Hans Journal of Data Mining 数据挖掘, 2023, 13(2), 107-116
Published Online April 2023 in Hans. https://www.hanspub.org/journal/hjdm
https://doi.org/10.12677/hjdm.2023.132011

Hans汉斯

具有重叠集合约束的实体解析问题研究

樊沁怿，李　贵，李征宇

沈阳建筑大学，信息与控制工程学院，辽宁 沈阳

收稿日期：2023年3月3日；录用日期：2023年4月3日；发布日期：2023年4月13日

摘　要

本文研究了具有重叠集合约束的实体解析集合相似性连接问题。给定两个集合内元素为集合的集合以及一个常数c，找到数据集当中至少共享了c个共同元素的所有集合对。这一问题是许多领域诸如信息检索、数据挖掘和实体解析当中的基本操作。现有的方法都受到了$O(n^2)$的限制，其中n是数据集总的大小。本文提出了一种算法复杂度为$O\left(n^{2-\frac{1}{c}}k^{\frac{1}{2c}}\right)$的集合大小感知算法。集合大小感知算法根据集合大小将所有集合分为大集合与小集合并分别进行处理。本文通过现有的方法来处理大集合，对于小集合本文提出了集中启发式算法来提高算法性能。由于大集合与小集合的大小边界对于效率至关重要，本文还提出了一种有效的大小边界的选择方法来选择合适的大小边界。本文通过在真实数据集上的实验结果证明了该方法的有效性。

关键词

相似连接，重叠约束，集合

Entity Resolution Algorithm Based on Locality Sensitive Hash and Fuzzy Join

Qinyi Fan, Gui Li, Zhengyu Li

School of Information & Control Engineering, Shenyang Jianzhu University, Shenyang Liaoning

Received: Mar. 3rd, 2023; accepted: Apr. 3rd, 2023; published: Apr. 13th, 2023

Abstract

In this paper, the entity resolution set similarity connection problem with overlapping set constraints is studied. Given two sets whose elements are sets of sets and a constant *c*, find all sets

文章引用: 樊沁怿, 李贵, 李征宇. 具有重叠集合约束的实体解析问题研究[J]. 数据挖掘, 2023, 13(2): 107-116.
DOI: 10.12677/hjdm.2023.132011

that share at least *c* common elements in the data set. This problem is a fundamental operation in many fields such as information retrieval, data mining, and entity resolution. Existing methods are limited by $O(n^2)$, where *n* is the total size of the dataset. This paper presents a set size awareness algorithm with $O\left(n^{2-\frac{1}{c}}k^{\frac{1}{2c}}\right)$ complexity. The set size sensing algorithm divides all sets into large sets and small sets according to the set size and processes them separately. In this paper, the existing methods are used to deal with large sets. For small sets, a centralized heuristic algorithm is proposed to improve the algorithm performance. Since the size boundary of large sets and small sets is very important for efficiency, this paper also proposes an effective size boundary selection method to select the appropriate size boundary. Experimental results on real data sets demonstrate the effectiveness of the proposed method.

Keywords

Set Similarity, Overlap, Set

1. 引言

判断相似实体作为实体解析的主要任务之一，对于模式未知或具有多值属性的实体，我们可以通过令牌化的方式将实体转换成是一个(令牌)标记集合，实体属性值的标记看作是集合当中的元素[1]，这样一来，通过判断集合的相似性来判断实体是否为相似实体。

具有重叠约束的集合相似性连接，给定两个集合(例如文档的主题集合)和一个常量 c，找到所有共享至少 c 个公共元素的集合对，我们可以认为这些共享有 c 个公共元素的集合对之间是相似的。

给定一组文档，其中每个文档都包含多组字符串，对于每个词，我们可以获得一组包含该词的文档。许多词嵌入模型均利用一对单词同时出现的文档数量，例如基于深度学习的模型 GloVe [2]和 Word2Vec [3]以及基于经典矩阵分解的模型 HAL [4]和 PPMI [5]。重叠集相似度连接可以为这些模型构建单词共现矩阵，因为两个单词的共现与其对应文档集的重叠大小相同。此外，推荐系统[6]经常使用重叠集合来解释推荐，以获得更好的透明度和用户体验。

