参赛密码 _____

(由组委会填写)

“华为杯”第十四届中国研究生数学建模竞赛

学　　校	南京信息工程大学
参赛队号	10300016
队员姓名	1. 宁　俊
	2. 张　旭
	3. 施必成

参赛密码 _____

（由组委会填写）

“华为杯”第十四届中国研究生数学建模竞赛

题 目 多波次导弹发射中的规划问题

摘 要：

当前高技术条件下战争的突发性急骤增强，机动路线方案制定的好坏直接决定着导弹部队的暴露时间和导弹最终的打击效果。执行多波次导弹发射任务时，需要安排一个较优的机动部署方案，以在短时间内将导弹从待机地域部署到发射阵地，并合理分配转载地域与机动路线，实现多波次打击。车载运输装置的机动部署会涉及阵地路网的冲突性、隐蔽性、道路路况限制等多方面因素，通过建立优化模型，可以为导弹部队制定作战计划提供定量支持，有助于提高导弹部队作战指挥的自动化和整体决策能力。

针对问题一，采用 Dijkstra 算法得到各功能地域间的最短机动路径，基于甘特图建立模型约束，量化机动过程中额外的会车耗时。在满足模型约束的情况下，通过多阶段线性规划，确定各波次打击中发射点位与转载地域分配，基于时序策略确定发车次序与发车间隔。

针对问题二，以各转载地域为中心计算发射点位的密度分布和节点度中心度 degree centrality，建立各潜在转载地域区位状态模型。基于线性规划寻找最优转载地域。

针对问题三，根据第一波次打击暴露时间，第二波次机动耗时，寻找需被代替的最慢车，根据潜在节点到各剩余发射点位的最短路径，安排三辆代替发射装置待机位置。

针对问题四，构建了基于 k 均值聚类模型-道路节点 Betweeness Centrality 值发射任务规划模型，通过建立道路节点密度值、Betweeness Centrality 值和两者相结合的 P 值指标，来衡量路网中某个节点潜在受到敌方导弹攻击破坏的可能

性。

针对问题五，基于问题四获得的路网节点的潜在收攻击可能性，利用每个节点的 Betweeness Centrality 值对道路里程进行加权，使得道路流通度高的边消耗时间更长，在优化模型中减小分配发射装置通过其次数，从而在分散策略规划任务时，使得发射装置的最长暴露时间。

关键词: Dijkstra 算法，k 均值聚类，多阶段决策优化，Betweeness Centrality

一、问题重述

1.1 问题背景

导弹是现代高科技武器的结晶和化身，具有不同于一般进攻性武器的突出特点，尤其是威力大、射程远、精度高、突防能力强的显著特性，使其成为具有超强进攻性和强大威慑力的武器，成为维持战略平衡的支柱，特别是战时火力打击的效果往往是决定战争胜负的关键。

导弹的使用使战争的突发性和破坏性增大，规模和范围扩大，进程加快，从而改变了过去常规战争的时空观念，给现代战争的战略战术带来深远的影响。导弹的车载发射装置，平时部署在待机区域，收到发射任务后，能携带导弹沿阵地路网，快速抵达指定发射点位实施导弹发射。

在当前高技术条件下的现代化战争，由于战争的突发性急骤增强，对导弹部队的机动能力提出了更高的要求。所以机动路线方案制定的好坏直接决定着导弹部队的暴露时间和导弹最终的打击效果。因此，在给定的作战意图下，指挥人员需要根据具体的打击目标和现有的武器条件，将可用的弹型、弹量、火力投射单位最优配置到各个发射点位，以追求最快和最优的打击效果。

1.2 问题分析

导弹部队的车载发射装置，在待机地域隐蔽待命。收到发射任务后沿阵地路网机动到发射点位实施发射。

1.2.1 阵地情况

导弹部队的作战阵地由待机地域、转载地域、发射点位、道路节点组成，阵地道路连接各单位。道路分为主干道路和其它道路，其中主干道路是双车道，其它道路是单车道。当阵地道路阻塞时，车载发射装置在各道路节点等待会车。

待机地域是车载发射装置第一波次攻击的出发点，平时车载发射装置在待机地域隐蔽待命。

当车载发射装置完成一个波次的发射任务后，将前往转载地域重新装弹。转载地域最多同时容纳两台发射车，每次只能给一台车载发射装置装弹，每次装弹作业耗时 10 分钟。

车载发射装置机动到发射点位执行导弹发射任务。每个发射点位只能容纳 1 台车载发射装置，执行多波次发射任务时(以两波次为例)，每个发射点位仅能使用一次。

1.2.2 车载发射装置

导弹部队使用导弹车载发射装置进行攻击，车载发射装置只能载弹一枚。本问题中共有车载发射装置 A、B、C 三类，共 24 台。其中 A、B、C 三类发射装置的数量分别为 6 台、6 台、12 台。

A、B、C 三类车载发射装置在主干道路上的平均速度分别为 70 公里/小时、60 公里/小时、50 公里/小时；在其他道路上的平均行驶速度分别是 45 公里/小时，35 公里/小时，30 公里/小时。

1.2.3 作战流程

本问题假设执行两波次导弹发射命令：

第一波次攻击:车载发射装置从待机地域沿阵地路网机动至第一波次发射点位实施导弹发射攻击。

第二波次攻击:车载发射装置从第一波次发射点位沿阵地路网机动至转载地域

重新装弹。装弹完成后机动至第二波次发射点位实施导弹发射攻击。

1.2.4 评价指标

暴露时间：对于单台车载发射装置，是指车载发射装置从待机地域出发时刻至第二波次发射时刻为止的时间段。其中车载发射装置位于转载地域内的时间不计入暴露时间内。

整体暴露时间：表示所有车载发射装置的暴露时间之和。

1.3 需要解决的问题

1）需要给出具体的发射点位分配及机动路线方案，使得完成两个波次发射任务的整体暴露时间最短。

2）需要在道路节点 J25、J34、J36、J42、J49 附近临时增设 2 个转载地域，使得完成两个波次发射任务的整体暴露时间最短。

3）新增 3 台 C 类发射装置用于第二波次发射。在道路节点 J04、J06、J08、J13、J14、J15 附近隐蔽待机，合理选择这 3 台车载发射装置的隐蔽待机点位置，在转载地域不变的情况下，使得完成两个波次发射任务的整体暴露时间最短。

