足球奖池问题的数学研究

张鑫野
北京师范大学附属实验中学

2024 年 11 月 5 日

摘要

在本篇论文中我们尝试将过往对于足球奖池问题(Football Pool Problem, FPP)的研究结果运用在真实的足彩中，并通过将FPP中的汉明距离更改为欧氏距离后，对小样例及其上下界进行了探究。

关键词: 足球奖池问题，组合，组深度优先搜索

目录

1 引入

足球奖池问题(Football Pool Problem, FPP)最早于1940年代在芬兰运动杂志(*Veikkaaja*)中被提出。问题的描述如下，一场足球比赛中会有$q = 3$种不同结果: 主场球队获胜，平局，客场球队获胜。在足球彩票中，玩家可以花一定的钱对未来确定的n场足球比赛下注，最终获得的奖金取决于预测正确的场数以及下注的钱数。通常情况下，猜对全部的n场或$n-1$场可以获得奖金。获得奖金并不容易，因为可下注的结果呈指数级增长。比如，对于足彩中的“胜负平十四场”，一张彩票需要预测14场比赛，如果想要保证获得头等奖，即猜对n场，玩家需要确保全部的可能性都被买到，即$q^n = 3^{14} = 4,782,969$张。

显然，这样做需要投入与大量的资金，与玩家购买彩票的初心相违。因此，玩家可能会放弃头等奖，并尝试其他的奖项，一般来说，这需要猜对$n-1$或者$n-2$场。尽管玩家为了确保全部猜对需要购买的彩票数量是显而易见的，玩家为了确保øn场正确所需要购买的彩票数量的最小值却是一个很难的问题，事实上，这一问题非常困难。在过去几十年中，数学家与信息学家们从杂志中注意到了这个问题，并将其抽象成数学与信息学语言，探索并研究了很多相关问题。例如：在5场[1]，$6-8$场球赛时FPP的上下界[2]，或当$q = 2$时[3]，即二进制码情况下的FPP问题，被广泛利用在信息研究中。在Verstraten, Brink[4, 5]的论文中，作者讨论了一种相反的情况，逆足球奖池问题(Inverse Football Pool Problem, IFPP), 该问题为求最小需要的下注数，使得全部的场次预测错误。尽管从FPP中抽象出来的数学问题已经被广泛研究，但是将FPP的结果运用到现实中的案例却很少。这大概是出于以下几种原因：

1）大多数FPP变种问题与现实中的足彩很难产生关联

2）对于$q = 3, n = 14$的情况，为了确保能够获得奖金(即猜对13场)，基于过往的研究结论的上下界，玩家至少需要购买166,610[6]张彩票。这对一位真实的玩家来说显然是不现实的。

3）如果奖金数量固定，当玩家数量足够多时，使用FPP方法将失去套利空间。

但对于n较小的情况，理论研究仍是有用的。例如在荷兰甲级联赛中[4]，一张彩票只需要预测9场比赛。玩家只需要购买1,269张彩票就可以保证猜对8场[6]，尽管数量仍然很多，但只要奖金足够多，就仍在可接受的范围之内。

在本文第二节，我们将给出FPP的数学表达。在第三节，我们将计算不同策略下彩票的预期收入以探究FPP是否可以帮助现实生活中的足彩问题。在第四节，我们将具体探究一种FPP的变种。在第五节，我们将把FPP抽象为集合覆盖问题，并简要说明其在现实生活中的应用价值。在第六节，我们简要总结整篇论文。

2 FPP的数学表达

针对一次有n场比赛，每场比赛有q种结果的足球赛事，我们定义有限域上的向量空间$F_q^n := \{0,1,2,\cdots,q–1\}^n$。当$q=3$时，$F_q^n$中的每一个词汇（向量），都可以代表$n$场比赛的一种预测结果。在这里，我们定义0为主场胜，1为平局，2为客场胜。汉明距离(The Hamming Distance)为两个等长的字符串中字符不相同的位置数量。数学表达为 $d^H(w,v) := |A|, A = \{i \mid w_i \,!= \, v_i\}$

定义2.1 在FPP中，给定有限域上的向量空间F_q^n。设C为F_q^n的子集，对任意的实数$R \in \mathbb{N}$且$0 \leq R \leq N$，我们说C是F_q^n半径为R的一个覆盖，如果$\forall w \in F_q^n, \exists v \in C$，$d^H(w,v) \leq R$。我们将以$q,n,R$为参数的FPP问题记为$FPP_q\{n,R\}$

定义2.2 设集合$\mathrm{S} = \{\mathrm{C} \mid \mathrm{C}$是 $FPP_q\{n,R\}$ 中的一个覆盖 $\}$，我们记$FPP_q\{n,R\}$的最优解$min\{|S_i|\}$为 $K_q\{n,R\}$