在算法设计方面，有一个简单解决方案，即简单地比较所有的集合对，并在 $O(n^2)$时间内完成，其中 n 是数据集总的大小。

现有方法[7] [8] [9]利用各种启发式方法来提高效率。然而，不幸的是，所有这些解决方案仍然受到 $O(n^2)$限制，即渐近地与暴力的解决方案一样糟糕。

本章研究了具有重叠约束的集合相似性问题。给定两个集合以及一个常数 c，在数据集中找到至少共享 c 个公共元素的所有集合对。该问题是诸如数据挖掘、信息检索和机器学习等诸多领域的基本操作。现有方法的时间复杂度都是 $O(n^2)$，n 是所有集合的总的大小。在本章中提出了一种大小感知算法，其算法复杂度为：$O\left(n^{2-\frac{1}{c}}k^{\frac{1}{2c}}\right)$，其中 k 指的是结果的数量。大小感知算法根据集合的大小将所有集合分为大集合与小集合并对它们进行分别处理。然后使用现有的方法对大集合进行处理，在本节中重点关注

被划分为小集合的集合。本节中为小集合提出了几种启发式算法，来显著提高其处理效率。由于大集合与小集合之间的大小边界对于处理效率至关重要，因此本章中还提出了一种大小边界的选择方法。本章的主要贡献如下：

本章研究了具有重叠约束下的集合相似性连接问题，设计了一种集合大小尺寸感知算法，将所有集合分为大集合与小集合，分别进行处理。对于小集合我们枚举他们所有大小为 c 的子集，并选取至少共享一个大小为 c 的子集的集合作为结果。为了避免小集合枚举不必要的大小为 c 的子集，我们还设计了两种优化方案。针对大小集合选择的尺寸边界，提出了一种有效的方法来寻找理论上最佳的尺寸边界。实验结果证明该边界是较优的解。

2. 问题定义

输入：集合 R、集合 S 以及常量 c，其中集合 R 与集合 S 当中的元素均为集合

输出：$\left\{\left\langle R',S'\right\rangle \mid \left\langle R',S'\right\rangle \in R\times S, \text{且} \left|R'\cap S'\right| \geq c\right\}$

给定两个元素均为集合的集合 R 与 S，并给定常量 C。这里的常量 C 定义了集合相似的标准。问题就是如何在 $R \times S$ 空间中找到所有相似的集合对 $\left\langle R',S'\right\rangle$，其中 R'和 S'是相似的。并且为了接下来叙述的简洁在这里定义，任意一组从集合 R 与集合 S 中获取的元素 c 称之为 c-subset。基于两个集合相似当且仅当他们共享一个任意的 c-subset。这一发现促使我们在所有的 c-subset 上建立倒排索引，找出那些共享公共 c-subset 的集合，并依次通过检查每一个倒排索引来计算连接结果。因我们避免枚举不同的集合对，即不共享任何公共 c-subset 的集合对。但是这种方法仅适用于小尺寸的集合，因为小尺寸的集合具有少量的 c-subset。另一方面，大集合的数量也不能太大，因为我们对大集合采用的是暴力的方法。

下面对这个问题进行举例说明：

$R=\left\{R_1,R_2,R_3\right\}$

$R_1=\left\{e_1,e_2\right\}$，$R_2=\left\{e_1,e_3,e_4,e_6\right\}$，$R_3=\left\{e_2,e_5,e_7\right\}$

$S=\left\{S_1,S_2\right\}$，

$S_1=\left\{e_2,e_5,e_6\right\}$，$S_2=\left\{e_1,e_2,e_4,e_5,e_6,e_7,e_8\right\}$

给定常量 $c=2$

则仅当两个集合 R'，S'交集大小 $\left|R_1\cap S_1\right|\geq 2$ 时认为 R'和 S'相似

$R\times S=\left\{\left\langle R_1,S_1\right\rangle,\left\langle R_2,S_1\right\rangle,\left\langle R_3,S_1\right\rangle,\left\langle R_1,S_2\right\rangle,\left\langle R_2,S_2\right\rangle,\left\langle R_3,S_2\right\rangle\right\}$ 其中相似集合对有 $\left\langle R_3,S_1\right\rangle$，$\left\langle R_1,S_2\right\rangle$，$\left\langle R_2,S_2\right\rangle$，$\left\langle R_3,S_2\right\rangle$，因此输出的结果应该为这四个集合对。

