自然界中，青蛙群体会分散到不同的地方寻找食物，这些分散的小群体可以被看作是“记忆复合体”。在SFLA中，这个概念被用来描述算法中的一个子群体，用于模拟青蛙在解空间内的局部搜索行为**。**这个SFLA\_SIGMOID函数是使用随机蛙跳算法（SFLA）来优化SVM分类器的sigmoid核参数的函数。整个函数的逻辑过程如下：

#### 搜索参数简介：

初始化参数（Step 0）：

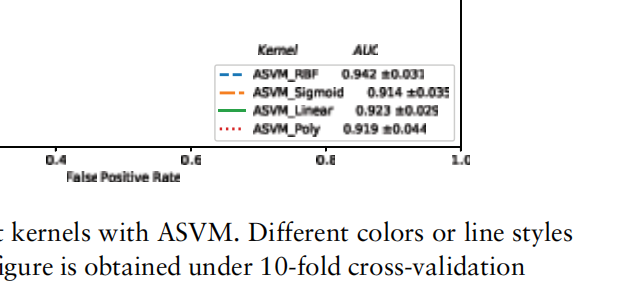
sizeC、sizeGamma和sizeCoef0是控制参数搜索步长的变量。

* -C是SVM中的一个惩罚参数，它控制了模型对误分类训练样本的惩罚程度。较小的C值会导致较小的误差惩罚，模型可能会更倾向于一个更大的决策边界，这可能会增加训练数据的误分类。
* 径向基函数（Radial Basis Function, RBF）核的参数γ的取值范围。γ控制了单个训练样本影响的范围，从而影响了决策边界的平滑程度。较小的γ值意味着更远的距离会被考虑在内，从而得到更平滑的决策边界。
* SVM中用于sigmoid核和多项式核的参数coef0的取值范围。对于sigmoid核，它控制了核函数的独立项，可以调整模型的非线性特性。对于多项式核，coef0控制了高阶项与低阶项的影响。

max\_step是基于参数范围和步长大小计算出的最大步长。（类似于学习率，更新参数的步长）

#### 原始ASVM逻辑简介

以sigmoid核为例（注意sigmoid核是原文指出的性能最差的核）



### 生成初始种群（Step 1）：

frogC、frogGamma和frogCoef0是分别针对C、gamma和coef0参数随机生成的初始青蛙种群。这里的青蛙代表不同的参数组合，举例来说就是一个三元元组就代表一个青蛙(1,0.001,0.5)。计算每只青蛙在验证数据上的性能：使用交叉验证的方式（KFold），将训练数据分为多个子集，每次留一部分作为验证集。对于每一只青蛙，将其作为SVM（svc）的参数，在训练集上进行训练，同时函数计算在当前验证集上的ROC AUC分数。

### 对青蛙进行排序（Step 2）：

根据ROC AUC分数对所有青蛙（参数组合）进行排序后进行全局搜索，记录迭代次数和搜索停止条件。

### 划分青蛙到记忆复合体（Step 3）：

在SFLA中，所有的青蛙（即候选解）首先根据它们的SVM性能被排序。然后，这些青蛙被分配到不同的记忆复合体中。每个记忆复合体包含了若干数量的青蛙（候选解），例如n只。每个记忆复合体内的青蛙将会进行局部搜索，以期望找到更好的解。这个分配过程是一个随机过程。

### 在每个记忆复合体中进行局部搜索（Step 4）：

局部搜索是在每个记忆复合体内进行的。在每个记忆复合体中，选取一部分青蛙形成一个子记忆复合体。这个过程涉及到对青蛙的加权随机选择，其中权重由之前的性能排名来决定。换句话说，性能好的青蛙被选中的概率会更高。

