武漢理工大學

数学建模暑期培训论文

第 2 题

基于灾区存活率与货物运输问题的非线性整数规划模型

第 21 组

姓名	方向
肖善	建模
易雨谦	建模
林荣武	编程

2016 年 8 月 13 日

基于灾区存活率与货物运输问题的非线性整数规划模型

摘要

本文针对地震灾区人员存活与货物运输问题分别建立了存活率模型与非线性整数规划模型，并用基于贪心决策的自适应权重粒子群算法进行求解，用 Lingo 中的全局最优解进行验证。

对于问题一，要求建立存活率与受灾时间的关系，考虑到地震中我们往往直接了解的是人的受伤程度，因此我们从伤亡程度入手建立伤亡程度与个体存活率之间的关系，首先，通过查阅大量文献我们引用伤亡函数建立的伤亡状态模型；接着，我们根据存活率变化趋势构建关于伤亡函数的中间函数从而建立存活率模型，根据黄金救援时间等约束条件对模型进行修正并求解参数，画出曲线图形；最后，用文献中 1998 年日本阪本地震的存活率的指数函数经验公式进行模型检验，得到相对误差在 7%左右，我们认为结果较为准确且符合实际，最后结合实际针对不同身体素质和不同环境对模型做了进一步改进和推广。

对于问题二，建立非线性整数规划模型，首先，假设时间具有累积性，以累积总运输时间最小为目标，以 4 个受灾区 A、B、C、D 的需求量、13 个配送点的储备量为约束条件，利用 52 个(0,1)决策变量表示某配送点是否向某受灾区运货，用 Lingo 求解，经过 2506 次迭代，在 10 个临时配送点中选用了 6、7、11、12、13 这 5 个配送点，求得累积最短时间为 186 小时，通过多次约束检验认为最优解唯一；接着，考虑到实际上时间具有同步性，即所有车同时出发，则最短时间即为最大运货时间中的最小值，在 matlab 中用贪心算法求解最短时间为 27 小时，选用了 6 个临时配送中心；最后，用 Lingo 的全局最优解进行验证， Lingo 求得同时出发最短时间为 27 小时，在 10 个临时配送点中选择了 6 个配送点；结果完全相同，我们认为结果正确且符合实际。

对于问题三，首先我们沿用问题二的模型，在此基础上扩大了数据规模，考虑三种扩大情况：单一扩大灾区点数目，单一扩大资源点数目，同时扩大灾区点和资源点数目。在这三种情况增加灾区数目 3，资源点 4，并用基于贪心决策的自适应权重粒子群算法进行求解，用 Lingo 中的全局最优解进行验证。比较运输结果是否贴近实际，运输效率是否高以及算法执行时间是否较快从而说明各个算法的优点与缺点从而确定大规模运输问题主算法并得出结果。

关键词：伤亡状态函数　整数非线性规划　　贪心算法　粒子群算法

一、问题重述

某次灾害发生后，经勘测确定的受灾区域分别为 A,B,C,D，现已设有三个紧急物资固定配送中心，现需从待选的 10 个紧急物资临时配送中心中选取若干，配合固定配送中心对灾区运送紧急物资以提高救灾效率。各个受灾区域对物资的需求量、各个配送中心的储备量（其中 1-3 为固定的配送中心，4-13 为备选配送中心），以及各个受灾区域到配送中心的最短时间（小时）等数据。试构建数学模型分析下列问题：

（1）建立地震灾害发生后人的存活率与受灾时间的关系；
（2）建立数学模型并设计算法对临时配送中心进行选址；
（3） 若当突发事件（各类节点数量规模）较大时，请验证所提模型及算法是否可行。

二、问题分析

2.1 问题一的分析

问题一要求建立存活率与受灾时间的关系，考虑到地震中我们往往直接了解的是人的受伤程度，因此我们从伤亡程度入手建立伤亡程度与个体存活率之间的关系，首先，通过查阅大量文献我们知道早在 1998 年赵振东提出伤亡函数并考虑身体素质和地震囤积环境因素深入研究，得出了伤亡状态模型，建立了人员伤亡指数与时间的关系；2013 年刘志鹏在基于玉树地震的上亡分析与预测文中提出并验证了地震两期规律[1]，通过引用赵振东提出的伤亡函数建立伤亡模型；2015 年高娜等人在地震应急救灾效能一文中用多次地震的数据拟合并结合“黄金 24 小时”等救灾常识得出了存活率随时间的变化关系。然后我们根据存活率变化趋势构建关于伤亡函数的中间函数从而建立存活率模型，根据约束条件对模型进行修正并求解参数；最后，用文献中 1998 年日本阪本地震的存活率的指数函数经验公式进行模型检验，并分析误差。

2.2 问题二的分析

问题二属于基于非线性整数规划的多地选址运输问题，首先，考虑最坏的发车情况，即假设每辆车运完了下一辆才出发，此时时间具有累积性，以累积总运输时间最小为目标，以 4 个受灾区 A、B、C、D 的需求量、13 个配送点的储备量为约束条件，利用 52 个(0,1)决策变量表示某配送点是否向某受灾区运货，用 Lingo 求解；然后，考虑最好的发车情况，即所有货车同时出发，则时间具有同时性 ，以全部组合中最大时间的那辆货车所用时间最短为目标函数，约束条件不变，在 matlab 中用贪心算法求解；最后，由于贪心算法不能保证得出的结果是全局最优解，因此我们用 Lingo 得出的结果进行检验。

2.3 问题三的分析

对于问题三，为了检验结果的正确性并预测规模扩大算法的可行性，我们比较了不同算法的优劣以及复杂度，首先我们沿用问题二的模型，然后在此基础上扩大了数据规模，我们考虑了三种扩大情况：单一扩大灾区点数目，单一扩大资源点数目，同时扩大灾区点和资源点数目。在这三种情况下比较运输结果是否贴近实际，运输效率是否高以及算法执行时间是否较快从而说明各个算法的优点与缺点从而确定大规模运输问题主算法并得出结果。

三、模型假设

1.问题一假设存活率只于自身的伤亡状态以及环境状态有关。
2.问题二假设运输过程能够顺利进行。
3.问题二假设单位运输量与运输时间无关，不考虑成本因素。
4.问题二三假设供应与需求量在相当长的一段时间内不发生变化。

四、符号说明

符号	含义
t	时间
$c(t)$	伤亡状态函数
c_0	初始伤亡状态
S_0	环境因素
$f(t)$	存活率函数
a_j	第 j 个资源点供应量
b_i	第 i 个灾区点的需求量
x_{ij}	从第 j 个资源点运往第 i 个灾区点的运输量

五、模型的建立和求解

5.1 问题一模型的建立与求解

5.1.1 地震两期规律分析

1、地震伤亡增长通过明显拐点分为“伤亡增长期”与“伤亡持续期”两部分，地震伤员的增长以及死亡人员的发现随着救援力量进入灾区而不断上升，在经历初期的伤亡快速增长后，出现明显的时间拐点，然后转变为增长速度越来越小的伤亡持续期，直至最终救灾工作完成。

2、曲线拐点出现的时间与地震震情、救援力量覆盖速度有关。地震伤亡规模不同，时间拐点出现的时间也不同，一般伤亡规模较小的地震，时间拐点出现越早，一般都在震后一周内，而对于伤亡规模较大的地震，时间拐点出现得越晚，多在震后一周后，甚至震后两周时出现。

5.1.2 人员伤亡指数与地震伤亡函数

1、人员伤亡指数

为了衡量人的伤亡状态，我们定义伤亡指数[2]$C \in (0,1)$ ， 表示地震后被埋压人员的受伤程度，C=0 表示没有受伤是健全的，C=1 表示人已经死亡。

表 1 人员伤亡指数

伤残指数 C	0-0.1	0.1-0.3	0.3-0.6	0.6-0.9	0.9-1.0
伤残情况	基本无伤残	轻微伤残	中等伤残	严重伤残	生命垂危

2.地震伤亡状态函数

经过查阅相关文献，我们得到地震伤亡状态函数[3]如下所示：

$$C(t) = (C_0^{\frac{1}{n}} + S_i t)^n \qquad (1\text{-}1)$$

参数范围：

$$0 \le C_0 \le 1$$
$$0.004 \le S_0 \le 0.1$$
$$1.0 \le n \le 3.0$$

其中，C_0 为初始伤亡指数即震后瞬时人的受伤程度，属于（0，1）范围，n 为身体素质指数，衡量了不同人的个人体质条件，属于（1，3）范围，S_i 为受困环境参数属于（0.004，1）范围，衡量了震后人所处环境的恶劣程度，t 为受困时间。