定义2.3 对一个n维向量$(a_1,a_2,a_3,\cdots,a_n)$，我们称其为一个整点，如果对任意$i \in [1,n], a_i \in \mathbb{Z}$

事实上，F_q^n 可以被理解为一个n维直角坐标空间里由$(0,0,0,\cdots,0)$到$(q-1,q-1,q-1,\cdots,q-1)$中每一个整点所组成的集合。在$R=1$时，对F_q^n中任意一个包含单个词汇w的子集w，它所可以覆盖到的词汇为平行于每一个坐标轴且过w的直线所经过的所有整点构成的集合。例如：当$q=3, n=2$时，F_3^2可以被表示为平面直角坐标系中从$(0,0)$到$(2,2)$内的每一个整点。而$C = \{(0,1),(1,2),(2,2)\}$即为$F_3^2$的一个半径为1的覆盖，如下图1所示:

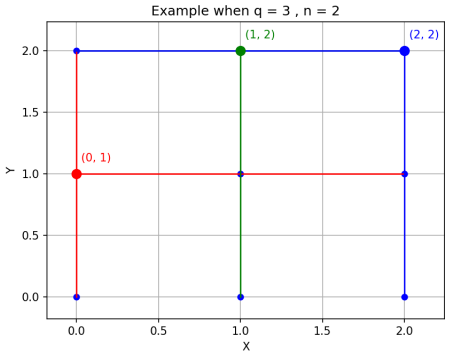

图 1: $q=3, n=2$ 时的例子

可以看到，点$(0,1)$ (第一场主场胜第二场平)可以覆盖到$(0,0)$, $(0,1)$, $(0,2)$, $(1,1)$, $(2,1)$ ，同理可得点$(1,2)$和点$(2,2)$的覆盖范围，不难看出，这三个点覆盖到了$(0,0)$至$(2,2)$中的所有整点，故C为F_3^2的一个半径为1的覆盖。

又因为对于每一个点，必须至少存在一个点在该行或该列才可以覆盖到该行，因此不难证明：当$k \leq 2$时，不存在F_q^n的一个子集$|C| = k$，使得其为F_q^n半径为1的覆盖。因此，$FPP_q(n,R)$的最优解$K_3(2,1) = 3$。

3 FPP在现实中的应用

在本节中，我们将讨论FPP中的结论在现实足彩中的应用。由于足球彩票中的情况过于复杂，我们在此进行简化，假设总奖金是一个固定的数，首先假设只有1个玩家。

我们仍然采用中国足彩网(zgzcw.com)中的十四胜负平玩法，依据相应规则[7]，十四场胜负平仅设立2个奖项，头等奖与二等奖分别需要猜中n与$n-1$场比赛，头等奖平分70%的奖金，二等奖平分30%的奖金，我们假设奖金共有p元，则一等奖为$0.7p$元，二等奖为$0.3p$元，假设每一场比赛的结果是随机的，胜负平概率相等，均为1/3。我们将通过分别计算与比较随机下注与使用FPP中结论的预期收益以探究FPP是否能够在现实生活中帮助我们套利。

我们定义$P(x)$为发生事件x的概率，”F”为中了头等奖，”S”为中了二等奖。

我们先来讨论n较小，例如$n=4$的情况

方法一：随机下注

假设我们随机购买一注彩票，下注四场比赛，那么头等奖的概率$P(F)=\frac{1}{3^4}=\frac{1}{81}$，二等奖的概率$P(S)=\frac{2\times C_4^1}{3^4}=\frac{8}{81}$

依据数学期望计算公式$E(x)=P(x)\times k(x)$，可以计算出随机下注的期望收益:

$E(\text{随机下注})=P(F)\times 0.7p+P(S)\times 0.3p=\frac{3.1p}{81}$

减去成本，即购买彩票的2元，就可以得到我们的净收益 总净收益 $=\frac{3.1p}{81}-2$

方法二：使用FPP的结果

假设我们使用FPP中的策略，即购买若干张彩票，保证最多只错一场。

因为$K_3(4,1)=9$[6]，即我们需要购买9张不同的彩票以保证最多只错一场。

此时，$P(F)=\frac{9}{81}$（9张彩票都有可能中头等奖），$P(S)=1$

计算出使用FPP的期望收益:

$E(\text{使用FPP})=P(F)\times 0.7p+P(S)\times 0.3p=\frac{30.3p}{81}$

减去成本，即购买9张彩票的18元，得到净收益为：$\frac{30.3p}{81}-18$

比较结果如下图2所示:

