计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(7):9−16 [doi: 10.15888/j.cnki.csa.007000]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Spark 的改进蚁群算法对带时间窗车辆路径问题的求解①

李奕颖[1,2], 秦　刚[1]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学, 北京 100049)
通讯作者: 秦　刚, E-mail: gqin@cnic.cn

摘　要: 为应对大数据时代对带时间窗车辆路径问题 (VRPTW) 的实时求解要求, 提出基于 Spark 平台的改进蚁群算法. 在算法层面, 利用改进的状态转移规则和轮盘赌选择机制构建初始解, 结合 *k*-opt 邻域搜索进行路径构建优化, 改进最大最小蚁群算法中的信息素更新策略; 在实现层面, 利用 Spark 提供的 API 对蚁群 RDD 进行操作, 实现蚁群分布式并行求解. 在标准算例 Solomon benchmark 和 Gehring & Homberger benchmark 的实验结果表明, 该算法在大规模问题的求解精度和速度上有明显提升.

关键词: 带时间窗车辆路径问题; Spark 平台; 蚁群算法; 邻域搜索

引用格式: 李奕颖,秦刚.基于 Spark 的改进蚁群算法对带时间窗车辆路径问题的求解.计算机系统应用,2019,28(7):9–16. http://www.c-s-a.org.cn/1003-3254/7000.html

Solving Vehicle Routing Problem with Time Window Based on Spark's Improved Ant Colony Algorithm

LI Yi-Ying[1,2], QIN Gang[1]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: In order to cope with the real-time solution requirements of the Vehicle Routing Problem with Time Window (VRPTW) in the era of big data, this study proposed an improved parallel ant colony algorithm based on Spark platform. At the algorithm level, the improved state transition rule and roulette selection mechanism are used to construct the initial solution, and the k-opt local search is used to optimize the path construction. In addition, the improved pheromone update strategy of max-min ant colony algorithm is applied. At the implement level, the ant colony is encapsulated into RDD, which is operated by Spark API to realize distributed construction solution. The experimental results of the Solomon benchmark and Gehring & Homberger benchmark show that the proposed algorithm can improve the accuracy and speed of large-scale problems.

Key words: vehicle routing problem with time window; Spark; ant colony algorithm; local search

引言

车辆路径问题是指在一个存在供求关系的系统中, 在一定的约束条件下, 合理安排车辆的行车路线和出行时间, 把客户需求的货物从配送中心送达客户, 并使目标函数取得最优化[1]. 随着研究的深入和实际应用场景的需求, 增加了对客户偏好服务时间的约束, 即个性化地限定了客户被服务时间的范围, 延伸出了带时间窗的车辆路径问题 (VRPTW), 在实际生产应用中有广

① 收稿时间: 2019-01-10; 修改时间: 2019-02-03, 2019-02-18; 采用时间: 2019-02-25; csa 在线出版时间: 2019-07-01

泛应用, 比如 O2O 外卖配送、冷链物流、生产线上的工作调度、网络路由策略等.

VRPTW 是 NP-hard 的组合优化问题[2], 计算复杂度高, 难以有效率地使用精确算法求得最优解, 目前的研究大多集中在如何设计高质量的启发式算法, 求取近似解以换取计算效率的提高, 如蚁群算法[3]、粒子群算法[4]、遗传算法[5]、模拟退火等群体智能算法. 目前算法研究主要集中于两方面, 一是提升 VRPTW 的求解精度, 二是提升计算速度.

现阶段 VRPTW 求解存在的问题是: (1) 算法过早收敛, 易陷入局部最优解; (2) 相较于集中式节点分布, 算法对随机分布的节点处理精度差; (3) 算法能处理的节点数少, 一般为 100 节点之内; (4) 面对大规模问题出现单机处理效率瓶颈, 忽视使用分布式平台进行并行计算[6]. 针对上述问题, 本文对蚁群算法进行改进, 基于 Spark 平台进行编程求解 VRPTW, 在 Solomon benchmark 和 Gehring & Homberger benchmark 上进行实验, 为快速有效地求解大规模 VRPTW 提供了一种新思路.

1 传统蚁群算法求解 VRPTW

1.1 VRPTW 多目标 0-1 规划模型

VRPTW 可被定义为: 某物流配送网络中, 记0为配送中心, $\{1,2,\cdots,n\}$为n个待配送客户点, 已知各客户点的位置坐标为(x_i,y_i)、货物需求为D_i, 允许的开始服务时间窗为$[ST_i,ET_i](i=1,2,\cdots,n)$, 安排装载能力为$Q$的$m$辆车从0出发, 共同完成对$n$个客户的配送后回到起点0, 优化目标是最小化配送路径长度和车辆数目.