在子记忆复合体中，我们会找到性能最好的青蛙（Pb）和性能最差的青蛙（Pw）。然后，我们尝试通过结合Pb的信息来改进Pw的位置，这个过程类似于Pw从Pb那里“学习”或者说在参数上模拟Pb之后，Pw作为SVC模型的参数在验证集上的性能有所提升。Pw的新位置（Uq）通过结合Pb的特性来计算得出，换句话说，就是尝试让Pw的参数向Pb的参数靠拢，从而希望Pw能够在模型评价指标上（如ROC AUC分数）取得更好的表现。如果更新后的位置Uq在参数的可行空间内，并且性能有所提升，那么我们就接受这个新位置。并替换掉原来的Pw。如果Uq的性能没有提升，或者它不在参数的可行空间内（超出了预先定义的参数范围），那么都需要采取另外一个策略，本文将从全局最佳青蛙（frog\_gb）随机选取一个青蛙作为学习对象，同时设定一个最大步长，用于控制学习过程可以对最坏青蛙最大改变量。如果从全局青蛙处更新的位置Uq’的性能仍然没有提升或者它不在参数的可行空间内，那么就完全随机生成一组新的参数（青蛙），作为最坏青蛙的新状态，以此来尝试改进Pw的性能。

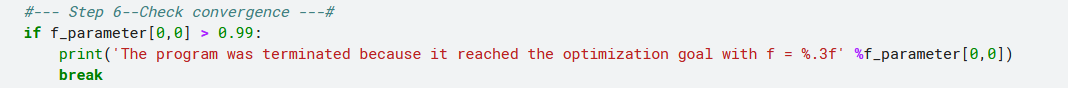
### 洗牌记忆复合体（Step 5）：

一旦所有的记忆复合体都完成了局部搜索，我们就会对所有青蛙进行混合洗牌，生成和之间的不同组合作为新的记忆复合体中。这个过程相当于是为了跳出局部最优解，通过“洗牌”来重新组合不同的青蛙，增加了算法的多样性，有助于在全局解空间中探索更多可能的解。

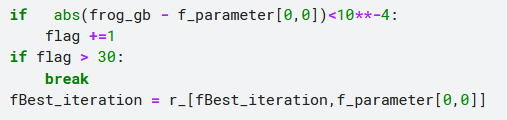
### 检查收敛性（Step 6）：

在每次全局迭代之后，算法会检查是否达到了收敛条件。

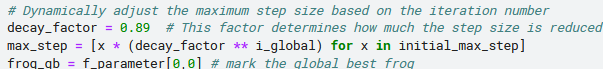
收敛条件性能目标达成：如果在任何全局迭代步骤中，最好的青蛙（即最佳参数组合）的性能指标f超过了0.99，就认为已经达到了优化目标，程序将终止。这里的性能指标f在实现中使用的是ROC-AUC分数。代码中的这一行：

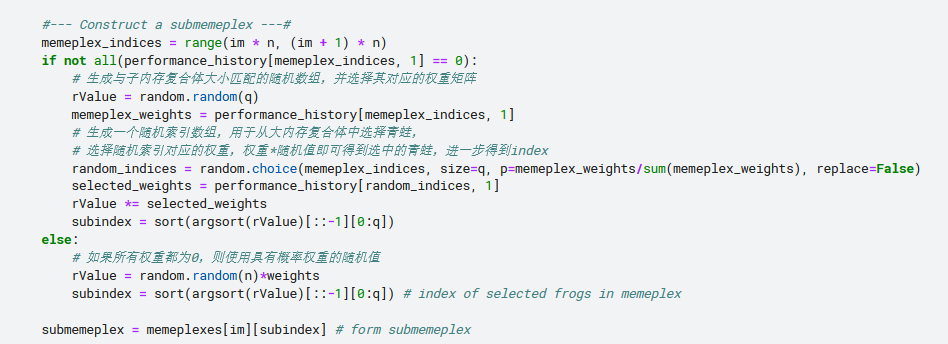


性能改进停滞：如果在连续多次全局迭代中，最佳青蛙的性能没有显著改善（这里是指改善小于10^-4），则会增加flag计数器的值。如果flag超过30，也就是说在超过30次迭代中没有显著的性能提升，程序会认为已经收敛，并终止。



