|  |  |
| --- | --- |
| **分类号————————————————————————密级————————————————————————————**  **UDC \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |
|  |
|  |

**本 科 毕 业 论 文**

运筹学在选址问题中的应用研究

**学生姓名 张君秋 学号 15160021045**

**指导教师 肖 汉**

**院、系、中心 数学科学学院**

**专业年级 2015级信息与计算科学**

**论文答辩日期 2019 年 5 月 30 日**

**中 国 海 洋 大 学**

运筹学在选址问题中的应用研究

完成日期：\_\_\_\_\_\_\_\_\_\_\_\_

指导教师签字：\_\_\_\_\_\_\_\_\_\_\_\_

答辩小组成员签字：\_\_\_\_\_\_\_\_\_\_\_\_

运筹学在选址问题中的应用研究

摘 要

设施选址问题是一类复杂的运筹学问题，选址问题的研究具有重要的理论价值和现实意义。本文在何永梅对软容量限制的设施选址问题中的竞争决策算法研究的基础上，对容量限制下的设施选址问题的数学性质进行了归纳总结，并将竞争决策算法推广到强容量限制的设施选址问题中，给出了强容量限制的设施选址问题中的竞争决策算法，并通过大量算例验证了算法的正确性和有效性。此外，本文还在竞争决策算法的基础上引入混合策略的思想，提出了一种基于混合策略的竞争决策算法，即在竞争决策算法满足竞争终止条件后按照一定的混合策略再分配一次顾客，通过比较混合策略解的期望与竞争决策算法解的优劣，选取较优的解作为最终解，并通过大量算例验证了算法的正确性和有效性。

关键词：容量限制的设施选址问题；竞争决策算法；混合策略

Application of Operational Research in Facility Location

Abstract

Facility location is a classic operational research problem. The research of facility location problem has theoretical importance and practical significance. Based on He Yongmei's research on the competitive decision algorithm (CDA) in the soft-capacitated facility location problem, this paper summarizes the mathematical properties of capacitated facility location problem, and extends the CDA to the hard-capacitated facility location problem by giving the CDA procedure in the hard-capacitated facility location Problem. The correctness and effectiveness of the algorithm is verified by a large number of examples. In addition, mixed strategy was introduced to CDA in this paper. After the algorithm meets the competition termination condition, we reassign the customer according to a certain mixed strategy, compare the expectation of the mixed strategy solution with the solution of CDA, and return the better solution as the final solution. The correctness and effectiveness of the algorithm is also verified by a large number of examples.

**Key Words**: capacitated facility location problem; competitive decision algorithm; mixed strategy

目录

1 绪论 1

2 选址问题 2

2.1 中值选址问题 3

2.2 覆盖选址问题 5

2.3 中心选址问题 7

2.4 其他选址问题 8

2.5 主要研究问题和研究方法 8

2.5.1 容量限制的设施选址问题 8

2.5.2 竞争决策算法简介及基本概念 10

3 软容量限制的设施选址问题 12

3.1 模型描述 12

3.2 数学性质 13

3.3 竞争决策算法 14

3.3.1 竞争决策算法概念 15

3.3.2 竞争决策算法流程 16

3.3.3 算例 17

3.4 基于混合策略的竞争决策算法 20

3.4.1 算例 21

4 强容量限制的设施选址问题 25

4.1 模型描述 25

4.2 数学性质 26

4.3 竞争决策算法 26

4.3.1 竞争决策算法概念 26

4.3.2 竞争决策算法流程 27

4.3.3 算例 28

4.4 基于混合策略的竞争决策算法 31

4.4.1 算例 31

5 总结与讨论 35

参考文献 37

# 

# 1 绪论

运筹学（operational research）是应用数学的一个分支，主要是运用统计学、数学模型和算法等知识，研究实际生活中的复杂问题，得到复杂问题的最优解或近似最优解，进而得到实际生活中复杂问题的最优解决方案或近似最优解决方案。所以，运筹学也称为广义最优化理论。

运筹学的萌芽是在18世纪到20世纪初。1736年，Euler运用图论的思想对哥尼斯堡七桥问题进行了研究，成为图论研究的起源；1738年，Bernoulli提出了效用这一概念，为决策论的研究奠定了基础；1777年，Buffon提出了利用随机投针法来计算的方法，这是蒙特卡洛法的最早应用；1896年，Pareto提出了Pareto最优的概念，为多目标优化问题的研究奠定了基础；1909年，Erlang运用概率论相关知识研究了电话服务问题，成为排队论研究的起源；1912年，Zermelo首次研究了博弈论问题；1915年，Harris对商业中的库存问题的研究标志着库存论研究的起源；1916年，Borel提出了博弈论中的最优策略的概念，证明了一些博弈论问题存在最优策略；1928年，Von Nuemann深入研究了博弈论问题；1932年，Weibull研究了维修问题，成为可靠性数学理论的起源；1939年，Kantorovich提出了线性规划模型，成为线性规划理论研究的起源。

运筹学的起源是在第二次世界大战期间。二战期间，英美等国家成立了军事数学小组，研究军事中的数学问题，并称这些研究为“operational research”，也就是“运筹学”。二战之后，这些研究逐渐发表，并广泛应用于军事之外的其他领域。

运筹学的发展是在20世纪40年代到60年代。1947年，Dantzig研究了线性规划问题的一种解法——单纯形法，在1949年，线性规划理论建立，标志着运筹学大发展的到来。之后，1951年，非线性规划理论建立；1954年，网络流理论建立；1955年，随机规划理论建立；1958年，整数规划理论建立。同时，排队论、存储论、决策论和博弈论也都得到了巨大的发展。这期间，计算机的发明大大促进了运筹学的发展，使得运筹学中许多复杂的模型得以求解；同时，运筹学的发展也反过来促进了计算机的发展，使得计算机向更大规模更快速度的方向快速发展。如今，运筹学已经被广泛应用在生产、生活、经济、管理、政治和军事的各个领域中，创造了巨大的社会财富。

运筹学中一个很重要的分支就是博弈论。博弈论起源于20世纪40年代，但是博弈论思想的起源却要久远的多，2000多年前我国的田忌赛马的故事中就应用了博弈论的思想；1838年Cournot提出的寡头垄断模型和1883年Bertrand提出的Bertrand模型中就有了博弈论的雏形；1928年Von Nuemann提出了非合作二人零和博弈的最大最小定理，这是博弈论发展历史上的一个里程碑；1944年Von Nuemann与Morgenstern共同写作出版了《博弈论和经济行为》一书，这标志着博弈论的正式建立；1950年Nash在博士论文中给出了纳什均衡的概念和了均衡点存在定理，为博弈论的更广泛应用打下了基础。之后，博弈论的思想被应用到很多启发式算法中，例如计算机领域的强化学习算法。

# 2 选址问题

在生产、生活、经济、管理中我们遇到的这类问题：在哪里建造工厂可以更好的降低成本？在哪里建造超市可以获得更多的利润？在哪里建造仓库可以提高物流的效率？在哪里布设路由器可以提升网络服务的速度？归纳起来，这类问题研究的是建造一定数量的设施（这些设施一般有固定的建设费用）来服务一定数量的顾客（有固定的服务费用），使得成本最低或利润最大的问题。这类问题在运筹学中统称为设施选址问题。

因为设施选址决策受到设施的类型和数量、顾客的类型和数量、资金限制、政策限制等因素的影响；并且，设施选址决策会影响设施为顾客服务的成本、效率和质量等方面；同时，设施选址决策的影响是深远的，所以设施选址决策是重要的长期决策，研究选址问题具有重要的理论价值和现实意义。

1909年，Weber研究了在平面上选取单个设施的位置使得设施与多个顾客的总距离最小的问题，这被认为是选址问题研究的开始。1964年，Hakimi首次在论文中研究了p - 中值问题和p - 中心问题，这篇论文大大推动了选址问题研究。自此之后，有关于选址问题研究的论文的数量急剧增多，选址问题研究也开始活跃了起来。

根据目标的不同，选址问题大致可以分为中值问题、覆盖问题和中心问题三种经典问题，下面对这三种问题分别进行了介绍。

## 2.1 p - 中值选址问题

p - 中值选址问题研究选取 p 个设施的位置来服务若干个顾客，使得总的成本最小的问题。p - 中值问题的目标可以是使顾客到最近的设施的总距离最短，可以是总服务时间最少，也可以是总服务费用最少。

p - 中值问题由Hakimi于1964年提出，可以分为绝对p - 中值问题和顶点p - 中值问题，绝对p - 中值问题是指可以选取网络上的任意位置作为设施的选址，而顶点p - 中值问题则是指只能选取顶点作为设施的选址。绝对p - 中值问题与顶点p - 中值问题的有相同的最优解，区别就仅仅在于约束条件的不同，所以一般不区分这两种问题。在之后的研究中，Garey证明了p - 中值问题是NP-hard问题；Hakimi和Kariv证明了p - 中值问题是NP-C问题；Goldman给出了一种适用于树和单环网络的1 - 中值问题的简单算法；Wesolowsky和Drezner研究了动态p - 中值问题；Chen和Drezner研究了条件p - 中值问题；Li研究了可靠p - 中值问题；Diabat研究了封闭环中的中值问题；Revelle和Swain建立了p - 中值问题的整数规划模型。

