基于计算机仿真的中心设施选址问题研究

摘要

选址问题是组合优化领域中的一类重要问题，它是对于一些网络服务器、核电站或者物流中心等有限且重要的资源进行选址决策，在生产管理与调度，网络通信，理论计算机科学等方面有广泛的应用。本文就紧急医疗救助中心选址问题展开研究。

对于问题一，根据已知数据求出当前中心设施的平均响应时间。首先，对数据进行预处理，将时间错位的请求改正过来。然后利用**Dijkstra算法**求出中心设施到各个位置的最短路径。再以事件为步长利用计算机仿真模拟整个的呼叫请求处理过程并求出平均响应时间。因为假设救助的服务时间服从均值为15方差为1的正态分布，所以对数据进行20次仿真模拟，然后求平均得到区域16左下角作为中心设施、救助队行驶速度为72km/h时的平均响应时间为148.3分钟。

对于问题二，首先分析并得出通过改变中心设施位置，可能可以降低平均响应时间的结论。然后再使用第一问的Dijkstra算法+**计算机仿真模拟算法**对47个区域分别计算其作为中心设施时的平均响应时间（模拟20次取平均的结果见表2）。从结果中我们知道当区域26作为中心设施、救助队行驶速度为72km/h时的最短平均响应时间为133.1分钟。

对于问题三，首先分析证明中心设施确定时通过增加救助队的数量，将有可能降低平均响应时间并使其控制在5分钟以内的结论。然后通过不断修改第一问的计算机仿真模拟算法中的救助队数量并分别求得其相应的平均响应时间。最后绘制出救助队数量在3到10之间的平均响应时间变化情况并得出当救助队数量增加到8时平均响应时间下降到5分钟。

对于问题四，在考虑交通拥堵的情况下，确定中心设施的最佳位置。首先假设高峰期时经过城市中心路段车速为正常时的一半即为36km/h。然后用Dijkstra算法求出高峰期与非高峰期时中心设施到各区域的最短路径，再修改计算机仿真模拟算法，使时间处于不同时期时使用不同的路径信息模拟整个的呼叫处理过程。最后类比问题二遍历区域1到47分别得出其作为中心设施时的平均响应时间（模拟20次取平均的结果见表5）。发现还是当区域26作为中心设施时平均响应时间最短其值为158.4分钟。

最后，本文对以上模型进行了评价与推广。

关键词： Dijkstra 算法　　计算机仿真　　网络优化　　选址问题

1 问题的背景与重述

无论在何种社会环境和历史条件下，人们都离不开设施选址问题。选址决策对于人们的生产生活，社会组织都有着重要影响。本文就建设紧急医疗救助问题展开研究，并解决以下问题：

1．对于紧急医疗救助，平均响应时间是重要的，它指的是，从一个呼叫产生到救助队赶到该呼叫地点所用的平均时长。请根据附件数据，计算当前（中心设施位置如题所述）的平均响应时间。

2．是否可以通过改变中心设施位置，降低平均响应时间？最小值为多少？此时中心设施应建在何处？

3．高效的医疗救助应将平均响应时间控制在 5 分钟以内。那么是否需要增加救助队的数量？应该增加多少？

4．地理简图中的灰色区域代表中心城区，假设在一天中的 11:00-13:00 和 17:00-19:00 时段为交通高峰期，此时有大量车辆进出中心城区，那么在考虑交通拥堵情况下，解决以上问题，还需要知道哪些参数？请设定合理的参数，并在考虑交通拥堵情况下，研究中心设施的最佳位置。

2 模型的假设

简化分析起见，做了如下假设：

(1)各区域的出入口在左下角。

(2)相邻两个交叉路口之间的道路视为直线。

(3)救助队的出行道路均是双行道。

(4)救助队每天出行的平均速度恒定。

(5)救助队出行过程中无意外事故发生。

(6)不考虑任务派遣调度过程花费的时间。

(7)当救助队接到任务后,将选择最近路线前往案发现场。

(8)一次救助所花费的时间服从均值为 15min，方差为 1min^2 正态分布。

(9)第一天开始时三个救助队均为空闲状况。

3 符号的说明

为简化对问题的分析和对数字的处理,我们在以后的文字中将使用如下的符号代表变量：

符号	含义
t_i	第 i 次呼叫产生时刻
L_{pi}	中心设施 p 到第 i 区域的距离
L'_{pi}	交通高峰期时中心设施 p 到第 i 区域的距离
v	救助队运行速度
T_i	救助队从中心设施到达事发地区域 i 的时间
S_i	接到 i 任务时，三个救助队中最早完成任务回到控制中心时刻
E_k	第 k 个救助队完成 j 任务并返回到控制中心的时刻
R_p	区域 p 作为中心设施时每次任务的响应时间
$\overline{R}_p$	以 p 为中心设施时的平均响应时间

n	事件的总数

*其他符号说明详见正文

4 模型的建立与求解

4.1 问题一

4.1.1 数据预处理

通过观察已知呼叫时间数据，发现其中有多个相邻时间呼叫地点相同的情况。本文把此种情况分析成由于等待时间过长造成的同一事故发出的多次呼叫请求。这样的数据将使在实际计算平均响应时间时产生重复计算的误差影响实验结果的准确性。因此，在解决以下问题之前应先做预处理。本文通过使用 Excel 只保留其相邻时间的第一个请求。相同地点的相邻时间数据删除。

4.1.2 问题的分析

由于题目没有明确指出救助队需到达区域的具体位置，所以我们统一假设当救助队到达每个区域的左下角的时候算是抵达了该区域。区域 16 作为救助中心时，我们首先利用 Dijkstra 算法求出区域 16 到各个区域的最短路径。再以事件为步长利用计算机仿真模拟整个的呼叫请求处理过程并求出平均响应时间。因为我们假设救助的服务时间服从均值为 15 方差为 1 的正态分布，所以我们对数据进行 20 次仿真模拟然后取均值作为其的平均响应时间。简化后的城市地图如下：