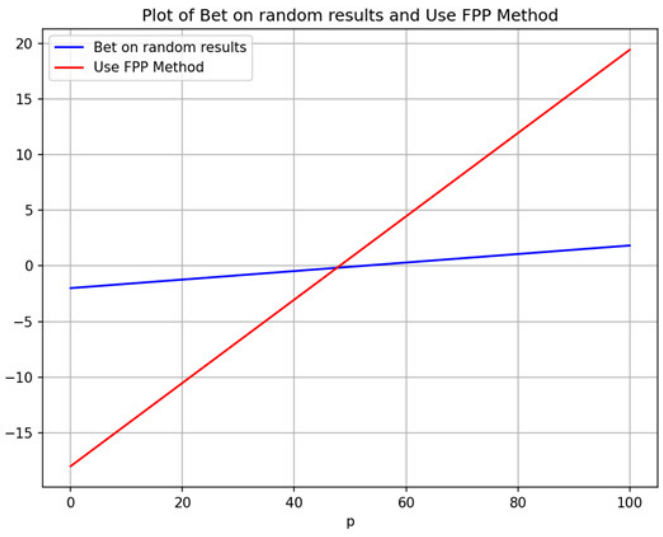

图 2: 使用FPP方法与随机下注n=4时期望收益比较

可以看出，当总奖金大约小于50元时，随机下注期望得到更高的钱，而当奖金高于50元时，使用FPP方法更胜一筹。

接下来，讨论n较大的情况，例如n=9：

同理，计算出使用两种方法的数学期望

方法一：随机下注

获奖的概率分别为$P(F)=\frac{1}{3^9}=\frac{1}{19,683}$，$P(S)=\frac{2\times C_9^1}{3^9}=\frac{18}{19683}$

依据数学期望计算公式$E(x)=P(x)\times k(x)$，可以计算出随机下注的期望收益：

$E(\text{随机下注})=P(F)\times 0.7p+P(S)\times 0.3p=\frac{6.1p}{19,683}$

减去成本，即购买彩票的2元，就可以得到我们的净收益

总净收益 $=\frac{6.1p}{19,683}-2$

方法二：使用FPP的结果

对于$K_3(9,1)$并没有明确的最优解，但其上界为1,269，即至多买1,269张彩票就可以保证最多只错一场[6]

此时，$P(F)=\frac{1,269}{19,683}fiP(S)=1$

$E(\text{使用FPP})=P(F)\times 0.7p+P(S)\times 0.3p=\frac{6,786.2p}{19,683}$

减去成本，即购买1,269张彩票的2,538元，得到净收益为：$\frac{6,786.2p}{19,683}-2,538$元

比较结果如下图3所示:

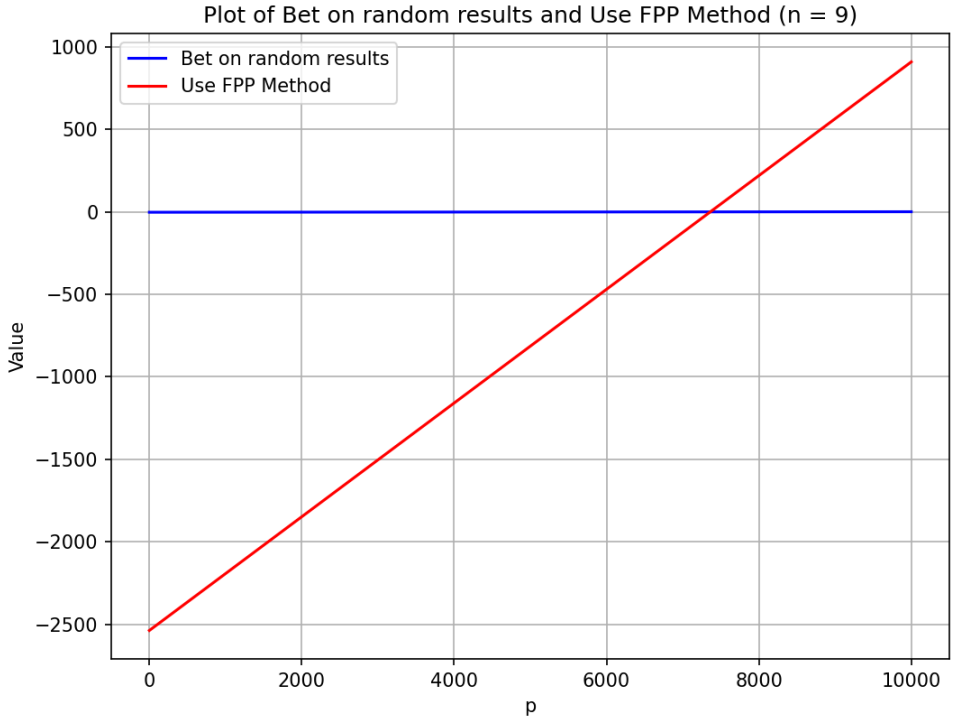

图 3: 使用FPP方法与随机下注n=9时期望收益比较

可以看出，在n=9时，只有奖金在超过越7,300元时，使用FPP方法的期望收益才能高于随机下注的收益。并且随着n的增长，两种方法期望收益相同时的总奖金数也在逐渐增加，为了进一步探

