LSH2

2023 年 9 月 17 日

摘要 主要讲述了最近邻问题（nearest neighbor problem）和近似最近邻问题（approximate nearest neighbors problem）的背景和研究现状。最近邻问题是指在给定一个点集 P 和一个查询点 q 的情况下，预处理 P，以便高效地回答查询，找到离查询点 q 最近的 P 中的点。特别是在 d 维欧几里德空间下的情况下，当前的解决方案远远不令人满意。近似最近邻问题是指找到一个点 p，使得 p 是查询 q 的 ϵ-近似最近邻点，即对于所有 $p' \in P$，有 $d(p,q) \leq (1+\epsilon)d(p',q)$。

接下来，作者介绍了两个算法结果来解决近似最近邻问题，并显著改进了已知的结果上界。第一个算法的预处理成本多项式依赖于 n 和 d，而查询时间则是次线性的（对于 $\epsilon > 1$）。第二个算法的查询时间是多项式依赖于 $\log n$ 和 d 的，而预处理成本则是略微指数级的 $\tilde{O}(n) \times O(1/\epsilon)^d$。此外，应用一个关于随机投影的经典几何引理（作者给出了一个更简单的证明），我们得到了第一个已知的预处理和查询时间都是多项式的算法，其多项式依赖于 d 和 $\log n$。不幸的是，对于较小的 ϵ，后者只是一个纯理论的结果，因为指数依赖于 $1/\epsilon$。实验结果表明，我们的第一个算法在真实数据集上相对于运行时间有数量级的改进。它的关键之处在于局部敏感哈希（locality-sensitive hashing）的概念，它可能具有独立的研究价值；在这里，我们给出了信息检索、模式识别、动态最近对和快速聚类算法的应用。

动机 这两段文字讲述了最近邻问题在各种应用中的重要性和动机。最近邻问题在多个应用领域中都非常重要，通常涉及相似性搜索。一些例子包括：数据压缩、数据库和数据挖掘、信息检索、图像和视频数据库、机器学习、模式识别、统计和数据分析等。通常，我们将感兴趣的对象（文档、图片等）的特征表示为 d 维空间中的点，并使用距离度量来衡量对象之间的（不）相似性。基本问题是为查询对象执行索引或相似性搜索。特征的数量（即维度）可以从几十到几千不等。例如，在多媒体应用中，如 IBM 的 QBIC（以图像内容查询），特征数量可能达到几百个。在文本文档的信息检索中，向量空间表示涉及

到几千维度，降维技术（如 LSI、主成分分析或 Karhunen-Loéve 变换等）可以将维度降低到几百维。

近来，越来越多的人开始对避免维度灾难感兴趣，通过采用近似最近邻搜索。由于应用中特征的选择和距离度量往往是主观的，只是试图在本质上美观的相似性概念上做出数学上精确的描述，因此坚持绝对的最近邻可能有些过度。事实上，对于合理的 ϵ 值（如一个小的常数），找到一个 ϵ-近似最近邻对于大多数实际目的来说可能已经足够了。不幸的是，即使放松了目标，维度灾难依然存在，尽管 Kleinberg 的最近结果有所改善。

先前工作 这些段落讲述了关于最近邻问题的先前工作和技术。Samet [58] 综述了一系列用于最近邻搜索的数据结构，包括 k-d 树、R 树和基于空间填充曲线的结构；[59] 中对最近的研究进行了调查。虽然在 2-3 维度中的表现良好，但在高维空间中，它们在最坏和典型情况下表现都很差（如 Arya, Mount 和 Narayan [3] 所述）。Dobkin 和 Lipton [22] 是第一个提供 $\Re^d$ 中最近邻算法的人，其查询时间为 $O(2^d \log n)$，预处理成本为 $O(n^{2^{d+1}})$。Clarkson [15] 将预处理成本降低到了 $O(n^{[d/2](1+\delta)})$，但查询时间增加到 $O(2^{O(d \log d)} \log n)$。后来的结果，如 Agarwal 和 Matoušek [1]，Matoušek [50] 和 Yao 和 Yao [64]，查询时间都是指数级的。Meiser [51] 得到了查询时间 $O(d^5 \log n)$，但预处理成本为 $O(n^{d+\delta})$。所谓的“视点”技术 [11,12,61,62] 是一种最近流行的启发式方法，但我们不知道其在高维欧几里德空间中的分析。总的来说，即使对于在 $\Re^d$ 中分布的点的平均情况分析，启发式算法的查询时间也是指数级的。

