分枝定界法寻找最优差分特征

by wfz

我仔细阅读了论文，对整个算法有了大致的了解，并将其用 c 语言实现了出来。总的来说，这个算法并不难理解，唯一觉得有点烦人的地方就是引入了太多乱七八糟的符号。为便于理解这个论文，可以定义下列符号：

R：算法轮数；

N：s 盒个数；

L_1：一个 s 盒输入范围为 $0 \sim L_1 - 1$；

L_2：一个 s 盒输出范围为 $0 \sim L_2 - 1$；

PNmaxsb[N]：任意 1~N个 s 盒的最优概率，首先需要明确的是每个 s 盒都有一个最优差分转移概率，当然要求 s 盒输入非 0 时。我们将 N 个 s 盒的最优概率按照从大到小进行排序，不妨设为 $p_1, p_2, ..., p_N$, 那么令

$$PNmaxsb[1] = p_1, PNmaxsb[2] = p_1p_2, ..., PNmaxsb[N] = p_1p_2...p_N$$

这意味着任取 n 个激活的 s 盒，剩下的 N-n 个非激活的 s 盒，它们对应的所有差分转移概率上界是 PNmaxsb[n], 显然 $PNmaxsb[1], PNmaxsb[2], ..., PNmaxsb[N]$ 也是按照从大到小。

allsbdiffP[N][L]：所有 s 盒的差分概率，$L = L_1 * L_2$，你可以理解为 N 个 2 维矩阵，如果选取第 n 个 2 维矩阵的第 i 行第 j 列元素，它的值就表示第 n 个 s 盒输入差分为 i 且输出差分为 j 时所对应的差分转移概率；

Alpha[R][N]：每轮的输入差分；

Beta[R][N]：每轮的输出差分；

actIndex[R][N]：每轮活跃 s 盒的下标

actIndexnumber[R]：每轮活跃 s 盒的个数

DiffP[R][N]：每轮的差分概率；这可以通过上面的五个变量来计算得到。

maxAlpha[R][N]：每轮的最优输入差分；

maxBeta[R][N]：每轮的最优输出差分；

maxbacktable[N][L2]：你可以理解为 N 个 1 维数组，如果选取第 n 个数组的第 i 个元素，不妨设它的值为 x, 则第 n 个 s 盒固定输出差分为 i 时, 其输入差分取 x 时对应差分转移概率最大；当然概率最大时对应的 x 可能不唯一，但这个数组里面只选择了其中一个, 这不会对计算最优差分特征产生影响；

maxforwardtable[N][L1]：类似上面的 maxbacktable[N][L2]，这里是固定输入差分，所对应的使差分转移概率最大的输出差分；

decreasort[L1][L2]：这是一个 s 盒的差分差分转移概率表，可以理解为二维矩阵，选择第 i 行，当列数增加时，对应的值递减。通过这个表，给定一个输入差分，可以得到使其差分转移递减的输出差分序列。事实上，这应该有 N 个表，如果有 N 个不同 s 盒的话；

Pbest[R]：表示当算法轮数取 1~R 时所检索到的最优差分转移概率；

Pestim：阈值，作为分支定界的截断标准。

写到这里发现我定义的符号也挺多的，不过这对于编程实现来说是很自然的。我认为理解一个算法最好的方法就是结合具体例子，下面将以 PRESENT 密码为例，介绍如何使用分枝定界法来检索最优差分特征。