4）建立合理的评价指标，结合阵地路网特点，考虑攻防双方的对抗博弈，量化分析该路网最可能受到敌方攻击破坏的 3 个道路节点。

5）充分考虑规避敌方的侦察和打击等，综合考虑采用适当分散机动的策略，缩短单台发射装置的最长暴露时间等因素，重新讨论问题（1）。

二、问题分析

2.1 战术分析

本问题中的多次波次导弹打击任务[1]，选择了齐射和不重复使用同一发射点位的战术手段。

不重复使用同一发射点位：该战术的目的是为了隐蔽发射点位，防止阵地布置信息泄露，延长发射阵地使用寿命。

齐射：导弹齐射战术可以让指挥员掌握的火力在极短时间内发射完毕，同时车载发射装置可以立刻隐蔽疏散。由于在阵地布置时考虑到指挥通信和作战保障等问题，一般将车载发射装置集中部署在方圆数千米或数十千米的范围内，当执行打击任务时，敌人可以很快通过空、天侦察系统发现，并组织反击。选择突然袭击的齐射战术，不仅可以降低执行多波次打击任务过程中被报复打击的风险。还可以瞬间覆盖目标地区，成倍提高打击效率。

基于以上战术选择，问题要求我们合理安排发射点位，规划车载发射装置的机动路径，以及合理安排发车次序和发车间隔，实现最短整体暴露时间和多波次齐射的作战目标。

2.2 耗时分析

多次波次打击任务中，车载发射装置的机动位置变化及耗时包括：

1）道路上的机动耗时

表示车载发射装置在各个道路节点间行驶过程中消耗的时间。

2）道路节点上的会车等待耗时

由于道路车道有限，所以车载发射装置间的行进路线会有重合的情况，此时晚来的车载发射装置需要在道路节点处等待会车。

3）发射点位处的齐射等待耗时

由于齐射的战术安排，先到达发射点位的车辆需要等待，直到所有车载发射装置就位。

4）转载地域内的装弹作业耗时

转载地域为完成一个波次发射任务的车载发射装置补充弹药，装弹作业耗时10分钟。

5）转载地域内的等待耗时

转载地域每次只能为一台车载发射装置进行装弹作业，转载地域内的另一辆车只能等待。

6）转载地域外的等待耗时

由于转载地域仅能最多同时容纳2台车载发射装置，当转载地域满员时，车载发射装置停留于转载地域附近的道路节点处等待。

而问题所设定的暴露时间评价指标即由：机动耗时，会车等待耗时，齐射等待耗时，以及转载地域满员时的等待耗时组成。

2.3 最短路径与耗时等待的矛盾分析

在理想情况下，自攻击任务开始，所有的车载发射装置沿最短路径保持机动，无任何的会车等待耗时和转载点等待耗时，但是往往因为实际的阵地路网分布和发车次序与发车间隔等差异，达不到理想状况。机动路线发规划和发车时刻的安排中往往存在以下矛盾：

1）最短路径与会车等待的矛盾

由于实际的阵地路网分布，不同车载发射装置前往发射点位的最短路径往往存在路段重合的情况，导致潜在的会车等待耗时，而为了避免潜在的会车耗时而绕路，又会增加机动耗时。

2）齐射等待耗时与机动耗时的矛盾

为了实现一个波次的齐射，先到达发射点位的车辆必须等待最慢的车辆。为减少每辆车的齐射等待耗时，要求每辆车自本波次开始至到达发射点位的耗时方差尽可能的小，而这可能增加额外的机动耗时。

3）发车时刻与齐射等待耗时的矛盾

为减少发生会车的可能，我们可以根据发射点位到达会车地点的距离和车辆的速度的快慢来选择发车时刻和发车次序。但发车时刻的变化可能增加齐射等待耗时。

4） 转载地域的选择与转载等待耗时的矛盾

车载发射装置一般会选择最近的转载地域，但由于实际的阵地路网分布,导致转载地域间的负荷不均，反而增加转载等待耗时。

三、模型假设

根据问题信息，进行如下假设与约束：

1) 阵地路网的道路节点位置分布和道路连接情况已知。除主干道可以双向通行外，其它道路只能单向行驶，各道路节点处可以会车。
2) 每辆车只能载弹一枚，车载发射装置在作战前均已载弹。
3) A、B、C三类车载发射装置共24台，其中A、B、C三类的数量分别为6台、6台、12台。
4) 车载发射装置平均部署在各个待机地域。
5) A、B、C三类车载发射装置在主干道路上的平均速度分别为70公里/小时、60

公里/小时、50 公里/小时；在其他道路上的平均行驶速度分别是 45 公里/小时，35 公里/小时，30 公里/小时。

6) 每个转载地域存放的导弹种类齐全，数量充足。
7) 实施多波次发射时，同一发射点位只能使用一次。
8) 转载地域最多容纳 2 辆车载发射装置，每次只能为一辆车装弹，单辆发射车平均作业时间为 10min。
9) 不考虑发射装置在发射点位必要的技术准备时间和发射后发射装置的撤收时间。
10) 不考虑发射车发生技术故障的情况（行驶障碍，发射障碍）。
11) 不考虑执行多波次打击任务期间道路发生损坏的情况。

四、符号说明与名词解释

4.1 通用符号说明

序号	符号	符号说明
1	$D_i(i=1,2)$	2 个待机地域
2	$Z_i(i=1,2,...,6)$	6 个转载地域
3	$J_i(i=1,2,...,62)$	62 个道路节点
4	$F_i(i=1,2,...,60)$	60 个发射点位
5	A、B、C	车载发射装置的类型
6	$v_b(b=1,2,3)$	A,B,C 三种车型在主干道的速度
7	$v_c(c=1,2,3)$	A,B,C 三种车型在非主干道的速度

4.2 名词解释

整体暴露时间为每辆车暴露时间之和:

每辆车暴露时间 = 路径机动耗时 + 会车等待耗时 + 转载点外等待耗时 + 齐射等待耗时

每辆车的路径距离 = 主干道路径距离+普通道路路径距离

每辆车路径机动耗时 = 主干道路径距离 / 主干道行驶速度 + 普通道路路径距离 / 普通道路行驶速度

不同车型在主干道和普通道路行驶速度如表 1 所示：

表 1.

