奖金范围 & 奖金优化

无线技术部 李小朋

【摘要】本文介绍竞彩足球混投 n 串 1 的奖金优化算法，包括平均、搏冷、搏热优化以及最大最小奖金范围计算。本算法具有很好的可扩展性，对于混合过关、设胆以及其它彩种(北单、竞篮)均能适用。

关键词：奖金优化 互斥关联 位图 分类器 连通分量

1. 奖金优化

奖金优化在彩民调整奖金范围时是一个很好的参考，有三种优化方式。平均优化是即使命中都是最低赔，奖金也会远大于本金，不会出现中奖不盈利的尴尬；搏冷优化既能保证命中其它热门注时无亏损，又可保证在搏冷门时，拿到最高价值的回报率；博热优化使投注的本金可以足够多得投注在热门的场次上，让命中热门场次收益最大化。

1.1. 平均优化

根据平均优化的定义，其实是针对投注列表合理分配用户的投注金额，使每一注的奖金尽可能地接近且大于本金。由于分配给每一注的投注倍数必须是整数，所以这是一个整数规划问题，即求解一组投注倍数的整数解，满足如下的目标函数与约束条件：

$$\text{目标函数}: z = \min\left\{\sum_{i,j=1}^{N}(p_i n_i - p_j n_j)^2\right\}$$

$$\text{约束条件}: \begin{cases} n_i \geq 1 \\ M = \sum_{i=1}^{N} n_i \\ p_i n_i \geq M \end{cases}$$

p_i 是单注奖金,n_i 为对应的投注倍数，M 为投注本金、N 为投注列表注数。其中第三个约束条件受单注奖金影响，不一定能满足：

1.2. 优化算法

1.2.1. 单一玩法

若不考虑投注之间的关联性(如单一玩法)，也就是说不会存在某一注中奖则另一注必中的情况，那么奖金优化是一个简单的贪心问题。我们可以将投注本金分步投注，若保证每一步的局部最优解，那么当所有的本金用完则可得到全局最优解。其算法步骤大致如下：

(1).以各注奖金为大小建立最小堆

(2).每次将奖金最小的投注倍数加 1

(3).调整最小堆，判断是否还有余额

(4).重复(2)~(3)，直至余额为 0

伪码：

```
build_min_heap(&heap[0], tzlist)
    while m>0:
        heap[0]->sprize += heap[0]->uprize
        heap[0]->num++
        min_heapfy(&heap[0])
        m -= 2.0
```

不难得到算法的时间复杂度为 O(mlog(n))，其中 m 为投注本金，n 为投注数；空间复杂度为 O(n)。

这里需要注意的细节有两点：一个是找最小奖金投注时不要排序或者遍历，维护一个最小堆即可；二是调整投注顺序时，用对象指针操作提高效率。

1.2.2. 混合玩法

若考虑投注之间的关联性，例如混投中若”1:0”玩法中奖，则”胜“、”-1 平”、”进 1 球”等玩法均必中奖。这种情况下平均优化之后用户获得奖金的对象不应只是单注彩票，而是相关联的一组彩票，即目标函数变为：

$$\text{目标函数}: z = \min\left\{\sum_{i,j=1}^{N_G}\left(\sum_{ii=1}^{G_i} p_{ii}n_{ii} - \sum_{jj=1}^{Gjj} p_{jj}n_{jj}\right)^2\right\}$$

这种情况下，算法稍微复杂一些：

(1).将投注按关联关系分组(投注分组将在奖金范围计算中详细描述)

(2).将分组建立一个最小堆，组内部各注再建立一个最小堆

(3).每次找总奖金最小的组，将组内奖金最小的投注倍数加 1

(4).调整组内最小堆，调整分组最小堆，判断余额

(5).重复 3~4，直至余额为 0

1.2.3. 贪心算法问题

贪心算法不是解决这个非线性整数规划问题最准确的算法。例如对于如下情况：

投注编号	单倍奖金	倍数	总奖金
1	17	3	51
2	50	1	50

若用户再增加一注，按照贪心策略则投注在第 2 注上，此时奖金差为 49 元；更合理的方案是投注在第 1 注上，此时奖金差为 18 元。考虑到奖金优化仅为用户提供一个优化参考，贪心算法简单便于理解。

