计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210400249

空间众包任务的路径动态调度方法

沈 彪 沈立炜 李 弋
复旦大学计算机科学技术学院 上海 201203
上海市数据科学重点实验室(复旦大学) 上海 201203
(1105125966@qq.com)

摘 要 空间众包用于解决带时空约束的线下众包任务,近几年得到了快速发展。任务调度是空间众包的重要研究方向,难点在于调度过程中任务和工作者的动态不确定性。为了高效地进行任务路径动态调度,提出了同时考虑任务和工作者的不确定性的空间众包任务路径动态调度方法,该方法进行了3方面的改进。首先,扩展了调度需要考虑的因素,除了考虑新增任务的时空属性不确定性之外,还考虑了新增工作者的交通方式和时空属性的不确定性。其次,对调度策略进行改进,通过使用聚合调度策略,对动态新增任务先进行聚合处理,随后再进行任务分配和路径优化,相比传统非聚合调度计算时间显著减少。最后,对调度算法进行改进,基于传统遗传算法,将任务分配和路径优化操作迭代进行,相比先进行任务分配再进行路径优化的调度算法,提高了获取最优结果的准确性。此外,文中设计并实现了基于真实地图导航的空间众包任务路径动态调度模拟平台,并基于该平台验证了所提方法的有效性。

关键词:空间众包;任务分配;任务调度;路径规划;遗传算法
中图法分类号 TP311

Dynamic Task Scheduling Method for Space Crowdsourcing

SHEN Biao,SHEN Li-wei and LI Yi
School of Computer Science,Fudan University,Shanghai 201203,China
Shanghai Key Laboratory of Data Science (Fudan University),Shanghai 201203,China

Abstract Space crowdsourcing is used to solve offline crowdsourcing tasks with time and space constraints,and it has developed rapidly in recent years. Task scheduling is an important research direction of space crowdsourcing. The difficulty lies in the dynamic uncertainty of tasks and workers in the scheduling process. In order to efficiently perform task scheduling,a dynamic task scheduling method for space crowdsourcing that considers the uncertainty of tasks and workers at the same time is proposed. The method has been improved in three aspects. First,the factors that need to be considered for scheduling are expanded. In addition to considering the uncertainty of the temporal and spatial attributes of the newly added tasks,it also considers the uncertainty of the transportation mode and temporal and spatial attributes of the newly added workers. Then,the scheduling strategy is improved. By using the aggregate scheduling strategy,the dynamically added tasks are aggregated first,and then the task allocation and path optimization are performed. Compared with the traditional non-aggregated scheduling,the calculation time is significantly reduced. The last aspect is to improve the scheduling algorithm. Based on the traditional genetic algorithm,the task allocation and path optimization operations are performed iteratively. Compared with the scheduling algorithm that first allocates tasks and then optimizes the path,it improves the accuracy of the optimal results. In addition,a simulation platform for dynamic scheduling of space crowdsourcing task paths based on real map navigation is designed and implemented,and the method is verified by this platform.

Keywords Space crowdsourcing,Task allocation,Task scheduling,Route planning,Genetic algorithm

1 引言

随着互联网技术的不断发展,众包模式得到了越来越广泛的应用。众包由工作者、请求者和服务器组成。请求者根据自身需求提交任务,工作者主动接收或根据服务器分配合适的任务并执行,请求者与工作者通过服务器进行反馈沟通。而随着可定位移动设备的发展和普及,众包逐渐发展出了各式各样的线下服务,因此产生了一种新的众包模式,即空间众包。空间众包被认为是一种新型的软件服务[1],具有时空约束的特点,需要工作者到达任务规定的线下位置去提供服务,

到稿日期:2021-04-23 返修日期:2021-06-08
基金项目:国家高技术研究发展计划(863计划)(2018YB1004800)
This work was supported by the National High Technology Research and Development Program of China(2018YB1004800).
通信作者:沈立炜(shenliwei@fudan.edu.cn)

从而完成任务并获得奖励。现实世界中存在很多空间众包，例如众包物流（如达达）、食品配送（如美团众包）、打车（如滴滴打车）和拍照（如谷歌地图的街道拍照）等。

任务调度问题是空间众包一个重要的研究方向，即在将任务分配给特定工作者[2]的基础上安排任务的执行顺序[3]。将众包平台上出现的任务分配给合适的工作者并为对应工作者安排合适执行路径能够实现不同的优化目标，提高平台效率。大多现实的空间众包中任务都是动态实时出现的，工作者也能随时注册自己的服务，然而这些动态性会影响空间众包平台的路径优化调度。目前已有很多路径优化研究考虑了请求任务的动态性，但仍存在以下不足之处。

在空间众包中，工作者可以灵活地选择自己的工作时间以及起点和终点，因此众包平台中工作人员的数量以及工作者的位置可能随着时间动态变化。另外，在众包服务中，工作者可根据自身的情况选择合适的交通方式，如步行、骑自行车、骑电动车或乘汽车等。由于行人不能上高架桥，车辆无法在小路上行使等原因，在相同起点和终点的情况下，不同的交通方式会有不同的行驶路径和距离。而行驶路径和距离的不一致会对空间众包的任务分配和路径规划产生影响。而在已有的动态车辆路径优化问题的研究中，工作者的数量以及起点和终点都是确定的。例如 Sun 等[4]仅考虑了单工作者的路径规划问题，没有考虑多工作者任务分配和路径规划问题。Tao 等[5]虽然考虑了多工作者的情形，但是没有考虑不同工作者的不同交通方式对路径规划的影响，其假设过于简单，不符合实际情况。本文针对这些特点，提出了考虑众包工作者的不确定性的众包任务路径动态调度方法，及时响应工作者的动态变化，更新调度系统中工作者的信息，在路径规划调度中根据工作者实时位置信息和交通方式，并基于真实地图导航获取行驶路径和距离数据，来对工作者的任务执行路径进行规划调度。

空间众包任务的路径规划问题属于 NP 难问题，现有的解决方法大多用到了启发式算法，相比精确式算法节省了运行时间，但是时间效率还有待提高。对于效率要求较高的众包服务平台来说，更高效地获取计算结果将会给平台节省开支，带来更大的收益。在近几年的研究中，大多数动态路径规划算法都是对新增动态任务进行单独处理完成调度。例如文献[6-9]对一段时间周期内出现的动态任务分别确认合适的工作者，并通过路径优化算法确认每一个任务合适的执行顺序。将所有任务单独加入路径规划调度会产生较大的解空间，从而增加求解最优解的计算时间。针对这个问题，本文提出了聚合调度方法：对位置相近动态任务先进行任务聚合处理，形成聚合任务集，然后再对聚合任务集进行统一全局调度，分配给合适的工作者，并安排每一个工作者的聚合任务集执行顺序，最后在合适的时间对聚合任务集内的任务进行局部调度，为工作者安排具体的任务执行路径。相比非聚合调度，本文方法可避免重复计算，减少解空间，缩短求解最优解的计算时间。

另外，传统的任务路径动态调度方法[5]是先对动态新增任务进行任务分配，将任务分配给合适的工作者，再对工作者的所有任务进行路径优化。先分配的任务没有考虑后续需要分配的任务的影响，而是直接决定了对应工作者，这降低了调度的准确性。本文使用任务分配和路径优化迭代运行的方法，对动态任务进行调度，可得到更优的调度结果。