车载发射装置	普通道路	主干道
A	70km/h	45km/h
B	60km/h	35km/h
C	50km/h	30km/h

五、模型的建立与求解

5.1 问题一分析与求解

5.1.1 模型建立

使用甘特图表示整体暴露时间，甘特图又称横道图，条状图，其通过条状图来显示项目进度等随时间的变化的状态。横轴表示时刻，纵轴表示车载发射装置，条状色块表示车载发射装置的机动耗时和所处阵地位置。每种色块表示阵地上的一个位置，如位于道路节点处等待会车还是行驶于道路上。色块宽度表示车载发射装置停留在该位置的时间。见图 1

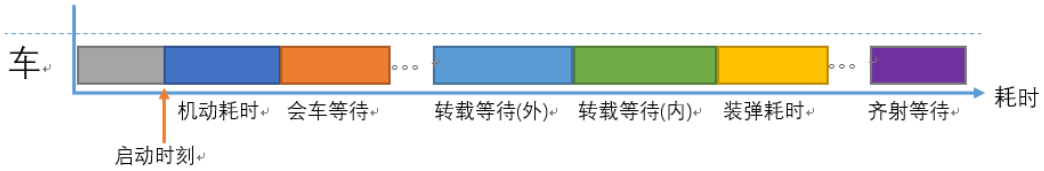

图 1：整体暴露时间

当色块高度为 1 时，色块的总面积表示整体暴露时间(需排除启动时刻前的色块，和表示转载地域的色块)。

已知各种车载发射装置在各种道路的行驶速度，从而得出车载发射装置在各道路的耗时情况如图 2 所示：

当甘特图上存在多个车载发射装置时，需合理设置会车等待色块以满足模型假设，即当道路发生阻塞或转载地域满员时，安排充足的等待时间色块。

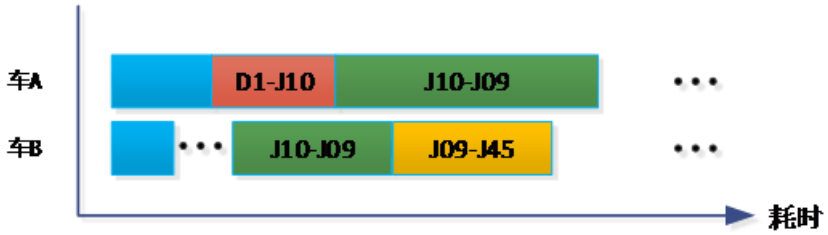

图 2：车载发射装置在各道路的耗时情况

如图中，J10,J45 表示道路节点，J10-J09 表示道路，当车 B 位于 J10-J09 时，车 A 也要驶入该路段，此时安排车 A 在 J10 处等待，直到车 B 驶离该路段。

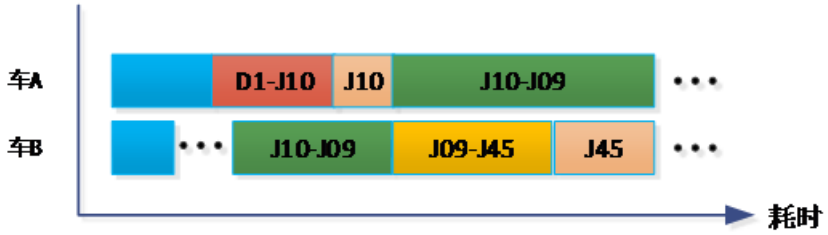

图 3

对于转载等待情况以此类推。当所有车辆满足模型约束时，即可获得当前机动路径规划下，发生的会车次数，转载等待次数，齐射等待情况，以及具体耗时。结合根据车速与阵地路网获得机动耗时，可以获得整体暴露时间值。

为了渲染甘特图，需要提供每台车载发射装置的机动路径规划与发车时刻。甘特图渲染算法结合各车载发射装置在各路况下的行驶速度，计算路径行车的耗时。

然后使用碰撞算法计算出每台车载发射装置的会车等待耗时，转载满员耗时，转载等待耗时，齐射等待耗时。渲染流程图如图 4 所示。碰撞算法见附录程序

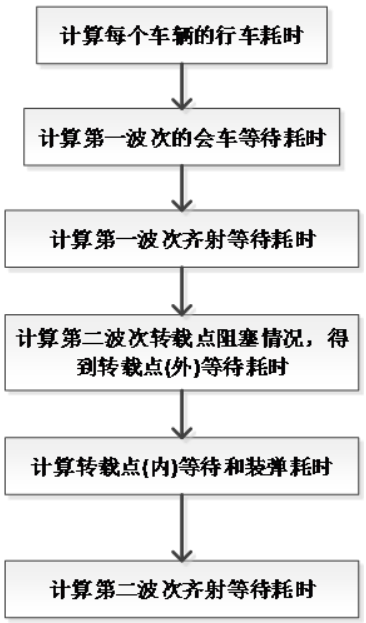

图 4：甘特图渲染算法流程图

齐射等待耗时计算，假设每辆车载发射装置自待机地域到达发射点位的总时间为 $T_i(i=1,2,...,24)$。获取最慢车的耗时结果为 T_{min}，其它每辆车的等待耗时等于每辆车的耗时减去最慢车的耗时 $T_q=\sum_{i=1,2,3,...,24}(T_i-T_{min})$，从而得到每辆车的齐射等待耗时。

由甘特图渲染算法，可以得出每种机动路径组合，除路径机动外产生的耗时。

5.1.2 问题求解

为了实现整体最小暴露时间，首先追求每辆车的机动耗时最小，即机动路径最短。机动路径可以看成是由一系列道路节点和地域组成的序列[3]，由于是两波次攻击任务，该序列又可进一步分成三个部分，即第一波次发射序列(D-F1)，转载序列(F1-Z)，第二波次发射序列(Z-F2)。使用 Dijkstra 算法分别求解三种类型序列的最短路径。Dijkstra 算法见附录程序