暴力方法通过枚举 $R \times R$ 中的每个集合并计算他们的重叠部分大小。暴力方法的时间复杂度为 $O(n^2)$。

3. 集合大小感知算法

3.1. 算法主要思想

1) 将所有的集合元素进行划分，根据他们的大小将他们划分为大集合 R_l 以及小集合 R_s，选定边界 x，大集合即为集合大小大于 x 的集合，小集合为集合大小小于 x 的集合。将 $R \times R$ 笛卡尔积空间划分为 $R_l \times R$ 与 $R_s \times R_s$。($R_s \times R_l$ 的集合已经被包含在 $R_l \times R$ 当中，只是当中的顺序并不相同，因此在这里不做考虑)。

2) 对于 $R_l \times R$，可以采用现有的方法进行处理，在本节中采用暴力搜索，即遍历大集合当中的每一个元素 R_i，再遍历 R 中的每一个集合，直接进行 $\left|R_i\cap R\right|$，看大小是否 $\geq c$。

3) 对于 $R_s \times R_s$，为每个 c-subset 建立倒排索引 $L[r]$。这样使得 $L[r]$中的每个不同集合都至少包含这个 c-subset 子集，满足相似性阈值 c。而从 $L[r]$中不同集合所组成的集合对也满足相似性阈值 c。

3.2. 算法框架

Input：集合 $R=\{R_1,R_2,\cdots,R_m\}$，指定常量 c

Output： $A=\{\langle R_i,R_j\rangle \mid |R_i\cap R_j|\geq c\}$

① x = Get Size Boundary (R, c) //确定大集合与小集合之间的划分标准

② 依据 x 将 R 的元素划分为大集合 R_l 和小集合 R_s，初始化 $A=\phi$

③ for each 大集合 $R_i\in R_l$ do

for each $R_j\in R$ do

If $|R_i\cap R_j|\geq c$

将 $\langle R_i,R_j\rangle$ 插入 A 中//对于大集合直接暴力搜索

④ for each 小集合 $R_j\in R_s$ do

for each R_j 的大小为 c 的子集 r do

将 R_j 插入到 $L[r]$中//小集合对 c-子集建立倒排索引

⑤ for each $L[r]$ do

将 $L[r]$中不同集合组成集合对加入 A 中

⑥ return A

该算法囊括了 $R_l \times R$ 笛卡尔积空间的所有元素，对于 $R_s \times R_s$ 部分，对于其中任意两个集合 R_1、R_2 满足相似性阈值 c，则他们的交集大于等于 c，取该交集的一个子集 r，r 的大小为 c，则必有 R_1、R_2 均在 $L[r]$中；而若 $L[r]$中有两个集合 R_1、R_2 中，则说明 R_1、R_2 的交集中至少有一个 c-subset，因此 $|R_1\cap R_2|\geq c$。由此可以得出结论 $R_s \times R_s$ 算法是找到小集合中所有满足条件的集合对的充要条件。又因为 $R_s \times R_l$ 的集合已经被包含在 $R_l \times R$ 当中，所以 $R_s \times R_l$ 和 $R_s \times R_s$ 已经囊括了解空间，因此一定能够生成所有的解。这就是整个算法的框架，针对该算法框架还提出了对于小集合处理方法的优化以及划分边界 X 的确定。

Table 1. Data set R
表 1. 数据集 R

	id	Set	Size
R_s	R_1	$\{e_1, e_2, e_3, e_4\}$	4
	R_2	$\{e_1, e_2, e_5, e_6\}$	4
	R_3	$\{e_2, e_3, e_5, e_7\}$	4
	R_4	$\{e_1, e_3, e_5, e_7\}$	4
	R_5	$\{e_4, e_6, e_7\}$	3
R_l	R_6	$\{e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$	7
	R_7	$\{e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}\}$	8
	R_8	$\{e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{16}\}$	8

Table 2. $R_l \times R$
表 2. $R_l \times R$

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
R_6	3	3	4	4	3	/	/	/
R_7	0	0	0	0	0	1	/	/
R_8	0	0	0	0	0	1	7	/

示例：调用表 1 中的数据集 R，并且假设阈值 c 为 2，大小边界 x 为 5。根据大小边界 x 我们将集合划分为大集合与小集合，得到 $R_s=\{R_1,R_2,R_3,R_4,R_5\}$ 以及 $R_l=\{R_6,R_7,R_8\}$。第一步我们首先枚举 $R_l\times R$ 中的所有集合对，并对它们的交集大小进行计算得到的结果如表 2 所示，符合阈值 c 条件的有 $\langle R_1,R_6\rangle$ $\langle R_2,R_6\rangle$ $\langle R_3,R_6\rangle$ $\langle R_4,R_6\rangle$ $\langle R_5,R_6\rangle$ $\langle R_7,R_8\rangle$。然后我们为小集合当中找到的所有大小为 2 的子集构建倒排索引。该倒排索引如表 3 所示，倒排索引 $L\{e_1e_2\}$有两个集合 R_1 与 R_2，根据该倒排列表我们可以得到相似对 $\langle R_1,R_2\rangle$。类似地还能从倒排列表 $L\{e_1e_3\}L\{e_1e_5\}L\{e_2e_3\}$……等获得相似对。

