厦门大学学位论文著作权使用声明

本人同意厦门大学根据《中华人民共和国学位条例暂行实施办法》等规定保留和使用此学位论文，并向主管部门或其指定机构送交学位论文（包括纸质版和电子版），允许学位论文进入厦门大学图书馆及其数据库被查阅、借阅。本人同意厦门大学将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于：

（　　）1.经厦门大学保密委员会审查核定的保密学位论文，于　年　月　日解密，解密后适用上述授权。

（ √ ）2.不保密，适用上述授权。

（请在以上相应括号内打“√”或填上相应内容。保密学位论文应是已经厦门大学保密委员会审定过的学位论文，未经厦门大学保密委员会审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权。）

声明人（签名）：何玲

2013 年 6 月 7 日

摘 要

随着信息化的高速发展和客观上硬件技术的有效支撑，使得数据集中的海量数据不免存在冗余、缺失、不确定数据和不一致数据等诸多情况，这些统称为“脏数据”。人们需要从数据集中获取真实可靠的数据就必须进行数据清洗。而重复记录检测是数据清洗领域中的研究热点。本文首先介绍了数据仓库、数据清洗以及重复记录检测的相关知识，包括数据清洗的原理、方法、基本流程和工具以及重复记录检测匹配算法和重复记录检测清除算法。在此基础上，本文提出了三个改进算法。分别是基于传递闭包的重复记录检测算法，基于属性分析的重复记录检测算法和基于完全子图的重复记录检测算法。基于传递闭包的重复记录检测算法在 SNM 算法的基础上提出了三个方面的改进，分别是在排序步骤进行多趟排序，引入判断机制和引入特定权值和有效权值。基于属性分析的重复记录检测算法是在基于传递闭包的重复记录检测算法的基础上，提出两个方面的改进，通过属性分析，并引入过滤机制。在保证正确率的同时，有效的提高了算法的效率，减少算法的运行时间。基于完全子图的重复记录检测算法是针对前两个算法中因为应用传递闭包而产生误识别的问题而提出的改进算法。算法的解决方法是将相似记录集视为一个完全子图，将合并相似重复记录的问题转换为在连通图中寻找完全子图。最后，论文通过实验验证，表明改进的算法取得了良好的效果。

关键词：重复记录检测；SNM；传递闭包；完全子图

Abstract

Approximately duplicate records' cleaning is important in the field of data cleaning. Duplicate records detection is the process of identifying multiple records that refer to one unique real-world entity or object. However, due to different data representations in different data sources or errors because of various reasons, determining whether two records are equal is not a simple arithmetic predicate. Based on the existing duplicate records identification algorithms SNM and MPN, this paper proposes three improved algorithms about duplicate record detection algorithm. The first one is based on the transitive closure which analyzes attributes and sorts the dataset multiple times to make duplicate records more clustered considering that the sliding window size w is hard to select in SNM algorithm. Secondly, it gives a special weight to each attribute based on its contribution in the comparison and introduces the concept of effective weight so that to make the comparison more accurate and finally it merge the duplicate records by the method of transitive closure. The second algorithm proposed in this paper is based on attributes analysis, it analyze the attributes and orders them by their weights, then to improve the efficiency of detection by introducing filtering mechanism based on the analysis mentioned above. The third algorithm is based on complete sub-graph. By analyzing the MPN algorithm, it is clear that transitive closure in the merge step will cause higher false-positive rate. Our improved method treats a similar dataset as a complete sub-graph, and therefore the problem of duplicate records detection is converted to finding complete sub-graphs from an association graph where the vertexes represent data records and the edges reflect the similarity between records. At the same time, our algorithm effectively avoids the redetection of some parts of an already detected sub-graph. Further more, another improved algorithm based on the assumption that any two complete sub-graphs only have one common vertex in the association graph is proposed, as any two duplicate record sets only contain one same record, because after analyzing we find that it is a common phenomenon. And finally, the experimental results show that these advanced algorithms solve problem well.

Key Words: Duplicate Records Detection; SNM; Transitive Closure; Complete Sub-Graph

目 录

Contents

第一章 绪论

1.1 课题目的及意义

随着信息技术领域的不断发展和信息化建设的不断深入，当面对海量数据的同时，现实中的数据集也变得越来越错综复杂。数据中不可避免的存在冗余数据、缺失数据、不确定数据和不一致数据等诸多情况，这样的数据统称为“脏数据”。据统计，一些具有代表性的大公司的数据错误率预期在1%-5%左右，个别公司可能更高[1]。据报道，美国商业公司每年在处理“脏数据”上的花费都在上亿美元[48]。普化永道会计事务所在纽约的研究也表明，75%的被调查公司存在因“脏数据”问题而造成经济损失的现象，只有35%的被调查公司对自己的数据充满信心[49]。根据“垃圾进，垃圾出”的原理，错误的数据不仅将影响从数据集中抽取模式的正确性和导出规则的准确性，使得系统产生错误的分析结果，导致错误的决策[20]。而且从效率上来说，还将导致昂贵的操作费用和漫长的响应时间。因此，如何将“脏数据”有效的转化成高质量的干净数据是需要解决的首要问题，这涉及到数据清洗技术。

数据清洗也称为数据清理(Data Cleaning，Data Cleansing 或 Data Scrubbing)，目的是检测数据中存在的错误数据和不一致数据，并将它们剔除或者改正，以提高数据的质量[2]。假如输入数据的质量很差，那么数据挖掘返回的结果多半也将令人失望。数据清洗作为数据仓库技术中的重要一环，其实施效果如何直接决定了进入数据仓库的数据的质量，而数据质量是影响数据挖掘成功与否的重要因素，因其将进而影响决策支持系统的正确分析。因为数据仓库需要频繁地从各式各样的数据源中进行装载和刷新，而这些数据中不可避免地存在很多异常、冗余和错误，这就要求进入数据仓库前对数据进行清洗[3]。通过清洗，可以对残缺的数据进行修复、对错误的数据进行纠正和对多余的数据进行清除，将不正确的数据格式转换为所要求的格式，从而达到数据类型相同化、数据格式一致化、数据信息精练化和数据存储集中化的效果[4]。

“脏数据”按其不同的表现形式可具体概括为不完整数据、相似重复数据和错误数据三种类型[5]。其中由于多数据源合并而造成的信息重复是最关键的问题，因此重复信息的检测和清除成为一个研究的热点[6,7]。

1.2 重复记录检测研究现状

1.2.1 国外研究现状

国外对数据清洗技术方面的研究最早出现在美国，是从对全美社会保险号的错误进行纠正开始的。美国信息业和商业的发展促进了这方面工作的相关研究。识别并消除数据集中的相似重复对象，也就是重复记录的检测与清洗是数据清洗技术的一项重要研究内容。

相似重复记录识别，也称字段匹配，即选用合适的算法检测出标识同一现实实体的重复记录。这是重复记录检测中的核心步骤。现已有的算法多事针对不同错误类型的此类算法。大致分为以下几类。一是基于字符的相似性度量，能够很好的处理印刷、字符排序上的错误。主要有编辑距离算法(edit distance)[12]，仿射距离算法(affine gap distance)[13]，Smith-Waterman算法[14]，Jaro距离算法(Jaro distance metric)[15]，和Q-gram算法[16, 17]。二是基于令牌的相似性度量，主要用来处理词语错位重排的问题，比如“Richard Smith”和“Smith，Richard”。主要有原子字符串算法(Atomic Strings)[18]和WHIRL算法[19]。三是语音相似性度量。基于字符和令牌的相似性度量都是主要针对基于字符串表示的数据集记录。但也有一种类型的字符串语音上相似但就字符而言并不相似。比如单词Kageonne与Cajun在语音上是相似的，尽管它们的字符表示相距甚远。语音相似性度量就是用来处理这类问题并进行字符串匹配。主要有探测法(Soundex)，纽约模式识别与智能系统(New York State Identification and Intelligence System)[22]，名字压缩算法(Oxford Name Compression Algorithm)[23]，音位算法(Metaphone Algorithm)[24]，双音位算法(Double Metaphone Algorithm)[25]。

聚类算法能够辅助相似度的计算，在相符重复记录的识别中得到很好的应用。主要算法有，基于DBSCAN算法的自适应的密度聚类算法[34]；空间局部密度聚类算法[35]；基于网格和密度的聚类算法[37,38]；基于蚁群的聚类算法，主要应用在多维数据空间中[39]；自适应的粒子群优化聚类算法[40]。

对于一组检测出的相似重复记录有两种处理的方法。一是清除，即把一条记录看成是正确的，而其他记录则是含有错误信息的重复记录；二是合并，即把每一条检测出的重复记录看成是数据源的一部分，对这些记录进行合并，产生一条具有更完整信息的新记录。常用的算法有近邻排序法(Sorted

Neighborhood Method)[41,42]，多趟近邻排序法(Multi-pass Sorted Neighborhood)[41,42]，Delphi 算法(Delphi Algorithm)[43]，优先队列算法(Priority Queue Strategy, PQS)[44]等。

1.2.2 国内研究现状

由于中西文本身的差异性，国外数据清洗技术不能完全适用于中文数据的清洗。国内对于数据清洗的研究较晚，并且直接针对中文的数据清洗研究的成果也不多。尽管在一些学术期刊及会议上也能看到一些有关这方面的文章，但直接针对数据清洗，特别是中文清洗的研究还很少。国内现在主要是在数据仓库、决策支持、数据挖掘等方面的研究中做了些简单的阐述，而对于商业性的数据清洗工作主要是针对各自的具体应用，理论性不强。银行、证券等对数据的准确性要求较高的行业，都针对自己的具体应用开发相应的数据清洗软件，但很少公布理论性的东西。中文数据清洗在理论研究上的欠缺，也使市场上几乎看不到关于中文数据清洗的软件和工具[26]。然而随着数据仓库、客户关系管理系统等在企业中的大量应用，必然要求高质量的数据集支持，同时也将带动对中文数据清洗技术的研究和提高数据质量方法的研究，以及中文数据清洗工具的开发。

中文字段匹配指的是基于中文数据的字段匹配，中文字段匹配方法主要包括以下三类[68]：

一是字符串匹配方法。字符串匹配方法主要分为五种：单个字符的匹配方法、汉语自动分词方法、特征词匹配方法、词法分析得到的字符串匹配方法和中文缩写的回归字段匹配方法。单个字符的匹配的主要思想是逐步抽取某字符串中的单个字符与另一个字符串中的所有字符进行逐一比较，并将匹配成功的字符个数记录下来进行相似度计算，从而判断相似性。汉语自动分词方法是利用汉语的分词技术对字符串进行分词处理，得到单个分词的字符串，再用分词字符串作为匹配单位，进行匹配。特征词匹配方法是指只使用能够代表字段语言的关键词进行匹配，而不考虑其他的字符匹配。词法分析字符串匹配方法是指，先在字段中查找出一些具有特殊标识的字或词，然后将其从字段中取出，用这些字符串作为匹配单位，并根据这些具有特殊标识的字或者词的重要程度进行权值设定，最后通过匹配结果与权值的乘积值来判断相似性。中文缩写的回归字段匹配方法使用了文本值字段的回归结构进行匹配操作。

二是拼音匹配方法。中文常会出现同音字的现象。为了增大匹配的几率，有些时候需要用匹配单位的字符拼音进行匹配。其目的是解决汉语中一音多字的问题，具有实际意义，可作为字符串匹配方法的一种辅助方法，提高匹配精确度。主要步骤是，首先得到一张汉字与拼音的对照表，其中每个汉字和其对应的拼音组成数据库的一条记录，然后，使用字符型匹配算法进行匹配，将分词或汉字变成对应的拼音表示。接着用拼音串和字符或者分词作为匹配单位进行匹配。最后对于匹配结果通过事先定好的规则或策略进行筛选和抉择。

主要的记录匹配算法有，基于 Q-gram 层次空间的检测算法，主要用于大数据量的相似重复记录检测[27]；PCM 重复记录检测算法，它不仅可以应用在英文字符集合中，在中文字符集中效果也很好[28]；基于 N-Gram 的检测算法，主要适用于检测常见的拼写错误造成的重复记录[29]；基于优先队列的改进算法[30]；文献[31]中提出的基于 Canopy 聚类技术[32]的聚类算法，融合倒排检索对重复记录进行聚类以减小计算量；基于整体相似性的序列聚类算法(global similarity clustering)以及基于局部相似性的序列聚类算法(local similarity clustering)[33]，该算法具有较快的处理速度，并能取得较好的聚类质量；结合空间索引结构树与网格密度聚类算法的基于网格-密度与空间划分树的聚类算法，它能够仅在耗费线性时间复杂度的情况下发现任意形状的树[36]。

1.3 本文主要研究内容与结构安排

本文共分为六章。内容如下

第一章为概述。主要介绍课题的研究意义和目的。以及国内外重复记录检测的研究现状。

第二章主要介绍重复记录检测的相关理论知识。包括数据挖掘过程介绍；数据清洗的定义、原理、方法、基本流程和相关应用工具；重复记录检测知识介绍以及字段匹配算法和重复记录清洗算法介绍。

第三章提出基于传递闭包的重复记录检测算法，它是基于 SNM 算法的改进算法，主要改进的方面有排序步骤，引入判断机制和引入特定权值和有效权值的概念，在合并步骤，通过传递闭包对不同窗口中检测出的重复记录进行合并。最后通过实验验证算法的有效性。

第四章提出基于属性分析的重复记录检测算法，它在基于上一章算法的基

础上，提出了两个方面的改进，通过属性分析，并引入过滤机制。在保证正确率的同时，有效的提高了算法的效率，减少算法的运行时间。最后通过实验验证算法的有效性。

第五章提出基于完全子图的重复记录检测算法，它是针对前两章算法中因为应用传递闭包而产生误识别的问题提出的改进算法。算法的解决方法是将相似记录集视为一个完全子图，将合并相似重复记录的问题转换为在连通图中寻找完全子图。最后通过实验对算法进行验证。