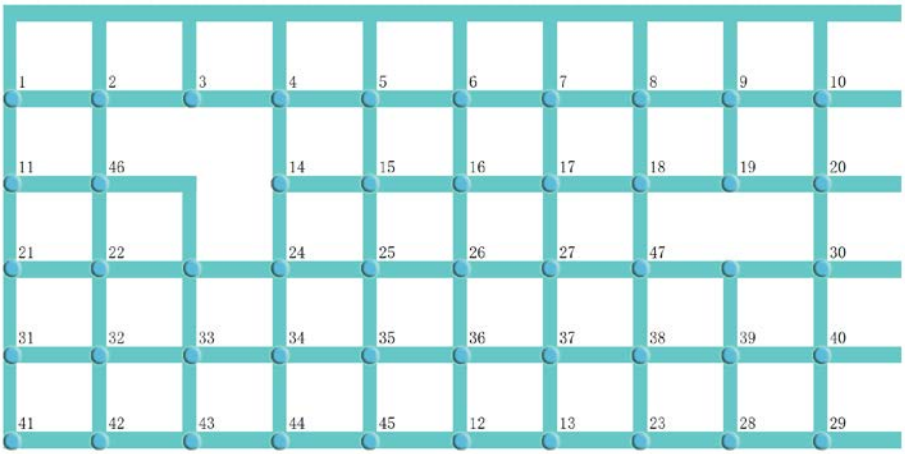

图 1 城市地理简图

4.1.3 模型的建立

4.1.3.1 规划中心设施到各区域最短路径的 Dijkstra 算法

在无向图 $G=\{V,E\}$ 中，求 V_0 到其他各顶点的最短路径

算法步骤如下：

1. 初始时令 $S=\{V_0\}$, $T=V-S=\{$其余顶点$\}$， T 中顶点对应的距离值

若存在 $<V_0,V_i>$， $d(V_0,V_i)$ 为 $<V_0,V_i>$ 弧上的权值

若不存在 $<V_0,V_i>$， $d(V_0,V_i)$ 为 ∞

2. 从集合 T 中选取一个与 S 中顶点有关联边且权值最小的顶点 W，加入到 S 中并记其 V_0 到 W 的最短路径长度为 $dist(W)$。

3. 对其余T中顶点的距离值进行修改：若当W作中间顶点时，从V_0到V_i的距离值即$dist(W)+d(W,V_0)$的长度缩短，则修改此距离值。

重复上述步骤 2、3，直到S中包含所有顶点，即$W=V_i$为止。

4.1.3.2 建立计算机仿真模型

设t_i为第i次呼叫的产生时刻，急救车从控制中心到达呼叫地所花时间T_i为$\frac{L_i}{v}$，其中L_i表示从控制中心到事发地的最短距离，v为急救车的运行速度。

① 当$i\le 3$时，在接到呼叫的第一时间就能派出救助队，每次呼叫的响应时间等于前往事发地的路途中所花费的时间R为T_i。

② 当$i\ge 4$时，需考虑产生呼叫时刻救助中心是否有空闲的救助队。

救助队是否空闲体现在呼叫产生时刻t_i与三个救助队中最早完成任务的救助队回到救助中心时刻S_i的时间前后关系。$S_i=\min(E_k)$，E_k表示第k个救助队完成呼叫任务j并返回到控制中心的时刻，$E_k=S_j+2T_j+F_j$，F_j为第j次救助的服务时间$F_j\sim N(15,1)$。a表示救助队是否空闲，则$a=\begin{cases}0, S_i>t_i\\1, S_i<t_i\end{cases}$，$a=0$时，表示救助中心没有空闲救助队；$a=1$时，有空闲救助队。

当$a=0$时，所以救助队都正在出勤，此时响应时间包括前往事发地的路途中所花费的时间T_i和等待急救车返回控制中心的时间S_i-t_i；$a=1$时，响应时间等于在前往事发地的路途中所花费的时间T_i。综合考虑得到响应时间的数学表达式为$R=T_i+(1-a)(S_i-t_i)$。

综合①②得到每一次呼叫的响应时间$R=\begin{cases}T_i, i\le 3\\T_i+(1-a)(S_i-t_i), i\ge 4\end{cases}$，$i$为正整数。

n次任务的平均响应时间$\overline{R}=\sum_{i=1}^{n}R\Big/n$。

下来仿真的数学模型如下：

$$\overline{R}_p=\sum_{i=1}^{n}R_p\Big/n \qquad p=16 \tag{1}$$

$$R_p=\begin{cases}T_i, i\le 3\\T_i+(1-a)(S_i-t_i), i\ge 4\end{cases} \qquad p=16 \tag{2}$$

$$a=\begin{cases}0, S_i>t_i\\1, S_i<t_i\end{cases} \tag{3}$$

$$E_k=S_j+2T_j+F_j \tag{4}$$

(5)