1.3. 搏冷搏热

有了平均优化作为基础，搏冷搏热优化算法就变得简单：在贪心过程中，若各注奖金已能收回本金且还有投注余额，则将余额全部投注赔率最高或最低投注上。

2. 奖金范围

2.1. 互斥与关联

互斥关系表示条件 A 与条件 B 不能同时存在，双向互斥；关联关系是指条件 A 成立的

请款下条件 B 必成立，具有方向性，反之则不一定。互斥与关联关系是奖金范围计算的基础，最大最小奖金的可以定义如下：

“最大奖金：奖金最大的不互斥投注集合”

“最小奖金：奖金最小的相关联投注集合”

因此针对用户的投注列表，必须首先理清投注之间的互斥与关联关系，而这又建立在每一场球赛玩法之间互斥与关联基础之上。

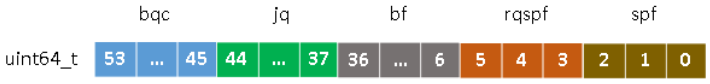
2.2. 玩法关系位图

2.2.1. 玩法 bitmap

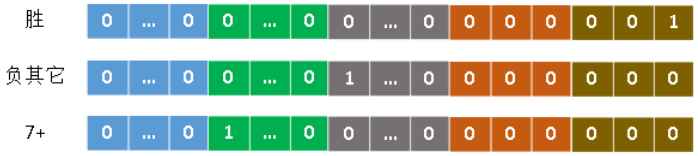
对于竞彩足球混投，一场球赛共有如下 5 大类共 54 种玩法：

胜平负		让球胜平负		比分				总进球				半全场	
胜	4.00	胜	1.83	1:0	8.50	2:0	17.00	0	8.00	1	3.55	胜胜 5.40	胜平 15.00
平	3.30	平	3.45	0:0	8.00	1:1	6.50	2	2.90	3	3.50	平胜 7.50	平平 4.40
负	1.75	负	3.45	展开				展开				展开	

为了能够方便快速地表征各种玩法并分析其互斥与关联关系，可以将 54 种玩法压缩到一个无符号的 64 位整数中(uint64_t)，每一位表示一种玩法仍有富余，形成一个 bitmap：

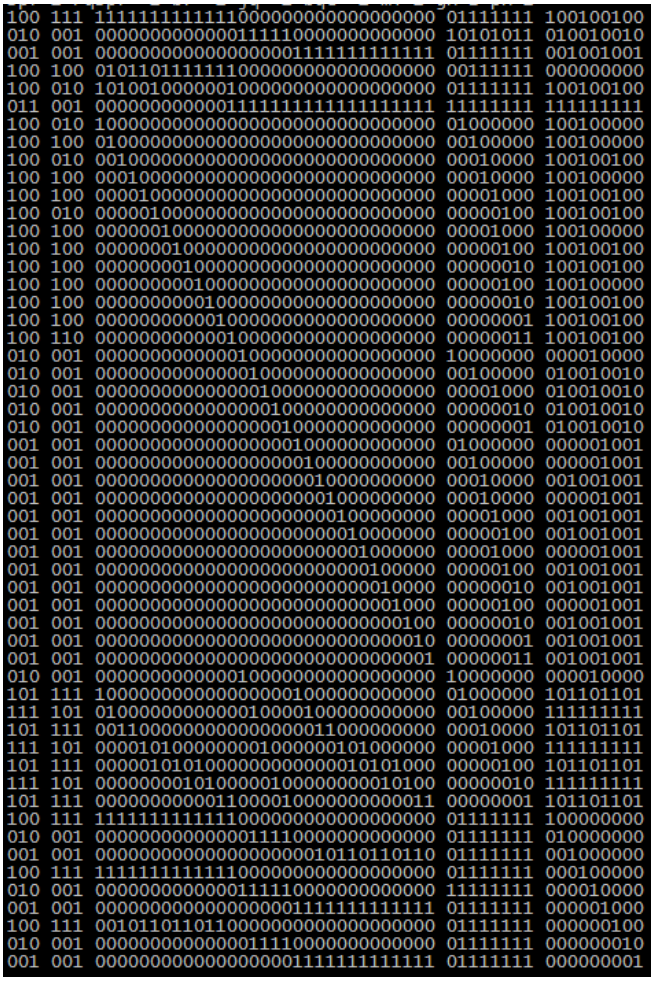


用 bitmap 来表示各种玩法，只需将其对应的二进制位置为 1 即可，例如：



通过枚举一种玩法与其他玩法两两之间互斥关系，结合位运算操作，我们可以得到这一玩法的互斥关系图。下图是 54 种玩法(让 1 球)的互斥关系图,一个 54x54 的稀疏矩阵。