究两种方法收益的差别，我们绘制两种方法收益相同时总奖金数量与n的折线关系图4，如下图所示：

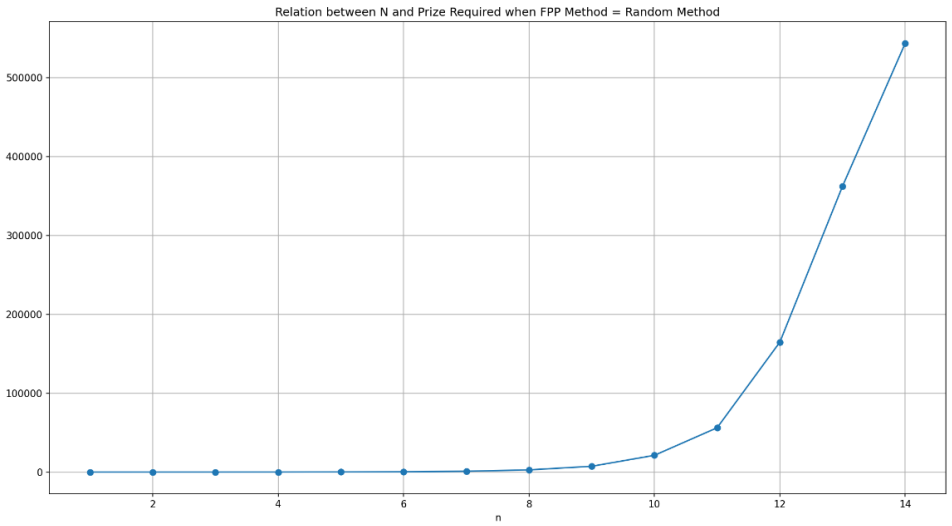

图 4: n与使用FPP方法和随机方法期望收益相同时总奖金数关系图

可以看出，n与总奖金数的关系图可以近似为指数关系，而由中国足彩网[7]规则中可知，单注最高奖金不超过500万元，因此，可以确认使用FPP后确实会比随机下注获得更高的利润，但由于需要购买的彩票数量过多且现实中足球彩票的情况更加复杂，并且我们仅仅将FPP的结论与随机下注进行比较，因此通过使用FPP方法在现实生活足彩中获得优势并不现实。

在现实中的多玩家情况下，一旦有人发现使用FPP方法可以获得稳定收益，那么一定会有更多的玩家加入，并进行套利，由于奖金数量是固定的不变，越来越多的玩家加入必然导致每位玩家得到的奖金的稀释，最终会导致使用FPP方法没有套利空间。

4 FPP的变种问题

4.1 问题定义

在上述问题中，我们所讨论的FPP是基于汉明距离$d^H(w,v)$而进行的。而在这一节中，我们将两个词汇之间的距离从汉明距离更换为欧氏距离。即$\sqrt{\sum_{i=1}^{n}(w_i-j_i)^2}$

因此，一个由F_q^n中单个词汇组成的集合{w}可以覆盖到的范围由平行于每一个坐标轴且过w的直线所经过的整点所组成的集合变为了以w为圆心，R为半径的n维球，球上与球内所覆盖的所有整点即为{w}可覆盖到的范围。

我们记使用欧氏距离的$FPP_q(n,R)$问题为$EFPP_q(n,R)$，其最优解为$KE_q(n,R)$。

我们首先仅讨论n=2时的情况，此时，该问题转化为如下问题：

问题4.1 给定一个特定的二维平面，其中存在$(0,0)$至$(q-1,q-1)$的方格，其中给定一个半径R，我们规定一个圆覆盖范围为这个圆上以及圆内的整点，求最少需要多少个圆心在格点的圆可以将$(0,0)$至$(q-1,q-1)$的所有格点全部覆盖。

4.2 问题解决思路

对于该问题，我们可以使用深度优先搜索以及对他的剪枝进行实现，其中的步骤为：

1) 对于每一个点，预处理以该点为圆心，R为半径的圆上与圆内经过的所有格点，并存入数组中。

2) 深度优先搜索：每一次从(0, 0)开始逐个点遍历至(q, q)，一旦遇到没有被覆盖到的点，选中，使用在1)中预处理的点，将以该点为圆心内的点标记覆盖，并递归至下一层循环。使用计数器记录仍未被选中的格点数量，一旦计数器清零，记录使用圆的数量m，并与当前最优解进行比较，若m≤当前最优解，更新最优解为m，并记录这种构造，并返回到上一层递归。