本文提出了基于真实地图导航的空间众包任务路径动态调度模拟平台，根据工作者不同的交通方式，提供不同的真实地图导航路径，在模拟系统中针对不同的交通方式对工作者进行模拟移动。

本文第 2 节介绍了相关工作；第 3 节对本文提出的空间众包任务路径动态调度方法进行整体介绍，说明该方法中主要操作的决策变量、目标函数和约束条件，构建数学模型；第4节详细介绍了所提方法中用到的求解策略和算法设计；第 5 节对空间众包任务路径动态调度模拟平台进行了介绍；第 6 节开展了实验研究，对所提方法的有效性进行了说明。

2 相关工作

根据空间众包任务路径规划问题中任务、工作者或环境是否已知，可以将路径规划问题分为静态路径规划问题和动态路径规划问题[3,9]。目前的路径规划研究主要集中在静态路径规划问题上，然而动态路径规划问题更符合空间众包的实际情况。下面分别介绍这两种路径规划问题的相关研究。

国内外对静态路径规划问题相关研究如下：为实现最大化任务分配总数或最大化工作者收益，即效用最大化，Deng 等[11]针对空间众包中任务和工作者输入参数已知的情况下，考虑差旅费用和任务的截止日期来求解最大有效任务序列；Costa 等[12]研究了最大化总收益的静态路径规划，认为工作者会综合考虑利益和旅行费用，通过启发式方法提出绕行的路径推荐方案。为实现成本最小化的任务分配，Zhang 等[13]应用粒子群算法求解了以最小路程和车辆为目标的带时间约束的静态路径优化问题，为了将粒子群算法应用于路径优化问题，对算法进行了离散化，将路径段设定为粒子速度，并设计了速度和位置的更新规则，引入了进化算子和遗传算法来提升寻优性能。

上述静态任务分配和路径规划算法对静态问题有较好的效果，但是实际空间众包中的任务和工作者等信息通常是未知的，因此静态路径规划在实际空间众包中作用较小[3,14]。

国内外对动态路径规划问题的相关研究如下：针对请求者任务需求的动态不确定性，Sun 等[4]提出了一种为空间众包中单个工作者规划路径的方案，该方案考虑了工作者对任务选择的 3 个影响因素，通过贪心算法进行求解来保证最大化工作者的总利润；Li 等[15]研究了动态地为单个工作者安排任务执行路径的问题，保证工作者可以按时到达任务所在位置并获取最大利益，通过距离最近和最早截止日期启发式方法和修剪策略来快速求解；Coslovich 等[16]采用两阶段路径规划算法，第一阶段是在部分需求任务已知的情况下通过静态路径规划求得初始路径，第二阶段是在出现动态新增任务的情况下将新增任务插入已有路径当中，静态路径规划计算次数的增多可以减少动态新增任务实时处理的计算花费时间；为求得最大化总利润，Tao 等[5]分别使用延迟处理和快速处理两种启发式算法来进行求解，在延迟处理中动态新增任务

不会立即分配给工作者，而快速处理中动态新增任务到达时会更新工作者的路径；Liu[6]通过周期性延迟处理的方式对动态新增任务进行路径规划调度，为动态任务选择合适的工作者，然后再对工作者的任务执行路径进行优化；Asghari 等[17]为获得最大化分配总数，解决延迟批处理无法实时求解结果的问题，提出了工作者为动态新增任务竞标的实时分配方法，通过平台确认最高竞标者作为对应任务的工作者；Hu 等[18]首先通过预测不确定任务需求信息来求解预测路径，之后再基于预测路径对动态新增任务进行路径规划，更新任务执行路径以完成动态任务需求。针对工作者及设备的动态不确定性，Mu 等[19]针对路径规划中车辆发生故障的问题，通过使用两种禁忌搜索算法来快速生成新的路径，降低意外造成的成本。针对交通路网的动态不确定性，Güner 等[20]对路网的拥堵情况进行分类，分为可预测拥堵和不可预测拥堵，并在不同情况下规划动态路由。

上述动态路径规划的相关工作对空间众包任务路径动态调度具有指导意义，但是没有考虑到空间众包中工作者会动态新增且不同工作者使用不同交通方式对路径规划调度的影响；另外，将动态新增任务单独加入全局调度的方式会导致计算效率不高，并且部分研究通过先进行任务分配再进行路径优化的方式对动态新增任务进行调度，没有考虑后分配任务的影响，降低了最优结果的准确性。综上所述，当前相关研究工作不能很好地解决本文所提出的工作者和任务存在动态不确定性的空间众包任务路径动态调度问题。因此，本文提出了一种空间众包任务路径动态调度方法，同时考虑了空间众包中任务和工作者的不确定性，并结合聚合调度和迭代调度方法来提高调度的效率和有效性。

3　调度方法及模型构建

3.1　问题描述

空间众包任务路径动态调度问题是将众包平台中请求者提出的任务和工作者进行匹配，并为工作者安排合适的任务执行路径以获得所有工作者路径总长度最短的调度优化问题。空间众包任务路径动态调度问题包括以下 3 个组成元素。

(1)请求者发布的任务：请求者在众包平台上发布单点任务，即每个任务包括且仅包括一个经纬度位置。新增任务将等待众包平台分配给指定工作者，由对应工作者到达任务所在位置点完成任务。

(2)工作者：工作者在众包平台上注册自己的服务，每个工作者都有对应的起点位置和终点位置，并且不同工作者可以选择自己合适的交通方式。工作者将等待众包平台分配任务，并通过众包平台规划的轨迹路径依次完成任务。

(3)任务执行路径：众包平台将任务分配给对应工作者后会为每一个工作者安排任务执行路径，工作者根据任务执行路径的顺序依次完成请求者发布的任务。

空间众包任务路径动态调度问题不同于静态调度问题，静态调度问题中提前明确了工作者和任务，而动态调度问题中会存在以下不确定性。

(1)工作者上线时间和数量的不确定性：众包工作者可随时在平台中注册自己的服务，因此众包平台中工作者的数量可能随时发生改变。

(2)工作者位置的不确定性：传统的物流配送路径优化问题中，所有工作者的起点和终点都是配送中心。而空间众包中工作者的起点位置和终点位置并不是唯一和固定的，众包工作者在注册时会以当前位置作为起点位置，并自由设置终点位置，如家庭所在位置。

(3)工作者交通方式的不确定性：众包工作者可以根据自己的情况选择合适的交通方式，如步行、骑自行车、骑电动车、开汽车等。不同的交通方式可能会导致两点间存在不同的路径长度，因此需要考虑不同交通方式对路径规划产生的影响。

(4)请求者发布任务的时间和数量的不确定性：众包请求者需要发布的任务不是提前预知的，请求者可随时在平台中提交任务。

(5)请求者发布任务的位置的不确定性：众包请求者需要发布的任务的位置不是提前预知的，当任务发布后才能得知需要服务的位置。

空间众包任务路径动态调度需要将以上不确定性因素考虑在内，来对空间众包任务进行路径调度。

3.2　方法设计

为及时响应多种不确定性事件，并准确高效地完成调度，本文提出了一种空间众包任务路径动态调度方法，具体如图 1 所示。其中圆角矩形表示开始事件，实线表示流程进行方向，矩形表示处理操作，灰色矩形为本文方法的主要操作，虚线和斜四边形表示数据传递，虚线框表示数据集合，菱形表示决策判断。

