文章编号:1671-6833(2024)04-0062-08

基于 Spark 的双阶段 SA 及 GA 求解 MTSP

孙 鉴[1,2], 刘 品[1], 李 昊[1], 陈 攀[1]

(1. 北方民族大学 计算机科学与工程学院,宁夏 银川 750021;2. 北方民族大学 图像图形智能处理国家民委重点实验室,宁夏 银川 750021)

摘 要: 针对总路径长度最小的单站点多旅行商问题,提出了基于 Spark 的模拟退火和遗传算法结合的两阶段 KSAGA 算法。在第一阶段,通过 k-means 聚类将多旅行商问题拆分为多个单旅行商问题,并使用模拟退火算法对组内城市的遍历次序进行优化。在第二阶段,通过遗传算法对城市的分组进行优化,并基于染色体分组编码方式设计了交叉、变异算子以及混合局部优化算子,以提高算法的搜索空间和收敛速度。随着城市数量的增加,计算规模变大,利用遗传算法的特性实现算法的并行,以加快算法运行效率。最后,通过选取 TSPLIB 的部分数据集进行仿真实验,将 KSAGA 与 ACO、GA、SPKSA、ALNS 和 NSGA-Ⅱ的求解质量以及 GA 和 NSGA-Ⅱ的收敛速度进行对比。研究结果表明:KSAGA 在解决单站点多旅行商问题时能够快速收敛,并且相较于其他算法,求解质量得到了很大提升。同时,随着城市数量和旅行商数量增加,KSAGA 的优势更为明显。

关键词: 多旅行商问题; 并行; 遗传算法; 分组编码; 局部优化算子

中图分类号: TP301.6 **文献标志码**:A **doi**:10.13705/j.issn.1671-6833.2024.01.019

旅行商问题(traveling salesman problem,TSP)是一个著名的组合优化问题:一个旅行商访问 n 个城市后返回起始点,每个城市只能被访问一次,目的是找到总距离最短的路线。TSP 问题的现实应用有很多,例如物流调度、交通运输、生物迁徙以及电路设计等。

而实际问题中让一个旅行商访问所有城市的工作量巨大,因此当需要多个旅行商去访问城市以缩短访问时间时,经典的 TSP 模型不再适用。为解决上述问题,对 TSP 进行扩展,引入了多旅行商问题(multiple traveling salesman problem,MTSP)[1]。本文考虑了单站点的 MTSP,即 m 个旅行商从同一城市(出发城市称作仓库)出发访问 n 个城市,保证每个城市只被访问一次且每个旅行商至少要访问一个除仓库以外的城市,问题的求解目标为 m 个旅行商访问城市路径的总距离最小。相较于 TSP 问题,MTSP 具有更高的实际应用价值,如热门的车辆路线问题、校车路线规划问题、任务分配问题以及航班调度问题等。

求解 MTSP 的方法主要包括精确算法、近似算法和启发式算法。随着问题规模的扩大,问题解空间膨胀速度与问题规模呈指数关系,这成为限制精确算法求解 MTSP 性能的主要瓶颈,同时由于近似算法求解精度并不理想,因此目前基于群体的启发式算法是求解 MTSP 最常用、最有效的方法。

求解 MTSP 早期通常使用一种启发式算法,如 Mohammad 等[2]使用了两部分染色体编码方式的遗传算法(genetic algorithm,GA),进行选择、交叉和变异操作后,在传统遗传算法的基础上使用了 opt 算子对结果进行优化。Yang 等[3]提出了新的交叉和变异算子来增强局部和全局搜索能力,然后应用 NSGA-Ⅱ框架来加快算法收敛速度,增加解的多样性。

近年来,一些研究提出用混合算法分阶段解决 MTSP,这些算法结合了不同的启发式方法和技术。Harrath 等[4]所提出的方法是 3 种算法的组合:改进的 ACO、2-opt 和 GA,使用 ACO 来生成 TSP 初始解,在该解基础上用 2-opt 算子来优化所获得的解后,将

收稿日期:2023-07-26;**修订日期**:2023-09-15

基金项目:国家自然科学基金资助项目(62062002);宁夏自然科学基金资助项目(2022AAC03289,2022AAC03245,2022AAC03261)

作者简介:孙鉴(1982—),男,山东烟台人,北方民族大学讲师,博士,主要从事大数据存储与管理等研究,E-mail:2014132@nun.edu.cn。

引用本文:孙鉴,刘品,李昊,等. 基于 Spark 的双阶段 SA 及 GA 求解 MTSP[J]. 郑州大学学报(工学版),2024,45(4):62-69, 94.(SUN J, LIU P, LI H, et al. Solving MTSP with two-stage SA and GA based on Spark[J]. Journal of Zhengzhou University (Engineering Science),2024,45(4):62-69, 94.)