对于近似最近邻问题，Arya 和 Mount [2] 提出了一个查询时间为 $O(1/\epsilon)^d \log n$，预处理成本为 $O(1/c)^d n$ 的算法。Clarkson [16] 和 Chan [14] 将其对 ϵ 的依赖关系降低到了 $e^{-(d-1)/2}$。Arya, Mount, Netanyahu, Silverman 和 Wu [1] 获得了最优的 $O(n)$ 预处理成本，但查询时间增长为 $O(d^3)$。Bern [7] 和 Chan [14] 考虑了 ϵ 多项式依赖于 d 的情况，并成功避免了指数依赖。最近，Kleinberg [45] 提出了一个预处理成本为 $O(n \log d)^{2d}$，查询时间多项式依赖于 d，c 和 $\log n$ 的算法，以及一个预处理多项式依赖于 d，ϵ 和 n 的算法，但查询时间为 $O(n + d \log^2 n)$。后者改进了暴力算法的 $O(dn)$ 时间界限。

对于汉明空间 $\{0,1\}^d$，Dolev，Harari 和 Parnas [24] 和 Dolev，Harani，Linial，Nisan 和 Parnas [23] 提出了一个算法，用于检索离查询点 q 距离为 r 的所有点。不幸的是，对于任意 r，这些算法在查询时间或预处理上都是指数级的。Greene，Parnas 和 Yao [37] 提供了一种方案，对于随机选择的二进制数据，在时间 $O(dn^{r/d})$ 内检索到距离 q 为 r 的所有点，预处理成本为 $O(dn^{1+r/d})$。

最近，Kushilevitz，Ostrovsky 和 Rabin [16] 得到了与下面的命题 3 类似

的结果。

我们的主要结果是下面描述的 $\varepsilon - NNS$ 近似最近邻算法。命题 1 对于 $\epsilon > 1$，存在一个在 l_s 范数下的 R^2 中的 $\varepsilon - NNS$ 算法，预处理成本为 $O(n^{1+1/\varepsilon} + dn)$，查询时间为 $O(dn^{1/\varepsilon})$。命题 2 对于 $0 < \epsilon < 1$，存在一个在 l_p 范数下的 $\varepsilon - NNS$ 算法，预处理成本为 $\tilde{O}(\mathrm{n}) \times O(1/\varepsilon)^d$，查询时间为 $\tilde{O}(d)$。命题 3 对于任意 $c > 0$，存在一个在 x^d 中的最近邻算法，预处理成本为 $(nd)^{O(1)}$，查询时间为 $\tilde{O}(d)$。

我们通过将 $\varepsilon - NNS$ 简化为一个称为相等球体点位置问题的新问题来得到这些结果。这是通过一种名为环形树的新颖数据结构实现的，该数据结构在第 3 节中进行了描述。我们的技术可以看作是参数搜索的一种变体 [52]，因为它们允许我们将优化问题减小到其决策版本。主要区别在于，在回答查询时，我们只能要求得到属于预先指定集合的决策问题的解，因为解决决策问题（即点位置在相等球体中）需要在预处理过程中创建的数据结构。我们相信这种技术将在参数搜索有帮助的问题中找到进一步的应用。

在第 4 节中，我们给出了两种解决点位置问题的方法。一种基于类似于 Elins 分箱算法 [63] 的方法 - 我们将每个球体分解成有界数量的单元格，并将它们存储在字典中。这样可以实现 $\bar{O}(d)$ 的查询时间，而预处理成本在 d 上是指数级的，从而推导出命题 2。对于第二种解决方案，我们介绍了局部敏感哈希的技术。关键思想是使用哈希函数，使得相邻的对象发生碰撞的概率远高于远离的对象。我们证明，对于任何领域（不一定是度量空间），存在这样的函数，可以得到该领域的快速 $\varepsilon - NNS$ 算法，其预处理成本仅线性依赖于 d 和次线性依赖于 n（对于 $\varepsilon > 1$）。我们然后提供了两个这样的函数族 - 一个用于汉明空间，另一个用于 Broder 等人用于聚类网络文档的相似度度量中的一类对象。基于第一个函数族的算法用于通过以保持距离的方式将 $\Re^d$ 中的点嵌入到一个汉明立方体上，得到最近邻算法。基于相似度度量的算法被证明在信息检索和模式识别中有几个应用。我们还给出了局部敏感哈希到动态最近对问题和快速聚类算法的其他应用。所有基于此方法的算法都易于实现，并具有其他优点 -它们利用数据的稀疏性，并且在实践中的运行时间比理论分析预测的要低得多 [36]。我们期望这些结果将产生重要的实际影响。