第六章总结了本文所做的研究工作，并展望下一步的工作。

第二章 重复记录检测相关理论介绍

2.1 数据挖掘过程介绍

2.1.1 数据挖掘介绍

在信息爆炸的时代，人们面对以指数级增长的数据信息，渴望能够去粗取精、去伪存真的能将浩瀚无垠的数据转换成知识的技术，同时在客观条件上计算机硬件技术稳定进步，数据库技术日趋成熟。在这样的大背景下数据挖掘应运而生。

数据挖掘是按照既定的业务目标从海量的数据中提取出潜在的，有效的并能够被人理解的模式的高级处理过程。它是一门交叉性学科，融合了数据库技术、人工智能、机器学习、模式识别、统计学和数据可视化等多个领域的技术和理论。在较浅的层次上，它利用数据库管理系统的查询、检索和报表功能，并结合多维分析、统计分析方法进行联机分析处理(OLAP)，得出可供决策参考的统计分析数据。从深层次上来说，数据挖掘则从数据库中发现隐含的、潜在的知识。OLAP的概念最早是由关系数据库之父E. F. Codd于1993年提出的[45]。OLAP和数据挖掘都是从数据库中抽取有用的信息，就决策支持的需要而言两者是相辅相成的，OLAP旨在简化和支持联机分析，而数据挖掘的目的是尽可能使这一过程自动化。

图 2-1 显示了数据库中知识发现的过程[46]。我们可以看出，数据挖掘是数据库中知识发现(knowledge discovery in database, KDD)的核心步骤。

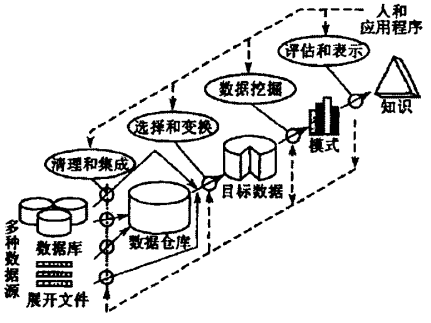


图 2-1 数据库中的知识发现

2.1.2 数据挖掘的方法

根据数据挖掘方法所属领域的不同大致可以分为以下几类。一是数学统计方

法。该方法一般首先建立一个数学模型或统计模型，然后根据模型提取出有关的知识。例如，可通过训练数据建立一个贝叶斯网，然后根据贝叶斯网的一些参数和联系权重提取出相关的知识；二是面向数据库的方法。随着数据库技术的发展日趋成熟，其中一些数据处理方法不断完善,在KDD中，利用现有的一些数据库技术和专门针对数据库的启发式方法，可以对数据库中的一些特征知识进行提取；三是机器学习方法。大多数机器学习的方法是通过采用人类的认识模型，从而模仿人类的学习方法并从数据中提取知识,机器学习经过多年的研究，已经取得了一些较令人满意的成果。同时还有诸如数据可视化技术，知识表示技术等等其他方法。

应用较为广泛的数据挖掘方法主要有以下几种

（1）决策树方法。利用信息论中的信息增益，寻找数据库中具有最大信息量的字段，并据此建立决策树的一个结点,再根据字段的不同取值来建立树的分支。在每个子分支中，建立树的下层结点和分支，重复此过程即可建立决策树。J. R. Quinlan 在1986年提出的ID3方法是最具影响力和代表性的决策树方法[67]，它对越大的数据库，处理效果越好。后人在ID3方法的基础上又提出了各种决策树方法。比如SLIQ算法，SPRINT算法、PUBLIC算法和RainForest算法等。

（2）粗糙集理论。该理论是由波兰华沙理工大学的Z. Pawlak教授于1982年提出的一种研究不完整、不确定的知识和数据的表达、学习、归纳的理论方法[47]。粗糙集理论的特点是无需提供所需处理的数据集合以外的任何先验信息。它可以用于分类，以及发现不准确数据或噪声数据的内在联系。在现实世界的数据中，往往有些类不能被可用的属性区分，那么此时就可以用粗糙集来近似地定义这些类。

（3）神经网络方法。神经网络由多个神经元按照某种方式相互连接而成，它靠网络状态对外部输入的信息进行动态响应来处理信息。能够完成分类、聚类和特征挖掘等多种数据挖掘任务。神经网络模型主要有：前馈式网络、反馈式网络和自组织网络。由于神经网络分类方法获取的模式隐含在网络结构中，而不是显示地表达为规则,因此不容易被人们理解。另外因为训练数据要进行多次扫描，训练时间较长。因此神经网络用于数据挖掘，要解决好降低训练时间和挖掘结果可理解性这两个问题。

2.1.3 数据挖掘的过程

数据挖掘的处理过程一般分为以下五个步骤。第一，问题定义。即了解相关领域的情况，熟悉背景知识，弄清用户需求。第二，数据提取。根据需求从数据库中提取相关的数据。第三，数据预处理。对前一阶段提取的数据进行加工，主要是检查数据的完整性及一致性，对其中丢失的数据进行填充，对噪音数据进行处理等。第四，知识提取。运用知识发现算法，从数据中提取用户所需要的知识，并用特定的方式对这些知识进行表示。第五，知识评估。将发现的知识以用户能够理解的方式进行呈现，再根据实际情况对知识发现过程中具体的处理阶段进行优化，直到满足用户的要求。

比如，某家电信公司的数据分析员想弄清为什么近来拖欠该公司电话费的客户越来越多，并且希望找出解决该问题的办法。那么，首先，该分析员需要收集公司内部和外部的大量数据，包括客户安装电话时间长短的数据，客户所安装电话开通服务的种类的数据，安装电话的客户身份特征等数据。然后，该分析员可利用抽样工具对过去几年的客户数据进行抽样，以便得到一个用于进行分析的样本集。其次，在挖掘工具的帮助下，该分析员注意到很多客户在一年后就不再按时交纳电话费这一现象呈现增长的趋势。将客户进行分类后发现，存在这种现象的多为安装私人电话的客户。应用挖掘工具的模型分析技术，分析员提取了影响客户按时交纳电话费的因素，并计算出它们的相关度，结果发现分析得出的模型解释度并不高。显然，这一结果并不理想。该分析员注意到在某些特定地区亏欠电话费的客户特别多。他重新运用挖掘工具进行分析，结果表明在这些地区缺少该公司的办事处，而且客户多为工薪阶层。在这一分析结果的基础上，为解决实际问题提供了两个办法：在这些地区适当增加办事处，或者派专人定期上门收费。最后，该分析员将分析结果和解决方案整理成书面报告，送交公司决策者。

2.2 数据清洗知识

2.2.1 数据清洗的定义

数据清洗技术主要应用于数据仓库、数据挖掘和数据质量管理三个方面，针对不同的方面，因为在不同的领域对数据清洗有不同的要求，所以对它的认识也不尽相同。目前为止，数据清洗还没有一个公认的定义，但其主要内容大致相同。

一般来说，只要是对解决数据质量问题有帮助的处理过程就被认为是数据清洗。

（1）在数据仓库领域，数据清洗被定义为清除错误和不一致数据，并解决对象识别问题,元组重复和数据孤立点等问题的过程[50]。数据清洗一般应用于几个数据库合并或者对多个数据源进行集成的过程中。代表同一个实体的记录在不同的数据源中的表示形式不同或被错误地表示，这样进行合并后的数据库中就会出现重复记录。数据清洗过程就是将这些重复记录进行识别并消除，也就是所说的合并/清除问题。

（2）在数据挖掘领域中，首先需要进行数据预处理以保证数据质量，这个过程就是数据清洗的过程。不同的数据挖掘系统针对特定的应用领域进行数据清洗。根据是否依赖具体业务领域的知识,数据清洗可分为特定领域数据清洗和领域无关数据清洗。特定领域数据清洗要求参与清洗过程的人员,必需掌握仅适用于特定领域的原则和规则[51]。

（3）在数据信息质量管理领域中，数据清洗是一个学术界和商业界都很感兴趣的问题。麻省理工学院一直致力于全面数据质量管理的研究。全面数据质量管理可以有效的解决整个信息业务过程中的数据质量问题和数据集成问题。在该领域中，没有对数据清洗过程进行直接定义。一般将其视为一个评价数据正确性并改善其质量的过程。

2.2.2 数据清洗的原理和方法

数据清洗的原理，是通过对“脏数据”产生的原因和存在的形式进行分析，利用已有的技术和方法清洗“脏数据”，将“脏数据”转换为满足数据质量或用户要求的数据，从而提高数据库中数据的质量。数据清洗主要是利用回溯的思想，从“脏数据”产生的源头开始分析，考察、分析数据流的过程，并总结出诸如数理统计，数据挖掘或预定义规则等方法，最后在数据集上应用这些策略和规则发现“脏数据”并清洗“脏数据”。

数据清洗的对象和方法因数据质量的不同而有所不同。文献[2]将数据质量问题分为 4 类：单数据源模式层问题，单数据源实例层问题，多数据源模式层问题和多数据源实例层问题，如图 2-2 所示。

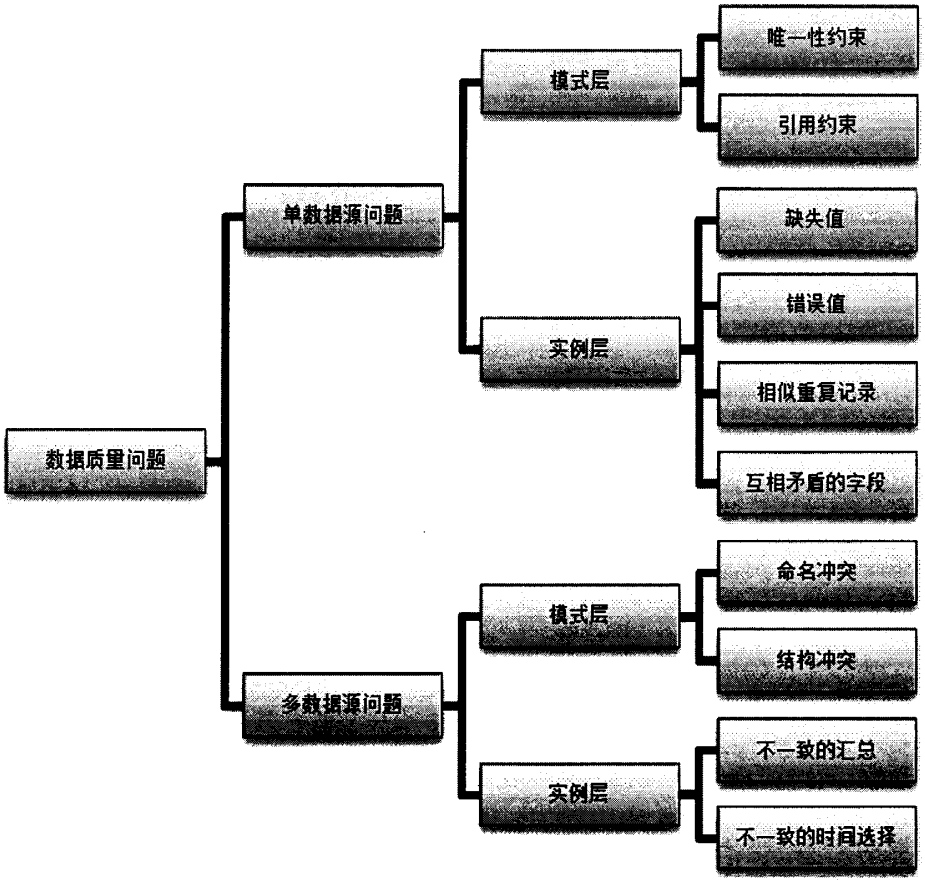


图 2-2 数据质量问题的分类

单数据源情形中出现的问题在多数据源情况下会变得更加严重。图中对多数据源问题没有列出在单数据源中已经出现的问题。模式层上出现的问题主要有，糟糕的模式设计、缺少完整性约束的定义以及多个数据源间不同的数据模型、命名和结构冲突等。命名冲突主要是同名异义和异名同义，同名异义是指同一名称代表不同的对象，异名同义是指不同的名称代表同一对象。结构冲突主要是由在不同的数据源中的不同对象表示引起的。例如，关键字冲突，依赖冲突和类型冲突等[52]。实例层上出现的问题主要有数据记录的错误和冗余、互相矛盾或者不一致的数据。

针对上述数据质量问题的分类，将数据清洗的方法分为模式层的数据清洗方法和实例层的数据清洗方法。

（1）模式层的数据清洗方法

模式层的数据质量问题可以通过设计程序，由程序自动发现问题。但由于模式设计与数据本身的理解和对数据所属领域的认识联系较为紧密，因此，即便发现了模式问题，也并不表示能够很容易的解决问题。一般需要在基于理解数据意义和应用领域的基础上才能做修改，从而很难应用计算机自动地对模式层问题进行修正。因此，通常情况下都需要运用手工清洗的方法[53]。现在，研究人员也在研究自动的模式层问题清洗工具，但还不成熟。

（2）实例层的数据清洗方法

实例层上数据问题的情形主要有：缺失值、拼写错误、缩写形式不同、自由格式的文本串、字段之间不相对应、词语的移位、相似重复记录、互相矛盾的记录和错误的引用[2]。

对于缺失值的情况，例如对于某人信息的电话号码属性，可能由于在数据输入时不知道该电话字段的值,因此在数据库中以一个无效值表示，或者此人没有电话号码，属性用空值表示。处理方法主要有：定义一个规则，存放在数据清洗库中,清洗工具能够根据这条规则判断出哪些是无效值。某些缺失值可以从本数据源或其他数据源推导出来，或者可用最大值，最小值，平均值，中间值更为复杂的概率统计函数来代替缺失的值，或者可通过人工操作输入一个可接受的值。