$$S_i = \min(E_k)$$

$$T_i = \frac{L_{pi}}{v} \qquad p = 16 \tag{6}$$

计算机模拟仿真流程图如下：

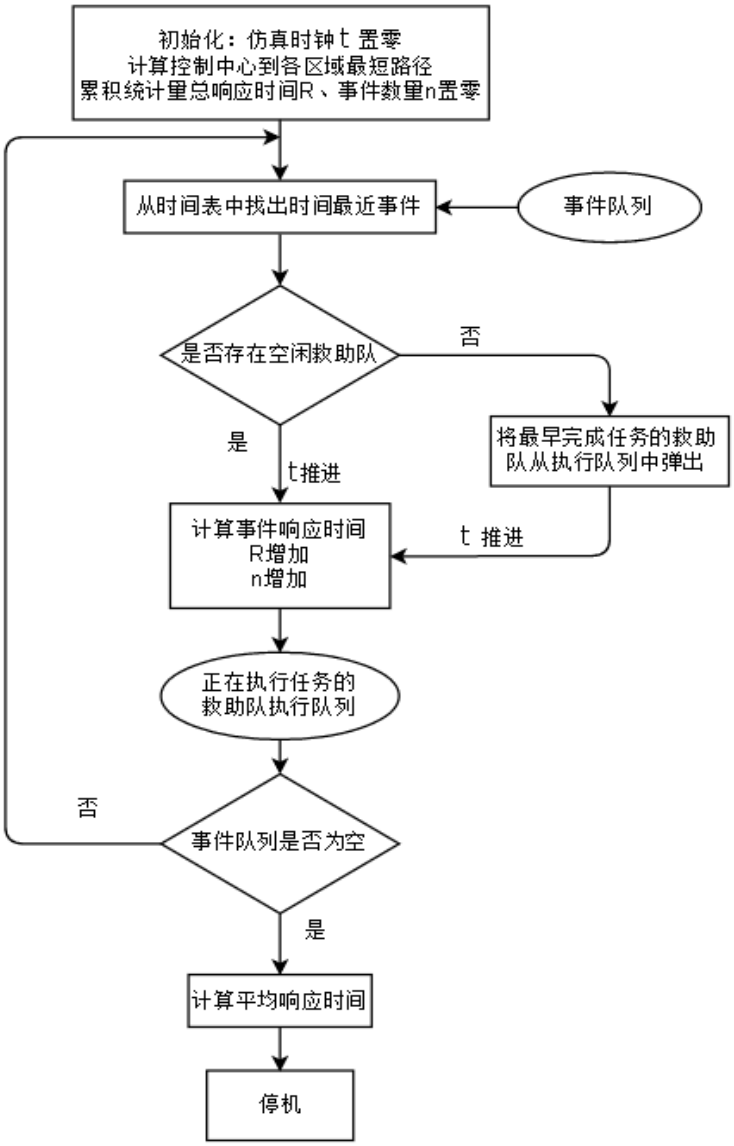

图 2 问题分析流程图

4.1.4 结果分析

根据模型编写时间复杂度为$O(n+k^2)$的 C++程序（其中 n 为呼叫总数，k 为区域总数），在 vs2017 平台下运行 20 次取平均，得到结果如下表：

表 1 各天平均响应时间

天数	第 1 天	第 2 天	第 3 天	第 4 天	第 5 天	第 6 天	第 7 天	第 8 天
平均响应	302.5	35.1	121.4	253.3	108.9	101.1	98.3	166.1

时间(min)

对以上 8 天的各天的平均响应时间求均值得出将区域 16 左下角作为中心设施、救助队行驶速度为 72km/h 时的总的平均响应时间为 148.3 分钟，即自呼叫求救起大约两个半小时左右救助队才会到达，时间过于长，待实施改进措施缩短平均响应时间。

4.2 问题二

4.2.1 问题的分析

4.2.1.1 对改变中心位置使平均响应时间降低的可行性分析

$\because \overline{R}=\sum_{i=1}^{n}R\Big/n$，要求平均响应时间$\overline{R}$最小，任务量$n$为确定值

$\therefore \sum_{i=1}^{n}R_i$ 要最小

又$\because R_i=\begin{cases}T_i & ,i\le 3\\ T_i+(1-a)(S_i-t_i) & ,i\ge 4\end{cases}$

$\therefore$① 当$i\le 3$时，要求T_i最小，

$\because T_i=\frac{L_i}{v}$，$v$表示速度为常数

$\therefore L_i$要最小

$\because L_i$是控制中心到事发地的最短距离

$\therefore$可以通过改变中心设施位置从而使各呼叫地点到中心的距离改变并总的路程降低，从而实现降低平均响应时间的要求

② 当$i\ge 4$时，要求T_i和S_i-t_i的和最小，所以可以要求T_i和S_i-t_i分别最小，T_i最小的分析同①。

$\because t_i$表示第i次呼叫的时刻，为确定值。

$\therefore$要求S_i最小

$\because S_i=\min(E_k)$

$\therefore E_k$要最小

$\because E_k=S_j+2T_j+F_j$，$F_j$表示行一次救助所花费的平均时间，为确定值

$\therefore T_i$要最小，分析同①

$\therefore$可以通过改变中心设施位置从而使各呼叫地点到中心的距离改变并总的路程降低，降低平均响应时间

综上所述，可以通过尝试改变中心设施位置，降低平均响应时间。

4.2.1.2 最佳中心设施的选取分析

基于上面考虑改变中心设施对降低平均响应时间的分析，我们可以使用问题一中的以某个固定位置为中心设施求解平均响应时间的程序再计算47个区域分别作为中心设施时的平均响应时间。然后再选择平均响应时间最短的区域作为中心设施。

4.2.2 模型的建立

以区域 p 作为中心设施时总共 n 次任务的平均响应时间为 $\overline{R}_p$，我们从1到46遍历 p 分别计算其的平均响应时间 $\overline{R}_p$，$\overline{R}_p$ 最小的即为最适合作为中心设施的区域。

综合以上分析与问题一的模型可建问题二模型如下：

目标：$\min = \overline{R}_p$

$$\overline{R}_p = \sum_{i=1}^{n} R_p \Big/ n \tag{7}$$

$$R_p = \begin{cases} T_i \ , i \le 3 \\ T_i + (1-a)(S_i - t_i) \ , i \ge 4 \end{cases} \tag{8}$$

$$T_i = \frac{L_{pi}}{v} \tag{9}$$

$p = 1, 2, ..., 46$ 且为整数

问题一模型中式（3）、（4）、（5）也作为该模型条件

4.2.3 结果分析

由以上模型编写时间复杂度为 $O(n*k+k^3)$ 的 C++程序（其中 n 为呼叫总数，k 为区域总数），在 vs2017 平台下运行 20 次取平均，得到结果如下表：