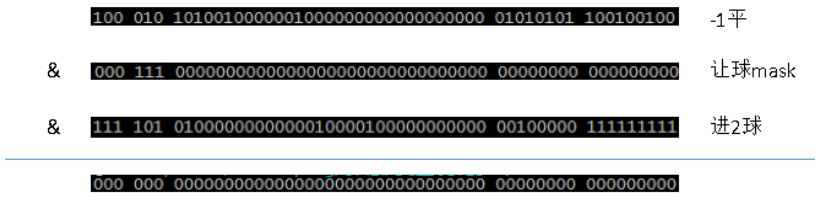
```
#define UNSET(x,y)  ((x) ^= (1ULL << (y)))
#define SET(x,y)    ((x) |= (1ULL << (y)))
#define MASK54      0x3fffffffffffff
```



2.2.2. 关系算子

有了各种玩法两两互斥关系，通过以下两种关系算子快速地测试互斥与关联关系。

(1).玩法 A 与 B 互斥: (bitmap(A) & segmask(A)) & bitmap(B)

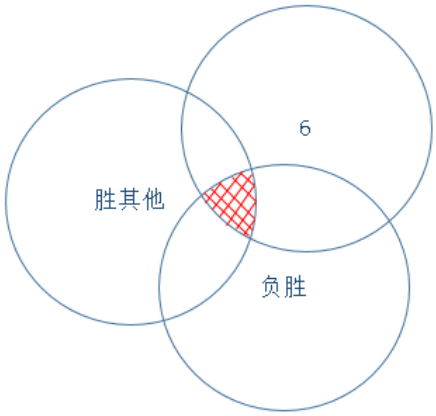


(2).玩法 A 与 B 关联: (bitmap(A) & segmask(B)) ^ (bitmap(B) & segmask(B))

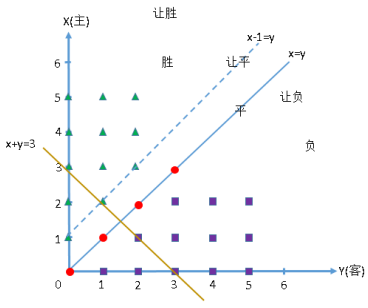
-1胜	100 100		100 111	胜
胜平负mask	111 000	&	111 000	胜平负mask
	100 000	^	100 000	
-1胜	100 100		100 111	胜
让球mask	000 111	&	000 111	让球mask
	000 100	^	000 111	

2.3. 多玩法互斥

遗憾的是互斥关系并不仅局限在两两玩法之间，也可能存在两两之间不互斥但多玩法之间互斥的情况，玩法关系位图无法检测出这种互斥关系：



此时需要判断多玩法组合的线性规划问题是否有整数解来判断互斥关系：若无整数解则互斥。如图所示，竞足混投的各种玩法的均可抽象到比分坐标系中：




```
#define SPF_S(x,y)   ((x) >  (y))
#define SPF_P(x,y)   ((x) == (y))
#define SPF_F(x,y)   ((x) <  (y))
#define RQSPF_S(x,y,rq) ((x) + (rq) >  (y))
#define RQSPF_P(x,y,rq) ((x) + (rq) == (y))
#define RQSPF_F(x,y,rq) ((x) + (rq) <  (y))
#define BF_SQT(x,y)     ((x) >  (y) && ((x) > 5 || (y) > 2))
#define BF_PQT(x,y)     ((x) == (y) && (x) > 3)
#define BF_FQT(x,y)     ((x) <  (y) && ((x) > 2 || (y) > 5))
#define JQ(x,y,jq) ((jq) > 6 ? (x) + (y) >= (jq ): (x) + (y) == (jq))
#define BQC_SS(x,y)  ((x) >  (y))
#define BQC_SP(x,y)  ((x) == (y) && (y) > 0)
#define BQC_SF(x,y)  ((x) <  (y) && (x) > 0)
#define BQC_PS(x,y)  ((x) >  (y))
#define BQC_PP(x,y)  ((x) == (y))
#define BQC_PF(x,y)  ((x) <  (y))
#define BQC_FS(x,y)  ((x) >  (y) && (y) > 0)
#define BQC_FP(x,y)  ((x) == (y) && (y) > 0)
#define BQC_FF(x,y)  ((x) <  (y))
```