3) 通过python进行画图输出。

4.3 结果说明

如下表5所示，记录了在使用欧氏距离时，$q,r\in[1,5]$的最优解。（书写的代码可以在附录中找到）

R \ q	1	2	3	4	5
1	1	2	3	4	7
2	1	1	1	3	1
3	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1

图 5: q，R在1到5 区间时的最优解结果 Time Elapsed: 3,479 Seconds

由于该问题是NP问题，即无法在多项式复杂度内解决，在$R=1$时时间消耗最为显著，在舍掉部分最优解并扩展范围后，我们得到了图6：

R \ q	1	2	3	4	5	6	7	8
1	1	2	3	4	7	12*	18*	26*
2	1	1	1	3	4	4	6*	9*
3	1	1	1	1	1	3	4	4
4	1	1	1	1	1	2	2	3
5	1	1	1	1	1	1	1	2
6	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1

图 6: q，R在1到8区间时的结果

其中，*代表非最优解，这是因为最优解无法再短时间内通过代码得出。

图7表示了n, R均小于等于5时的构造，图6中的完整构造结果可以在附录中找到。

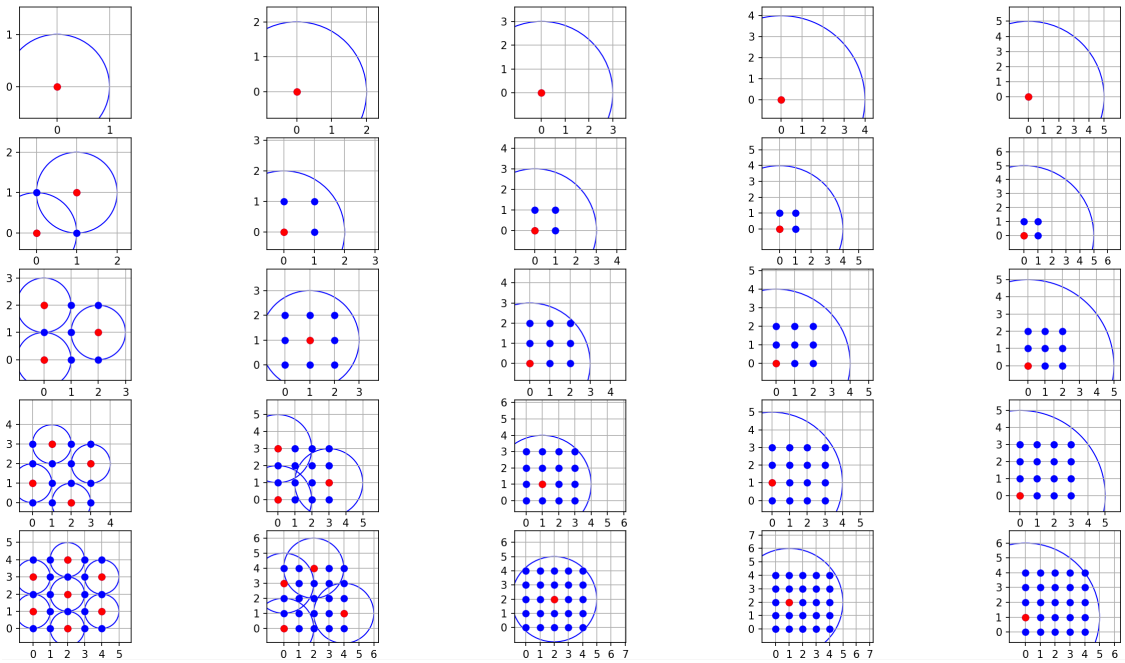

图 7: q，R在1到5区间时的最优解构造

图中，第i行第j列代表$q = i$，$R = j$时的构造，圆心使用红色点代替，而被标注出的格点为需要覆盖到的格点（包括红色和蓝色）

虽然我们无法准确求出n, R过大时的准确结果，我们仍然可以通过组合估计其上下界以及部分情况下的最优解。

定理 4.3.1 对于任意偶数q, 总能保证当$R \geq \left\lceil \frac{q}{2} \times \sqrt{2} \right\rceil$时存在最优解1

证明: 显然，我们令$k = \frac{q}{2}$, 设圆O的圆心在(k , k)，此时距离圆心最远的四个点为(q, q), (0, 0), (0, q), (q, 0)，距离均为$\frac{q}{2} \times \sqrt{2}$，因此，当$R \geq \frac{q}{2} \times \sqrt{2}$ 时，总能够覆盖到所有格点，又因为R需要为整数，因此我们给结果进行向上取整。