分别控制 s 和 n,得到 C(t)函数图像如下：

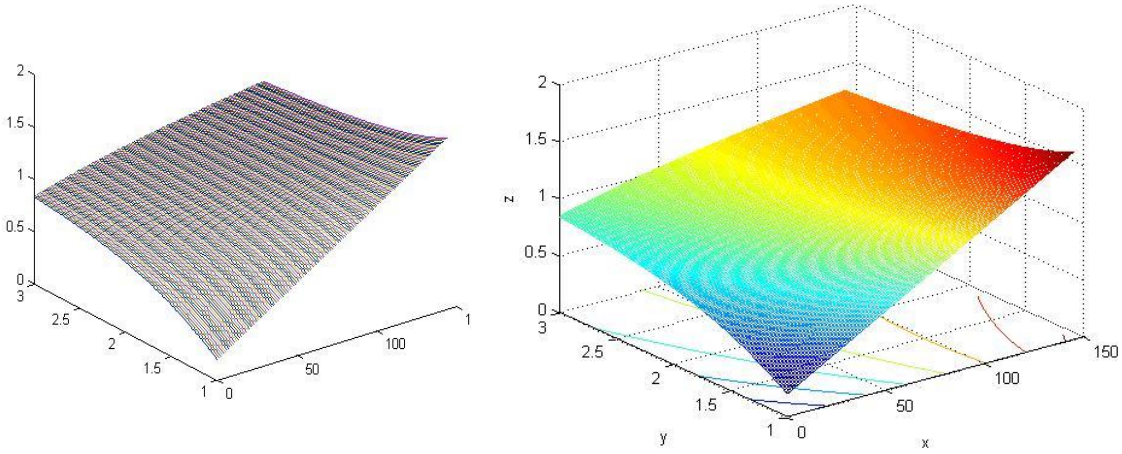


图 1 控制 s 得到的图像

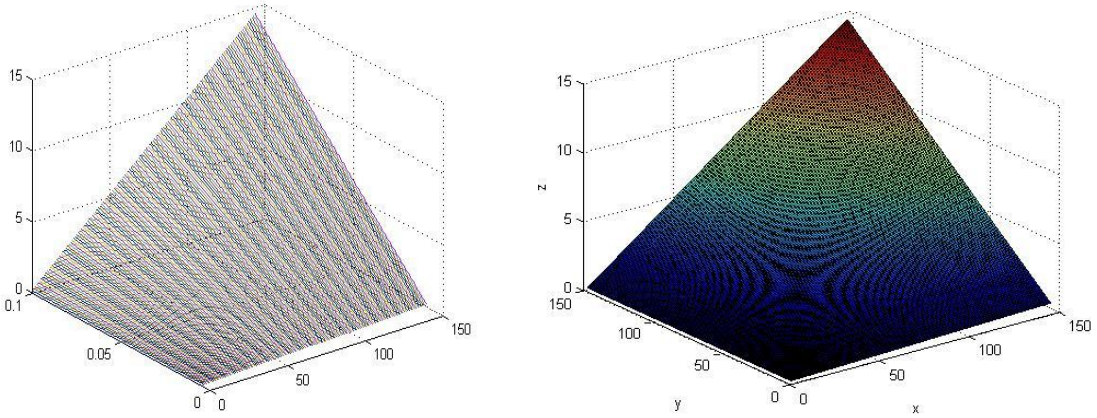


图 2 控制 n 得到的图像

我们给定 $C_0 = 0.2$ $S_0 = 0.01$ $n = 1$ 画出伤亡状态函数图像：

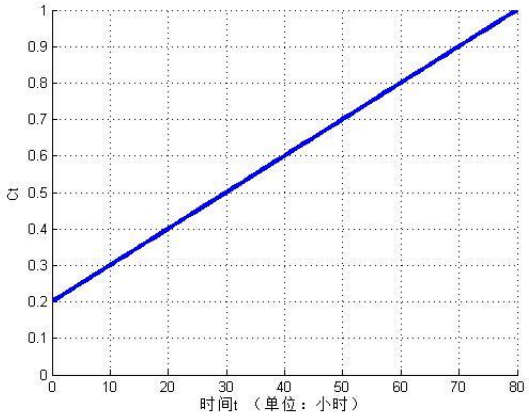


图 3 伤亡状态函数图像

5.1.3 震后生存率模型建立

已知伤亡状态函数 C(t)，希望求出生存率状态函数 f(t)，则需建立二者关系，由生活常识可知，震后人被掩埋的时间越长，伤亡程度越高，存活可能性越小即伤亡指数 C 越大生存概率 f 越小，我们发现，前期伤亡指数受环境因素影响比个人体质因素影响更大[4]，后期环境稳定，伤亡指数受个人体质影响比环境因素更大；震后瞬时，存活可能性非常大接近 1，而经过 144 小时存活可能性非常小几乎为 0，在这 144 时小时以内刚开始由于地面变化情况大，人会突然受到强大冲击，存活率急剧下降，随着环境稳定，下降速率越来越缓慢，则可以确定函数 f 是一个斜率递减的减函数。考虑到 C(t)为斜率不减的增函数建立生存率模型：

$$f(\mathrm{t}) = \mathrm{A}\frac{1}{\mathrm{C(t)}} + B \qquad (1\text{-}2)$$

定义中间函数 $g(\mathrm{t}) = \frac{1}{C(\mathrm{t})}$ ，并绘制 g(t)函数图象如下：

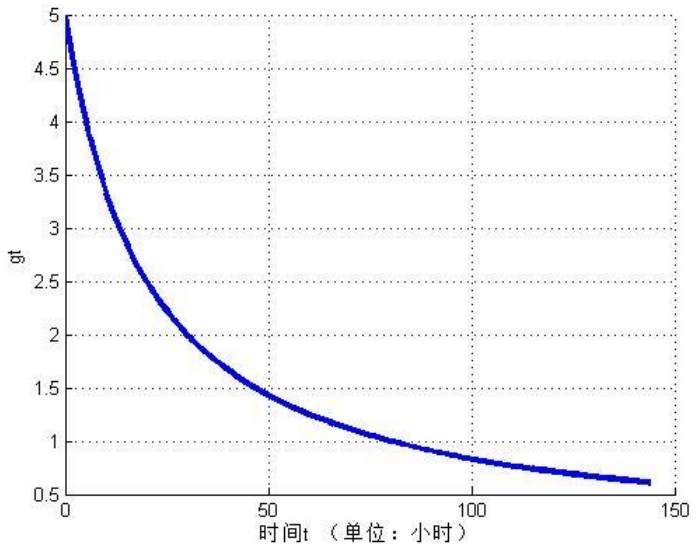


图 4 g(t)函数图像

观察其趋势基本符合，根据上文分析近似的有

$$f(144) = 0$$
$$f(0) = 1$$

且 72 小时为黄金救援时间，即 $t \le 72$ 时 f(t)应在 0.1 以上，$t > 72$ 时存活可能性很小，为了达到以上要求，对 g(t)函数做相应变换得到参数 $\mathrm{B}=\frac{g(144)}{g(144)-\mathrm{g}(0)}$

$\mathrm{A}=\frac{1}{g(0)-\mathrm{g}(144)}$ ，则生存率函数为：

$$f(\mathrm{t})=\frac{1}{g(0)-\mathrm{g}(144)}\frac{1}{C(\mathrm{t})}+\frac{g(144)}{g(144)-\mathrm{g}(0)}$$

代入 C(t)得

$$f(\mathrm{t})=\frac{1}{g(0)-\mathrm{g}(144)}(\frac{1}{C_0+S_0t})+\frac{g(144)}{g(144)-\mathrm{g}(0)} \qquad (1\text{-}3)$$

我们给定 $C_0=0.2$ $S_0=0.01$ $n=1$ ，画出函数图象并将结果与资料[4]中1995 日本阪神地震的经验公式： $e(\mathrm{t})=\mathrm{e}^{-0.037t}$ 进行对比如下图：