记顾客集合为，候选设施集合，任意顾客和候选设施之间都有一定的服务费用，目标就是在候选设施集合中选取部分设施，使得每个顾客都有设施服务，并且尽量使总成本最小，则p - 中值问题的模型可以描述为：

（1）

（2）

（3）

（4）

（5）

其中，为候选设施的序号，为顾客的序号，为候选设施是否为顾客服务，若候选设施为顾客服务，则为1，否则为0，为候选设施是否建设，若候选设施建设，则为1，否则为0。约束（2）式表示每个顾客有且只有一个设施为其服务，约束（3）式表示只有建立设施才能为顾客服务，约束（4）式表示建立设施的最大数量不能超过P，约束（5）式是变量的取值范围约束。

p - 中值问题可以用整数规划的求解方法来求解，例如对偶的多阶段法、对偶定界法、Lagrange松弛法等等；同时，一些启发式算法也可以用来求解p - 中值问题，例如交换启发式算法、邻域搜索算法等等。

p - 中值问题常常应用在企业选址中，例如工厂、仓库的选址，因为企业选址问题常常以成本最低为目标，这一类追求经济效益最大化的目标通常被称为“经济效益性”目标。同时，p - 中值问题也可以应用于公共设施的选址中，例如学校、图书馆的选址。这一类追求公共服务效果最好的目标通常被称为“集体福利性”目标。p - 中值问题常常解决的就是这两类目标的问题。

## 2.2 覆盖选址问题

首先，我们引入覆盖的概念： 当某个顾客可以在规定的服务费用内被某个设施服务，则称该顾客被该设施覆盖。而覆盖选址问题研究的就是用最少成本的设施来最大程度的满足顾客需求的问题。根据目标的不同，覆盖问题又可以分为集合覆盖问题和最大覆盖问题两种。

集合覆盖问题最早由Toregas于1971年提出，其目标是建设若干设施在满足所有顾客需求的条件下使建设成本最小。记顾客集合为，候选设施集合，候选设施集合中的每个设施都有一定的建设费用，为顾客是否被设施覆盖，若顾客被覆盖，则为1，否则为0，目标就是在候选设施集合中选取部分设施，使得每个顾客都被覆盖，并且尽量使总成本最小，则模型可以描述为：

（6）

（7）

（8）

其中，为候选设施的序号，为顾客的序号。为候选设施是否建设，若候选设施建设，则为1，否则为0。约束（7）式表示每个顾客都应该满足至少有一个覆盖该顾客的设施是建设的，约束（8）式是变量的取值范围约束。

该模型假设了所有的顾客的需求都应被满足，但是实际上，如果某个顾客的需求量很小，那么为了满足这个顾客的需求而建设设施很可能导致入不敷出，所以就有了最大覆盖问题。

最大覆盖问题最早由Church和ReVelle于1988年提出，其目标是建设个设施使顾客被满足的需求量最大。同样地，记顾客集合为，候选设施集合，顾客集合中的每个顾客都有一定的需求量，为顾客是否被设施覆盖，若顾客被覆盖，则为1，否则为0。目标就是在候选设施集合中选取部分设施，尽量使被覆盖的顾客的需求量之和最大，其模型可以被描述为：

（9）

（10）

（11）

（12）

其中，为候选设施的序号，为顾客的序号，为顾客是否被设施覆盖，若顾客被覆盖，则为1，否则为0。为候选设施是否建设，若候选设施建设，则为1，否则为0。约束（10）式表示顾客被覆盖时应该满足至少有一个覆盖该顾客的设施是建设的，约束（11）式表示建立设施的最大数量不能超过，约束（12）式是变量的取值范围约束。

一般网络上的覆盖选址问题是NP-C问题。集合覆盖问题的求解方法主要有精确求解方法，例如分支定界法、基于动态次梯度的分支定界法等等；同时，集合覆盖问题还可以用一些启发式算法来求解，例如替换启发式方法、混合启发式算法等等。而最大覆盖问题的求解方法主要有分支定界法，除此之外就是贪婪添加算法、贪婪添加-替换算法等启发式算法。

覆盖选址问题常常应用在应急服务设施的选址上，例如消防站、急救中心等，因为应急服务设施通常满足在一定时间内或者一定距离内可以服务到顾客即可（即覆盖到顾客），并且通常希望一定的应急服务设施服务到的顾客越多越好。

## 2.3 p - 中心选址问题

p - 中心选址问题研究选取p个设施的位置来服务若干个顾客，使得最差的情况达到最优的问题。p - 中心问题的目标可以是最大距离最小，可以是最大反应时间最小，也可以是最大损失最小。

p - 中心问题也是由Hakimi提出的，可以分为绝对p - 中心问题和顶点p - 中心问题，其区别与p - 中值问题的分类中的两种问题的区别类似，所以一般也不区分这两种问题。在之后的研究中，Kariv和Hakimi证明了p - 中心问题是NP-hard问题；Hedetnie研究了无权网络中的1-中心问题，Migiddo和Supowit证明了p - 中心问题是NP-C问题的。

记顾客集合为，候选设施集合，任意顾客和候选设施之间都有一定的服务费用，目标就是在候选设施集合中选取部分设施，使得服务成本最大的顾客的成本最小，记W为服务成本最大的顾客的成本，即，则p - 中心选址模型可以描述为：

（13）

（14）

（15）

（16）

（17）

其中，为候选设施的序号，为顾客的序号，为候选设施是否为顾客服务，若候选设施为顾客服务，则为1，否则为0，为候选设施是否建设，若候选设施建设，则为1，否则为0。约束（14）式表示每个顾客有且仅有一个设施为其服务，约束（15）式表示只有建立设施才能为顾客服务，约束（16）式表示建立的设施数量为P，约束（17）式是变量的取值范围约束。

p - 中心问题与p - 中值问题的求解有很大的不同：对于无权顶点p - 中心问题有“猜测”算法；对于绝对p - 中心问题有Drezner和Wesolowsky提出的Drezner-Wesolowsky方法；对于顶点p - 中心问题有Daskin提出的二分搜索算法。同时，一些启发式算法也可以用来求解p - 中心问题，例如禁忌搜索算法、可变邻域搜索算法等等。

p - 中心选址问题常常应用在军队、医院、应急服务设施的选址上，因为这些设施通常有服务承诺，或者说这些设施的目标通常是降低最坏情况发生的风险，这一类目标通常被称为“经济平衡性”目标。

## 2.4 其他选址问题

后来，研究学者们以这三个选址问题作为基础，研究了更贴近实际也更复杂的一些选址问题模型，这些模型主要有带固定费用和容量限制的选址问题模型、多目标选址问题模型、可靠性选址问题模型、竞争选址问题模型、截流问题模型、hub选址问题模型、随机选址问题模型和动态选址问题模型等等。

## 2.5 主要研究问题和研究方法

### 2.5.1 容量限制的设施选址问题

本文主要研究带固定费用和容量限制的设施选址问题。带固定费用和容量限制的设施选址问题考虑现实中设施的建设费用以及设施的容量限制。而这一类模型又通常分为两类： 一类是无容量限制的固定费用选址问题（Uncapacitated Facility Location Problem, UFLP），这一类问题考虑设施建设费用，但是不考虑设施的容量限制。记顾客集合，一个候选设施集合，候选设施集合中的每个设施都有一定的建设费用，任意顾客和候选设施之间都有一定的服务费用，目标就是在候选设施集合中选取部分设施，使得每个顾客都有设施服务，并且尽量使总成本（设施的建设费用和设施与顾客的服务费用之和）最小，则强容量限制的选址问题模型的可以描述为下列的整数规划模型：

（18）

（19）

（20）

（21）

其中，为候选设施的序号，为顾客的序号，为候选设施是否为顾客服务，若候选设施为顾客服务，则为1，否则为0，为候选设施是否建设，若候选设施建设，则为1，否则为0。约束（19）式表示只有建立的设施才能为顾客提供服务，约束（20）式表示每个顾客有且仅有一个设施为其服务，约束（21）式是变量的取值范围约束。

另一类是带容量限制的选址问题（capacitated facility location problem, CFLP），本文主要研究的就是这一类问题。这一类问题又可以分为弱容量限制的选址问题（soft-capacitated facility location problem, SCFLP）和强容量限制的选址问题（hard-capacitated facility location problem, HCFLP）。弱容量限制的选址问题对设施的连接数加以限制，每个设施最多连接个顾客，但是每个地点允许建设多个设施；而强容量限制的是选址问题，除了限制每个设施最多连接个顾客，还限制每个地点仅能建设一个设施[1]。

### 2.5.2 竞争决策算法简介及基本概念

宁爱兵等人在文章中提出了一种新型的启发式算法—-竞争决策算法，这种算法可以用来求解组合优化的问题，并且取得了不错的解。选址问题也是一种组合优化问题，何永梅等人采用竞争决策算法对软容量限制的选址问题进行了求解，证实了竞争决策算法可以用来求解选址问题。