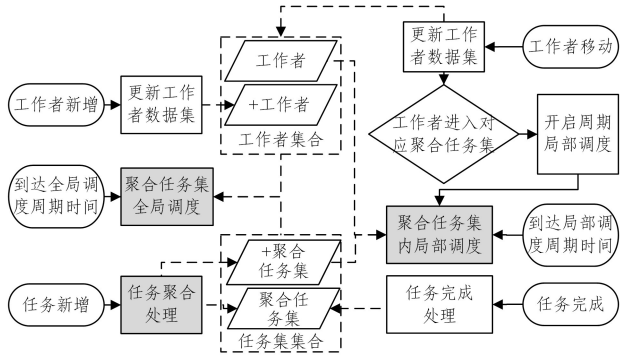


图 1　空间众包任务路径动态调度方法

Fig. 1　Dynamic task scheduling method for space crowdsourcing

众包平台中的不确定性和动态事件包括请求者在众包平台上发布任务、工作者在众包平台上注册服务、工作者当前位置发生改变以及工作者完成请求者发布的任务。

对于请求者在众包平台上发布任务，当前相关研究主要有两种策略：一种是立刻分配动态任务，为任务分配合适的工作者；另外一种是延迟分配动态任务，等待一段时间，或者等待动态任务达到一定数量再统一对已出现的动态任务进行任务分配处理。与立刻分配动态任务相比，延迟分配可降低平台调度成本[6]。本文参考了延迟分配策略的思想，提出了任务聚合调度方法，当动态任务出现时，首先会将对应任务通过聚合算法加入已有的聚合任务集中，或者新增新的聚合任务

集。随后通过周期性聚合任务集全局调度来为未匹配工作者的聚合任务集安排合适的工作者。

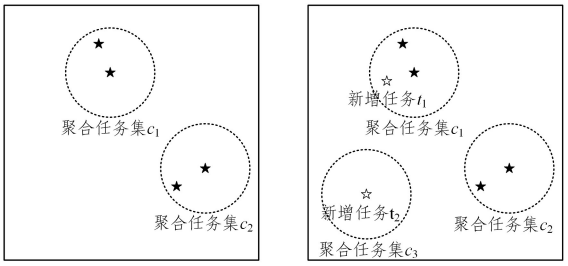
对于工作者在众包平台上注册服务，本方法会将对应工作者加入当前工作者集合中，记录工作者当前位置、终止位置和交通方式，在后续周期性聚合任务集全局调度中基于真实地图导航来为不同位置和不同交通方式的新增工作者安排工作任务。

当工作者位置发生改变时，更新工作者集合中的工作者信息，用于后续参与聚合任务集全局调度，以及判断工作者是否需要开启周期性聚合任务集内局部调度。

当工作者移动到某一任务点并完成任务后进行任务完成处理。首先删除工作者路径上的该任务并判断该任务所属的聚合任务集是否还存在其他未完成任务。当对应聚合任务集已无其他任务，则删除对应聚合任务集，并安排工作者前往下一聚合任务集区域，若无下一聚合任务集则原地等待后续调度。当对应聚合任务集还存在其他任务时，则安排工作者执行下一个任务。

为提高算法计算效率，本文提出了聚合调度策略来对动态任务进行调度，主要包括以下 3 个操作。

(1)任务聚合处理：如图 2 所示，对于动态新增任务，以距离其最近的聚合任务集为目标，找到最近的已有聚合任务集，决策是否将该动态任务加入对应聚合任务集，如果动态任务在聚合任务集范围内则加入，如果不在，则根据该动态任务的位置新建聚合任务集，并将其加入新增聚合任务集中。

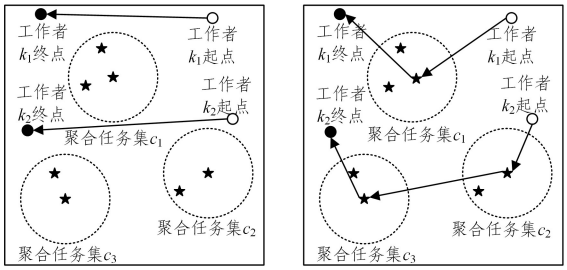


(a)n 时刻 (b)$n+1$ 时刻

图 2 任务聚合处理

Fig. 2 Task clustering

(2)聚合任务集全局调度：如图 3 所示，对于动态事件响应中实时更新的工作者数据集和任务聚合处理中得到的聚合任务集，以求得最短总路径为目标，将聚合任务集通过决策分配给合适的工作者，并确定每个工作者的聚合任务集执行顺序，调度需要满足所有聚合任务集都得到且仅得到一次处理和工作者最终路径为前往工作者终点的约束。

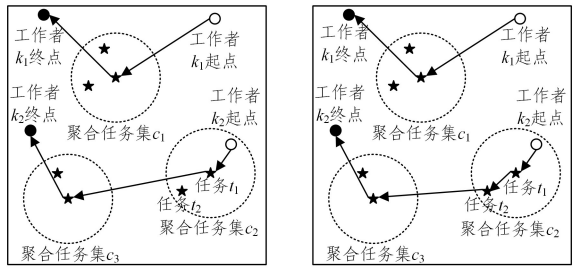


(a)聚合任务集全局调度之前 (b)聚合任务集全局调度之后

图 3 全局调度

Fig. 3 Global scheduling

(3)聚合任务集内局部调度：如图 4 所示，在对任务进行聚合处理并将聚合任务集分配给合适的工作者之后，当工作者进入对应聚合任务集范围内时，以任务集内任务总路径最短为目标，为工作者决策合适的聚合任务集内任务执行顺序，调度需要满足聚合任务集内所有任务都得到且仅得到一次服务和工作者在该聚合任务集内的最后路径为前往下一聚合任务集或工作者终点的约束。



(a)聚合任务集局部调度之前 (b)聚合任务集局部调度之后

图 4 局部调度

Fig. 4 Local scheduling

3.3 模型构建

聚合调度方法中的 3 个主要操作都存在不同的决策、目标和约束。下面将从模型符号与参数、决策变量、目标函数和约束条件几个方面进行形式化描述。

3.3.1 模型符号与参数

表 1 列出了本文模型所涉及的符号及其意义。

表 1 模型符号与意义

Table 1 Model symbol and meaning

符号	符号意义
φ	任意时刻
τ	聚合任务集全局调度时刻
$K(\varphi)$	φ 时刻工作者集合
$T_n(\varphi)$	φ 时刻新增任务集合
$T_w(\varphi)(k)$	φ 时刻工作者 k 需进行局部调度的聚合任务集内任务集合
$C(\varphi)$	φ 时刻聚合任务集集合
$C_w(\varphi)$	φ 时刻待调度聚合任务集集合
$l_{ks}(\varphi)(k)$	φ 时刻工作者 k 当前位置
$l_{ke}(\varphi)(k)$	φ 时刻工作者 k 终点位置
$l_{kn}(\varphi)(k)$	φ 时刻工作者 k 下一个聚合任务集或终点位置
$l_t(t)$	任务 t 所在位置
$l_c(c)$	聚合任务集 c 中心点位置
$l_{ks}(\varphi)$	φ 时刻工作者当前位置集合
$l_{ke}(\varphi)$	φ 时刻工作者终点位置集合
$l_{tn}(\varphi)$	φ 时刻新增任务位置集合
$l_{tw}(\varphi)(k)$	φ 时刻工作者 k 需进行局部调度的聚合任务集内任务位置集合
$l_c(\varphi)$	φ 时刻聚合任务集位置集合
$l_{cw}(\varphi)$	φ 时刻待调度聚合任务集位置集合
$d_s(i)(j)$	位置 i 到位置 j 的直线距离
$d_g(k)(i)(j)$	工作者 k 通过对应交通方式从位置 i 到位置 j 的导航距离
D	聚合任务集半径长度