减少维度引起的复杂性的一种优雅技术是将点投影到较低维度的随机子空间，例如通过将 P 投影到经过原点的小集合的随机线集。具体来说，我们可以使用 Frankl 和 Maehara [32] 的结果，该结果改进了 Johnson-Lindenstraass 引理 [41]，表明将 P 投影到由大约 $9\varepsilon^{-2} \ln n$ 条随机线定义的子空间，可以在高概率下将所有点之间的距离保持在相对误差 ϵ 内。将此结果应用于查询时间为 $O(1)^d$ 的算法，我们可以得到一个查询时间为 $n^{9\varepsilon^{-2}}$ 的算法。不幸的是，这只

有在较大的 ε 值时才会导致子线性的查询时间。在附录的部分 A 中，我们提供了随机投影结果的一个版本，其证明比 Frankl 和 Machara 的证明要简单得多。我们还考虑将随机投影方法推广到对 $p \neq 2$ 的 l_p 范数。利用随机投影和命题 2，我们得到了命题 3 中描述的算法。不幸的是，高的预处理成本（其指数随着 $1/\varepsilon$ 的增长）使得这个算法对于较小的 ε 不实际。

总结:

这些段落主要讲述了关于最近邻问题的先前研究和技术。之前的算法在高维空间中表现不佳，无论在最坏情况下还是典型情况下都如此。已有的算法要么具有指数级的查询时间，要么具有指数级的预处理成本。

作者介绍了一些改进的算法，其中一些具有多项式的查询时间或预处理成本。其中一种方法是使用局部敏感哈希，它可以在高维空间中高效地搜索近似最近邻。另一种方法是将问题简化为等球体上的点定位问题，并使用一种称为环形树的数据结构来解决该问题。

作者还讨论了随机投影的方法，通过将点投影到较低维度的随机子空间，可以降低问题的复杂度。

总的来说，这些研究结果对解决高维最近邻问题提供了一些新的思路和方法，有望在实践中产生重要的应用价值。

符号说明

这些部分讲述了符号和定义的一些说明。首先，l_p^d 表示 l_p 范数下的 $\Re^d$ 空间。对于任意点 $v \in \Re^d$，我们用 $\|\vec{v}\|_p$ 表示向量 $\vec{v}$ 的 l_p 范数；当 $p = 2$ 时，我们省略下标。此外，$H^d = \left(\{0,1\}^d, d_H\right)$ 表示维度为 d 的汉明度量空间。对于任意度量空间 $\mathcal{M} = (X, d)$，$P \subset X$ 和 $p \in X$，我们使用以下记法：$d(p, P) = \min_{q \in P} d(p, q)$，$\Delta(P) = \max_{p,q \in P} d(p, q)$ 表示 P 的直径。

定义 1 中给出了以 p 为中心、半径为 r 的球的定义，记作 $B(p, r) = \{q \in X \mid d(p, q) \leq r\}$。以 p 为中心、内半径为 r_1、外半径为 r_2 的环的定义为 $R(p, r_1, r_2) = B(p, r_2) - B(p, r_1) = \{q \in X \mid r_1 \leq d(p, q) \leq r_2\}$。

接下来，$V_p^d(r)$ 表示 l_p^d 中半径为 r 的球的体积。事实 1 说明了球体体积的计算公式，其中 $\Gamma(.)$ 表示 gamma 函数。具体公式为 $V_p^d(r) = \frac{(2\Gamma(1+1/p))^d}{\Gamma(1+n/p)} r^d$，而 $V_2^d(r) = \frac{2\pi^{d/2}}{d\Gamma(d/2)} r^d$。

这部分讲述了将 ϵ-NNS 简化为点位置在等球体中的问题。首先给出了等球体点位置问题的定义。对于给定的 n 个半径为 r 的球，问题是设计一种数据结构，对于任何查询点 q，能够判断是否存在一个球中心 c_i 使得 q 在距离为 r 的球 $D(c_i, r)$ 内，如果存在则返回 c_i，否则返回"No"。