由上述定义可知, VRPTW 是一个离散组合优化问题, 我们将其抽象为多目标 0-1 规划问题[6], 其数学模型具体如下:

决策变量:

$$x_{ijk}=\begin{cases}1, & \text{车辆}k\text{选择路径}(i,j)\\ 0, & \text{其它}\end{cases} \tag{1}$$

目标函数:

$$Z=\min\left(\rho_1\sum_{k=1}^{m}\sum_{j=1}^{n}\sum_{i=1}^{n}l_{ij}x_{ijk}+\rho_2\sum_{k=1}^{m}\sum_{j=1}^{n}x_{0jk}\right) \tag{2}$$

约束条件:

$$\sum_{j=1}^{n}x_{0jk}=1 \ \text{且} \ \sum_{i=1}^{n}x_{i0k}=1,\forall k \tag{3}$$

$$\sum_{k=1}^{m}\sum_{j=0}^{n}x_{ijk}=1,\forall i \tag{4}$$

$$\sum_{i=0}^{n}\sum_{j=0\wedge j\neq i}^{n}D_ix_{ijk}\leqslant Q,\forall k \tag{5}$$

$$ST_i<t_i<ET_i,\forall i \tag{6}$$

$$x_{ijk}\in\{0,1\},\ \forall i,j,k \tag{7}$$

式 (2) 中l_{ij}为点i和点j间路径长度, ρ_1和ρ_2分别为目标函数中路径长度和车辆数目的优化重要度, 如$\rho_1\in[0,1]$越大则表示以路径长度最短为第一目标. 约束目标中式 (3) 表示车辆从0出发并回到0形成闭环; 式 (4) 表示各客户点有且只有一辆车对其服务; 式 (5) 表示各车辆的载重约束; 式 (6) 表示时间窗约束; 式 (7) 表示决策变量为 0-1 变量.

1.2 传统蚁群算法求解 VRPTW

蚁群算法启发于真实蚂蚁自组织的觅食行为[3], 对 VRPTW 这种难解的离散优化问题有优秀的求解能力, 得益于正负反馈机制的协同作用和其并行性. 核心思想是: 单只蚂蚁在其走过的路径上释放信息素, 根据信息素浓度和启发式信息决策转移路径; 蚂蚁之间通过信息素进行通信, 单次迭代后信息素进行挥发且增加优秀路径上的信息素浓度, 使得大概率选择目前优秀解的同时扩展搜索范围, 通过多只蚂蚁间相互协作多次迭代寻优.

蚁群算法是构建解和更新信息素的相互作用[7], 具体如下:

(1) 构建解: 通过计算状态转移概率逐步构建完整解后, 对解进行评估. 状态转移概率公式见式 (8):

$$p_{ij}=\begin{cases}\dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum\limits_{j\in allowed_i}\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}, & \text{if} \ \ j\in allowed_i\\ 0, & \text{否则}\end{cases} \tag{8}$$

其中, p_{ij}表示由节点i选择转移到节点j的概率; τ_{ij}表示路径(i,j)上的信息素浓度; η_{ij}表示路径(i,j)的启发式因子, 通常是路径长度的倒数; α和β分别表示信息素和启发式因子的重要程度.

(2) 更新信息素: 一次迭代后, 根据解评估结果更新信息素浓度, 之后继续迭代寻优. 信息素更新包括追溯增加和挥发减少两种, 具体见式 (9):

$$\begin{cases}\tau_{ij}(t+1)=\rho\tau_{ij}(t)+\Delta\tau_{ij}(t,t+1)\\ \Delta\tau_{ij}(t,t+1)=\sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t,t+1)\end{cases} \tag{9}$$

其中, 蚂蚁t时刻位于节点i, $t+1$时刻位于节点j, ρ表示残留因子, $\Delta\tau_{ij}(t,t+1)$表示信息素增量, $\Delta\tau_{ij}^{k}(t,t+1)$表示第$k$辆车信息素增量, 通常为$1/C_k$, C_k为第k辆车路径总长度.

传统蚁群求解 VRPTW 仍存在引言中提到的 4 个共性问题, 为了避免陷入局部最优, 提升各类点分布和大数据下的求解精度和速度, 下面提出基于 Spark 平台的改进蚁群算法求解 VRPTW.

2 基于 Spark 的改进蚁群算法求解 VRPTW

2.1 算法思路

本文采用基于 Spark 的改进蚁群算法求解带时间窗车辆路径问题, 该算法从算法层和实现层进行改进, 在算法层面, 通过改进状态转移规则[8]、加入轮盘赌选择机制、结合 k-opt 邻域搜索[9]进行路径构建优化, 改进最大最小蚁群中的信息素更新策略, 在实现层面, 用 Spark 计算平台对改进蚁群算法并行实现, 采用 Spark 提供的 API 对蚁群弹性分布式数据集 (Resilient Distributed Dataset, RDD) 进行操作, 实现蚁群并行构建解的过程.

2.2 路径构建优化

蚁群算法是从最初的空解开始, 逐步添加解成分直到构建完整解, 只能生成数量有限的解, 邻域搜索最初依赖一个好的初始解, 不断通过局部调整尝试改进当前解. 本文采用蚁群算法和邻域搜索相结合的方式进行路径构建, 通过状态转移规则和轮盘赌选择机制构建初始解后, 使用邻域搜索进行局部优化获得高质量解.

