浙江大学第十七届
大学生数学建模竞赛

2019 年 5 月 7 日—5 月 17 日

编号　　3922

题目　　A√　　B

（在所选题目上打勾）

	参赛队员 1	参赛队员 2	参赛队员 3
姓名	王轩	彭官妍	张天哲
学号	3170101524	3160104633	3170102266
院（系）	地球科学学院	计算机科学与技术学院	竺可桢学院
专业	大气科学	软件工程	交叉创新平台
手机	18888924865	18072737205	18888920997
Email	3170101524@zju.edn.cn	3160104633@zju.edu.cn	3170102266@zju.edu.cn

浙江大学本科生院

浙江大学数学建模实践基地

追踪目标数量最大化策略

摘要

本文根据题目所给条件及预设情景，针对一维单探测器问题设计了一种最大化追踪目标的策略，并推广至多追踪器及三维情况。通过假设探测器可同时探测多个目标，对模型进行了扩展。

首先，考虑到无论最优解如何，所有目标必是二次追踪时段不冲突的，而在首次追踪开始时间不确定而二次区间固定的情况下，二次追踪区间能互相兼容的最大数量和相应解集会对后续求解，带来缩小搜索范围的便利和避免暴力枚举的做法，因此考虑二次追踪时间段尽可能不冲突安排，处理一个单因素活动安排资源调度问题。

在该步，以二次追踪结束时刻最早为选取原则，使用贪心算法获得 1 个二次追踪下兼容的目标集sec_0，作为扩展解集所依据的初始解，根据sec_0将其中包含的每一个目标tar_i在其邻近区域$\mathrm{seq} \in [tar_i, tar_{i+1}]$寻找编号为 seq 的可替换项，并将二次追踪时间区域在某一个时间区间附近的目标都放进同一分区进行存储。各个分区的界限即为原始解中各个目标项的二次追踪结束时间，并由这若干个分区维护所有的贪心算法最优解的解集。

在保证二次追踪时间可兼容性的情况下，每一次用贪心算法与其扩展解得到的最优解池当中随机取解，作为参数参与下一步根据当前给定目标序列下，选取最大可追踪目标数的计算，在这一步中遇到判断为不可探测的目标时，我们将使用替换选项的方式，在当前目标所在分区中按一定顺序遍历选择新的替代目标，采取类似回溯的方式尽可能在少丢失目标项的情况下进行可放置的目标补偿，以此达到尽可能保留和安排更多的目标被探测，并且在生成扩展的同时，每次操作都利用洗牌算法随机打乱分区池内的目标顺序，使得在选择替换项时引入更多的随机性。

针对一维多探测器问题，首先重复使用 N 次单探测器情况下的算法，每一次循环计算都重新生成若干分区，对未被追踪的目标进行重新分段，而每次所探测的初始目标集合是除去之前被探测目标的剩余目标的集合。

所有的探测器时间调度安排结束以后，组合 N 个探测器的空闲区间，再对剩余目标集使用初始算法。结合两次的解集，获得最终解集。

对三维情形下探测器追踪时段的分配问题，采用与一维情形相同的算法。考虑预设的目标抛物模型，采用柱坐标，置探测器于目标当前秒内起始和结束位置中点处。

关键词：贪心算法；解集扩展；洗牌算法；目标替换

0 问题重述

（1）问题背景

在目标追踪的过程中，不同追踪器在不同时段选择不同的追踪目标将在很大程度上影响追踪成功的目标个数。本文研究的目标追踪问题包含N个目标T_j和M部相同的追踪器。

M个相同型号的追踪器每部有一定的探测范围，其探测中心可根据需要随时变化。对目标T_j，若在存在s_j，使得在$[s_j, s_j + p_j) \cup [a_j, b_j]$时段内的任意时刻，目标的位置位于某部追踪器的探测范围内，则成该目标被成功追踪。不同时刻用于追踪某个目标的追踪器可以不同，但同一追踪器在某一时刻只能追踪一个目标

（2）两种具体情形

1）所有目标的运动轨迹在同一条直线上。每个目标从某一时刻开始至某一时刻结束，以某一位置为起点，以一定速率向同一方向做匀速直线运动。不同目标的运动开始时刻、结束时刻、起点位置、速率大小可能不同。每部追踪器的探测中心也在该直线上，探测范围为一以探测中心为中点、长度固定的闭区间。

2）所有目标的运动轨迹可近似视作在某一时刻、某一高度，将球向某一方向以某一速度抛出后所形成的的轨迹。不同目标对应的抛掷时刻、高度、速率和方向未必相同。追踪器的探测范围为一以探测中心为球心、半径固定的球面及其内部。

（3）任务要求

1）针对$\mathrm{M} = 1$和M为某个大于 1 的固定常数（通常不超过 5）两种情形，确定每部追踪器任意时刻探测中心的位置，以及部分目标首次开始追踪时刻s_j，使得被成功追踪的目标数量尽可能多。

2）对附件给出的情形（1）的数据，分别计算$\mathrm{M} = 1$和$\mathrm{M} = 5$时的相应结果，根据要求填写在三个表单的相应位置。

1 问题分析

本题需要我们根据附件所给出的物体运动时间和两次追踪时间，在两种具体运动情形下，给出探测器探测中心的调度方案，使得被成功追踪的目标数量尽可能多。

探测时间安排分为两种情况，探测器数量分别为 1 和 5，在问题一中解决。

对于问题一，分析如下：

若探测器集合用M_s表示，其数量为M，其中的元素使用$m_i, i = 1,2,\dots,M$，表示。

$\mathrm{M} = 1$：

观察得到，对同一目标，其需要被追踪时段由两部分组成，首次追踪时段：$[s_j, s_j + p_j)$，二次追踪时段：$[a_j, b_j]$，问题就转化为，在探测器m_1可调度的总时间$[0, \mathrm{T}], \mathrm{T} = \max(b_j), j = 1,2,\ldots,M$，为了达到上述目标，计划先利用贪心算法对二次追踪这一固定时间段进行安排，保证选择的集合中二次追踪时间之间互相不冲突，减少选择目标时的影响因素。随后根据二次追踪时间段互相兼容的最大数量和解集特征将目标分成若干分区，分区数等于二次追踪时间安排的解的个数，每个分区存放可以用于替换原解中对应项的目标编号。

在此基础上，每次构造一个待选目标序列并按照一定的优先放置原则设置可追踪目标的首次追踪开始时间。

$\mathrm{M} = 5$：

在多个探测器情况下，优先分配探测器完整追踪同一个目标的两段追踪时间，即将$\mathrm{M} = 1$的情况进行推广，重复其过程 5 次，每一次所使用的目标初始集合是除去上一次被探测点剩余的目标组成的集合。

对于最后剩下的目标集合Rem，我们在之前的 5 次区间填充工作完成之后维护一个探测器剩余空闲区间集合V，从Rem中找出能完整填入这些空闲空间集合的目标，使其被多个探测器组合探测。

最后的解集即为上述解集之和。

对于问题二，建立运动模型并结合时段安排结果求解。

（1）直线运动情况下探测中心坐标的计算：

观察附件所给数据和条件，探测器的探测区间长度为 10，其中心位于探测区间中点，可以随时变化，而物体最大速度为 10 个长度单位每秒，若物体运动通过记录每秒其坐标的变化来表示，则在一秒内物体位移不会超过 10 个长度单位。

若探测器从一开始探测时，物体坐标恰好为探测器探测区间最左端，即物体坐标与探测中心坐标的距离为 5，则该秒内物体能持续被探测器探测，一秒后，探测中心将位于物体新的记录坐标的右端，距离其 5 个单位长度的地方。

具体示意图将在下文模型中给出。

（2）圆周运动情况下探测中心坐标的计算

对于圆周运动，在三维空间中，由于初始所给数据仅有直线运动的初始坐标，在圆周运动情形下，探测器中心坐标与探测目标坐标重合，并随之变化，则可以保证时刻探测到目标。

具体的表示方法将在下文模型建立中给出。

2 模型假设

1、假设三维情况下，目标从同一轴不同高度沿不同方向抛出，且高度值即为所给初始数据中，直线运动情形下的初始坐标值。

2、假设一秒内物体做抛物线运动时，不会运动超出探测器探测范围。

3、假设首次追踪发生在二次追踪前且不能与二次追踪有时间重合，即$Sj + Pj \leq aj$。

3 符号说明

符号	符号意义
S_j	目标 j 首次追踪开始时间
p_j	首次追踪时长
a_j	目标 j 二次追踪开始时间
b_j	目标 j 二次追踪结束时间
seq_j	按b_j大小自小到大排列的目标
Sec	仅考虑二次追踪，以贪心法所得目标集
tar_j	目标集sec中元素
Rep_j	tar_j可替换项集合
O_j	第j次探测的初始目标集
M_i	第i个探测器
Card(S)	集合 S 中的元素个数
Slot[0,200]	规定探测器可用时间区间为[0,200]
subs	从Rep_j中选出的可替换项