TSP 转化为 MTSP,使用 GA 再次提高解的质量,最后更新信息素映射以继续进行下一次迭代。但由于在单处理器环境下只能以串行方式对解空间进行搜索,当问题规模较大时,传统算法求解效率及性能大大降低,因此通过并行方法[5]来提高算法的性能成为研究热点之一。

针对上述算法求解质量上存在的不足以及混合算法计算规模较大的问题,本文对单站点多旅行商问题进行研究,提出了一种基于 Spark 的两阶段启发式算法 KSAGA。算法由组内优化阶段和分组优化阶段组成,首先通过 k-mean 聚类和贪心算法生成初始解,第一阶段应用模拟退火(simulated annealing, SA)对每个旅行商的访问顺序进行优化,第二阶段通过并行 GA 对城市分组进行优化。在本文中,所有实验都将数据集中的第一个城市视为仓库。

1　相关背景

1.1　MTSP 数学描述

给定一个完整的无向图 $G(V,E)$,其中 V 表示城市的集合($V=\{1,2,\cdots,n\}$),E 为成对的边。$\boldsymbol{c}_{ij}$ 表示边 e_{ij} 的权重(即从城市 i 到城市 j 的旅行距离),多旅行商问题实际上就是寻找加权图中的最短巡回路径问题。式(1)~(6)给出多旅行商问题的数学模型,其中 m 为旅行商数量;n 为城市数量;x_{ijk} 为一个二进制变量,表示在旅行商 k 的路径中城市 j 是否在城市 i 之后立即被访问,若是,则 $x_{ijk}=1$,否则$x_{ijk}=0$;u_i 和 u_j 为关于城市 i 和 j 的访问顺序的变量,如果城市 i 在路径中排在城市 j 之前,则 $u_i>u_j$;p 表示任何销售人员可以访问的最大城市数量。

目标函数:

$$\text{Minimum}\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{c}_{ij}\sum_{k=1}^{m}x_{ijk}。\tag{1}$$

约束条件:

$$\sum_{i=1}^{n}\sum_{k=1}^{m}x_{ijk}=1,j=1,2,\cdots,n;\tag{2}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{m}x_{ijk}=1,i=1,2,\cdots,n;\tag{3}$$

$$\sum_{j=1}^{n}x_{1jk}=1,k=1,2,\cdots,m;\tag{4}$$

$$\sum_{i=1}^{n}x_{i1k}=1,k=1,2,\cdots,m;\tag{5}$$

$$u_i-u_j+p\sum_{k=1}^{m}x_{ijk}\leqslant p-1,i\neq j=2,3,\cdots,n。\tag{6}$$

其中,式(1)为目标函数,优化目标为 m 个旅行商访问所有城市后的总路程之和最小;式(2)、(3)表示每个城市都要被访问一次,有且只有一个旅行商从此城市出发;式(4)、(5)表示所有旅行商都要从仓库出发最终再回到仓库位置;式(6)[6]将子路径消除约束扩展到一个三维度模型,这个约束限制了子路径的产生。

1.2　Spark 框架

Apache Spark[7]是一个用于大数据处理的开源框架,是目前最流行、速度最快的分布式计算框架之一,允许集群或云架构中的应用程序并行化。Spark 将计算机集群划分为一个主节点和多个工作节点:主节点负责任务调度、资源分配和错误管理;工作节点并行处理映射任务和减少任务,用户程序分发和输入数据拆分由平台自动完成。Spark 附带了包括多种语言(Scala、Java、Python、SQL 和 R 语言)的丰富的 API,用于执行复杂的分布式数据的操作。

Spark 的核心数据结构是弹性分布式数据集(resillient distributed dataset,RDD),借助 RDD 的高效数据共享和各种操作,可以高效地设计和实现不同的工作负载。RDD 是一个只读的、分区的记录集合,提供了容错的并行数据结构。RDD 允许用户将数据显式存储在磁盘或内存中,控制其分区并使用操作符进行操作。操作符主要分为转换和执行两种。转换是指从现有 RDD 的基础上创建新的 RDD,是返回一个新的 RDD 的操作。转换操作是惰性求值的,不会立即触发执行实际的转换,而是先记录 RDD 之间的转换关系,只有当触发执行操作时才会真正地进行转换操作,并返回计算结果。常用方法有 map、filter 等。执行操作则是对最后一个 RDD 数据执行计算后产生结果,并输出到外部数据源。常用到的方法有 reduce、collect 等。

RDD 的一次执行过程如下:通过对一个集合进行并行化或者读入外部数据源进行创建,利用 textFile 函数加载本地数据;经过一系列的转换操作,每次操作都会产生不同的 RDD;最后一个 RDD 经过“执行”操作进行转换,并输出到外部数据源。这一系列处理被称为一个 Lineage。RDD 的一次执行过程如图 1 所示。

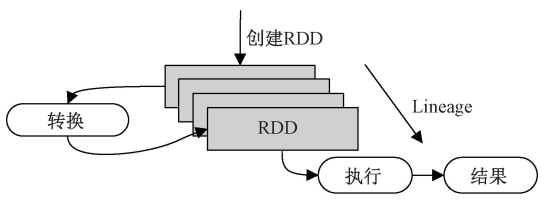

图 1　RDD 一次执行过程

Figure 1　An RDD's Lineage

2 KSAGA 算法

本文提出的 KSAGA 算法将 MTSP 求解过程划分成两部分:城市的分组优化和组内遍历次序优化。算法的基本思想如下。

(1)由 k-means 聚类和贪心算法生成问题的初始解。根据设定的 m 个旅行商将 n 个城市分成相应数量的组,若当前组内不包括仓库,则组内增加该城市,每个分组代表分配给该旅行商的城市,分组后通过贪心算法可得到每个旅行商访问的城市的顺序,生成较优的初始解。