为了合理安排三种序列的组合，使得连接到每个待机地域与转载地域的发射点位间的路径，产生的会车等待次数最少，同时满足每个发射点位仅使用一次。根据发射点到各待机地域和各转载地域距离不同，将所有的发射点位分为两类，将每个发射点位到转载地域的最短路径进行排序，选出最短的 24 条，将对应的发射点位归为第二波次发射点位，其余归为第一波次发射点位。依据各自最近的转载地域，确定第二波次的机动路径。

根据到各待机地域间和各转载地域间的最短路径距离的排序，确定每个第一波次发射点到各待机地域和转载地域的机动路径。

由于前往不同发射点位的路径长度不同，每种车型的耗时也不同，需要合理安排发车次序与发车间隔，实现最短的齐射等待。采用如下策略确定发车次序和发车：

1） 长距离的车早发车
2） 慢车比快车早发车

5.2 问题二的求解
5.2.1 问题分析

转载地域为第二波次提供装弹服务，增设转载地域可以减少等待转载作业耗时，减轻其他转载地域的作业负荷。

5.2.2 模型建立

为了量化表示转载地域的服务能力和服务效率，用发射点位密度分布和潜在转载地域的节点度中心度 degree centrality 表示潜在转载地域区位状态。

转载地域周围发射点位密度大，则转载地域潜在能服务的对象多。发射点位密度热力图，如图 3 所示。

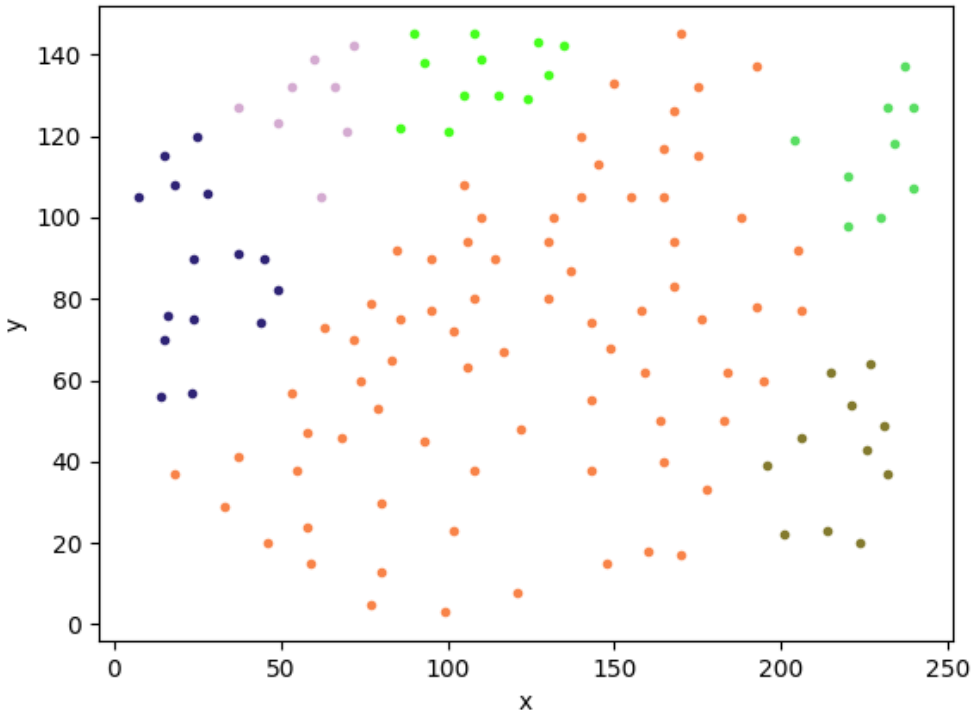

图 5 发射点位分布密度热力图

转载地域的度中心越大，则车载发射装置出入转载地域越方便，路径规划越方便。

潜在节点 J25、J34、J36、J42、J49 的度中心度如表 2 所示。

表 2.

节点	度中心度
J25	0.046511627906976744
J34	0.03875968992248062
J36	0.031007751937984496
J42	0.031007751937984496
J49	0.031007751937984496

求解 Max(度中心+密度值)，则 J25,J34 为最佳临时转载地域。

5.3 问题三的求解

5.3.1 问题分析及模型建立

当第一波次发射结束后，评估所有车载发射装置的暴露时间，将暴露时间最长的 3 台 C 类发射装置替换掉。

对所有第一波次使用过的发射点位进行记录，从而在第二波次发射任务分配时，排除这些发射点位。

因为第一波次使用了 24 个发射点位，则在第二波次可以分配的发射点位还剩下 36 个。

使用 Dijkstra 算法，计算各潜在隐蔽待机点 J04、J06、J08、J13、J14、J15，到各剩余发射点位的最短机动路径。选择路径长度最短且满足每一隐蔽待机点至多容纳 2 台发射装置约束条件的的三条路径。路径对应的隐蔽待机点即为最佳隐蔽待机点。

5.4 问题四的求解

5.4.1 模型建立

可以使用以下指标衡量，道路节点对于导弹发射任务影响程度：

1）K-means 聚类中心

K-means 算法计算节点周围路段数量的密度（F&J->F 点的权重较大，在计算得出的密度中心中选出距离 D1、D2 点最近的密度中心）。

由节点密度评价指标可以得出：道路节点周围路段密度较大的，说明发射车辆经过此节点的概率更高，受到攻击破坏后对导弹发射任务的影响也更大，从而其受到敌方攻击破坏的潜在可能更大；

2）Betweeness Centrality 值

BC 值（Betweeness Centrality）指标反映了道路节点的车流通量即经过频率，BC 值越大说明会有越多的发射车辆经过该节点，其受到攻击后对导弹发射的影响也较大。