竞争决策算法（competitive decision algorithm, CDA）是在某种规则下，模拟竞争者和资源的竞争来求解问题的算法。

竞争决策算法的思想来自于自然界中优胜劣汰，适者生存的性质。自然界中资源的分配往往是在一定的初始状态下，在竞争者的绝对实力、竞争者间的相对实力差距、竞争者对环境的适应能力等多因素的影响下，经过多轮竞争和决策，最终达到另一种资源分配的均衡状态，如果这一状态的结果要优于初始状态，那么就达到了优化的目的。

竞争决策算法就是利用这一原理，通过模拟多个竞争者对资源进行竞争，根据一定的竞争原则对资源重新进行分配，最终使得一些更具优势的竞争者分配到更多的资源，而处于劣势的竞争者失去资源，最终达到总体目标最优或近似最优。

下面介绍一下竞争决策算法中的基本概念[7]：

竞争者：参与竞争的对象，可以占有和争夺资源。

资源：竞争者竞争的对象，可以被竞争者占有和争夺。

竞争决策状态：指的是竞争者占有资源的状态。初始状态就是竞争开始前竞争者占有的资源状态。结束状态就是最终竞争者占有的资源状态。

竞争决策均衡：指的是一种特殊的竞争决策状态。竞争决策均衡状态下，不存在一个竞争者根据竞争力函数和决策函数占有更多的资源。

竞争力函数：用于计算某个竞争者对某个资源的竞争力。

决策函数：用于每局竞争中分配资源给竞争者，并计算新的竞争决策状态。

总费用函数：用于计算当前竞争决策状态的优劣，通常就是问题的目标函数。

资源交换规则：在竞争决策均衡状态下，通过资源交换规则使竞争进入一个非稳定的竞争决策状态。

# 3 软容量限制的设施选址问题

## 3.1 模型描述

软容量限制的选址问题研究的是设施提供服务的能力有限，但是可以通过多次建设设施来使设施的服务能力得到提升，目标是追求建设费用和服务费用之和最小。

在软容量限制的设施选址问题中，我们有一个顾客集合，一个候选设施集合，候选设施集合中的每个设施都有一定的建设费用，且有一定的服务能力限制（最多服务几个顾客），但是，每个候选设施可以建设多次，每次建设都需要支付建设费用，任意顾客和候选设施之间都有一定的服务费用，目标就是在候选设施集合中选取部分设施，使得每个顾客都有设施服务，并且尽量使总成本（设施的建设费用和设施与顾客的服务费用之和）最小。

软容量限制的选址问题模型的可以描述为下列的整数规划模型：

（22）

（23）

（24）

（25）

（26）

（27）

其中，为候选设施的序号，为顾客的序号，为候选设施是否为顾客服务，若候选设施为顾客服务，则为1，否则为0，为候选设施是否建设，若候选设施建设，则为1，否则为0。约束（23）式表示只有建立的设施才能为顾客提供服务，约束（24）式表示候选设施服务的顾客数不能超过它的容量上限，约束（25）式表示每个顾客有且仅有一个设施为其服务，约束（26）（27）式是变量的取值范围约束。

## 3.2 数学性质

何永梅等人在文章中总结了软容量限制的设施选址问题的数学性质[2]，本文对何永梅等人总结的数学性质重新进行了整理和归纳，得到如下四条性质：

**性质1**（何永梅, 宁爱兵, 彭大江, 尚春剑, 张惠珍[2]）如果设施和设施满足下列条件：

（1）的建设费用小于的建设费用。

（2）存在顾客，的服务费用加的建设费用小于的服务费用。

则称对于顾客来说，设施相对设施为劣势设施，设施相对设施为优势设施。如果顾客可以由设施服务，则一定不由设施服务。

**证明：**在容量限制的选址问题中，如果顾客被设施服务，那么可以用设施来替代设施，且替代后可以使目标函数更优，故设施为顾客提供服务的解要优于设施为顾客提供服务的解。

特殊地，如果对于顾客来说，所有设施相对某一设施都为劣势设施，则该设施称为对于顾客来说的绝对优势设施。所以有下列性质：

**性质2**（何永梅, 宁爱兵, 彭大江, 尚春剑, 张惠珍[2]）如果设施满足下列条件：

（1）对任意设施，都满足的建设费用小于的建设费用。

（2）对任意设施，存在顾客，满足设施的服务费用加的建设费用小于设施的服务费用。

则设施称为对于顾客来说的绝对优势设施，设施一定建设为顾客服务。

**证明：**在容量限制的选址问题中，此性质显然成立。

**性质3**（何永梅, 宁爱兵, 彭大江, 尚春剑, 张惠珍[2]）如果设施和设施满足下列条件：

（1）的建设费用小于的建设费用。

（2）对于任意顾客，的服务费用小于的服务费用。

则称设施相对设施为劣势设施，设施相对设施为优势设施。如果顾客可以由设施服务，则一定不由设施服务。

**证明：**在容量限制的选址问题中，设施可以用设施来替代，且替代后可以使目标函数更优，故不建设设施的解要优于建设设施的解。

特殊地，如果所有设施相对某一设施都为劣势设施，则该设施称为绝对优势设施。所以有下列性质：

**性质4**（何永梅, 宁爱兵, 彭大江, 尚春剑, 张惠珍[2]）如果设施满足下列条件：

（1）对任意设施，都满足的建设费用小于的建设费用。

（2）对任意设施，和任意顾客，都满足的服务费用小于的服务费用。

则设施称为绝对优势设施，且设施一定建设。并且，设施一定服务个顾客。

**证明：**在容量限制的选址问题中，此性质显然成立。

## 3.3 竞争决策算法

何永梅等人利用竞争决策算法对软容量限制的设施选址问题进行了求解[2]，本文同样利用竞争决策算法对软容量限制的选址问题进行求解，并且在何永梅等人研究的基础上对算法流程进行了进一步简化和整理。

### 3.3.1 竞争决策算法概念

在选址问题中，我们可以将设施看作竞争者，将顾客看作被设施竞争的资源，设施为顾客服务即竞争者获得了资源；当然，也可以将顾客看作竞争者，将设施看作被竞争的资源，顾客获得了设施的服务即竞争者获得了资源。但是为了便于理解，我们通常将设施看作竞争者，将顾客看作被设施竞争的资源。设施为顾客服务的情况就是竞争决策状态；设施为顾客服务的初始情况就是竞争决策状态的初始状态；设施服务顾客的最终情况就是竞争决策状态的结束状态，通常也是竞争决策均衡状态。

容易想到，设施对顾客的服务成本和设施的建设成本越低，竞争力越大；所以，我们规定竞争力函数与设施的对顾客的服务费用和设施的建设费用成反比。因为在软容量限制的设施选址问题中，设施可以开设多次以提高自己的服务容量，所以如果服务该顾客不需要重新建立设施，即设施未确定建设，则竞争力函数值仅与设施的对顾客的服务费用成反比；否则，设施已经确定建设，竞争力函数值就与设施的对顾客的服务费用与设施的建设费用之和成反比。故软容量限制的设施选址问题中竞争力函数的表达式为：

（28）

相应地，因为软容量限制的设施选址问题中，设施可以开设多次，所以当设施容量已满，且仍要争夺资源时，就只需要重新开设一个设施，然后将资源分配给他即可。所以，软容量选址的设施选址问题中的决策函数为：在满足新的竞争决策状态的总费用函数值更小的前提下，分配顾客给竞争力函数值最大的设施。总费用函数就是目标函数，资源交换规则就是交换任意两个设施服务的任意两个顾客，使竞争进入一个非稳定的竞争决策状态。

### 3.3.2 竞争决策算法流程

**step 1：** 设所有设施服务的顾客均为空集。

根据性质1至性质4，绝对优势设施一定建设且服务的一定的顾客，相对劣势设施一定不服务顾客，所以关闭相对劣势设施，使之不参与竞争；同时，关闭绝对优势设施服务的顾客，使之不参与竞争。

**step 2：** 根据初始的竞争决策状态，计算每一个设施对每一个顾客的竞争力函数值。

**step 3：** 按照次序分配顾客

根据目前的竞争决策状态，将顾客按照次序分配给竞争力最大且容量未满的设施。

**step 4：** 按照次序争夺顾客：

**step 4.1：** 根据目前的竞争决策状态（假设被争夺的顾客没有被设施服务），更新每一个设施对被争夺的顾客的竞争力函数值。

**step 4.2：** 如果设施对顾客的竞争力大于服务顾客的设施，则根据决策函数，如果满足分配顾客给设施可以使总费用函数值减少，则将顾客分配给设施，并更新竞争决策状态，重新按照次序争夺顾客。

**step 5：** 如果任一设施都不能根据竞争力函数和决策函数获得更多的顾客，即达到竞争决策均衡，或者超过了最大竞争步数，则进入**step 6**；否则进入**step 4**。

**step 6：** 关闭没有服务顾客的设施，根据资源交换规则交换设施的顾客，并进入**step 4**重新进行竞争，如果竞争结果优于目前最优的竞争结果，则更新最优竞争结果。

**step 7：** 如果不能根据资源交换规则交换顾客，使竞争进入非稳定的竞争决策状态，或者超过了最大资源交换步数，则进入**step 8**；否则进入**step 6**。

**step 8：** 算法结束，输出目前最优的竞争结果。

### 3.3.3 算例

**算例 1**

设候选设施集合，顾客集合，设每个候选设施的服务容量均为2，候选设施的建设费用：

候选设施与顾客的服务费用：

**算法计算过程：**

第一步：根据性质1-4，没有设施必定开设，同时也没有设施必定为顾客服务，故初始的竞争决策状态为：所有设施服务的顾客为。

第二步：根据初始的竞争决策状态，计算所有设施的竞争力函数值，得到竞争力函数矩阵：

第三步：将剩余的顾客分配给竞争力函数最大且容量未满的设施。得到新的竞争决策状态，此时设施服务顾客的情况：，，，。

第四步：更新设施的竞争力函数值，得到竞争力函数矩阵：

第五步：设施按照次序争夺顾客：

（1）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力小于设施，且从中剥夺顾客给能够使总费用函数值更小，所以根据决策函数，从中剥夺顾客给，并更新竞争决策状态，重新按照次序争夺顾客。