然后引入了 c-等球体点位置问题（c-PLEB）的定义，相比之前的问题，c-PLEB 提供了更多的信息。对于给定的 n 个半径为 r 的球，c-PLEB 的目

标是：- 如果存在一个球中心 c_i 使得 q 在距离为 r 的球 $B(c_i, r)$ 内，则返回"YES" 和一个球心 c_f 使得 q 在距离为 $(1+c)r$ 的球 $D(c,(1+c)r)$ 内；- 如果 q 不在任何球 $B(c_i,(1+\epsilon)r)$ 内，则返回"No"；- 如果 q 离最近的 c_i 的距离在 r 和 $(1+\epsilon)r$ 之间，则返回"YES" 或"No"。

观察到 PLEB 和 c-PLEB 问题可以通过将近似最近邻问题（NNS 和 c-NNS）简化成它们来解决。这个简化的过程可以在预处理和查询的成本上只有很小的额外开销。这种简化依赖于一种称为环形树的数据结构。该结构利用了对于任何点集 P，我们可以找到一个环分离器或者一个覆盖集，其中的任一种构造都可以将 P 分解成更小的集合 $S_1,\ldots,S_t$，满足对于所有的 i，$|S_i| \le c|P|$，且 $\sum_i |S_i| \le b|P|$，其中 $c<1$，$b<1+1/\log^2 n$。这种分解具有一个性质，即在搜索 P 时可以快速将搜索限制在 S_t 中的一个集合上。

此外，还给出了一个从 ϵ-NNS 到 ϵ-PLEB 的简化，但这个简化的效果较弱。该简化将问题转化为生成一系列具有不同半径的球的序列，然后通过二分搜索找到最小的半径 l，使得存在一个 i 使得 q 在球 B_i^t 内，并将 p_i 作为近似最近邻返回。这种简化的附加开销为 $O(\log\log R)$ 的查询时间开销和 $O(\log R)$ 的空间开销。这种简化的优点是在实践中非常简单易用，但当 R 较大时空间开销是无法接受的。

首先给出了环分离器和聚类的定义。环分离器是指对于给定的 P，如果存在一个球 $B(p,r_1)$ 使得 P 中的点至少有 α_1 比例位于该球内，同时存在一个球 $B(p,r_2)$ 使得 P 中的点至少有 α_2 比例位于该球外，其中 $r_2/r_1=\beta$。

接着给出了聚类的定义，如果对于聚类 S 中的每个点 p，p 在以 $\gamma\Delta(S)$ 为半径的球 $B(p,\gamma\Delta(S))$ 中的点的比例不超过 δ，则 S 被称为 (γ,δ)-聚类。

然后给出了覆盖集的定义。序列 $A_1,\ldots,A_l$ 是 P 的一个 $a(b,c,d)$-覆盖集，如果存在一个半径 $r \ge d\Delta(A)$（其中 $A=\cup_i A_i$），使得 $S\subset A$，并且对于 $i=1,\ldots,l$，满足：- $|P\cap(\cup_{p\in A_i}B(p,r))| \le b|A_i|$，- $|A_i|\le c|P|$。

定理 1 说明，对于任何 P、$0<\alpha<1$ 和 $\beta>1$，必定满足以下两个性质之一：1. P 存在一个 (α,α,β)-环分离器，2. P 包含一个大小至少为 $(1-2\alpha)|P|$ 的 $a\left(\frac{1}{2\beta},\alpha\right)$-聚类。

定理 2 说明，如果 S 是 P 的一个 (γ,δ)-聚类，则对于任何 b，存在一个算法可以生成一系列的集合 $A_1,\ldots,A_k\subset P$，构成 S 的一个 $\frac{\gamma}{(1+\gamma)\log_b n}$-覆盖集。