5.4.2 基于 k-means 算法模型求解

从攻击方的角度选择攻击路网节点，使得受到攻击破坏的路网节点对防御方导弹发射影响最大，则一定会选取那些发射车辆经过道路节点最频繁的节点进行攻击破坏，那些路段密度较大的道路节点对周边道路的车辆流通影响较大，同时考虑到此处道路节点的密度，道路节点密度越小，则受到破坏后，绕路产生的额外机动耗时越大，将这一原则抽象成目标函数：

$$P = J_i * \log(1 + BC_i) \quad i = 1,2,...,k \qquad (1)$$

其中，J_i 表示路网节点 i 的密度(这里的道路节点包含了发射节点和交叉节点)，

BC_i 表示路网节点 i 的流通量。

现采用 K 均值聚类算法[7]和网路节点 BC 值的计算来得到每个道路节点潜在受攻击的评价指标。

K 均值聚类算法是 cluster analysis 的算法，其主要通过不断地取离种子点最近均值来求取数据中密度中心点。K 均值算法见附录程序

K 均值聚类算法解决该问题的过程为：

1）选取路网节点中发射节点和交叉节点$\{x^{(1)},...,x^{(122)}\}$，作为待聚类数据集$D$；

2）随机在数据集中选取k个节点作为起始质心μ_j；

3）对数据集中所有数据节点进行遍历

4）计算每个数据节点与k个质心μ_j的距离dis_{ij}，选取距离最近的质心μ_j作为数据节点D_i的簇类别；

$$c^{(i)} := \arg\min_j \| x^{(i)} - \mu_j \|^2 \qquad (2)$$

5）每一轮遍历后，从每一类簇中按照已归类的数据集重新选取聚类中心点μ_j；

$$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}} \qquad (3)$$

6）直到数据节点的簇分类不再变化，停止迭代。

7）得到k个聚类中心道路节点。

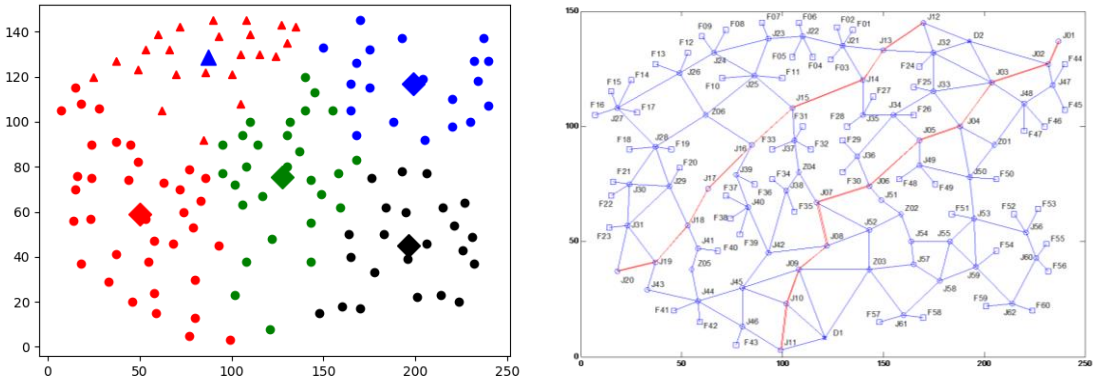

图 6：路网密度图与原图对比

选取路网节点中发射节点和交叉节点作为待聚类数据集 D
随机在数据集中选取 K 个节点作为起始质心 Rent
当任意一个点的簇分配结果发生改变时：
对数据集中的每个节点
对每个质心
计算质心与数据点之间的距离
将数据点分配到距离其最近的簇
对每一个簇计算簇中所有点的均值并将均值作为质心

图 7：基于 k-means 算法流程

BC 值指标是表示所有节点对之间通过该节点的最短路径条数。BC 值很好地描述了一个网络中节点可能需要承载的流量。一个节点的 BC 值越大，流经它的数据分组越多。

$$C_B(v_i)=\sum_{vs\neq vi\neq vt\in V,s<t}\frac{\sigma_{st}(v_i)}{\sigma_{st}} \qquad (4)$$

σ_{st}：指的是 S 到 t 的最短距离，$\sigma_{st}(v_i)$：指的是 s 通过 v_i 到 t 的最短路径的数目。

直观上来说， betweenness 反映了节点 v_i 作为“桥梁”的重要程度。

$$\sigma_{st}(v)=\begin{cases}\sigma_{st}.\sigma_{vt} & d(s,v)+d(v,t)=d(s,t)\\ 0 & other\end{cases} \qquad (5)$$

对网路节点聚类选取 5 个聚类中心节点 $J18$、$J53$、$J25$、$J07$、$J03$，利用 dijktra 算法计算出路网节点间道路距离，得到中心节点分别与 $D1$、$D2$ 节点的最短道路距离（km）：

$$D1\_J18\approx 129; D1\_J07\approx 83; D1\_J03\approx 187; D1\_J53\approx 104; D1\_J25\approx 150.$$

$$D2\_J18\approx 175; D2\_J07\approx 126; D2\_J03\approx 21; D2\_J53\approx 87; D2\_J25\approx 125.$$

路网聚类中心节点密度值（节点个数）：

$$J18=30; J07=33; J03=24; J53=23; J25=20.$$

路网聚类中心节点 BC 值（百分比）：

$$BC_{J03}=0.10531649\ 816661783;$$

$$BC_{J07}=0.16348247\ 371594035;$$

$$BC_{J18}=0.06888039\ 599703841;$$

$$BC_{J25}=0.16051687\ 25931167;$$

$$BC_{J53}=0.13564989\ 27810701;$$

最终评价指标值 P：

$$P_{J03}=2.40316122\ 33235675;$$

$$P_{J07}=4.99678212\ 7627859;$$

$$P_{J18}=1.99835225\ 33074047;$$

$$P_{J25} = 2.97730971\ 39743148;$$

$$P_{J53} = 2.92571683\ 59367003;$$

最后选取最优的三个道路节点：$J07$、$J25$、$J53$，这些节点的道路节点密度相对较高，距离待机节点$D1, D2$距离较近，且其BC值也较大，综合各评价指标得出这三个节点潜在受到攻击的可能性更大。因而防御方在进行路径规划时，应尽量避开这三个节点。