（2）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（3）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力小于设施，且从中剥夺顾客给能够使总费用函数值更小，所以根据决策函数，从中剥夺顾客给，并更新竞争决策状态，重新按照次序争夺顾客。

（4）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（5）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（6）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（7）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（8）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

第六步：此时，对于、、、、，均没有竞争者可以根据竞争力函数和决策函数争夺到顾客，所以此时达到竞争决策均衡状态，将此时的竞争决策均衡状态记为最优竞争结果。

第七步：关闭没有服务顾客的设施，根据资源交换规则交换任意两个设施的任意两个顾客，重新竞争，竞争结果均不优于最优竞争结果，故转到第八步。

第八步：输出竞争结果，此时每个设施占有的顾客：，，，其余设施占有的顾客为，总费用函数值（即目标函数值）为60。

经验证，该解为整数规划的最优解，验证了算法的有效性。

## 3.4 基于混合策略的竞争决策算法

基于混合策略的竞争决策算法是一种改进的竞争决策算法。因为受到计算资源的限制，在竞争决策算法中，竞争并不是无限进行下去的，而是有最大竞争步数限制，当竞争的步数超过了最大竞争步数限制，也就达到了竞争决策算法的终止条件，我们会终止竞争，进入算法的下一步或者输出竞争结果。这时得到的解往往不是竞争决策均衡状态的解，仍有继续优化的空间。所以，我们引入混合策略的概念，在竞争决策算法达到终止条件之后，不输出竞争结果，而是根据当前的竞争决策状态，按照混合策略再分配一次顾客。具体的分配顾客混合策略就是，将顾客按照一定的概率分配给竞争力大于平均值的设施，分配的概率为：

（29）

其中，为所有设施对顾客的竞争力的平均值，即

按照混合策略分配完顾客后，计算按照混合策略分配的总费用函数值，如果按照混合策略分配的总费用函数值要小于按照纯策略分配的总费用函数值，则采用混合策略分配的结果作为输出的结果，否则就采用纯策略分配的结果作为输出结果。最终的输出结果可以表示为：

（30）

其中，表示竞争决策算法得到的解，表示基于混合策略重新分配得到的解，表示竞争决策算法得到的解的总费用函数值，表示基于混合策略重新分配得到的解的总费用函数值。

在得到混合策略的解之后，我们可以通过一定的方法对混合策略的解进行取整，从而得到不差于混合策略解的纯策略解，也就是更优的整数规划解。

### 3.4.1 算例

**算例 2**

以竞争决策算法求解的**算例 1**为例，验证基于混合策略的竞争决策算法的有效性。因为**算例 1**已经求得了整数规划问题的最优解，使用基于混合策略的竞争决策算法必然不能找到更优的解，所以，我们限制竞争决策算法流程中第四步的最大竞争步数，使得竞争决策算法经过竞争阶段后输出的解不是最优解，然后采用混合策略继续分配，查看混合策略得到的解是否更优。

对于**算例 1**，我们限制其最大竞争步数为1次，此时每个设施占有的顾客：，，，，总费用函数值（即目标函数值）为68。

此时，竞争力函数矩阵为：

根据竞争力函数矩阵，依次将顾客按照概率分配给设施：

（1）对于顾客，所有设施的竞争力平均值约为0.11，竞争力大于平均值的设施有、、，所以将顾客分别以0.33、0.29、0.38的概率分配给设施、、。

（2）对于顾客，所有设施的竞争力平均值约为0.12，竞争力大于平均值的设施有、、，所以将顾客分别以0.35、0.35、0.30的概率分配给设施、、。

（3）对于顾客，所有设施的竞争力平均值约为0.10，竞争力大于平均值的设施有、、，所以将顾客分别以0.30、0.30、0.40的概率分配给设施、、。

（4）对于顾客，所有设施的竞争力平均值约为0.10，竞争力大于平均值的设施有、、，所以将顾客分别以0.33、0.33、0.33的概率分配给设施、、。

（5）对于顾客，所有设施的竞争力平均值约为0.12，竞争力大于平均值的设施有、、，所以将顾客分别以0.33、0.4、0.3的概率分配给设施、、。

此时，计算总费用函数（即目标函数）的期望值：。可以看到，使用混合策略进行分配得到的总费用函数（即目标函数）值更小，验证了使用混合策略进行分配的有效性。

**其他算例**

为了更好的验证基于混合策略的竞争决策算法的有效性，我们采用NodeJS实现了软容量限制的设施选址问题下的基于混合策略的竞争决策算法，采用随机生成的方法，计算不同规模和限制下（设施数、顾客数、容量限制、最大竞争步数不同）的1000个算例的结果，然后选取了5种不同规模和限制的问题的20个纯策略解和混合策略解绘制了折线图进行比较：

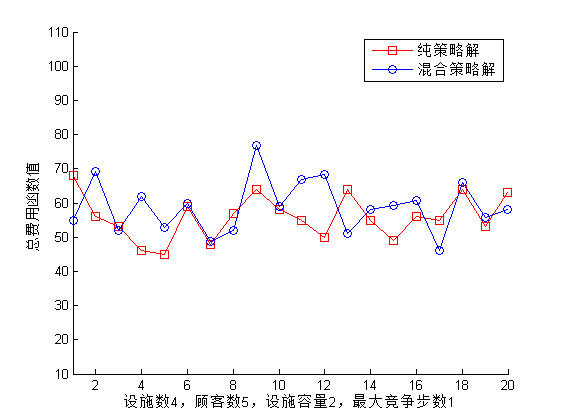
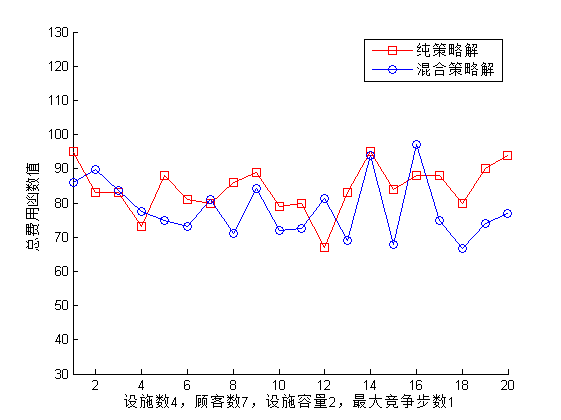
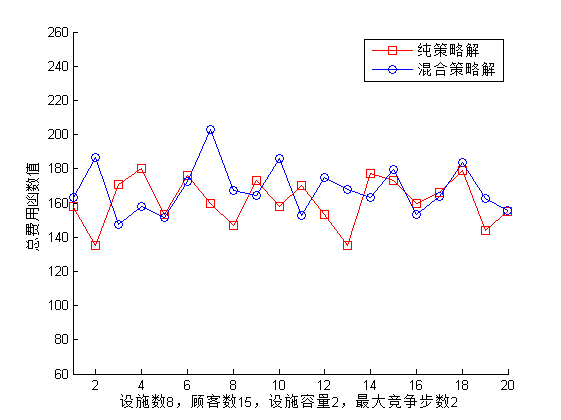
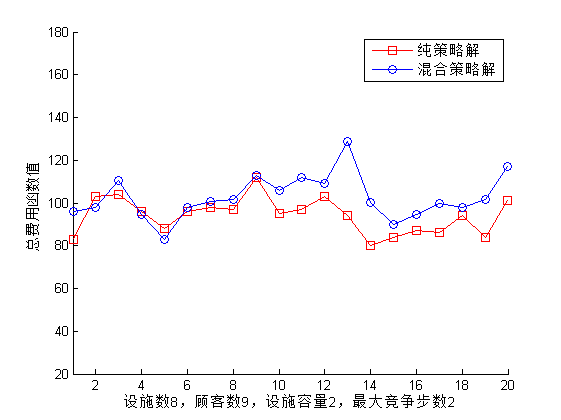
 