针对 VRPTW 问题, 由于增加了时间窗约束, 因此状态转移规则除了以信息素浓度高、路径长度小为优先选择原则外, 还要在满足车载限制和时间窗的前提下, 以等候时间短、时间窗紧为优先选择原则, 则蚂蚁由节点i选择转移到节点j的概率见式 (10):

$$p_{ij}=\begin{cases}\dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{j\in allowed_i}\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}*\dfrac{1/(|t_{ij}-ST_j|+|t_{ij}-ET_j|)}{\sum_{j\in allowed_i}1/(|t_{ij}-ST_j|+|t_{ij}-ET_j|)}, & \text{if } j\in allowed_i\\ 0, & \text{其它}\end{cases} \tag{10}$$

记t_{ij}为到达节点j的时间, 当$t_{ij}\geqslant ST_j$时, 等待时间为0, $|t_{ij}-ST_j|+|t_{ij}-ET_j|$为时间窗大小, 否则当$t_{ij}<ST_j$时, $|t_{ij}-ST_j|+|t_{ij}-ET_j|$为时间窗和等候时间的综合指标.

在式 (10) 中参数α和β的作用如下: α和β分别表示信息素和启发式因子的相对重要程度, 如果$\alpha=0$, 则算法未使用信息素, 而大概率地选择距离最近的节点, 属于随机贪心算法; 如果$\beta=0$, 则算法没有任何启发式信息带来的偏向性, 特别是当$\alpha>1$时, 所有蚂蚁按照最初信息素浓度最大的同一条路径移动, 算法很快停滞于一个精度很差的解. 因此这里选择最大最小蚁群算法 (MMAS) 具有良好性能的参数设置: α取值 1~2, β取值 2~5.

若每次迭代均按照状态转移规则p_{ij}直接选择最大概率的节点作为下一个待配送客户, 则信息素向局部最优路径聚集, 使算法过早收敛停滞于次优解, 为了增加算法随机性, 本文使用轮盘赌选择机制对路径构建进行优化, 具体做法是: 随机选取一个实数$T\in[0,1]$, 分别减去各候选节点被选择的概率p_{ij}, 若$T-p_{ij}\leqslant 0$, 则选择城市j作为下一个配送节点, 否则重复上述过程.

因此每只蚂蚁的搜索过程是从候选节点中依据改进的状态转移规则并采用轮盘赌选择机制选择下一个待服务的节点, 逐步添加解成分直至构成完整初始解, 相较于传统蚁群算法的初始解构建过程该方法可以得到更合理的初始解, 状态转移规则改进后使得减少配送过程中车辆的等待时间并能根据顾客的具体时间需求调配配送方案, 轮盘赌选择机制增强了初始解选择的随机性, 使得各节点均有概率被选中且被选择的可能性与其状态转移概率成正比, 能保持发现新路径的能力, 避免算法陷入局部最优.

获得完整初始解后, 通常用 k-opt(k=2, 3) 局部搜索策略对当前解邻域进行局部调整优化, 核心思想是任意选取k个点$(a,b,\cdots,k)$, 删除当前解中的相应k条边$[(a,a+1),(b,b+1),\cdots,(k,k+1)]$, 替换为相关节点重组的另外$k$条边以期获得更优解, 若当前解满足$k$最优, 则对$k'<k$一定也满足最优[10], k-opt 的时间复杂度为$O(n^k)$, k越大时虽然效果越好但耗费的计算时间近似指数级增长[11], 本文采用 Lin 和 Kernighan 提出的 Lin-Kernighan(LK 算法) 局部搜索策略处理大规模数据, 在时间和求解效率间取得较好的平衡, LK 算法通过迭代、回溯不断寻找边y_i代替初始路径的边x_i, 构成序列$X=[x_1,x_2,\cdots,x_r]$和$Y=[y_1,y_2,\cdots,y_r]$, 其中r不固定, 是此序列的最大长度, 具体的算法流程如图 1 所示.

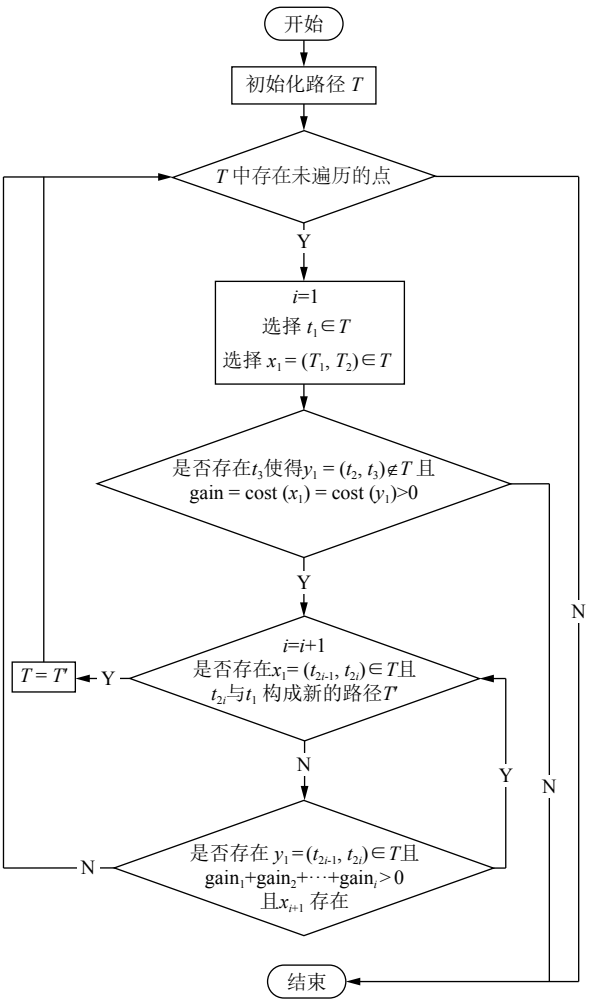

图 1 Lin-Kernighan 邻域搜索算法程序

2.3 信息素更新优化

通过上述路径构建方法, 每只蚂蚁已经可以得到 VRPTW 问题的一个较优可行解, 但是要想获得精度高的解, 还需要群体进行分布式学习的过程, 蚂蚁之间通过每轮迭代后的信息素更新来交互协作, 其中信息素初始化、蒸发策略、更新策略都会影响算法寻优能力[12].