对于错误值的情况，处理方法主要有：用统计分析的方法识别可能的异常值或错误值；使用外部规则库或查找表检测和修正错误数据；使用不同属性间的约束检测和修正错误数据等。

对于数据不一致的情况，主要情形有数据形式、格式的不同或字段间不相对应以及系统硬件故障等原因产生的不一致数据。处理方法是在对不一致数据产生原因进行分析的基础上，应用多种格式函数、汇总分解函数、变换函数和标准库函数来实现清洗。

对于相似重复记录的情况，处理的主要方法是匹配与合并。首先,要选择一个合适的编辑距离函数。此外,记录的匹配过程非常耗时,如果采用原始的方法,对所有记录进行两两比较,以此来决定是否匹配的话,其计算的时间开销对于大容量的数据库来说是无法忍受的。重复记录的检测方法将在下一小节中进行介绍。

2.2.3 数据清洗的基本流程

数据清洗的流程如图 2-3 所示

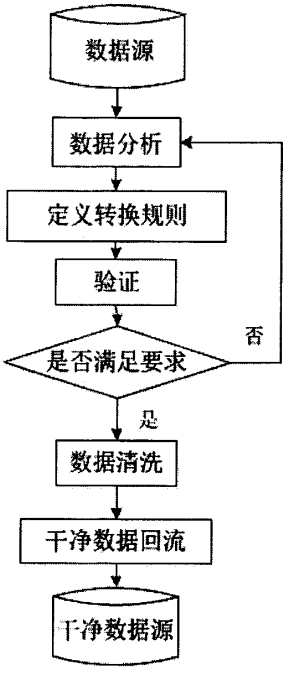


图 2-3 数据清洗流程

对于数据清洗的一般流程，首先进行数据分析，以此来检测数据中的错误和不一致等数据问题。第二，定义数据清洗转换规则。根据上一步分析得出的结果，即数据源中“脏数据”的程度定义相关的转换规则。第三，验证。对数据清洗转换规则的正确性和效率进行评估和验证，验证操作可在数据样本上进行。当不满足要求时，需要对转换规则和相关参数进行修改和调整，因此这往往是一个迭代的过程。第四，数据清洗。即清洗数据中存在的错误。根据“脏数据”的不同形式，执行定义并经过验证的数据清洗转换规则，解决不同的数据质量问题。第五，回流干净数据。即在数据清洗后，用干净的数据替代原来的“脏数据”。

2.2.4 数据清洗的工具

下面介绍几种数据清洗研究领域的清洗模型

（1）AJAX 模型[54]。该模型分离逻辑层和物理层，在逻辑层设计数据处理

流程，确定清洗过程需执行的数据转化操作。在物理层实现这些数据转化操作。分别通过如下五个过程，数据映射、匹配、聚集操作、数据合并以及数据视图显示进行数据清洗。该模型从数据输入开始检测，直到数据输出检测结束。

（2）Trillium 模型[55]。它是由 Harte Hanks Data Technologies 的 Trillium Software System 创建的应用于企业领域的数据清洗工具。主要处理包括名称、头衔、电话号码、身份证等常见数据，它可接受多种格式和编码的数据，具有平台无关性，并适用于多种环境。此模型主要用于的保险金融等行业。

（3）Potter’s Wheel 模型[56]。它是一个交互式系统框架。用户能够通过直观的图形化方式建立数据转化过程。用户的执行或者撤消等数据转化操作，能在屏幕上马上看到操作结果。后台进程以增量方式检测转化后的数据是否存在问题,若存在则及时报告给用户。在此过程中，用户无须书写复杂的程序就可以完成数据清洗任务，并且每步转化操作没有很长的延时。该系统具有较好的交互性。

2.3 重复记录检测

2.3.1 数据记录清洗简介

在构造数据仓库的过程中，需要从各种数据源中导入大量的数据。理想的情况下，对于现实世界中的一个实体，数据库或数据仓库中应该只有一条与之对应的记录，但在对多个数据源进行集成时，由于实际数据中可能存在多种形式的数据问题，导致数据库管理系统不能区分标识同一个实体的多条记录，而这些记录其实在逻辑上指向的是同一个现实世界的实体。在关系型数据库中，如果两条记录在所有属性上的值都完全相同，才认为它们是重复的。而所谓近似重复记录是指表现形式不同但语义相同的记录，即它们在某些属性上的值相等或足够相似。我们把同一实体对应的多条记录称之为相似重复记录[57]。

相似重复记录清洗是数据清洗过程中最关键的问题之一。它的目的是匹配、合并和清除那些冗余的，客观上映射同一实体但语义表示却存在差异的数据库记录[57]。重复记录清洗有许多别的提法[58]，如记录连接、语义集成、实例识别等。业界通常称其为合并-清除或匹配-合并。

重复记录检测可分为 5 个基本步骤[8,9]：

（1）数据预处理：主要是对数据的格式进行标准化处理，并清理一些较为明显的，可以初步识别的错误。

（2）缩小搜索空间：为了减少搜索空间，通常使用启发式的搜索算法。如在文献[9]中提出的分类近邻法，文献[10]中提出的增强型的笛卡尔积算法，这些算法都使计算在数量和有效性上得到大幅提高。

（3）相似重复记录识别：也称字段匹配，即选用合适的算法检测出标识同一现实实体的重复记录。这是重复记录检测中的核心步骤。在数据库中字符串数据的印刷错误通常是造成数据不匹配的常见来源，因此重复检测通常依赖于字符串比较技术。现已有的算法多是针对不同错误类型的此类算法。

上述描述的算法是用来匹配一个记录的各个字段。然而，在真实的生活情况中，记录往往包括多个字段，这使得重复检测问题更加复杂。用于多个字段的记录匹配方法，可以大致分为两个类别：一是依赖训练数据的方法，并基于此“学习”如何进行记录匹配。包括概率方法和有监督的机器学习。二是依赖领域知识或一般距离度量的记录匹配方法。包括使用声明性语言的匹配方法，以及为适应重复检测任务而设计的距离度量方法。

（4）相似重复记录清除：对于一组检测出的相似重复记录有两种处理的方法。一是清除，即把一条记录看成是正确的，而其他记录则是含有错误信息的重复记录；二是合并，即把每一条检测出的重复记录看成是数据源的一部分，对这些记录进行合并，产生一条具有更完整信息的新记录。目前的重复记录清除算法主要采用排序-合并的思想[21]，先将数据库中的记录进行排序，然后通过对临近记录进行比较来检测记录是否重复。

（5）验证：文献[11]指出重复记录清理的有效性主要通过以下两个指标进行度量。召回率(recall rate = tp /(tp + fn))，误识别率(false merge rate = fp /(tp + fp))，其中 tp 为正确识别出的重复记录数，fp 为错误识别出的重复记录数，fn 为未识别出的重复记录数。所以 tp+fp 为识别出的重复记录总数，tp+fn 为实际的重复记录数。即召回率为识别出的重复记录数占实际包含的重复记录数的比值。误识别率为错误识别的重复记录数占总共识别出的重复记录数的比值。

2.3.2 数据字段匹配算法

为了从数据集中检测并消除重复记录，首先要解决的问题就是如何判断两条记录是否重复。这就需要比较记录的各对应字段，计算其相似度，并在此基础上计算得到记录的相似度，如果两条记录的相似度超过了某一阈值，则认为两条记录是相似重复记录，反之不是。其中字段相似度值的计算，即字段匹配算法，是重复记录检测问题的核心。下面介绍几种主要的算法。

（1）字段匹配算法(field matching algorithm)

基本的字段匹配算法和递归的字段匹配算法是两种最为基础的字段匹配算法。

基本的字段匹配算法利用两条记录中顺序匹配的分词个数除以它们所有分词个数的平均值，由此得出的结果为两条记录的匹配度[18,59]。举个例子，有相似重复记录的实例如表 2-1 所示。

表 2-1 相似重复记录例子

记录条数	名字	地址	电话
A	Join Kow	Comput. Sci. & Eng. Dept., University of California, San Diego	888-1234
B	K. Jion	Department of Computer Science, Univ. Calif., San Diego	8881234

在上述例子中，删去两条记录的停止词后，记录 A 中有 6 个单词与记录 B 中的单词相匹配，为{Comput Sci University California San Diego}，A 中的分词个数为 8，B 中的分词个数为 7。因此这两条记录的匹配度为 6/[(7+8)/2]=0.8。基本字段匹配算法的缺点是不能处理非前缀的缩写情况，也不能对子字段的排序信息进行应用。

递归的字段匹配算法主要是利用字符串的嵌套结构。在一个字符串中，可以含有多种分割符，它们具有不同的优先级。首先采用优先级最高的分割符来划分字符串，对于划分出的子串，采用优先级次之的分割符来划分。如是重复，直到划分出的子串不可再分割为止。那么两个字符串的子串的匹配平均值即为它们的匹配值，其计算公式为：$match(A,B)=\frac{1}{|A|}\sum_{i=1,j=1}^{|A|,|B|}\max match(A_i,B_j)$，其中$A_i$，$B_j$是字符串 A，B 的子串，$|A|$，$|B|$分别为字符串 A，B 子串的个数。

递归匹配算法采用递归的方法，从上而下，计算字符串中每一对被分割出的子串的匹配值。若两个子串相同或一个是另一个的缩写，那么其匹配度为 1.0，否则为 0.0。递归的字段匹配算法优点是简单直观，并能处理子串顺序颠倒的匹配及缩写的匹配等情况。其缺点是时间复杂度较高，且匹配规则较复杂，效率不高。

（2）编辑距离算法(edit distance algorithm)

编辑距离由Levenshtein于1966年提出，故也称Levenshtein距离[60]。它通过计算两个字符串间相互转化所需要的最少编辑操作数量来衡量两个字符串的相似性。所需的操作数越少，说明两个字符串的相似性越高。传统的编辑距离算法包含三个操作：替换、插入和删除。两个字符串x和y间的编辑距离$D(x, y)$为把字符串 x 转化为字符串 y 所需采取的编辑操作的最小代价。计算方法如下式所示

$$D(i,j)=\begin{cases}0 & i=0\text{且}j=0\\ min\begin{cases}D(i-1,j-1)+C(x_i\to y_j)\\ D(i-1,j)+C(x_i\to\varepsilon)\\ D(i,j-1)+C(\epsilon\to y_j)\end{cases} & i>0\text{或}j>0\end{cases}$$

这是一个动态规划的方法，其中对于字符串x和y，有$x=x_1\cdots\cdots x_m$，$y=y_1\cdots\cdots y_n$，$C(x_i\to y_i)$，$C(x_i\to\varepsilon)$ 和 $C(\varepsilon\to y_i)$分别表示操作替换$(x_i\to y_i)$，删除$(x_i\to\varepsilon)$和插入$(\varepsilon\to y_i)$的代价，其中任何一致替换的代价都等于0。最后得出的编辑距离$D(m, n)$即为字符串 x 和 y 的相似度。

（3）Smith-Waterman 算法(Smith-Waterman algorithm)

Smith和Waterman在1981年提出了著名的Smith-Waterman算法[14]，它是一种动态编程的方法，最初用于在相关的DNA或蛋白质序列中发现最优对位。该算法通过迭代计算两个字符串所有可能的比对分值，再通过回溯寻找最优比对结果。对于给定的序列 $S = s_1\cdots\cdots s_i$ 和 $T = t_1\cdots\cdots t_j$，构造矩阵D [n+1, m+1]存放序列的比对结果，计算公式如下

$$D(i,0)=D(0,j)=0,\qquad D(i,j)=\max\begin{cases}0\\ D(i-1,j-1)+f(s_i,t_j)\\ max_{1\le x\le i-1}\{D(i-x,j)-W_x\}\\ max_{1\le y\le j-1}\{D(i,j-y)-W_y\}\end{cases}$$

其中，$1\le i\le n$，$1\le j\le m$，计分函数 $f(x,y)$ 表示字符 x 和 y 进行比较时

的分值，一般当 x 和 y 相同时，$f(x,y)$为正值，否则为 0 或负值。W_x为在序列 S 中添加一个长度为 x 的空位的罚分，Wy 为在序列 T 中添加一个长度为 y 的空位的罚分。在序列中任意连续的最长的间隔称为一个空位，每个空位中间隔的个数称为空位的长度，空位的引入是为了补偿因字符串变异而产生的插入或缺失，每引入一个空位，比对的分值都会有所降低。通常空位罚分由初始化空位的罚分值和空位延伸间隔的罚分值来决定。这样 $D(i,j)$即表示 $s_1\ldots\ldots s_i$ 和 $t_1\ldots\ldots t_j$ 的最优比对分值。

2.3.3 相似重复记录清洗算法

在匹配和识别重复记录后，需要进行数据的合并或删除操作，这就是相似重复记录清除。目前清除算法主要采用的是排序–合并的思想,即先将数据库中的记录进行排序，然后通过对相邻的记录进行匹配识别来判断记录是否重复。最可靠的方法是对任两条记录进行相互比较，但对于大数据集来说时间复杂度太高。目前常用的算法有近邻排序算法(Sorted Neighborhood Method, SNM)，多趟近邻排序算法(Multi-pass Sorted neighborhood, MPN)，优先队列算法(Priority Queue Strategy, PQS)等，这里重点介绍前两种。

（1）SNM，近邻排序算法

近邻排序算法的基本思想可以分为下面三步。

第一，创建排序关键字。抽取记录属性的一个子集序列或者属性值的子串，为数据集中的每一条记录计算出一个键值。

第二，排序。根据选取的排序关键字对数据集进行排序。尽可能使潜在的重复记录调整到一个邻近的区域内，从而能够将要进行记录匹配的对象限定在一个范围内。

第三，合并。在排序后的数据集上滑动一个固定大小的窗口，数据集中每条记录仅与窗口内所有记录进行比较。窗口的大小为 w，即窗口内有 w 条记录，则每条新进入窗口的记录都要与先前在窗口内的后 w–1 条记录进行比较，来检测是否存在重复记录，每次滑动窗口，原窗口中的第一条记录滑出窗口，原窗口中最后一条记录的下一条记录移入窗口，将此 w 条记录作为新一轮比较对象，直到记录集的最后一条记录。过程如图 2-4 所示。