4 模型的建立和求解

（1）一维单探测器追踪模型

一维情形下，目标可视为在同一直线上向同一方向运动，探测器的中心也在该直线上。

1）探测器的位置

依据模型所做假设，探测器应保证在一秒内能连续追踪相应目标，这就要求在一秒内目标的运动轨迹都在探测范围内。一维情形下，目标运动一秒内运动轨迹为一线段，探测器探测范围为一以探测中心为中点的线段。只要探测中心置于距一秒内目标起始位置一个探测半径，于目标运动方向侧即可保证在一秒内连续探测目标。

示意图如下：

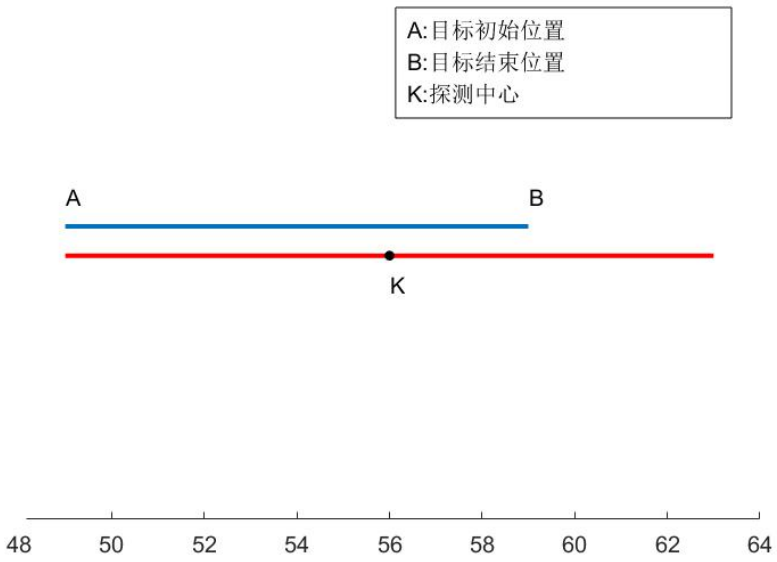

图 1 直线运动位置计算示意图

2）探测器的时间调度

任何时刻探测器的位置都是由当时目标决定的，这样，需要考虑的仅仅是目标的选择，也就是探测器的时间调度。为达成对目标的追踪，探测器需要在首次和二次追踪时段都保持对目标的连续追踪，这就要求所选目标的首次、二次追踪时间不能冲突。

3）二次追踪的时间调度

追踪问题的时间调度是两段的时间分配问题，其最优解并不一定是在仅考虑某一段情况下的最优解。但在这里优先考虑二次追踪时段的单时段最优解。

考虑追踪器可用时间段： $\mathrm{Slot}[0,\max(b_j)]$，$\max(b_j)$为理论最大工作时刻，在分析时，为了简便表示，根据所给数值选择了 200 作为时间区间的上限值，所有的目标不会超过这个运动时间，并将每个探测器可用时段设为$[0,200]$；

目标时段 $[a_j,b_j]$，$j=1,2,\dots,M$。显然，存在的追踪方案为有限个，因此，仅考虑二次追踪时间不互斥的话，必存在若干个最优解。

根据活动安排类问题的贪心算法求解可知，以b_j从小到大将目标重新排序，并按该序列对目标进行$0-99$编号，即产生$seq_i, i=0,1,\dots 99$。

按照后一项的开始时间晚于前一项的结束时间，通过贪心算法我们可以一个包含最早结束二次追踪的目标seq_0的最优解。

证明该解为若干最优解之一的过程如下：

假设二次追踪不互斥的目标安排问题有一个最优解sec_l，$sec_l=\{tar_i,\mathrm{i}=1,2,\dots,\mathrm{Card}(sec_l)\}$,由$b_{seq_0}\le b_{tar_1}<a_{tar_2}$可知，元素$tar_1$被$seq_0$替换后，$sec_l$中所含的

元素不变，仍为最优解。

将第一项tar_i从解中去掉，则第二项tar_{i+1}可以按照同样的方法，从剩下的目标中选择bj最小的目标替换tar_{i+1}，以此类推，二次追踪时间安排具有最优子结构，即考虑从以结束时间早优先为原则的贪心算法必能获得一个最优解。

由上，我们知道，任意一最优解可通过替换变换为贪心法所得最优解。考虑到实际计算时，仅有一组解可供下一步操作也许意味着解的局限性，我们从这个易得的包含若干个目标的原始最优解$\mathrm{sec}_0 = \{tar_i, \mathrm{i} = 1,2,\dots,\mathrm{Card}(sec)\}$出发，通过扩充操作，使贪心法所得最优解变为任意最优解。

下面简述扩充操作的步骤：

Step1:对于贪心最优解$sec_0 = \{tar_i, \mathrm{i} = 1,2,\dots,\mathrm{Card}(sec)\}$,找出每个$tar_i$的对应替换集$Rep_i$，$Rep_i$中元素$seq_i$(目标用编号值表示)满足$bj_{seq_i} < aj_{tar_{i+1}}, tar_i \le seq_i \le tar_{i+1}$

Step2:步骤一结束后，从得到的$\mathrm{Card}(sec_0)$个替换集中依次选取可选目标tar_i，组成集合即最优解$sec_l = \{tar_i, tar_i \in Rep_i\}$。

这样，我们就通过贪心算法获得了所有的二次单时段最优解。

4）首次追踪的时间调度

考虑原始最优解$sec_l =$ {0,1,2,4,6,9,12,14,19,27,38,45,53,54,56,58,65,67,71,75,80,88,96,99}，总共有 24 项，理想状态下，若 24 个目标能全部放进，则理论最大值至少为 24。

A）找替换项时仅考虑$b_{i+1} < a_{tar_{i+1}}$

通过选择可替换项操作，得到如下 24 个可替换项集合Rep：

表 1 分区及分区内目标集合

分区编号	目标集合
Rep_0	{0}
Rep_1	{1}
Rep_2	{2,3}
Rep_3	{4,5}
Rep_4	{6}
Rep_5	{9,10}
Rep_6	{12}
Rep_7	{14,15,16,17}
Rep_8	{19,20}
Rep_9	{27,28,29,30,31,33,34,35,36,37}
Rep_10	{38,39,40,41,42,43}
Rep_11	{45,46,47,48,49,50,51}
Rep_12	{53}
Rep_13	{54}

Rep_14	{56,57}
Rep_15	{58,59}
Rep_16	{65}
Rep_17	{67,68}
Rep_18	{71,72}
Rep_19	{75,76,77,78}
Rep_20	{80,81,82,83,84}
Rep_21	{88,89,90,91,92,93,94,95}
Rep_22	{96,97,98}
Rep_23	{99}

B）找替换项时考虑$b_{i+1} < a_{tar_{i+1}}$ and $a_{i+1} > b_{tar_i}$

同样的得到如下 24 可替换分区：

分区编号	目标集合
Rep_0	{0}
Rep_1	{1}
Rep_2	{2,3}
Rep_3	{4}
Rep_4	{6}
Rep_5	{9}
Rep_6	{12}
Rep_7	{14,16,17}
Rep_8	{19}
Rep_9	{27}
Rep_10	{38,42,43}
Rep_11	{45,47}
Rep_12	{53}
Rep_13	{54}
Rep_14	{56,57}
Rep_15	{58}
Rep_16	{65}
Rep_17	{67}
Rep_18	{71,72}
Rep_19	{75}
Rep_20	{80,83,84}
Rep_21	{88, 90,91,92}
Rep_22	{96 98}
Rep_23	{99}

可以看到少数几个目标无法完成替换，其分区中只有自己这一个数，导致所有Rep_i的并集并不是所有的目标集合，而其余的目标都被成功划分到 24 个分区当中。

两次计算可替换分区元素的条件 A）比条件 B）更为松散，导致了 A 比 B 包含的解的总个数多了几倍，也许也意味着一些选择可能因为部分目标的二次追踪开始时间

过早而导致一次比较的浪费，但是在最后的结果上反映出来 A）条件的最大追踪个数在 M=5 时优于 B）条件，在 M=1 时则相同。

针对这一点不同，可能是 A）条件情况下，备选方案增多，且被添加进来的点可能有较早的开始时间，和较短的二次或者首次追踪时长，从而解的个数增多，且排点更为密集，导致了碎片区间并不能再安排进其他目标，而 B）条件则排点更为严格的互不干扰，导致了单个探测器探测目标减少，共同探测目标增多。