3.3.2 任务聚合处理

下面给出任务聚合处理的决策变量、目标函数和约束条件。

(1)决策变量

式(1)表示对于任意时刻 φ 新增的任务 t，选择是否加入 φ 时刻存在的某个聚合任务集。

$$G_{tc}=\begin{cases}1, & \text{任务 } t \text{ 加入聚合任务集 } c\\ 0, & \text{否则}\end{cases},t\in T_n(\varphi),c\in C(\varphi) \tag{1}$$

式(2)表示对于任意时刻 φ 新增的任务 t，是否以任务 t 的位置为中心点新建聚合任务集。

$$NC_t=\begin{cases}1, & \text{新建聚合任务集}\\ 0, & \text{否则}\end{cases},t\in T_n(\varphi) \tag{2}$$

(2)目标函数

式(3)表示在对任务进行聚合操作时，将任务分配到最近的聚合任务集中。

$$Min_a=d_s(i)(j)G_{tc},t\in T_n(\varphi),c\in C(\varphi),i=l_t(t),j=l_c(c) \tag{3}$$

(3)约束条件

式(4)表示任务聚合过程中，任务到所分配的聚合任务集之间的距离小于 D。

$$d_s(i)(j)G_{tc}<D,t\in T_n(\varphi),c\in C(\varphi),i=l_t(t),j=l_c(c) \tag{4}$$

3.3.3　聚合任务集全局调度

下面给出聚合任务集全局调度的决策变量、目标函数和约束条件。

(1)决策变量

式(5)表示在时刻 τ 对还未调度的聚合任务集，选择加入某一工作者路径中的某一位置。

$$X_{kij}=\begin{cases}1, & \text{工作者 } k \text{ 从点 } i \text{ 到点 } j\\ 0, & \text{否则}\end{cases},k\in K(\tau),\ i\in L_{cw}(\tau),j\in L_{cw}(\tau) \tag{5}$$

(2)目标函数

式(6)表示在时刻 τ 将所有还未调度的聚合任务集加入合适工作者路径中的合适位置之后，工作者的聚合任务集总路径最短。

$$Min_x=\sum_{k\in K(\tau)}\sum_{i\in L_c(\tau)\cup L_{ks}(\tau)}\sum_{j\in L_c(\tau)\cup L_{ke}(\tau)}d_g(k)(i)(j)X_{kij} \tag{6}$$

(3)约束条件

式(7)表示所有的工作者最终会回到对应工作者的终点位置。

$$\sum_{i\in L_c(\tau)\cup L_{ks}(\tau)}X_{kie}=1,k\in K(\tau),e=l_{ke}(\tau)(k) \tag{7}$$

式(8)和式(9)确保每个聚合任务集都只能被一个工作者处理且只能被处理一次。

$$\sum_{k\in K(\tau)}\sum_{j\in L_c(\tau)\cup L_{ke}(\tau)}X_{kij}=1,i\in L_c(\tau)\cup L_{ks}(\tau) \tag{8}$$

$$\sum_{k\in K(\tau)}\sum_{i\in L_c(\tau)\cup L_{ks}(\tau)}X_{kij}=1,j\in L_c(\tau)\cup L_{ke}(\tau) \tag{9}$$

3.3.4　聚合任务集内局部调度

下面给出聚合任务集内局部调度的决策变量、目标函数和约束条件。

(1)决策变量

式(10)表示在时刻 φ 对工作者 k 需要局部调度的任务，选择加入工作者 k 路径中的某一位置。

$$Y_{ij}=\begin{cases}1, & \text{工作者 } k \text{ 从点 } i \text{ 到点 } j\\ 0, & \text{否则}\end{cases},i\in L_{tw}(\varphi)(k),\ j\in L_{tw}(\varphi)(k) \tag{10}$$

(2)目标函数

式(11)表示在时刻 φ 将需要局部调度的聚合任务集内的任务选择加入工作者 k 路径中的合适位置之后，工作者 k 在聚合任务集内的任务总路径最短。

$$Min_y=\sum_{i\in L_{tw}(\varphi)(k)\cup l_{ks}(\varphi)(k)}\sum_{j\in L_{tw}(\varphi)(k)\cup l_{kn}(\varphi)(k)}d_g(i)(j)Y_{ij} \tag{11}$$

(3)约束条件

式(12)确保参与局部调度的工作者 k 在完成对应任务集内的所有任务后会到达下一聚合任务集或者对应工作者的终点位置。

$$\sum_{i\in L_{tw}(\varphi)(k)}Y_{in}=1,n=l_{kn}(\varphi)(k) \tag{12}$$

式(13)和式(14)确保每个任务集内的任务都能且只能被处理一次。

$$\sum_{j\in L_{tw}(\varphi)(k)\cup l_{kn}(\varphi)(k)}Y_{ij}=1,i\in L_{tw}(\varphi)(k)\cup l_{ks}(\varphi)(k) \tag{13}$$

$$\sum_{i\in L_{tw}(\varphi)(k)\cup l_{ks}(\varphi)(k)}Y_{ij}=1,j\in L_{tw}(\varphi)(k)\cup l_{kn}(\varphi)(k) \tag{14}$$

下面将根据调度方法模型对调度方法的求解设计进行详细描述。

4　调度求解设计

4.1　任务聚合算法

由于路径规划问题属于 NP 难问题，将每一个新增任务都单独加入全局进行路径规划计算会极大地增加计算时间，因此对相似位置的任务进行聚合打包处理可减少计算时间，提高平台运行效率。本文使用空间距离算法对动态任务进行聚合处理，其执行逻辑如算法 1 所示。

算法 1　任务聚合算法

输入：$T_n(\varphi)$

输出：$C(\varphi)$

```
1. foreach newTask∈Tn(φ) do
2.   foreach cluster∈C(φ) do
3.     ds(lt(newTask))(lc(targetCluster))←getStraightDistance(lt(newTask),lc(cluster))
4.     if ds(lt(newTask))(lc(targetCluster))<minDistance
5.     then minDistance←ds(lt(newTask))(lc(targetCluster))
6.       targetCluster←cluster
7.     end
8.   end
9.   if minDistance<D then
10.    addNewTaskToTargetCluster(newTask,targetCluster)
11.  end
12.  else
13.  clusters.add(newCluster←createNewCluster(newTask))
14.  addNewTaskToTargetCluster(newTask,targetCluster)
15. end
16. end
```

首先计算动态新增任务 *newTask* 到所有聚合任务集的直线距离，选择距离最短的一个聚合任务集 *targetCluster*，判断距离 $d_s(l_t(newTask))(l_c(targetCluster))$ 是否小于某一固定阈值 D，如果小于，则将该任务加入对应聚合任务集，否则根据该动态任务新建聚合任务集 *newCluster*，并将该动态

任务加入新增聚合任务集中。假设新增任务集合$T_n(\varphi)$中的任务数量为t,聚合任务集集合$C(\varphi)$中的聚合任务集数量为c,则时间复杂度为$O(tc)$。