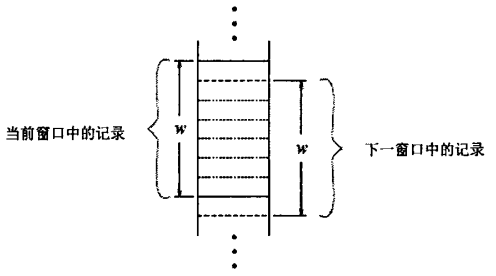


图 2-4 SNM 算法过程

SNM方法存在一些缺陷[61]。一是，对排序关键字的依赖太大。重复记录检测算法的精度很大程度上依赖于排序结果，但是如何选取关键字并没有统一的规则。如果选取不当，可能造成漏配。比如可能使本来互为重复记录的数据在排序后物理位置相距很远，无法同时落在同一个滑动窗口内中，因此没有机会被识别为重复记录。二是，滑动窗口大小 w 难以选取。若 w 较大，会增加本无必要的比较次数，增加复杂性；若 w 过小则可能产生漏配。

（2）MPN，多趟近邻排序算法

针对SNM算法的缺陷，Hernandez等人提出了MPN算法，该算法的基本思想是独立地执行多趟SNM算法，每趟创建不同的排序关键字并使用相对较小的滑动窗口。采用基于规则的知识库，生成一个等价原理，作为合并记录的判断标准，将每趟扫描识别出的重复记录进行合并，在合并时假定记录的重复具有传递性，即计算其传递闭包(transitive closure)[62]。所谓传递闭包，是指若记录 *R1* 与 *R2* 互为重复记录，*R2* 与 *R3* 互为重复记录，则 *R1* 与 *R3* 也互为重复记录。通过将每趟扫描识别出的重复记录通过传递闭包的方法进行合并，可以得到较完全的重复记录集合。

该算法也有一定的不足：传递闭包容易引起误识别，例如 *R1* 和 *R2* 相似，*R2* 和 *R3* 相似，但 *R1* 和 *R3* 可能差别很大，但是闭包传递却认为 *R1* 和 *R3* 是重复记录。从而导致误识别。

2.4 本章小结

本章首先对数据挖掘进行介绍，主要介绍了数据挖掘的方法和过程。接下来，对数据挖掘中的重要环节，数据清洗进行介绍。主要内容包括数据清洗的定义、原理和方法、基本流程和主要的清洗工具。最后介绍数据清洗领域中的

研究热点重复记录检测。包括重复记录检测中常用的字段匹配算法和相似重复记录清除算法。下面我们将在 SNM 和 MPN 算法的基础上，提出三个改进的相似记录清除算法。

第三章 基于传递闭包的重复记录检测算法

3.1 算法描述

本章提出基于传递闭包的重复记录检测算法。在基于 SNM 算法的基础上提出如下两点改进。首先，考虑到在 SNM 算法中滑动窗口难以选取的问题，我们先进行属性分析并根据分析结果对数据集进行多次排序，使相似重复记录在物理位置上相近，这些重复记录就能尽可能的落在同一个滑动窗口中，这样仅需在排好序的记录上滑动一个固定大小的较小的窗口便可，提高了记录匹配的正确率和效率。第二，当进行记录匹配时，基于属性在记录比较中的贡献给每个属性赋予一个特殊的权值并引入了有效权值的概念。使记录的相似度值更加准确可靠。该算法分为三个步骤，如图 3-1 所示。

```
Input: data set R, k, m, w, LowThreshold
Output: duplicate records
1.  data preprocessing
2.  select attributes to constitute sort set R_k and mode set R_m
3.  do
4.      sort R with attributes in R_k
5.  until (time of sort == k || SortGE(R,m,w,LowThreshold) == true)
6.  detect duplicate records
```

图 3-1 算法步骤

算法中，第 1 和第 2 行是步骤一，属性分析。第 3 到第 5 行是步骤二，数据排序。第 6 行是步骤三，重复记录检测。每一个步骤将在下文详细介绍。算法的流程图如图 3-2 所示。

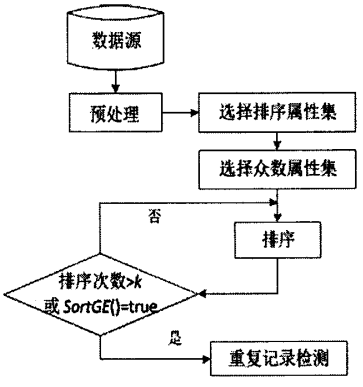


图 3-2 算法流程图

3.1.1 属性预处理

首先，算法对属性进行预处理，将空格和标点符号作为分隔符对属性的字符串进行分割，在自然分割后对属于相同语言的字段进行比较。然后，选出两种类型的属性，并构成属性集合。一类是排序属性集，另一类是众数属性集。如上文中提到的，排序属性集合中的属性作为排序关键字对匹配操作的效率和正确率有直接的影响。如果属性选择得不合适，算法的执行时间可能很长，效率和匹配的正确率都会受到影响。

排序属性集：排序属性集合中的属性被用来作为关键字对数据集合进行排序，因此“重要”的属性应该被纳入进来，因为它们对整条记录的贡献率和影响较大。在通常情况下，“重要”的属性的值域较大。举个反例，比如“性别”这个属性的值域只有男和女，值域范围很小，若用该属性作为关键字来排序，显然无法保证相似重复的记录在物理位置上相近。根据这个原则，对属性的值域进行统计，并从高到低排序，选择前 k 个属性组成属性集 R_k，如此，对于排序后的记录集，便可使用一个较小的滑动窗口完成匹配比较的步骤。

众数属性集：众数属性集合中的属性的特点是，这些属性的属性值取值个数尽可能少，即值域尽可能小，该属性集将用在稍后介绍的函数 *SortGE* 中，根据这一原则，选择 m 个这样的属性，组成属性集 R_m

3.1.2 数据排序

数据排序是算法的第二步，依次用属性集 R_k 中的属性对数据集合进行多次排序，并在每次排序后调用函数 *SortGE* 作为判断机制来提高算法的效率，函数 *SortGE* 用来检测当前的排序状况是否足够好，若当前排序足够好，则不再进行下一轮排序。因此，排序过程或者执行 k 次，或者当函数 *SortGE* 返回 true 时停止。

函数 *SortGE*：该函数如图 3-3 所示。函数 *SortGE* 中，第 1 行，算法根据属性集 R_m 取 m 个 w 大小的窗口快照。每个窗口快照的起始位置取众数属性集 R_m 中 m 个属性里每个属性值出现次数最多的属性值。例如，属性集 R_m 中存在一个属性为“专业”，该属性的所有取值中“计算机”出现的次数最多，那么就将“计算机”作为其中一个窗口快照的起始位置，即选择要进行清洗的记录中，属性为“专业”且取值为“计算机”的第一条记录作为窗口快照的起始

位置。第 2 行到第 7 行，函数在每个窗口快照中对记录进行两两比较并计算相似度。如果任意两条记录的相似度都大于给定的阈值 *LT*,说明该窗口中的排序足够好。函数 *RecordMatch* 将在图 3-5 中详细说明。如果对于 *m* 个窗口快照中的排序都足够好，那么函数返回 true，否则返回 false

```
Input: sorted data set R, m, w, LT
Output: true/false

1. take m window snapshots with size of w
2.   for i = 1 to m do begin
3.         for j = 1 to w-1 do
4.               for k = j+1 to w do
5.                     if (! RecordMatch(Rj, Rk, LT))
6.                           return false;
7.   end;
8.   return true;
```

图 3-3 函数 *SortGE*

3.1.3 重复记录检测

为了检测并消除数据集中的重复记录，首先应解决如何确定两条记录是否是重复记录。因此，需要对记录中对应的属性进行比较并计算相似度。如果两条记录的相似度值超过给定的阈值，这两天记录就被认为是相似的，否者，他们就不相似。计算属性的相似度，即字段匹配问题，是该问题的核心。

字段匹配：算法中的字段匹配算法采用 Smith-Waterman 算法[14]的核心思想，涉及到的定义如下

记分函数：设 Σ 为序列中可能出现的字符的集合，记分函数 $f(x, y)$表示 x 和 y 进行比较时的分值，体现字符 x 和 y 的相似的程度。一般情况下，若 x 和 y 相似时 $f(x,y)$ 为正值,否则为零或负值。

空位：在单个序列中任意连续的尽可能长的间隔称为一个空位，在空位中的间隔('-')个数称为空位的长度，用 spaces 来表示，序列中空位的个数用 gaps 来表示。

空位罚分：空位的引入是为了补偿由于字符串变异而产生的插入或缺失，每引入一个空位，比对的分值都会有所降低，通常情况下，空位罚分函数可表示为：$Penalty = W_g + W_s * spaces$，其中 W_g 表示初始化一个空位的罚分，W_s 表示空

位延伸一个间隔的罚分。

对于两条序列 S 和 T 的全局比对可以用序列 S' 和 T' 来表示，S' 和 T' 分别是在 S 和 T 中加入一些空位使这两条序列长度相等而得到的序列。将序列 S' 和 T' 相应的位置进行一一比较，其分值score可用如下公式来表示

$$score = \sum_{i=1}^{l} f(s_i, t_i) + W_g \times gaps + W_s \times spaces$$

其中，l 为序列 S' 与 T' 的长度，s_i 和 t_i 分别为序列 S' 和 T' 的第 i 个字符。

对于两个比对属性序列 $S = s_1 \ldots\ldots s_n$ 和 $T = t_1 \ldots\ldots t_m$，他们对应的长度分别为 n 和 m，根据动态规划的方法，我们构造一个大小为 $(n+1)*(m+1)$ 的矩阵 D 来存放可能的匹配分值，函数命名为函数 *Sim*，具体过程如图 3-4 所示。

```
Input: two attribute sequences S, T
Output: match score

1. Initialize matrix D[p+1][q+1] with 0
2. for i = 1 to p do begin
3.       for j = 1 to q do
4.            if(Si == Tj)
5.                 D[i][j] = D[i-1][j-1]+f(Si,Tj);
6.            for x = 1 to i-1do
7.                 row = max(D[i-x][j]-Wx);
8.            for y = 1 to j-1 do
9.              column = max(D[i][j-y]-Wy);
10.         D[i][j] = max(D[i][j], row, column);
11. end;
12. return D[p][q]
```

图 3-4 函数 *Sim*

函数 *Sim* 中，第 1 行，算法首先初始化矩阵 D 为 0。2 到 11 行，该循环是为矩阵 D 赋值的过程。4 到 5 行，函数 $f(x, y)$代表 x 与 y 的匹配结果，反映字符 x 与 y 的相似度水平。如果序列 S 和 T 中两个对应的字符是相似的，矩阵的值按第 5 行所示进行计算。第 7 行中，W_x 为在序列 S 中添加一个长度为 x 的空位的罚分。第 9 行中，W_y 为在序列 T 中添加一个长度为 y 的空位的罚分。第 10 行中，计算 $D[i][j]$的值，它是第 5、7 和 9 行中计算出来结果的最大值。最后，返回 $D[p][q]$的值为两个属性字段的匹配分值，即它们相似度的结果。这个数值是在 0 和 1 之间的值。越接近 1 说明两字段的匹配值越高，两字段越

相似。

记录匹配：每条记录有许多不同的字段属性组成。在进行字段匹配操作后，我们需要进行记录匹配。对于所有的记录，不同属性字段的贡献是不同的。参考文献[63]中，给每个属性 C_i 一个特定的权值 W_i。这样可以通过不同的权值计算出相似度值。*Null* 值的属性也应被考虑进来。为了消除属性值缺失造成的影响，在此引入有效权值的概念。假设 *R1* 和 *R2* 代表两条记录，其中各有 *n* 个属性字段，这两条记录的相似度可通过如下公式进行计算

$$\mathrm{Tsim(R1,R2)} = \sum_{i=1}^{n} Valid[i] * W_i * Sim(R1_i, R2_i)$$

其中，$W1+W2+\ldots+Wn=1$，$0<Wi<1$。函数 *Sim* 如图 3-4 中所述。记录匹配算法 *RecordMatch* 如图 3-5 所示。

```
Input: records R1, R2, LowThreshold
Output:   true/false

1.  for i  = 1 to n do
2.  begin
3.     if ((R1.Field[i] == NULL && R2.Field[i]!= NULL)||
            (R1.Field[i] != NULL && R2.Field[i] == NULL))
4.        Valid[i] == 0;
5.     else
6.        Valid[i] == 1;
7.  end;
8.  for i = 1 to n do
9.  begin
10.    validWeightMatch +=
                Valid[i]*weight[i]*Sim(R1.Field[i], R2.Field[i]);
11. end;
12. if (validWeightMatch >= LowThreshold)
13.       return true;
14. else
15.       return fasle;
```

图 3-5 函数 *RecordMatch*

对所有的属性字段 *i*，只有当两条记录中的对应的属性都为空或都非空时，该属性字段的比较结果才有效，并置 *Valid*[*i*]为 1.否者 *Valid*[*i*]为 0.从算法第 1 行到第 7 行，对数组 *Valid* 的值进行填充，使两个属性字段在同时为空或同时非空时进行比较，这样还能够保证空值属性的作用。8 到 11 行，算法按照记录

匹配中提到的公式计算记录 *R1* 和 *R2* 的相似度值。最后，12 到 15 行，如果比较结果 *validWeightMatch* 大于或等于阈值 *LowThreshold*，函数返回 *true*，即表示相似。否则返回 *false*，表示不相似。

最后采用传递闭包的方式对不同窗口中检测出的相似重复记录进行合并操作。

3.2 算法复杂度分析

算法的执行时间 *T* 由三部分组成：步骤一属性预处理的时间复杂度用 *T1* 表示，步骤二多次排序以及排序结果的验证的时间复杂度用 *T2* 表示，步骤三重复记录检测的时间复杂度用 *T3* 表示。步骤一仅需对数据集进行一次排序以确定数据集 R_k 和 R_m。所以 $T1=O(N)$，其中 *N* 是数据集中所有记录的总数。步骤二，算法进行多次排序并在排序后调用函数 *SortGE*，因为排序的次数不大于 *k*，因此 *T2* 不大于 *kNlogN+kmw2*，其中 *m* 是窗口快照的个数，*w* 是滑动窗口的大小，并且在通常情况下 *m* 和 *w* 都比 *N* 小。因此 $T2= O(NlogN)$。步骤三，记录匹配的时间复杂度是 $N/w*w^2*n$，其中 *n* 是每条记录中属性字段的个数，若 *n* 小于 *logN*，*T3* 为 *O(N)*，若 *n* 大于 *logN*，则 *T3* 为 *O(NlogN)*。综上，该算法的时间复杂度 $T=O(NlogN)$.