Table 3. The c-subset inverted index for small sets
表 3. 小集合 c-subset 倒排索引

	e_1e_2	e_1e_3	e_1e_4	e_1e_5	e_1e_6	e_1e_7	e_2e_3	e_2e_4	e_2e_5	e_2e_6	e_2e_7	e_3e_4	e_3e_5	e_3e_6	e_3e_7	e_4e_5	e_4e_6	e_4e_7	e_5e_6	e_5e_7	e_6e_7
R_1	√	√	√				√	√				√									
R_2	√			√	√				√	√									√		
R_3							√		√		√		√		√					√	

集合大小感知算法划分大小集合灵感来源：现有方法为集合中的元素构建倒排索引 I。每个倒排列表 $I[e]$都要被扫描$|I[e]|$次，其中$|I[e]|$指的是倒排列表的长度。因此现有方法需要 $O(|I[e]|^2)$来处理每个倒排列表 $I[e]$。然而在集合大小感知算法中，因为总共有 n 个元素，且大集合内包含的元素大于 X，所以大集合的数量不超过 n/x 个。因此大集合对于倒排列表长度的贡献最多为 n/x。相反，因为小集合的大小没有限制，所以小集合最多可以为倒排列表长度的贡献为 n，从而导致时间复杂度为 $O(n^2)$。这就是现有方法无法在小集合上执行，但是我们在大集合上执行现有办法的原因。

3.3. 小集合处理法方法的优化

枚举每一个小集合中的所有 c-subset 成本十分巨大，尤其是当小集合具有大尺寸的时候。对于每个小集合，大小感知算法需要枚举其所有 c-subset 子集 r 以构建完整的倒排索引 $L[r]$并基于它生成结果。如果一个 $L[r]$只有单一的集合，即它在所有小的集合中只出现一次，我们可以避免生成它并获得一个可以生成所有结果的精简倒排索引，因为唯一的 $L[r]$不能产生任何结果。

由于存在大量唯一的 c-subset 子集，因此避免通过他们生成 $L[r]$很重要。例如在表 3 当中有诸如 e_1e_3、e_1e_4、e_1e_7 等 c-subset。他们都是唯一的 c-subset，因此并不需要生成他们。将这一优化命名为 Heapskip。

我们给出的跳过 c-subset 的方法如下：

首先对所有小集合中的 c-subset 进行一个全局的顺序。我们按照升序的方式来访问 c-subset，并将最小的未访问的 c-subset 表示为 min-subset。最小堆 H 用来管理所有最小的 min-subset。每次通过堆顶获得一个当前最小的 c-subset 集合 r，假定 r 来自小集合 R，设 r_2 是剔除 r 之后的堆顶元素，则 r_2 可以表示为来自 R_2 的 $R_s\backslash R$ 集合中最小的 c-subset。

R 中的 c-subset 集合 r'满足 $r<r'<r_2$。由于 r_2 代表了 $R_s\backslash R$ 中最小的 c-subset，因此 $R\backslash R_s$ 中一定不存在集合 R_3，使得 R_3 包含 r'，否则应有 $r_2=r'$为更小的 c-subset。从而 r'子集是无需建立倒排索引的，因为其中只可能包含集合 R 而不会出现集合对。

由此，因为每个集合的 c-subset 已经被我们排序了，我们可以在 R 中使用二分搜索，找到 R 中第一个“不小于”r_2 的 c-子集 r_3，将其加入堆 H，从而跳过 r 和 r_3 之间的所有 c-subset。

我们的优化除了去除唯一的 c-subset 还有去除掉冗余的 c-subset。我们将这一优化命名为 HeapDedup。

具体优化算法如下：

Input：R_s：所有小集合的集合；c：常量，集合相似的判断标准

Output：精简的倒排索引 L

① Fix a global order for all the elements in R_s;

② Insert all the min-subsets of small sets to a heap H;