那么结合玩法关系位图与多玩法互斥检测，给定一场比赛的玩法列表求解所有的不互斥组合(经测试最多约 2000+)的算法如下：

(1).将一场比赛的玩法初分类(spf,rqspf,bf,jq,bqc)

(2).各分类间做笛卡尔积

(3).两两玩法间用 bitmap 做互斥剪枝(能检测 99%的互斥关系)

(4).多玩法间互斥检测(能检测 0.99%的互斥关系)

a.若存在 bf 玩法，代入其他玩法测试宏

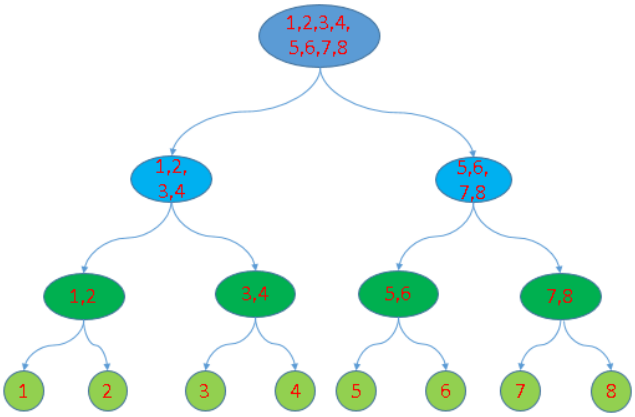
b.若不存在 bf 玩法，则 x,y∈[-10,10]范围内求解(折衷)

(5).重复(1)~(4)求解所有比赛的不互斥组合

时间复杂度：O(n^5)，n 为各类玩法的数量，很小；空间复杂度:O(2000*5*m)，m 为比赛的场次。

2.4. 最大奖金

有了前一节的所有比赛的不互斥组合之后，我们可以利用各场比赛的不互斥玩法组合看做分类器，逐层对投注列表做分类，形成一颗 m 叉树，叶子节点即为不互斥的投注组合。



求解最大奖金就是遍历一颗多叉树的所有叶子节点，找到奖金最大的那个组合：

```
dfs_max_prize(int m, uint64_t tzlist):
    if m == MAX_MATCH_NUM || count_of_1(tzlist) == 1:
        cal_max_prize(tzlist)
        return
    for i in groupnum[m]:
        dfs_max_prize(m+1, tzmap[m][i] & tzlist)
```

其中：tzmap 是将投注记录映射到各场比赛的分组;tzlist 是一个当前节点投注记录的位图

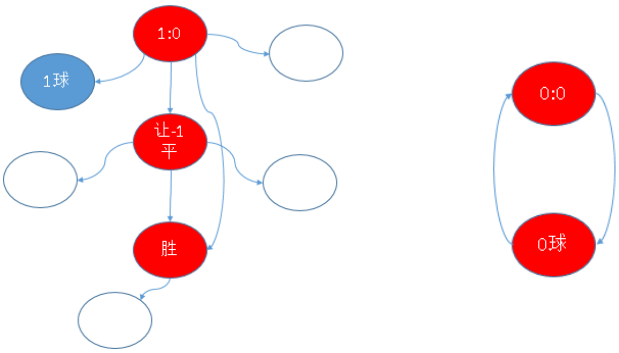
算法的时间复杂度：O($\log_m n$)，m 为各场比赛玩法分组,n 为投注列表注数。