本文选择最大最小蚁群算法 (MMAS) 中的信息素更新策略, 强调对最优路径区域的搜索, 每轮迭代后只选择一只构建出最优路径的蚂蚁释放信息素, 最优路径分为迭代最优T^{Ibest}和全局最优T^{Gbest}, 使用全局最优更新规则会使得搜索加速集中到T^{Gbest}附近, 而使用迭代最优规则能相对减弱搜索导向性[13]. 针对节点数较多的情况, 本文采用T^{Ibest}和T^{Gbest}轮流进行更新的策略.

信息素蒸发策略使得所有路径上的信息素均随时间而衰减,避免信息素在某些边上无限积累或者快速减少[14]. 传统蚁群算法中的信息素蒸发因子$1-\rho$为一个常数, 当ρ过大时, 信息素留存持久度高, 信息素在当前局部最优路径上聚集, 正反馈作用增强, 全局搜索能力差, 当ρ过小时, 信息素衰减过快, 留存时间少, 信息素在路径选择过程中的作用降低, 最终会导致算法收敛性降低, 搜索时间过长且找不到最优路径. 为了平衡全局搜索和正反馈寻优能力, 解决在迭代中后期出现长期找不到新的全局最优解的问题, 本文采用一种自适应信息素蒸发策略, 见式 (11), 具体为在迭代初始采用较大的信息素残留因子$\rho_{\max}$, 加速收敛且增强寻优能力, 随后当全局最优解的搜索停滞超过总迭代次数的 1/10 时, 自适应减少残留因子, 扩大搜索范围, 同时设置残留因子的最小值$\rho_{\min}$. 这里取最大残留值$\rho_{\max}=0.9$, 最小残留值$\rho_{\min}=0.5$, 由于搜索停滞超过总迭代次数的 1/10 时残留因子自适应衰减, 则至多衰减 10 次, 因此衰减系数为$\sqrt[10]{\rho_{\min}/\rho_{\max}}=0.95$.

$$\rho=\begin{cases}0.9, & \rho_{\max}\\ 0.95*\rho, & \text{停滞}\\ 0.5, & \rho_{\min}\end{cases} \tag{11}$$

通过信息素择优和衰减策略可以确定一轮迭代后每条路径上的信息素浓度τ_{ij}, t次迭代后τ_{ij}的值为$(1-\rho)^t\tau_0+\sum_{i=1}^{t}(1-\rho)^{t-i}C_i^{\text{best}}$, 各条路径上的信息素浓度差值过大会造成所有蚂蚁向局部最优路径聚集, 为了避免算法陷入停滞, 这里采用 MMAS 中对信息素浓度大小的限制$[\tau_{\min},\tau_{\max}]$, 见式 (12).

$$\begin{cases}\tau_{\max}=1/(1-\rho)C^{\text{best}}\\ \tau_{\min}=\tau_{\max}(1-\sqrt[n]{0.05})/(n/2-1)\end{cases} \tag{12}$$

信息素初值τ_0一般将其设为略高于每次迭代中蚂蚁释放的信息素大小的期望值, 若 τ_0太小, 信息素更新会有较强的引导性作用, 搜索会快速集中到最初产生的几条路径上, 导致搜索陷入局部空间, 若 τ_0太大, 迭代初始阶段收敛速度慢, 直到信息素衰减后更新才开始发挥作用. 本文取τ_0为$\tau_{\max}$, 使得在循环初始阶段, 具有很强的探索性.

2.4 基于 Spark 平台并行化

目前 VRPTW 问题的许多实际应用面对爆炸式的数据增长和实时性求解的要求, 出现单机求解的效率瓶颈, 而蚁群算法是一个分布式学习的过程[15], 有内在的并行性, 每次迭代中多只蚂蚁独立的根据规则同时

同地的构建可行解, 因此本文将基于 Spark 分布式平台来实现该算法.

Spark 是一个快速且通用的集群计算平台, 扩充了 Map-Reduce 计算模型, 与 Hadoop 不同的是多个任务之间数据通信不需要借助硬盘而是通过内存, 减少了 IO 时间, 提高了程序的执行效率[16]. Spark 的核心概念是 RDD, RDD 的一大特性是分布式存储, 每个 RDD 可以在不同工作节点存储并计算,另一大特性是延迟计算, 一个完整的 RDD 运行任务被分为 Transformation 和 Action 两部分[17], Transformation 创建 RDD, 同时提供 map、filter、join 等大量操作方法生成新的 RDD, Action 通过执行 count、reduce、collect 等方法真正执行数据的计算部分.