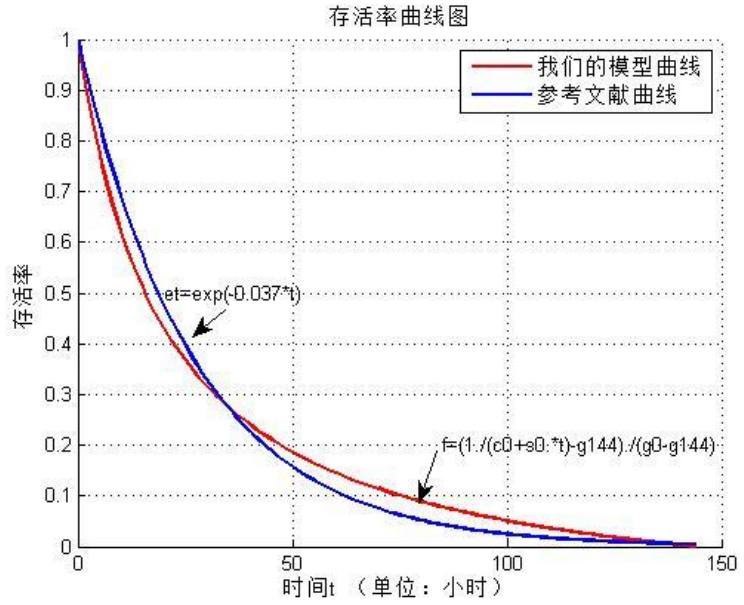


图 5 存活率曲线对比

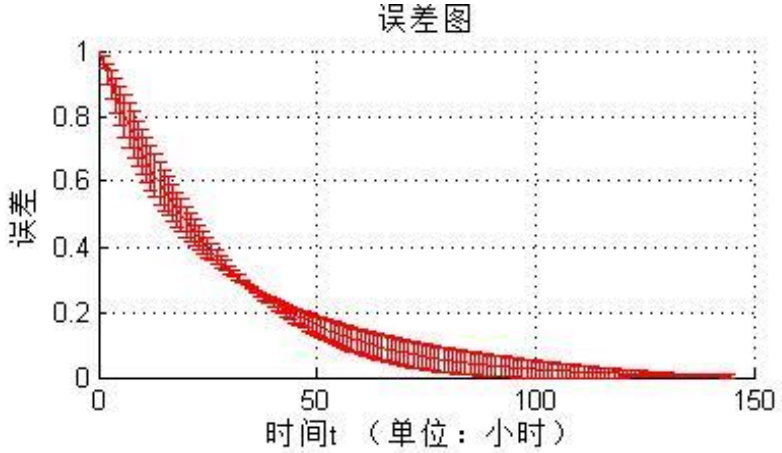


图 6 存活率误差图

5.1.4 误差分析

首先分别计算绝对误差$u=\left|f(\mathrm{t})-\mathrm{e}(\mathrm{t})\right|$ 以及相对误差 $v=\dfrac{\left|f(\mathrm{t})-\mathrm{e}(\mathrm{t})\right|}{e(\mathrm{t})}$，得到绝对误差在（0，0.7）之间，相对误差为 7.72%，在可以允许的范围内，结果是正确且符合实际的。绝对误差和相对误差随时间变化如下图：

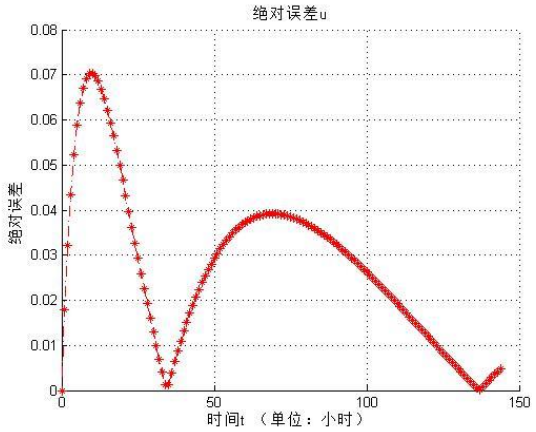


图 7 存活率绝对误差图

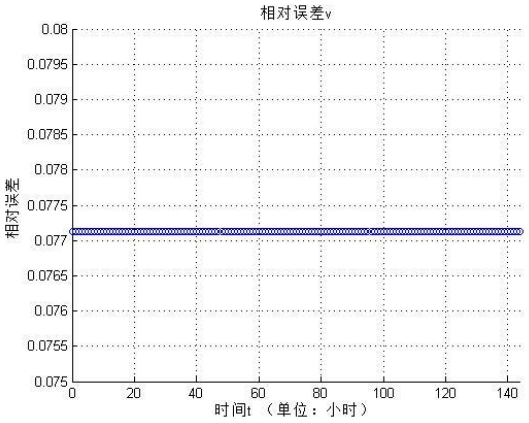


图 8 存活率绝对误差图

5.1.5 模型推广

根据实际情况，灾区不同级别的环境下对于不同体质的人生存率会有所不同[5]，因此实际上S_i 和 n 以及时间 t 都是变量，因此我们考虑更多因素影响用控制变量法对模型进行进一步推广。

首先控制身体素质n=1,研究存活率在环境条件改变时随环境、时间两个变量的变化情况:

$$f(\mathrm{s},\mathrm{t})=\frac{1}{gt(0)-\mathrm{gt}(144)}\frac{1}{C_0+S\times t}+\frac{gt(144)}{gt(0)-\mathrm{gt}(144)}\quad(\mathrm{n}=1)\qquad(1\text{-}4)$$

如下图：

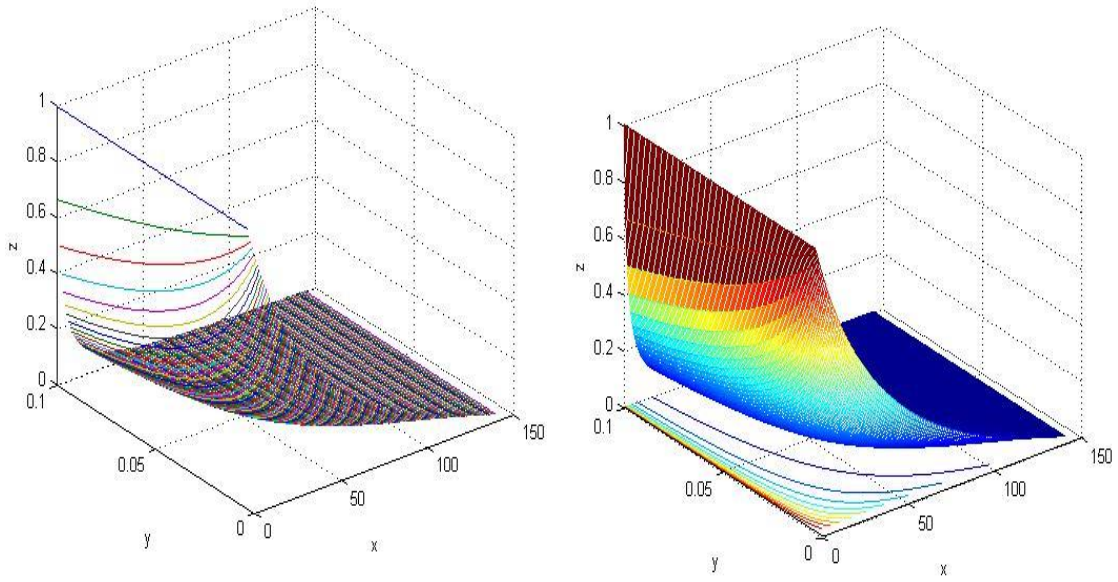


图 9 控制 n 的存活率变化图

接着控制环境参数 s=0.01,研究存活率对于不同身体素质的人随时间变化情况:

$$f(\mathrm{n},\mathrm{t})=\frac{1}{gt(0)-\mathrm{gt}(144)}\frac{1}{(C_0^{\frac{1}{n}}+S_0\times t)^n}+\frac{gt(144)}{gt(0)-\mathrm{gt}(144)}\quad (S_0=0.01\ )\qquad (1\text{-}5)$$

如下图：

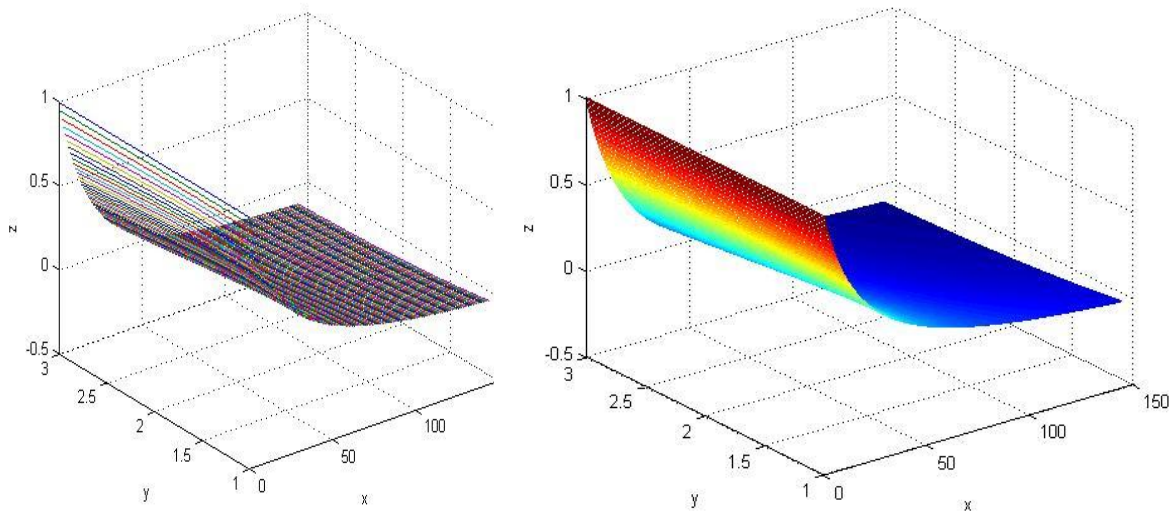


图 10 控制 s 的存活率变化图

5.2 问题二模型的建立与求解

5.2.1 模型的建立：

题目要求设计一种最优的资源分配方案，我们依据附表二所给的各个灾区需求量，各个资源中心的供应量以及从灾区到资源中心所消耗的时间数据，要想要达到资源的最优分配，我们选取分配的时间作为资源分配的量化指标[6]。首先我们选取累计时间作为目标函数，并且设立(0,1)逻辑变量:

$$y_{ij}=\begin{cases}0 & \text{选取从第}i\text{个资源中心运到第}j\text{个灾区}\\ 1 & \text{否则}\end{cases} \qquad (2\text{-}1)$$