(2)通过将城市分组把 MTSP 转化成多个 TSP 问题,通过并行 SA 求解 TSP。

(3)优化组内城市访问顺序后,用遗传算法优化城市的分组。本文的遗传算法采用染色体分组编码方式,并设计了适用于此方式的交叉、变异算子,通过变异操作跳出局部最优,同时设计了局部优化算子加快算法收敛速度。通过将第一阶段生成的种群分成若干个域,再将数据存入 RDD 中来实现遗传算法的并行。

2.1 组内访问顺序优化

2.1.1 模拟退火算法

模拟退火算法是一种基于概率的搜索算法,它模拟了物理系统的热力学过程以找到最优解。算法以一定的概率接受更差的解决方案,以避免陷入局部最优解,并最终找到全局最优解。由于 SA 在求解 TSP 问题上具有较高的优势,因此在第一阶段本文采用优化的 SA[8] 来求解单旅行商问题,该方法的降温函数为

$$T_0 = T_0\alpha; \tag{7}$$

$$T = T_0(1 + \cos \pi t/gapIter)。 \tag{8}$$

式中:T_0 为初始温度;T 为变化后的温度;α 为降温系数;t 为迭代次数;$gapIter$ 为浮动的间隔数。

2.1.2 并行模拟退火算法

在组内优化阶段,采用了基于 Saprk 的并行策略。通过 k-means 聚类将 MTSP 转化成多个 TSP 求解,每个分组的城市求最短路径即是一个 TSP 问题,通过并行模拟退火算法分别求出对应 TSP 的最优解,将结果合并即为第一阶段的最终结果。其具体步骤如下。

步骤 1 将城市分为 m 个组,m 为旅行商数量,由于每个旅行商都要求从仓库出发,所以将仓库添加至未包含仓库坐标的组中;

步骤 2 分组后使用 SparkContext 对象的 parallelize 方法将以上每组城市创建为分布式 RDD;

步骤 3 调用 RDD 的 map 算子将 m 组城市分别映射到各子节点并调用贪心算法生成初始解;

步骤 4 调用 SA 优化组内遍历顺序,最后使用 RDD 的 collect 算子将分布式的 RDD 本地化,合并后得到第一阶段的最优解。

2.2 城市分组优化

将 k-means 聚类和 SA 生成的结果作为 GA 的初始种群,通过对种群进行改进,完成对城市分组的优化。GA 主要的操作算子包括以下 6 个部分。

(1)编码设计。常见的染色体编码方式有单染色体设计[9]、两部分染色体设计[10]、双染色体设计[11]等。以上编码方式的搜索空间相对较小,搜索效率较高,有利于加快 GA 的收敛速度,但是也存在明显缺陷:当把城市分配优化以及组内访问顺序优化 2 个环节融合在一起时会导致求解 MTSP 的进化算子难以设计。相较于以上几种编码方式,分组编码设计简单、易于实现,十分适用于多旅行商问题的求解,因此本文采用了染色体分组编码设计思想,将各旅行商的路径访问顺序编码成 m 个组,便于问题的分步优化。

(2)适应度函数。适应度函数是根据个体的适应值对其优劣判定的评价函数。本文的优化目标是最小化总路程,用访问路径总距离的倒数作为适应度函数来评价求解结果是否最优。适应度函数为

$$F = 1/\sum_{1}^{m} li。 \tag{9}$$

式中:li 为一个旅行商访问组内所有城市的路程;m 为旅行商数量。

(3)选择操作。选择策略是 GA 进化操作的重要步骤,会影响算法的执行效率。本文采用了轮盘赌算法作为选择策略,这是一种非随机选择策略,适应度越高的个体被选中的概率就越大。

(4)交叉操作。本文的交叉算子随机移除每个旅行商部分城市,移除比例为 Pr,用贪心算法将移除的城市逐个插入到使总路程增加最少的位置,其操作如图 2 所示。

(5)变异操作。变异算子的主要作用是防止搜索过程陷入局部最优。本文针对染色体分组编码方式,提出了适用于解决 MTSP 的两种变异算子:变异算子 1 对城市组内的访问顺序以及不同组城市之间的分组都进行了一定重组,这个过程随机移除了父代部分城市,移除比例为 Pw,将移除的城市随机分配给各旅行商;变异算子 2 则随机选择 2 个旅行商,生成断点 1 和断点 2,断点 1 之前的城市与断点 2 之后的城市组合,断点 2 之前的城市与断点 1 之后的