本文将蚂蚁封装为 Ant 类, 可行路线的搜索过程实现在 Ant 类的 search() 方法中, 实例化n个 Ant 对象生成蚁群, 蚁群间通过 ant.set_parameter 方法共享参数, 利用 Spark 的 parallelize() 方法将蚁群转换为分布式的 RDD, 在每轮迭代中通过 Spark 中的 map() 方法实现n只蚂蚁的并行 search() 过程, 一轮迭代结束后通过 Spark 中的 collect() 方法收集蚁群计算结果, 得到本轮最优结果后统一更新参数, 核心计算过程如图 2 所示, 从而实现蚁群分布式并行构建可行解, 提升计算效率.

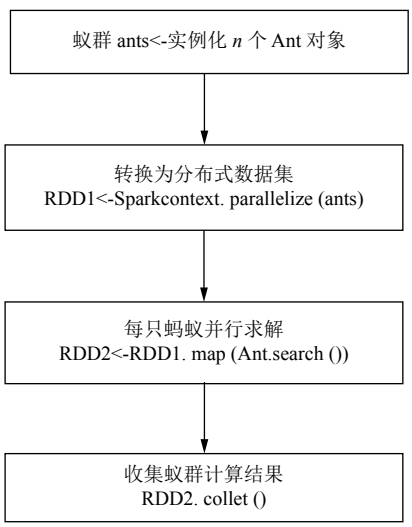

图 2 Spark 核心计算过程

2.5 算法描述

基于 Spark 的改进蚁群算法对 VRPTW 的求解流程如图 3 所示.

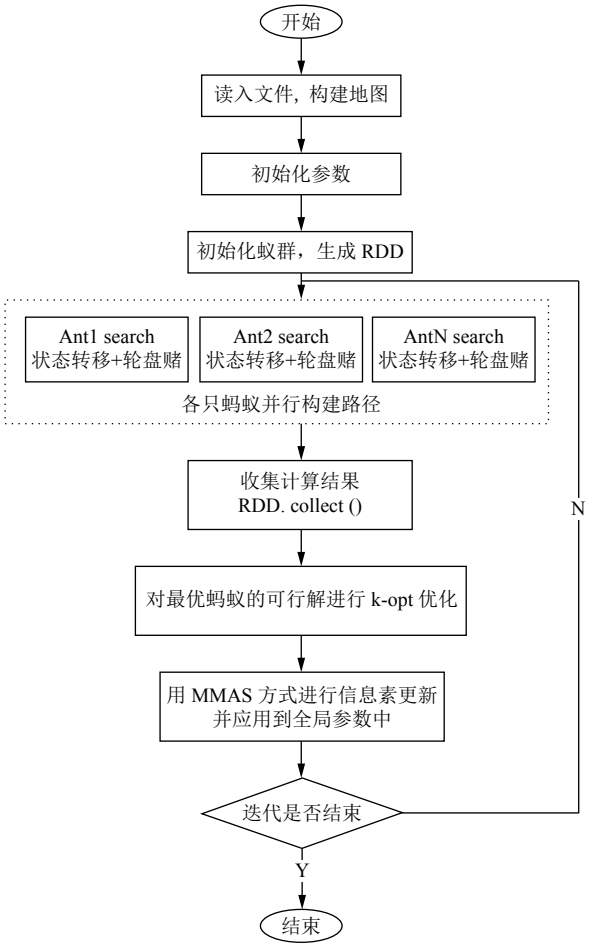

图 3 基于 Spark 的改进蚁群算法求解 VRPTW 流程图

具体算法步骤如算法 1.

算法 1. 基于 Spark 的改进蚁群算法

(1) 读入算例文件, 初始化车辆数 m、车辆容量 $load$, 构造地图 (横坐标 x、纵坐标 y、需求 $demand$、最早开始时间 $ready$、最晚结束时间 due、服务时间 $service\_time$), 并计算地图上各节点间距离;

(2) 采用最近邻贪心算法获得初始可行解和初始路径长度 C_0;

(3) 实例化参数类, 设置 $\alpha, \beta, \rho, \tau_0$ 等参数, 基于初始路径长度设置信息素初值 $\tau_0\tau_{max}=1/C_0$, 生成全局参数;

(4) 将蚂蚁封装为 Ant 类, 根据蚂蚁数目实例化数个 Ant 对象生成蚁群 $ants$, 通过 $ant.set_parameter$方法设置每只蚂蚁的参数, 利用 Spark 的 $parallelize(ants)$ 方法将蚁群转换为分布式的 RDD;

(5) 在每轮迭代中通过 Spark 中的 $map()$ 方法实现 n 只蚂蚁的并行 $search()$ 过程, 每只蚂蚁 $search()$ 方法如下:

① 初始化当前路径 $road$, 加入原点, 设置当前时间 $current\_time$=0, 车辆剩余负载 $remaind\_load$=$load$;

② 构造候选节点集, 满足负载、时间窗要求;

③ 从候选节点中依据状态转移规则 (10) 并采用轮盘赌选择下一个需要服务的节点 $next$ 并加入当前路径中;

④ 若 $next$ 节点为空, 将路径加入路径集中, 新派遣一辆车, 置 $current\_time$=0, $remaind\_load$=$load$;

⑤ 判断是否还有未服务的点, 若存在则返回步骤 (1), 否则结束

搜索.