接下来进行首次追踪的时间安排。

安排首次追踪时间的原则为在可选区间内寻找最早时间值，以减小安排上靠后区间时对中间部分目标运动时间密度比较大的区间产生干扰，若当前处理目标tar_i无法加入探测区间，即该目标的首次追踪开始时间sj_{tar_i}无法安排，按照替换项添加顺序在可替换集Rep_i中寻找替换项subs_{tar_i}，若该分区内无其他替换项能被追踪，则该分区被放弃，若能找到被探测的目标，将该目标的状态值state置为BEENTRACED，表示已经被成功追踪。

考虑到仅按照顺序寻找替换只能计算一组解，可能没有办法评估解个数好坏，增加一个随机打乱函数，对分区中的目标排序进行打乱，每次安排之前所用的原始解是不同的。

整个流程如下：

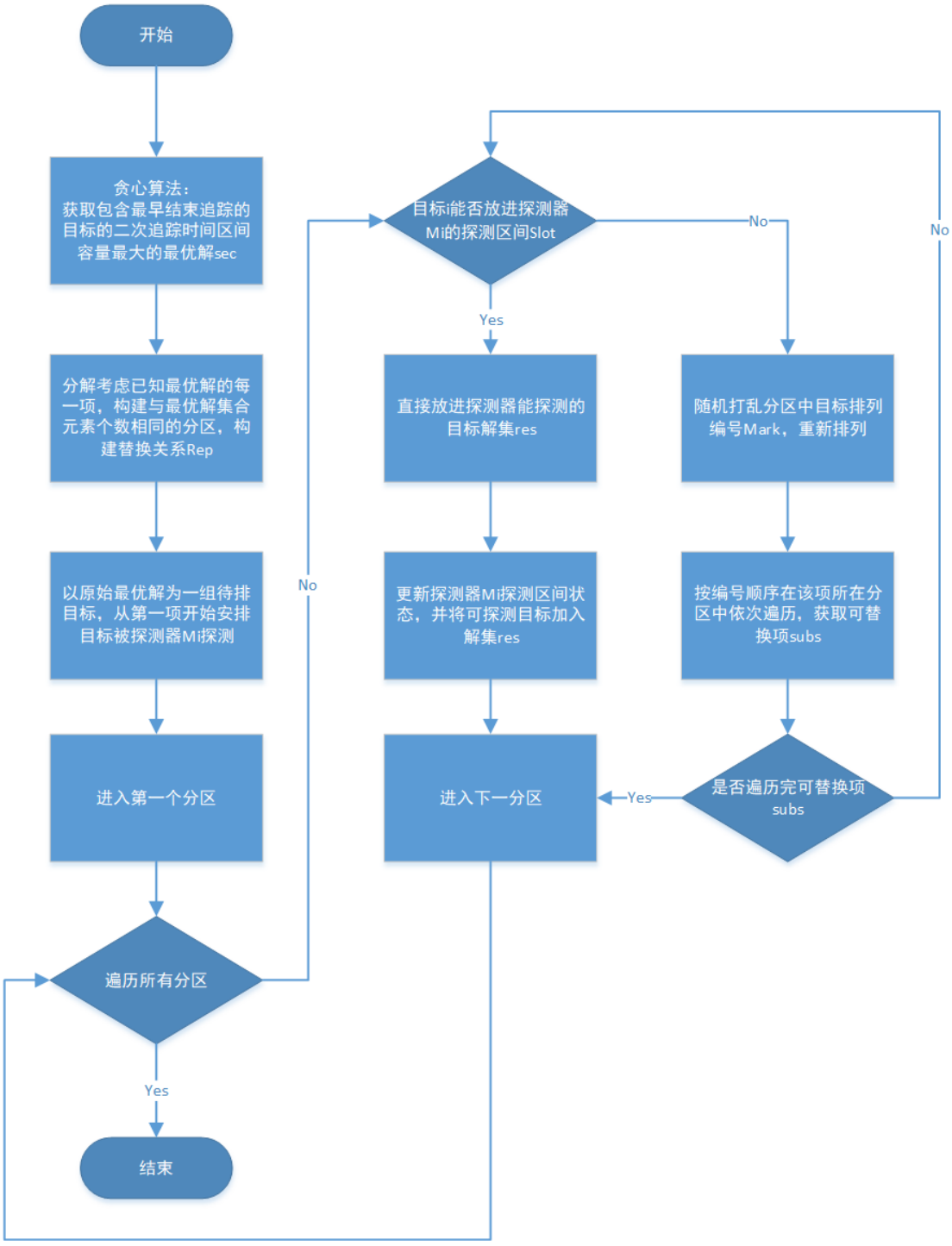

图 2 整体算法大致流程（不包括探测器区间处理）

（2）一维多探测器追踪模型

多探测器的模型可由单探测器的模型推广而来，探测器的位置可依照单探测器的模型。在时间调度上，可类比装箱问题：

Step1:考虑第一个探测器，对初始目标集O_1调用一维单探测器的策略，获得解集V_1。

Step2：对 i 次初始集合中$O_i = O_{i-1} \backslash V_{i-1}$使用一维单探测器的策略，获得解集$V_i$，重复该步骤 M-1 次

Step3：从M个探测器中收集空余时段，组成第$M+1$个探测器，使用一维单探测器的策略，获得解集V_{M+1}。

Step4：组合以上所有解集，获得最终解集 V。

其中第 3 步的流程如下：

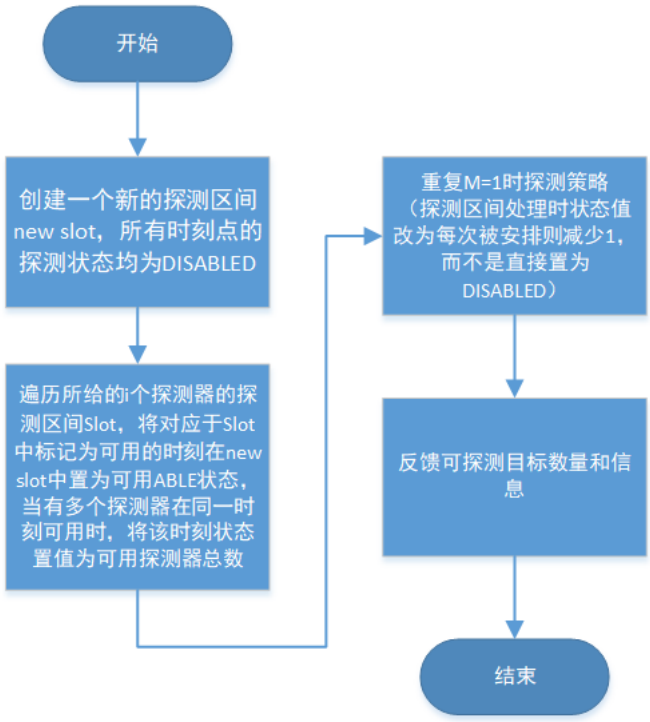

图 3 空余可探测区间处理算法流程

可以想到，由于每个分区可以贡献分区数量个选择，以首次操作时包含编号为 0 的原始最优解为例，将各个分区中的目标可能选择计算在内，整个的最优解集的数量将达到 227,082,240 种，过于庞大，不可能完全遍历，那么采取随机打乱排序之后再使用顺序遍历，通过循环数增加的做法可以选出一个有限子集进行试验。

一个探测器的安排结束后，到下一个探测器开始安排时，会重新计算当前未被追踪目标的最优解集，并用于最大兼容目标数量的寻找。

令循环次数 k=100,1000,10000，多次试验之后，下列结果出现次数最多。

设置探测器数量为 1，最终的结果如下，可追踪数量为 14：

表 2 条件 A）下 M=1 时探测器调度表

No.	Target ID	Target seq	sj	aj	bj
1	64	0	4	16	17
2	25	1	7	24	27
3	49	2	18	29	31
4	14	9	32	45	48
5	50	16	53	58	62
6	72	19	49	66	67
7	28	38	73	77	78
8	43	47	54	82	84
9	66	54	87	94	98
10	15	56	89	99	100
11	51	67	101	116	118
12	94	75	68	122	124
13	84	90	125	140	144
14	54	99	145	174	178