城市组合,将这两条新生成的路径作为新的解,这样既保留了父代的部分优化解,又可以扩大搜索范围,跳出局部最优。相较于变异算子 1,变异算子 2 对组内城市的访问顺序未进行过多破坏,其具体操作过程如图 3 所示。

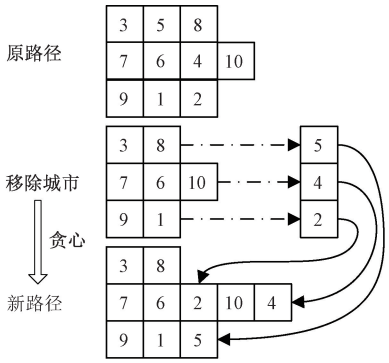

图 2　交叉算子

Figure 2　Cross operate

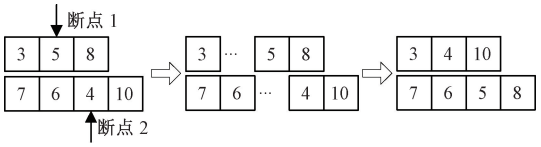

图 3　变异算子 2

Figure 3　The second mutation operator

(6)局部优化算子。GA 用于优化不同旅行商之间的城市分组问题,因此这里的局部优化算子用于对不同旅行商之间的城市进行操作,局部搜索过程包括 opt、insert、swap 这 3 个优化算子,其具体操作过程如下。

opt:将第一组城市 p_0 和第二组城市的 p_1 作为断点,把各组的城市分为两部分,第一组城市 p_0 及其之后的城市逆序后与第二组城市的第二部分合并,第一组的第一部分与第二组 p_1 之前的城市逆序后合并,若操作后总路程减少则保留此路径。opt 操作如图 4(a)所示。

insert:插入算子操作如图 4(b)所示,将第二组城市 p_0 插入到第一组城市的 p_1、p_2 之间以改进解决方案。

swap:swap 操作符如图 4(c)所示。它通过交换不同组的 2 个城市来优化路径,图 4(c)为城市 p_0 和 p_1 城市交换示例。

2.2.1　遗传算法流程

遗传算法具体操作流程如下。

步骤 1　进行参数设置,包括迭代次数 *maxOutIter*,种群数量 *Psize*,执行交叉、变异算子的阈值 *Pm*,2 个变异算子的阈值 *Pt* 等;

步骤 2　使用轮盘赌算法选择一个父代,适应

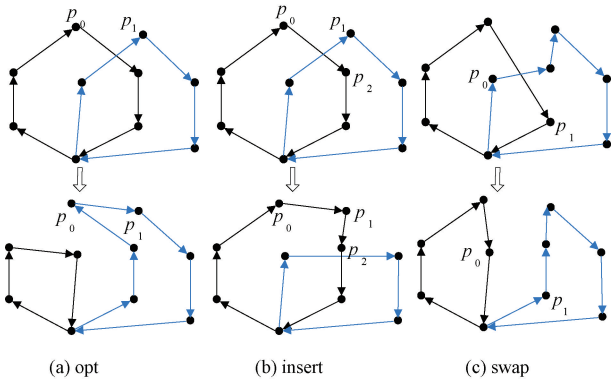

图 4　局部优化算子

Figure 4　Local optimization operator

度高的个体被选中的概率更大;

步骤 3　生成 0 到 1 的随机数,若随机数小于阈值 *Pc*,跳过步骤 4 执行交叉操作,否则进行变异操作;

步骤 4　生成 0 到 1 的随机数,若小于阈值 *Pt* 则执行变异算子 1,否则执行变异算子 2;

步骤 5　用局部优化算子对后代局部优化;

步骤 6　若种群数量未达到规定数量,则将新的个体放入种群中,若超过规定种群数量且当前个体的解优于种群中的最差解,则用新的子代替换种群中适应度最低的个体,然后根据适应度将种群排序;

步骤 7　进化结束判断(是否达到最大进化代数),若满足则循环结束,否则返回步骤 2 继续执行。

2.2.2　并行遗传算法

GA 的种群规模通常很大,并且要经过多次进化,因此需要很大的计算量。并行可以提高计算速率,并有利于增大计算规模、扩大解的搜索空间。将数据放在 RDD 分区中将 GA 并行,在结束后对种群按个体适应度大小进行排序,从而得到更精确的解,并行 GA 流程图如图 5 所示。

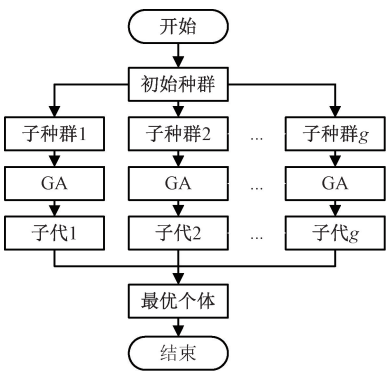

图 5　并行遗传算法流程图

Figure 5　Parallel GA flowchart

并行 GA 流程如下。

(1)初始化种群:将 k-means 和 SA 优化后的路径作为遗传算法的初始种群。

(2)并行化处理:为了提高算法执行效率,将初始种群分为 g 个线程,其中 g 在本文中设置为 15。每个线程将负责处理 1 个子种群,因此可以同时进行多个遗传操作。