③ While $H \neq \emptyset$

记 min = H 的堆顶元素，删除 min。得到 min 的来源 R，将 R 加入 L[min]中。

top 为删除 min 之后的堆顶元素

if top ≠ min then

for each R_i in L[min] do

对 R_i 进行二分搜索，找到 R_i 中第一个 ≥top 的 c-subset，插入 H 中

else

continue

④ return L

对 R_s 中所有小集合中的所有元素 e 进行全局标号(标号规则任意选取，如随机生成下标)，标号小，则认为该元素“更小”，同时利用这种标号新定义集合间的“大小关系”：对集合 R_1，R_2，若 R_1 中“最小”(标号)的元素比 R_2 中“最小”的元素“更小”，则 R_1 小于 R_2，若相等，则比较“次小”元素・・・；据此，将每个小集合中的 c-subset 按照这种“大小关系”进行排序。

对于每个小集合，将每个小集合的“最小”(依据标号)的 c-subset 拿出来建立一个最小堆 H，入堆的同时记录每个子集来自哪一个小集合 R_i。

提出该优化的主要是基于如下的思想：

$L[r_1] \subseteq L[r_2]$，则 r_1 是冗余的，因为 $L[r_1]$产生的集合对一定包含在 $L[r_2]$的集合对中。

②对于 r_1 和 $L[r_1]$，设 r_2 为 $R_s \backslash L[r_1]$的集合中 min-subset，则满足：$r_1 < r < r_2$ 的集合 r 是冗余的。首先 r 一定是 $L[r_1]$中集合的某个 c-subset，否则应有 $r_2 = r$。其次 $L[r] \cap (R_s \setminus L[r_1]) = \emptyset$.否则在 $R_s \backslash L[r_1]$中存在集合 R 包含集合 r，由①可知，R 应在 $L[r_1]$中，矛盾。从而 $L[r] \subseteq L[r_1]$，r 是冗余的。

每次循环时若 min = top，直接 continue 实现 lazy 处理，从而下次选取时将 top 所在集合 R'加入 L[min]中

每次循环时若 min ≠ top，由于 top 是当前 min-subset，只有两种可能：

① top 为 $R_s \backslash L$[min]的 min-subset。则由上面的分析，所有位于 min 和 top 之间的 c-子集都是冗余的，可以跳过。

② top 为 L[min]的最小 c-subset (除 min 之外)，设 r 为 $R_s \backslash L$[min]的 min-subset，则有 r 大于等于 top。故位于 min 和 r 之间的所有 c-subset 都是冗余的，但由于 r 未知，我们转而利用 top，因为 top 小于等于 r，必有 min 和 top 之间的 c-subset 都是冗余的。

无论如何，min 和 top 之间的 c-子集均是冗余的，从而我们对于 L[min]中的每个集合都利用二分搜索找到第一个大于等于 top 的 c-subset，插入到堆 H 中，将那些冗余子集跳过。同时由于 min = top 时的 lazy 处理，我们在 min ≠ top 时的处理是针对 L[min]中一组集合的操作，包含了 Heapskip 的功能。

3.4. 边界 X 的选择

我们用 x 来表示算法的时间复杂度，然后找到最佳的 x 以最小化成本。

① 首先计算大集合 $R_l \times R$ 的时间复杂度。我们为所有的大集合建立哈希表，从而可以在常数时间内判断元素 e 是否属于该集合。并且由于大集合的大小大于等于 x，所以大集合的数量至多为 n/x 个。通过 R'中的每一个元素检索 R 的哈希表。所以对于一个大集合 R_1，计算交集 $|R_1 \cap R'|$ 的时间是 $O(|R'|)$，因此大集合的时间复杂度为：

$$\sum_{R' \in R} O\left(|R'| \cdot \frac{n}{x}\right) = O\left(\frac{n^2}{x}\right) \tag{1}$$

② 然后计算 $R_s \times R_s$ 小集合的时间复杂度。第一步枚举所有 c-subset 以构建倒排索引；第二步从倒排索引中生成相似的对。一个小集合 R 有 $\binom{|R|}{c}$ 个，由于 $|R| \le x$ (因为 R 是一个小集合)，所以小集合 c-subset 的总数最多为：

$$\sum_{R \in \mathcal{R}_s} \binom{|R|}{c} \le \sum_{R \in \mathcal{R}_s} |R|^c \le x^{c-1} \sum_{R \in \mathcal{R}_s} |R| \le x^{c-1} n \tag{2}$$