设置探测器数量为 5，最终的结果如下，一共可追踪数量为 43：

探测器 1 追踪了 14 个目标；

探测器 2 追踪了 9 个目标；

探测器 3 追踪了 8 个目标；

探测器 4 追踪了 7 个目标；

探测器 5 追踪了 5 个目标。

并且空余区间 Rem（对应于算法图中的 new slot）也无法再成功探测目标，即没有目标能被多个探测器共同追踪。

表 3 条件 A）下 M=5 时探测器调度表

Monitor id	No.	Target ID	Target seq	sj	aj	bj
1	1	64	0	4	16	17
	2	25	1	7	24	27
	3	49	2	18	29	31
	4	14	9	32	45	48
	5	50	16	53	58	62
	6	72	19	49	66	67
	7	28	38	73	77	78
	8	43	47	54	82	84
	9	66	54	87	94	98
	10	15	56	89	99	100
	11	51	67	101	116	118
	12	94	75	68	122	124

	13	84	90	125	140	144
	14	54	99	145	174	178
2	15	85	3	7	31	32
	16	32	4	11	35	37
	17	86	11	38	45	50
	18	17	22	51	65	70
	19	89	42	71	77	79
	20	65	55	80	98	99
	21	8	66	100	110	116
	22	16	80	117	128	131
	23	13	91	132	142	145
3	24	2	5	18	29	38
	25	97	10	39	40	48
	26	36	25	58	67	71
	27	90	48	49	76	85
	28	98	57	72	99	100
	29	38	69	101	118	119
	30	62	78	86	120	125
	31	34	84	112	129	135
4	32	41	6	7	42	44
	33	57	12	45	50	53
	34	11	17	26	59	63
	35	70	27	54	69	71
	36	69	51	72	80	91
	37	48	71	92	119	121
	38	31	93	107	124	147
5	39	56	7	29	36	45
	40	77	24	46	61	70
	41	39	53	71	92	93
	42	79	72	94	119	121
	43	47	89	105	131	143

另外在前述的 B）条件约束下的最优解为：

M=5 时能追踪 42 个目标：

探测器 1 追踪了 14 个目标；

探测器 2 追踪了 9 个目标；

探测器 3 追踪了 7 个目标；

探测器 4 追踪了 5 个目标；

探测器 5 追踪了 5 个目标；

探测器共同追踪 2 个目标。

表示如下：

表 4 条件 B）下 M=5 探测器调度表

Monitor id	No.	Target ID	Target seq	sj	aj	bj
1	1	64	0	4	16	17
	2	25	1	7	24	27
	3	49	2	18	29	31
	4	14	9	32	45	48
	5	50	16	53	58	62
	6	72	19	49	66	67
	7	28	38	73	77	78
	8	43	47	54	82	84
	9	66	54	87	94	98
	10	15	56	89	99	100
	11	51	67	101	116	118
	12	94	75	68	122	124
	13	84	90	125	140	144
	14	54	99	145	174	178
2	15	85	3	7	31	32
	16	32	4	11	35	37
	17	86	11	38	45	50
	18	17	22	51	65	70
	19	89	42	71	77	79
	20	65	55	80	98	99
	21	8	66	100	110	116
	22	16	80	117	128	131
	23	13	91	132	142	145
3	24	2	5	18	29	38
	25	57	12	40	50	53
	26	36	25	58	67	71
	27	39	53	72	92	93
	28	98	57	44	99	100
	29	38	69	101	118	119
	30	33	96	120	151	166
4	31	41	6	7	42	44
	32	11	17	26	59	63
	33	70	27	45	69	71
	34	7	65	73	106	115
	35	30	98	116	151	172
5	36	56	7	29	36	45
	37	77	24	46	61	70
	38	48	71	71	119	121
	39	58	79	86	122	128
	40	3	95	105	133	149

共同追踪目标：

表 5 共同追踪目标表

No.	Target ID	Target seq	sj	aj	bj
41	79	72	94	119	121
42	1	83	103	131	132

进一步来看其运动时间内被追踪情况。

对于 No.41 号目标，$S_j = 94, p_j = 10$，探测器追踪时间安排：

表 6No.41 号目标运动区间探测器 ID 表

Time	Monitor ID
94	1
95	1
96	2
97	1
98	2
99	2
100	2
101	1
102	1
103	2
104	2
119	3
120	3
121	2

对于 No.42 号目标，$S_j = 103, p_j = 15$，探测器追踪时间安排：

表 7No.42 号目标运动区间探测器 ID 表

Time	Monitor ID
103	1
104	1
105	2
106	1
107	2
108	2

109	2
110	1
111	1
112	2
113	2
114	3
115	3
116	2
117	2
118	1
131	1
132	1

M=1 时的情况与 A）情况下 M=1 的最优解一样。

（3）**三维单探测器抛物追踪模型**

依据题目预设条件，即目标从不同高度、不同方向扔出，轨迹为抛物线。建立柱坐标，依据模型假设，目标从同一轴扔出，即转化为平面追踪问题。

1）探测器的位置

在探测范围一定的情况下，要完整探测到目标在一秒内的轨迹，考虑目标轨迹为抛物线，置探测中心于该秒内目标初始位置和结束位置的中点。这样，运动轨迹上的点距探测中心最远处即初始位置和结束位置，最大化利用了探测范围。

示意图如下：

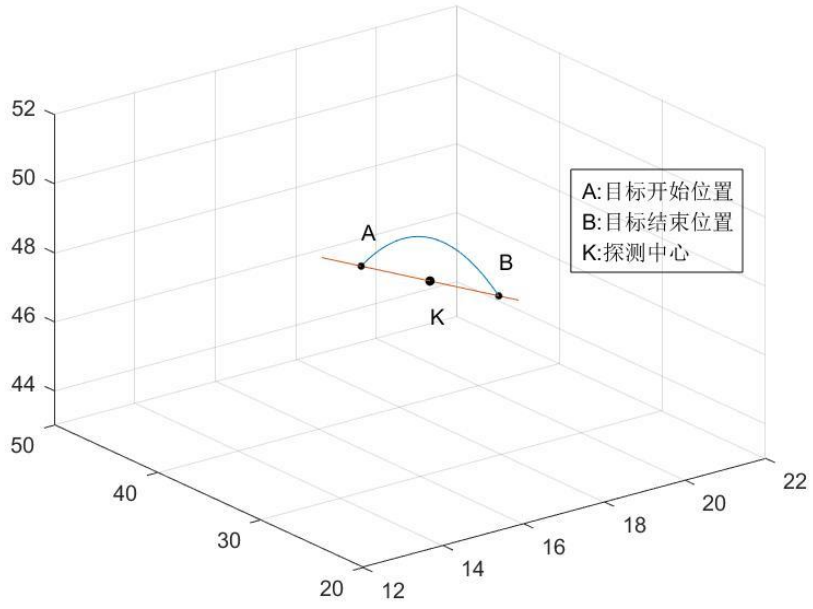

图 4 空间运动位置计算示意图

2）探测器的时间调度

在确定探测器的位置调度，只需重复一维情况下的时间调度即可。

具体实现时利用柱坐标系表示探测中心位置，利用原始数据中的初始位置数据，将其假设为所有目标从同一条

关于坐标和具体时间计算结果，可以通过附件的 Excel 表格查看。

5 模型的分析

（1）模型的优点

1）逻辑清晰，利用多个方法将数量庞大的可能最优解存在的解集分解成块，利用洗牌算法，贪心算法等算法思想分别模拟近似和取最优解的过程，在化大为小的处理上有可取之处。

2）模型的优化策略简单易懂，代码运行速度较快，结果比较直观。

（2）模型的缺点

1）结果依赖于给定的优先选择原则。

选择在首次追踪时间安排时，采用的策略单一，从最终的结果来看，不管是多个探测器同时工作还是单一探测器工作，空余时刻的数量都在 80 个以上，几乎接近总工作时长的 1/2，说明有优化的余地。

2）目前的探测器时间利用还不够紧密。

在前面选择的影响下，可能影响到最后的剩余空间的利用问题，导致最后的空余

空间并没能让多个探测器配合探测成功目标，而且在本模型中没有考虑多个探测器配合的情况，而是认为在目标可被成功追踪时，相比于单个探测器追踪，如果在该时段内有多个探测器同时空闲，则也意味着本来就在追踪它的探测器继续追踪也可以进行，也将会是最好表示的；但是分析可以得到，如果在某个关键时间点更换可行探测器，也许能更好的利用之前探测器的碎片时间，通过移植不可行时间片到已经排列饱和的探测器时间片中，从而减轻当前探测器的探测压力，意味着更多的可用空间。

3）循环次数对于结果的影响很小。

这是由于分区不当，部分分区的元素个数过少，或者只有一项，导致每一次生成新的最优解序列时，相当于不是完全随机而是部分确定的，因此在时间安排上的影响坏的点很有可能每次都会被选中。

（3）模型的改进

1）建模工具的改进：以后尽量使用 MATLAB 建模，这次比赛因为没有 MATLAB 编程经验，不太会使用该软件，本队使用 C++进行建模，也因此放弃了一些令模型自主学习以趋近最优解的算法，退而求其次选择了用随机的思想去设计一些其他的函数，并且在实际计算时，自己设计数据结构也花了比较长的时间。