(3)遗传操作:对每个子种群执行遗传操作,包括选择、交叉、变异和局部优化等步骤。

(4)结果本地化:一旦所有线程完成遗传操作,使用 RDD 的 collect 算子将所有线程的结果汇总并本地化,生成 1 个包含多个可能的解决方案的集合。然后,从中选择具有最短路径的解作为问题的最优解,这个过程确保了在并行处理中找到最优的解决方案。

3 实验与结果分析

本实验用 Python 语言实现 KSAGA 算法,在配置为 Spark2.4.0、Intel(R) Xeon(R) Gold 6154 CPU @3.00 GHz 的环境下进行,选取了不同规模的 TSPLIB 标准数据集进行仿真实验。

3.1 参数设置及其影响

KSAGA 中有 4 个重要的参数:交叉、变异算子阈值 Pm,2 个变异算子的阈值 Pt,交叉算子的移除比例 Pr,以及变异算子 1 的移除比例 Pw,以上参数取值都会影响整个算法的性能。本文参考王勇臻等[12]的做法,选取 eil51 实例(旅行商数量为 5)研究参数对 KSAGA 算法性能的影响。各参数取值如表 1 所示。

表 1 参数取值

Table 1 Parameter value

参数	数值组 1	数值组 2	数值组 3	数值组 4
Pm	0.5	0.6	0.7	0.8
Pt	0.4	0.5	0.6	0.7
Pr	0.2	0.3	0.4	0.5
Pw	0.4	0.5	0.6	0.7

算法在每种参数组合下运行 20 次,本文 GA 的种群数量设置为 20,最大迭代次数为 200,将算法运行 20 次后得到的平均值作为评价指标,实验后得到的正交表如表 2 所示。

根据表 1、2 求得各参数在不同取值下总距离的平均值,计算这 4 个参数对应极差,通过极差的值可以反映本文算法中 4 个参数的重要程度,各参数对算法性能的影响如表 3 所示。

根据表 3 的参数性能测试实验结果可得以下结论。

表 2 正交结果及均值

Table 2 Orthogonal results and mean value

序号	Pm	Pt	Pr	Pw	均值
1	1	1	1	1	473.25
2	1	2	2	2	474.00
3	1	3	3	3	475.85
4	1	4	4	4	475.40
5	2	1	2	3	474.90
6	2	2	1	4	474.25
7	2	3	4	1	475.25
8	2	4	3	2	475.00
9	3	1	3	4	474.20
10	3	2	4	3	475.70
11	3	3	1	2	477.55
12	3	4	2	1	476.90
13	4	1	4	2	477.15
14	4	2	3	1	478.40
15	4	3	2	4	476.70
16	4	4	1	3	477.25

表 3 各参数响应值

Table 3 Response values of each parameter

数值组	响应值			
	Pm	Pt	Pr	Pw
1	474.63	474.88	475.58	475.95
2	474.85	475.45	475.62	475.93
3	476.09	475.95	475.49	475.93
4	477.38	475.83	475.74	475.48
极差	2.75	1.07	0.25	0.47
重要程度	1	2	4	3

(1)参数 Pm 的极差在几个参数中最大,说明交叉、变异算子阈值对算法性能影响最大。在本文中,Pm 取值越大执行交叉算子的概率越高,但是随着 Pm 增加,解的质量变差,这是由于 KSAGA 增加了局部优化算子,在加快收敛速度的同时,可能会使算法陷入局部最优,因此在 0.5~0.8 时 Pm 值越小越好。

(2)对算法性能影响第二大的参数是 2 个变异算子的阈值 Pt,Pt 在取 0.6 时效果最好,Pt 越大,算法更偏向于执行变异算子 1;Pt 过小,变异算子 2 破坏程度不足以使算法跳出局部最优。

(3)Pw 在算法中的影响较小,Pw 会影响解空间的搜索范围,Pw 取值为 0.7 时解的质量更好。

(4)Pr 在 4 个参数中对算法的性能影响最小,此参数越大需要进行贪心操作的城市增多,会影响算法运行效率,但在算法中对解的质量影响不大。

综合上述分析,算法的交叉、变异算子阈值

Pm、2 个变异算子的阈值 Pt,交叉算子的移除比例 Pr,以及变异算子 1 的移除比例 Pw 设置如下:Pm = 0.5,Pt = 0.6,Pw = 0.7,Pr = 0.4。

3.2　改进的变异策略有效性验证

本文提出了 2 个变异算子,为验证策略组合的有效性设计了 4 个模型,在 att48 测试集上进行实验。表 4 展示了不同模型(旅行商数量为 5)进行 5 次实验后的总路程均值结果,其中,A 为基础模型,"√"表示算法中保留该算子,模型的最优结果加粗表示,下同。

表 4　变异策略有效性验证表

Table 4　Validation table for the effectiveness of mutation strategy

模型	变异算子 1	变异算子 2	结果均值
A			37 487.2
B	√		37 404.4
C		√	37 364.8
D	√	√	**37 171.0**