3.3 实验及结果分析

我们通过计算召回率(recall rate)和误识别率(false merge rate)来验证改进后的算法。召回率指被重复记录检测算法正确识别出的重复记录占数据集实际包含的重复记录的百分比。

假设我们有 8 条记录，*A1*，*A2*，*B1*，*B2*，*B3*，*B4*，*C1*，*C2*，其中{*A1*，*A2*}，{*B1*，*B2*，*B3*，*B4*}和{*C1*，*C2*}分别为记录 *A*、*B* 和 *C* 的重复记录。通过一个检测算法识别出{*A1*，*A2*，*C1*}，{*B1*，*B2*，*B3*}和{*C1*，*C2*}为重复记录，则召回率=7/8=87.5%，

误识别率为被重复记录检测算法错误的识别为重复记录的数目占被算法识别为重复记录总数的百分比。误识别率越低，表明算法结果的可信度越高。

在上面的例子中，检测算法错误的将 *C1* 识别为一条重复记录，则误识别率=1/8=12.5%

对算法进行验证的实验均在下述环境中完成，2.83Hz Intel Core i3 处理器，

3GB 内存，Windows XP professional SP3 操作系统，算法用 VC 6.0 实现。实验用三组数据集进行检验，分别包含 1000,5000 和 10000 条数据，每条数据包含 16 个属性字段，代表个人及家庭的信息，由 Pudjijono 提供的方法产生并可模仿真实世界的数据[64]。实验中，我们选取 5 个属性组成排序属性集，再选取 5 个属性组成众数属性集。第一组实验选取固定大小的滑动窗口和相似度阈值对召回率和误识别率进行计算。设定滑动窗口 w=6，相似度阈值 *LowThreshold* =0.65.实验结果如表 3-1 所示

表 3-1 使用算法 *RecordMatch* 计算不同数据集的召回率和误识别率

数据集	召回率	误识别率
1000	91.8%	0%
5000	98.3%	0%
10000	94.5%	0%

由表 3-1 可见，该算法具有较高的稳定性和可靠性。当数据集增大时，误识别率很低，始终为 0%，并且召回率保持在较高的水平。同时，对于不同的数据集，召回率和误识别率没有出现较大的波动。

图 3-6 为不同大小的滑动窗口下的召回率，其中相似度阈值 *LowThreshold* 为 0.65。从图中可以看出，随着滑动窗口的增大，召回率也会随之增大，算法能够识别出更多的重复记录，准确率得到提高

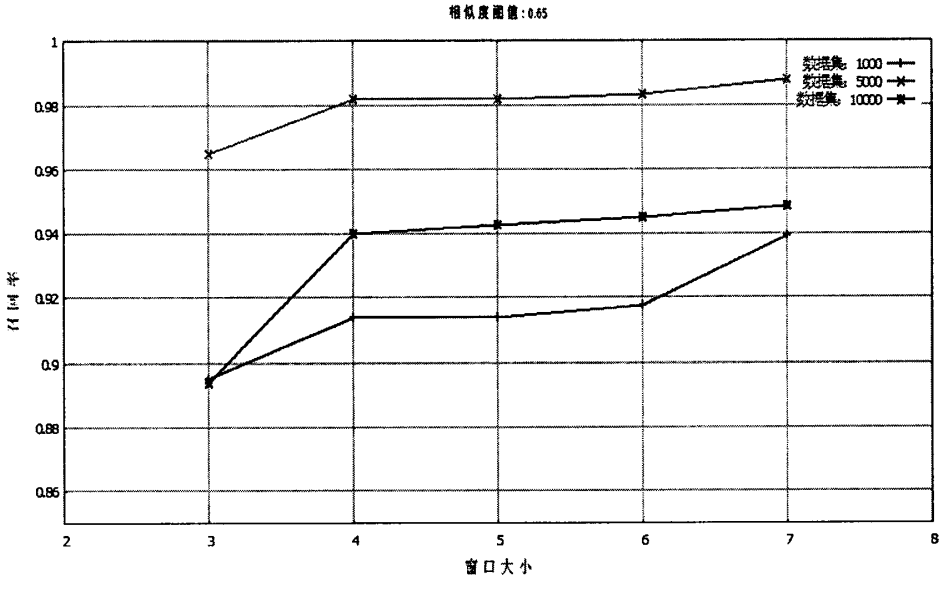


图 3-6 不同窗口大小(w)的召回率(recall rate)

图 3-7 为不同相似度阈值下的误识别率，其中窗口大小 w 固定为 6。从图中可以看出随着相似度阈值的增大，误识别的重复记录数目迅速下降，对于所有的数据集最后都降为 0。

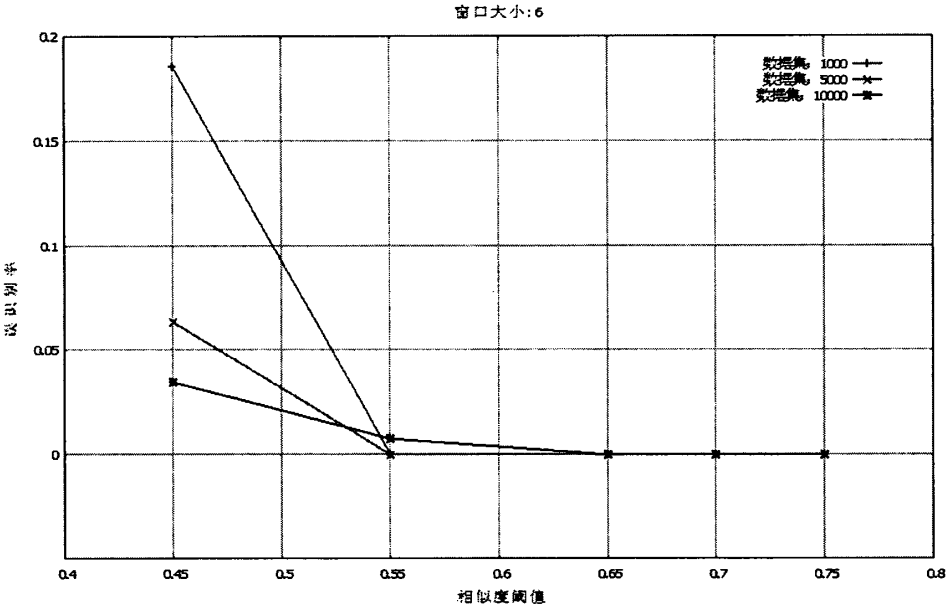


图 3-7 不同相似度阈值(*LowThreshold*)的误识别率(False Merge Rate)

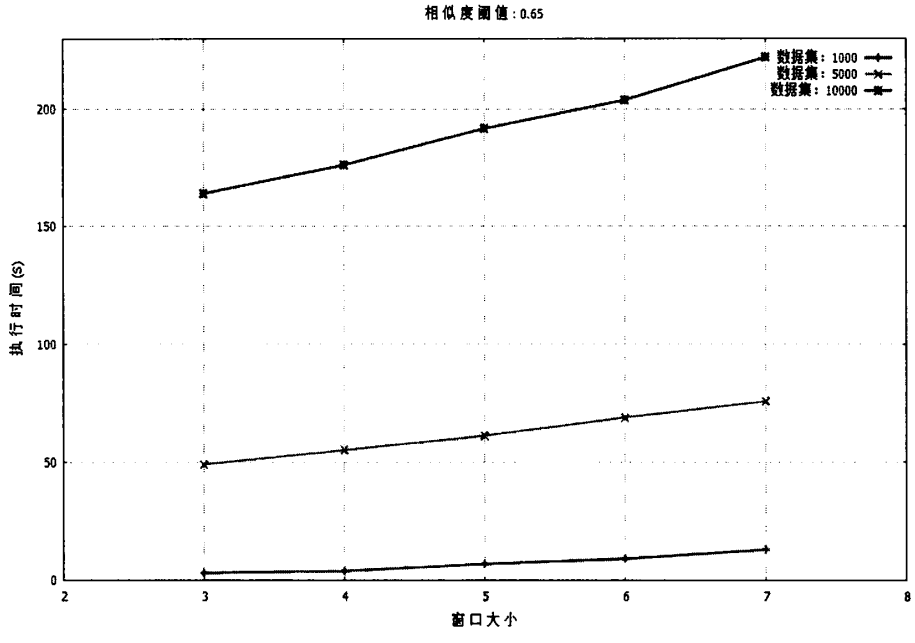


图 3-8 不同窗口大小(w)的算法执行时间(Execution Time)

图 3-8 为不同大小的滑动窗口下的算法执行时间，其中相似度阈值

LowThreshold 为 0.65。从图中可以看出，随着窗口的增大，执行时间的增长率相对平缓，算法的执行效率较为稳定。

3.4 本章小结

本章提出了基于 SNM 算法的改进算法，在排序步骤通过多趟排序使相似重复记录在物理位置上尽可能相近；通过引入判断机制，适时中止多趟排序的步骤，减少算法的复杂度；在记录匹配步骤，引入特定权值和有效权值的概念，提高记录匹配的准确率；最后在合并步骤，通过传递闭包对不同窗口中检测出的重复记录进行合并。

下一章节中，将提出基于本章算法的改进算法，主要针对在保证算法准确率的基础上，提高算法效率，减少算法复杂度。

第四章 基于属性分析的重复记录检测算法

4.1 算法描述

本章提出的算法是在基于上一章算法的基础上，提出两个方面的改进，算法的基本流程如图 3-1 所示，步骤一，属性分析，与步骤二，数据排序和上一章节中的算法相同。

在进行属性分析和数据排序的操作后，数据集合按照相似性聚合的原则排好序，接下来进行属性匹配的操作。在进行字段匹配操作后，进行记录匹配的操作。

针对上一章算法中的记录匹配算法，即图 3-5 所示的函数 *RecordMatch*，我们发现记录匹配步骤耗费太多的时间，当两条记录进行比较时，若在中间某一时刻两条记录的相似度值已经超过阈值，那么就没有再继续比较的必要了，相反如果相似度太低，已得到的相似度值即使加上剩下所有的权重值也小于相似度阈值，则也不必再进行比较，这样可以有效的减少算法运行时间。基于此，对原算法进行改进，将改进后算法命名为 *RecordMatch*$^+$，如图 4-1 所示。

```
Input: records R1, R2, LowThreshold
Output:   true/false

1.  for i  = 1 to n do begin
2.       if ((R1.Field[i] == null && R2.Field[i] != null) || (R1.Field[i] != null && R2.Field[i] == null))
3.            Valid[i] == 0;
4.       else
5.            Valid[i] == 1;
6.  end;
7.  remainWeight = 1.0;
8.  for i = 1 to n do begin
9.       validWeightMatch += Valid[i]*weight[i]*Sim(R1.Field[i], R2.Field[i]);
10.      remainWeight -= weight[i];
11.      if (validWeightMatch >= LowThreshold)
12.           return true ;
13.      else if ((validWeightMatch + remainWeight) <
                          LowThreshold)
14.           return false;
15. end:
```

图 4-1 函数 *RecordMatch*$^+$

算法 1 到 6 行和 *RecordMatch* 相同，对所有的属性字段 *i*，只有当两条记录中的对应的属性都为空或都非空时，该属性字段的比较结果才有效，并置 *Valid*[*i*] 为 1.否者 *Valid*[*i*]为 0。首先对数组 *Valid* 的值进行填充，使两个属性字段在同

时为空或同时非空时进行比较。第 8 到第 15 行，根据记录匹配中提到的公式计算记录 *R1* 和 *R2* 的相似度值。第 7 行，将变量 *remainWeight* 初始化为 1.0，它代表还未进行比较的属性字段的权重的和。第 10 行，在比较过程中，动态改变 *remainWeight* 的值，使其始终记录未比较属性的权重值。第 11 行和 12 行，*validWeightMatch* 表示当前比较的结果值，若该值大于阈值，表示两条记录相似，则此时比较终止并返回 *true*。否则，在第 13 行和 14 行，当前比较的结果值加上剩下所有的权重值仍比阈值小，则无需再继续比较，算法终止并返回 *false*

4.1.1 算法改进说明

本章的算法在上一章节算法的基础上，提出了两点改进

（1）在进行属性匹配时，按照每个属性的“贡献率”和“重要程度”给属性赋予权值，并按照相应权值从高到低，对属性进行匹配操作。这样做有助于过滤机制的引入和应用，并且使算法在增加一次线性时间复杂度的条件下，为每条记录的匹配都大幅减少了比较时间，提高了效率。