第一步的枚举成本趋近于上述的公式结果。即第一步的成本以 $O(x^{c-1}n)$ 为界限。

第二步的成本来自生成每个倒排索引的集合对。因此第二步的时间复杂度为 $\left(\sum_{i=1}^{l} |\mathcal{L}_i|^2\right)$。由于所有倒排索引的总的长度是所有小集合中 c-subset 的个数，所以 $\sum_{i=1}^{l} |\mathcal{L}_i| \le x^{c-1} n$。因为 k 是 $R \times R$ 相似集合对的总数，因此 $\mathcal{L}_i$ 中生成的相似集合对的数量不能超过 k，即对于任意的 $\mathcal{L}_i$，都有 $\frac{|\mathcal{L}_i|(|\mathcal{L}_i|-1)}{2} \le k$。因此第二步的复杂度为：

$$O\left(\sum_{i=1}^{l} |\mathcal{L}_i|^2\right) = O\left(\sqrt{k} \sum_{i=1}^{l} |\mathcal{L}_i|\right) = O\left(x^{c-1} n \sqrt{k}\right) \tag{3}$$

假设 k 已知的时候，算法的复杂度为：

$$O\left(\frac{n^2}{x} + x^{c-1} n \sqrt{k}\right) \tag{4}$$

我们取 $x = \left(n/\sqrt{k}\right)^{1/c}$ 使得时间复杂度最低。在这种情况下时间复杂度为：

$$O\left(n^{2-\frac{1}{c}} k^{\frac{1}{2c}}\right) \tag{5}$$

例如按照表 1 的数据集，当阈值 $c = 2$ 时，$n = \sum_{i=1}^{7} |R_i| = 40$，$k = 5$，因此设置 $x = \left(40/\sqrt{5}\right)^{1/2} \approx 5$。

但 k 往往是未知的，因此我们使用“doubling Trick”，用 k'猜测 k，从 $k' = 1$ 开始尝试，如果正确，则应该执行的步骤个数为 $\alpha \cdot n^{2-\frac{1}{c}\hat{k}^{\frac{1}{2c}}}$ (α 是总复杂度中隐含的常量)。当执行的步骤为 $1+\alpha \cdot n^{2-\frac{1}{c}\hat{k}^{\frac{1}{2c}}}$ 时，我们知道了 $k' < k$，重复操作直到 $k' \ge k$，每一轮的时间为 $O\left(n^{2-\frac{1}{c}} k^{\frac{1}{2c}}\right)$。为了达到所需要的复杂度，我们从 $k' = 1$ 开始。在其终止时，k'的值最多为 $2k$，否则算法将会在前一轮终止。因此所有轮次的总的运行时间为：

$$O\left(\sum_{i=0}^{\log_2(2k)} n^{2-\frac{1}{c}}\left(2^i\right)^{\frac{1}{2c}}\right)=O\left(n^{2-\frac{1}{c}}k^{\frac{1}{2c}}\right) \tag{6}$$

4. 实验

4.1. 数据集

本节使用了从“面向领域 Web 数据抽取”项目所抽取的多个数据源的房地产数据进行实验。每个楼盘、楼栋都是一个集合，绿化率、建筑面积、总户数、楼栋数、房屋面积等等楼盘、楼栋的属性都是元素。

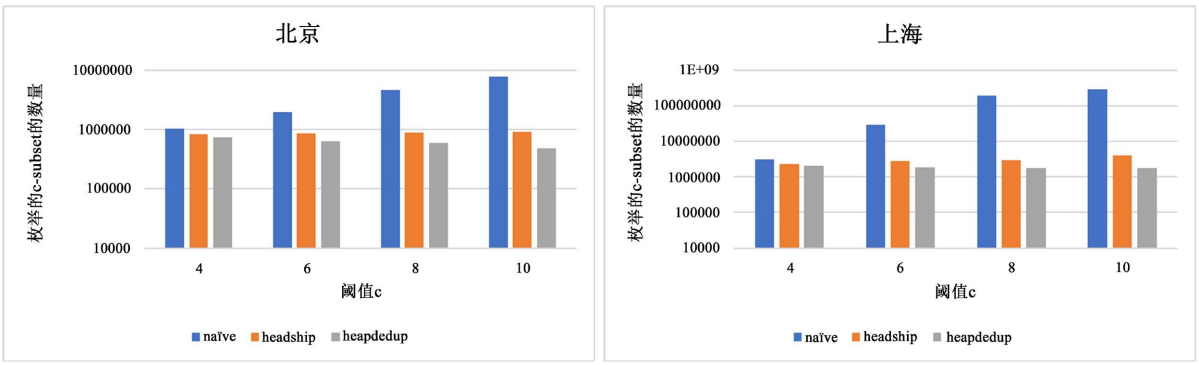
第一组实验旨在确定处理小集合的最佳的基于堆的处理方案。为此使用了两组来自不同城市的楼盘、楼栋、户型数据，如表 4 所示：