观察表 4 可以发现,未加入任何变异算子的模型 A 得到的均值结果为 37 487.2,单独保留变异算子 1 的模型 B 或保留变异算子 2 的模型 C 相对于 A 模型的解更精确;进一步将 2 个变异操作组合后的模型 D 与模型 B 和 C 相比得到了更优解,与模型 A 相比二者相差了 316.2。综上所述证明了变异策略的有效性。

3.3　算法的求解性能测试

为了验证 KSAGA 算法的性能,本文选取了 4 个不同规模的数据集 eil51、eil76、kroD100、kroA150(设置不同旅行商数量)进行实验,不同数据集旅行商数量 m 取值参考其他文献中的常用值设置。与 ACO[13]、GA[14]、NSGA-Ⅱ、SPKSA[15] 以及自适应大规模邻域搜索算法[16](adaptive large-scale neighborhood search,ALNS)进行对比。为确保实验的公平性和可比性,所有算法采取统一的停止准则:最大迭代次数设置为 1 000 次,在算法终止时的最优解作为最终结果。各对比算法重要参数取值按照对应文献设置:GA 种群规模为 320,每次选择 8 个个体进行锦标赛选择;ACO 设置信息素重要程度为 1,路径可见性重要程度为 3;NSGA-Ⅱ种群规模为 100,变异概率为 0.05;ALNS 移除城市的最大比例为 0.3,最小移除量为 4 个;SPKSA 的初始温度为 100,降温系数为 0.99,浮动间隔为 100。实验都在配置为 Intel(R) Xeon(R) Gold 6154 CPU @3.00 GHz 的环境下进行。其中 ACO 和 SPKSA 的重要更新公式按其对应文献设置。

由于对比算法 ACO 在原文献中限制了旅行商对城市的最大访问个数,为保证算法求解目标相同,在对该限制修改后进行仿真实验。其中 NSGA-Ⅱ虽然为多目标优化算法,但该文献实验部分提到由于该算法交叉算子是基于最短距离的,因此在某些数据集下实验结果在最小化总距离的目标上相较于其他算法表现出明显的优势,因此本文将算法针对单目标 MTSP 做了优化后将其作为对比算法。各算法的实验结果如表 5 所示。其中,参数 m 为旅行商数量,同一参数设置下得到的最优解加粗表示。从表 5 看出,GA 使用了染色体分组编码方式,便于交叉、

表 5　6 种算法运行结果对比

Table 5　Comparison of running results of six algorithms

数据集	m	GA		ACO		SPKSA		ALNS		NSGA-Ⅱ		KSAGA	
		均值	最优	均值	最优	均值	最优	均值	最优	均值	最优	均值	最优
eil51	3	502	482	493	489	498	491	465	461	465	456	**450**	**450**
	5	578	560	535	519	563	558	530	524	520	511	**472**	**470**
	10	756	706	724	709	782	749	770	752	644	618	**590**	**586**
eil76	3	746	717	603	598	628	624	584	579	579	577	**563**	**560**
	5	848	803	674	664	746	737	662	658	639	627	**593**	**588**
	10	1 117	1 012	890	885	990	962	967	951	835	815	**685**	**684**
kroD 100	3	37 488	35 484	24 792	24 372	29 806	28 472	24 956	24 374	23 756	23 295	**22 198**	**22 074**
	5	43 616	38 326	27 321	26 517	34 778	34 022	29 433	28 801	26 581	26 157	**23 607**	**23 285**
	10	60 085	51 535	37 022	36 327	45 196	44 740	46 206	44 930	37 036	35 288	**28 269**	**28 017**
	20	78 070	74 603	64 373	63 964	70 173	67 782	78 646	77 863	57 843	55 519	**41 111**	**40 223**
kroA 150	3	65 090	61 215	31 081	30 894	29 806	28 472	29 570	29 446	30 313	29 345	**27 313**	**27 094**
	5	73 374	69 987	33 801	33 405	34 778	34 022	34 182	34 120	32 503	31 562	**28 937**	**28 522**
	10	88 625	84 125	41 697	41 123	45 196	44 740	46 775	46 504	39 229	38 404	**33 306**	**32 814**
	20	108 694	103 464	61 821	60 796	70 173	67 782	73 790	73 159	59 951	57 220	**45 372**	**44 811**
	30	120 395	115 477	82 992	82 490	94 551	92 734	100 644	99 376	82 144	79 615	**59 014**	**58 511**