对于 PRESENT 密码，$N = L_1 = L_2 = 16$。当 R=1 时，显然最优差分转移概率 $Pbest[1] = PNmaxsb[1]$，其中 PNmaxsb[1] 可以通过 allsbdiffP[N][L] 查表得到。当 R=2 时，我们很自然的想到穷举法，显然时间复杂度为 2^192，但我们如果换种思路，首先穷举第一轮输出差分即 Beta[1][N]，对每一个 Beta[1][N]，都可以通过查表 maxbacktable[N][L2] 得到一个使第一轮差分转移概率最大的 Alpha[1][N]，通过线性变换可以得到确定第二轮输入差分 Beta[2][N]，对每一个 Beta[2][N] 又可以通过查表 maxforwardtable[N][L1] 得到使第二轮差分转移概率最大的输出差分 Alpha[2][N]，这样就可以得到两轮的最优差分概率，时间复杂度为 2^64, 显然效率大大提升，当然空间复杂度有所增加，但对于 N 个相同的 s 盒来说是无所谓的。你是否觉得这个复杂度已经很低了，无法再次降低？答案是否定的。

我们现在改变对第一轮输出差分 Beta[1][N] 的穷举或者说遍历的方式，首先激活第一个 s 盒，其它 s 盒不激活即输入输出差分为 0，然后对第一个 s 盒进行遍历 $(L_2 - 1)$ 次，然后前文所描述的得到对应的 2 轮差分转移概率，显然这个差分转移概率的上界是 $PNmaxsb[1]^2$, 同理, 还可以激活第 i 个 s 盒, 其它不激活, 遍历得到差分转移概率, 上界依然是 $PNmaxsb[1]^2$。这样如果只激活一个 s 盒，所需要的时间复杂度为 $C_N^1(L_2 - 1)$。我们将上面遍历 1 个 s 盒求得的所有 2 轮差分概率中的最大值记为 Pestim。下面我们激活第一轮输出差分的两个 s 盒, 由前文可以知道第一轮差分概率最大值为 $PNmaxsb[2]$, 那么 2 轮差分转移概率的上界就是 $PNmaxsb[2]Pbest[1] = PNmaxsb[2]PNmaxsb[1]$, 如果 $PNmaxsb[2]Pbest[1] \leq Pestim$，那么就不可能找到激活 2 个 s 盒的第一轮输出差分使得检索的 2 轮差分转移概率大于 Pestim，又 PNmaxsb[N] 是单调递减的，所以激活更多 s 盒的差分概率更不可能超过 Pestim。如果 $PNmaxsb[2]Pbest[1] > Pestim$，则

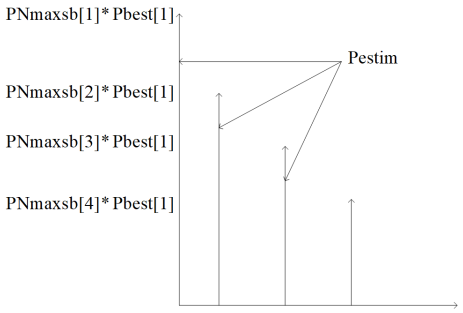

图 1: 激活 n 个 s 盒所对应 2 轮差分概率上界示意图

有可能找到激活 2 个 s 盒的第一轮输出差分使得检索的 2 轮差分转移概率大于 Pestim，此时我们对激活 2 个 s 盒的第一轮输出差分进行遍历，时间复杂度为 $C_N^2(L_2-1)^2$，我们将上面遍历 2 个 s 盒求得的所有 2 轮差分概率中的最大值 x 与 Pestim 进行比较，如果 $x > Pestim$，则令 $Pestim = x$。然后开始激活 3 个 s 盒，类似的，2 轮差分转移概率的上界就是 $PNmaxsb[3]Pbest[1] = PNmaxsb[3]PNmaxsb[1]$，如果 $PNmaxsb[3]Pbest[1] \leq Pestim$，那么就不可能找到激活 3 个 s 盒的第一轮输出差分使得检索的 2 轮差分转移概率大于 Pestim，又 PNmaxsb[N] 是单调递减的，所以激活更多 s 盒的差分概率更不可能超过 Pestim。上述过程结合图1可以更直观理解。显然，如果当激活 n+1 个 s 盒时，发现 $PNmaxsb[n+1]Pbest[1] \leq Pestim$，就可以判断找到了最优的差分转移概率即 Pestim。怎么得到对应的差分特征呢，只需要在更新 Pestime 时，把对应的差分特征 Alpha[R][N] 和 Beta[R][N] 复制到 maxAlpha[R][N] 和 maxBeta[R][N] 中。容易知道，总的时间复杂度为 $C_N^1(L_2-1)+C_N^2(L_2-1)^2+...+C_N^n(L_2-1)^n, 1 \leq n \leq N$，显然复杂度在 $[N(L_2-1), L_2^N-1]$ 这个闭区间内，对于 PRESENT 密码就是 $[240, 2^{64}-1]$，也就是说最好的情况下只需要 240 次遍历就可以得到最优差分概率，事实上，对于一些密码来说，这种最好的情况不难遇到。