Table 4. Data sets
表 4. 数据集

城市	楼盘	楼栋	总计
北京	2212	12,348	14,560
上海	5404	307,450	312,854

4.2. 优化方案评估

我们使用了以下的三种方法：1)：naive，枚举每一个小集合的 c-subset；2)：headship，利用最小堆跳过唯一的 c-subset；3)：heapdedup，利用最小堆跳过唯一的 c-subset 以及相邻的冗余 c-subset。我们通过改变阈值并就不同阈值下 c-subset 的数量(即堆弹出操作的数量)进行比较如下图 1 所示：



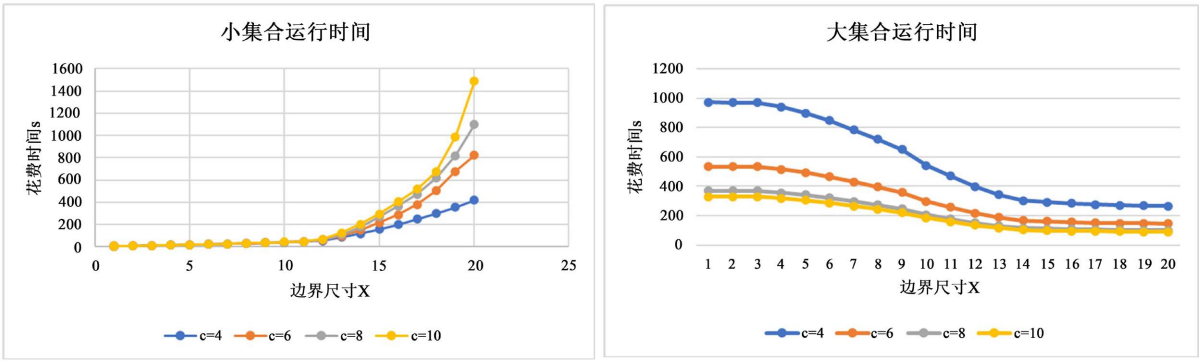
(a) (b)

Figure 1. Comparison of the number of c-subset enumerated by each method
图 1. 各方法枚举出的 c-subset 数量的比较

首先我们观察到随着阈值 c 的增大，使用 navie 方法枚举出的 c-subset 的数量指数级的往上增加，headship 与 heapdedup 两种优化方法都显著的减少了枚举的 c-subset 的数量。我们还发现了一个现象随着阈值 c 的增大，采用 headship 优化后枚举的 c-subset 的数量还是在缓慢增加，而采用 heapdedup 优化后枚举的 c-subset 随着阈值 c 的增大而减小。该对比试验验证了 headship 与 heapdedup 两种优化的有效性，并且 heapdedup 方案优于 headship 方案。

4.3. 尺寸边界选择评估

本小节的实验集中在对于尺寸边界的选择上。我们试验了在不同阈值 c 下边界 x 对于实验时间的影响。评估了尺寸感知算法对于大小集合的处理时间。如下图 2 所示。



(a) (b)

Figure 2. The influence of boundary X on the running time of small sets under different thresholds
图 2. 不同阈值下边界 X 对于大小集合运行时间的影响

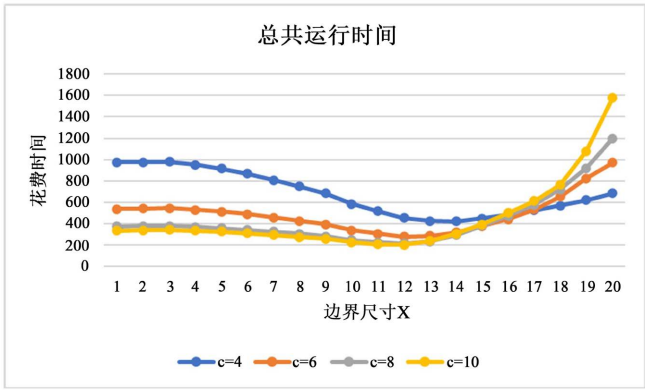


Figure 3. The influence of boundary X on the total time of the set under different thresholds
图 3. 不同阈值下边界 X 对于集合总共运行时间的影响

