在求解一般 MTSP 问题及其衍生加权、代价均衡 MSTP 问题上,GPSO-AC 在执行速度和收敛精度上均优于 CPU 串行算法,且随着模型规模增加,其速度优势更加明显。

关键词: 多旅行商问题; CUDA 并行算法; 代价均衡; 粒子群聚类; 蚁群算法

中图分类号: TP18; TP303.6 **文献标志码**: A **doi**: 10.13705/j.issn.1671-6833.2021.04.009

0 引言

多旅行商(MTSP)问题是旅行商(TSP)问题的拓展,属于 NP-hard 组合优化问题,即 m 个旅行商前往 n 个城市销售商品,每个城市有且只能有一个旅行商经过,要求所有旅行商经过的总距离之和最短。寿涛等[1]通过 Delaunay 三角剖分及树分解算法,将 MTSP 问题转化为多个 TSP 问题。刘楠[2]采用哈密顿圈分割覆盖方法,设计了求解 MTSP 问题的近似精确解法。这些算法性能稳定、时间复杂度小,但只能解决小规模 MTSP 问题。于是,启发式算法得到了广泛应用,主要包括混合迭代法、分治法、协同进化策略等。Trigui 等[3]将多机器人任务分配(MRTA)看作 MTSP 的实例,同时优化最大行驶距离和总行驶距离,问题成本是这 2 个指标的组合,以简化为单一目标优化问题; 张美燕等[4]将自主式水下潜器(AUV)的能量消耗和能量均衡作为 MTSP 问题路径上的代价,运用 MTSP-GA 算法模型在水下三维空间内进行 AUV 路径规划。在确保工作负载均衡的情况下,有研究者提出了 MTSP 问题的两阶段启发式算法 TPHA,通过改进的 *K*-means 分组确保城市数量均衡,再用改进的遗传算法(GA)求解各聚类城市点 TSP 问题[5-7]。也有学者使用分治策略(如单亲 GA)将 MTSP 问题分组,再用 ACO 算法逐个求解 TSP 问题[8-10],如在 GA 迭代过程中加入 ACO 算法作为算法收敛约束,则将 GA 改为单亲 GA 以提高算法速度[11-12]。分治策略算法之间无交互,因而简化了问题的复杂度,但算法鲁棒性不强,尤其是大规模数据仍面临搜索空间指数增长和维数灾难问题[13-14]。

针对混合迭代法算法精度高但执行时间长的问题,本文基于并行软硬件体系 CUDA 运算平台,提出一种基于 GPU 的混合粒子群聚类蚁群的并行算法,在保证算法精度的同时,提高了混合迭代法的执行速度。由于实际场景中旅行商在每个

收稿日期: 2021-02-03; **修订日期**: 2021-05-09

基金项目: 国家自然科学基金资助项目(61375121, 41801303); 金陵科技学院高层次引进人才科研项目(jit-rcyj-201505)

作者简介: 苏守宝(1965—),男,安徽六安人,金陵科技学院教授,博士,主要从事群智能大数据挖掘的研究,E-mail: sushowbo@163.com。

路段上开销不同,将其抽象为每段路程区间上都有一个与之对应的代价,将路程代价考虑到 MTSP 问题中。

1　相关研究

1.1　加权 MTSP 问题建模

单点出发的加权 MTSP 问题: m 个旅行商(记为 $b_1,b_2,\cdots,b_m$) 前往 n 个城市(记为 $T_1,T_2,\cdots,T_n$) 售货,所有旅行商从同一个城市 T_1 出发并最终在城市 T_1 会合,要求每个城市有且只能有 1 个旅行商经过。考虑到每条路段的开销不同,记连接 i、j 两地的路段上的单位代价为 v_{ij},则 i、j 两地旅行代价 C_{ij} 为

$$C_{ij} = d_{ij} \cdot v_{ij}。\tag{1}$$

求所有旅行商总代价和最短路线方案,目标函数为

$$\min Z = \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}v_{ij}p_{ij}^{k}。\tag{2}$$

其中,p_{ij}^{k} 为旅行商 k 在 i、j 两地的方向标记:

$$p_{ij}^{k} = \begin{cases}1, & \text{旅行商 } k \text{ 从 } i \text{ 前往 } j;\\ 0, & \text{旅行商 } k \text{ 从 } j \text{ 前往 } i。\end{cases}\tag{3}$$

1.2　粒子群 *K*-means 聚类算法

粒子群 K-means 聚类是将聚类中心设为粒子的位置 $\boldsymbol{X}$,维度是聚类个数,则粒子群 K-means 聚类粒子最佳位置即为聚类最优解。将 N 个样本聚类到 K 个类中,需要满足:

$$\min J = \sum_{j=1}^{K}\sum_{i=1}^{N}\omega_{ij}\sum_{p=1}(\boldsymbol{X}_{ip} - C_{jp})^2。\tag{4}$$