4.2 聚合任务集全局迭代调度算法

本文使用聚合任务集全局迭代调度算法对任务聚合算法中生成的多个聚合任务集进行分配和路径规划,其执行逻辑如算法2所示。

算法2 聚合任务集全局迭代调度算法

输入:$C_w(\tau)$

输出:X

```
1. Repeat
2.    initGene←greedy(C_w(τ))
3.    genes.add(initGene)
4.    repeat
5.       chosenGenes←choose(genes)
6.       crossedGenes←cross()
7.       genes.add(crossedGenes)
8.       mutatedGenes←mutation()
9.       genes.add(mutatedGenes)
10.      tempOptimalGene←getTempOptimalGene(genes)
11.      until the stop condition is reached
12.      optimalGene←getOptimalGene(tempOptimalGene)
13. until the stop condition is reached
14. X←getNewDecision(optimalGene)
15. updateWorkersPath(X)
```

聚合任务集全局迭代调度算法首先通过重启迭代法来防止算法陷入局部最优,假设重启迭代次数为r,最差情况下的时间复杂度为$O(r)$。每次重启都会调用贪心插入算法产生较优的初始解,依次将每个待调度聚合任务集尝试插入每一个工作者的每一个路径的两点之间,求得新的路径相比原始路径增加的距离,选择新增距离最短的插入点插入,以加快后续遗传算法的求解速度。假设待调度聚合任务集集合$C_w(\tau)$的聚合任务集数为c_w,已调度聚合任务集集合$C(\tau)$中的聚合任务集数量为c,工作者集合$K(\tau)$中的工作者数量为k,则使用贪心插入算法进行初始化的时间复杂度为$O(c_w(c+k))$。然后通过遗传算法对种群中的染色体进行选择、交叉、变异操作,将产生的新染色体与种群合并形成新的种群。当遗传算法迭代次数达到指定次数,或者适应值不变的次数达到指定次数,则停止遗传算法的迭代。假设遗传迭代次数为g,种群大小为p,则最差情况下的时间复杂度为$O(gp)$。当达到重启迭代次数,或者最优适应值不变的次数达到指定次数时,则停止重启迭代。通过两类迭代最后获得新的聚合任务集调度优化结果,两次迭代的总时间复杂度为$O(rgp)$。

本文针对众包平台中形成的聚合任务集,使用改进的遗传算法进行求解,具体改进点包括对遗传编码进行改进和对遗传操作进行改进。

(1)对遗传编码进行改进。改进的遗传编码标记了每个工作者对应的聚合任务集路径,并且标记了聚合任务集是否为未确认待分配状态,防止交叉变异操作中将工作者已分配安排的聚合任务集分配到其他工作者路径中。遗传编码样例如图5所示,其中标记了每个工作者的聚合任务集路径,蓝色表示在之前的调度中已经安排分配的聚合任务集,黑色表示本次调度中需要安排的聚合任务集。图5表示3个工作者的聚合任务集路径,其中工作者1的聚合任务集路径为1-2-8-3,工作者2的聚合任务集路径为4-9-10-5,工作者3的聚合任务集路径为6-7。序号1—7为已安排的聚合任务集。序号8—10为暂定安排的聚合任务集,一般是由贪心算法操作,或通过后续遗传算法操作为等待调度的聚合任务集安排暂定位置。

图5 遗传编码(电子版为彩色)

Fig. 5 Genetic code

(2)对遗传操作进行改进。为了求得更优的规划路径结果,同时防止已确定分配给工作者的聚合任务集被分配给其他工作者,本文对遗传算法中的交叉操作和变异操作进行了改进。

交叉操作改进如下:

Step1 选择种群中前$2n$个较优的染色体,并两两配对。

Step2 随机选择一个工作者,并交换配对染色体中对应工作者的聚合任务集路径,若两条染色体中该工作者聚合任务集路径一致,则随机选择另一个工作者来进行交叉操作。

Step3 对于交叉得到的两条子代染色体,可能会存在聚合任务集重复或者聚合任务集缺失的情况。对于重复存在的聚合任务集,保留新交换工作者内的聚合任务集,删除原始其他工作者内的聚合任务集。对于缺失的聚合任务集,将缺失聚合任务集加入除新交换工作者外的其他工作者路径中。

变异操作改进如下:

Step1 对于一条染色体,随机选择两个工作者,选择第一个工作者的路径中待定安排的聚合任务集,将其插入到第二个工作者的聚合任务集路径上的随机位置,如果工作者1的路径中不存在待安排的聚合任务集,则随机选择另一个工作者进行变异;

Step2 随机选择一个工作者,对其路径上的任意两个聚合任务集交换位置,获得一条新的子染色体;

Step3 重复Step1,直到达到指定n次,获取n条变异后的子染色体,计算适应值,选取其中最优的变异加入种群中。

4.3 聚合任务集内局部路径优化算法

对于聚合任务集内部的任务,通过局部路径优化的方式确定工作者任务执行路径,其执行逻辑如算法3所示。

算法3 聚合任务集内局部路径优化算法

输入:$T_w(\varphi)(k)$

输出:Y

```
1. initGene←greedy(T_w(φ)(k))
2. genes.add(initGene)
3. repeat
4.    chosenGenes←choose(genes)
5.    mutatedGenes←mutation()
6.    genes.add(mutatedGenes)
```

7.　optimalGene←getOptimalGene(genes)
8. until the stop condition is reached
9. Y←getNewDecision(optimalGene)
10. updateWorkersPath(Y)

局部路径优化同样基于遗传算法进行求解,分别将聚合任务集内每个未调度的任务通过贪心算法插入到工作者路径中,假设待调度任务集合$T_w(\varphi)(k)$的任务数为t_w,对应聚合任务集内已调度任务数为t,则使用贪心插入算法进行初始化的时间复杂度为$O(t_w t)$。首次调度时刻,工作者在该聚合任务集内的任务路径不存在,因此设置初始路径,起点为工作者当前位置,终点为下一个聚合任务集所在位置或者工作者的终点。然后通过遗传算法对种群中的染色体进行选择、变异操作,将产生的新染色体与种群合并形成新的种群。当达到遗传算法迭代次数,或者暂时最优适应值不变的次数达到指定次数,则停止遗传算法的迭代,最后获得新的聚合任务集路径优化结果。假设遗传迭代次数为g,种群大小为p,则最差情况下的时间复杂度为$O(gp)$。

聚合任务集内局部路径优化算法与聚合任务集全局路径优化算法有以下不同之处:

(1)遗传编码不同。聚合任务集内局部路径优化仅需对单工作者的单聚合任务集内的任务进行路径优化,不需要考虑其他工作者,也不需要考虑聚合任务集内的任务和同一工作者的其他聚合任务集的执行顺序。因此遗传编码可以表示为$[c_1,\cdots,c_n]$的形式,例如[1,3,5]表示工作者接下来对该任务集的路径为:1-3-5。

(2)聚合任务集内局部路径优化算法不用交叉操作。由于染色体编码较为简单,为加快求解速度,因此取消交叉操作。

(3)变异操作不同。聚合任务集内局部路径优化算法仅通过随机选择计算路径中的两个点,交换对应位置上的任务来实现变异操作。

5　调度模拟平台

本文结合了空间众包中任务和工作者不确定性的特点,设计并实现了基于真实导航的空间众包任务路径动态调度模拟平台,如图6所示。空间众包任务路径动态调度模拟平台包括3个模块,分别为模拟数据管理模块、模拟调度管理模块和模拟结果管理模块。