变异操作的实现,但由于搜索空间不足导致解的质量欠佳,同时 GA、ACO 和 NSGA-Ⅱ随机产生初始解的方式大大影响了算法的搜索效率。NSGA-Ⅱ使用了两部分染色体编码,同时设计了适用于此编码方式的交叉、变异算子,虽然加快了算法收敛速度,但是算法后期容易陷入局部最优。SPKSA 算法收敛速度快,在求解每个组内城市的访问顺序有优秀的表现,但由于此算法没有对不同旅行商之间的城市分组进行优化,因此求解质量一般。而 ALNS 算法在小规模数据集上表现较好,但随着旅行商数量及城市数量的增加性能逐渐下降,主要原因是大规模邻域搜索算法容易陷入局部最优。在不同规模的数据集和不同旅行商数量的任务中,由于 KSAGA 算法结合了 SA 和 GA 的优势,同时在局部优化算子的加持下,算法收敛速度大大加快,且变异算子可以在迭代过程中使算法跳出局部最优,因此 KSAGA 在多次实验中相较于其他算法都可以得到更优质的解。

随着旅行商数量、城市规模的增加,其他算法的求解性能普遍发生下降,当数据集规模越大,或旅行商数量规模更大时,KSAGA 的优势更为明显。

3.4 限制最大访问城市个数后算法性能测试

在限制销售人员可以访问的最大节点数之后,选择在测试集 pr76、pr152 和 pr226 上进行实验(独立运行 10 次后取最优值),与 ACO、NMACO[17]算法比较来验证算法在限制访问城市数量后针对最小化总距离目标的有效性,实验结果如表 6 所示。

表 6 限制最大访问数量后的性能测试结果

Table 6 Performance test results after limiting the maximum number of accesses

实例	n	m	u	测试结果		
				ACO	NMACO	KSAGA
pr76	76	5	20	178 597	**157 413**	160 629
pr152	152	5	40	130 953	127 781	**122 648**
pr226	226	5	50	167 646	167 239	**166 827**

表 6 中 n 为城市数量;m 为旅行商数量;u 为每个旅行商对城市的最大访问数量。在 pr76 测试集中,表现最优的算法为 NMACO,其次是 KSAGA 方法。但是随着问题规模的扩大,KSAGA 算法展现出优势,逐步超越了 NMACO 算法,这不但证明了提出方法的有效性,还证明了所提方法对大规模问题求解的有效性。

3.5 算法收敛速度测试

由于 GA、NSGA-Ⅱ以及本文所提出的方法均构建在遗传算法的基础上,因此通过对这些算法的收敛性进行比较,以评估它们在求解 MTSP 过程中的性能表现。测试数据集选择 TSPLIB 实例 berlin52,旅行商数量设置为 5,3 种算法收敛效果如图 6 所示。

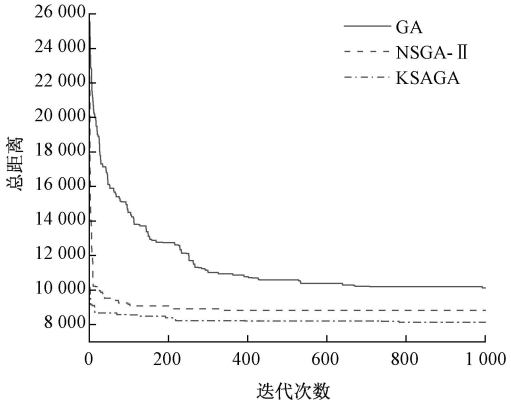

图 6 相同迭代次数下算法的收敛曲线

Figure 6 Convergence curve of the algorithm with the same number of iterations

从图 6 可以看出,在相同的迭代次数下,KSAGA 都有比 GA 和 NSGA-Ⅱ更快的收敛速度。所提算法通过聚类和贪心操作初始化一个可行解的方式以及引入局部优化算子,都加快了算法的收敛效率。在 NSGA-Ⅱ已经开始趋于平稳时,KSAGA 中依然还有更好的解产生,种群的多样性得到了保证。在相同条件下,KSAGA 从收敛速度及精度上都优于其他 2 个算法。

4 结论

针对单站点多旅行商的最小化总距离问题,本文采用了分阶段优化策略,结合 SA 和 GA 这 2 个算法对问题进行求解,并提出了适用于分组遗传算法的交叉、变异算子。在 Spark 环境下利用并行机制加快遗传算法的运行效率,同时增加解的搜索空间。通过多次实验确定了遗传算法重要参数的取值,与其他 2 个遗传算法在相同迭代次数下对收敛速度进行对比,实验验证了 KSAGA 的收敛速度更快,且求解质量相比于其他算法得到很大提升。

由于旅行商数量达到一定阈值后,继续增加不会使效率明显提高反而可能会造成资源浪费,旅行商最佳分配数量在参考其他文献得到取值范围后,分别取范围内的值进行多次仿真实验,通过分析各个取值下算法的收敛速度和结果精度确定单站点多旅行商问题中的最佳旅行商数量,但由于此方法在大规模数据集中缺乏有效性,如何在不同数据集中确定最优旅行商数量的问题仍需进一步深入研究。目前对于 MTSP 的研究主要集中在如何最小化所有

销售商的距离之和上,但实际应用中,若优先考虑的是所有的城市访问时间而不是距离、业务员的工作量或者是推销员的成本限制,则单站点多旅行商的求解目标变换为最小化最大访问距离求解,因此后续工作可基于此对 KSAGA 算法做进一步的改进,使其更加贴近于实践应用。

参考文献:

[1] 苏守宝, 赵威, 李智. 求解加权 MTSP 问题的 CUDA 并行群智能方法[J]. 郑州大学学报(工学版), 2021, 42(6): 34-41.
SU S B, ZHAO W, LI Z. CUDA-based parallel swarm intelligence method for solving weighted MTSP[J]. Journal of Zhengzhou University (Engineering Science), 2021, 42(6): 34-41.