聚类中心更新公式为

$$C_{jp} = \frac{\sum_{i=1}^{N}\omega_{ij}\boldsymbol{X}_{ip}}{\sum_{i=1}^{N}\omega_{ij}}。\tag{5}$$

式中: $\omega_{ij} = \begin{cases}1, & \text{该样本属于 } J \text{ 类;}\\ 0, & \text{该样本不属于 } J \text{ 类。}\end{cases}$

粒子群优化(particle swarm optimization, PSO) 算法中每个粒子根据自身经验数据和全局经验数据利用迭代来逼近全局最优解。每个粒子用如下计算式更新自己的聚类中心坐标和速度[6-7]:

$$V_i(t+1) = wV_i(t) + C_1 \cdot \text{rand}() \cdot (P_{besti}(t) - X_i(t)) + C_2 \cdot \text{rand}() \cdot (G_{best}(t) - X_i(t));\tag{6}$$

$$X_i(t+1) = X_i(t) + V_i(t+1)。\tag{7}$$

式中: P_{besti} 为第 i 个粒子所经历过的最优位置; G_{best} 为目前全局最优位置; 粒子的运动用位置 X 和速度 V 表示; w 为惯性因子,其值为(0,1),影响着粒子的全局搜索能力,w 越大粒子全局搜索能力越强,反之越弱; rand() 为(0,1) 的随机值; C_1 和 C_2 通常取值在 2 附近。粒子群 K-means 聚类算法流程如图 1 所示。

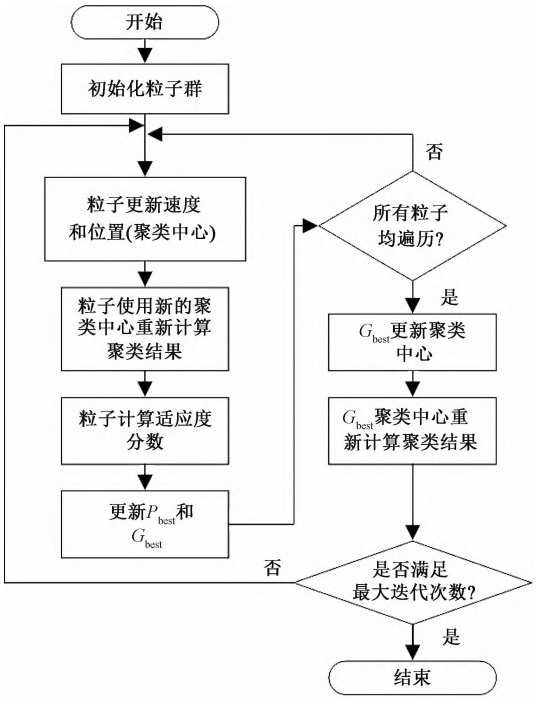

图 1　粒子群 *K*-means 聚类算法流程图

Figure 1　Flow chart of PSO-based *K*-means method

1.3　蚁群优化算法

蚁群优化(ant colony optimization, ACO) 算法是一种模拟蚂蚁觅食行为的群智能算法,具有可并行执行、易于与其他算法混合、较强鲁棒性等特点[8-9]。蚁群优化算法求解 TSP 问题时,初始化各个城市间的信息素浓度,在时刻 t 蚂蚁 k 从当前城市 i 转移到下一个城市 j 的概率为

$$P_{ij}^{k}(t) = \begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{s\in J_k(i)}[\tau_{is}(t)]^{\alpha}[\eta_{is}]^{\beta}}, & j \in J_k(i);\\ 0, & j \notin J_k(i)。\end{cases}\tag{8}$$

式中: α 为信息素因子,表示各个路径被选择的比重大小; β 为启发式因子,表示启发函数 η_{ij} 的重要程度。蚂蚁 k 周游一周后形成路径,计算路径长度更新各边信息素。当所有蚂蚁都走完后,由最短路径对信息素和禁忌表进行更新:

$$\tau_{ij}(t+n) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij},$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m}\Delta\tau_{ij}^{k},\Delta\tau_{ij}^{k} = \begin{cases}\dfrac{Q}{L_k}, & \text{蚂蚁 } k \text{ 经过 } i、j;\\ 0, & \text{其他。}\end{cases}\tag{9}$$