2.5. 最小奖金

对于最小奖金，需要首先构建投注之间的关联关系图，而求解最小奖金就是寻找这个有向图中奖金最小的连通分量。构建关联关系图的伪码如下：

```
for i in tz_num:
    for j in tz_num, i != j:
        if test_inclusion(tzlist[i], tzlist[j]):
            tzlist[i].chld[tzlist[i].chldnum++] = &tzlist[j]
```

算法时间复杂度：O($m*n^2$)，m 为比赛场次,n 为投注数。



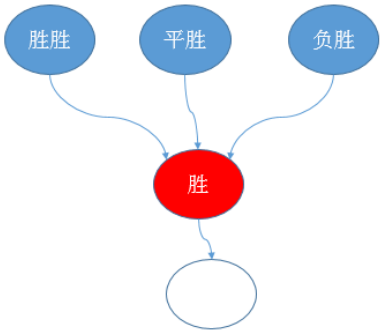
需要注意的两点是(1)同一节点可能有多种路径可达；(2)关联图可能存在环路,如上图所示，因此在搜索时要用游标给访问过的节点着色，避免重复访问；通过维护不同版本的游标，避免不同搜索路径之间的相互影响。

```
for i in tz_num: dfs_min_prize(&tzlist[i], i)
    dfs_min_prize(pTZNODE tz, int times):
        if tz->visited   >   times:
            return
        tz->visited = times+1
        for i in tz.chldnum:
            dfs_min_prize(tz->chld[i], times)
            tz->mprize    += tz->chld[i]->mprize
```

时间复杂度：$O(n\log_m n)$ n 为投注数,m 为各注的子节点数。

2. 6. 玩法全包

玩法全包是类似这样一种情况：[胜、胜胜、平胜、负胜]。正常情况下胜玩法对于半全场玩法没有必然的关联联系，但当用户全包[胜胜、平胜、负胜]时，则必有一种玩法伴随胜玩法同时出现。玩法全包对最大奖金没有影响，对最小奖金有直接影响。



按照最小奖金算法，最小奖金为胜玩法入口搜索到的联通分量。对于玩法全包情况，应该摒弃这一入口，从上游节点作为搜索入口。前提是做好玩法全包检测：

1.用户单场玩法集合位图 ap

2.对于每一种玩法 pl_i 到 5 大玩法区间检测

(1). p = bitmap[pl_i] & segmask[j]

(2). if p & ap == p，则存在全包情况

3.对于包含全包玩法的投注，不作为最小奖金搜索入口

3. 算法扩展

关系位图将具体的算法与投注之间的互斥(关联)关系分离，因此算法本身具有很好的可

扩展性。

3.1. 混合过关

混合过关无论是 m 串 1(m<n)还是 n 串 m，都存在某些投注不是由所有的选择的比赛组成。对于这些投注，我们可以将其比赛场次补齐：即在投注中添加没有的比赛，标记为-1。在判断互斥与关联关系时有如下两条规则：

(1).任何比赛都不与-1 的比赛互斥

(2).任何比赛都关联-1，反正不然

3.2. 设胆

设胆其实影响的是投注拆分阶段，通常我们用比赛玩法拆分投注列表时，用笛卡尔积得到各种串法投注。通过将玩法中补充某种空玩法实现 m 串 1(m<n)：

比赛 A[-1、胜、平] => (-1)、(胜)、(平)

比赛 B[-1、0:0、1:1]=>(-1)、(0:0)、(1:1)

比赛 C[-1、1 球、2 球]=>(-1)、(1 球)、(2 球)

那么将 A 设胆，则将 A 中的空玩法去掉即可：

比赛 A[胜、平] => (胜)、(平)

比赛 B[-1、0:0、1:1]=>(-1)、(0:0)、(1:1)

比赛 C[-1、1 球、2 球]=>(-1)、(1 球)、(2 球)

3.3. 其它彩种

其它彩种，如北单、竞篮，只需定制各个彩种对应的玩法关系位图之后就能套用本文的算法。

4. 结论

(1).拆分投注列表是本文算法的基础。

(2).关系位图能解决 99%的互斥关系，多玩法互斥解决 0.99%。

(3).算法复杂度由投注金额、比赛场次、玩法选择决定。

(4).关系位图将算法与互斥关联关系分离,方便扩展。

(5).在客户端最多 15 场比赛、64 注彩票的前提下，算法效率在 50ms 左右。