这部分的内容主要是给出了定义和定理，为后面讨论中的环形树结构和算法提供了基础。

算法

这部分给出了算法正确性的证明，证明了算法满足以下四个断言：

- $S\subset A=\cup_j A_j$ - 这个断言的正确性来自于外层循环的终止条件。- 对于所

有的 $j \in \{1,\ldots,k\}$ 和任意的 $p \in S$，$\left|P \cap \cup_{q \in A_j} B(p,r)\right| \leq b\,|A_j|$ - 这个断言的正确性来自于内层循环的终止条件。- 对于所有的 $j \in \{1,\ldots,k\}$，$|A_j| \leq \delta|P|$ - 显然，对于任何 j，内层循环最多重复 $\log_b n$ 次。因此，$\max_{q \in A_j} d\,(p_j,q) \leq r\log_b n \leq \gamma\Delta(S)$。由于 S 是一个 (γ,δ)-聚类，我们有 $|B(p,\gamma\Delta(S)) \cap P| \leq \delta|P|$。因此，$|A_j| \leq \delta|P|$。- $r \leq \frac{\gamma\Delta(S)}{(1+\gamma)\log_b n}$ - 由于 $\Delta(A) \leq \Delta(S) + r\log_b n = \Delta(S) + \gamma\Delta(S) = (1+\gamma)\Delta(S)$。

这些断言证明了算法的正确性，说明算法生成的集合 $A_1,\ldots,A_k$ 构成了一个覆盖集，满足 $a(b,\delta,\frac{\gamma}{(1+\gamma)\log_b n})$-cover。

这段文字主要讲述了构建环覆盖树（ring-cover tree）的递归过程。根据定理 1 中的性质（1）和（2），将给定的点集 P 分解为一些较小的子集 $S_1,\ldots,S_l$，将这些子集分配给 P 的子节点。在节点 P 中还存储了一些附加信息，可以通过距离计算或点定位查询将最近邻搜索限制在 P 的一个子节点中。根据是性质（1）还是性质（2），分为两种情况，分别称为环节点（ring node）和覆盖节点（cover node）。

构建环覆盖树的过程中，根据特定条件将点集 P 分解为一些子集，并将这些子集分配给 P 的子节点。同时，在节点 P 中存储一些附加信息，用于最近邻搜索。

所述的搜索过程需要进行一定数量的距离计算或点定位查询，最终可以找到给定点 q 的一个近邻。

接下来的文章将分析环覆盖树的构建和搜索过程的复杂度，并证明环覆盖树的有效性。

构建环覆盖树的过程是递归的。对于根节点 P，根据定理 1 中的性质（1）和（2），将 P 分解为一些较小的集合 $S_1,\ldots,S_l$，并将这些集合分配给 P 的子节点。在节点 P 中还存储一些额外的信息，这些信息可以通过距离计算或点定位查询将最近邻搜索限制在 P 的一个子节点中。为了简化问题，假设我们可以调用一个精确的 PLEB（而不是 -PLEB）；这个构建过程可以很容易地修改为近似点定位。

根据性质（1）和性质（2），有两种情况。令 $\beta = 2\left(1+\frac{1}{e}\right), b = \frac{1}{\log^2 n}$，以及 $\alpha = \frac{1-1/\log n}{2}$。

情况 1：如果 P 满足性质（1），我们称之为环节点（ring node）。定义其子节点为 $S_1 = P \cap B(p,\beta r)$ 和 $S_2 = P - B(p,r)$。同时，在节点 P 中存储有关环分离器（ring separator）R 的信息。

情况 2：如果 P 满足性质（2），我们称之为覆盖节点（cover node）。定义 $S_i = P \cap \cup_{p \in A_i} B(p,r)$ 和 $S_0 = S - A$。在节点 P 中存储的信息如下：令 $r_0 = (1+1/e)\Delta(A)$，并且对于 $i \in \{1,\ldots,k\}$，定义 $r_i = r_n/(1+c)^i$，其中

$k = \log_{1+c} \frac{(1+1/c)\log_b n}{\gamma} + 1$。注意到 $r_n = \frac{\eta\Delta(A)}{\sqrt{\cos n(1+2)}} = \frac{r}{i+1}$。对于每个 r_n，生成一个球 $B(p, r_i)$ 的 PLEB 实例，其中 $p \in A$；所有实例都存储在节点 P 中。