2）需要考虑更多的优化策略：对于目标不可行的替换策略，以及如何使探测器工作时间段安排的更满以实现多个探测器合作探测的策略等。

6 模型的推广

下面考虑一个探测器可以同时探测 X 个目标的推广问题。这里讨论 X=2，M=1 的一维追踪问题。显然，探测器可同时探测 2 个目标O_i，O_j的充要条件是目标追踪时间相同，目标运动轨迹都在探测范围内。

（1）探测器的位置

同一时间只探测一个目标的位置可依照之前的情形。同一时间探测两个目标时，要保证在一秒内两段目标运动轨迹都在探测范围内，应将探测中心置于线段 AB 中点，A=min{O_i初始位置，O_j初始位置},B= max{O_i结束位置，O_j结束位置}。

（2）探测器的时间调度

Step1：对所有目标进行判定，存在一定时间段可被探测器同时探测的两个目标记为Con_k=(O_i, O_j)

Step2：以一维单探测器策略获得多个二次最优解集sec_i

Step3：使用一维单探测器首次追踪策略，将Con_k中O_i置入调度时间时，尝试将O_j

同时置入

Step4：所得最优集即为解集

7 参考文献

[1]Thomas H.Corman, Charles E.Leiserson, Ronald L.Rivest Clifford Stein. Introduction to Algorithms(Third Edition)[M],414-450.

[2]孙佩歆.无线传感器网络中干扰最小化问题的后悔贪心算法[J].计算机工程与科学,2017,39(12):2217-2223.

[3]杜磊,付喜梅,杨文瀚.无线网络干扰最小化问题的一种缩边贪心算法[J/OL].计算机应用研究:1-7[2019-05-17].https://doi.org/10.19734/j.issn.1001-3695.2018.10.0804.

[4]张鹏程,茹江燕.基于贪心策略的单一下料问题求解研究[J].河北水利电力学院学报,2018(04):44-47.

[5]李兆阳,强韶华,沙洋.基于贪心策略的施工项目群资源调度研究[J].知识经济,2015(12):111-112+114.

8 附录

代码实现时需要读取 Excel 文件，请将题目原始数据文件 Problem A.xls 中“目标数据”工作表另存为 Problem A.csv 文件，以“二次追踪结束时间”为升序排序后保存，并与编译产生的可执行.exe 文件放在同一文件夹下，方能正常运行。

建议使用 Dev C++5.11 建立工程，对源代码进行编译。

（1）detect.h

```
/**
 ** Definitions for structures
 ** Statements of functions
 ** @name:detect.h
 ** @author:3160104633@zju.edu.com
 ** @date:2019/5/13
 **/
#ifndef _DETECT_H_
#define _DETECT_H_

#define N 100
#define MAXM 5
#define MAXTIME 200
```

```
#define ABLE 0
#define DISABLED -1
#define DEBUG 0
#define MANUAL 10
#define LINE_MODEL 1
#define CIRCLE_MODEL 0
#define LENGTH(a) ((sizeof(a))/(sizeof(a[0])))

typedef struct _object Object;
typedef struct _object* ObjectPtr;
struct _object {
    int startcor[3]; //start cooridinate (x(t),y(t),z(t))
//  int curcor[3]; //current cooridinate
    int oriseq; //object number, constant
    int seq; //object sequence, variable
    int velocity;
    //for time denotation
    int begint;
    int endt;
    int sj;
    int pj;
    int second[2];
    int mark;
};

typedef struct _monitor Monitor;
typedef struct _monitor* MonitorPtr;
struct _monitor
{
    int center[3]; //center cooridinate (x(t),y(t),z(t))
    int id; //serial number for monitors
    int worklen;
    int position[MAXTIME][3];
    int* slot; //work time for monitors
};

typedef struct _monitorset Mset;
struct _monitorset
{
    int amount;
    Monitor m[MAXM];
};

//被选中的目标，subs 存放其分区目标
typedef struct _selected Selected;
```

```
typedef struct _selected* SelectedPtr;
struct _selected {
    int obseq;
    SelectedPtr next;
    SelectedPtr subs;
    int subnum;
    int mark;
};

typedef struct _greedysolution Greedy;
typedef struct _greedysolution* GreedyPtr;
struct _greedysolution
{
    SelectedPtr objectarr;
    int capacity;
};

//二次追踪贪心算法解集
typedef struct _secondtracesolution SecondT;
struct _secondtracesolution
{
    int amount;
    int* secondset;
};


//贪心算法内部接口
static int greedy(int s[], int f[], int a[], int k);
//扩展解集函数（内部接口）
static SecondT* ExtendGreedySolution(GreedyPtr fs, int s[], int f[], int index[]);
//获取理想结果数
static void getIdealRes(int totaltime[], int maxbj, int* MAXN);
static void MaxArrangeNumber(int aj[], int bj[], int pj[], int* MAXN);
//目标数据处理内部接口
static void handleObject(ObjectPtr ob, int aj[], int bj[], int pj[], int ts[], int te[], int v[], int startP[][3], int seq_orinum[][2]);
//探测器内容初始化
static void initialMonitors(MonitorPtr m,int id);
//获取当前未被追踪的目标集合
static SecondT* getUntracedSet();
//资源是否被占用
static int isAvailable(MonitorPtr m, int obseq, int* _sj,int flag);
//获取当前状态下所有探测器的剩余可用时间区间
static void getAvailableSlot(Mset* ms);
//检查剩余区间 Rem 是否可以成功追踪同伴
```

```
103. static void CheckRem(MonitorPtr m,SecondT* secres);
104. //打乱可替换项顺序
105. static void shuffle(GreedyPtr aim);
106. //计算成功追踪目标数
107. static int getTracedNum(int* arr);
108. //打乱顺序函数
109. static int* random(int n,int i);
110. //计算探测中心位置
111. void calculateCenterPos(int* targetList,MonitorPtr m,int AN,int id);
112. //按照二次追踪时间实现的贪心算法
113. SecondT* SecondTraceGreedy(ObjectPtr ob);
114. //对 object 对象进行基本数据处理
115. void ObjectHandle(ObjectPtr ob);
116. //得到所有探测器的安排
117. void getArrangement(Mset* monitors, SecondT* secres, int amount, int flag);
118. //对于同一个探测器安排其调度
119. int* arrangeTarget(SecondT* secres, MonitorPtr m,int* AN,int flag);
120. //读 Excel 文件接口
121. void readExcel();
122. //打印结果
123. void printResult(int* resarr, int id,int AN,int flag);
124. bool compare(const int &a, const int &b);
125. bool compare1(const int &a, const int &b);
126.
127. #endif // !_DETECT_H_
```