定理 4.3.2 对于任意奇数q, 总能保证当$R \geq \left\lceil (\left\lfloor \frac{q}{2} \right\rfloor + 1) \times \sqrt{2} \right\rceil$时存在最优解1

证明: 与定理4.3.1的证明方法相似，我们令$\left\lfloor \frac{q}{2} \right\rfloor$, 设圆O的圆心在(k , k)，此时距离圆心最远的点为(q, q)，距离为$(\left\lfloor \frac{q}{2} \right\rfloor + 1) \times \sqrt{2}$，因此，当$R \geq (\left\lfloor \frac{q}{2} \right\rfloor + 1) \times \sqrt{2}$时，总能够覆盖到所有格点，同样的，我们对结果进行向上取整。

定理4.3.3 对于任何相同的n, 总能保证$EK_q(n, R) \geq EK_q(n, R + p)$, 其中p为[1 , q − R]中的整数

证明: 考虑$EK_q(n, R)$时的构造，我们发现，当R改为R+p时，圆的半径扩大，因此原来可以被覆盖到的地方仍然可以被覆盖到，证毕。

定理4.3.4 假设一个在原点，半径为R的圆可以覆盖到M个格点，则对于任意的q, R, 都有$EK_q(2, R) \geq \left\lceil \frac{q^2}{M} \right\rceil$

证明: 对于一个$q \times q$的网格，其中共有q^2的格点，由于一个在原点上半径为R的圆可以覆盖到M个格点，我们假设每一个格点都仅被一个圆覆盖，但由于实际摆放时不同圆之间可能会出现重叠与浪费，因此最优解的结果应大于等于$\frac{q^2}{M}$

对于$n \geq 3$时的情形，上述算法仍然适用，但随着n的增长，时间消耗也将呈指数增长。

上述公式也同样可以扩展到$n \geq 3$的情况

定理 4.3.5 对于任意偶数q以及任意的n,总能保证当$R \geq \left\lceil \frac{q}{2} \times \sqrt{n} \right\rceil$时存在最优解1

定理 4.3.6 对于任意奇数q以及任意的n, 总能保证当$R \geq \sqrt[2]{\left\lceil \frac{q}{2} \right\rceil^2 \times \left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{q}{2} \right\rfloor^2 \times \left\lfloor \frac{n}{2} \right\rfloor}$时存在最优解1

5 FPP变种的其他应用

FPP隶属于集合覆盖问题。集合覆盖问题是一个NP完全问题，该问题为给定全集U以及U的部分子集集合S，求最少需要从S中选取多少个元素，使得这些子集的并集等于全集U，集合覆盖问题可以通过线性规划与贪心算法进行，除此之外，在过往的研究中使用了不同的方法来计算集合覆盖问题[8]并降低其复杂度[9]。

而集合覆盖问题在现实生活中也可以被广泛使用，例如：现实生活中救护车的分布[10]，船舶的停靠时间与调度[11]，以及信号基站的选址[12]，可以看出，除了足球彩票外，FPP的变种问题在现实生活中也具有很大的应用空间。

6 总结

在本论文中，我们首先探究了足球奖池问题在足球彩票中的应用，并探究了将其问题中汉明距离换位欧几里得距离时在n=2时解决问题的算法与部分结果，该算法仍然有非常大的改进空间，其中，很大一部分运算出现了重叠，产生了浪费。该代码在$n \geq 3$时的情况仍然没有进行探究。最后，我们探究了FPP变种问题的其他应用。

7 参考文献

参考文献

[1] Kamps H J L , Lint J H V .The football pool problem for 5 matches[J].Journal of Combinatorial Theory, 1967, 3(4):315-325.DOI:10.1016/S0021-9800(67)80102-9.

[2] Laarhoven P J M V , Aarts E H L , Lint J H V ,et al.New upper bounds for the football pool problem for 6, 7, and 8 matches[J].Journal of Combinatorial Theory, 1989, 52(2):304-312.DOI:10.1016/0097-3165(89)90036-8.

[3] Helleseth,T,Klove,et al.On the covering radius of binary codes (Corresp.)[J].Information Theory, IEEE Transactions on, 1978, 24(5):627-628.DOI:10.1109/TIT.1978.1055928.

[4] Verstraten K P F .A Generalization of the Football Pool Problem[J].[2024-09-15]

[5] Brink D .The Inverse Football Pool Problem[J]. 2009.

[6] $http://www.sztaki.hu/\ keri/codes/$

[7] $https://cp.zgzcw.com/help/rules/a/content_1.shtml$

[8] Al-Sultan K S , Nizami M F H S .A Genetic Algorithm for the Set Covering Problem[J].Journal of the Operational Research Society, 1996, 47(5):702-709.DOI:10.2307/3010021.