5.5 问题五的求解

5.5.1 问题分析

本问题主要有两方面需要考虑，一是分析路网各节点的流通度和分布密度，对那些流通度和分布密度高的道路节点减少发射车辆通过次数，以避免遭受敌方导弹的攻击破坏；二是在适当分散机动策略下尽可能的缩短单台发射装置的最长暴露时间。

5.5.2 模型建立

首先使用介数中心性(Betweenness Centrality)模型对路网节点的联通度进行分析，获得其节点分布的 BC 值，节点 BC 值越大说明在路网中该节点流通度很高，容易受到敌方攻击破坏，而我们要做的就是对那些 BC 值高的道路节点减少发射装置通过的次数，来达到分散机动的策略。

基于分散机动策略下按照问题一的模型对发射装置进行调度分配，使得每台发射装置的暴露时间尽可能短。

利用路网节点 BC 值反映道路流通程度，对路网中每条道路按两边节点的 BC 值赋予权重，使得被赋予权重的路网中道路流通度越高的边，其长度变得越长，需要的时间也越久；流通度低的道路，其长度相对变得越短，需要的时间也相对的越少。需要注意的是，利用加权后的路网在问题一模型中进行调度分配时，得出的时间是由经过的每个加权的道路里程数计算得出的，并不是真是道路中行驶的耗时。要得到任务中真实的时间消耗情况，只需要对每辆发射装置所行驶的路径节点间道路里程除以权重，即可得到真实的时间消耗。

流程图：

1）对路网道路节点间里程进行加权：

对路网节点间道路加权的公式如下：

$$\tilde{L}_{ij} = e^{(BC_i + BC_j)} * L_{ij} \qquad (6)$$

$\tilde{L}_{ij}$是加权后的节点i和j之间的里程数，BC_i是节点i的BC值，BC_j是节点j的BC值，L_{ij}是原始的节点i和j之间的里程数。

2）利用问题一中建立的模型进行发射装置调度分配，使得在分散机动策略下尽可能的缩短单台发射装置的暴露时间。

3）将得出的每台发射装置经过加权后的暴露时间，还原成每台发射装置真实的暴露时间。

六、模型的评价与改进

本文所用的 Dijkstra 算法有效解决了附权有向图中的最短路问题，适用于大数据的处理。利用节点密度和节点 BC 值相融合得到的 *P* 值指标能很好的反映了路网道路的流通度和重要程度。利用路网节点 BC 值对道路里程进行加权能够有效的分散高流通节点的发射装置通过次数，具有很好的分散机动性。

同时，本文所建立的模型还有很多改进的地方，比如说：可以以待机地域为根，建立最短路径树，依据树上各节点途径的发射点位种类更快速的安排发车次序和发车间隔。

七、参考文献

[1] 金宏，余跃，张如飞．常规导弹联合火力打击统一分配模型[J]．火力与指挥控制，2014(7):27-30.

[2] 汪民乐，房茂燕．基于改进单亲GA的多波次攻击最优火力分配[J]．系统仿真学报，2009，21(23):7697-7699.

[3] 吴瑞峰,蒋欢欢,张占美．导弹火力打击任务分配[J]．中国高新区,2017,(11):37-38+40.［2017-09-19].

[4] 季青梅，辛文芳．多波次导弹火力打击任务研究[J]．信息技术与信息化，2017(z1):122-128.

[5] 王桐，杨萍，欧阳海波．基于马尔可夫链的多波次导弹作战研究[J]．战术导弹技术，2011(4):20-22.

[6] 白军，刘新学，郭峰,等．基于遗传算法的导弹编队攻击任务优化分配策略[J]．兵器装备工程学报，2011，32(10):51-53.

[7] 原福永，张晓彩，罗思标．基于信息熵的精确属性赋权K-means聚类算法[J]．计算机应用，2011，31(6):1675-1677.