#### ASVM改进思路：

1. 并行计算：计算K-Fold的每一折使用joblib库进行并行，每一折都放在一个单独的CPU上运行，优化的改进时间取决于硬件配置，实测运行是从3377---->1800秒。参考结果示意图
2. 动态调整步长：当局部青蛙搜索失败时，我们将先在全局最好的青蛙中随机抽取的一个作为新的学习对象，但是这样会使得学习量（最坏青蛙的参数改变量）变的不确定，原始方法中将步长根据参数范围和固定的大小sizeC, sizeGamma, sizeCoef0计算的。本文进一步将步长随着迭代次数动态调整，随着算法的进行减小步长，以便在搜索空间内更精细地搜索。本文引入了一个衰减因子decay\_factor，并且在每次全局迭代中更新max\_step。随着迭代次数的增加，步长会逐渐减小，使得算法在后期能够更精细地探索参数空间。这样的调整有助于在全局搜索和局部搜索之间取得更好的平衡，可能会提高算法的收敛速度和找到的解的质量。
3. 自适应权重调整：在选择子记忆复合体时，权重是根据青蛙的排名预先计算好的。我们可以让权重基于青蛙的历史性能来动态调整，这样可以增加那些性能逐渐提升的青蛙被选中的概率。本文在选择子记忆复合体时，不再使用静态的权重，而是根据青蛙在过去的性能变化来调整权重。本文跟踪每只青蛙过去性能的变化，并根据这个变化来调整其被选中的概率。



实现逻辑：

1. 首先，计算当前记忆复合体中所有青蛙的索引范围memeplex\_indices。
2. 如果performance\_history中对应的权重不全为0，则进行子记忆复合体的选择。
3. 使用random.choice根据权重memeplex\_weights随机选择q个青蛙，这里size=q表示要选择的青蛙数量，p=memeplex\_weights/sum(memeplex\_weights)表示按权重概率选择。（memeplex\_indices 列表中随机选择 q 个元素,选择概率与 memeplex\_weights 列表中的值成正比。）
4. random\_indices是根据权重选择的青蛙的索引数组。
5. selected\_weights是这些青蛙对应的权重。
6. 然后，根据rValue和selected\_weights的乘积来确定最终选择的青蛙。
7. subindex是根据权重和随机值选择的青蛙的索引。

总结：本文应用权重来改进了子记忆复合体的选择，使得性能更好的青蛙有更高的概率被选中。



如果SVM分类器性能没有改善，就会降低这个青蛙再次被选择为初始迭代状态的概率，通过将最小权重设置为0.1，我们确保每只青蛙都有机会被选中，但是那些在迭代中具有更好潜在性能提升的具有更高的概率。（潜在性能提升：如果这只青蛙在迭代中被优化过，就说明他具有进一步改进的可能性）

1. 改进青蛙生成策略：当在局部迭代过程中，原ASVM是通过让最差青蛙学习最好青蛙的特征，尝试对最差青蛙进行改进的学习策略改进最差青蛙，同通过不断的局部迭代也可以改进子记忆体整体（平均）性能。

（原始局部迭代过程描述）

改进一个子记忆复合体中最差青蛙的搜索策略的本质逻辑是，首先从当前子记忆复合体的局部最优青蛙入手，让最差青蛙加上它们之间的差值，得到最差青蛙1，判定最差青蛙1是否仍然属于这个子记忆复合体，如果属于，进一步判断最差青蛙1在验证集上的性能是否提升，如果性能没有提升，那么就会在这个子记忆复合体范围内随机生成一个合法青蛙作为最差青蛙的更新，如果性能提升了，那么最差青蛙1将作为本次的最差青蛙的更新。如果最差青蛙1不再属于这个子记忆复合体了，那么将会在全局最优青蛙中随机选一个让最差青蛙加上它们之间的差值重复上述过程，如果最终性能没有提升，那么就会在这个子记忆复合体范围内随机生成一个合法青蛙作为最差青蛙的更新。

改进目的：为了提升搜索策略的效率，必须减少随机搜索行为及减少陷入局部最佳解的风险。目标是增强每个搜索步骤的针对性，确保搜索过程符合特定的导向性。特别是在遇到当前策略失效时，应避免在可能的解空间内随机青蛙，以减少无效操作和重复波动的发生。这要求搜索算法具备更高的智能化，能够在遇到挑战时采取更系统性和有目的性的搜索路径。

在上面的过程中，如果青蛙的性能没有提升时，本文设计了更智能的青蛙生成策略。

首先介绍**适应性高斯插值方法：**（adaptative\_gaussian\_perturbation）描述了一个适应性高斯扰动过程，用于进化计算或优化算法中，特别是在模拟生物进化的算法（如粒子群优化或差分进化等）中用于更新个体位置的一个步骤。这个过程的作用是在搜索全局最优解的过程中，动态调整解的搜索范围和方向。具体来说，该过程包含以下几个步骤：