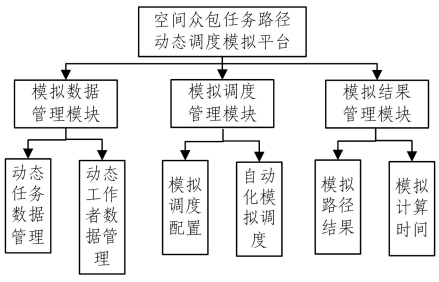


图6　模拟平台的设计

Fig. 6　Design of simulation platform

5.1　模拟数据管理模块

本文所实现的空间众包任务路径动态调度模拟平台可对模拟数据集进行管理,用于后续的模拟调度。数据集包括动态任务数据集、动态工作者数据集。一个动态任务数据集包括一个或多个动态新增任务,任务属性包括任务所在经纬度以及任务创建时间。一个动态工作者数据集包括一个或多个动态工作者数据,工作者属性包括工作者起始经纬度、终点经纬度、交通方式、行驶速度以及服务上线时间。

5.2　模拟调度管理模块

在开始模拟之前,可对模拟调度的参数进行配置,可以选择动态任务数据集、动态工作者数据集和此次模拟调度用到的调度算法。

创建成功后可选择是否前往模拟调度执行界面,如图7所示,对于之前已创建的未完成的历史模拟调度,进入模拟执行界面时,会自动加载显示对应历史模拟调度的执行时刻,并绘制对应时刻的任务、工作者和路径信息。通过点击"自动执行"或"执行一次"按钮来控制模拟调度是逐步执行还是自动执行。

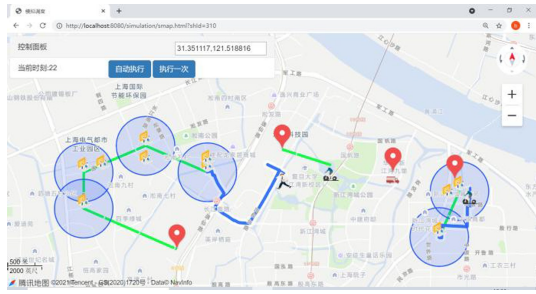


图7　模拟调度

Fig. 7　Simulation scheduling

5.3　模拟结果管理模块

模拟结果管理模块可以记录工作者的行驶轨迹,即每一调度时刻工作者所在位置。平台会获取并记录模拟调度中每次算法计算的时间,为后续模拟结果统计做准备。结果统计功能会根据记录数据统计工作者的总路程和算法计算的总时间。

6　实验研究

本节通过实验探索不同聚合任务集范围和问题规模对算法的计算时间和结果的影响,并分别对本文提出的聚合调度和迭代调度策略算法的有效性进行验证。

6.1　问题描述

针对本文所提出的空间众包任务路径动态调度方法,存在以下4个研究问题需要通过实验来进行分析验证。

(1)空间众包任务路径动态调度方法中不同聚合任务集半径长度D是否会对调度的计算时间和结果产生影响?

(2)空间众包中不同数量的动态任务对调度的计算时间和结果会产生什么影响?

(3)与传统非聚合调度相比,本文提出聚合的调度策略在节省计算时间以及在求解优化目标结果方面效果如何?

(4)本文提出的迭代调度算法相比传统的先进行任务分配再进行路径规划的调度算法,在求解优化目标结果上是否有所改进?

6.2 实验设置

6.2.1 实验环境

本实验将在空间众包任务路径动态调度模拟平台中模拟进行，以对聚合调度策略算法的有效性进行验证。模拟平台的开发语言为 java。模拟平台运行环境为笔记本电脑，操作系统为 Windows10，处理器为 Intel(R)Core(TM)i5-6300HQ CPU，机带 RAM 为 12.0GB。

6.2.2 实验数据

本文通过真实数据集和合成数据集来进行实验。真实数据集来自基于位置的社交网络 Gowalla[19]，该数据已被多个空间众包研究所使用[20-21]。由于本文基于腾讯地图进行路径调度，腾讯地图目前只支持中国地区的路径求解和导航，因此本文选择上海作为研究边界区域(东经 120°52′～122°12′，北纬 30°40′～31°53′之间)。Gowalla 数据集包含 384 条上海的人员签到数据，签到数据包含经纬度位置信息和签到时间。本文选取其中 300 条作为动态任务数据，选取其中 60 条作为动态工作者数据，并通过抽取不同数量的任务形成不同数据集 g60-50，g60-100，g60-200 和 g60-300，这 4 个数据集的工作者数量均为 60，任务数量分别为 50，100，200 和 300。

针对分区域或小范围内的任务分配调度的空间众包问题，如食品配送等，本文通过空间众包任务路径动态调度模拟平台提供的随机生成数据集工具，分别生成了多组合成数据集，数据集生成方案如表 2 所列，每组数据集都包括一定数量的动态工作者和动态任务，模拟为逐次进行，因此用整数表示每一模拟时刻。其中动态工作者的上线时间为创建时间范围内的随机时刻，工作者的起点和终点在对应经纬度范围内随机取点，起点和终点不一定相同。其中动态任务的创建时间为创建时间范围内的随机时刻，任务所在位置在对应经纬度范围内随机取点。

表 2 合成数据集生成方案

Table 2 Data set generation

数据集	类别	总数	经度范围	纬度范围	时间范围
s10-50	工作者	10	31°27′～31°33′	121°42′～121°52′	0～10
	任务	50	31°27′～31°33′	121°42′～121°52′	0～240
s10-100	工作者	10	31°27′～31°33′	121°42′～121°52′	0～10
	任务	100	31°27′～31°33′	121°42′～121°52′	0～240
s20-200	工作者	20	31°27′～31°33′	121°42′～121°52′	0～10
	任务	200	31°27′～31°33′	121°42′～121°52′	0～240
s20-400	工作者	20	31°27′～31°33′	121°42′～121°52′	0～10
	任务	400	31°27′～31°33′	121°42′～121°52′	0～240

6.2.3 对照算法

本文通过以下 4 种算法对聚合调度和迭代调度策略算法的有效性进行验证。

(1)贪心算法(GI)：本文使用已有空间任务分配调度研究[4]中所用的贪心插入法作为基础对照算法，周期性地对动态任务进行调度。

(2)贪心遗传算法(GIGA)：基于文献[5-8]所提出的策略，本文在贪心插入法的基础上加入遗传算法，即贪心遗传算法，周期性地对动态任务进行调度。

(3)聚合调度算法(CGA)：在贪心遗传算法的基础上加入本文所提出的聚合调度方法，即对动态新增任务先进行聚合，再将聚合任务集通过周期性全局调度分配给合适的工作者并规划执行路径，最后在工作者进入聚合任务集范围内时再对任务集内的任务进行局部调度。

(4)聚合迭代调度算法(CIGA)：在聚合调度算法的基础上加入迭代调度，对动态新增任务迭代进行任务分配和路径规划操作。

6.3 实验方法

基于 Gowalla 真实数据集和合成数据集，根据不同聚合任务集半径长度 D，对比算法路径总长度和计算时间。在不同任务数量的情况下，使用 4 种算法进行模拟调度，记录 4 种算法的计算总时间并进行对比，同时对聚合调度算法的 3 个主要操作的计算时间进行分析。另外通过记录 4 种算法模拟调度中工作者的交通轨迹，来计算各策略路径总长度，对最短路径优化目标结果进行对比。