(6) 每轮迭代结束后通过 Spark 中的 *collect*() 方法收集蚁群计算结果, 对每只蚂蚁搜索得到的可行解进行排序, 根据图 1 对迭代最优解进行 *k*-opt 局部调整优化;

(7) 得到迭代最优解后通过 *ant.set_parameter* 方法统一更新参数, 各路径信息素浓度更新公式参考式 (9)、(11)、(12), 采用 T^{Ibest} 和 T^{Gbest} 轮流进行更新、自适应信息素蒸发策略且限制信息素范围为$[\tau_{min}, \tau_{max}]$;

(8) 判断迭代是否终止, 迭代终止输出全局最优解, 否则返回步骤 (4) 继续并行化求解.

3 实验设计及结果分析

为测试基于 Spark 的改进蚁群算法对 VRPTW 的求解效果, 本文在物理机上利用 Virtual Box 构建 3 台虚拟机搭建 Spark 集群, 并分别在单台虚拟机和 Spark 集群上进行实验.

3.1 实验环境及算例说明

实验环境如下:

(1) 单机运行环境

操作系统: Ubuntu 16.04.

硬件环境: Virtual Box 虚拟机.

Intel Core i7.

2 GB 内存.

(2) Spark 集群运行环境

操作系统: Ubuntu 16.04.

硬件环境: Virtual Box 虚拟机.

Intel Core i7.

2 GB 内存.

部署方式: Standalone 模式.

集群规模: 1 台 Master, 2 台 Slave.

实验采用 Solomon benchmark 和 Gehring & Homberger benchmark 两个国际通用基准算例库中的数据, 数据集按照影响 VRPTW 求解的地理数据分布、客户节点数、时间窗松紧程度、车辆载重上限 4 个因素进行分类, Solomon 数据集规模包括 25、50、100 节点数, 标号为$X_1X_2X_3X_4$, X_1取值 R、C、RC, 分别表示客户节点随机、聚集、混合分布, X_2表示车辆载重, 取值 0 或 1, X_3X_4取值一个两位十进制数, 表示时间窗松紧程度, Gehring & Homberge 数据集是大规模数据集, 包括 200、400、600、800、1000 节点数. 数据包含车辆数上限、载重上限、客户位置坐标、客户需求量、时间窗、服务时间[18].

3.2 实验设计及结果分析

3.2.1 信息素矩阵可视化表达

为直观表达蚁群算法求解 VRPTW 的行为, 选取 Solomon-r101 的前 25 节点进行求解并给出信息素矩阵的可视化表达, 信息素含量被转化成灰度值, 颜色越暗的方格所对应的信息素值越高, 在第 0、10、100 次迭代之后的实验效果如图 4.

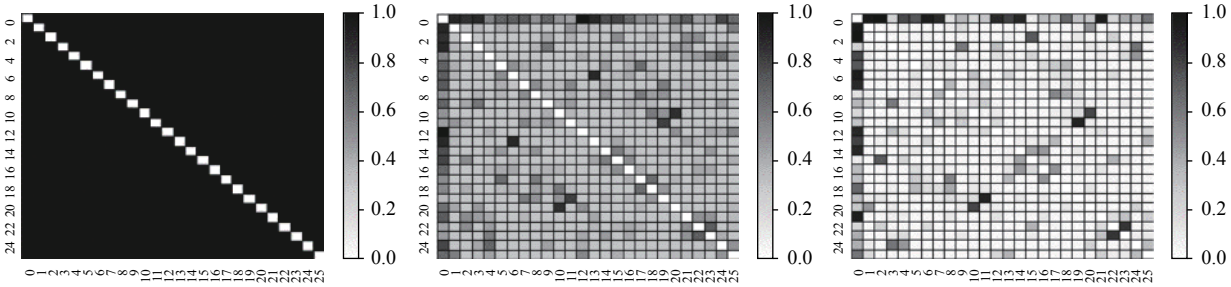

图 4 信息素矩阵可视化表达

由图 4 可以看出, 主对角线的单元格始终为白色, 表示这些单元格的信息素被初始化为 0 并且永不更新, 其余单元格初始为黑色表示 $\tau_0 = \tau_{max}$, 随着迭代过程反复执行, 单元格间的信息素含量差异越来越大, 最后只有少数边含有较高信息素而被选择为全局最优解.

3.2.2 收敛性比较实验

为验证本算法的改进效果, 分别将不同信息素更新策略、不同邻域搜索方法作为控制变量进行实验. 选取 Solomon-c101 和 Gehring & Homberger-c121 作为实验数据, 分别采用信息素全局更新和交替更新的求解收敛过程如图 5, 选取 Gehring & Homberger-rc121 作为实验数据, 分别采用 2-opt、3-opt、*k*-opt 这 3 种邻域搜索的求解收敛过程如图 6.

由图 5 可以看出, 对 100 节点数据, 信息素更新方式对求解过程无明显影响, 而对 200 节点数据, 采用全

局信息素更新方式使得求解陷入局部最优解而过早收敛, 采用信息素交替更新方式能在执行结束后得到更高质量的解. 由图 6 可以看出, 使用 *k*-opt 进行局部搜索优化比 2-opt 和 3-opt 得到的解的精度高. 综上, 改进蚁群算法求解较大规模 VRPTW 时在迭代初期收敛速度较快, 在迭代中后期能保持全局搜索能力使求解持续收敛, 最终在期望的时间内收敛于一个精度较好的近似解.