式中: $\rho \in (0,1)$ 为信息素挥发系数; Q 为信息素

总量;L_k 为蚂蚁 k 周游一次的路径长度。

2 本文方法

为了利用 CUDA 实现并行计算,先用粒子群 K-means 算法将各个城市点按旅行商个数进行分类,再使用蚁群算法进行分类评估,n 个粒子搜索最优解过程中有 n 个蚁群参与迭代,二者混合迭代过程中会出现大量独立运算过程,即粒子、蚁群间和蚁群内各个蚂蚁间的独立运算。这些独立运算过程中使用的数据重新排列为向量化数据,GPU 的流处理器单个时钟周期内同时调度多个数据,如果向量的元素彼此独立则可以并行计算向量,这也与 GPU 利用 SIMT 指令架构加速原理是一致的[15],因此,提出一种基于 CUDA 的并行群智能方法,记为 GPSO-AC。

2.1 GPSO-AC 算法设计

GPSO-AC 算法前半部分采用基于粒子群的 K-means 算法,聚类中心为粒子位置,维度为聚类个数,粒子速度为聚类中心的偏移量。由于 K-means算法初始聚类中心对结果影响很大,选择合适的初始聚类中心可以加快算法收敛速度和提升聚类质量。本文采用粗分类方法对初始城市分类,则从各个城市前往其他城市的所有代价和全局城市路段互通的代价总和分别为

$$R_i = \sum_{j=0}^{n} d_{ij}v_{ij}, \quad i = 0,1,\cdots,n; \tag{10}$$

$$R = \sum_{i=0}^{n}\sum_{j=0}^{n} d_{ij}v_{ij}, \quad i \neq j。 \tag{11}$$

式中:n 为城市数量;v_{ij} 为城市 i 和 j 之间的单位代价。

将城市到聚类中心距离排序,再依次遍历,若遍历条件满足式(12),则将已经遍历的城市归为一类,按照式(5)更新聚类中心。

$$\sum_{i=0}^{n} f_{ik}R_i \leqslant \frac{R}{m}。 \tag{12}$$

式中:m 为旅行商个数即聚类个数;f 为布尔标记,如果旅行商 k 经过城市 i 则为 1,反之为 0。

在粒子群迭代流程中,所有重新分类的过程均使用式(12)进行约束,目的是确保每类城市之间遍历代价总和大致均衡。接着细分类,计算适应度,使用 ACO 算法计算遍历某一类城市的最小代价,使用式(9)更新信息素和禁忌表,得出的最小代价为 C_k。当前粒子的适应度为

$$fitness = \frac{C_k}{\sum_{i=0}^{t} C_k} + q\frac{S(C_k)}{\sum_{i=0}^{t} S(C_k)}。 \tag{13}$$

式中:t 为粒子个数;$S(\cdot)$ 为计算方差的函数;$S(C_k)$ 表示聚类结果之间的离散程度,$S(C_k)$ 越小则各类之间遍历代价越均衡;$q \in [0,1]$ 为调谐系数,当 $q=0$ 时,表示不考虑代价均衡,只计算最低代价,可根据实际问题调整取值。

GPSO-AC 算法如下:

```
初始化粒子群
do
  parallel_for 粒子 in 粒子群;
    parallel{更新位置和速度,使用式(5)~(7)};
    parallel {  // 重分类
      for 聚类中心 in 所有聚类结果;
        计算样本到当前聚类中心距离;
        sort 样本 by 距离;
        聚类中心选择样本,使用式(4);
      end
    }
    parallel {  // 计算适应度
      for 聚类中心 in 所有聚类结果;
        初始化蚁群
        do
          parallel_for 蚂蚁 in 蚁群;
            parallel{搜索路径,使用式(8)};
            parallel{更新信息,使用式(9)};
        end
        while 达到最大迭代次数
        计算最优路径适应度,使用式(10)~(13);
      end
    }
    parallel{更新个体最优};
    parallel{更新群体最优};
  end
  while 达到最大迭代次数。
```

2.2 GPU 并行算法实现与优化

2.2.1 算法实现

在 CUDA 架构中,将算法拆解成多个并行算子,单个并行算子称为线程(thread),多个线程组成 1 个线程块(block)。根据粒子群算法固有的并行特点,可将图 1 中的算法流程进行拓展,如图 2所示。

GPU 的 1 个 SM 上有 32 个 CUDA 核心,每 32 个线程组成 1 个线程束(warp),CUDA 核心只能以线程束作为基本单元执行指令,所以 1 个 block 内的线程束必须为 32 的倍数。在设计 kernel 函