（2）detect.cpp

```
1.  #pragma warning(disable:4996)
2.  /**
3.   ** Implementation for functions
4.   ** @name:detect.cpp
5.   ** @author:3160104633@zju.edu.com
6.   ** @date:2019/5/13
7.   **/
8.  #include <iostream>
9.  #include <stdlib.h>
10. #include <cstdlib>
11. #include <string>
12. #include <string.h>
13. #include <streambuf>
14. #include <fstream>
15. #include <ctime>
16. #include <iostream>
```

```
#include <math.h>
#include <algorithm>
#include "detect.h"
#include "errormsg.h"
using namespace std;

#define PI 3.14
#define TRYTIME 1
#define marked -1
#define NOOBJECT -1
#define BEENTRACED -1
#define NUM 1000 //设置 0-1 内随机数计算精度为小数点后 4 位
#define LENGTH(a) ((sizeof(a))/(sizeof(a[0])))

//通过目标数据可以得到的基本数据
Object obarr[100],tmpobarr[100];
double weight[3] = { 0,0,0 };
int aj[N], bj[N], pj[N], ts[N], te[N], v[N];
int seq_orinum[N][2];
int startP[N][3];
GreedyPtr fs,remfs;
int colnum;
int* mid;
int sjSet[6 ][N];
int ASlot[MAXTIME];

//按照二次追踪时间实现的贪心算法
SecondT* SecondTraceGreedy(ObjectPtr ob)
{
    //the object array has already been sorted according to end time of second trace

    int* s;
    int* f;
    int* res;
    s = (int*)malloc(sizeof(int)*N);
    f = (int*)malloc(sizeof(int)*N);
    res = (int*)malloc(sizeof(int)*N);
    if (s == NULL || f == NULL || res == NULL) {
        errorReport("Can't mallocating room for array.");
        exit(-1);
    }
    memset(res, 0, sizeof(int)*N);
    int k, i;
    k = LENGTH(obarr);
    for (i = 0; i < N; i++)
```

```
61.     {
62.         s[i] = ob[i].second[0];
63.         f[i] = ob[i].second[1];
64.     }
65.     k = greedy(s, f, res, k);
66.     colnum = k;
67.     int* index;
68.     index = (int*)malloc(sizeof(int)*k);
69.     int j = 0;
70.     for (i = 0; i < N; i++)
71.     {
72.         if (res[i] == 1)
73.         {
74.             index[j] = i;
75.             j++;
76.         }
77.     }
78.
79.     //generate the first solution for greedy alogorithm
80.     SelectedPtr resarr;
81.     resarr = (SelectedPtr)malloc(sizeof(Selected)*k);
82.     memset(resarr, 0, sizeof(Selected)*k);
83.
84.     for (i = 0; i < k - 1; i++)
85.     {
86.         resarr[i].obseq = index[i];
87.         resarr[i].next = &resarr[i + 1];
88.     }
89.     resarr[k - 1].obseq = index[k - 1];
90.     resarr[k - 1].next = NULL;
91.     SelectedPtr tmp = resarr;
92.     fs = (GreedyPtr)malloc(sizeof(Greedy));
93.     fs->objectarr = resarr;
94.     fs->capacity = k;
95.     SecondT* sec = new SecondT;
96.     //扩展二次追踪最优解集
97.
98.     sec = ExtendGreedySolution(fs, s, f, index);
99.     return sec;
100. }
101.
102. static int greedy(int s[], int f[], int a[], int k)
103. {
104.     int i,z;
105.     for(i = 0; i < N;i++)
```

```
106.     {
107.         if(tmpobarr[i].mark!=BEENTRACED){
108.             a[i] = 1;
109.             break;
110.         }
111.
112.     }
113.     int cnt = 1;
114.     for (z = i+1; z < k; z++)
115.     {
116.         if (s[z] > f[i] && tmpobarr[z].mark!=BEENTRACED)
117.         {
118.             a[z] = 1;
119.             i = z;
120.             cnt++;
121.         }
122.     }
123.     return cnt;
124. }
125.
126. //扩展解集函数
127. static SecondT* ExtendGreedySolution(GreedyPtr fs, int s[], int f[], int index[])
128. {
129.     int i, j, id1, id2;
130.
131.     for (i = 0, j = 1; j <= colnum - 1; i++, j++)
132.     {
133.         id1 = index[i];
134.         id2 = index[j];
135.         int subs;
136.         int* cnt = &(fs->objectarr[i].subnum);
137.         *cnt = 0;
138.         SelectedPtr tmp = fs->objectarr[i].subs;
139.         for (subs = id1; subs < id2; subs++)
140.         {
141.             //for the beginning of the loop, every item of selected sequence has th
    e substitute of itself
142.             //solution B) term:
143.             if (f[subs] < s[id2] && s[subs] >= s[id1])
144.             //solution A) term:
145.             //if(f[subs] < s[id2] )
146.             {
147.                 SelectedPtr node = new Selected;
148.                 node->obseq = subs;
149.                 node->next = node->subs = NULL;
```

```
150.                 if (tmp == NULL) {
151.                     fs->objectarr[i].subs = node;
152.                     tmp = fs->objectarr[i].subs;
153.                 }
154.                 else {
155.                     tmp->next = node;
156.                     tmp = tmp->next;
157.                 }
158.                 node->mark = *cnt;
159.                 *cnt = *cnt + 1;
160.             }
161.         }
162.         if (j == colnum - 1)
163.         {
164.             int* cnt = &(fs->objectarr[j].subnum);
165.             *cnt = 0;
166.             SelectedPtr tmp = fs->objectarr[j].subs;
167.             for (subs = id2; subs < N; subs++)
168.             {
169.                 //solution B) term:
170.                 if (f[id1] < s[subs] && s[subs] >= s[id1])
171.                 //solution A) term:
172.                 //if(f[id1] < s[subs])
173.                 {
174.                     SelectedPtr node = new Selected;
175.                     node->obseq = subs;
176.                     node->next = node->subs = NULL;
177.                     if (tmp == NULL) {
178.                         fs->objectarr[j].subs = node;
179.                         tmp = fs->objectarr[j].subs;
180.                     }
181.                     else {
182.                         tmp->next = node;
183.                         tmp = tmp->next;
184.                     }
185.                     node->mark = *cnt;
186.                     *cnt = *cnt + 1;
187.                 }
188.             }
189.         }
190.     }
191.
192.     SecondT* resset = new SecondT;
193.     resset->amount = TRYTIME;
194.     int* arr = new int[colnum];
```

```
195.     for (int n = 0; n < colnum; n++)
196.     {
197.         arr[n] = index[n];
198.     }
199.     Selected tmp = fs->objectarr[0];
200.     int col = 0;
201.     while ((tmp.next) != NULL)
202.     {
203.         SelectedPtr cur = tmp.subs;
204.         int choose = 1;
205.         for (int cnt = 1; cnt < choose; cnt++)
206.         {
207.             cur = cur->next;
208.         }
209.         arr[col] = cur->obseq;
210.         col++;
211.         SelectedPtr t = tmp.next;
212.         tmp = *t;
213.     }
214.     resset->secondset = arr;
215.     return resset;
216. }
217.
218. static void getIdealRes(int totaltime[], int maxbj, int* MAXN)
219. {
220.     //assume sum(bj[i]-aj[i]+pj[i] is sorted in ascending array
221.     int cnt = 0;
222.     int timesum = 0;
223.     while (timesum <= maxbj)
224.     {
225.         timesum += totaltime[cnt];
226.         cnt++;
227.     }
228.     *MAXN = --cnt;
229. }
230.
231. static void MaxArrangeNumber(int aj[], int bj[], int pj[], int* MAXN)
232. {
233.     int* maxbj = max_element(bj, bj + N);
234.     int totaltime[N];
235.     for (int i = 0; i < N; i++)
236.     {
237.         if(tmpobarr[i].mark != BEENTRACED)
238.             totaltime[i] = bj[i] - aj[i] + pj[i];
239.     }
```

```
240.     sort(totaltime, totaltime + N, compare1);
241.     getIdealRes(totaltime, *maxbj, MAXN);
242. }
243.
244. //对 object 对象进行基本数据处理
245. void ObjectHandle(ObjectPtr ob)
246. {
247.     handleObject(ob, aj, bj, pj, ts, te, v, startP, seq_orinum);
248. }
249.
250. //目标数据处理内部接口
251. static void handleObject(ObjectPtr ob, int aj[], int bj[], int pj[], int ts[], int
    te[], int v[], int startP[][3], int seq_orinum[][2])
252. {
253.     for (int i = 0; i < N; i++)
254.     {
255.         aj[i] = ob[i].second[0];
256.         bj[i] = ob[i].second[1];
257.         pj[i] = ob[i].pj;
258.         ts[i] = ob[i].begint;
259.         te[i] = ob[i].endt;
260.         v[i] = ob[i].velocity;
261.         for (int j = 0; j < 3; j++)
262.         {
263.             startP[i][j] = ob[i].startcor[j];
264.         }
265.         seq_orinum[i][0] = ob[i].seq;
266.         seq_orinum[i][1] = ob[i].oriseq;
267.     }
268. }
269.
270. //探测器内容初始化