1. 计算适应性因子 (alpha): 这个因子随着迭代次数的增加而减小，意味着随着优化过程的进行，解的探索随机性会逐渐减小。这里使用指数衰减函数来实现这一点，其中 iteration 代表当前迭代次数，max\_iterations 代表最大迭代次数。当 iteration 接近 max\_iterations 时，alpha 将接近于。
2. 生成高斯扰动因子 (gauss\_factor): 这里使用正态分布（高斯分布）来生成一个随机扰动因子，这个因子有一个均值为1和一个给定的标准差（这里是0.3）。使用高斯分布而不是均匀分布可以使得生成的随机数更加集中于均值附近，这样可以在探索过程中提供更细致的调整。
3. 假设传入的参数是Pw[1:]，那么最终结果为Uq = Pw[1:] + alpha \* gauss\_factor \* (frog\_gb - Pw[1:])这样，新位置 Uq 不仅依赖于当前位置和全局最佳位置，而且还考虑了迭代过程和随机性的影响。

**退火算法的变体：**

模拟退火算法（变体）描述：

我们在引入高斯插值之后，如果得到的新青蛙性能表现仍比原来差，那么本文进一步对高斯插值的结果进行升温处理，基于性能差距的绝对值大小，作为接受新青蛙的概率，需要进一步在每次迭代中引入一个温度参数T，该参数随迭代逐渐升高。在模拟退火（变体）的每一步中，仍然是先从当前子记忆复合体的局部最优青蛙入手，让最差青蛙加上它们之间加入高斯插值后的差值，得到最差青蛙1，判定最差青蛙1是否仍然属于这个子记忆复合体，如果属于，进一步判断最差青蛙1在验证集上的性能是否提升，如果性能没有提升即fq < Pw[0]，则根据fq以及当前温度来决定是否接受这个更新后最差青蛙1，如果不接受那么就会在这个子记忆复合体范围内随机生成一个合法青蛙作为最差青蛙的更新，如果性能提升了，那么最差青蛙1将作为本次的最差青蛙的更新。如果最差青蛙1不再属于这个子记忆复合体了，那么将会在全局最优青蛙中随机选一个让最差青蛙加上它们之间的差值重复上述过程，如果最终性能没有提升即fq < Pw[0]，则根据fq以及当前温度来决定是否接受这个更新后最差青蛙1，如果不接受，那么就会在这个子记忆复合体范围内随机生成一个合法青蛙作为最差青蛙的更新。

进一步解释：在引入高斯插值之后，每做一步高斯插值同时根据退火算法判定这个值被接受的概率，当然因为高斯插值越往后迭代，最坏青蛙的参数改变量会位于**最好青蛙和最坏青蛙差值**的一个变化区间（高斯插值的作用），所以为了让每个值都拥有平等的被接受率，所以我们需要让它们的每个都经过一遍退火算法，当然退火算法是升温算法，否则前面较低的温度导致较低的接受概率等于没有作用了。

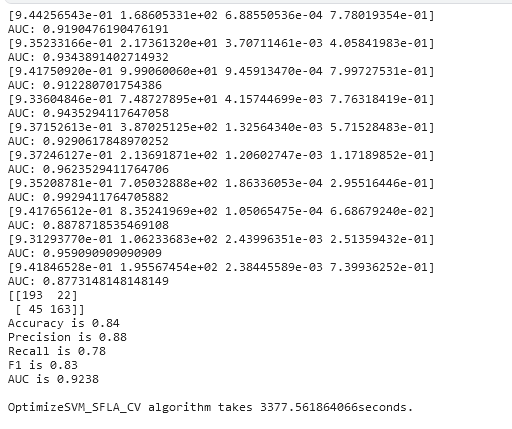
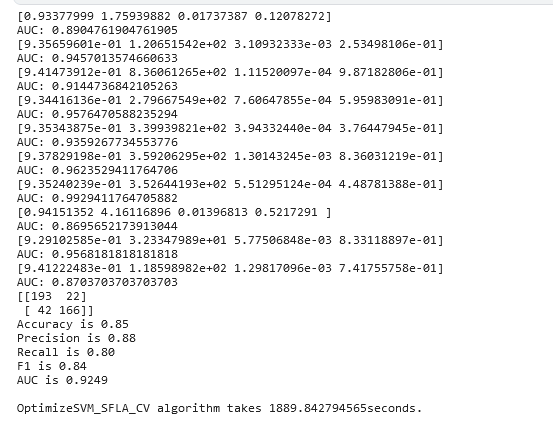
这里退火算法的变体是，我们是以差距绝对值衡量参数好坏而不是越往后迭代变化限制越紧。我们设置接受概率，如果学习后和最坏青蛙的原始值相差为0.0002（0.02%），那么逆退火算法的最大接受概率在0.4左右，简而言之：退火算法变体的评价指标改成一个单增的函数，表示如果最坏青蛙向最好青蛙学习之后，如果性能下降了，但下降的越少的学习结果越容易被接受。至于在局部搜索中依据迭代次数限制搜索步长的机制已经在计算适应性因子 (alpha)实现。