同时将供求关系以及前三个配置资源中心强制使用作为相应的约束条件，建立的整数非线性规划模型如下：

$$\min \sum_{i=1}^{13}\sum_{j=1}^{4} t_{ij}y_{ij}$$

$$s.t\begin{cases}\sum_{i=1}^{13} y_{ij}x_{ij}\ge a_j\,(j=1,2,3,4)\\ \sum_{j=1}^{4} y_{ij}x_{ij}\le b_i\,(i=1,2,...,13)\\ \sum_{j=1}^{4} y_{ij}\ge 1(i=1,2,3)\\ y_{ij}=0\text{或}1(i=1,2,...,13;\ j=1,2,3,4)\end{cases} \qquad (2\text{-}2)$$

我们在 Lingo 中进行相应的求解，得到的累计最短时间为 186 小时，平行运送的时间为 27 小时，详细的结果列表如下：

表 1 在 Lingo 中以累计时间为目标函数的选址及运输量结果

资源点 / 灾区	1	2	3	4	5	6	7	8	9	10	11	12	13
A				500				4000			500		
B				2500									
C					200		4000				2800		
D	400	500	800		2800								

由于在实际生活中，对于灾区这种紧急状况，所指派的各个资源中心到不同灾区点是可以同时进行的，由此我们在原有模型的基础上更新目标函数为各个指派段的最长时间以及约束条件使得更加实际化，所建立的最终模型如下：

$$\min \max_{i,j} t_{ij} y_{ij}$$

$$s.t\begin{cases}\sum_{i=1}^{13} y_{ij} x_{ij} \geq a_j (j=1,2,3,4)\\ \sum_{j=1}^{4} y_{ij} x_{ij} \leq b_i (i=1,2,...,13)\\ \sum_{j=1}^{4} y_{ij} \geq 1 (i=1,2,3)\\ y_{ij} \leq x_{ij} \leq M y_{ij} (i=1,2,...,13;\ j=1,2,3,4;\ M\text{是一个很大的数})\\ y_{ij} = 0\text{或}1 (i=1,2,...,13;\ j=1,2,3,4)\end{cases} \quad (2\text{-}3)$$

在这个模型中，我们考虑到了逻辑变量和运输量之间的隐含关系，并且把它显性化，为使得我们的模型所得到的运输结果更具有实际意义。我们增加了一个约束条件 $y_{ij} \leq x_{ij} \leq M y_{ij}$ 表示不会出现运“空车”的情况，即不会得到某条路径不使用但是却会出现运量的情况。

5.2.2 模型的求解：

据此我们同样在 Lingo 中编写相应代码，平均运送时间为 27 小时，得到的详细结果如下所示：

表 2 在 Lingo 中以平行时间为目标函数的选址及运输量结果

资源点 / 灾区	1	2	3	4	5	6	7	8	9	10	11	12	13
A				500				4000			500		
B				2500									
C					200		4000				2800		
D	400	500	800		2800								

同样，考虑到时间以及空间复杂度和数据规模对于算法的影响，我们运用贪婪算法来解决此整数非线性规划问题。贪心算法是指在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，所做出的仅是在某种意义上的局部最优解。

贪心算法[7]的基本思路可以有如下几点：

step 1:建立数学模型来描述问题。

step 2:把求解的问题分成若干个子问题。

step 3.对每一子问题求解，得到子问题的局部最优解。

step 4.把子问题的解局部最优解合成原来解问题的一个解。

由于本题目的运输是可以同时进行的，时间是平行的，具有无后效性，因此运用贪心算法求解此模型是合理的。

通过 matlab 软件，我们最终得到的结果依然是 27 小时。

所得到的结果如下：

表 3 在 Matlab 运用贪心算法中以平行时间为目标函数的选址及运输量结果

资源 / 灾区	1	2	3	4	5	6	7	8	9	10	11	12	13
A						500					4000		
B							4000	1100					200
C	400												3800
D		500	800									3700	

5.2.3 结果分析

比较其中两个程序得到的详细结果，我们发现，虽然得到的结果都是 27 小时，但是 Lingo 和 Matlab 对于同一个模型得到的具体的每一个资源点到每一个灾区点的运输量不同，而且求解出相同最优解所花费的时间也不同，Lingo 求解的时间是 1 分钟 46 秒，Matlab 中运行的时间是 0.019 秒，如此大的时间差距在于两者所使用的算法不同，在 Lingo 中使用的是其内置的算法，而 Matlab 中所使用的是贪心算法。不同的算法影响到的是模型的求解的效率。在 Lingo 中我们得到的是全局最优解，而根据贪心算法的理论，运用贪心算法所得到的是局部最优解，由此我们把贪心算法得到的结果与 Lingo 得到的真正的全局最优解作比较，以检验我们通过贪心算法得到结果的合理性。另外，我们发现在 Lingo 详细运输的结果中，出现了运输“1”个单位这种情况，而在 Matlab 中则没有出现这种现象。我们在这里定义这种现象为“小数”，按照实际情况，在灾区运输这种紧急情况下，只运输“1”单位的量是不具有经济以及现实意义的，这在实际问题的求解中至关重要，具体这两种算法的比较将在下面进行进一步的讨论。

同时，由于本题目较为开放，对于运量并没有太多的限制，这就导致出了最优解一样但是解的结构不同的情况，这从 Lingo 和 Matlab 的结果可以发现，由此我们认为，这道题目存在多组最优解使得目标函数达到最优的情况。

5.3 问题三模型算法的比较与检验

5.3.1 模型的求解

如第二问末尾所述，Lingo 和 Matlab 由于各自的算法不同对于同一个模型得到的具体结果是不同的，下面我们通过扩大数据规模，同时为了加强对比，我们运用自适应权重的 pso 智能算法[8]（以下简称 pso 算法）来求解此模型，设粒子群在一个 n 维空间中搜索，由 N 个粒子组成种群 $X=\{X_1,X_2,...,X_N\}$，其中每个粒子所处的位置 $X_i=\{x_{i1},x_{i2},...,x_{in}\}$ 都表示问题的一个解。粒子通过不断调整自己的位置 x_{id} 来搜索新解。每个粒子都能记住自己搜索到的最优解，记做 p_{id}，以及

整个粒子群经历过的最好的位置，即目前搜索到的最优解，记做 p_{gd}。此外每个粒子都有一个速度，记做 $V_i=\{v_{i1},v_{i2},...,v_{in}\}$，当两个最优解都找到后，每个粒子根据速度位移公式（3-1）来更新自己的速度和位移。

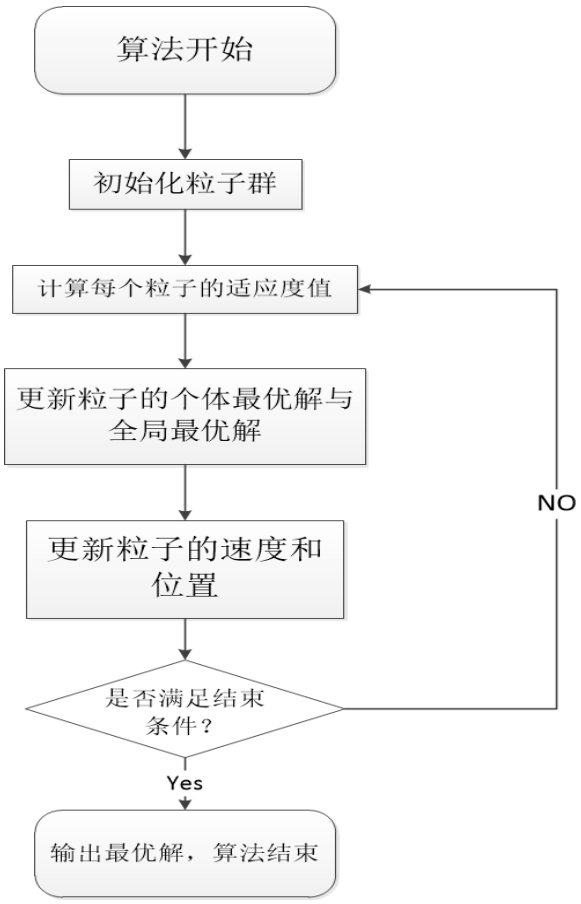
$$\begin{cases} v_{id}(t+1)=\omega v_{id}(t)+\eta_1 rand()(p_{id}-x_{id}(t))+\eta_2 rand()(p_{gd}-x_{gd}(t)) \\ x_{id}(t+1)=x_{id}(t)+v_{id}(t+1) \end{cases} \tag{3-1}$$