表 2 各地点平均响应时间

地点编号	1	2	3	4	5	6	7	8	9
平均响应时间(min)	377.4	313.1	265.9	214.8	184.8	203.2	196	233.9	288.6
地点编号	10	11	12	13	14	15	16	17	18
平均响应时间(min)	339.8	361.9	213.9	238.5	177.2	156.5	148.1	163.2	194.3
地点编号	19	20	21	22	23	24	25	**26**	27
平均响应时间(min)	249.5	258.5	321.9	298.5	271.9	163.7	141.6	**133.1**	149.3
地点编号	28	29	30	31	32	33	34	35	36
平均响应时间(min)	317.4	379	283.9	334.7	271.8	221.5	175.8	154.2	143.6
地点编号	37	38	39	40	41	42	43	44	45
平均响应时间(min)	161.2	191.4	240.7	297.1	415	355.4	302	229.4	250.1
地点编号	46	47							
平均响应时间(min)	300.4	179.1							

由以上结果可知当区域 26 作为中心设施、救助队行驶速度为 72km/h 时的最短平均响应时间为 133.1 分钟，比在位置 16 的平均响应时间 148.3 分钟缩短了 15.2 分钟，但响应时间依旧过于长。

4.3 问题三

4.3.1 问题的分析

考虑救助中心的救助队数量足够多的情况下，即从救助中心收到救助请求到救助队前去救助过程中没有发生等待，此时平均响应时间最短。由问题一程序可测试出在救助队数量足够多的情况下最短平均响应时间为 4.9 分钟，明显小于问题一的平均响应时间 148.3 分钟。由此可知第一问的平均响应时间大幅增加是由于救助中心救助队数量少，呼叫请求在发出到救助中心派出救助队赶往呼叫地点的过程中发生长时间的等待所造成的。因此当救助中心在 16 区域左下角，救助队平均速度为 72km/h 时，可以通过增加救助队数量来控制平均响应时间在 5 分钟之内。

4.3.2 模型的建立

平均响应时间=等待派出救助队时间+救助队前往呼叫地点的赶路时间。

因为救助队赶往呼叫地点的赶路时间是确定的。所以要降低平均响应时间使其在 5 分钟之内就只有通过降低等待派出救助队的时间来实现。当救助队数量 k 增加时，派出救助队前往第 i 个呼叫地点的时间 S_i 就可能越靠近第 i 个地点的呼叫时刻 t_i，从而降低派出救助队的等待时间从而使平均响应时间降低到 5 分钟之内。

所以我们可得目标一：$\min = \sum_{i=1}^{47} S_i - t_i$

其次一个合理的救助队数量要求在满足平均响应时间小于 5 分钟的情况下救助队数量最少。所以可得目标二：$\min = k$

其他约束条件在第一问模型上通过修改救助队数量来实现

所以可得如下数学模型：

$$\text{目标：}\min = \sum_{i=1}^{47} S_i - t_i$$

$$\min = k$$

$$
s.t.\begin{cases}
S_i = \min(E_k) \\
\overline{R}_p = \sum_{i=1}^{n} R_p \Big/ n \le 5 \\
R_p = \begin{cases} T_i \ , i \le k \\ T_i + (1-a)(S_i - t_i) \ , i \ge k+1 \end{cases} \\
a = \begin{cases} 0 \, , S_i > t_i \\ 1 \, , S_i < t_i \end{cases} \\
E_k = S_j + 2T_j + F_j \\
T_i = \dfrac{L_i}{v} \\
k\text{为正整数} \\
p = 1,2,\ldots,47, \text{且为正整数}
\end{cases}
$$

4.3.3 结果分析

由以上模型编写时间复杂度为$O(n+k^2)$的 C++程序（其中 n 为呼叫总数，k 为区域总数），在 vs2017 平台下运行 20 次取平均，得到不同救助队数量对应的平均响应时间，结果如下表：

表 3 不同救助队数量的平均服务响应时间

救助队数量	3	4	5	6	7	8	9	10
平均响应时间(min)	148.3	24	9.1	6.2	5.3	5	4.9	4.9

根据以上数据得到其变化趋势如下：

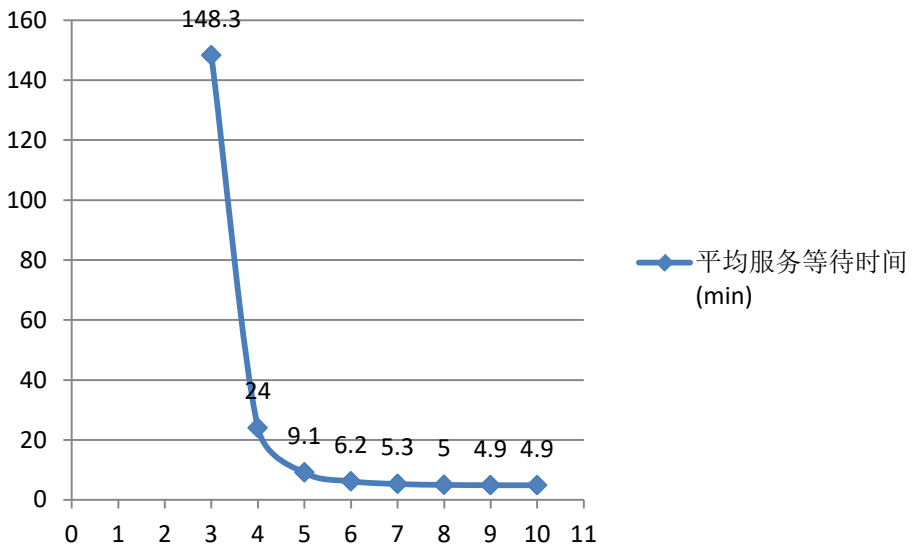

图 3 平均响应时间随救助队数量变化情况

由图像可知，当救助队数量达到 8 个时，即可将平均响应时间控制在 5 分钟以内。因此，应该增加 5 个救助队，此时每天的平均响应时间如下表：

表 4 各天平均响应时间

天数	第 1 天	第 2 天	第 3 天	第 4 天	第 5 天	第 6 天	第 7 天	第 8 天
平均响应时间(min)	5.3	4.8	4.7	5.2	5.1	4.9	4.8	5.2