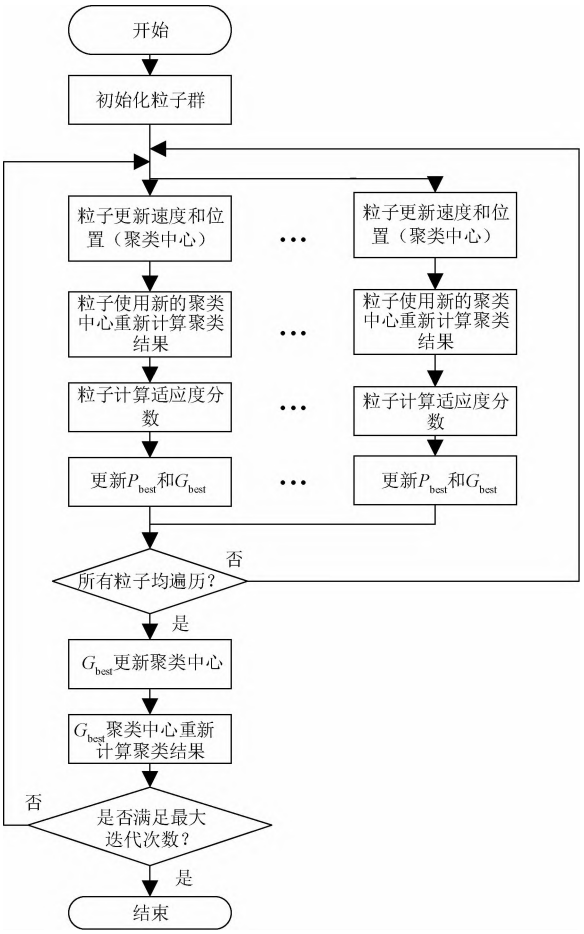

图 2　基于 GPU 的粒子群聚类算法流程图

Figure 2　Flowchart of PSO clustering based on GPU

数时，线程块数通过计算得到，例如假定每个 block 内线程数为 32，设粒子数即并行线程数 op-size 为 O，则线程块数 $B=(O+32-1)/32$。考虑到 GPU 使用 SIMT 指令架构，线程内指令相似度越高其执行速度越快，如果线程内算法流程过于复杂导致指令异化，那么这时一个线程束中的线程串行运行在 CUDA 核心上，一个线程采用多个 kernel 函数分解。

在更新 G_{best} 前，粒子间互不干扰，线程独立运行; 更新 G_{best} 时，线程间有数据交换。CUDA 上无锁机制，但是有同步原语 syncthreads( ) 函数，同步实际上是将线程串行运行，时间复杂度为 $O(N)$。使用并行规约算法求解 G_{best}，规约算法对传入的 N 个数据，使用一个二元的符合结合律的操作符 $\oplus$，生成 1 个结果。求 G_{best} 的规约可表示为 $G_{best}=P_{best}(1)\oplus P_{best}(2)\oplus P_{best}(3)\oplus\cdots\oplus P_{best}(N)$。首先把粒子所有的 P_{best} 数据放至共享内存中并编号，之后用 1 个线程计算前 2 个粒子的 P_{best}，再用 1 个线程计算中间 2 次结果，以此迭代最终求得 G_{best}，时间复杂度为 $O(\lg N)$。图 3 为并行规约算法的示意图。

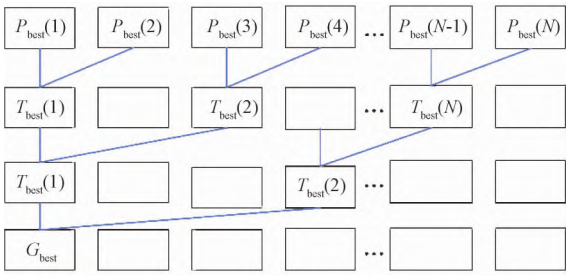

图 3　并行规约算法

Figure 3　Sketch of parallel protocol

ACO 算法中蚂蚁在搜索时互不依赖，各蚂蚁个体搜索逻辑一致仅数据不同，可将蚂蚁个体独立看待，每个个体对应 CUDA 中的一个线程，并行中的线程指令一致但处理的数据不一致。多个粒子同时进行适应度计算时，实际上是并行了多个蚁群，每个蚁群内部再进行二次并行。ACO 算法并行过程如下。

Step 1　计算需要的内存并分配给 device 端;

Step 2　数据预处理，初始化随机种子、计算距离矩阵、初始化禁忌表等;

Step 3　单个线程初始化，重复执行 $n-1$ 次，计算转移概率写入概率表，选择下一城市;

Step 4　线程同步，等待所有线程完成路径选择;

Step 5　计算已遍历的路径长度，与全局最优路径比较，如果优于全局最优则对其更新，根据当前路径更新禁忌表;

Step 6　更新信息素矩阵，清空禁忌表;

Step 7　如果满足程序结束条件则停止，否则跳转 Step 3。

2.2.2　针对 GPU 的编程优化

CUDA 程序 75% 的性能瓶颈在内存交互上[15]，定义数据结构时需要做内存优化，可用 2 种编码方式定义 n 个粒子，编码方式如下。

方式 1:

```
struct Particle_t{
    double x;
    double y;
    double fitness;
};
Particle_t* particle
=new Particle_t[n];
```