图1 图2

图3 图4

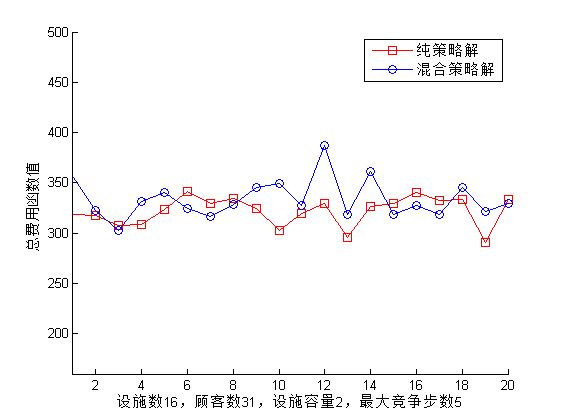


图5

从上面的数据中，我们可以看到：

（1）在相同的设施数、设施容量和竞争步数的情况下，顾客数越多，采用混合策略分配得到的解更优的比例就越大。

（2）在相同的顾客数、设施容量和竞争步数的情况下，设施数越少，采用混合策略分配得到的解更优的比例就越大。

（3）在相同的设施数、顾客数和竞争步数的情况下，设施容量越少，采用混合策略分配得到的解更优的比例就越大。

以上三点都说明了基于混合策略的竞争决策算法对于求解顾客数多、设施数和容量少的情景十分适用。

（4）在相同的设施数、设施容量和顾客数的情况下，竞争步数越少，采用混合策略分配得到的解更优的比例就越大，因为竞争步数越多，使用纯策略得到的解为最优解或局部最优解的几率就越大，使用混合策略得到的解的就往往不会比纯策略得到的解更优。

（5）对于较大规模（如图1所示）和较小规模（如图5所示）的问题，在达到最大竞争步数后使用混合策略再一次进行分配均有较高的较高的概率得到比纯策略更优的解。因此，基于混合策略的竞争决策算法是对竞争决策算法的一种有效的改进。

# 4 强容量限制的设施选址问题

## 4.1 模型描述

强容量限制的设施选址问题在生产和生活中十分常见。它研究的就是在考虑设施服务能力限制的条件下，选取设施的位置来服务顾客，使得总成本（设施建设成本和服务成本）最小的问题。

在强容量限制的设施选址问题中，我们有一个顾客集合，一个候选设施集合，候选设施集合中的每个设施都有一定的建设费用，且有一定的服务能力限制（最多服务几个顾客），任意顾客和候选设施之间都有一定的服务费用，目标就是在候选设施集合中选取部分设施，使得每个顾客都有设施服务，并且尽量使总成本（设施的建设费用和设施与顾客的服务费用之和）最小。

强容量限制的选址问题模型的可以描述为下列的整数规划模型：

（31）

（32）

（33）

（34）

（35）

其中，为候选设施的序号，为顾客的序号，为候选设施是否为顾客服务，若候选设施为顾客服务，则为1，否则为0，为候选设施是否建设，若候选设施建设，则为1，否则为0。约束（32）式表示只有建立的设施才能为顾客提供服务，约束（33）式表示候选设施服务的顾客数不能超过它的容量上限，约束（34）式表示每个顾客有且仅有一个设施为其服务，约束（35）式是变量的取值范围约束。

## 4.2 数学性质

可以证明，在强容量限制的选址问题中介绍的性质1至性质4在软容量限制的设施选址问题中依然成立。

## 4.3 竞争决策算法

该模型同样可以使用竞争决策算法进行求解，本文在软容量限制的设施选址问题的竞争决策算法的基础上进行了改进，给出了强容量限制的设施选址问题下的竞争决策算法流程。

### 4.3.1 竞争决策算法概念

因为在强容量限制的设施选址问题中，设施只能服务一定数量的顾客，且无法开设多次，所以，如果服务该顾客不需要重新建立设施，即设施未确定建设，则竞争力函数值仅与设施的对顾客的服务费用成反比；否则，设施已经确定建设，竞争力函数值就与设施的对顾客的服务费用与设施的建设费用之和成反比，故强容量限制的选址问题中的竞争力函数表达式为：

（36）

相应的，因为在强容量限制的设施选址问题中，设施只能服务一定数量的顾客，且无法开设多次，所以强容量限制的设施选址问题中的决策函数为：在满足新的竞争决策状态的总费用函数值更小的前提下，分配顾客给竞争力函数值最大的设施，如果设施的容量未满，则直接分配顾客给设施；如果设施的容量已满，则在设施服务的顾客中选取一个竞争力函数最小的顾客作为交换，然后分配顾客给设施。而强容量限制的设施选址问题中，总费用函数和资源交换规则与软容量限制的相同。

### 4.3.2 竞争决策算法流程

因为强容量限制的设施选址问题中，设施的服务能力有限，所以在分配顾客给设施的时候还需要考虑设施是否达到服务能力上限，具体的算法流程如下：

**step 1：** 设所有设施服务的顾客均为空集。

根据性质1至性质4，绝对优势设施一定建设且服务的一定的顾客，相对劣势设施一定不服务顾客，所以关闭绝对优势设施和相对劣势设施，使之不参与竞争；同时，关闭绝对优势设施服务的顾客，使之不参与竞争。

**step 2：** 根据初始的竞争决策状态，计算每一个设施对每一个顾客的竞争力函数值

**step 3：** 按照次序分配顾客

根据目前的竞争决策状态，将顾客按照次序分配给竞争力最大且容量未满的设施。

**step 4：** 按照次序争夺顾客

**step 4.1：** 根据目前的竞争决策状态（假设被争夺的顾客没有被设施服务），更新每一个设施对被争夺的顾客的竞争力函数值。

**step 4.2：** 如果设施对顾客的竞争力大于服务顾客的设施，则根据决策函数：

（1）如果设施容量未满，且满足分配顾客给设施可以使总费用函数值减少，则将顾客分配给设施，并更新竞争决策状态

（2）如果设施的容量已满，则在设施服务的顾客中选取一个竞争力函数最小的顾客作为交换分配给，如果满足分配顾客给设施可以使总费用函数值减少，则将顾客分配给设施，并更新竞争决策状态，重新按照次序争夺顾客。

**step 5：** 如果任一设施都不能根据竞争力函数和决策函数获得更多的顾客，即达到竞争决策均衡，或者超过了最大竞争步数，则进入**step 6**；否则进入**step 4**。

**step 6：** 关闭没有服务顾客的设施，根据资源交换规则交换设施的顾客，并进入**step 4**重新进行竞争，如果竞争结果优于目前最优的竞争结果，则更新最优竞争结果。

**step 7：** 如果不能根据资源交换规则交换顾客，使竞争进入非稳定的竞争决策状态，或者超过了最大资源交换步数，则进入**step 8**；否则进入**step 6**。

**step 8：** 算法结束，输出目前最优的竞争结果。

### 4.3.3 算例

**算例 3**

设候选设施集合，顾客集合，设每个候选设施的服务容量均为2，候选设施的建设费用：

候选设施与顾客的服务费用：

**算法计算过程：**

第一步：根据性质1-4，没有设施必定开设，同时也没有设施必定为顾客服务，故初始的竞争决策状态为：所有设施服务的顾客为。

第二步：根据初始的竞争决策状态，计算所有设施的竞争力函数值，得到竞争力函数矩阵：

第三步：将剩余的顾客分配给竞争力函数最大且容量未满的设施。得到新的竞争决策状态，此时设施服务顾客的情况：，，，其余设施服务的顾客为。

第四步：更新设施的竞争力函数值，得到竞争力函数矩阵：

第五步：设施按照次序争夺顾客：

（1）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力小于设施，且从中剥夺顾客给能够使总费用函数值更小，所以根据决策函数，从中剥夺顾客给，并更新竞争决策状态，重新按照次序争夺顾客。

（2）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（3）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（4）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力小于设施，且从中剥夺顾客给并从中换出竞争力小的顾客给能够使总费用函数值更小，所以根据决策函数，从中剥夺顾客给并从中换出顾客给，并更新竞争决策状态，重新按照次序争夺顾客。

（5）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（6）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（7）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（8）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

（9）对于，更新所在的列的竞争力函数矩阵值为：，当前服务的设施的竞争力最大，所以根据决策函数，不重新分配。

第六步：此时，对于、、、、，均没有竞争者可以根据竞争力函数和决策函数争夺到顾客，所以此时达到竞争决策均衡状态，将此时的竞争决策均衡状态记为最优竞争结果。

第七步：关闭没有服务顾客的设施，根据资源交换规则交换任意两个设施的任意两个顾客，重新竞争，竞争结果均不优于最优竞争结果，故转到第八步。

第八步：输出竞争结果，此时每个设施占有的顾客：，，，其余设施占有的顾客为，总费用函数值（即目标函数值）为59。

经验证，该解为整数规划的最优解，验证了算法的有效性。

## 4.4 基于混合策略的竞争决策算法

强容量限制的设施选址问题中，基于混合策略的竞争决策算法的基本概念与软容量限制的设施选址问题中的基本概念完全相同，这里直接通过算例来进行演示。

### 4.4.1 算例

**算例 4**

以竞争决策算法求解的**算例 3**为例，验证基于混合策略的竞争决策算法的有效性。类似的，因为**算例 3**已经求得了整数规划问题的最优解，使用基于混合策略的竞争决策算法必然不能找到更优的解，所以，我们限制竞争决策算法流程中第四步的最大竞争步数，使得竞争决策算法经过竞争阶段后输出的解不是最优解，然后采用混合策略继续分配，查看混合策略得到的解是否更优。