271. static void initialMonitors(MonitorPtr m,int id)
272. {
273.     m->id = id;
274.     m->slot = new int[MAXTIME];
275.     for(int i = 0; i < MAXTIME; i++)
276.     {
277.         m->slot[i] = 0;
278.     }
279.     for(int i = 0; i < MAXTIME; i++)
280.     {
281.         m->position[i][0] = m->position[i][1] =m->position[i][2] =0;
282.     }
283. }
```

```
284.
285. //获取当前未被追踪的目标集合
286. static SecondT* getUntracedSet()
287. {
288.     return SecondTraceGreedy(tmpobarr);
289. }
290.
291. //得到所有探测器的安排
292. void getArrangement(Mset* monitors, SecondT* secres, int amount, int flag)
293. {
294.
295.     for (int i = 0; i < amount; i++)
296.     {
297.         initialMonitors(&monitors->m[i],i+1);
298.         int AN = 0;
299.         int* resarr = NULL;
300.         if(i != 0)
301.             secres = getUntracedSet();
302.         resarr = arrangeTarget(secres, &(monitors->m[i]), &AN,flag);
303.         //calculate coordinate
304.         calculateCenterPos(resarr,&monitors->m[i],AN,i+1);
305.         //print result
306.         printResult(resarr,i+1,AN,0);
307.     }
308. }
309.
310. //对于同一个探测器安排其调度
311. int* arrangeTarget(SecondT* secres, MonitorPtr m,int* AN,int flag)
312. {
313.     //对当前的探测器 m 进行操作,secres 选出来的是我当前选择的一组二次不互相干扰的目标序
    列
314.     int MAXN = 0;//考虑所有的目标，按照总时间长短得到的能探测的目标的理论最大值
315.     MaxArrangeNumber(aj, bj, pj, &MAXN);
316.     //初始化结果列表
317.     int* resarr;
318.     resarr = new int[MAXN];
319.     memset(resarr, NOOBJECT, sizeof(int)*MAXN);
320.     //初始化 sj 列表
321.     int curcol = 0;
322.     int* tmp = secres->secondset;
323.     for (int k = 0; k < colnum; k++)
324.     {
325.         //对一个探测器进行安排
326.         SelectedPtr cur = fs->objectarr[k].subs;
327.         SelectedPtr ppos = cur;
```

```
328.         SelectedPtr precur = NULL;
329.         SelectedPtr pos = NULL;
330.         if(k > 0){
331.             precur = fs->objectarr[k-1].subs;
332.             pos = precur;
333.         }
334.         while(cur != NULL)
335.         {
336.             //补偿措施：当当前的目标不能放进去时，在其替换项中按顺序找出可替换项安排进
    去
337.             if(tmpobarr[cur->obseq].mark != BEENTRACED && isAvailable(m,cur->obseq,
    &sjSet[m->id-1][cur->obseq],flag) == ABLE)
338.             {
339.                 tmpobarr[cur->obseq].mark = BEENTRACED;
340.                 resarr[curcol] = cur->obseq;
341.                 curcol++;
342.                 break;
343.             }
344.             cur = cur->next;
345.         }
346.         cur = ppos;
347.     }
348.     *AN = curcol;
349.     return resarr;
350. }
351.
352. //资源是否被占用
353. static int isAvailable(MonitorPtr m, int obseq, int* _sj,int flag)
354. {
355.     if(flag == 0)
356.     {
357.         for (int i = aj[obseq]; i <= bj[obseq]; i++)
358.         {
359.             if (m->slot[i] == DISABLED) {
360.                 return DISABLED;
361.             }
362.         }
363.         for (int i = ts[obseq]; i <= (aj[obseq] - pj[obseq]); i++)
364.         {
365.             int cnt = 0;
366.             int j = i;
367.             //选 sj 的位置
368.             for (; j < i + pj[obseq]; j++)
369.             {
370.                 if (m->slot[j] == ABLE)
```

```
371.                     cnt++;
372.             }
373.             if (cnt == pj[obseq]) {
374.                 *_sj = i;
375.                 for (int z = aj[obseq]; z <= bj[obseq]; z++)
376.                     m->slot[z] = DISABLED;
377.                 for (int z = *_sj; z < *_sj + pj[obseq]; z++)
378.                     m->slot[z] = DISABLED;
379.                 return ABLE;
380.             }
381.         }
382.         return DISABLED;
383.     }
384.     else
385.     {
386.         for (int i = aj[obseq]; i <= bj[obseq]; i++)
387.         {
388.             if (m->slot[i] == DISABLED) {
389.                 return DISABLED;
390.             }
391.         }
392.         for (int i = ts[obseq]; i <= (aj[obseq] - pj[obseq]); i++)
393.         {
394.             int cnt = 0;
395.             mid = new int[pj[obseq]+bj[obseq]-aj[obseq]+1];
396.             int j = i;
397.             //选 sj 的位置
398.             for (; j < i + pj[obseq]; j++)
399.             {
400.                 if (m->slot[j] >= 0){
401.                     //mid 记录当前时刻由探测器 m[id]探测该目标
402.                     mid[cnt] = m->slot[j]+1;
403.                     cnt++;
404.                 }
405.             }
406.             if (cnt == pj[obseq]) {
407.                 *_sj = i;
408.                 for (int z = aj[obseq]; z <= bj[obseq]; z++)
409.                 {
410.                     mid[pj[obseq]+z-aj[obseq]] = m->slot[z]+1;
411.                     m->slot[z] -= 1;
412.                 }
413.                 for (int z = *_sj; z < *_sj + pj[obseq]; z++)
414.                     m->slot[z] -= 1;
415.                 return ABLE;
```

```
416.             }
417.         }
418.         return DISABLED;
419.     }
420. }
421.
422. //获取当前状态下所有探测器的剩余可用时间区间
423. static void getAvailableSlot(Mset* ms)
424. {
425.
426.     for(int i = 0; i < MAXTIME;i++)
427.     {
428.         ASlot[i] = -1;
429.     }
430.     for(int i = 0;i < ms->amount;i++)
431.     {
432.         for(int j = 0; j < MAXTIME;j++)
433.         {
434.             if(ms->m[i].slot[j]==0)
435.                 ASlot[j]++;
436.         }
437.     }
438. }
439.
440. //检查剩余区间 Rem 是否可以成功追踪同伴
441. static void CheckRem(MonitorPtr m,SecondT* secres)
442. {
443.     int AN = 0;
444.     int* resarr = NULL;
445.     secres = getUntracedSet();
446.     resarr = arrangeTarget(secres, m, &AN,1);
447.     //calculate coordinate
448.     calculateCenterPos(resarr,m,AN,6);
449.     //print result
450.     printResult(resarr,m->id,AN,1);
451.
452. }
453.
454.
455. //打乱可替换项顺序
456. static void shuffle(GreedyPtr aim)
457. {
458.     GreedyPtr tmp = aim;
459.     for(int i = 0;i < colnum;i++)
460.     {
```

```
461.         int* newind = random(tmp->objectarr[i].subnum+1,i);
462.         SelectedPtr head = new Selected;
463.         head->next = NULL;
464.         SelectedPtr ptr = head;
465.         SelectedPtr ppos = tmp->objectarr[i].subs;
466.         for(int j = 0; j < tmp->objectarr[i].subnum; j++ )
467.         {
468.             SelectedPtr pos = tmp->objectarr[i].subs;
469.             SelectedPtr node = new Selected;
470.             while(pos != NULL){
471.
472.                 if(pos->mark == newind[j] && head->next != NULL)
473.                 {
474.                     node->obseq = pos->obseq;
475.                     node->mark = pos->mark;
476.                     if(newind[j] == 0)
477.                         node->subnum = pos->subnum;
478.                     ptr->next = node;
479.                     ptr = node;
480.                 }
481.                 if(pos->mark == newind[j] && head->next == NULL){
482.                     node->obseq = pos->obseq;
483.                     node->mark = pos->mark;
484.                     if(newind[j] == 0)
485.                         node->subnum = pos->subnum;
486.                     head->next = node;
487.                     ptr = node;
488.                 }
489.                 pos = pos->next;
490.             }
491.         }
492.         ppos = head->next;
493.     }
494. }
495.
496. //计算成功追踪目标数
497. static int getTracedNum(int* arr)
498. {
499.     int cnt = 0;
500.     for (int i = 0; i < colnum; i++)
501.     {
502.         if (arr[i] == NOOBJECT)
503.             break;
504.         cnt++;
505.     }
```

```
506.     return cnt;
507. }
508.
509. //计算探测中心位置
510. void calculateCenterPos(int* targetList,MonitorPtr m,int AN,int id)
511. {
512.     int time = 0;
513.     for(int i = 0;i < AN; i++)
514.     {
515.         int tv = v[targetList[i]];
516.         int taj = aj[targetList[i]];
517.         int tbj = bj[targetList[i]];
518.         int tts = ts[targetList[i]];
519.         int tpj = pj[targetList[i]];
520.         int tsj = sjSet[id-1][targetList[i]];
521. #if LINE_MODEL
522.         //目标坐标