方式 2:

```
struct Particle_t{
    double* x;
```

```
    double* y;
    double* fitness;
} particle;
particle.x=new double[n];
particle.y=new double[n];
particle.fitness=new double[n];
```

首先要减少 CUDA 线程读取数据时产生的 cache miss，cache miss 是指 warp 从 L1 Cache 寻址失败继而请求从全局内存(global memory)中读取数据的过程。warp 是 CUDA 中最小指令执行单元，如图 4(a)所示，线程从 global memory 中读取数据不是单个线程依次读取，而是读取整个 warp 所需要的数据，经由 L2 Cache 到达 L1 Cache，然后按照数据地址线性访问 L1 Cache 中的数据块。

使用第 1 种编码方式，内存访问情况如图 4(b)所示。假设此刻 n 个粒子线程同时操作 x 变量，线程 2 对应的 x 变量地址位于线程 1 的 x 地址+size of(Particle_t)处；若粒子线程 n 从数据块中找不到对应的 x 变量，则会从全局内存重新读取一块数据，这就产生了一次 cache miss。最坏的情况下，n 个线程能产生 n 次 cache miss，这会对程序性能产生严重影响。使用第 2 种编码方式则不会产生频繁的内存加载、内存访问，如图 4(c)所示，能极大提高程序性能，同理，蚁群的数据结构定义也如此。另外，优先使用 CUDA 中的共享内存(shared memory)，一个 block 内所有线程是共享内存的，相比于全局内存速度更快，ACO 算法的禁忌表和转移概率表需要放在全局内存中。

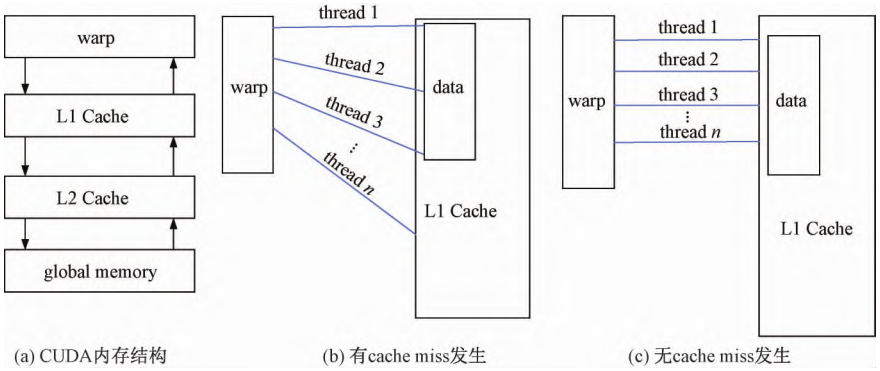

图 4 CUDA 线程读取内存过程

Figure 4 CUDA thread read memory process

3 实验结果与分析

实验所用 CPU 为 Intel © Core™ i5-8400，RAM8 GB，GPU 为 NVIDIA GeForce GTX 1060，Ubuntu 18.04.5 LTS 操作系统，GCC7.5 CUDA10.1 编译器。

使用 TSPLIB 中 eil51 数据集，旅行商数为 3，不考虑代价均衡的情况，各个城市点间的权值设为 1，要解决单点出发的 MTSP 问题，分别用 GPSO-AC 算法(使用 GPU 加速)和 PSO-AC 算法(未使用 GPU 加速)作对比。由表 1 可知，PSO-AC 算法运行时间远远大于 GPSO-AC。在实时性要求较高的系统中使用，PSO-AC 算法无实用性，并且随着群规模增大，其运行时间呈线性增长。而 GPSO-AC 随群规模增大，算法运行时间增幅较小，加速比增大，这是因为算法聚类过程趋于稳定后，质心和聚类结果在算法中后期不再变化，算法程序在 GPU 中指令异化情况大大减少，进一步提高了并行的执行效率。

表 1 算法运行时间对比

Table 1 Algorithm running time comparison

群规模	运行时间/s		GPSO-AC 加速比
	GPSO-AC	PSO-AC	
32	6.045	28.357	4.69
64	6.133	54.830	8.94
128	7.761	101.191	13.04
256	10.741	195.987	18.25

使用 TSPLIB 中 6 个数据集，将 GPSO-AC 和 PSO-AC[7]、TPHA[5]及 K-means-AC[11]算法进行对比，旅行商数为 3，群规模为 64，最大迭代次数为 500，根据经验，学习因子 C_1 和 C_2 均为 1.97，$\alpha=1$，$\beta=3$，实验数据如表 2 所示，GPSO-AC 算法部分实验最优路线如图 5、6 所示。