八、附录

8.1 甘特图渲染碰撞算法：使用 python 基于 networkX，pandas 编写

```
def stage1Meeting(stage1):
    #终止计算条件-到达发射点了
    #同时过滤已经到发射点的车
    should,st=[],{}
    for car,state in stage1.items():
        a=state["D-F 时序"]["位置"].iloc[state["当前位置"]+1]
        b=state["发射点"]
        if a!=b:
            should.append(car)
    if len(should)==0:
        return stage1
    else:
    #装填过滤的车
```

```
        for i in should:
            st[i]=stage1[i]

    #耗时计算
    t=[(car,state["出发时刻"]+sum(state["D-F 时序"]['耗时'].iloc[1:state["当前位置"]+1])) for car,state in st.items()]
    tdic={car:state["出发时刻"]+sum(state["D-F 时序"]['耗时'].iloc[1:state["当前位置"]+1]) for car,state in st.items()}

    #t=sorted(t, key=lambda item: item[1])
    #fastCar,fastTime=t[0][0],t[0][1]
    tz=min(t, key=lambda item: item[1])
    fastCar,fastTime=tz[0],tz[1]

    #时间线比较
    tt=st[fastCar]["D-F 时序"].iloc[st[fastCar]['当前位置']+2]
    if tt['类型'] != 'edge':
        print("出错了：001")
        return stage1
    ttt=tt["位置"]
    roadSize=G.edge[ttt[0]][ttt[1]]['容量']
    #时间线比较-之重叠
    meetingCars=[]
    for car,state in st.items():
        loc=state["D-F 时序"]['位置'].iloc[state["当前位置"]]
        if set(ttt)==set(loc):
            meetingCars.append(car)

    #计算其是否在区间时间范围内
    amtz=[]
    for car in meetingCars:
        a=sum(stage1[car]["D-F 时序"]['耗时'].iloc[1:stage1[car]["当前位置"]+1])
        b=sum(stage1[car]["D-F 时序"]['耗时'].iloc[1:stage1[car]["当前位置"]])

        if a <= fastTime and fastTime < b:
            amtz.append(car)
    meetingCars=amtz

    if len(meetingCars)>=roadSize:
        #说明需要会车等待
        if len(meetingCars)>roadSize:
            print(meetingCars)
            print("出错了：002")
            return stage1
```

```
        #To-do 会车等待决策+1
        waitTime = min([tdic[i] for i in meetingCars]) - fastTime
        #写入等待时间
        stage1[fastCar]["D-F 时序"].loc[stage1[fastCar]['当前位置']+1,'耗时']=waitTime
    stage1[fastCar]['当前位置']=stage1[fastCar]['当前位置']+2

    return stage1Meeting(stage1)

    def stage2Meeting(stage2):
        #终止条件-齐射
        should,st=[],{}
        for car,state in stage2.items():
            a=state["Z-F 时序"]["位置"].iloc[state["当前 Z-F 位置"]+1]
            b=state["发射点"]
            if a!=b:
                should.append(car)
        if len(should)==0:
            return stage2

        #过滤到达终点的
        #装填过滤的车
        for i in should:
            st[i]=stage2[i]

        #重叠检测
        #耗时计算
        t,tdic=[],{}
        for car,state in st.items():
            if state["转载完成"]:
                h=sum(state["Z-F 时序"]['耗时'].iloc[1:state["当前 Z-F 位置"]+1])+sum(state["F-Z 时序"]['耗时'].iloc[1:state["当前 F-Z 位置"]+1])
            else:
                h=sum(state["F-Z 时序"]['耗时'].iloc[1:state["当前 F-Z 位置"]+1])
            t.append((car,h))
            tdic[car]=h

        #t=sorted(t, key=lambda item: item[1])
        #fastCar,fastTime=t[0][0],t[0][1]
        tz=min(t, key=lambda item: item[1])
        fastCar,fastTime=tz[0],tz[1]

        #时间线比较
        if st[fastCar]["转载完成"]:
            tt=st[fastCar]["Z-F 时序"].iloc[st[fastCar]['当前 Z-F 位置']+2]
```

```
                if tt['类型'] != 'edge':
                    print("出错了：003")
                    return stage2
                ttt=tt["位置"]
                roadSize=G.edge[ttt[0]][ttt[1]]['容量']
                #时间线比较-之重叠
                meetingCars=[]
                for car,state in st.items():
                    if state["转载完成"]:
                        loc=state["Z-F 时序"]['位置'].iloc[state["当前 Z-F 位置"]]
                    else:
                        loc=state["F-Z 时序"]['位置'].iloc[state["当前 F-Z 位置"]]
                    if set(ttt)==set(loc):
                        meetingCars.append(car)

                #计算其是否在区间时间范围内
                amtz=[]
                for car in meetingCars:
                    if state["转载完成"]:
                        a=sum(stage2[car]["Z-F 时序"]['耗时'].iloc[1:stage2[car]["当前 Z-F 位置
"]+1])+sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:stage2[car]["当前 F-Z 位置"]+1])
                        b=sum(stage2[car]["Z-F 时序"]['耗时'].iloc[1:stage2[car]["当前 Z-F 位置
"]])+sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:stage2[car]["当前 F-Z 位置"]+1])
                    else:
                        a=sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:stage2[car]["当前 F-Z 位置
"]+1])
                        b=sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:stage2[car]["当前 F-Z 位置
"]])

                    if a <= fastTime and fastTime < b:
                        amtz.append(car)
                meetingCars=amtz

                if len(meetingCars)>=roadSize:
                    #说明需要会车等待
                    if len(meetingCars)>roadSize:
                        print("出错了：004")
                        return stage2
                    #To-do 会车等待决策+1
                    waitTime = min([tdic[i] for i in meetingCars]) - fastTime
                    #写入等待时间
                    stage2[fastCar]["Z-F 时序"].loc[stage2[fastCar]['当前 Z-F 位置']+1,'耗时
']=waitTime
                stage2[fastCar]['当前 Z-F 位置']=stage2[fastCar]['当前 Z-F 位置']+2
```

```
        else:
            tt=st[fastCar]["F-Z 时序"].iloc[st[fastCar]['当前 F-Z 位置']+2]
            if tt['类型'] != 'edge':
                print("出错了：005")
                return stage2
            ttt=tt["位置"]
            roadSize=G.edge[ttt[0]][ttt[1]]['容量']
            #时间线比较-之重叠
            meetingCars=[]
            for car,state in st.items():
                if state["转载完成"]:
                    loc=state["Z-F 时序"]['位置'].iloc[state["当前 Z-F 位置"]]
                else:
                    loc=state["F-Z 时序"]['位置'].iloc[state["当前 F-Z 位置"]]
                if set(ttt)==set(loc):
                    meetingCars.append(car)
            #计算其是否在区间时间范围内
            amtz=[]
            for car in meetingCars:
                if state["转载完成"]:
                    a=sum(stage2[car]["Z-F 时序"]['耗时'].iloc[1:stage2[car]["当前 Z-F 位置
"]+1])+sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:stage2[car]["当前 F-Z 位置"]+1])
                    b=sum(stage2[car]["Z-F 时序"]['耗时'].iloc[1:stage2[car]["当前 Z-F 位置
"]])+sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:stage2[car]["当前 F-Z 位置"]+1])
                else:
                    a=sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:stage2[car]["当前 F-Z 位置
"]+1])
                    b=sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:stage2[car]["当前 F-Z 位置
"]])

                if a <= fastTime and fastTime < b:
                    amtz.append(car)
            meetingCars=amtz
            if len(meetingCars)>=roadSize:
                #说明需要会车等待
                if len(meetingCars)>roadSize:
                    print(meetingCars)
                    print("出错了：006")
                    return stage2
                #To-do 会车等待决策+1
                waitTime = min([tdic[i] for i in meetingCars]) - fastTime
                #写入等待时间
                stage2[fastCar]["F-Z 时序"].loc[stage2[fastCar]['当前 F-Z 位置']+1,'耗时
']=waitTime
```

```
            #时间线比较-之转载爆仓
            zt=st[fastCar]["F-Z 时序"].iloc[st[fastCar]['当前 F-Z 位置']+3]
            if zt['类型'] != 'node':
                print("出错了：007")
                return stage2
            if zt['位置'] == st[fastCar]['转载点']:
                #检测是否爆仓
                #所有是这个转载点的车
                #且到过转载点的
                stz=[car for car,state in stage2.items() if state["转载完成"] and state['转载点
']==zt['位置']]
                ztTime=sum(st[fastCar]["F-Z 时序"]['耗时'].iloc[1:st[fastCar]['当前 F-Z 位置
']+2])
                #计算其是否在区间时间范围内
                stzz,mtz=[],[]
                for car in stz:
                    a=sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:-1])
                    b=sum(stage2[car]["F-Z 时序"]['耗时'].iloc[1:])
                    if a <= ztTime and ztTime < b:
                        mtz.append(b)
                        stzz.append(car)
                #基础转载时间
                zz=10
                if len(stzz) >= 2:
                    if len(stzz) > 2:
                        print("报错：特殊")
                    #说明爆仓
                    waitt=min(mtz)-ztTime
                    stage2[fastCar]["F-Z 时序"].loc[stage2[fastCar]['当前 F-Z 位置']+1,'耗时
']=waitt
                    #计算新的到达时间
                    ztt=sum(st[fastCar]["F-Z 时序"]['耗时'].iloc[1:st[fastCar]['当前 F-Z 位置
']+2])
                    zz+=max(mtz)-ztt
                #未爆仓，进场计算转载耗时
                if len(stzz) ==1:
                    #表示存在正在转载物
                    zz+=sum(stage2[stzz[0]]["F-Z 时序"]['耗时'].iloc[1:])-ztTime
                #并写入转载完成和位置，转载时间
                stage2[fastCar]['转载完成']=True
                stage2[fastCar]['当前 F-Z 位置']=stage2[fastCar]['当前 F-Z 位置']+3
                stage2[fastCar]["F-Z 时序"].loc[stage2[fastCar]['当前 F-Z 位置'],'耗时']=zz
            stage2[fastCar]['当前 F-Z 位置']=stage2[fastCar]['当前 F-Z 位置']+2
```

```
        return stage2Meeting(stage2)



    '''
    目前权值是以小时为单位的，所以最终结果需要转为分钟 即 * 60
    d_f:表示所有 24 辆车的路径
    {车号:path}
    departure:表示每辆车的发车时间，出发时刻，分钟为单位
    {车号:时刻}
    '''
    #波次 1 的耗时计算
    def stage1Consum(d_f,departure):
        stage1={}
        for car,path in d_f.items():
            stage1[car]={
                "车型":car[0],
                "D-F 时序":generDF(path,car[0]),
                "发射点":path[-1],
                "出发时刻":departure[car],
                "当前位置":1
            }
        #To-do:发车次序纠错
        #重叠计算
        newst1=stage1Meeting(stage1)#此处为范式，由于是可变式，所以 newst1 还是和原来的引用一样
        #齐射罚时
        newst1=alignLaunch(newst1)

        return newst1

    '''
    f_z:{车号:path}
    z_f:{车号:path}
    '''
    def stage2Consum(f_z,z_f):
        stage2={}
        for car in f_z.keys():
            stage2[car]={
                "车型":car[0],
                "F-Z 时序":generFZ(f_z[car],car[0]),
                "当前 F-Z 位置":1,
                "转载点":f_z[car][-1],
                "转载完成":False,
```

```
                "Z-F 时序":generZF(z_f[car],car[0]),
                "当前 Z-F 位置":1,
                "发射点":z_f[car][-1]
            }
        #重叠计算-转载计算-转载爆仓问题？->中断所有到此 Z 的车辆
        newst2=stage2Meeting(stage2)#此处为范式，由于是可变式，所以 newst1 还是和原来的引用一样
        #齐射罚时
        newst2=alignLaunch2(newst2)

        return newst2
```