上面主要说了 2 轮最优差分概率检索的方法，下面我们讨论当 R=3 时会如何进行检索。首先我们先利用上面的方法得到 2 轮最优概率 $Pbest[2]$，然后再开始检索三轮。具体过程如下：（1）遍历第一轮 s 盒输出差分 Beta[1][N]，按照 s 盒激活数量从 1 到 N 的方式；（2）对于每一个 Beta[1][N]，都可以通过线性变换确定 Alpha[2][N]，然后计算出 actIndex[R][N] 和 actIndexnumber[R]，就知道了有 actIndexnumber[2] 个 s 盒在第二轮被激活，然后开始遍历第二轮输出差分即 Beta[2][N]，也是按照 s 盒激活数量增加的方式，当 Beta[2][N] 激活了 n 个 s 盒时，我们可以对其进行估计，此时还有 $actIndexnumber[2]-n$ 个 s 盒没有被激活，可以计算出当前的第二轮差分概率 DiffP[2][N]，如果我们继续激活 s 盒的话第二轮差分概率可以达到多大呢？显然是

$DiffP[2][N] * PNmaxsb[actIndexnumber[2] - n]$，如果我们再深入考虑，第三轮的差分概率最大值为 $Pbest[1]$, 而第一轮的差分概率 DiffP[1][N] 已经确定了，因此当前状态下如果继续激活 s 盒的话，三轮的差分概率的上界为 $DiffP[1][N] * DiffP[2][N] * PNmaxsb[actIndexnumber[2] - n] * Pbest[1]$，到了这里就很自然的想到和 Pestime 进行比较（这里的 Pestim 是前面激活更少 s 盒的三轮差分概率中最大值），如果 $DiffP[1][N] * DiffP[2][N] * PNmaxsb[actIndexnumber[2] - n] * Pbest[1] \leq Pestim$, 那么意味着激活更多 s 盒是没有意义的，无法找到更大的三轮差分概率。有这个时候有一个遍历技巧，对于每个固定 s 盒输入差分即 Alpha[2][i], 按照使差分概率递减的方式取遍历也就是查表 decreasort[L1][L2]，这样做的好处是可以使每次遍历时得到的上界都比上一次遍历时要小，也就是可以更快的找到这个 break 的时刻。特别的，当线性置换是比特置换时，可以将上界进一步降低，具体来说是，当激活 n 个 s 盒时，就可以通过比特置换的方式得到第三轮最少激活的 s 盒数量，不妨设为 m，则第三轮的差分概率最大值为 $PNmaxsb[m] \leq Pbest[1]$，所以三轮差分上界可以降低为 $DiffP[1][N] * DiffP[2][N] * PNmaxsb[actIndexnumber[2] - n] * PNmaxsb[m]$。

同理，当 $R > 3$ 时，可以在已经得到 $Pbest[1], Pbest[2], ..., Pbest[R-1]$ 的前提下利用 3 轮的上界条件进行判断，这一点结合具体编程实践或许更容易理解。

综上所述，可以看出，利用差分概率的上界来找到 break 的方法就是使用分枝定界法来检索最优差分特征的核心思想。对于不同的密码算法，所对应的上界应该具体分析，不能一概而论。