对于**算例 3**，我们限制其最大竞争步数为1次，此时每个设施占有的顾客：，，，其余设施服务的顾客为，总费用函数值（即目标函数值）为68。

此时，竞争力函数矩阵为：

根据竞争力函数矩阵，依次将顾客按照概率分配给设施：

（1）对于顾客，所有设施的竞争力平均值约为0.12，竞争力大于平均值的设施有、，所以将顾客分别以0.62、0.38的概率分配给设施、。

（2）对于顾客，所有设施的竞争力平均值约为0.14，竞争力大于平均值的设施有、、，所以将顾客分别以0.39、0.28、0.33的概率分配给设施、、。

（3）对于顾客，所有设施的竞争力平均值约为0.10，竞争力大于平均值的设施有、、，所以将顾客分别以0.30、0.39、0.30的概率分配给设施、、。

（4）对于顾客，所有设施的竞争力平均值约为0.13，竞争力大于平均值的设施有、，所以将顾客分别以0.58、0.42的概率分配给设施、。

（5）对于顾客，所有设施的竞争力平均值约为0.13，竞争力大于平均值的设施有、，所以将顾客分别以0.45、0.55的概率分配给设施、。

此时，计算总费用函数（即目标函数）的期望值：。可以看到，使用混合策略进行分配得到的总费用函数（即目标函数）值更小，验证了使用混合策略进行分配的有效性。

**其他算例**

为了更好的验证基于混合策略的竞争决策算法的有效性，我们采用NodeJS实现了强容量限制的设施选址问题下的基于混合策略的竞争决策算法，采用随机生成的方法，计算不同规模和限制下（设施数、顾客数、容量限制、最大竞争步数不同）的1000个算例的结果，然后选取了不同规模和限制的问题的20个纯策略解和混合策略解绘制了折线图进行比较：

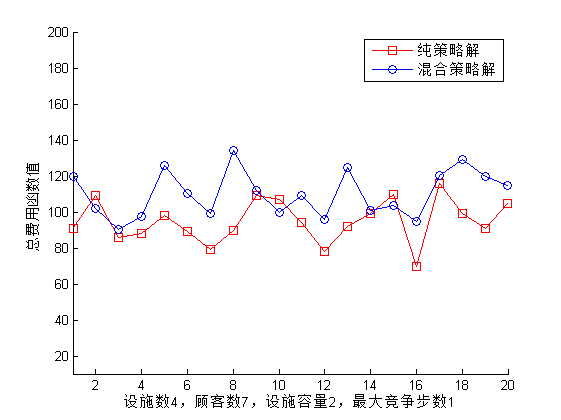
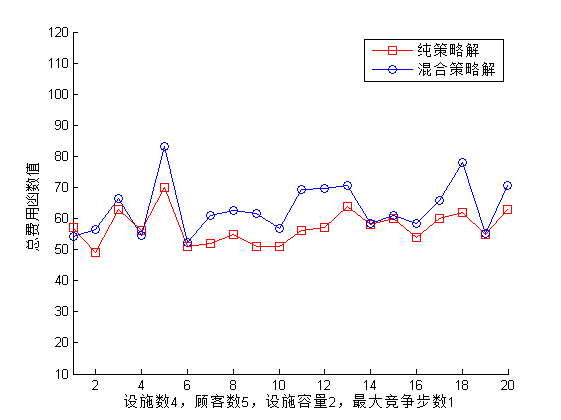


图6 图7

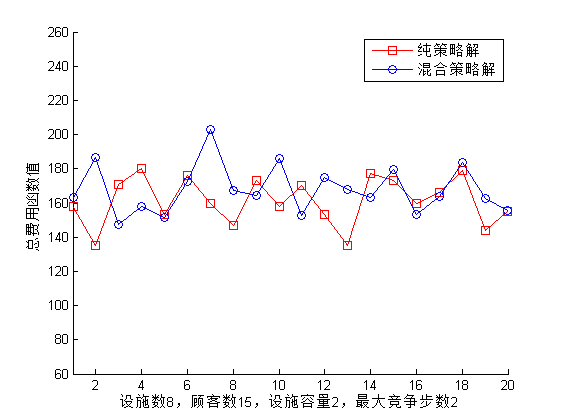
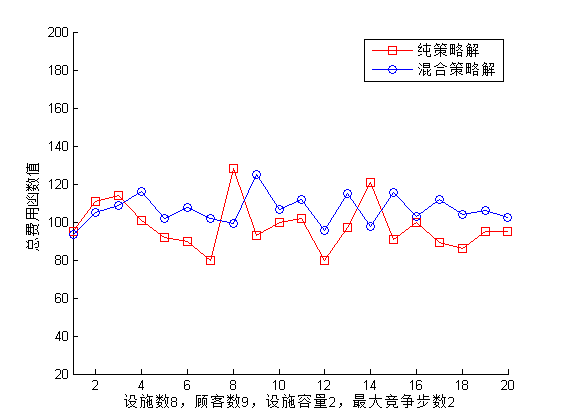


图8 图9

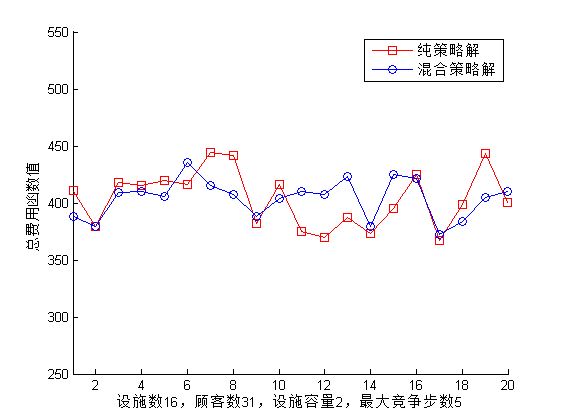


图10

从上面的数据中，我们可以看到，基于混合策略的竞争决策算法对于强容量限制的设施选址问题同样具有有效性，并且与软容量限制的设施选址问题中得到的结果大部分具有相同的性质，除此之外，可以看到：

（1）对于相同规模和最大竞争步数的问题，强容量限制的设施选址问题得到的解往往比软容量限制的设施选址问题的解更差，因为从整数规划的角度来讲，强软容量限制的设施选址问题的解的可行域要小于软容量限制的设施选址问题的解的可行域，所以最优解一般要差一些。

# 5 总结与讨论

本文在何永梅等人对软容量限制的设施选址问题下的竞争决策算法的研究基础上，重新归纳整理了软容量限制的设施选址问题下的竞争决策算法，并推广到强容量限制的设施选址问题中。同时，本文还对竞争决策算法进行了改进，提出了基于混合策略的竞争决策算法，即在竞争决策算法分配达到终止条件之后，再使用混合策略分配一次顾客，最终输出的结果取竞争决策算法的纯策略结果和混合策略的结果中的最优值。

本文通过NodeJS实现了两种问题下的竞争决策算法和基于混合策略的竞争决策算法，随机生成了大量算例来验证算法的有效性。结果表明，竞争决策算法在解决容量限制的选址问题时效果明显，能够在较短的时间内得到近似最优解，对于小规模的算例甚至能够得到全局最优解；而对于大规模的算例，在一定的计算资源限制下，可以得到较优的解。同时，通过引入混合策略，即使在计算资源限制下竞争决策算法仅仅得到了一般的解，再经过混合策略的再一次分配后，其混合策略解的期望有较高的概率优于纯策略解，因此我们可以通过一定的取整技巧得到比纯策略的竞争决策算法得到的解更优的解，这部分内容有待进一步的研究。并且，可以从算例结果数据中看到，本文提出的基于混合策略的竞争决策算法在处理顾客数相对设施数的较多的问题中表现优秀，而这类问题在现实中有很多应用场景，例如学校、医院等服务中心的选址，均属于顾客数远远大于设施数的选址问题。

此外，本文仅仅在竞争决策算法分配完成之后引入了混合策略，这其实并不是完全的引入混合策略，也就没有完全发挥出混合策略的优势。如果能够从算法一开始就引入混合策略，使每一次分配顾客都采用混合策略分配，这样就更加提高了算法的容错性。但是，由于混合策略是一种按概率的分配策略，所以在进行设施争夺顾客的迭代时，每种顾客分配情况都有一定概率发生，所以最终的分配情况会随迭代次数的增加而呈现指数倍的增加，这样算法显然是不能在可接受的时间和空间范围内得到结果的。所以，我们仅仅在竞争决策算法分配完成后引入了混合策略，这样就避免了迭代造成的分配情况指数递增的糟糕情况。而如何在竞争决策算法中完全引入混合策略也具有很大的讨论空间，有待于进一步的研究。

# 参考文献

[1]:安凤仙. k-层无容量限制的设施选址问题的一种算法. 淮阴工学院学报, 2009, 18(5):1-4.

[2]:何永梅, 宁爱兵, 彭大江, 尚春剑, 张惠珍. 软容量限制设施选址问题的竞争决策算法. 计算机工程与应用, 2019(3).

[3]:金婷婷, 王波, 宁爱兵. 最小顶点覆盖问题的竞争决策算法. 计算机工程与应用, 2011, 47(1):32-34.

[4]:刘勇, 马良, 宁爱兵. 给定限期条件下应急选址问题的量子竞争决策算法. 运筹与管理, 2011, 20(3):66-71.