在搜索过程中，可以通过进行一小部分测试将搜索限制在节点 P 的一个子节点中。搜索过程的具体步骤如下，此处省略了显而易见的基本情况。

这部分主要讲述了在 p 维空间中进行相似度搜索的算法。首先，通过均匀网格分割将 p 维空间划分为多个网格单元，每个网格单元之间的距离不超过 c。然后，对于每个球 B_i，定义其网格单元的集合为 $\vec{B_i}$。接着，在预处理阶段，将所有属于 $U_i\overline{B_i}$ 的元素存储到哈希表中，并记录相应的球的信息。在查询阶段，只需要找到包含查询点 q 的网格单元，并检查是否存在于哈希表中。为了保证算法的正确性，引入了局部敏感哈希（Locality-Sensitive Hashing）的概念，并应用于子线性时间的相似度搜索。局部敏感哈希定义了一族哈希函数，并要求具有一定相似度的点映射到同一个哈希值的概率较大，而具有较小相似度的点映射到同一个哈希值的概率较小。通过选择适当的哈希函数和参数，可以保证算法的准确性和效率。最后，给出了定理 4，根据定理 4 可以得出，如果存在一个满足条件的局部敏感哈希族，那么可以构建一个用于相似度搜索的算法，该算法的空间复杂度为 $O(dn + n^{1+\epsilon})$，查询时的时间复杂度为 $O(n^p)$，其中 $\epsilon = -\frac{\ln P_1}{\ln \frac{1}{7n}}$。

这部分主要讲述了选择适当的哈希函数和参数，使得函数满足局部敏感哈希族的性质 (1) 和 (2)。通过计算概率，可以得出满足性质 (1) 的哈希函数也会以至少 1/2 的概率满足性质 (2)。进一步地，通过选择一组随机的哈希函数，并使其满足性质 (1) 和 (2)，可以构建一个有效的相似度搜索算法。为了示例说明，分别考虑汉明度量和集合相似度这两种度量方式。对于汉明度量，使用一族投影函数进行快速哈希运算。对于集合相似度，使用之前使用过的草图函数来估计给定集合之间的相似度。根据定理 4 和命题 4 的结果，可以得出如果存在满足条件的局部敏感哈希族，那么可以构建一个用于相似度搜索的算法。该算法的空间复杂度为 $O(dn + n^{1+\epsilon})$，其中 d 是维数，n 是数据集的大小，ϵ 是一个常数。查询时的时间复杂度为 $O(n^p)$，其中 p 是 p 维空间中的 p 范数。通过选择适当的哈希函数和参数，可以实现高效的相似度搜索。最后，给出了推论 3，根据推论 3 可以得出，对于任何大于 1 的常数 ϵ，存在一个用于相似度搜索的算法，该算法的空间复杂度为 $O(dn + n^{1+1/c})$，查询时的时间复杂度为 $O(n^{1/\epsilon})$。该算法的哈希函数的计算复杂度为 $O(d)$。为了计算得到最优的参数 ρ，需要通过一系列推导和运算来估计 ρ 的值，并得出其上下界。

在这部分中，首先讨论了满足条件的哈希函数族对于汉明度量和集合相似度度量的应用。对于汉明度量，使用一族投影函数进行快速哈希运算；对于集合相似度，使用一组特定的哈希函数来计算相似度。根据命题 5 的结

果，可以得出，对于给定的 r_1 和 r_2，使用特定的哈希函数族 $\mathcal{H}$ 可以实现 (r_1, r_2, r_1, r_2)-敏感性。

根据推论 4，对于给定的 r 和 ϵ，存在一个用于集合相似度度量的算法。该算法的空间复杂度为 $O(dn + n^{1+p})$，其中 d 是维数，n 是数据集的大小，p 是查询时的权重。查询时的时间复杂度为 $O(n^p)$，其中 $\rho = -\frac{\ln \pi}{\ln}$。

然后，讨论了上述算法的一些应用。例如，在信息检索和分子聚类等领域中，可以使用点积作为相似度度量。通过合适的参数替换和技术，可以将点积相似度转化为集合相似度的问题，从而可以使用相似的算法来处理。

此外，还讨论了在数据增量更新的情况下，如何应用上述方法。通过简单的方法，可以实现插入、删除和查询操作。对于动态求解最近邻问题，结合了上述两种方法，可以实现近似的最近邻估计。最后，通过结合多个 PLBB 副本，可以回答近似的最近邻距离问题。该方法的时间复杂度为 $O(\log \log_{1+c} M)$，其中 M 是点坐标的绝对值的上界。

综上所述，本部分主要讨论了满足条件的哈希函数族的应用，并介绍了在不同度量方式下的算法和应用场景。这些算法在相似度搜索、最近邻问题等方面具有重要的应用价值。

这篇论文主要讨论了如何解决高维度数据中的“维数灾难”问题，即在高维度空间中进行最近邻搜索的困难。论文介绍了一种近似最近邻搜索的方法，旨在通过牺牲一定的准确性来加快搜索速度，从而解决高维度数据中的问题。