将上述 8 天的各自的平均响应时间取均值得总的平均响应时间为 5.0 分钟。此时在保证能够及时到达求助地点的同时，救助队的数量最少。

4.4 问题四

4.4.1 问题的分析

因为一天中的11:00-13:00和17:00-19:00时段为城市中心地区交通高峰期，彼时会有大量车辆出入中心城区。因此高峰期经过城市中心的救助队会受到车流量增大的影响，造成平均行驶速度降低。但由于救助队从救助中心到呼叫地点的路程中，如果有经过中心城区，只会在经过中心城区的那一小段路上减速而其他路段还是维持原速，这样很难把握一次救助过程中车速的变化情况。因此可以从另一角度来考虑这个问题：由公式$t = s / v$知，路程不变车速降低使时间增加可以转换成车速不变路程增加时间增加的情况。考虑高峰期时救助队在城市中心区行驶的速度为正常情况下的一半。

因此可以将高峰期时候城市中心地区的路段长度增加一倍来实现一个救助队经过该时刻速度下降一半时间增长的效果。然后用 Dijkstra 算法分别求出高峰期与非高峰期时中心设施到各区域的最短路径，然后修改问题二中的模拟仿真程序使非高峰期时计算方式不变，高峰期时执行任务的救助队按照新的最短路径计算其到呼叫地点的响应时间然后求均值得出所有呼叫请求总的平均响应时间。

4.4.2 模型的建立

由问题分析可知，要解决问题四。们还要知道的参数有交通高峰期时助队在城市中心路段的平均行驶速度。经查阅资料我们可得在峰期时车辆行进速度大约为正常时候的一半，所以我们假设高峰期救助队经过中心城区路段的速度为正常速度的一半来进行仿真计算。另外我们按照高峰期时中心路段增长一倍替代速度减半的思想使用Dijkstra算法计算出高峰期时段以区域 p 作为控制中心到各区域 i 的最短距离 L'_{pi}，然后类比问题二建立如下模型：

目标：$\min = \overline{R}_p$

$$T_i = \begin{cases} \dfrac{L_{pi}}{v} & \text{非高峰期段} \\ \dfrac{L'_{pi}}{v} & \text{高峰期段} \end{cases} \tag{10}$$

$$\overline{R}_p = \sum_{i=1}^{n} R_p \Big/ n \tag{11}$$

（12）

$$R_p=\begin{cases}T_i & ,i\le 3\\ T_i+(1-a)(S_i-t_i) & ,i\ge 4\end{cases}$$

$p=1,2,...,46$ 且为整数

问题一模型中式（3）、（4）、（5）也作为该模型条件

4.4.3 结果分析

由以上模型编写复杂度为$O(n*k+k^3)$的 C++程序（其中 n 为呼叫总数，k 为区域总数），在 vs2017 平台下运行 20 次，得到平均结果如下表：

表 5 各地点平均响应时间

地点编号	1	2	3	4	5	6	7	8	9
平均响应时间(min)	396.4	335	294.1	248.5	221.7	211.5	232.9	266.2	313.1
地点编号	10	11	12	13	14	15	16	17	18
平均响应时(min)	360.1	380.2	236.7	263	207.2	181.9	171.7	187	226.7
地点编号	19	20	21	22	23	24	25	26	27
平均响应时(min)	275.6	320.2	339.8	280.7	301	187	165.3	154.8	171
地点编号	28	29	30	31	32	33	34	35	36
平均响应时间(min)	345.6	399.8	302.4	354.6	292.3	247.9	199.6	174.8	165.5
地点编号	37	38	39	40	41	42	43	44	45
平均响应时间(min)	219.8	183.8	267.6	316.3	435	373.3	324.5	279.9	253.1
地点编号	46	47							
平均响应时间(min)	318.7	206.1							

当高峰期时，救助队在经过城市中心城市路段速度减为为 36km/h，郊区速度仍然为 72km/h，由以上结果可知将中心选在 26 的位置仍然平均响应时间最短为 154.8 分钟。此时间比不考虑高峰期城市中心路段拥堵时的 133.1 分钟多 21.7 分钟，结果符合实际情况。

5 模型评价与推广

5.1 模型的评价

本文是通过 Dijkstra 与计算机仿真两种算法相结合建立起的模型来对问题求解，在合理的假设下，我们考虑了服务时间服从均值为 15 分钟方差 1 的正态分布来模拟仿真，使结果更接近真实情况。另外我们对每一个问题通过仿真 20 次取平均的做法有效的避免了一些偶然因素的影响，这样让结果更有说服力。另外在问题四的求解上，我们巧妙地将救助队在高峰时段经过中心城区时减速的情况用该时刻该路段距离的增长来实现有效的模拟出了高峰时段中心城区道路拥挤对救助队行进的影响。整体来说我们针对紧急医疗救助问题所建立的模型比较合理对一些细节的考虑较为充分，因此最后得到的结果也较优。

5.2 模型的推广

Dijkstra 算法作为求解单源最短路径的著名算法可以很好的运用于网络优化问题的求解中。另外基于该问题编写的以事件为步长的仿真程序可以推广到生活中很多类似的排队问题，如银行最优柜台设置、公路最佳车道设置等。

6 参考文献

[1]刘万宇.城市应急服务设施选址模型与算法研究[D].哈尔滨理工大学,2016.