6.4 结果分析

表 3 列出了在数据集 g60-300 上，不同聚合任务集半径 D 对调度路径总长度和计算时间的影响。选取半径 D 分别为 250 m，500 m，1 000 m，2 000 m。

表 3 不同聚合任务集半径下路径长度和计算时间

Table 3 Total length and run time of varying D

	250 m	500 m	1 000 m	2 000 m
路径长度/km	225.98	240.55	248.01	270.8
计算时间/ms	834 734	615 438	756 193	1 207 898

由表 3 可以看出，随着聚合任务集半径长度的不断增加，所有工作者的路径总长度也不断增加，这是因为在较大的聚合任务集中靠外的新增任务被分给了对应聚合任务集的工作者，而不是分配给距离较近的其他工作者，在聚合任务集范围较小且工作者密度较大的情况下，可求解更优的目标优化结果，即任务执行总路径更短。另外还可以看出，随着聚合任务集半径长度的不断增加，计算时间先减少后增加，这是因为在算法时间复杂度较高的聚合任务集全局调度和聚合任务集内局部调度算法中，聚合任务集范围过小或者过大都会导致聚合任务集全局调度和聚合任务集内局部调度所需的时间增加。例如存在 200 个任务点，形成 10 个聚合任务集，每个聚合任务集下有 20 个任务，针对一个新增任务，仅需要计算该新增任务形成的聚合任务集在 10 个聚合任务集中的合适位置，或者计算该新增任务在已有聚合任务集内 20 个任务中的合适位置。聚合任务集范围过小的情况下可能会存在 200 个聚合任务集，每个任务集内仅包括 1 个任务，聚合任务集范围过大的情况下可能仅存在 1 个聚合任务集，任务集内包括 200 个任务。此时对于新增任务则需要计算任务在 200 个聚合任务集或任务中的合适位置，因此根据第 4 节中的算法复杂度分析，聚合任务集范围过大或者过小都会导致计算时间增加。在众包平台能够提前预测动态任务密度和动态工作者密度的情况下，可以根据对计算时间和路径总长度的多目标需求选择合适的聚合任务集范围，本文研究选择聚合任务集范围为 500。

表 4 列出了不同数据集下 4 种调度算法的路径总长度。表 5 列出了不同数据集下 4 种调度算法的计算时间。

表 4　不同数据集下的路径总长度

Table 4　Total length of path on different datasets

（单位：km）

数据集	GI	GIGA	CGA	CIGA
g60-50	88.07	87.85	89.54	89.66
g60-100	140.17	131.6	138.72	137.13
g60-200	194.23	193.3	196.47	188.29
g60-300	242.4	241.17	243.4	240.55
s10-50	122.62	117.6	119.32	112.19
s10-100	185.1	184.65	185.2	180.59
s20-200	301.44	297.9	300.64	297.24
s20-400	413.93	411.8	423.12	417.86

表 5　不同数据集下的计算总时间

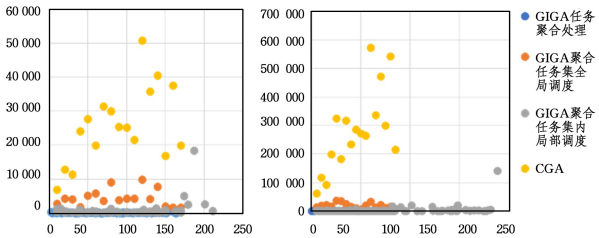
Table 5　Total time of calculation on different datasets

（单位：ms）

数据集	GI	GIGA	CGA	CIGA
g60-50	70978	78963	70071	79848
g60-100	457042	487738	139910	161695
g60-200	1590921	1801777	353196	361816
g60-300	3197107	4485426	602174	615438
s10-50	33869	35904	34934	50903
s10-100	101550	109157	96307	101747
s20-200	393474	434237	332505	351146
s20-400	2579701	2850384	1037603	1275974

由表 4 可以看出，随着任务数量的增加，4 种算法的路径总长度有不同程度的增加，即需要工作者到达更多的新任务所在位置，行驶更多路径。由表 5 可以看出，随着任务数量的增加，4 种算法的计算时间逐渐增加，这是因为更多任务加入任务分配调度，增加了算法需要求解的数据量。

对于两组数据集 g60-100 和 g60-300，在模拟调度过程中聚合调度算法和贪心遗传算法的计算时间如图 8 所示，其中横坐标为模拟调度时刻，纵坐标为算法计算时间（单位：ms）。



（a）g60-100　　（b）g60-300

图 8　单次计算时间

Fig. 8　Time of single calculation

图 8 给出了聚合调度与非聚合调度的单次求解效率对比。聚合调度算法通过 3 种操作来完成对动态任务的调度规划，分别为任务聚合处理、聚合任务集全局调度和聚合任务集内局部调度。贪心遗传算法不对任务进行聚合，直接对动态任务进行调度规划。聚合调度算法通过聚合任务集全局调度可以给工作者安排对应聚合任务集执行顺序，指导工作者直接前往对应聚合任务集所在区域；或者通过聚合任务集内局部调度来安排工作者在对应聚合任务集内的任务执行顺序，指导工作者前往对应任务所在位置，即先给出任务执行的大致方向，随后再规划任务具体执行路径。由图 8 可见，相比贪心遗传算法需要等待较长时间才能为工作者安排对应执行路径，聚合调度算法通过分阶段求解的方式，可在较短时间内求解路径规划结果并提供给工作者使用。另外，随着时间的推移和当前众包平台任务总量的逐渐增加，贪心遗传算法单次求解的时间快速增加，而聚合调度算法中单次求解时间增加缓慢。这是因为在聚合调度算法中，新增的动态任务可能会加入到已调度的聚合任务集中而不用再次调度，或者形成少量的待调度聚合任务集参与调度，因此增加的调度时间较少。相比之下，贪心遗传算法中的每一个新增任务都需要参与全局调度，而全局调度的任务数量可能会随着时间的推移不断增多，调度所需时间也快速增加。

表 5 对比了聚合调度算法和没有使用聚合调度方法的贪心遗传算法的总计算时间。可以看出，聚合调度方法花费的总时间远少于非聚合调度方法花费的时间。这是因为通过较短时间的任务聚合计算可以降低路径优化问题的规模和复杂性，对新增的动态任务进行聚合打包，可以将多个需要调度的动态任务合并成数量更少的聚合任务集来进行调度。另外，对于新增动态任务出现在已分配合适工作者的聚合任务集内的情况，聚合调度算法不用再对该聚合任务集进行全局分配调度。在工作者到达聚合任务集范围之前，不用对聚合任务集内的任务进行调度，以节省计算时间；当工作者进入聚合任务集内时进行聚合任务集内局部调度，不用再考虑将任务分配给哪一个工作者的问题，只需根据对应工作者的实时位置对聚合任务集内的任务进行路径优化。通过对比真实数据集和合成数据集的实验结果可以发现，小范围区域合成数据集的部分实验中聚合调度的执行效率提升不大，这是因为小范围区域内的聚合任务集随机生成，比较分散，并且任务容易在新增任务出现前就被处理完成，从而降低了聚合的效果。从表 4 还可以看出，聚合调度策略的路径总长度比非聚合调度略短，即在最短路径优化目标结果上聚合调度结果略差于非聚合调度，具体原因为聚合调度在路径规划过程中先给工作者安排的是聚合任务集路径，工作者在前往聚合任务集时是以聚合任务集初始任务的位置为目标点，但是当工作者到达聚合任务集范围内时会在聚合任务集内进行局部路径优化，此时工作者的目标点可能会发生改变。非聚合调度在任务调度结束后就明确了工作者的具体任务路径，因此避免了部分弯路。