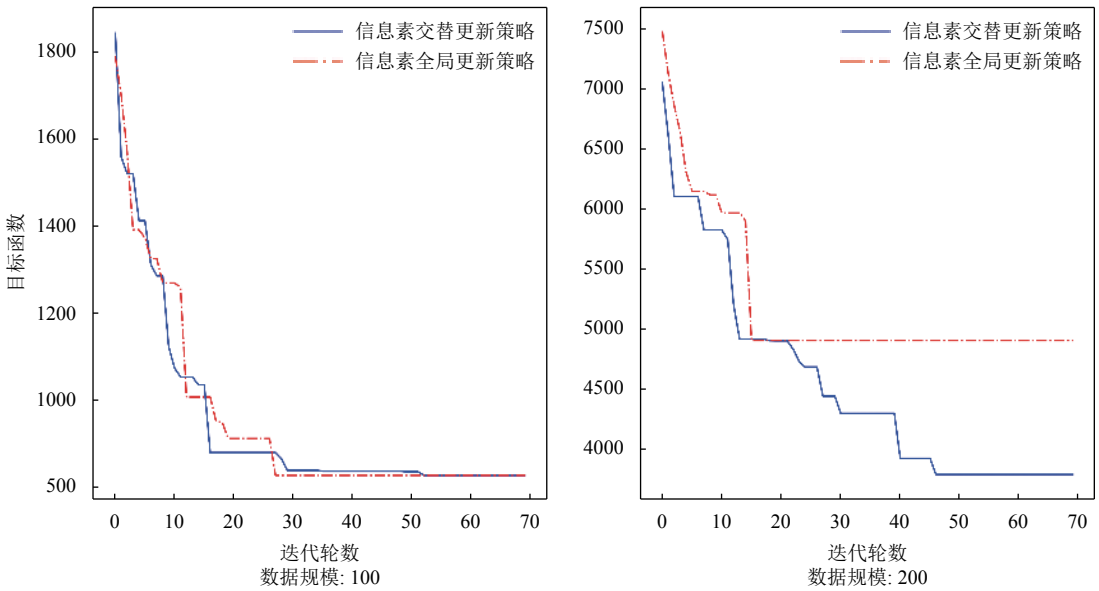

图 5 不同信息素更新策略收敛过程

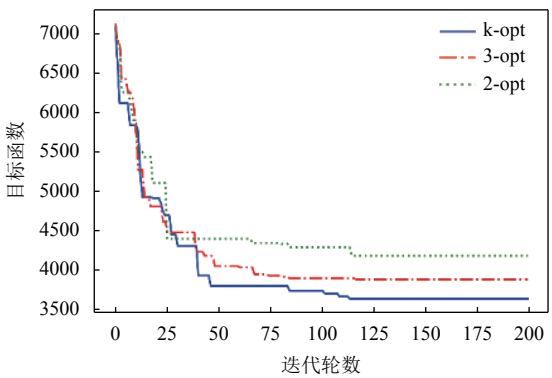

图 6 不同邻域搜索收敛过程

3.2.3 求解精度比较实验

为验证本算法的求解精度, 选取 Solomon 节点数为 100 的各类数据和 Gehring & Homberger benchmark 节点数为 200 及 400 的大规模各类数据进行测试, 参数设置为: $\alpha = 1, \beta = 2, \rho = 0.9$, 迭代 400 轮计算与当前最优解的误差率见表 1.

由表 1 可以看出, 对 100 以内小规模问题求解的精度基本可达最优解; 对混合分布的 RC 类问题求解精度有明显提升; 对随机分布的 R 类问题误差率仍较高, 尤其是对车辆载重小且最大可服务时间短的 R1 类问题精度更差, 原因是该类问题节点的随机分布增大了路径构造的难度, 各子路线可能包含的节点数差异大; 对 200、400 节点的大规模问题在计算速度提升的前提下精确度在可接受的误差范围内.

表 1 100、200、400 数据量求解精确度实验结果

算例名称	目前公布最优解 (距离/车辆数)	本文最优解 (距离/车辆数)	误差率 (%)
客户数: 100			
c101	828.94/10	828.94/10	0
c201	828.94/10	828.94/10	0
r101	1650.80/19	1720.38/17	4.21
r201	1252.37/4	1311.67/4	4.73
rc101	1696.95/14	1700.24/14	1.93
rc201	1406.94/4	1446.05/4	2.78
客户数: 200			
c1_2_1	2704.57/20	2945.27/23	8.90
r_1_2_1	4784.11/20	5412.11/25	13.12
rc1_2_1	3602.80/18	3939.66/19	9.35
客户数: 400			
c1_4_1	7152.02/40	8215.23/44	14.86
r1_4_1	10372.31/40	12239.33/41	18
rc1_4_1	8572.36/36	10212.11/39	19.10

3.2.4 求解速度比较实验

为比较单机求解和基于 Spark 平台并行求解的求

解效率, 选取客户数为 25、50、100、200 以及 400 的 Solomon-r101、Gehring & Homberge-R1_2_1、Gehring & Homberge-R1_4_1 作为实验数据, 基于相同参数和迭代轮数进行实验, 实验结果如图 7.