[2]魏崴.运输时间不确定性的应急设施选址研究[D].合肥工业大学,2012.

[3]刘东圆,徐园,耿彦斌,姜长杰.突发公共事件下应急救援设施选址模型与案例[J].交通标准化,2009,(14):81-85.

[4]赖寒.地震灾害中应急医疗设施选址问题研究[D].西南交通大学,2009.

[5]李艳杰.应急服务设施选址问题研究[D].辽宁科技大学,2008.

[6]韩强.多目标应急设施选址问题的模拟退火算法[J].计算机工程与应用,2007,(30):182-183+216.

[7]贾静. 突发公共卫生事件的应急医疗设施选址问题研究[D].北京交通大学,2007.

[8]殷代君.广义最大覆盖模型在应急设施选址中的应用研究[D].山东大学,2007.

[9]万波.公共服务设施选址问题研究[D].华中科技大学,2012.

[10]张金凤.设施选址问题的建模及优化算法研究[D].广东工业大学,2011

附录：

附录 1：问题一 Dijkstra 算法程序

```
/*
1.使用 Dijkstra 算法计算中心到各个点的最短路径
*/
#include<iostream>
#include<algorithm>
#include <fstream>
using namespace std;
const int maxn = 50;//一共有 47 个点
float c[maxn][maxn], //the matrix to storage graph
dist[maxn];// the shortest length from u to i
int vis[maxn];//if vis[i]==1,the i was calculated
const int maxint = 100000;
int cnt = 48;//the number of vertex
int s = 16;
void Dijk()
{
  int index = 0;
  c[s][s] = 0; vis[s] = 1; dist[s] = 0;
  for (int i = 1; i < cnt; i++)
  {
        float m = maxint;
        for (int j = 1; j < cnt; j++)
              if (!vis[j] && m > dist[j])
              {
                    m = dist[j];
                    index = j;
              }
        vis[index] = 1;
        for (int k = 1; k < cnt; k++)
        {
              if (!vis[k] && dist[index] + c[index][k] < dist[k])
                                    dist[k] = dist[index] + c[index][k];
        }
  }
}
int main()
{
  char pn[] = "E:/编程代码/shuMo/邻接矩阵.txt";
  freopen(pn, "r", stdin);
  //initial the graph
  int i, j, k;
  for (int i = 1; i < 50; i++)
```

```
    {
        dist[i] = maxint;
        for (int j = 0; j < 50; j++)
            c[i][j] = maxint;
    }
    memset(vis, 0, sizeof(vis));
    while (scanf("%d%d%d", &i, &j, &k) == 3)
    {
        float l = k*1.5;
        c[i][j] = c[j][i] = l;
        if (i == s)//要计算的初始点的编号
            dist[j] = l;
        else if (j == s)
            dist[i] = l;
    }
    Dijk();
    ofstream out("E:/编程代码/shuMo/距离信息.txt");
    for (int i = 1; i < cnt; i++){out << dist[i] << endl;}
    for (int i = 1; i < cnt; i++){printf("%d 到%d 的最短距离为:%.1f\n", s, i,
dist[i]);}
    freopen("CON", "r", stdin);
    system("pause");
    return 0;
}
```