综上所述，本文无论是在局部搜索还是全局搜索中，当一只最坏青蛙的性能没有提升时有优先调整学习比率进一步尝试是否能够从局部最好青蛙身上学习到的有用的信息，而不是像原文那样直接可行域内简单地随机生成。在遇到当前策略失效时，应避免在可能的解空间内随机青蛙，以减少无效操作和重复波动的发生。

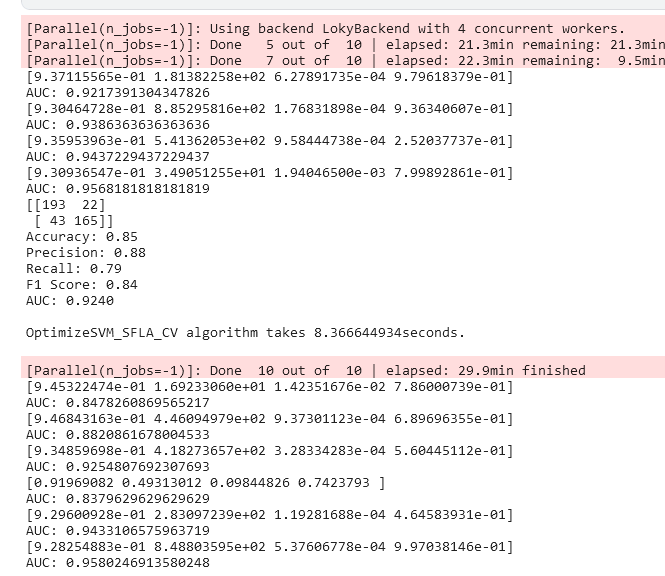
结果展示:

sigmoid核：

使用更窄参数搜索空间和更少的全局迭代和局部迭代次数也就是更快的搜索时间，取得了类似的平均AUC值（略高于原文，但是没有做p显著性检测，原因可能在于动态调整步长）

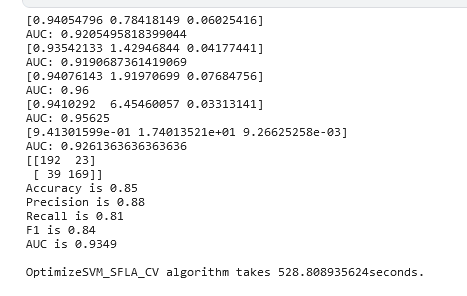


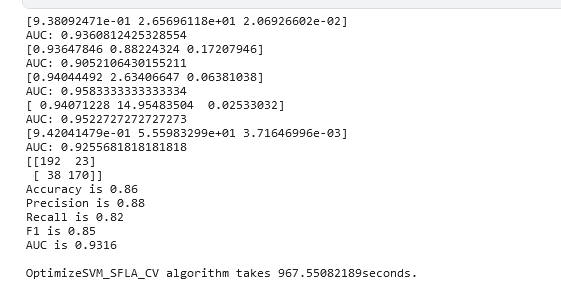
并行结果优化展示：



RBF核：

使用更窄参数搜索空间和更少的全局迭代和局部迭代次数也就是更快的搜索时间，取得了类似的平均AUC值（略低于原文，原因可能在于参数搜索范围限制），同时每一折交叉验证的结果的更加集和稳定，原文95% CI 落在 [0.885, 0.994]，本文在RBF核上实现了每一折的AUC结果均大于0.90，换句话说就是本文结果上的方差相对原文更小，生成的结果更稳定！