式中，$v_{id}(t+1)$ 表示第 i 个粒子在 $t+1$ 次迭代中第 d 维上的速度，$\eta_1\ \eta_2$ 为加速常数，$rand()$ 为 0 到 1 之间的随机数，ω 为惯性权重，为了平衡 pso 算法的全局搜索能力和局部改良能力，我们采用非线性的动态惯性权重系数公式，其表达式为：

$$\omega=\begin{cases} \omega_{\min}-\dfrac{(\omega_{\max}-\omega_{\min})(f-f_{\min})}{f_{avg}-f_{\min}}, f\le f_{avg} \\ \omega_{\max}, f>f_{avg} \end{cases} \tag{3-2}$$

其中 $\omega_{\max}$，$\omega_{\min}$ 分别表示 ω 的最大值和最小值，f 表示微粒当前的目标函数值，f_{avg} 和 $f_{\min}$ 分别表示当前所有微粒的平均目标值和最小目标值。由于这种惯性权重随微粒的目标函数的值而自动改变，故称自适应权重。

我们考虑到 pso 算法生成粒子的方式是一种随机的过程，而这种随机的生成方式往往不能很好的得到我们的最优目标函数值（平行时间），因此，我们采取了贪婪算法与 pso 相结合的方式，得到一种基于贪心的自适应权重的 pso 算法[9]（下面简称 pso），并用这种算法运行求解模型。



图十一 pso 算法图

具体的操作模式如下：

1.扩大灾区点数量

如下所示，我们增加了 4 个灾区，此时一共有 8 个灾区点，13 个资源点。

我们首先在 lingo 中运行，发现所得到的最优解为 27，但是所花费的时间为 2 分 34 秒。

表 4 在 Lingo 中的选址及运输量结果

资源 / 灾区	1	2	3	4	5	6	7	8	9	10	11	12	13
A									1800		2700		
B								4000			1300		
C			800										3400
D		400										4000	600
E				3000	1499								
F					1501				400	2199			
G							3499			801			
H	400					3500			1300				

在 MATLAB 中用贪心算法，可以迅速的求出结果，具体的结果如下：

表 5 贪心算法在 Matlab 中的选址及运输量结果

资源 / 灾区	1	2	3	4	5	6	7	8	9	10	11	12	13
A						500					4000		
B					100	1000	4000						200
C	400												3800
D			800	1900	2300							2900	
E		500		1100								1100	
F										3000			
G						300		4000					
H						1700			3500				

在 MATLAB 中用 pso 算法求得的结果如下：

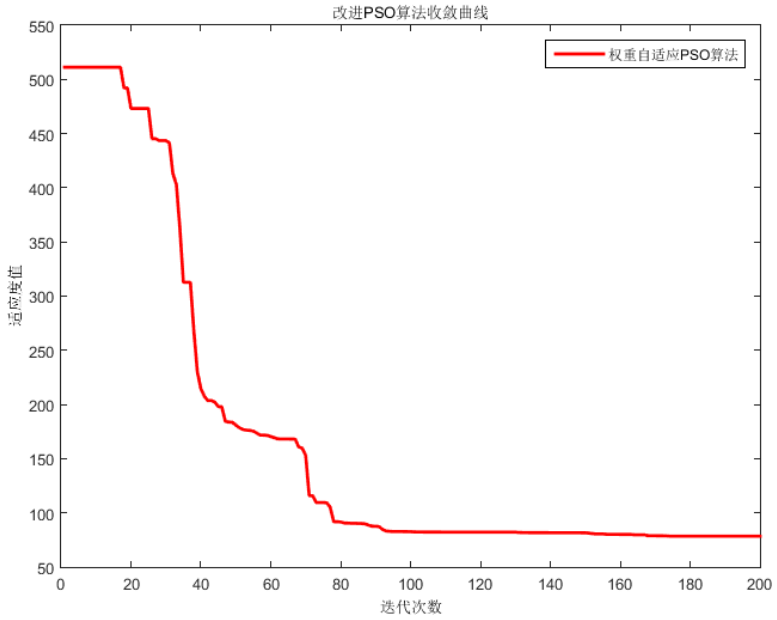


图 13 第一种扩大数据规模的 pso 收敛情况

表 6 pso 算法在 Matlab 中的选址及运输量结果

资源 / 灾区	1	2	3	4	5	6	7	8	9	10	11	12	13
A						500					4000		
B					100	1000	4000						200
C	400												3800
D			800	1900	2300							2900	
E		500		1099						3000		1100	
F						300		4000					
G						1700			3500				

2.扩大资源区数量

如下所示，我们增加了 3 个资源点，此时一共有 4 个灾区点，16 个资源点。

我们首先在 lingo 中运行，发现所得到的最优解为 25，但是所花费的时间为 4 分 31 秒。

表 7 在 Lingo 中的选址及运输量结果

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A			1		1	3500			998							
B	1		1				4000	1298								
C	1	1	1							3000					1196	1
D	202	1	797										400			

在 MATLAB 中用贪心算法求得的最优解依然为 25，所花费的时间为 0.005 的结果如下：

表 8 贪心算法在 Matlab 中的选址及运输量结果

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A						500					4000					
B							1300						4000			
C															3500	700
D	400	500	800									3300				

在 MATLAB 中用 pso 算法求得的结果如下：

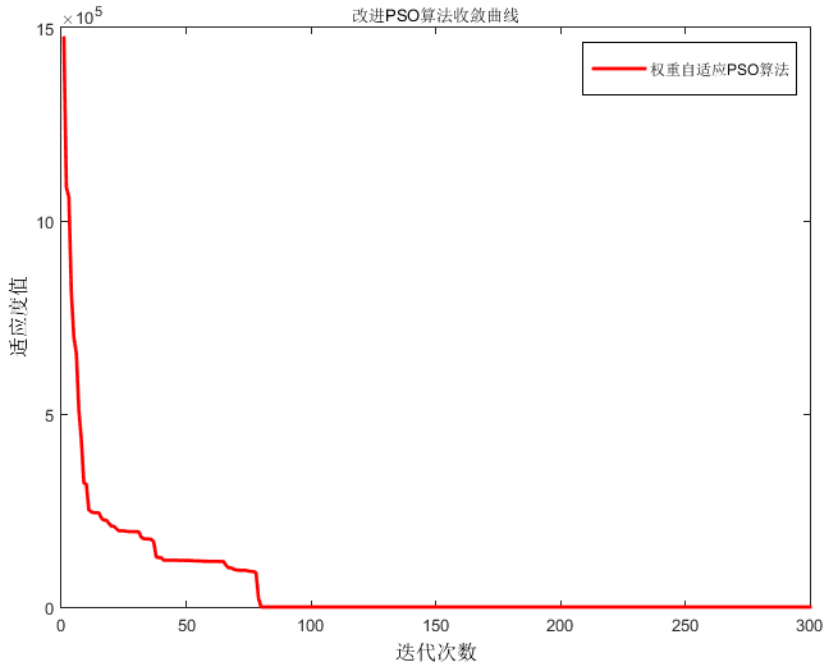


图 14 第二种扩大数据规模的 pso 收敛情况

表 9 pso 算法在 Matlab 中的选址及运输量结果

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A						2252			511	1865						
B							2016	3225			3367					
C	362	134	41											180		405
D		365	747										3962	3455		

3.同时扩大灾区点和资源区数量

如下所示，我们同时增加了 4 个灾区和 3 个资源点，此时一共有 8 个灾区点，16 个资源中心。

我们首先在 lingo 中运行，发现所得到的最优解为 25。

表 10 在 Lingo 中的选址及运输量结果

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A					3000						1500					
B							4000				1299		1			
C										699				1	3500	
D		500	800										3700			
E											1201	3299				
F								3200		900						
G						3500		800								
H	1								2199							3000

在 MATLAB 中用贪心算法求得的结果如下：

表 11 贪心算法在 Matlab 中的选址及运输量结果

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A						500					4000					
B							1300						4000			
C															3500	700
D		500	800				800					2900				
E														4500		
F										3000		1100				
G	400							1600								2300
H						1700			3500							