附录 2：问题一计算机仿真程序

```
/*
1.编程遍历以每一个节点作为中心时到各个节点的最短路径
2.计算以各个节点为中心时的平均响应时间
*/
#include<iostream>
using namespace std;
const int maxn = 50;//一共有 47 个点
float c[maxn][maxn], //the matrix to storage graph
dist[maxn];// the shortest length from u to i
int vis[maxn];//if vis[i]==1,the i was calculated
const int maxint = 100000;
int cnt = 48;//the number of vertex
void Dijkstra(int s)
{
    int index = 0;
    c[s][s] = 0; vis[s] = 1; dist[s] = 0;
    for (int i = 1; i < cnt; i++)
    {
        float m = maxint;
```

```
        for (int j = 1; j < cnt; j++)
            if (!vis[j] && m > dist[j])
            {
                m = dist[j];
                index = j;
            }
        vis[index] = 1;
        for (int k = 1; k < cnt; k++)
        {
            if (!vis[k] && dist[index] + c[index][k] < dist[k])
            {
                dist[k] = dist[index] + c[index][k];
            }
        }
    }
}
void shortestPath(int s)
{
    char pn[] = "E:/编程代码/shuMo/邻接矩阵 1.txt";
    freopen(pn, "r", stdin);
    //initial the graph
    int i, j, k;
    for (int i = 1; i < 50; i++)
    {
        dist[i] = maxint;
        for (int j = 0; j < 50; j++)
            c[i][j] = maxint;
    }
    memset(vis, 0, sizeof(vis));
    while (scanf("%d%d%d", &i, &j, &k) == 3)
    {
        float l = k*1.5;
        c[i][j] = c[j][i] = l;
        if (i == s)//要计算的初始点的编号
            dist[j] = l;
        else if (j == s)
            dist[i] = l;
    }
    Dijkstra(s);
    ofstream out("E:/编程代码/shuMo/距离信息.txt");
    for (int i = 1; i < cnt; i++){out << dist[i] << endl;}
    freopen("CON", "r", stdin);
}
/*
```

```
1.模拟仿真一天任务的平均响应时间
2.是事件为步长进行仿真
*/
float speed = 1.2;//救护车的速度 km/min
float day[9];//每一天的任务的平均响应时间
struct call {
  int number;float start;float dis;float end;
  bool operator < (const call &c)const{return end > c.end;}
};
float a[maxn];//存放中心为 i 时的 8 天平均响应时间

void simulation(int k)//第 k 个地点作为中心的时候的仿真情况
{
  random_device rd;
  mt19937 gen(rd());
  normal_distribution<double> normal(15, 1);//产生服从正态分布，均值为 15
方差为 1 的随机数
  char pn[] = "E:/编程代码/shuMo/距离信息.txt";
  freopen(pn, "r", stdin);int i = 1;      float d;
  while (scanf("%f", &d) == 1) {//读取最短路径信息
        dist[i++] = d;
  }
  for (int i = 1; i < 9; i++) {
        char pn2[] = "E:/编程代码/shuMo/呼叫记录  .txt";
        pn2[26] = i + '0';
        freopen(pn2, "r", stdin);
        priority_queue<call> pq;
        queue<call> q;int n;float s;float t = 0;//整个模拟过程的时间轴
        int cnt = 0;//整个模拟过程中的事件个数
        float rt = 0;//整个模拟过程中的所有时间总的响应时间
        while (scanf("%f%d", &s, &n) == 2) {//读取最短路径信息
              struct call c;c.number = n;c.start = s;c.dis = dist[n];q.push(c);
        }
        cnt = q.size();
        //初始化
        while (q.size() != 0) {
              struct call c = q.front(); q.pop();
              float s = (c.dis / speed) * 2 + normal(gen);//一次服务总共需要花费
的时间
              if (t < c.start)
                    t = c.start;//时间推进
              rt += (c.dis / speed) + t - c.start;
              c.end = t + s;//完成任务的时刻
              pq.push(c);
```

```
            if (q.empty())//如果所有事务都处理完了直接跳出
                break;
            struct call pc = pq.top();
            struct call c2 = q.front();
            if (c2.start < pc.end&&pq.size() < 3) { continue;} //请求发出，中心有空闲的救护车可以出车
            else {
                t = pc.end;//时间推进
                pq.pop();//处理完一个服务，将其从事件处理队列中弹出
            }
        }
        day[i] = rt / cnt;
    }
    freopen("CON", "r", stdin);
    float total = 0;
    for (int i = 1; i < 9; i++) {total += day[i];}
    a[k] = total / 8;
}
int main()
{
    int j;
    float m=500;
    for (int i = 1; i < 48; i++) {
        shortestPath(i);//规划最短路径
        simulation(i);
        if (m > a[i]) {
            j = i;
            m = a[i];
        }
        printf("第%d 个地点作为中心的时候的 8 天平均服务等待时间%.1f\n", i, a[i]);
    }
    printf("\n\n 服务时间最短的将中心选在%d 的位置,其最短服务时间为:%.1f\n", j, m);
    system("pause");
    return 0;
}
```

附录 3：问题二程序

```
/*
1.编程遍历以每一个节点作为中心时到各个节点的最短路径
2.计算以各个节点为中心时的平均响应时间
*/
#include<iostream>
using namespace std;
```

```
const int maxn = 50;//一共有 47 个点
float c[maxn][maxn], //the matrix to storage graph
dist[maxn];// the shortest length from u to i
int vis[maxn];//if vis[i]==1,the i was calculated
const int maxint = 100000;
int cnt = 48;//the number of vertex
void Dijkstra(int s)//该函数等同于附录二中的
void shortestPath(int s)
{
  char pn[] = "E:/编程代码/shuMo/邻接矩阵.txt";
  //其余部分等同于附录二中对应部分
  }
  Dijkstra(s);
  ofstream out("E:/编程代码/shuMo/距离信息.txt");
  for (int i = 1; i < cnt; i++)
        out << dist[i] << endl;
  freopen("CON", "r", stdin);
}
/*
1.模拟仿真一天任务的平均响应时间
2.是事件为步长进行仿真
*/
float speed = 1.2;//救护车的速度 km/min
float day[9];//每一天的任务的平均响应时间
struct call{};//等于与附录二对应结构体
float a[maxn];//存放中心为 i 时的 8 天平均响应时间
void simulation(int k)//第 k 个地点作为中心的时候的仿真情况
{
  random_device rd;
  mt19937 gen(rd());
  normal_distribution<double> normal(15, 1);//产生服从正态分布，均值为 15
方差为 1 的随机数

  char pn[] = "E:/编程代码/shuMo/距离信息.txt";
  freopen(pn, "r", stdin);
  int i = 1;
  float d;
  while (scanf("%f", &d) == 1) {//读取最短路径信息
      dist[i++] = d;
  }

  for (int i = 1; i < 9; i++) {
      char pn2[] = "E:/编程代码/shuMo/呼叫记录  .txt";
      pn2[26] = i + '0';
```

```
        freopen(pn2, "r", stdin);
        priority_queue<call> pq;
        queue<call> q;int n;float s;
        float t = 0;//整个模拟过程的时间轴
        int cnt = 0;//整个模拟过程中的事件个数
        float rt = 0;//整个模拟过程中的所有时间总的响应时间
        while (scanf("%f%d", &s, &n) == 2) {//读取最短路径信息，代码同上
        }
        cnt = q.size();
        //初始化
        while (q.size() != 0) {
            struct call c = q.front(); q.pop();
            float s = (c.dis / speed) * 2 + normal(gen);//一次服务总共需要花费
的时间
            if (t < c.start)
                t = c.start;//时间推进
            rt += (c.dis / speed) + t - c.start;
            c.end = t + s;//完成任务的时刻
            pq.push(c);
            if (q.empty())//如果所有事务都处理完了直接跳出
                break;
            struct call pc = pq.top();
            struct call c2 = q.front();
            if (c2.start < pc.end&&pq.size() < 3) { continue;} //请求发出，中心
有空闲的救护车可以出车
            else {
                t = pc.end;//时间推进
                pq.pop();//处理完一个服务，将其从事件处理队列中弹出
            }
        }
        day[i] = rt / cnt;
    }
    freopen("CON", "r", stdin);
    float total = 0;
    for (int i = 1; i < 9; i++) {total += day[i];}
    a[k] = total / 8;
}
int main()
{
    int j;
    float m=500;
    for (int i = 1; i < 48; i++) {
        shortestPath(i);//规划最短路径
        simulation(i);
```

```
            if (m > a[i]) {j = i;m = a[i];}
            printf("第%d 个地点作为中心的时候的 8 天平均服务等待时间%.1f\n",
    i, a[i]);
      }
      printf("\n\n 服务时间最短的将中心选在%d 的位置,其最短服务时间
    为:%.1f\n", j, m);
      system("pause");
      return 0;
    }
```