Table 5. 不同大小边界花费的时间
表 5. Time spent on different size boundaries

	尺寸边界感知算法		实际最佳	
	x	时间/s	x	时间/s
c = 4	13	427	14	420
c = 6	14	316.2	13	277.9
c = 8	13	234.2	12	215.2
c = 10	11	206.8	12	203.3

图 2 给出了处理小集合所需要的花费的时间，图 3 给出了处理大集合所需要的花费的时间，图 3 给出了总共需要花费的时间。我们发现随着尺寸边界的增加，大集合时间成本的下降最初是相当显著的，

但是后来变得微不足道。对于小集合一开始时间成本增长缓慢，之后时间成本急剧增加。

表 5 展示了由我们尺寸感知算法选出的最佳的 x 以及实际情况下最佳的 x，此外还展示了每个尺寸边界下运行的时间。可以看出我们尺寸边界感知算法非常有效给出的最佳边界是接近最佳值的较好的值。

5. 总结

本节中提出了一个在具有重叠约束的集合相似性连接问题中算法复杂度为$O\left(n^{2-\frac{1}{c}}k^{\frac{1}{2c}}\right)$的算法。其中 n 是所有集合的总的大小，常数 c 是相似度阈值，k 是结果中相似集合对的数量。将所有集合按照边界 x 分成大集合与小集合，分别进行处理。对于小集合，我们枚举其所有的 c-subset，为了避免不必要的 c-subset，还提出了两种优化方案，一种是对无法生成任何结果的唯一的 c-subset 进行删减。另一种对仅生成重复结果的冗余 c-subset 进行删减。并且设计了一种选择边界 x 的方法并验证了其有效性。实验发现随着尺寸边界的增加，大集合时间成本的下降最初是相当显著的，但是后来变得微不足道。对于小集合一开始时间成本增长缓慢，之后时间成本急剧增加。由尺寸感知算法选出的最佳的边界 x 是接近最佳值的较好的值。

参考文献

[1] Satuluri, V. and Parthasarathy, S. (2012) Bayesian Locality Sensitive Hashing for Fast Similarity Search. *Proceedings of the VLDB Endowment*, **5**, 430-441. https://doi.org/10.14778/2140436.2140440

[2] Pennington, J., Socher, R. and Manning, C. (2014) Glove: Global Vectors for Word Representation. In: *Proceedings of the* 2014 *Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), Association for Computational Linguistics, Doha, 1532-1543. https://doi.org/10.3115/v1/D14-1162

[3] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J. (2013) Distributed Representations of Words and Phrases and Their Compositionality. *Proceedings of the* 26*th International Conference on Neural Information Processing Systems*, **2**, 3111-3119.

[4] Lund, K. and Burgess, C. (1996) Producing High-Dimensional Semantic Spaces from Lexical Co-Occurrence. *Behavior Research Methods, Instruments, & Computers*, **28**, 203-208. https://doi.org/10.3758/BF03204766

[5] Bullinaria, J.A. and Levy, J.P. (2007) Extracting Semantic Representations from Word Co-Occurrence Statistics: A Computational Study. *Behavior Research Methods*, **39**, 510-526. https://doi.org/10.3758/BF03193020

[6] Bayardo, R.J., Ma, Y. and Srikant, R. (2007) Scaling up All Pairs Similarity Search. In P*roceedings of the* 16*th International Conference on World Wide Web*, Association for Computing Machinery, New York, 131-140. https://doi.org/10.1145/1242572.1242591

[7] Li, C., Lu, J. and Lu, Y. (2008) Efficient Merging and Filtering Algorithms for Approximate String Searches. 2008 *IEEE* 24*th International Conference on Data Engineering*, Cancun, 7-12 April 2008, 257-266. https://doi.org/10.1109/ICDE.2008.4497434

[8] Wang, J., Li, G. and Feng, J. (2012) Can We Beat the Prefix Filtering?: An Adaptive Framework for Similarity Join and Search. In: *Proceedings of the* 2012 *ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, New York, 85-96. https://doi.org/10.1145/2213836.2213847

[9] Malik, A.S., Boyko, O., Atkar, N. and Young, W.F. (2001) A Comparative Study of MR Imaging Profile of Titanium Pedicle Screws. *Acta Radiologica*, **42**, 291-293. https://doi.org/10.1080/028418501127346846