[2] MOHAMMAD S, MAJID Y, NARGES M. An effective genetic algorithm for solving the multiple traveling salesman problem[J]. Journal of Optimization in Industrial Engineering, 2011, 4: 73-79.

[3] YANG S, SHAO Y F, ZHANG K. An effective method for solving multiple travelling salesman problem based on NSGA-Ⅱ[J]. Systems Science & Control Engineering, 2019, 7(2): 108-116.

[4] HARRATH Y, SALMAN A F, ALQADDOUMI A, et al. A novel hybrid approach for solving the multiple traveling salesmen problem[J]. Arab Journal of Basic and Applied Sciences, 2019, 26(1): 103-112.

[5] 董建华, 王国胤, 雍熙, 等. 基于 Spark 的标准化 PCA 算法[J]. 郑州大学学报(工学版), 2017, 38(5): 7-12.
DONG J H, WANG G Y, YONG X, et al. Normalized PCA algorithm based on Spark[J]. Journal of Zhengzhou University (Engineering Science), 2017, 38(5): 7-12.

[6] ZHENG J Z, HONG Y W, XU W C, et al. An effective iterated two-stage heuristic algorithm for the multiple traveling salesmen problem[J]. Computers & Operations Research, 2022, 143: 105772.

[7] 冯兴杰, 王文超. Hadoop 与 Spark 应用场景研究[J]. 计算机应用研究, 2018, 35(9): 2561-2566.
FENG X J, WANG W C. Survey on Hadoop and Spark application scenarios[J]. Application Research of Computers, 2018, 35(9): 2561-2566.

[8] 孙鉴, 刘淞佐, 武晓晓, 等. 基于 Spark 的并行模拟退火算法求解 TSP[J]. 电子测量技术, 2022, 45(4): 53-58.
SUN J, LIU S Z, WU X X, et al. Solving TSP based on Spark-based parallel simulated annealing algorithm[J]. Electronic Measurement Technology, 2022, 45(4): 53-58.

[9] AL-FURHUD M A, AHMED H Z. Genetic algorithms for the multiple travelling salesman problem[J]. International Journal of Advanced Computer Science and Applications, 2020, 11(7): 553-560.

[10] 杨帅. 求解多旅行商问题的进化多目标优化和决策算法研究[D]. 武汉: 武汉科技大学, 2021.
YANG S. Research on evolutionary multi-objective optimization and decision making for solving multiple traveling salesman problem[D]. Wuhan: Wuhan University of Science and Technology, 2021.

[11] 孙冰, 王川, 杨强, 等. 面向多起点均衡多旅行商问题的进化算法[J]. 计算机工程与设计, 2023, 44(7): 2030-2038.
SUN B, WANG C, YANG Q, et al. Improved evolutionary algorithm for balanced multiple traveling salesmen problem with multiple starting points[J]. Computer Engineering and Design, 2023, 44(7): 2030-2038.

[12] 王勇臻, 陈燕, 于莹莹. 求解多旅行商问题的改进分组遗传算法[J]. 电子与信息学报, 2017, 39(1): 198-205.
WANG Y Z, CHEN Y, YU Y Y. Improved grouping genetic algorithm for solving multiple traveling salesman problem[J]. Journal of Electronics & Information Technology, 2017, 39(1): 198-205.

[13] PAN J J, WANG D W. An ant colony optimization algorithm for multiple travelling salesman problem[C]//First International Conference on Innovative Computing, Information and Control. Piscataway: IEEE, 2006: 210-213.

[14] KIRÁLY A, ABONYI J. A novel approach to solve multiple traveling salesmen problem by genetic algorithm[J]. Computational Intelligence in Engineering, 2010, 313: 141-151.

[15] 孙鉴, 李昊, 刘淞佐, 等. 基于 Spark 的并行 k 均值聚类模拟退火算法求解 MMTSP[J]. 电子测量技术, 2022, 45(20): 53-60.
SUN J, LI H, LIU S Z, et al. Spark-based parallel k-means clustering simulated annealing algorithm to solve MMTSP[J]. Electronic Measurement Technology, 2022, 45(20): 53-60.

[16] HUANG K Y, MA J B, LIU X Y. Research on vehicle route planning with capacity limitation based on adaptive large-scale neighborhood search algorithm[C]//2021 6th International Symposium on Computer and Information Processing Technology (ISCIPT). Piscataway: IEEE, 2021: 6-10.

[17] YOUSEFIKHOSHBAKHT M, DIDEHVAR F, RAHMATI F. Modification of the ant colony optimization for solving the multiple traveling salesman problem[J]. Romanian Journal of Information Science and Technology, 2013, 16(1): 65-80.

(下转第 94 页)