分析表 2 数据可知，未使用 GPU 加速的 PSO-AC 算法收敛结果优于 TPHA 和 K-means-AC 这 2 种两步式算法，但算法运行时间较长。两步

表 2　4 种算法实验结果对比

Table 2　Experimental results comparisons of four algorithms

模型	执行时间/s				代价最优解			
	TPHA	*K*-means-AC	PSO-AC	GPSO-AC	TPHA	*K*-means-AC	PSO-AC	GPSO-AC
bayg29	0. 15	0. 12	42. 00	1. 80	14 293. 10	14 223. 30	10 403. 20	10 403. 20
berlin52	0. 27	0. 30	87. 35	3. 50	9 754. 85	9 743. 70	8 423. 24	8 470. 35
st70	0. 40	0. 38	124. 94	11. 88	1 077. 00	1 083. 50	830. 09	826. 93
eil101	1. 00	1. 30	226. 72	18. 77	926. 74	906. 80	695. 27	681. 10
ch150	3. 18	3. 51	499. 82	32. 32	9 817. 40	1 006. 71	7 347. 33	6 673. 28
kroA200	7. 41	7. 60	704. 10	35. 13	36 926. 40	36 563. 10	31 940. 70	31 271. 10

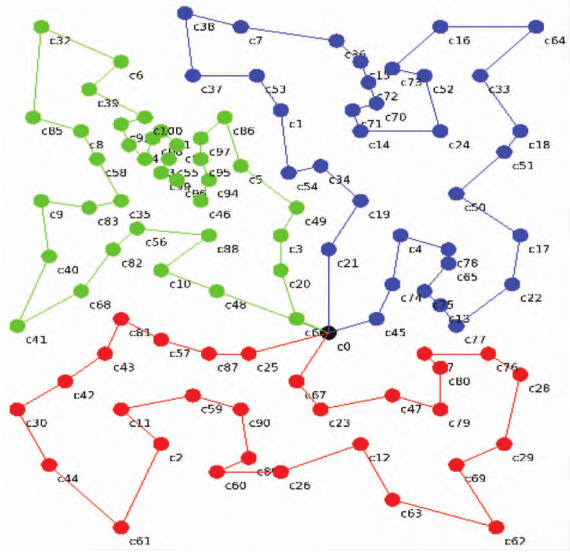

图 5　GPSO-AC 在 eil101 上的运行结果

Figure 5　Result of GPSO-AC on eil101

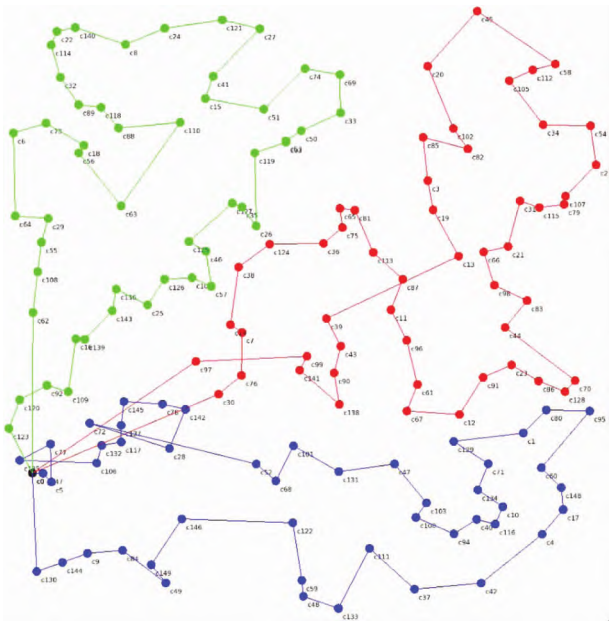

图 6　GPSO-AC 在 ch150 上的运行结果

Figure 6　Result of GPSO-AC on ch150

式算法遵循先分类再计算的思路，运行速度较快，但收敛精度较差。使用 GPU 加速的 GPSO-AC 算法虽然运行时间比两步式算法长，但仍处于合理可接受的范围内且收敛精度优于两步式算法。2 种算法均采用聚类对城市点进行有效分组，TPHA 只考虑分组而算法后期未考虑对分组进行优化，导致其收敛结果不佳，这也是两步式算法共同的弊端。

使用 chn31 数据集，在不考虑代价均衡、代价均衡约束、加权代价均衡的情况下，当旅行商数分别取 1、2、3、4 时 GPSO-AC 运行结果(遍历代价) 如表 3 所示。在旅行商为 4 时，3 种情况下的运行结果如图 7 所示。由表 3 可知，算法考虑代价均衡约束后总代价会变大，虽然各个旅行商的最优解都接近，但标准差远小于不考虑代价均衡的情况。考虑代价均衡的协调系数 $q=0.05$，如图 7(b) 所示。加权代价均衡时，v_{06} 和 v_{46} 设为 30，即 C0～C6 路段、C4～C6 路段上的单位代价为 30，协调系数 $q=0.1$，此时各个旅行商最优解偏差更小，同时绕开了 C0～C6 和 C4～C6 路段，选择了相邻代价小的路线，如图 7(c) 所示。各旅行商间的周游代价离散程度越小，总代价则越大，总代价最低和旅行商间的代价均衡不可能同时满足，需要根据实际问题进行取舍。