在 MATLAB 中用 pso 算法求得的结果如下：

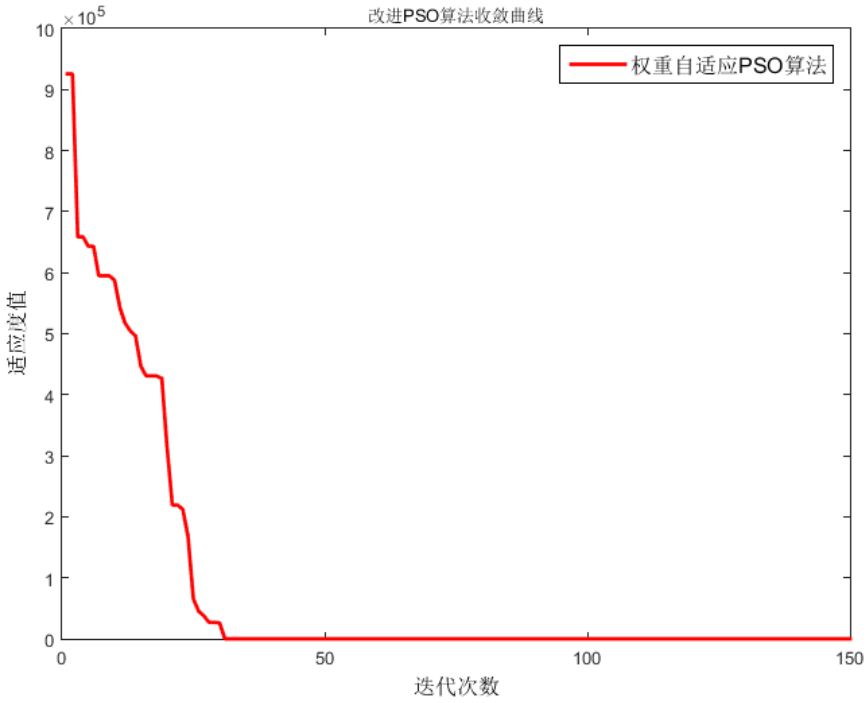


图 15 第三种扩大数据规模的 pso 收敛情况

表 12 pso 算法在 Matlab 中的选址及运输量结果

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A									909	592	1717			3155		
B			338			604	3086	546			346			1017		
C	41		65							632					3472	
D		215	135						2302			2476		217		
E		175		884	1799						1181	1460	610			
F								927	93	1418	125	61				
G							906	1512								
H	333					1705							3383			2580

综上所述，我们归纳这三种规模的数据为一张表格，将 Lingo 得到的运输时间结果作为最佳目标函数值，通过对比检验相同模型不同算法下的结果，时间效率，以及是否更加具有现实意义，进一步确定模型算法的合理性。

表 13 不同数据规模不同算法的运输结果比较

是否出现“小数”	Lingo 内置遍历算法	贪婪算法（Matlab 实现）	pso 算法（Matlab 实现）
扩大灾区数	有	无	无
扩大资源中心数	有	无	无
同时扩大灾区资源点数	有	无	无

表 14 不同数据规模不同算法的时间效率比较

所花时间（s）	Lingo 内置遍历算法	贪婪算法（Matlab 实现）	pso 算法（Matlab 实现）
扩大灾区数	2 分 34 秒	0.000	2 分钟
扩大资源中心数	4 分 31 秒	0.005	1 分钟
同时扩大灾区资源点数	2 分 14 秒	0.007	1 分钟

在这里，我们需要注意的是，所花的时间并不能完全全面的体现出一个算法的运行时间效率，所以我们查找了相关的资料，确定出了对于本次模型的三种算法的相应的时间复杂度，并与所花时间结合起来，对一个算法的时间效率进行评价。Lingo 所运行的时间并不在比较范围之内，我们在这里重点比较的是数据规模的不同对于贪婪算法和 pso 算法（在相同运行环境 Matlab 下）的影响。下表给出了相应的算法时间复杂度的定量化标准。

算法	Lingo 中的遍历	Matlab 中的贪婪	Matlab 中的 pso
时间复杂度	高	底	较低

表 15 不同数据规模不同算法的时间（目标函数）结果比较

目标函数值（h）	Lingo 内置遍历算法	贪婪算法（Matlab 实现）	pso 算法（Matlab 实现）
扩大灾区数	27	50	50
扩大资源中心数	25	27	26
同时扩大灾区资源点数	25	30	28

我们发现，仅仅从时间的复杂度来看，贪心算法是这三种当中时间复杂度最小的，但是也是最容易陷入局部最优的，比如在只扩大灾区数目的情况，贪心算法求到的结果与真正的全局最优解相差甚远，而时间复杂度相对较高的 pso 算法却可以得到一个相对于贪心算法更好的近似结果，由此，我们所确定的最终算法即是以贪心算法为基础，以 pso 算法为核心的算法。

综上所述，我们可以得出一个结论：对于灾区这种紧急突发性事件情况下，由于决策者需要快速做出一种资源分配方案，因此使用遍历的方式在很大程度上不被接受。如果灾区以及所安排的资源配送中心规模较小，则可以使用贪婪算法求得局部最优解，但是如果区域规模在很大的情况下，我们更加希望用 pso 算法求得一个近似最优解。由于时间关系，我们只进行了一次扩大数据规模以此来检验我们的模型和算法。

5.3.2 结果分析

从结果上，我们不难得一下出结论：

1. Lingo 内置算法：从运输量结果是否出现小数来看，这种 Lingo 内置算法只是考虑到要使得总的目标函数值及运送的平行时间要最小，此种算法是不具有经济以及现实意义的，而且这种遍历形式的算法所导致的求解时间也较长，但是由于此种算法一定可以得到全局最优解，因此对于其他的算法求解结果的比较与

检验具有很大的帮助。

2.贪婪算法（Matlab 实现）：从时间效率以及目标函数值（平行时间）的角度来看，与遍历算法相比较，贪婪算法是一种效率相对于较高的算法，对于数据规模的扩大，依然能在时间效率较高的情况下得到最优解，但是得到的只是局部最优解，当数据规模扩大之后并不能达到全局最优解。

3.pso（Matlab 实现）：这种算法的时间复杂度是较小的，但是所得到的结果相对于最优解来说也仅仅是一个局部最优解，对于灾区运输这种紧急的情况而言，时间效率目标是首要目标，因此在大规模数据运算中作为一种智能算法要相对优于贪婪算法。

4.我们知道，仅仅从时间的复杂度来看，贪心算法是这三种当中时间复杂度最小的，但是也是最容易陷入局部最优的，比如在只扩大灾区数目的情况，贪心算法求到的结果与真正的全局最优解相差甚远。而时间复杂度相对较高的 pso 算法却可以得到一个相对于贪心算法更好的近似结果，由此，我们所确定的最终算法即是以贪心算法为基础，以 pso 算法为核心的算法。

.综上所述，我们可以得出一个结论：对于灾区这种紧急突发性事件情况下，由于决策者需要快速做出一种资源分配方案，因此使用遍历的方式在很大程度上不被接受。如果灾区以及所安排的资源配送中心规模较小，则可以使用贪婪算法求得局部最优解，但是如果区域规模在很大的情况下，我们更加希望用贪心算法与 pso 算法相结合的方式求得一个近似最优解。由于时间关系，我们只进行了一次扩大数据规模以此来检验我们的模型和算法。多组数据规模并未在文中实施。

六、 模型的评价和推广

6.1.模型优点：

1.问题一所得到的存活率与时间的关系是建立在伤亡状态函数的基础上，并且与现实中的存活率“两期”变化得到，具有一定的理论基础以及现实意义。

2.问题一我们通过设置不同的参数观察了存活率与时间的变化，较为准确的的反应除了不同参数间接性的对于存活率的影响，同时也较为全面的反应出存活率与时间的关系。

3.问题二所建立的模型引用了逻辑变量，并且在模型中体现出了运输量变量以及逻辑变量的联系，具有一定的严谨性。

4.问题三在问题二的基础上增加了数据规模，并且采用的是逐步递进，控制递进的方式，从不同角度探讨了不同算法之间的优点和缺点，并且最终得到一个较为可靠的结论。

6.2.模型缺点：

1.问题一并未考虑到他人因素（如救援）对于个人存活率的影响，与实际的灾区存活率状况还有一定差距，同时具有一定的主观性。

2.问题二所编写的程序是与所花费的时间以及个人电脑的配置是存在一定关系的，用时间长度反应时间效率有些不准确。

3.问题三所设定的数据规模只有一种情况，在实际中应该存在多组数据规模情况。

6. 3 模型的推广与改进：

1.问题 1 可以与真实数据进一步做回归分析，使得模型更加准确可靠

2.问题三中可以扩大多组数据的规模以更好的说明数据规模的变化影响到了算法的效率，同时效率可以给出一个时间复杂度的表达式以更好的说明算法的优劣点，只扩大了一次的数据规模难免会收到偶然性影响。

3 问题二模型可以与运输的实际成本相结合，能够更加符合实际状态。

七、参考文献

[1]高建国. 地震应急期的分期[J]. 灾害学, 2004, 01:13-17.

[2]周素琴,郭子雄. 地震人员伤亡的动态评估[J]. 华侨大学学报(自然科学版),2004,01:54-57.