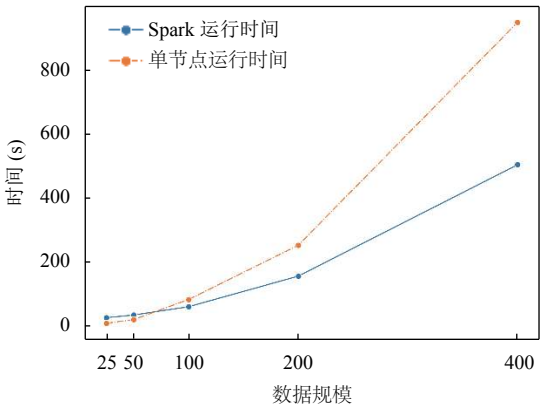

图 7 求解速度比较实验结果

从图 7 可以看出, 当问题规模较小时, 由于 Spark 内部的通信消耗, 基于 Spark 的并行蚁群算法的求解效率较单机蚁群算法没有显著的提升, 而当问题规模逐步增大时, 基于 Spark 的并行蚁群算法的求解时间要显著少于单机算法的求解时间, 并行算法的加速比逐步增加, 当客户数为 400 时, 并行算法的加速比为 1.95.

4 总结与展望

本文设计了一种基于 Spark 的改进蚁群算法求解带时间窗的车辆路径问题, 根据改进的状态转移规则和轮盘赌选择机制构建初始解, 结合 *k*-opt 邻域搜索进行局部搜索优化, 采用全局最优解和局部最优解交替更新策略、自适应信息素蒸发策略改进最大最小蚁群算法中的信息素更新策略, 借助 Spark 计算平台, 将蚁群封装为弹性分布式数据集, 实现蚂蚁的并行搜索.

实验结果表明本算法对 100 以内小规模问题求解的精度基本可达最优解, 对混合分布的 RC 类问题求解精度有提升, 对 200、400 节点的大规模问题求解效率有明显提升. 因此把蚁群算法这样的启发式算法和邻域搜索相结合是解决优化问题的有效途径, 另外利用 Spark 分布式平台处理蚁群算法中耗时的迭代过程可以有效降低计算时间.

本文试图对 C、R、RC 类 VRPTW 问题同时取得较好的优化效果, 但对 R 类分布的节点求解精度还不理想, 今后可结合其它群体智能算法或者根据随机分布类问题的特点进行合适的参数调整对 R 类 VRPTW 问题进行进一步优化求解. 另外本文尚未对 600、800、1000 等更大规模节点进行实验, 今后可以通过改进算法、充分利用分布式平台、提升硬件环境来解决更复杂的大规模求解问题.

参考文献

1 郎茂祥. 配送车辆优化调度模型与算法. 北京: 电子工业出版社, 2009.
2 Solomon MM. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 1987, 35(2): 254–265. [doi: 10.1287/opre.35.2.254]
3 Dorigo M, Maniezzo V, Colorni A. Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 1996, 26(1): 29–41.
4 Jin NB, Rahmat-Samii Y. Particle swarm optimization for antenna designs in engineering electromagnetics. Journal of Artificial Evolution and Applications, 2008: 9.
5 Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Upper Saddle River, New Jersey: Addison-Wesley, 1989.
6 陈迎欣. 基于改进蚁群算法的车辆路径优化问题研究. 计算机应用研究, 2012, 29(6): 2031–2034. [doi: 10.3969/j.issn.1001-3695.2012.06.007]
7 李琳, 刘士新, 唐加福. 改进的蚁群算法求解带时间窗的车辆路径问题. 控制与决策, 2010, 25(9): 1379–1383.
8 董攀. 带时间窗车辆路径问题的蚁群算法改进[硕士学位论文]. 长沙: 长沙理工大学, 2014.
9 Helsgaun K. An effective implementation of the Lin–Kernighan traveling salesman heuristic. European Journal of Operational Research, 2000, 126(1): 106–130. [doi: 10.1016/S0377-2217(99)00284-2]
10 崔文. 大规模多配送中心车辆路径问题研究[硕士学位论文]. 济南: 山东大学, 2012.
11 吴越钟. 改进的 Lin-Kernighan 局部搜索算法和杂交算法在旅行商问题中的应用[硕士学位论文]. 合肥: 中国科学技术大学, 2016.
12 葛斌. 求解车辆路径问题的蚁群优化算法研究及应用[博士学位论文]. 合肥: 合肥工业大学, 2016.
13 丁秋雷. 带有时间窗的车辆路径问题的混合蚁群算法研究[硕士学位论文]. 大连: 大连理工大学, 2006.
14 王颖, 谢剑英. 一种自适应蚁群算法及其仿真研究. 系统仿真学报, 2002, 14(1): 31–33. [doi: 10.3969/j.issn.1004-731X.2002.01.010]
15 Nalepa J, Czech ZJ. A parallel memetic algorithm to solve the vehicle routing problem with time windows. arXiv: 1402.6942, 2014.
16 郭宝恩. 基于 Spark 的蚁群算法在物流配送路径优化问题中的应用研究. 信息与电脑 (理论版), 2018, (3): 50–52.
17 王诏远, 王宏杰, 邢焕来, 等. 基于 Spark 的蚁群优化算法. 计算机应用, 2015, 35(10): 2777–2780, 2797.
18 金淳, 张雨, 王聪. 带时间窗车辆路径问题的分布式多 agent 蚁群算法. 计算机应用研究, 2018, 35(3): 666–670.