表 3　GPSO-AC 对各旅行商的运行代价结果

Table 3　Results of GPSO-AC for traveling salesmen

旅行商	不考虑代价均衡	代价均衡约束	加权代价均衡
1	6 818. 70	7 126. 05	7 349. 15
2	5 661. 20	7 224. 70	7 354. 95
3	5 500. 55	7 278. 90	7 357. 25
4	8 414. 75	7 199. 60	7 339. 80
总代价	26 395. 20	28 829. 25	29 401. 15
标准差	1 165. 26	54. 97	6. 74

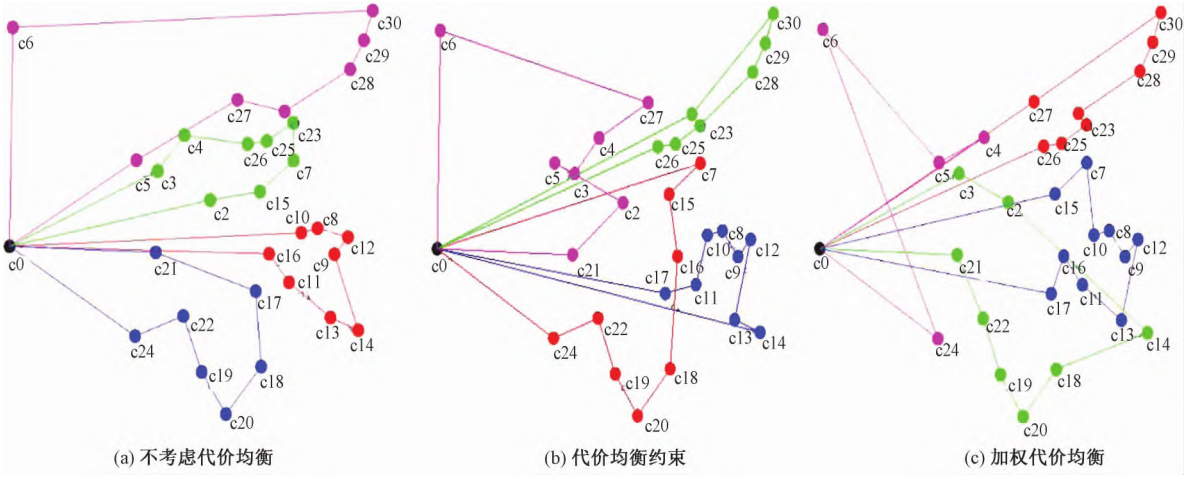

图7 旅行商为4时GPSO-AC在chn31上的运行结果

Figure 7 Results of GPSO-AC on chn31 with 4 salesmen

4 结论

针对混合迭代法算法运行时间长的问题,本文根据粒子群和蚁群算法良好的并行性,提出一种基于CUDA的混合算法GPSO-AC。该算法充分利用GPU多流处理器的设计和单指令多线程的指令架构特点,将算法中大量独立运算过程同时执行。GPSO-AC算法在求解一般MTSP问题及其衍生加权、代价均衡MSTP问题上,不仅加快了混合迭代法的执行速度同时确保了收敛精度。最后,讨论了加权MTSP问题中代价均衡和总代价最优之间的关系。在实际场景中旅行商在各个路段上开销不同,可抽象为代价权重加权MTSP问题。对于各个旅行商如何均衡开销即代价均衡的加权MTSP问题,在保证所有旅行商总路程最短的前提下,代价均衡和总代价最优难以同时满足,仍需进一步研究。

参考文献:

[1] 寿涛,刘朝晖.基于Delaunay三角剖分处理二维欧式空间MTSP的近似算法[J].华东理工大学学报(自然科学版),2017,43(6):895-898.

[2] 刘楠.巡检线路的哈密顿圈分割模型及算法[J].甘肃科学学报,2018,30(3):15-18.

[3] TRIGUI S,CHEIKHROUHOU O,KOUBAA A,et al.FL-MTSP:a fuzzy logic approach to solve the multi-objective multiple traveling salesman problem for multi-robot systems[J].Soft computing,2017,21(24):7351-7362.

[4] 张美燕,蔡文郁.基于多AUV间任务协作的水下多目标探测路径规划[J].传感技术学报,2018,31(7):1101-1107.