##### 附录1：

### 模拟退火算法(Simulated Annealing, SA)的逻辑和原理：

模拟退火算法是一种概率型优化算法，其灵感来源于金属的退火过程，即加热后再慢慢冷却，使其达到低能量状态的过程。在优化问题中，可以将“能量”理解为目标函数的值，即我们希望找到能让目标函数值最低的解。

1. **初始化**：在算法开始时，设置一个较高的“温度”T。
2. **概率接受差解**：在每一步，算法尝试对当前解进行一个小的随机扰动，得到一个新解。如果这个新解比当前解更好，就接受它；如果新解更差，也有一定概率接受它，这个概率与温度T和解的差异有关。
3. **降温**：随着迭代的进行，温度T逐渐降低，接受差解的概率也随之减小，这意味着算法越来越倾向于接受好的解，直到最后几乎不再接受差解。

在这段代码中，SA算法被用于改进“最差青蛙”的位置。当一个新的位置（新解）没有改善性能时，算法会根据模拟退火的概率来决定是否接受这个新位置。如果接受，就更新最差青蛙的位置；如果不接受，温度T会降低，算法继续尝试找到新的位置。

##### 附录2：

本文在进行性能较差的新青蛙接受和拒绝的过程中，选择使用模拟退火算法的逆过程。这实际上是模拟退火算法中Metropolis准则的一个变体。通常情况下，模拟退火算法接受新状态的条件是基于减小系统能量（即寻找目标函数的最小值）的原则。然而，本文的搜索场景下是希望寻找目标函数的最大值，同时给予系统跳出局部最优解的可能性。这种情况下，模拟退火算法依然是可行的，只是其接受准则的表达式需要相应调整。根据您的要求，如果一个新状态比当前状态更好（对于最大化问题，则意味着目标函数值更大），则无条件接受这个新状态；如果新状态较差，则以一定的概率接受这个新状态，这个概率随着时间（或温度T）的降低而减小，以便于算法最终收敛。

### 修改后的Metropolis准则

对于您提出的新接受条件，一个可能的Metropolis准则表达式可按如下方式调整：

* 如果 ，则无条件接受新状态）。
* 如果 ，则按概率接受新状态，概率为 。注意，这里的T是当前温度。如果p大于某个随机生成的值，那么则接受的，这样就是表示p越大，那么被接受的概率就越高。此时p是关于)|的单减函数，所以差距越小，被选择的概率越高的，同时的我们需要进一步调整温度和温度变化系数。
* 我们尝试使用高斯扰动来替换原有的均匀分布随机数生成和一个简单的适应性因子。高斯扰动能够模拟更自然的扰动，而适应性因子可以帮助在搜索过程中动态调整探索的广度。这段代码中，我们使用了一个适应性因子alpha，它随着迭代次数的增加（iteration变量）而递减。这意味着随着时间的推移，青蛙会减少它的探索范围，聚焦于当前区域的搜索。np.random.normal函数生成了高斯分布的随机数，该分布以1为中心，标准差0.3。

##### 附录3：

### 高斯插值法(Adaptive Gaussian Perturbation)的逻辑和原理：

高斯插值法是一种基于正态分布（高斯分布）的随机扰动方法。其核心思想是在当前解的周围生成一个新的解，新解生成的方式是在当前解的基础上增加一个随机扰动，这个扰动符合高斯分布。

1. **生成扰动**：对每个参数生成一个高斯分布的随机数作为扰动。
2. **适应性调整**：扰动的标准差可以根据迭代的进度动态调整，例如，随着迭代次数的增加，减小扰动的标准差，使得搜索逐渐集中于当前最优解附近。
3. **更新解**：将扰动加到当前解上，得到一个新的解。

本文高斯插值法被用于当模拟退火算法未能找到更优解时，即fq < Pw[0]条件不成立时，通过高斯分布的随机扰动来生成新的位置（新解）。这样做的目的是在全局最佳解（frog\_gb）和最差解（Pw）之间进行随机搜索，希望能够找到一个更好的解。

综合来看，局部搜索的目标是通过局部搜索策略在每个memeplex内部进行优化，使用SA算法和高斯插值法来改进青蛙的位置，以期望得到更好的解。这种结合了全局搜索和局部搜索的策略，旨在平衡算法的探索和利用能力，从而有效地找到全局最优解或近似最优解。