属性权值的确定方法主要采用基于粗糙集的方法[69]。首先对属性进行离散化处理，连续属性离散化的方法有很多，如领域知识法、动态层次聚类法、布尔推理法等等，视具体情况，确定应采用哪种方法及多大的离散粒度。经过离散化后得到粗糙集知识表达系统 S，可表示为$S = (U, R, V, f)$

U 表示实例库中所有实例的集合，R 表示实例库中实例特征属性的集合，$R=C\cup D$，C和D分别为条件属性和结果属性，V表示属性值的集合，$f : U \times R \rightarrow V$ 是信息函数，它指定U中每个实例的属性值，对于$\forall x\in U$，$a\in R$，存在 $f(x, a) \in Va$

$$IND(B) = \{(x, y) \in U^2 : \forall a \in B, f(x, a) = f(y, a)\}$$

表示对于属性集$B\in R$，B 的不可分辨关系，它描述的是两行数据对应属性集B的值域具有不可分辨的特性。IND(B)满足自反性、对称性和传递性。关系IND(B)构成了U的一个划分，用U/IND(B)表示，可简记为U/B

属性的约简，指对于给定的知识表达系统$S = (U, R, V, f)$，$a\in R$，如果IND(R)=IND(R-{a})，则 a 在 R 中是可以被约去的属性。

$$B\_(X) = \cup\{E_i \mid E_i \in U / IND(B) \wedge E_i \in X\}$$

表示X的下近似集，其中X⊆U，下近似集B_(X)是由哪些根据知识R判定肯定属于X的U中元素组成的集合。

$$POS_P(Q) = \sum_{X \in U/Q} P\_(X)$$

表示属性P相对于属性Q的正域，P, Q⊆R，Q的P正域是指U中所有根据分类U/IND(P)的信息可以准确的划分到关系Q的等价类中的对象集合。

对于属性 a_k，$k = 1 \ldots m, m$ 为条件属性的个数。属性 a_k 的重要度Weight(a_k)可由下式计算

$$Weight(a_k) = \frac{card(POS_c(D)) - card(POS_{c-\{a_k\}}(D))}{card(U)}$$

card(X)表示集合 X 的基，对于属性 $P, Q \in R$, 属性 P 相对于属性 Q 的正域表示为$POS_P(Q)$，它是指根据分类信息可以准确的划分到关系 Q 的等价类中的对象集合。上述公式中Weight(a_k)∈[0,1]，Weight(a_k)越大，表明该属性越重要。

若Weight(a_k) = 0时，说明a_k为冗余属性，应当将其约去，然后再对属性约简的决策表按照上述的公式进行计算，直到所有的Weight(a_k) > 0 为止。最后用规范化处理得到各属性的权重值，具体如下所示

$$W(a_k) = Weight(a_k) / \sum_{a_k \in C} Weight(a_k)$$

（2）引入过滤机制。对每两条相互比较的记录，计算当前已比较过的属性的相似度值，以及剩下未进行比较的属性的权重值的和。若当前已经过比较的属性的记录相似度值大于相似度阈值，则可判定两条记录相似，中止比较步骤；若当前的相似度值加上还未进行比较的属性权重值仍小于相似度阈值，说明即便剩下未比较的属性全都相同，这两条记录也不相似，可判定两条记录不相似，中止比较步骤。如此，可极大的节约比较时间。同时算法的第一点改进，即先比较权重大的属性，可有效的加快匹配进程。

同时，我们通过数学公式验证，改进算法能够与上一章中的属性匹配步骤得到相同的比较结果。说明算法在提高效率的同时，保持良好的准确率。

4.1.2 公式证明

下面对函数*RecordMatch*$^+$与函数*RecordMatch*将得到相同的比较结果进行证明。

引理1：对任两条记录*R1*和*R2*，若*RecordMatch*（*R1*, *R2*）= *true*，那么*RecordMatch*$^+$（*R1*, *R2*）= *true*，反之亦然。

证明：（1）如果 *RecordMatch*（*R1*, *R2*）= *true*，存在变量 k 使

$$\sum_{i=1}^{k} Valid[i] * W_i * Sim(R1_i, R2_j) \geq LowThreshold,$$

其中$1 \leq k \leq n$，所有的属性按照权重的大小从高到低进行排序，并且根据属性的重要程度进行计算。当变量i的值等于k时函数*RecordMatch*$^+$亦返回*true*如图13中12行所示。

（2）如果*RecordMatch*$^+$（*R1*, *R2*）= *true*，存在变量 k 使

$$\sum_{i=1}^{k} Valid[i] * W_i * Sim(R1_i, R2_j) \geq LowThreshold,$$

其中$1 \leq k \leq n$，因为$\sum_{i=k+1}^{n} Valid[i] * W_i * Sim(R1_i, R2_j)$总是大于或等于0，我们可以确定的得到$\sum_{i=1}^{n} Valid[i] * W_i * Sim(R1_i, R2_j) \geq LowThreshold$，函数*RecordMatch*返回*true*。

引理2：对任两条记录 *R1* 和 *R2*，如果*RecordMatch*（*R1*, *R2*）= *false*，那么*RecordMatch*$^+$（*R1*, *R2*）=*false*，反之亦然。

证明：（1）如果*RecordMatch*（*R1*, *R2*）= *false*，说明

$$\sum_{i=1}^{n} Valid[i] * W_i * Sim(R1_i, R2_j) < LowThreshold,$$

此时，当进行比较时函数*RecordMatch*$^+$ 将返回 *false* 如图13中14行所示。

（2）如果*RecordMatch*（*R1*, *R2*）$^+$ = *false*，有两种可能。或者存在变量 k 使

$$\sum_{i=1}^{k} Valid[i] * W_i * Sim(R1_i, R2_j) + \sum_{i=k+1}^{n} W_i < LowThreshold,$$

其中$k < n$，所有的属性字段都按照权重值排序，并按照属性的重要程度进行计算。

或者

$$\sum_{i=1}^{n} Valid[i] * W_i * Sim(R1_i, R2_j) < LowThreshold$$

对于第二种情况，显然函数 *RecordMatch* 会返回 *false*

对于第一种情况，因为$\sum_{i=k+1}^{n} Valid[i] * W_i * Sim(R1_i, R2_j) \leq \sum_{i=k+1}^{n} W_i$，

我们能得出$\sum_{i=1}^{n} Valid[i] * W_i * Sim(R1_i, R2_j) < LowThreshold$，因此函数 *RecordMatch* 将返回 *false*

4.2 算法复杂度分析

算法的执行时间 *T* 由三部分组成：步骤一属性预处理的时间复杂度用 *T1* 表示，步骤二多次排序以及排序结果的验证的时间复杂度用 *T2* 表示，步骤三重复记录检测的时间复杂度用 *T3* 表示。步骤一与步骤二同上一章相同，即有 *T1=O(N)*，*T2= O(NlogN)*，*N* 为记录集的记录条数。步骤三中，先对属性权重进行排序，复杂度为 *O(nlogn)*，其中 *n* 是每条记录中属性字段的个数。然后进行记录匹配，其复杂度小于$\frac{N}{w}w^2n$，w^2n 为在每个窗口内进行属性字段比较的次数，因为过滤机制的引入，因此每条记录的比较次数远远少于 *n* 次，*N/w* 为比较的窗口数。因为在一般情况下属性字段的个数 *n* 远远小于记录条数 *N*，所以有 *T3=O(NlogN)*。综上，该算法的时间复杂度 *T=O(NlogN)*.

4.3 实验及结果分析

实验环境为，2.83Hz Intel Core i3 处理器，3GB 内存，Windows XP professional SP3 操作系统，算法用 VC 6.0 实现。实验用三组数据集进行检验，分别包含 1000,5000 和 10000 条数据，每条数据包含 16 个属性字段，代表个人及家庭的信息，由 Pudjijono 提供的方法产生并可模仿真实世界的数据[64]。实验中，我们选取 5 个属性组成排序属性集，再选取 5 个属性组成众数属性集。

用固定变量的方法，选择相同的窗口大小，在相同的数据集中，将本章提出的新算法与上一章中的算法进行召回率的比较，实验结果如表 4-1 所示

表 4-1 召回率比较

数据集 \ 窗口大小	函数 *RecordMatch*				函数 *RecordMatch*$^+$			
	4	5	6	7	4	5	6	7
1000	91.39%	91.39%	91.76%	92.88%	91.39%	91.39%	91.76%	92.88%
5000	98.42%	98.50%	98.65%	98.80%	98.42%	98.50%	98.65%	98.80%
10000	94.67%	94.90%	95.24%	95.61%	94.67%	94.90%	95.24%	95.61%

由表 4-1 可见，与上一章的算法 *RecordMatch* 相比，本章的算法 *RecordMatch*$^+$的召回率没有下降，这表明算法 *RecordMatch*$^+$的正确率和准确率仍保持较高，并通过实验验证了上述的数学证明，即函数 *RecordMatch*$^+$与函数 *RecordMatch* 得到相同的比较结果。

在对算法的执行正确率进行验证的基础上，我们需要知道新算法的执行效率是否有所提高。下面，我们在不同数据集和不同窗口大小的条件下比较本章算法 *RecordMatch*$^+$与上一章算法 *RecordMatch* 的运行时间。如图 4-2 到 4-4 所示。

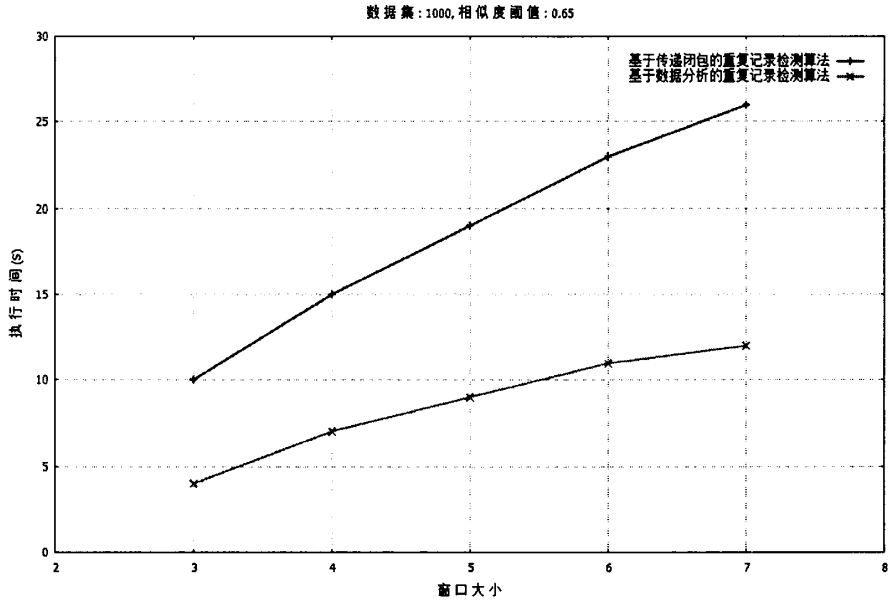


图 4-2 数据集为 1000 时的算法运行时间比较

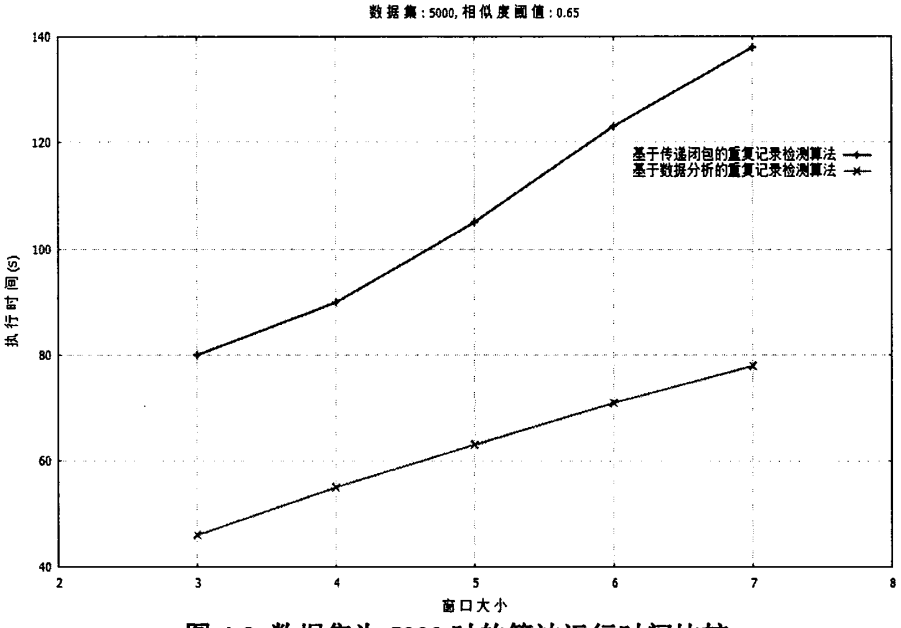


图 4-3 数据集为 5000 时的算法运行时间比较

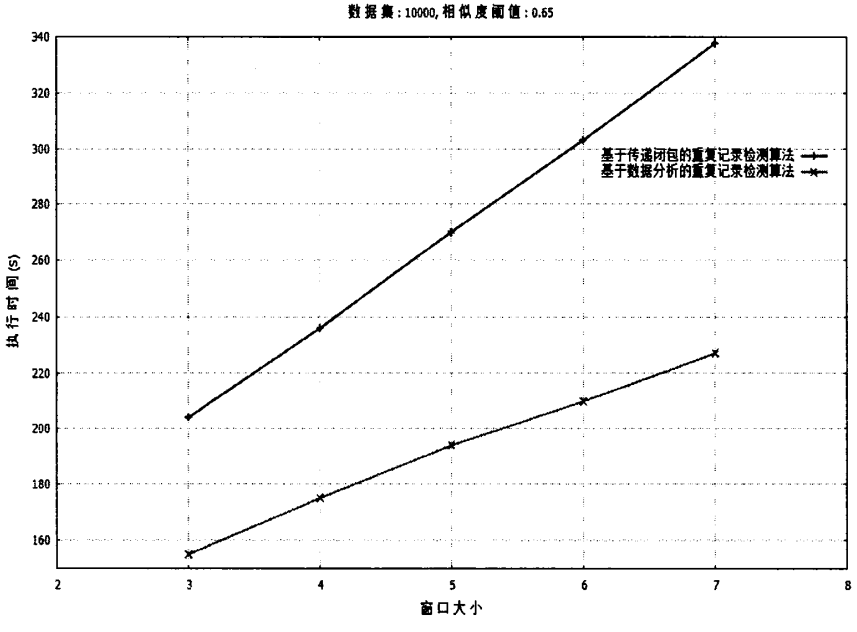


图 4-4 数据集为 10000 时的算法运行时间比较

图 4-2 为数据集为 1000 时，不同滑动窗口算法 *RecordMatch* 与算法 *RecordMatch*$^{+}$运行时间比较。图 4-3 为数据集为 5000 时，不同滑动窗口算法 *RecordMatch* 与算法 *RecordMatch*$^{+}$运行时间比较。图 4-4 为数据集为 10000 时，