[5]:宁爱兵, 马良. 度约束最小生成树(DCMST)的竞争决策算法. 系统工程学报, 2005, 20(6):630-634.

[6]:宁爱兵, 马良. 竞争决策算法及其在车辆路径问题中的应用. 管理科学学报,2005,8(6):10-18.

[7]:宁爱兵, 王波, 熊小华, 马良. 竞争决策算法原理及其应用. 上海理工大学学报, 2008, 30(4):369-373.

[8]:宁爱兵, 熊校华, 马良. 多目标旅行商问题的竞争决策算法. 计算机工程与应用, 2009, 45(34):13-16.

[9]:熊小华, 宁爱兵. 最小比率生成树的竞争决策算法. 计算机工程与应用,2012,48(28),47-51.

[10]:熊小华, 宁爱兵, 马良. 多目标 0-1 背包问题的元胞竞争决策算法. 计算机应用研究, 2010, 27(10):3680-3682.

[11]:杨丰梅, 华国伟, 邓猛, 黎建强. 选址问题研究的若干进展. 运筹与管理, 2005, 14(6):1-7.

[12]:Church R L, ReVelle C. Maximal covering location problem. Papers of the Regional Science Association, 1974, 32:101-118.

[13]:Hakimi S L. Optimum locations of switching centers and medians of a graph. Operations Research, 1964, 12:450-459.

[14]:Kariv O, Hakimi S L. An algorithmic approach to network location problem, part 2: the p-medians. SIAM Journal Applied Mathematics, 1979, 37:539-560.

[15]:Toregas C, Swian R, ReVelle C, et al. The location of emergency service facilities. Operations Research, 1971, 19:1363-1373.

[16]:Weber A. On the location of industries. Chicago: University of Chicago Press, 1929.

**致谢**

时光荏苒，白驹过隙，转眼间我的大学生活就要结束了。在论文即将完成之际，我要衷心的感谢在我的大学生活中为我提供过帮助和关心的人。

首先，要感谢我的指导老师肖汉老师，肖老师不仅在我论文研究的关键问题上为我提供了解决思路，还在我论文写作的每一个环节严格把关，可以说这篇论文的顺利完成都是老师悉心指导和严格要求的结果。

其次，要感谢学院各位老师四年来的悉心教导、各位同学四年的陪伴和关照以及所有爱我的人的默默支持。

最后，感谢百忙之中参加论文答辩的老师们，感谢您对本文提出的宝贵建议，使我能够意识到自己的不足并加以改正。

**附录1:**

cda.js

**const** { rand, copyMatrix, bigger, smaller, columnMatrix, sumArr } = require('./function');

**const** { Mode, F, C, H, D, U, MaxLoopTimes, MaxExchangeTimes } = require('./data');

*// 竞争力函数*

**const** compete = (i, j, x, y, H, D) **=>** {

if (sumArr(x[i]) > 0) {

*// return 1 / D[i][j];*

return +((1 / D[i][j]).toFixed(4));

}

*// return 1 / (H[i] + D[i][j]);*

return +((1 / (H[i] + D[i][j])).toFixed(4));

}

*// 竞争力函数*

**const** competeSoft = (i, j, x, y, H, D, U) **=>** {

if (sumArr(x[i]) % U > 0) {

*// return 1 / D[i][j];*

return +((1 / D[i][j]).toFixed(4));

}

*// return 1 / (H[i] + D[i][j]);*

return +((1 / (H[i] + D[i][j])).toFixed(4));

}

*// 目标函数*

**const** costFunction = (x, H, D, U) **=>** {

**let** sum, min = 0;

for (**let** i = 0; i < x.length; i++) {

sum = 0;

for (**let** j = 0; j < x[i].length; j++) {

sum += x[i][j];

if (x[i][j]) {

min += D[i][j];

}

}

min = min + H[i] \* Math.ceil(sum / U);

}

return min;

}

*// 期望目标函数*

**const** expectCostFunction = (x, H, D, U, K) **=>** {

**let** sum, min = 0;

**let** minX = rand(x.length, x[0].length, [0, 1]);

for (**let** i = 0; i < x.length; i++) {

sum = 0;

for (**let** j = 0; j < x[i].length; j++) {

**let** sumK = 0;

for (**let** indexI = 0; indexI < x.length; indexI++) {

if (sumArr(x[indexI]) > 0 && K[indexI][j] > (sumArr(columnMatrix(K, j)) / x.length)) {

sumK += K[indexI][j];

}

}

if (sumArr(x[i]) > 0 && K[i][j] > (sumArr(columnMatrix(K, j)) / x.length)) {

*// min += D[i][j] \* K[i][j] / sumK;*

*// minX[i][j] = K[i][j] / sumK;*

min += D[i][j] \* (+((K[i][j] / sumK).toFixed(4)));

minX[i][j] = +((K[i][j] / sumK).toFixed(4));

}

sum += minX[i][j];

}

*// min = min + H[i] \* Math.ceil(sum / U);*

min = min + H[i] \* Math.ceil(sum / U);

}

return [min, minX];

}

**const** cda = (Mode, F, C, H, D, U, MaxLoopTimes, MaxExchangeTimes) **=>** {

*/\*\* 参数*

*\* F设施数*

*\* C顾客数*

*\* U容量限制*

*\* H建造费用向量*

*\* D服务费用矩阵*

*\*/*

*//console.log('H', H);*

*//console.log('D', D);*

*//console.log('U', U);*

*/\*\**

*\* cda算法*

*\*/*

*/\*\**

*\* 第零步：初始化*

*\*/*

**let** mustX = rand(F, C, [0, 1]);

**let** mustY = rand(1, F, [0, 1]);

**let** x = rand(F, C, [0, 1]);

**let** y = rand(1, F, [0, 1]);

*/\*\**

*\* 第一步：根据性质降阶*

*\*/*

*//console.log('第一步：根据性质降阶');*

*// 性质2*

**let** minIndex = H.indexOf(Math.min(...H));

for (**let** j = 0; j < C; j++) {

**let** isFMustServerC = true;

for (**let** i = 0; i < F; i++) {

if (i !== minIndex && D[i][j] < H[minIndex] + D[minIndex][j]) {

isFMustServerC = false;

break;

}

}

if (isFMustServerC) {

mustX[minIndex][j] = 1;

mustY[minIndex] = 1;

}

}

*// 性质3*

for (**let** i = 0; i < F; i++) {

for (**let** j = i + 1; j < F; j++) {

if (H[i] <= H[j] && smaller(D[i], D[j])) {

mustY[j] = -1;

} else if (H[i] >= H[j] && bigger(D[i], D[j])) {

mustY[i] = -1;

}

}

}

*/\*\**

*\* 第二步：计算竞争力函数矩阵*

*\*/*

*//console.log('第二步：计算竞争力函数矩阵');*

x = copyMatrix(mustX);

y = copyMatrix(mustY);

**let** K = rand(F, C, [0, 1]);

**let** newCompete = () **=>** {

for (**let** i = 0; i < F; i++) {

for (**let** j = 0; j < C; j++) {

K[i][j] = compete(i, j, x, y, H, D);

}

}

for (**let** i = 0; i < F; i++) {

for (**let** j = 0; j < C; j++) {

if (mustX[i][j]) {

for (**let** newI = 0; newI < F; newI++) {

if (!mustX[newI][j]) {

K[newI][j] = 0;

} else {

K[newI][j] = 1;

}

}

}

}

if (sumArr(mustX[i]) === U) {

for (**let** newJ = 0; newJ < C; newJ++) {

if (!mustX[i][newJ]) {

K[i][newJ] = 0;

} else {

K[i][newJ] = 1;

}

}

}

}

}

**let** newCompeteSoft = () **=>** {

for (**let** i = 0; i < F; i++) {

for (**let** j = 0; j < C; j++) {

K[i][j] = competeSoft(i, j, x, y, H, D, U);

}

}

for (**let** i = 0; i < F; i++) {

for (**let** j = 0; j < C; j++) {

if (mustX[i][j]) {

for (**let** newI = 0; newI < F; newI++) {

if (!mustX[newI][j]) {

K[newI][j] = 0;

} else {

K[newI][j] = 1;

}

}

}

}

}

}

Mode ? newCompete() : newCompeteSoft();

*//console.log(K);*

*/\*\**

*\* 第三步：分配顾客*

*\*/*

*//console.log('第三步：分配顾客');*

**const** FacilityDistributeCustom = () **=>** {

for (**let** j = 0; j < C; j++) {

**const** ConstKCol = columnMatrix(K, j);

**let** KCol = columnMatrix(K, j);

KCol.sort();

for (**let** index = 0; index < F; index++) {

*// 如果根据数学性质当前顾客已有设施服务，则跳过顾客*

**let** mustXCol = columnMatrix(mustX, j);

if (sumArr(mustXCol) > 0) {

continue;

}

**let** maxNum = KCol[KCol.length - 1 - index];

**let** maxIndex = ConstKCol.indexOf(maxNum);

if (sumArr(x[maxIndex]) < U) {

x[maxIndex][j] = 1;

y[maxIndex] = 1;

break;

}

}

}

}

FacilityDistributeCustom();

Mode ? newCompete() : newCompeteSoft();