523.         int tmppos[3] ={0};
524.         //探测器坐标
525.         //由于物体运动速度≤10/s，每秒改变一次探测器位置可以保证完成追踪
526.
527.         for(int j = 0; j < 3;j++)
528.         {
529.             if(j == 0){
530.                 tmppos[j] = startP[targetList[i]][j]+(tsj-tts)*tv;
531.                 m->position[tsj][j] = tmppos[j]+5;
532.             }
533.             tmppos[j] = startP[targetList[i]][j];
534.             m->position[tsj][j] = tmppos[j];
535.         }
536.         for(time = tsj; time < tsj+tpj; time++)
537.         {
538.             tmppos[0] = tmppos[0]+(time-tsj)*tv;
539.             m->position[time][0] = tmppos[0]+5;
540.         }
541. #endif
542. #if CIRCLE_MODEL
543.         //设所有的目标从同一直线的不同高度抛出，探测中心位置就是物体位置，不再以秒为单位变化而是时刻变化，这里以秒为单位记录
544.         //返回三维柱坐标（初始高度，开始时间，初速度，速度&z 轴夹角，速度极坐标方向，当前时间） (r,phi,z)->(0,1,2)
545.
546.         srand(time(NULL));
547.         double theta = (rand()%(PI))+1;
```

```
548.         double tmp1 = v[targetList[i]]*sin(theta)*(tsj-tts)-0.5*9.8*(tsj-tts)*(tsj-
    tts);
549.         double tmp2 = tmp1-9.8*(tsj-tts)-0.5*9.8;
550.         m->position[tsj][2] = (tmp1+tmp2)/2;
551.         tmp1 = v[targetList[i]]*cos(theta)*(tsj-tts);
552.         tmp2 = tmp1+v*cos(theta);
553.         m->position[tsj][0] = (tmp1+tmp2)/2;
554.         for(time = tsj; time < tsj+tpj; time++)
555.         {
556.             theta = (rand()%(PI))+1;
557.             tmp1 = v[targetList[i]]*sin(theta)*(time-tsj)-0.5*9.8*(time-tsj)*(time-
    tsj);
558.             tmp2 = tmp1-9.8*(time-tsj)-0.5*9.8;
559.             m->position[time][2] = (tmp1+tmp2)/2;
560.             tmp1 = v[targetList[i]]*cos(theta)*(time-tsj);
561.             tmp2 = tmp1+v*cos(theta);
562.             m->position[time][0] = (tmp1+tmp2)/2;
563.         }
564. #endif
565.     }
566. }
567.
568. //读 Excel 文件接口
569. void readExcel()
570. {
571. #if LINE_MODEL
572.     char* data[8];
573. #endif
574. #if CIRCLE_MODEL
575.     char* data[10];
576. #endif
577.     ifstream iFile("Problem A.csv");
578.     if (!iFile.is_open())
579.     {
580.         errorReport("Can't open .csv file.");
581.         exit(-1);
582.     }
583.     char buffer[256];
584.     int i = 0, j = 0;
585.     char* tmp;
586.     const char *d = ",";
587.     while (!iFile.eof())
588.     {
589.         iFile.getline(buffer, 100);
590.         tmp = strtok(buffer, d);
```

```
591.         if (1 <= i && i <= 100) {
592.             while (tmp)
593.             {
594.                 data[j++] = tmp;
595.                 tmp = strtok(NULL, d);
596.             }
597.             obarr[i - 1].mark = 0;
598.             obarr[i - 1].seq = i - 1;
599.             obarr[i - 1].oriseq = atoi(data[0]);
600.             obarr[i - 1].begint = atoi(data[1]);
601.             obarr[i - 1].endt = atoi(data[2]);
602.             obarr[i - 1].startcor[0] = atoi(data[3]);
603. #if LINE_MODEL
604.             obarr[i - 1].startcor[1] = obarr[i - 1].startcor[2] = 0;
605.             obarr[i - 1].velocity = atoi(data[4]);
606.             obarr[i - 1].pj = atoi(data[5]);
607.             obarr[i - 1].second[0] = atoi(data[6]);
608.             obarr[i - 1].second[1] = atoi(data[7]);
609. #endif
610. #if CIRCLE_MODEL
611.             obarr[i - 1].startcor[1] = atoi(data[4]);
612.             obarr[i - 1].startcor[2] = atoi(data[5]);
613.             obarr[i - 1].velocity = atoi(data[6]);
614.             obarr[i - 1].pj = atoi(data[7]);
615.             obarr[i - 1].second[0] = atoi(data[8]);
616.             obarr[i - 1].second[1] = atoi(data[9]);
617. #endif
618.         }
619.         j = 0;
620.         i++;
621.     }
622. }
623. 
624. //打印结果
625. void printResult(int* resarr, int id,int AN,int flag)
626. {
627.     if(flag == 0)
628.     {
629.         printf("Monitor [%d] traced Actual num %d target:\n" ,id ,AN);
630.         printf("Target id\tTarget seq\tsj\tsj+pj\taj\tbj\n");
631.         for(int j = 0;j < AN; j++)
632.         {
633.             printf("\t%d\t\t%d\t%d\t%d\t%d\t%d\n",obarr[resarr[j]].oriseq,resarr[j]
    ,sjSet[id-1][resarr[j]],sjSet[id-1][resarr[j]]+pj[resarr[j]]-
    1,aj[resarr[j]],bj[resarr[j]]);
```

```
634.         }
635.     }
636.     else
637.     {
638.
639.         printf("Monitor new traced Actual num %d target:\n" ,AN);
640.         printf("Target id\tTarget seq\tsj\tsj+pj\taj\tbj\n");
641.         for(int j = 0;j < AN; j++)
642.         {
643.             printf("\t%d\t\t%d\t%d\t%d\t%d\t%d\n",obarr[resarr[j]].oriseq,resarr[j]
    ,sjSet[id-1][resarr[j]],sjSet[id-1][resarr[j]]+pj[resarr[j]]-
    1,aj[resarr[j]],bj[resarr[j]]);
644.
645.             for(int k = 0; k < pj[resarr[j]]+bj[resarr[j]]-aj[resarr[j]]+1;k++)
646.             {
647.                 if(k==0)
648.                     printf("In [sj,sj+pj]:\n");
649.                 if(k==pj[resarr[j]])
650.                     printf("In [aj,bj]:\n");
651.                 printf("This moment traced by monitor[%d] \n",mid[k]);
652.
653.             }
654.
655.             printf("\n");
656.         }
657.     }
658. }
659.
660.
661. bool compare(const int &a, const int &b)
662. {
663.     return a>b;
664. }
665.
666. bool compare1(const int &a, const int &b)
667. {
668.     return a<b;
669. }
670.
671. static int* random(int n,int i)
672. {
673.
674.     int index, tmp, k;
675.     srand(time(NULL));
676.     int* a = new int[n-1];
```

```
677.     int* b = new int[n-1];
678.     SelectedPtr node = fs->objectarr[i].subs;
679.     for(k = 0;k < n-1;k++)
680.     {
681.         a[k] = bj[node->obseq]-aj[node->obseq]+pj[node->obseq];
682.         b[k] = a[k];
683.         node = node->next;
684.     }
685.     sort(a,a+n-1,compare);
686.     int* c = new int[n-1];
687.     int j = 0;
688.     for(i = 0;i< n-1;i++)
689.     {
690.         for(j = 0;j < n-1;j++)
691.         {
692.             if(b[i] == a[j])
693.                 c[i] = j;
694.         }
695.     }
696.     return c;
697. }
698. 
699. int main()
700. {
701.     //探测器数量
702.     int amount;
703.     //探测器集合
704.     Mset* m = new Mset;
705.     //二次追踪贪心算法解集
706.     SecondT* secset = NULL;
707. 
708.     readExcel();
709.     //times 控制循环次数
710. 
711.     int times;
712.     int i = 0;
713.     if(DEBUG)
714.         times = 1;
715.     else
716.     {
717.         printf("==Please input loop times: ");
718.         scanf("%d",x);
719.     }
720.     while (i<times) {
721.         for (int i = 0; i < N; i++)
```

```
722.         {
723.         tmpobarr[i] = obarr[i];
724.         }
725.         ObjectHandle(tmpobarr);
726.         if (DEBUG)
727.         {
728.             printf("==Monitor amount: 5\n");
729.             m->amount = 5;
730.             amount = 5;
731.         }
732.         else
733.         {
734.             printf("==Please input Monitor amount: ");
735.             scanf("%d", &amount);
736.             m->amount = amount;
737.         }
738.         secset = SecondTraceGreedy(tmpobarr);
739.         shuffle(fs);
740.         getArrangement(m, secset, amount, 0);
741.         if(amount == 5){
742.             //收集剩余区间
743.             Monitor tmpm;
744.             getAvailableSlot(m);
745.             tmpm.id = 6;
746.             tmpm.slot = ASlot;
747.             CheckRem(&tmpm,secset);
748.         }
749.         i++;
750.     }
751.
752.     system("PAUSE");
753.     return 0;
754. }
```

（3）errormsg.h

```
1.  /**
2.   ** statement of error report function
3.   ** @name:errormsg.h
4.   ** @author:3160104633@zju.edu.cn
5.   ** @date:2019/5/13
6.   **/
7.
```

```
#ifndef _ERRORMSG_H_
#define _ERRORMSG_H_
#include <stdarg.h>
#include <stdio.h>

void errorReport(char* errormsg, ...);

#endif // _ERRORMSG_H
```

（4）errormsg.cpp

```
#include "errormsg.h"

void errorReport(char* errormsg, ...)
{
    va_list args;
    fprintf(stderr, "ERROR: ");
    va_start(args, errormsg);
    vfprintf(stderr, errormsg, args);
    va_end(args);
    fprintf(stderr, "\n");
    return;
}
```