[9] Bertolazzi P , Sassano A .An O(m n) algorithm for regular set-covering problems.[J].Theoretical Computer Science, 1987, 54(2-3):237-247.DOI:10.1016/0304-3975(87)90131-9.

[10] Stern M S D H .A Hierarchical Objective Set Covering Model for Emergency Medical Service Vehicle Deployment[J].Transportation Science, 1981, 15(2):137-152.DOI:10.1287/trsc.15.2.137.

[11] Maneengam A , Udomsakdigool A .A Set Covering Model for a Green Ship Routing and Scheduling Problem with Berth Time-Window Constraints for Use in the Bulk Cargo Industry[J].Applied Sciences, 2021, 11(11):4840.DOI:10.3390/app11114840.

[12] 代超.基于集合覆盖与智能优化算法的WCDMA网络基站位置优化[D].华南理工大学[2024-09-15].DOI:CNKI:CDMD:2.1013.319169.

8 致谢

在此，我要感谢英才计划的导师，唐舜，帮助我进行无偿的选题思路以及论文的指导，对我进行了兴趣上的引导，进行书籍推荐，并帮助解答了我探索期间的部分问题，以及进行了论文的修稿建议。

其次，我要感谢刘金老师，在书写论文期间也为我提供了无偿的论文修改建议。

以及感谢英才计划本身，引导我首次接触了数学相关的科研，并激励我继续在将来继续进行数学方面的探索。

同时，感谢我的父母，在我的科研过程中对我的悉心照顾。

9 附录A: 欧氏距离FPP DFS代码

```
// Written By Xinye Zhang
#include <bits/stdc++.h>
using namespace std;
#define HUGEINT 31415926;
//The Point is represented by (Line, Column) or (L , C)
struct Point{
        int l;
        int c;
};
struct ResultRecord{
        int ReqCircNum;
        vector<Point> Placement;
};
//The vector Covering represents The Set consists of every Point that (L, C) can cover with a
    radius R.
vector<Point > Covering[40][40][40];
ResultRecord Res[43][43];
int HasCovered[40][40]; //The Two−Dimension Array Represents Whether Point (L, C) has been
     covered by at least one circle.
int q , R;
Point CurPlace[49];

bool CanCover(Point a, Point b, int radi){ //Judge Whether the circle with center a and radius
    r can cover b
        if((b.l − a.l) ∗ (b.l − a.l) + (b.c − a.c) ∗ (b.c − a.c) <= radi ∗ radi){
                return true;
        }
        return false;
}

void DFS(int q, int R, int Counter, int CircNum){
        //Update when Counter Reach Zero
        if(Counter <= 0){
                if(CircNum < Res[q][R].ReqCircNum){
                        ResultRecord A;
                        A.ReqCircNum = CircNum;
                        for(int i = 1; i <= CircNum; i++){
                                A.Placement.push_back(CurPlace[i]);
                        }
                        Res[q][R] = A;
```

```
                cout << CircNum << endl;
                    for(int k = 0; k < Res[q][R].ReqCircNum; k ++){
                    cout << "(" << Res[q][R].Placement[k].l << "," << Res[q][R].
                        Placement[k].c << "),-";
                }
                cout << endl;
            }

            return;
        }
        //Dfs algorithm
        int NewCoverNum = 0;
        for(int i = 0; i < q; i ++){
            for(int j = 0; j < q; j ++){
                NewCoverNum = 0;
                if(HasCovered[i][j] == 0){
                    Point x;
                    x.l = i; x.c = j;
                    CurPlace[CircNum + 1] = x;
                    for(int k = 0; k < Covering[i][j][R].size(); k ++){
                        if(Covering[i][j][R][k].l <= q - 1 && Covering[i][j][R][k].
                            c <= q - 1){
                            if(HasCovered[Covering[i][j][R][k].l][Covering[i][j
                                ][R][k].c] == 0){
                                NewCoverNum ++;
                            }
                            HasCovered[Covering[i][j][R][k].l][Covering[i][j][R
                                ][k].c] ++;
                        }
                    }
                    Counter -= NewCoverNum;
                    DFS(q, R, Counter , CircNum + 1);
                    //trace back
                    Counter += NewCoverNum;
                    for(int k = 0; k < Covering[i][j][R].size(); k ++){
                        if(Covering[i][j][R][k].l <= q - 1 && Covering[i][j][R][k].
                            c <= q - 1){
                            HasCovered[Covering[i][j][R][k].l][Covering[i][j][R
                                ][k].c] --;
                        }
                    }
                }
```

```
                }
        }
        return;
}

int main(){
        for(int L = 0; L < 8; L ++){
                for(int C = 0; C < 8; C ++){
                        for(int r = 1; r <= 8; r ++){
                                Point a;
                                a.l = L; a.c = C;
                                for(int i = 0; i < 8; i ++){
                                        for(int j = 0; j < 8; j ++){
                                                Point TestP;
                                                TestP.l = i; TestP.c = j;
                                                if(CanCover(a, TestP, r)){
                                                        Covering[L][C][r].push_back(TestP);
                                                }
                                        }
                                }









                        }
                }
        }
        //The Code Above have already finish preprocessing of each point.

        //The Code Below is the DFS Algorithm.
        for(int i = 8; i <= 8; i ++){
                for(int j = 1; j <= 8; j ++){
                        memset(HasCovered, 0, sizeof(HasCovered));
                        memset(CurPlace, 0, sizeof(CurPlace));
                        Res[i][j].ReqCircNum = HUGEINT;
                        q = i; R = j;
                        DFS(q , R , i * i , 0);
                        /*
                        Four Parameters Each Represents:
                        q: the Size Of Grid
                        R: the Radius of Each Circle
                        Counter: Remaining Grid that has not been covered by any Circle
```

```
                    0 (CircNum): How much Circle we have used so far
                    */

                    cout << "q = " << q << ", R = " << R << ", Num = " << Res[i][j].
                        ReqCircNum << endl;
                    for(int k = 0; k < Res[i][j].Placement.size(); k ++){
                            cout << '(' << Res[i][j].Placement[k].l << ',' << Res[i][j].
                                Placement[k].c << "), ";
                    }
                    cout << endl;
                }
        }

        return 0;
}
```