附录 4：问题四程序

```
    /*
    1.编程遍历以每一个节点作为中心时到各个节点的最短路径
    2.计算以各个节点为中心时的平均响应时间
    */
#include<iostream>
using namespace std;
const int maxn = 50;//一共有 47 个点
float c[maxn][maxn], //the matrix to storage graph
dist2[maxn],//高峰期的时候的最短路径
dist[maxn];//  普通时刻从 u 到 i 的最短路径
int vis[maxn];//if vis[i]==1,the i was calculated
const int maxint = 100000;
int cnt = 48;//the number of vertex
void Dijkstra(int s) //该函数等同于附录二中的
void shortestPath(int s, int z)
{
      char pn[] = "E:/编程代码/shuMo/邻接矩阵  .txt";
      pn[26] = z + '0';
      freopen(pn, "r", stdin);
      //initial the graph
      int i, j, k;
      for (int i = 1; i < 50; i++)
      {
            dist[i] = maxint;
            for (int j = 0; j < 50; j++)
                  c[i][j] = maxint;
      }
      memset(vis, 0, sizeof(vis));
      while (scanf("%d%d%d", &i, &j, &k) == 3)
      {
            float l = k*1.5;
            c[i][j] = c[j][i] = l;
            if (i == s)//要计算的初始点的编号
                  dist[j] = l;
            else if (j == s)
                  dist[i] = l;
      }
```

```
    Dijkstra(s);
    ofstream out("E:/编程代码/shuMo/距离信息.txt");
    for (int i = 1; i < cnt; i++){out << dist[i] << endl;}
    freopen("CON", "r", stdin);
}
/*
1.模拟仿真一天任务的平均响应时间
2.是事件为步长进行仿真
*/
//float dist[maxn];
float speed = 1.2;//救护车的速度 km/min
float day[9];//每一天的任务的平均响应时间
float a[maxn];//存放中心为 i 时的 8 天平均响应时间
void simulation(int k)//第 k 个地点作为中心的时候的仿真情况
{
    random_device rd;
    mt19937 gen(rd());
    normal_distribution<double> normal(15, 1);//产生服从正态分布，均值为 15 方差为 1 的随机数
    char pn[] = "E:/编程代码/shuMo/距离信息.txt";
    freopen(pn, "r", stdin);
    int i = 1;
    float d;
    while (scanf("%f", &d) == 1) {//读取最短路径信息
        dist[i++] = d;
    }
    for (int i = 1; i < 9; i++) {
        char pn2[] = "E:/编程代码/shuMo/呼叫记录  .txt";
        pn2[26] = i + '0';
        freopen(pn2, "r", stdin);
        priority_queue<call> pq;
        queue<call> q;
        int n;
        float s;
        float t = 0;//整个模拟过程的时间轴
        int cnt = 0;//整个模拟过程中的事件个数
        float rt = 0;//整个模拟过程中的所有时间总的响应时间
        while (scanf("%f%d", &s, &n) == 2) {//读取最短路径信息，代码同上

        }
        cnt = q.size();
        //初始化
        while (q.size() != 0) {

            struct call c = q.front(); q.pop();

            if (t < c.start)
                t = c.start;//时间推进
```

```
                float sp;
                if (t > 660 && t < 780 || t>1020 && t < 1140)//分时段计算时间
                    sp = c.dis2 / speed;
                else
                    sp = c.dis / speed;
                rt += sp + t - c.start;
                float s = sp * 2 + normal(gen);//一次服务总共需要花费的时间
                c.end = t + s;//完成任务的时刻
                pq.push(c);

                if (q.empty())//如果所有事务都处理完了直接跳出
                    break;
                struct call pc = pq.top();
                struct call c2 = q.front();
                if (c2.start < pc.end&&pq.size() < 3) { continue;} //请求发出，中心有
空闲的救护车可以出车
                else {
                    t = pc.end;//时间推进
                    pq.pop();//处理完一个服务，将其从事件处理队列中弹出
                }
            }
            day[i] = rt / cnt;
        }
        freopen("CON", "r", stdin);
        float total = 0;
        for (int i = 1; i < 9; i++) {total += day[i];}
        a[k] = total / 8;
}
int main()
{
        int j;
        float m = 500;
        for (int i = 1; i < 48; i++) {
            shortestPath(i, 2);//规划高峰期时候的最短路径
            for (int j = 0; j < 48; j++)
                dist2[j] = dist[j];
            shortestPath(i, 1);//规划普通时候的最短路径
            simulation(i);
            if (m > a[i]) {j = i;m = a[i];}
            printf("第%d 个地点作为中心的时候的 8 天平均服务等待时间%.1f\n", i,
a[i]);
        }
        printf("\n\n 服务时间最短的将中心选在%d 的位置,其最短服务时间
为:%.1f\n", j, m);
        system("pause");
        return 0;
}
```