*//console.log(x);*

*/\*\**

*\* 第四步：争夺顾客*

*\*/*

*//console.log('第四步：争夺顾客');*

**const** FacilityCompeteCustom = (x, y) **=>** {

**let** loopTimes = 0;

**let** cost = costFunction(x, H, D, U);

while (loopTimes < MaxLoopTimes) {

**let** j = 0;

for (j = 0; j < C; j++) {

*// 如果根据数学性质当前顾客已有设施服务，则跳过顾客*

**let** mustXCol = columnMatrix(mustX, j);

if (sumArr(mustXCol) > 0) {

continue;

}

*// 当前服务顾客的设施*

**let** xCol = columnMatrix(x, j);

**let** serverF = xCol.indexOf(1);

if (serverF < 0) {

continue;

}

*// 更新竞争力函数*

x[serverF][j] = 0;

if (sumArr(x[serverF]) === 0) {

y[serverF] = 0;

}

Mode ? newCompete() : newCompeteSoft();

*/\*\**

*\* 争夺顾客*

*\*/*

*//console.log('当前顾客', j);*

*//console.log('当前竞争力函数矩阵', K);*

*// 竞争力最大的设施*

**let** KCol = columnMatrix(K, j);

**let** maxIndex = KCol.indexOf(Math.max(...KCol));

*// 如果竞争力最大的设施没有服务顾客*

if (serverF !== maxIndex) {

*// 强容量限制的设施选址问题中如果竞争力最大的设施容量已满，则换出竞争力最大的设施服务的一个顾客*

if (Mode && sumArr(x[maxIndex]) == U) {

*// 换出竞争力最大的设施服务的顾客中竞争力最小的一个顾客*

**const** ConstKRow = copyMatrix(K[maxIndex]);

**let** KRow = copyMatrix(K[maxIndex]);

**let** KRowMinNum, KRowMinIndex;

KRow.sort();

for (**let** index = 0; index < C; index++) {

KRowMinNum = KRow[index];

KRowMinIndex = ConstKRow.indexOf(KRowMinNum);

if (x[maxIndex][KRowMinIndex] === 1) {

x[maxIndex][KRowMinIndex] = 0;

x[serverF][KRowMinIndex] = 1;

break;

}

}

*// 分配顾客给设施*

x[maxIndex][j] = 1;

y[serverF] = 1;

*// 如果分配顾客可以使总费用函数更小，则分配顾客给设施*

**let** newCost = costFunction(x, H, D, U);

if (newCost < cost) {

cost = newCost;

*//console.log('重新分配', x);*

break;

} else {

x[maxIndex][KRowMinIndex] = 1;

x[serverF][KRowMinIndex] = 0;

x[maxIndex][j] = 0;

x[serverF][j] = 1;

if (sumArr(x[serverF]) === 0) {

y[serverF] = 0;

}

}

} else {

*// 如果软容量限制的设施选址问题中或竞争力最大的设施容量未满，则分配顾客给设施*

x[maxIndex][j] = 1;

y[maxIndex] = 1;

*// 如果分配顾客可以使总费用函数更小，则分配顾客给设施*

**let** newCost = costFunction(x, H, D, U);

if (newCost < cost) {

cost = newCost;

*//console.log('重新分配', x);*

break;

}

}

} else {

*// 如果竞争力最大的设施已经服务顾客*

x[maxIndex][j] = 1;

y[maxIndex] = 1;

}

}

if (j === C) {

*//console.log('竞争决策均衡');*

break;

}

loopTimes++;

}

}

FacilityCompeteCustom(x, y);

*/\*\**

*\* 第五步：资源交换*

*\*/*

*//console.log('第五步：资源交换');*

Mode ? newCompete() : newCompeteSoft();

**const** FacilityExchangeCustom = () **=>** {

**let** exchangeTimes = 0;

**let** cost = costFunction(x, H, D, U);

while (exchangeTimes < MaxExchangeTimes) {

**let** newX = copyMatrix(x);

**let** newY = copyMatrix(y);

**let** first = rand(1, 1, [0, C]);

**let** second = rand(1, 1, [0, C]);

if (first !== second) {

*//console.log('交换顾客', first, second);*

**let** firstCol = columnMatrix(newX, first);

**let** secondCol = columnMatrix(newX, first);

**let** firstIndex = firstCol.indexOf(1);

**let** secondIndex = secondCol.indexOf(1);

if (firstIndex !== secondIndex) {

newX[firstIndex][second] = 1;

newX[secondIndex][first] = 1;

newX[firstIndex][first] = 0;

newX[secondIndex][second] = 0

}

FacilityCompeteCustom(newX, newY);

**let** newCost = costFunction(newX, H, D, U);

if (newCost < cost) {

x = newX;

y = newY;

cost = newCost;

}

}

exchangeTimes++;

}

}

FacilityExchangeCustom();

*/\*\**

*\* 第六步：输出结果*

*\*/*

**let** cost = costFunction(x, H, D, U);

*//console.log('纯策略解', x);*

*//console.log('总费用函数', cost);*

*// 期望*

Mode ? newCompete() : newCompeteSoft();

**let** [expectCost, expectCostX] = expectCostFunction(x, H, D, U, K, mustX);

*//console.log('混合策略解', expectCostX);*

*//console.log('总费用函数', expectCost);*

return [cost, expectCost];

}

cda(Mode, F, C, H, D, U, MaxLoopTimes, MaxExchangeTimes);

module.exports = {

compete,

competeSoft,

costFunction,

expectCostFunction,

cda,

}

function.js

*// 随机矩阵*

**const** rand = (x, y, scope) **=>** {

if (!scope || !scope.length || scope[0] > scope[1]) {

console.log('Error: 未输入取值范围或取值范围错误');

return null;

}

*// 随机数组*

**const** randArray = (x, scope) **=>** {

*// 随机数*

**const** randNumber = (scope) **=>** {

return Math.floor(Math.random() \* (scope[1] - scope[0]) + scope[0]);

}

*// 1 x 1数组*

if (x === 1) {

return randNumber(scope);

}

*// 1 x n数组*

**let** randArray = [];

for (**let** i = 0; i < x; i++) {

randArray.push(randNumber(scope));

}

return randArray;

}

*// 1 x n矩阵*

if (x === 1) {

return randArray(y, scope);

}

if (y === 1) {

return randArray(x, scope);

}

*// n x n矩阵*

**let** randMatrix = [];

for (**let** i = 0; i < x; i++) {

randMatrix.push(randArray(y, scope));

}

return randMatrix;

}

*// 复制矩阵*

**const** copyMatrix = (x) **=>** {

**const** copyArray = (x) **=>** {

**let** yArray = [];

for (**let** j = 0; j < x.length; j++) {

yArray.push(x[j]);

}

return yArray;

}

*// 1 x n矩阵*

if (typeof x[0] === 'number') {

return copyArray(x);

}

*// n x n矩阵*

**let** y = [];

for (**let** i = 0; i < x.length; i++) {

y.push(copyArray(x[i]));

}

return y;

}

*// 获取矩阵的一列*

**const** columnMatrix = (matrix, index) **=>** {

if (matrix.length === 0 || matrix[0].length === 0) {

console.log('Error: 矩阵长度为0');

return null;

}

if (matrix[0].length < index + 1) {

console.log('Error: 索引超出矩阵长度');

return null;

}

**let** columnMatrix = [];

for (**let** i = 0; i < matrix.length; i++) {

columnMatrix.push(matrix[i][index]);

}

return columnMatrix;

}

**const** bigger = (x, y) **=>** {

if (x.length !== y.length) {

console.log('Error: 数组长度不相同');

return null;

}

for (**let** i = 0; i < x.length; i++) {

if (x[i] < y[i]) {

return false;

break;

}

}

return true;

}

**const** smaller = (x, y) **=>** {

if (x.length !== y.length) {

console.log('Error: 数组长度不相同');

return null;

}

for (**let** i = 0; i < x.length; i++) {

if (x[i] > y[i]) {

return false;

break;

}

}

return true;

}

*// 数组求和*

**const** sumArr = (arr) **=>** {

if (!arr || !arr.length) {

console.log('Error:参数应为数组');

return null;

}

**let** sum = 0;

for (**let** i = 0; i < arr.length; i++) {

sum += arr[i];

}

return sum;

}

module.exports = {

rand,

copyMatrix,

columnMatrix,

bigger,

smaller,

sumArr,

}

data.js

**const** { rand } = require('./function');

*/\*\**

*\* Mode: true--强容量限制， false--软容量限制*

*\*/*

**const** Mode = false;

**const** F = 8;

**const** C = 9;

**const** U = 2;

**const** MaxLoopTimes = 3;

**const** MaxExchangeTimes = 0;

**let** H = rand(1, F, [5, 15]);

**let** D = rand(F, C, [5, 10]);

**const** dataFunction = () **=>** {

**let** H = rand(1, F, [5, 15]);

**let** D = rand(F, C, [5, 10]);

return {

Mode,

F,

C,

U,

H,

D,

MaxLoopTimes,

MaxExchangeTimes,

};

}

module.exports = {

Mode,

F,

C,

U,

H,

D,

MaxLoopTimes,

MaxExchangeTimes,

dataFunction,

}