[5] XU X L,YUAN H,LIPTROTT M,et al.Two phase heuristic algorithm for the multiple-travelling salesman problem[J].Soft computing,2018,22(19):6567-6581.

[6] ZHAO M R,TANG H L,GUO J,et al.Data clustering using particle swarm optimization[J].Lecture notes in electrical engineering,2014,309:607-612.

[7] SU S B,CAO X B.Jumping PSO with expanding neighborhood search for TSP on a cuboid[J].Chinese journal of electronics,2013,22(1):202-208.

[8] TUANI A F,KEEDWELL E,COLLETT M.Heterogenous adaptive ant colony optimization with 3-opt local search for the travelling salesman problem[J].Applied soft computing,2020,97:106720.

[9] 许凯波,鲁海燕,程毕芸,等.求解TSP的改进信息素二次更新与局部优化蚁群算法[J].计算机应用,2017,37(6):1686-1691.

[10] 叶多福,刘刚,何兵.一种多染色体遗传算法解决多旅行商问题[J].系统仿真学报,2019,31(1):36-42.

[11] TUANI A F,KEEDWELL E,COLLETT M.H-ACO:a heterogeneous ant colony optimisation approach with application to the travelling salesman problem[C]//International Conference on Artificial Evolution.Berlin:Springer,2018:144-161.

[12] BALI O,ELLOUMI W,ABRAHAM A,et al.ACO-PSO optimization for solving TSP problem with GPU acceleration[C]//Intelligent systems design and applications.Berlin:Springer,2017:559-569.

[13] JIANG C,WAN Z P,PENG Z H.A new efficient hybrid algorithm for large scale multiple traveling salesman problems[J].Expert systems with applications,2020,139:112867.

[14] 梁静,刘睿,瞿博阳,等.进化算法在大规模优化问题中的应用综述[J].郑州大学学报(工学版),2018,39(3):15-21.

CUDA-based Parallel Swarm Intelligence Method for Solving Weighted MTSP

SU Shoubao[1,2], ZHAO Wei[1,2], LI Zhi[1,2]

(1.School of Computer, Jiangsu University of Science and Technology, Zhenjiang 212003, China; 2.Jiangsu Key Laboratory of Data Science and Smart Software, Jinling Institute of Technology, Nanjing 211169, China)

Abstract: To solve the low running speed of the multi-traveling salesman problem (MTSP) using the heuristic method, a CUDA-based hybrid particle swarm clustering-ant colony algorithm (GPSO-AC) was proposed by integrating their parallel characteristics with programming techniques optimally. GPSO-AC used GPU's instruction architecture with multiple stream processors (SM) and single instruction multithreading (SIMT) to parallel the search process of numerous independent individuals, so as to accelerate the execution speed of the hybrid iterative method. GPSO-AC was tested on 6 datasets compared with other methods, such as PSO-AC, TPHA and *K*-means-AC. Then the influence of cost equilibrium constraint on the convergence performance of the optimal solution of weighted MTSP problem was discussed. Furthermore, the cost standard deviations obtained from GPSO-AC on chn31 with different traveling salesmen, were 1 165.26, 54.97 and 6.74 in the three cases respectively. The experimental results showed that the proposed algorithm was much faster than other CPU based algorithms and the advantage becomed more obvious with the expansion of the model size, and the convergence precision of the algorithm was better than the similar algorithms for solving MTSP problems.

Keywords: multiple traveling salesman problem(MTSP); CUDA parallel algorithm; cost-balanced; particle swarm clustering; ant colony algorithm

(上接第 6 页)

Robust H_∞ Filtering for Markov Jump Systems with Unknown Transition Probabilities and Packet Dropouts

ZHANG Duanjin, WANG Zhongkun

(School of Information Engineering, Zhengzhou University, Zhengzhou 450001, China)

Abstract: The robust H_∞ filtering problem of Markov jump systems with partly unknown transition probabilities and packet dropouts was investigated. Assuming that the probability of packet dropouts would obey Bernoulli distribution, a discrete-time Markov jump system with uncertain parameters and mode-dependent full-order filter were constructed based on the Delta operator. The slack matrix variables were introduced to solve the cross coupling between the system matrices and the Lyapunov positive matrices. The Lyapunov function, Schur complement and linear matrix inequalities were used to obtain sufficient conditions for the system to be stochastically stable and satisfy H_∞ performance. The optimal H_∞ performance index of the Delta operator system and the shifting operator system were obtained respectively with the known probability of packet dropouts. When the probability of the packet dropouts were lower, the robust performance as well as the optimal H_∞ performance of Delta operator system were better than the shift operator system. Numerical simulation proved that the method proposed in this paper not only was effective and feasible, but also had certain advantages.

Keywords: Markov jumping systems; uncertain parameters; packet dropouts; Delta operator