从表 4 可以看出，迭代调度方法可以在一定程度上缩短工作者的最短总路径，避免了非迭代调度中先分配的任务没有考虑后续需要分配的任务的影响而降低调度准确性的问题。但是与非迭代调度相比，迭代调度在计算效率上需要花费更多的时间。对于拥有较强计算服务器并且追求更优最短路径的众包平台，可选择使用迭代调度方法来求解更优的目标结果。

结束语　近年来空间众包得到了越来越广泛的应用，国内外研究者对空间众包的各个方向进行了深入研究，其中对于空间众包任务分配和任务调度问题的研究取得了丰硕的成果，但是由于相关研究缺少对空间众包中任务和工作者的动态不确定性的考虑，空间众包任务路径动态调度问题仍需要进一步研究。

本文针对空间众包任务路径动态调度问题开展了以下研究：首先针对空间众包中工作者上线时间、数量、位置和交通方式的不确定性以及请求者发布任务的时间、数量和位置的不确定性等问题，提出了一种空间众包任务路径动态调度方法来对空间众包任务进行调度，通过聚合调度策略方法，对动态新增任务先聚合再进行任务分配和路径规划。通过实验验证，聚合调度方法相比已有相关研究工作中用到的非聚合调度可节省大量的计算时间。另外，传统方法对动态新增任务先进行任务分配，将任务分配给合适的工作者，再对工作者的

所有任务进行路径优化。但是先分配的任务没有考虑后续需要分配的任务的影响，降低了部分最优结果的准确性。本文通过在聚合任务集全局调度过程中使用迭代调度的算法，将任务分配和路径优化操作迭代进行，来对动态任务进行调度，实验验证，相比已有相关研究工作中用到的非迭代调度，迭代调度算法可提高获取最优结果的准确性。最后，本文结合空间众包任务路径调度过程中存在多种不确定性的特点，设计并实现了基于真实导航的空间众包任务路径动态调度模拟平台，相比传统模拟平台，本文所实现的模拟平台简单、方便、易用，且可扩展性强。

在未来的工作中，将对动态任务和工作者进行预测，根据预测结果辅助调度策略和参数的选定，再综合考虑更多空间众包的工作者异常等不确定性，对该方法进行进一步改进，使其灵活适应多种优化目标和约束条件，此外将尝试进一步提高模拟平台的仿真度和运行效率。

参 考 文 献

[1] FAN Z J, SHEN L W, PENG X, et al. Multi stage task allocation on constrained spatial crowdsourcing[J]. Chinese Journal of Computers, 2019, 42(12): 2722-2741.

[2] WANG K, WANG Z J. Crowdsourcing Collaboration Process Recovery Method[J]. Computer Science, 2020, 47(10): 19-25.

[3] GUMMIDI S R B, XIE X, PEDERSEN T B. A survey of spatial crowdsourcing[J]. ACM Transactions on Database Systems, 2019, 44(2): 1-46.

[4] SUN D, XU K, CHENG H, etal. Online delivery route recommendation in spatial crowdsourcing[J]. World Wide Web, 2019, 22(5): 2083-2104.

[5] TAO Q, ZENG Y, ZHOU Z, et al. Multi-worker-aware task planning in real-time spatial crowdsourcing[C]// International Conference on Database Systems for Advanced Applications. Springer, 2018: 301-317.

[6] LIU G R. Study on express route optimization problem with simultaneous delivery and pickup under dynamic demands[D]. Chongqing: Chongqing University, 2018.

[7] LIU M, SHEN Y, SHI Y. A hybrid brain storm optimization algorithm for dynamic vehicle routing problem[C]//International Conference on Swarm Intelligence. Springer, 2020: 251-258.

[8] LU X, TANG K, MENZEL S, et al. A competitive co-evolutionary optimizationmethod for the dynamic vehicle routing problem[C]// Symposium Series on Computational Intelligence. IEEE, 2020: 305-312.

[9] SBAI I, KRICHEN S. An adaptive genetic algorithm for dynamic vehicle routingproblem with backhaul and two-dimensional loa-ding constraints[C]// International Multi-Conference on Organization of Knowledge and Advanced Technologies. IEEE, 2020: 1-7.

[10] TONG Y, ZHOU Z, ZENG Y, et al. Spatial crowdsourcing: a survey[J]. The VLDB Journal, 2020, 29(1): 217-250.

[11] DENG D, SHAHABI C, DEMIRYUREK U. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing [C]// 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2013: 324-333.

[12] COSTA C F, NASCIMENTO M A. In-route task selection in crowdsourcing[C]// 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2018: 524-527.

[13] ZHANG J, YANG F, WENG X. An evolutionary scatter search particle swarm optimization algorithm for the vehicle routing problem with time windows[J]. IEEE Access, 2018, 6: 63468-63485.

[14] YU D H, CHENG T, YUAN X. Software Crowdsourcing Task Recommendation Algorithm Based on Learning to Rank[J]. Computer Science, 2020, 47(12): 106-113.

[15] LI Y, YIU M L, XU W. Oriented online route recommendation for spatial crowd sourcing task workers[C]// International Symposium on Spatial and Temporal Databases. Springer, 2015: 137-156.

[16] COSLOVICH L, PESENTI R, UKOVICH W. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem[J]. European Journal of Operational Research, 2006, 175(3): 1605-1615.

[17] ASGHARI M, SHAHABI C. On on-line task assignment in spatial crowdsourcing[C]// International Conference on Big Data (Big Data). IEEE, 2017: 395-404.

[18] HU Z H, SHEU J B, ZHAO L, et al. A dynamic closed-loop vehicle routingproblem with uncertainty and incompatible goods [J]. Transportation Research Part C: Emerging Technologies, 2015, 55: 273-297.

[19] MU Q, FU Z, LYSGAARD J, et al. Disruption management of the vehicle routingproblem with vehicle breakdown[J]. Journal of the Operational Research Society, 2011, 62(4): 742-749.

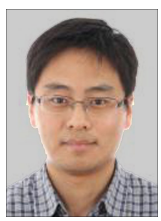
[20] GÜNER A R, MURAT A, CHINNAM R B. Dynamic routing under recurrentand non-recurrent congestion using real-time its information[J]. Computers & Operations Research, 2012, 39(2): 358-373.

[21] CHO E, MYERS S A, LESKOVEC J. Friendship and mobility: user movement inlocation-based social networks[C]// 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2011: 1082-1090.

[22] SUN Y, WANG J, TAN W. Dynamic worker-and-task assignment on uncertain spatial crowdsourcing[C]// 22nd International Conference on Computer Supported Cooperative Work in Design. IEEE, 2018: 755-760.

[23] FELDMAN J, MEHTA A, MIRROKNI V, et al. Online stochastic matching: Beating 1-1/e[C]// 50th Symposium on Foundations of Computer Science. IEEE, 2009: 117-126.

SHEN Biao, born in 1996, postgraduate. His main research interests include the fusion of the ternary human-machine-thing and mobile cloud computing software system.



SHEN Li-wei, born in 1982, Ph.D, associate professor. His main research interests include the fusion of the ternary human-machine-thing, mobile application development and analysis, etc.

(责任编辑：李亚辉)