不同滑动窗口算法 *RecordMatch* 与算法 *RecordMatch*$^+$运行时间比较。固定相似度阈值 *LowThreshold*=0.65。从图中可以看出，在相同的滑动窗口和相同的数据集的条件下，算法 *RecordMatch*$^+$的运行时间比上一章算法 *RecordMatch* 有明显的下降。算法的效率得到很大的提高。而且，随着窗口的增大，原算法 *RecordMatch* 的增长率较高，曲线较陡，增加速度较快。而本章算法 *RecordMatch*$^+$的增加幅度较低，曲线斜率较小，增长的速度较为平稳。说明随着窗口的增大，原算法 *RecordMatch* 的执行时间将大幅增加，而本章算法较为平缓，且当相似度阈值设定为合适的值时，运行时间将保持在很小的区间值。

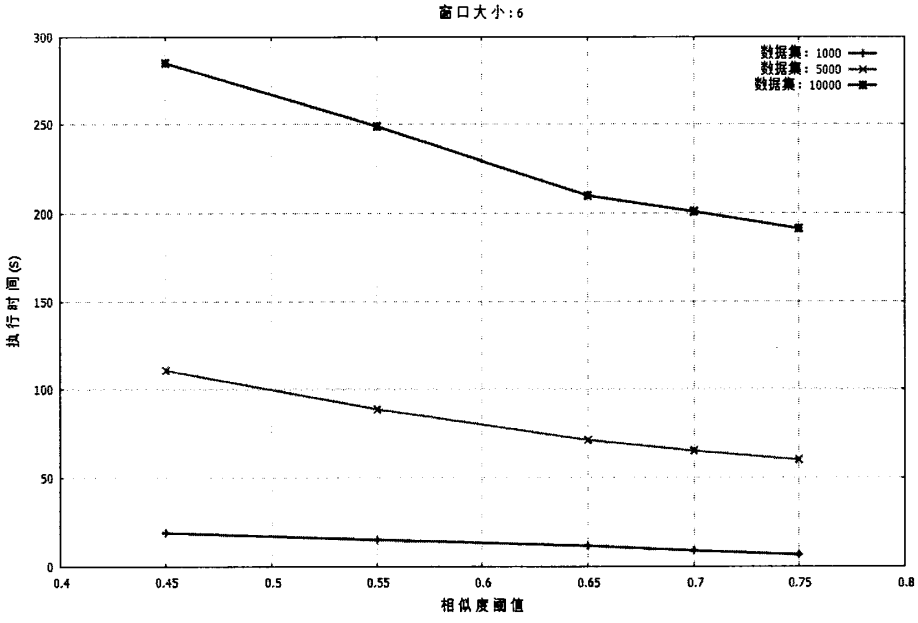


图 4-5 不同相似度阈值 (*LowThreshold*)的算法执行时间(Execution Time)

图 4-5 为不同相似度阈值下的算法执行时间，其中窗口大小 w 为 6。从图中可以看出，随着相似度阈值的增大，算法的执行时间逐渐减少并趋于平缓，当相似度阈值取合适的值时，算法的执行效率较为稳定。

4.4 本章小结

本章的算法在基于上一章算法的基础上，提出了两个方面的改进，一是进行属性分析，并按照权重的大小为顺序进行记录匹配，二是引入过滤机制，相当于剪枝，适时中止不需要再进行的记录匹配操作。在保证正确率的同时，有效的提高了算法的效率，减少算法的运行时间。

这两个算法在合并步骤都采用传递闭包的方法，即在滑动窗口中对记录进行匹配和比较后，采用传递闭包的方式对检测出的重复记录进行合并操作。但是通过分析发现传递闭包会引起误识别的问题。比如，记录 *R1* 和记录 *R2* 相似，记录 *R2* 和记录 *R3* 相似，但是记录 *R1* 和记录 *R3* 可能差别很大。而传递闭包却认为记录 *R1* 与记录 *R3* 是相似重复记录。在下一章节中我们将针对这一问题提出新的改进。

第五章 基于完全子图的重复记录检测算法

5.1 重复记录合并

5.1.1 传递闭包的弊端

上一章提出的算法中，在进行重复记录检测后，需要对同一窗口中检测出的相似重复记录进行合并，将每次检测识别出的重复记录合并为一组，在合并时假定记录的重复是具有传递性的，即采用传递闭包的方法。所谓传递闭包，是指若 *R1* 和 *R2* 互为重复记录，*R2* 与 *R3* 互为重复记录，则 *R1* 与 *R3* 互为重复记录。通过此方法可以得到较完全的重复记录集合，能部分解决匹配遗漏的问题。但是如上章提到的，传递闭包容易引起误识别的问题。

因此本章将提出针对传递闭包的改进算法，在合并重复记录的步骤中，不采用传递闭包的方法，而是将记录集中的记录等价的转换为连通图中的点，并将记录的相似性用连通图中对应的点间的连通性来表示，即，若有两条记录，则分别用两个点来表示记录，若它们相似则这两个点是连通的，若不相似则不连通。那么合并重复记录的过程便转换为在连通图中寻找完全子图的过程。具体步骤将在下文中详细介绍。

5.1.2 合并方法介绍

为了方便下面算法的介绍，在此先给出一些相关的定义介绍。

定义1：如果两条记录 *R1* 和 *R2* 的相似度值大于给定的相似度阈值，那么 *R1* 和 *R2* 被认为是相似重复记录。

定义2：对于给定的记录集 $S = \{R1, R2, \dots, Rn\}$，如果属于集合 S' 中的任两条记录 *Ri* 和 *Rj* 是相似重复记录，且 $S' \subseteq S$，那么 S' 是一个相似记录集。

定义3：对于给定的记录集 $S = \{R1, R2, \dots, Rn\}$，有一个对应的关联图 $G(V,E)$。在图 G 中，记录集 S 中的某一条记录 *Ri* 代表一个点 $Ri \in V$，如果两条记录 *Ri* 和 *Rj* 是重复记录，那么有一条无向边 $e(Ri, Rj) \in E$ 将图 G 中的点 *Ri* 和点 *Rj* 相连。

比如，有一个记录集 $S = \{R1, R2, R3, R4, R5\}$，已知 *R1* 和 *R2*，*R1* 和 *R3*，*R1* 和 *R4*，*R1* 和 *R5*，*R3* 和 *R4*，*R3* 和 *R5* 是重复记录。那么与之相对应的连通图如图5-1所示

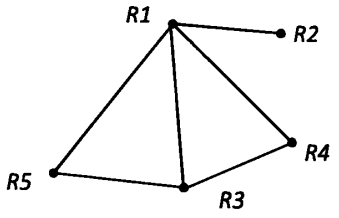


图5-1 连同图例子

定义4：对于给定的记录集 $S=\{R1, R2, \ldots, Rn\}$ 和它的子集 S'，如果 S' 是一个相似记录集，则存在一个完全子图 $G'(V',E')\subseteq G(V,E)$，其中 G 是集合 S 的对应连通图，G' 是集合 S' 的对应连通图，$V'\subseteq V$ 且 $E'\subseteq E$。

在上面的例子中，因为 $R1$ 与 $R3$, $R1$ 与 $R4$, $R3$ 与 $R4$ 是重复记录，这三条记录组成一个相似的记录集 S'，即$S'=\{R1, R3, R4\}$。S' 对应的连通图有三个点分别代表记录 $R1$, $R3$, 和 $R4$，三条边 $e(R1, R3)$, $e(R1, R4)$ 和 $e(R3, R4)$，分别连接 $R1$ 和 $R3$，$R1$和 $R4$，$R3$ 和 $R4$。

本章提出的算法，是在基于图 3-1 所示的算法流程的基础上，先完成第一和第二步，此时，已能够确定同一滑动窗口中任两条记录是否相似。然后，根据上述的定义，可以构造出一个无向连通图，图中的点与记录一一对应。合并重复记录的过程便转换为搜索完全子图的过程，其中一个完全子图就代表一个相似记录集[65,66]。通过这种方式，传递闭包引起的误识别的问题能够得到有效避免。

5.2 完全子图检测流程

因为将相似的记录集合视为一个完全子图，所以我们通过找完全子图的方法来合并一个滑动窗口内的重复记录。该方法将滑动窗口中的第一条记录所表示的点视为一个连通图中的第一个点，如果有存在记录与这个连通图中的所有点所代表的记录都相似，那么将该记录所代表的点加入连通图中。重复上述步骤直到所有记录都被处理完毕为止。该过程如图 5-2 所示。

n 是所有记录的数目，w 是窗口大小，数组 *adjacentmatrix* 用来记录两个点相连与否，数组 *AdjacentRi* 用来存储与记录 R_i 相似的记录，即它们所代表的点相连，数组 *assemble* 用来存储所有检测出的重复记录集合。该合并过程在滑动窗口中对从第一条记录到第$(n-w+1)$条记录进行操作，第一步，在每个滑动窗口中找出与窗口中第一条记录 R_i 相似的记录，并把它们保存至数组 *AdjacentRi* 中。

第二步，计算数组 *AdjacentRi* 中每条记录所代表点的关联度。第三步，找出在窗口中包含 R_i 的所有的完全子图。第一和第二步由算法 *calculateParameter* 实现，第三步由算法 *findCompleteG* 实现。

```
Input: adjacentmatrix[][], n, w
Output: assemble[][]

1.  for i = 1 to (n-w+1) do begin
2.        AdjacentRi[] = ∅;
3.        calculateParameter(i);
4.        findCompleteG(i, AdjacentRi[]);
5.  end for;
```

图 5-2 合并过程

算法 *calculateParameter* 如图 5-3 所示。

```
Input: i
Output: AdjacentRi[]

1. for x = (i+1) to (i+w-1) do begin
2.      if (e(R_i, R_x) ∈ E)
3.          AdjacentRi[]← R_x;
4. end for;
5. if (i == 1) begin
6.      for each R_j ∈ AdjacentRi[] do begin
7.          for k = i to i+w-1
8.              if (e(R_j, R_k) ∈ E)
9.                  R_j.degree ++;
10.                 R_k.degree++;
11.     end for;
12. end if;
13. else if(e(R_i, R_{i+w-1}) ∈ E)  begin
14.     for x = i to (i+w-2) do begin
15.         if (e(R_x, R_{i+w-1}) ∈ E)
16.             R_x.degree ++;
17.             R_{i+w-1}.degree ++;
18.     end for;
19. end if;
```

图 5-3 函数 *calculateParameter*

算法中，参数 i 代表当前窗口的起始位置。

第 1 行到第 4 行，根据矩阵 *adjacentmatrix*，找到与点 R_i 相连的所有点并存入数组 *AdjacentRi* 中。

第 5 行到第 19 行是步骤二。$R_i.degree$ 代表数组 *AdjacentRi* 中点 R_i 的度数，每个点的度是该点与其它点相连的无向边的数目。第 5 到 12 行显示第一个滑动窗

口中，即 i 为 1 时，度的计算方法。通过嵌套循环计算数组 *AdjacentRi* 中每个点与其它所有点的关联度，检查每两个点之间是否有边相连。第 13 行到 19 行，算法对第一个滑动窗口以外的记录进行处理。因为在一个新的窗口中，所有记录所代表的点除了最后一个之外，（最后一条记录是刚进入该滑动窗口的）它们的度在之前的窗口中都生成过并进行了处理，因此算法仅需处理窗口中的最后一条记录。如果最后一条记录所代表的点与窗口中的第一条记录所代表的点不相连，那么这条记录便不可能与窗口中第一条记录组成完全子图。反之如果相连的话，那么窗口中每个点的度都将被相应的更新，此过程如算法中第 14 行到第 18 行所示。

算法 *findCompleteG* 如图 5-4 所示

```
Input: i, AdjacentRi[]
Output: assemble[][]

1.  for each R_j ∈ AdjacentRi[] do begin
2.      while (R_j.degree > 0) begin
3.          potential[] ← R_i;
4.          completeG(AdjacentRi[], R_j);
5.          if (|potential[]| > 1)
6.              assemble[][] ← potential [] ;
7.              potential[] = ∅;
8.      end while;
9.  end for;
```

图 5-4 函数 *findCompleteG*

算法中，参数 i 代表当前窗口的起始位置。

我们将 R_i 视为当前窗口中完全子图的基准点，并找出窗口内所有的重复记录。数组 *potential* 用来暂时的存储这些相似重复记录，即隶属于某一个完全子图的所有点。算法遍历数组 *AdjacentRi* 找出所有度大于 0 的点。如果存在这样的 R_j 点，这说明 R_i 可能能够和其他的点组成完全子图。算法 *completeG* 找出所有与 R_i 组成完全子图的点，当找到一个有效的完全子图时，将数组 *potential* 中的值复制到 *assemble* 中，*assemble* 用来存储所有的完全子图，并将数组 *potential* 重置以备下一轮的数据处理，如图 5 到 7 行所示。