[3]赵振东,林均歧,钟江荣,郑向远,王毅超. 地震人员伤亡指数与人员伤亡状态函数[J]. 自然灾害学报,1998,03:91-97.

[4]刘志鹏. 地震伤亡发生分析与预测[D].第二军医大学,2013.

[5]高娜, 聂高众. 地震应急救灾效能研究[J]. 灾害学, 2015, 02:158-161.

[6]乔鹏亮. 地质灾害下区域应急物流配送网络研究[D]. 兰州理工大学, 2009.

[7] 刘志帅, 仝凌云, 魏利鹏, 朱凯, 茜晓立. 基于贪婪算法的货位优化模型[J]. 物流科技, 2013, 09:99-101.

[8]谈文芳, 赵强, 余胜阳, 肖人彬. 改进粒子群优化算法求解任务指派问题[J]. 计算机应用, 2007, 12:2892-2895.

[9]王俊伟, 汪定伟. 粒子群算法中惯性权重的实验与分析[J]. 系统工程学报, 2005, 02:194-198.

附录

第一问matlab代码：

```
clear all;close all;
s0=0.01;
n=1;
c0=0.2;
t=0:144;
ct=c0+s0.*t
gt=1./ct
g0=1./c0
c144=c0+s0*144
g144=1./c144
f=(1./(c0+s0.*t)-g144)./(g0-g144);
figure(1)
plot(t,f,'r','linewidth',2)
hold on
et=exp(-0.037*t)
```

```
plot(t,et,'b','linewidth',2)
xlabel('时间t （单位：小时）')
ylabel('存活率')
legend('我们的模型曲线','参考文献曲线')
title('存活率曲线图')
grid on
text(85,0.2,'f=(1./(c0+s0.*t)-g144)./(g0-g144)')
text(20,0.5,'et=exp(-0.037*t)')
box off
figure(2)
u=abs(et-f)
v=abs(et-f)/et
plot(t,u,'r-.*')
title('绝对误差u')
xlabel('时间t （单位：小时）')
ylabel('绝对误差')
grid on
box off
figure(3)
!plot(t,v,'linewidth',5)
plot(t,v,'bo','MarkerSize',5)
title('相对误差v')
xlabel('时间t （单位：小时）')
ylabel('相对误差')
axis([0 144 0.075 0.08])
grid on
zoom on
box off
figure(4)
errorbar(t,et,f-et)
title('误差图')
xlabel('时间t （单位：小时）')
ylabel('误差')
axis([0 150 0 1])
grid on
box off

% 相同体质的n=1的人在不同环境下存活率随环境、时间变化
figure(5)
s=0.004:0.094/144:0.1
T=0:1:144
[X,Y]=meshgrid(T,s)
% ctt=c0+s.*T
% gtt=1./ctt
```

```
% gtt0=1./c0
% ctt144=c0+s*144
% gtt144=1./ctt144
% !ff=(1./(c0+s.*T)-gtt144)./(gtt0-gtt144)
ctt=c0+Y.*X
gtt=1./ctt
gtt0=1./c0
ctt144=c0+Y*144
gtt144=1./ctt144

Z=(1./(c0+Y.*X)-gtt144)./(gtt0-gtt144)

plot3(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

figure(6)
mesh(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

figure(7)
meshc(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

figure(8)
surf(Z)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
% 环境因素s=0.01研究人的体质不同，人的存活率随体质、时间变化
figure(9)
nn=1:2/144:3
```

```
t=0:144
[A,B]=meshgrid(t,nn)
ct=c0+s0.*t
gt=1./ct
gt0=1./c0
ct144=c0+s0*144
gt144=1./ct144

C=(1./((c0.^(1./B)+s0.*A).^B)-gt144)./(gt0-gt144)

plot3(A,B,C)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

figure(10)
mesh(A,B,C)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

figure(11)
meshc(A,B,C)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

figure(12)
surf(C)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off
% 环境一定伤亡程度随体质和时间变化
figure(13)
s0=0.01
n=1:2/144:3
```

```
% ct=(c0.^(1/n)+s0.*t).^(1/n)
[D,E]=meshgrid(t,n)
F=(c0.^(1./E)+s0.*D).^(1./E)
plot3(D,E,F)

figure(14)
meshc(D,E,F)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

figure(15)
surf(F)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

% 体质n=1一定，不同环境下，伤亡指数随时间和环境因素的变化
figure(16)
s=0.004:0.094/144:0.1
[G,H]=meshgrid(t,s)
I=c0+H.*G
plot3(G,H,I)
grid on

figure(17)
meshc(G,H,I)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off

figure(18)
surf(I)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
box off
```

```
figure(19)
plot(t,ct,'linewidth',3)
xlabel('时间t （单位：小时）')
ylabel('Ct')
axis([0 80 0 1])
grid on
box off
figure(20)
plot(t,gt,'linewidth',3)
grid on
box off
xlabel('时间t （单位：小时）')
ylabel('gt')
grid on
box off
```

第二问累积性Lingo代码：

```
model:
sets:
source/1..13/:a;
need/A..D/:b;
links(source,need):t,x,y;
endsets
data:
a=400 500 800 3000 3000 3500 4000 4000 3500 3000 4000 4000 4000;
b=4500 5300 4200 5000;
t=40 25 17 20
40 35 25 25
20 25 22 15
35 30 45 30
25 50 30 40
20 28 56 30
30 20 45 30
30 25 55 35
25 45 30 28
26 34 22 32
19 23 42 30
40 50 30 27
40 20 20 25;
enddata
!min=@sum(source(i):sum(need(j):y(i,j)*t(i,j)));
min=@sum(links:y*t);
@for(need(j):@sum(source(i):y(i,j)*x(i,j))>=b(j));
@for(source(i):@sum(need(j):y(i,j)*x(i,j))<=a(i));
```

```
@for(source(i)|i #LE# 3:@sum(need(j):y(i,j))>=1);
@bin(y(1,1));@bin(y(1,2));@bin(y(1,3));@bin(y(1,4));
@bin(y(2,1));@bin(y(2,2));@bin(y(2,3));@bin(y(2,4));
@bin(y(3,1));@bin(y(3,2));@bin(y(3,3));@bin(y(3,4));
@bin(y(4,1));@bin(y(4,2));@bin(y(4,3));@bin(y(4,4));
@bin(y(5,1));@bin(y(5,2));@bin(y(5,3));@bin(y(5,4));
@bin(y(6,1));@bin(y(6,2));@bin(y(6,3));@bin(y(6,4));
@bin(y(7,1));@bin(y(7,2));@bin(y(7,3));@bin(y(7,4));
@bin(y(8,1));@bin(y(8,2));@bin(y(8,3));@bin(y(8,4));
@bin(y(9,1));@bin(y(9,2));@bin(y(9,3));@bin(y(9,4));
@bin(y(10,1));@bin(y(10,2));@bin(y(10,3));@bin(y(10,4));
@bin(y(11,1));@bin(y(11,2));@bin(y(11,3));@bin(y(11,4));
@bin(y(12,1));@bin(y(12,2));@bin(y(12,3));@bin(y(12,4));
@bin(y(13,1));@bin(y(13,2));@bin(y(13,3));@bin(y(13,4));
end
```

第二问考虑时间同步性 Lingo 代码：

```
model:
sets:
source/1..13/:a;
need/A..D/:b;
links(source,need):t,x,y;
endsets
data:
a=400 500 800 3000 3000 3500 4000 4000 3500 3000 4000 4000 4000;
b=4500 5300 4200 5000;
t=40 25 17 20
40 35 25 25
20 25 22 15
35 30 45 30
25 50 30 40
20 28 56 30
30 20 45 30
30 25 55 35
25 45 30 28
26 34 22 32
19 23 42 30
40 50 30 27
40 20 20 25;
enddata
!min=@sum(source(i):sum(need(j):y(i,j)*t(i,j)));
min=@max(source(i):@max(need(j):y(i,j)*t(i,j)));
@for(need(j):@sum(source(i):y(i,j)*x(i,j))>=b(j));
@for(source(i):@sum(need(j):y(i,j)*x(i,j))<=a(i));
```

```
@for(source(i)|i #LE# 3:@sum(need(j):y(i,j))>=1);
@bin(y(1,1));@bin(y(1,2));@bin(y(1,3));@bin(y(1,4));
@bin(y(2,1));@bin(y(2,2));@bin(y(2,3));@bin(y(2,4));
@bin(y(3,1));@bin(y(3,2));@bin(y(3,3));@bin(y(3,4));
@bin(y(4,1));@bin(y(4,2));@bin(y(4,3));@bin(y(4,4));
@bin(y(5,1));@bin(y(5,2));@bin(y(5,3));@bin(y(5,4));
@bin(y(6,1));@bin(y(6,2));@bin(y(6,3));@bin(y(6,4));
@bin(y(7,1));@bin(y(7,2));@bin(y(7,3));@bin(y(7,4));
@bin(y(8,1));@bin(y(8,2));@bin(y(8,3));@bin(y(8,4));
@bin(y(9,1));@bin(y(9,2));@bin(y(9,3));@bin(y(9,4));
@bin(y(10,1));@bin(y(10,2));@bin(y(10,3));@bin(y(10,4));
@bin(y(11,1));@bin(y(11,2));@bin(y(11,3));@bin(y(11,4));
@bin(y(12,1));@bin(y(12,2));@bin(y(12,3));@bin(y(12,4));
@bin(y(13,1));@bin(y(13,2));@bin(y(13,3));@bin(y(13,4));
end
```