8.2 dijkstra 算法：使用 MATLAB 编写

```
function[L,Z]=dijkstra(W,S,T)
N=length(W(:,1));%顶点数
W(find(W==0))=inf;
L=Inf*ones(1,N);
L(S)=0;
C=S;
Q = 1:N;  % 未走访的顶点集
Z = S*ones(1,N);
Z(S)=0;
for K = 1:N  %更新 L 和 Z 的循环
    Q = setdiff(Q,C);
    [L(Q),ind] = min([L(Q),L(C)+W(C,Q)]);
    Z(Q(find(ind==2))) = C;
    if T&C == T
        L = L(T); % 最短路径长度
        road = T; % 最短路径终点
        while T~=S % 追溯最短路径上的点
            T=Z(T); % 从终点往前寻找其父亲结点
            road = [road,T];
        end
        Z=road(length(road):-1:1); % 颠倒次序
        return;
    end
    [null,mC]=min(L(Q));
    if null == Inf
        % disp(' 到值是 Inf 的点的路不通！ ');
        Z(find(L==Inf))==nan;
        return;
    else
        C=Q(mC);
        C;
    end
```

```
end
End
```

8.3 k-means 算法：使用 python 编写

```
#计算两个向量的距离，用的是欧几里得距离
def distEclud(vecA, vecB):
    return sqrt(sum(power(vecA - vecB, 2)))

#随机生成初始的质心（ng 的课说的初始方式是随机选 K 个点）
def randCent(dataSet, k):
    n = shape(dataSet)[1]
    centroids = mat(zeros((k,n)))
    for j in range(n):
        minJ = min(dataSet[:,j])
        rangeJ = float(max(array(dataSet)[:,j]) - minJ)
        centroids[:,j] = minJ + rangeJ * random.rand(k,1)
    return centroids

def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):
    m = shape(dataSet)[0]
    clusterAssment = mat(zeros((m,2)))#create mat to assign data points
                                      #to a centroid, also holds SE of each point
    centroids = createCent(dataSet, k)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):#for each data point assign it to the closest centroid
            minDist = inf
            minIndex = -1
            for j in range(k):
                distJI = distMeas(centroids[j,:],dataSet[i,:])
                if distJI < minDist:
                    minDist = distJI; minIndex = j
            if clusterAssment[i,0] != minIndex:
                clusterChanged = True
            clusterAssment[i,:] = minIndex,minDist**2
        print centroids
        for cent in range(k):#recalculate centroids
            ptsInClust = dataSet[nonzero(clusterAssment[:,0].A==cent)[0]]#get all the point
in this cluster
            centroids[cent,:] = mean(ptsInClust, axis=0) #assign centroid to mean
    return centroids, clusterAssment
```