论文首先介绍了高维数据的特点，包括数据稀疏性和距离在高维空间中的失真。由于高维度空间的稀疏性，数据点之间的距离往往会变得更远，导致传统的最近邻搜索方法效果不佳。另外，由于维度的增加，距离度量的空间被拉伸和扭曲，使得传统的欧几里德距离等度量在高维空间中失效。

为了解决这个问题，论文提出了一种近似最近邻搜索的方法。该方法通过构建一个数据结构来组织数据集，以便在搜索时能够快速定位到最近邻的候选集。然后，通过计算候选集中的点与查询点之间的距离，并选择最接近的点作为近似最近邻。

论文介绍了两种常用的近似最近邻搜索方法：局部敏感哈希（LSH）和随机投影树（RP 树）。局部敏感哈希使用一组哈希函数，将相似的数据点映射到相同的桶中，从而实现快速的最近邻搜索。随机投影树则通过将数据点逐层分割为子空间，并在每个子空间中构建二叉树来进行搜索。这些方法都能够在高维度数据中有效地进行近似最近邻搜索。

论文还讨论了近似最近邻搜索的准确性与搜索速度之间的权衡。根据应用需求，可以选择不同的方法来进行搜索，以获得更快的速度或更高的准确性。

综上所述，这篇论文主要介绍了近似最近邻搜索的方法，并探讨了如何解决高维度数据中的“维数灾难”问题。这些方法在处理大规模高维度数据集时具有重要的应用价值。

这段定义可以这么理解：

在给定一个点集 P 和一组子集 $A_1, A_2, \ldots, A_l$ 的前提下，如果存在一个半径 r，满足以下条件：

1. 对于所有 $S \subset A$，其中 $A = \bigcup_{i=1}^{l} A_i$，都有 S 中的所有点 p 都满足 $d(p, A) \leq r$，其中 $d(p, A)$ 表示点 p 到集合 A 的最短距离。2. 对于所有 $i = 1, 2, \ldots, l$，都有满足 $|P \cap (\cup_{p \in A_i} B(p, r))| \leq b|A_i|$，其中 $B(p, r)$ 表示以点 p 为中心，半径为 r 的球形区域。3. 对于所有 $i = 1, 2, \ldots, l$，都有满足 $|A_i| \leq c|P|$。

那么这个子集 $A_1, A_2, \ldots, A_l$ 就称作是 P 的一个 $a(b, c, d)$-覆盖集。

其中，b, c, d 是一些预先设定的参数，$\Delta(A)$ 是集合 A 中最远点对之间的距离。总体来说，这个定义的意思是希望找到一组点的子集，使得这个子集能够覆盖原始点集 P 中的所有点，同时满足一些额外的限制条件，这样的话在进行一些算法设计时可以更加方便地操作。

假设我们有一个点集 P，其中包含了一些二维平面上的点。我们要找到 P 的一个覆盖集，这个覆盖集由一些子集 A_1, A_2, A_3 组成。

现在，让我们逐步解释定义中的条件：

1. 存在一个半径 r：意味着我们需要找到一个半径 r，它至少足够大，能够包含点集 P 中的所有点。也就是说，任何点 p 都满足 $d(p, A) \leq r$，其中 $A = \bigcup_{i=1}^{3} A_i$。

2. $|P \cap (\cup_{p \in A_i} B(p, r))| \leq b|A_i|$：这个条件要求对于每个子集 A_i，以其内的点为中心，半径为 r 的球形区域中的与点集 P 的交集大小不超过 b 乘以 A_i 的大小。简而言之，这个条件指定了在子集 A_i 的球形区域中，点集 P 中的点的数量。

3. $|A_i| \leq c|P|$：这个条件要求每个子集 A_i 的大小不超过 c 乘以点集 P 的大小。

综合起来，一个 $a(b, c, d)$-覆盖集就是由一些子集 A_1, A_2, A_3 组成的集合，满足上述条件。具体来说，它们可以覆盖整个点集 P，同时满足与点集 P 的交集数量的限制和子集大小的限制。

这只是一个简单的示例，希望可以帮助您更好地理解定义。实际应用中，具体的参数和具体的问题会导致更具体的定义和约束条件。