第二问贪心算法 matlab：

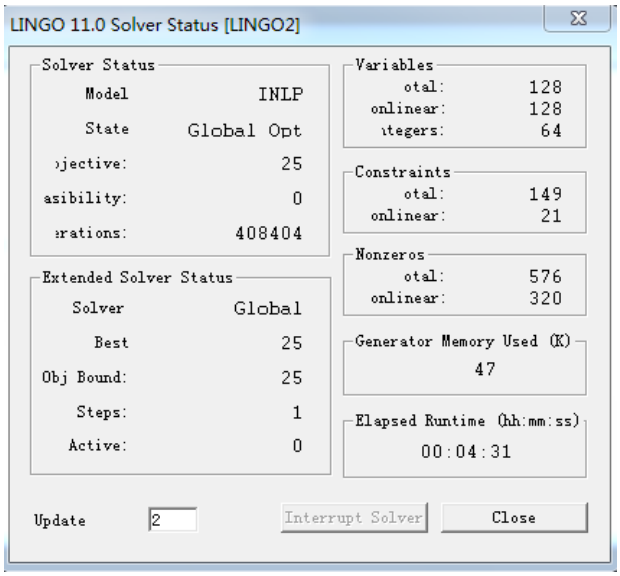
```
    juzhen=0;
t=0;
while  1
        if  need(:)==0
                break;
        else
                c=min(min(timer));     %找到矩阵中的最小值
                cposition=find(timer==c);%  找到 c 在矩阵中的位置，其值是按列排
列得到的
                cposition=cposition(size(cposition,1));%找值最大的地方
                yushu=mod(cposition,4);%第几行
                n=cposition/4;%第几个配送中心
                n=floor(n);
                if  yushu==0
                        m=demand(n);%第 n 个配送中心的供应
                        if  need(4)>0
                                if  need(4)>m
                                need(4)=need(4)-m;%供不应求
                                demand(n)=0;%供为 0
                                juzhen(4,n)=m;
                                t=[t  c];%加时间
                                else%供过应求
                                        demand(n)=demand(n)-need(4);%供减少
                                        juzhen(4,n)=need(4);
                                        need(4)=0;%求为 0
                                        ju(4,n)=demand(n);
                                        t=[t  c];%加时间
```

```
                    end
                end
                timer(4,n)=99999;%排除该点
            else
                n=n+1;%第几个配送中心
                m=demand(n);
                if  need(yushu)>0
                    if  need(yushu)>m
                    need(yushu)=need(yushu)-m;%供不应求
                    demand(n)=0;%供为 0
                    juzhen(yushu,n)=m;
                    t=[t  c];%加时间
                    else  %供过应求
                        demand(n)=demand(n)-need(yushu);%供减
少

                        juzhen(yushu,n)=need(yushu);
                        need(yushu)=0;%求为 0
                        ju(4,n)=demand(n);
                        t=[t  c];%加时间
                    end
                end
                timer(yushu,n)=99999;%排除该点
            end
        end
end
```

第三问 Lingo 代码：

1. 增加资源点数量



代码及结果：

```
!改变资源点数目,4 个灾区 16 个资源中心
    model:
```

```
sets:
source/1..16/:a;
need/A..D/:b;
links(source,need):t,x,y;
endsets
data:
a=400 500 800 3000 3000 3500 4000 4000 3500 3000 4000 4000 4000 4500 3500 3000;
b=4500 5300 4200 5000;
t=40 25 17 20
40 35 25 25
20 25 22 15
35 30 45 30
25 50 30 40
20 28 56 30
30 20 45 30
30 25 55 35
25 45 30 28
26 34 22 32
19 23 42 30
40 50 30 27
40 20 20 25
24 26 23 45
46 54 16 49
37 37 17 50;
enddata
!min=@sum(source(i):sum(need(j):y(i,j)*t(i,j)));
min=@max(source(i):@max(need(j):y(i,j)*t(i,j)));
!min=@sum(links:y*t);
@for(need(j):@sum(source(i):y(i,j)*x(i,j))>=b(j));
@for(source(i):@sum(need(j):y(i,j)*x(i,j))<=a(i));
@for(links(i,j):x(i,j)>=y(i,j));
@for(links(i,j):x(i,j)<=10000000*y(i,j));
@for(links:@bin(y));
```

第三问贪心算法求解代码：

```
k=length(need)
juzhen=0;
t=0;
while 1
    if need(:)==0
        break;
    else
        c=min(min(timer));   %找到矩阵中的最小值
        cposition=find(timer==c);%  找到 c 在矩阵中的位置，其值是按列排列得到的
        cposition=cposition(size(cposition,1));%找值最大的地方
```

```
            yushu=mod(cposition,k);%第几行
            n=cposition/k;%第几个配送中心
            n=floor(n);
            if yushu==0
                m=demand(n);%第 n 个配送中心的供应
                if need(k)>0
                    if need(k)>m
                    need(k)=need(k)-m;%供不应求
                    demand(n)=0;%供为 0
                    juzhen(k,n)=m;
                    t=[t c];%加时间
                    else%供过应求
                        demand(n)=demand(n)-need(k);%供减少
                        juzhen(k,n)=need(k);
                        need(k)=0;%求为 0
                        ju(k,n)=demand(n);
                        t=[t c];%加时间
                    end
                end
                timer(k,n)=99999;%排除该点
            else
                n=n+1;%第几个配送中心
                m=demand(n);
                if need(yushu)>0
                    if need(yushu)>m
                    need(yushu)=need(yushu)-m;%供不应求
                    demand(n)=0;%供为 0
                    juzhen(yushu,n)=m;
                    t=[t c];%加时间
                    else %供过应求
                        demand(n)=demand(n)-need(yushu);%供减少
                        juzhen(yushu,n)=need(yushu);
                        need(yushu)=0;%求为 0
                        ju(k,n)=demand(n);
                        t=[t c];%加时间
                    end
                end
                timer(yushu,n)=99999;%排除该点
            end
        end
    end
```

Profile Summary

Generated 12-Aug-2016 16:33:08 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
tanxin	1	0.005 s	0.005 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

第二种扩大方式在 Matlab 中的运行时间

Profile Summary

Generated 12-Aug-2016 16:03:25 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
tanxin	1	0 s	0.000 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

第一种扩大方式在 Matlab 中的运行时间

Profile Summary

Generated 12-Aug-2016 16:38:23 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
tanxin	1	0.007 s	0.007 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

第三种扩大方式在 Matlab 中的运行时间

表1 需求点应急资源需求量 (单位：吨)

需求点	A	B	C	D	E	F	G	H
数量	4500.00	5300.00	4200.00	5000.00	4500.00	4100.00	4300.00	5200.00

表2 配送中心某类资源储备量

资源点	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
储备规模	400.00	500.00	800.00	3000.00	3000.00	3500.00	4000.00	4000.00	3500.00	3000.00	4000.00	4000.00	4000.00	4500.00	3500.00	3000.00

表3 需求点与资源点的最短运输时间

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	40.00	40.00	20.00	35.00	25.00	20.00	30.00	30.00	25.00	26.00	19.00	40.00	40.00	24.00	46.00	37.00
B	25.00	35.00	25.00	30.00	50.00	28.00	20.00	25.00	45.00	34.00	23.00	50.00	20.00	26.00	54.00	37.00
C	17.00	25.00	22.00	45.00	30.00	56.00	45.00	55.00	30.00	22.00	42.00	30.00	20.00	23.00	16.00	17.00
D	20.00	25.00	15.00	30.00	40.00	30.00	30.00	35.00	28.00	32.00	30.00	27.00	25.00	45.00	49.00	50.00
E	20.00	24.00	25.00	26.00	27.00	34.00	36.00	35.00	34.00	41.00	21.00	23.00	22.00	23.00	26.00	27.00
F	35.00	41.00	32.00	41.00	22.00	31.00	32.00	22.00	16.00	17.00	23.00	22.00	30.00	33.00	31.00	40.00
G	18.00	34.00	34.00	56.00	34.00	22.00	25.00	20.00	17.00	18.00	34.00	37.00	43.00	29.00	54.00	19.00
H	19.00	36.00	34.00	34.00	43.00	23.00	30.00	45.00	15.00	35.00	46.00	32.00	28.00	55.00	51.00	20.00