10 附录B: 欧氏距离FPP构造答案

q = 1							
R = 1	R = 2	R = 3	R = 4	R = 5	R = 6	R = 7	R = 8
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,1)	(0,1)	(0,1)
q = 2							
R = 1	R = 2	R = 3	R = 4	R = 5	R = 6	R = 7	R = 8
(0,0), (1,1)	(0,0)	(0,0)	(0,0)	(0,0)	(0,1)	(0,1)	(0,1)
q = 3							
R = 1	R = 2	R = 3	R = 4	R = 5	R = 6	R = 7	R = 8
(0,0), (0,2), (2,1)	(1,1)	(0,0)	(0,0)	(0,0)	(0,1)	(0,1)	(0,1)
q = 4							
R = 1	R = 2	R = 3	R = 4	R = 5	R = 6	R = 7	R = 8
(0,1), (1,3) (2,0), (3,2)	(0,0), (0,3), (3,1)	(1,1)	(0,1)	(0,0)	(0,1)	(0,1)	(0,1)
q = 5							
R = 1	R = 2	R = 3	R = 4	R = 5	R = 6	R = 7	R = 8
(0,1), (0,3), (2,0), (2,2), (2,4), (4,1), (4,3)	(0,0), (0,3), (2,4), (4,1)	(2,2)	(1,2)	(0,1)	(0,1)	(0,1)	(0,1)
q = 6							
R = 1	R = 2	R = 3	R = 4	R = 5	R = 6	R = 7	R = 8
(0,0), (0,2), (0,4), (1,1), (1,3), (1,5), (2,2), (3,0), (3,4), (4,3), (5,1), (5,5)	(0,0), (0,4), (3,2), (3,6), (5,0), (6,4)	(0,0), (1,5), (5,2)	(0,0), (3,3)	(1,2)	(0,2)	(0,1)	(0,1)
q = 7							
R = 1	R = 2	R = 3	R = 4	R = 5	R = 6	R = 7	R = 8
(0,0), (0,2), (0,4), (0,6), (1,1), (1,3), (1,5), (2,0), (2,2), (2,4), (2,6), (3,1), (3,5), (4,3), (5,0), (5,6), (6,2), (6,4)	(0,0), (0,4), (3,2), (3,6), (5,0), (6,4)	(0,0), (0,4), (4,0), (4,4)	(0,3), (5,3)	(2,3)	(1,3)	(0,3)	(0,1)
q = 8							
R = 1	R = 2	R = 3	R = 4	R = 5	R = 6	R = 7	R = 8
(0,0), (0,2), (0,4), (0,6), (1,1), (1,3), (1,5), (1,7), (2,0), (2,2), (2,4), (2,6), (3,1), (3,3), (3,5), (3,7), (4,0), (4,2), (4,4), (5,1), (5,6), (6,0), (6,3), (6,7), (7,1), (7,5)	(0,0), (0,3), (0,6), (2,1), (2,4) (4,7), (5,2), (6,0), (7,5)	(0,0), (0,5), (5,0), (5,5)	(0,0), (2,7), (7,3)	(0,0), (4,4)	(3,3)	(2,3)	(1,2)

图 8: q , R均属于[0, 8]时欧氏距离FPP的构造答案