函数 *completeG* 如图 5-5 所示，我们已经知道，R_i 在数组 *potential* 中，并且 R_j 可能成为第一个被加入数组 *potential* 中的值，第 3 到 7 行，函数检验 R_j 是否与数组 *potential* 中所有的点都相连，如果是，在 8 到 13 行，更新 R_j 和数

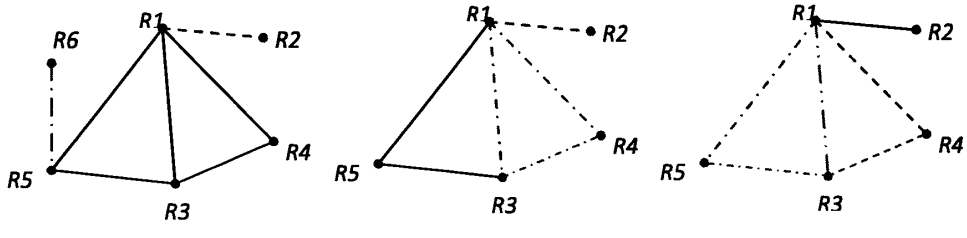
组 *potential* 中所有点的度数，并将 R_j 加入数组 *potential* 中。如此，不断的填充数组 *potential*。在 14 行，增加变量 *j* 的值，并继续寻找其它符合条件能够加入数组 *potential* 中的点，直到属于数组 *AjacentRi* 中的点都被检测完为止。

```
Input: AdjacentRi[], Rj
Output: potential[]

1.   while (∃ (Rj.degree>0 && Rj∈AdjacentRi[] && Rj∉potential[] && j<=i+w-1)) do begin
2.       flag = 1;
3.       for each x ∈ potential[] do begin
4.           if (e(x, Rj) ∉ E)
5.               flag = 0;
6.               break;
7.       end for;
8.       if (flag == 1) begin
9.           Rj.degree = Rj.degree - |potential[]|;
10.          for each x ∈ potential[] do
11.              x.degree --;
12.          potential[] ← Rj;
13.      end if;
14.      j++;
15. end while;
```

图 5-5 函数 *completeG*

对连通图中的任意两个完全子图，存在三种可能的情况。一是没有共边也没有共点，如图 5-6(a)所示，二是有共点的情况，如图 5-6 (b)所示，三是有共边的情况，如图 5-6 (c)所示。文章中的算法通过数组 *adjacentmatrix* 来判断任两个点是否有边相连。比如，如果 R_i 和 R_j 有边相连，*adjacentmatrix*[R_i][R_j]的值为 1，否则值为 0。算法在合并的过程中动态的改变并维系每个点的度数，并用这些度的信息来决定对应的点能否被加入完全子图。通过这种方式，能够避免完全子图的漏识别和误识别。



(a) (b) (c)

图 5-6 完全子图情况

下面用一个例子更直观的阐述上述提出的算法。我们仍以图 18 为例，设定

滑动窗口的大小为 5。

步骤一，窗口中与 *R1* 相连的点被存储在数组 *AdjacentRi* 中，为{*R2*, *R3*, *R4*, *R5*}

步骤二，计算每个点的度，结果如下，*deg*(*R1*) = 4, *deg*(*R2*) = 1, *deg*(*R3*) = 3, *deg*(*R4*) = 2, *deg*(*R5*) = 2

步骤三，*R1* 首先被加入数组 *potential* 中，即 *potential*[*0*]=*R1*，执行函数 *completeG* 的过程中，点 *R2* 与数组 *potential* 中的所有点相连，*R2* 随即被加入数组 *potential* 中，有 *potential*[*1*]=*R2*，并修改相应的度数如下，*deg*(*R1*)=*3*, *deg*(*R2*)=*0*。此时发现数组 *AdjacentRi* 中已没有其他的点同时与 *R1* 和 *R2* 相连，如此便找到了第一个由 *R1* 和 *R2* 组成的完全子图。

因为 *R3* 的度大于 0，函数 *findCompleteG* 继续搜索其它的完全子图。此时仍有 *potential*[*0*]=*R1*。执行函数 *completeG* 时，点 *R3* 与数组 *potential* 中所有的点相连，*R3* 便被加入数组 *potential* 中，有 *potential*[*1*]=*R3*，并修改相应的度数如下，*deg*(*R1*)=*2*, *deg*(*R3*)=*2*。点 *R4* 也与数组 *potential* 中所有的点相连，将 *R4* 加入数组 *potential* 中，有 *potential*[*2*]=*R4*，并修改相应的度数如下，*deg*(*R1*)=*1*, *deg*(*R3*)=*1*, *deg*(*R4*)=*0*。现在数组 *AdjacentRi* 中没有其它的点与 *R1*，*R3* 和 *R4* 相连。如此便找到了第二个由 *R1*，*R3* 和 *R4* 组成的完全子图。通过此方法，我们能找到另一个由 *R1*，*R3* 和 *R5* 组成的完全子图。

5.3 特殊情况改进

对于如上所述的算法，通过搜索完全子图进行重复记录合并。一个完全子图即代表一个相似重复记录集。通过分析我们发现，大部分情况下，不同的相似重复记录集合间很少拥有两个或两个以上相同的记录，转换为对应的完全子图的情况，即为任意两个完全子图只存在一个公共点。针对这类较为普遍的特殊情况，对上述的算法提出一点改进。

改进算法的主要流程同上面所述的算法一致，主要区别为函数 *findCompleteG*$^{+}$，具体步骤如图 5-7 所示。

算法中，我们将数组 *AdjacentRi* 中的第一条记录所代表的点视为完全子图的基准点，并找出窗口内所有与该点构成完全子图的点所代表的记录，即相似重复记录集合。对 *AdjacentRi* 数组中与当前窗口第一条记录所代表的的点相连

接的点，按照度数进行分类，并存储在二维数组 *degreeGroup* 中。比如，将数组 *AdjacentRi* 中度数为 1 的点依次存放在数组 *degreeGroup*[1][]中，将度数为 2 的点依次存放于数组 *degreeGroup*[2][]中，如算法 1 到 3 行所示。对数组 *degreeGroup*[1][]中的每一个元素，因为它们的度数均为 1，分别与当前窗口中第一条记录所代表的点构成完全子图，因此分别将这些元素放入用于存放寻找到的完全子图的数组 *assemble* 中，如算法 4 到 5 行所示。在接下来的 6 到 14 行中，算法处理每个度为 2 以上的点，即从 *AdjacentRi*[2][]开始遍历数组，在该数组的每个维度中做如下操作，若存在度数大于 0 的点，调用函数 *completeG*，如图 5-5 所示。函数 *completeG* 寻找所有与窗口中第一条记录所代表的点，即 *Ri* 构成完全子图的所有点。当找到一个完全子图后，将数组 potential 中的元素复制到二维数组 assemble 中，并清空数组 potential 为下一轮操作做准备。如是，直到找出窗口内的所有的完全子图，即重复记录集合。

```
Input: i, AdjacentRi[], degreeGroup[][]
Output: assemble[][]

1.  degreeGroup[w][w] ← 0
2.  for each Rj ∈ AdjacentRi[] do begin
3.      degreeGroup[Rj.degree][] ← Rj
4.  for Rj ∈ degreeGroup[1][]
5.      assemble[][]← Rj
6.  for each Rj ∈ AdjacentRi[m++][]//m=2
7.      if (Rj.degree > 0) begin
8.          potential[] ← Ri;
9.          completeG(AdjacentRi[], Rj);
10.         if (|potential[]| > 1)
11.             assemble[][] ← potential [] ;
12.             potential[] = ∅;
13.     end if;
14. end for;
```

图 5-7 函数 *findCompleteG*$^+$

该改进算法主要是针对只有共点，没有共边的完全子图的情况，因为我们发现这一情况是普遍现象，因此有必要专门提出来加以研究。对于这一特殊情况，我们采取的处理方法是，将每条记录按照度数的不同进行分类。对于度数为 1 的点，它们相互间肯定构成一个完全子图，因此不必通过计算可找到第一个完全子图。接下来以每个点的度数大小为顺序，寻找完全子图，并动态维护每个点的度数。这样做的好处是，相同度数的点更有可能构成完全子图，可减

少循环步骤和处理的操作。

下面通过一个例子来解释上述的算法，例子如图 5-8 所示。

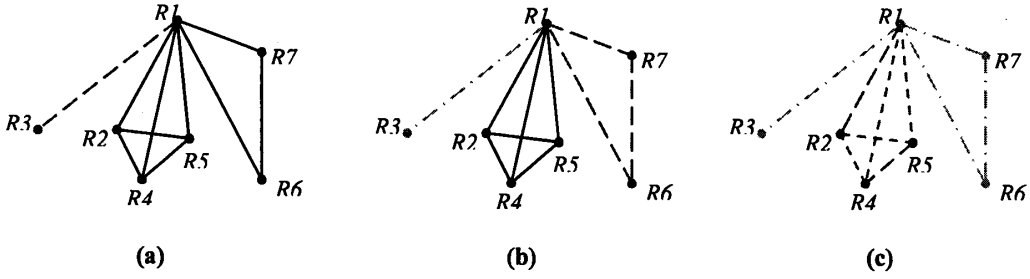


图 5-8 完全子图例子

在这个例子中，窗口大小为 7，通过函数 *calculateParameter* 的计算，将与 *R1* 相连接的点存入数组 *AdjacentRi* 中，有 *R2*，*R3*，*R4*，*R5*，*R6* 和 *R7*。数组 *AdjacentRi* 中每个点的关联度如下所示，deg(*R2*) = 3, deg(*R3*) = 1, deg(*R4*) = 3, deg(*R5*) = 3, deg(*R6*) = 2, deg(*R7*) = 2。

在函数 *findCompleteG* 中，根据点的度数进行分类，有 *degreeGroup*[1][0] =*R3*, *degreeGroup*[2][0]=*R6*, *degreeGroup*[2][1]=*R7*, *degreeGroup*[3][0]=*R2*, *degreeGroup*[3][1]=*R4*, *degreeGroup*[3][2]=*R5*。对于数组 *degreeGroup*[1][]中的点，找到第一个由 *R1* 和 *R3* 组成的完全子图，如图 5-8(a)所示。对于数组 *degreeGroup*[2][]中的点，通过函数 *completeG* 计算得到第二个由 *R1*、*R6* 和 *R7* 组成的完全子图如图 5-8(b)所示。对于数组 *degreeGroup*[3][]中的点，通过函数 *completeG* 计算得到第三个由 *R1*、*R2* 和 *R4* 组成的完全子图如图 5-8(c)所示。

5.4 算法复杂度分析

合并的步骤由函数 *calculateParameter* 和 *findCompleteG* 组成。*calculateParameter* 的时间复杂度是 $O(w^2)$，w 是滑动窗口的大小。对于函数 *findCompleteG*，它需要对每条边至少遍历一次，最极端的情况就是滑动窗口中的所有点都是全连接的，那么 *findCompleteG* 的时间复杂度是 $O(w^2)$，这两个算法的执行次数接近 N 次，N 为数据集的数据总数，那么合并过程的时间复杂度是 $O(Nw^2)$。重复记录检测的执行时间 T 由三部分组成，属性处理执行时间 $T1$，多次排序和排序结果验证时间 $T2$，重复记录检测执行时间 $T3$。在上一章中已做过分析，$T1=O(N)$，$T2=O(NlogN)$。步骤三由属性字段匹配，记录匹配和合并重复记录组成，字段匹配和记录匹配的时间复杂度为 $O(wnN)$，其中 n 是每条记录

中属性字段的个数。因为在大多数情况下 n 比 w 大，步骤三的时间复杂度是 $O(wnN)$。如果 n 比 $logN$ 小的话，步骤三的时间复杂度为 $O(N)$。如果 n 比 $logN$ 大的话，步骤三的时间复杂度为 $O(NlogN)$。综上这两种情况，重复记录检测步骤的时间复杂度 $T3= O(NlogN)$。因此算法的时间复杂度 $T= O(NlogN)$

对于特殊情况下的改进算法，主要区别在函数 *findCompleteG*$^+$，由于该算法将点根据它们的度数分成不同的组，并且在每组中能够在时间 $O(w)$内完成完全子图的搜索操作，所以函数 *findCompleteG*$^+$的时间复杂度为 $O(w)$。与 *findCompleteG* 函数 $O(w^2)$的时间复杂度比起来，能够有效的缩短运算时间，提高算法效率。

5.5 实验及结果分析

为衡量新合并方法的效率，我们仍旧通过计算召回率和误识别率作为度量的标准。

所有的实验均在下述环境中完成，2.93Hz Intel Core i3 处理器，4GB 内存，Windows 7 操作系统，算法用 VC 6.0 实现。三组实验数据集分别包含 1000,5000 和 10000 条数据。数据集由 Pudjijono 提供的方法产生[64]。每条数据包含 16 个属性字段。我们选取 5 个属性组成排序属性集，5 个属性组成众数属性集。

实验在固定大小的窗口和相似度阈值下比较召回率和误识别率。设定滑动窗口 w=6，相似度阈值 *LowThreshold*=0.7。结果如表 5-1 所示。

表 5-1 不同数据集的召回率和误识别率

数据集	召回率	误识别率
1000	94.0%	0%
5000	99.3%	0%
10000	97.8%	0%

从表 5-1 中，我们可以看到该算法具有较高的稳定性。当数据集的数目增大时，召回率保持在足够高的情况下，误识别率仍维持很低。同时，对于不同的数据集，召回率和误识别率没有很大的波动。

图 5-9 为不同大小的窗口下，不同数据集的算法执行时间，其中相似度阈值 *LowThreshold* 为 0.6。从图中可以看出，随着窗口的增大，算法的执行时间随之增加，这是因为随着窗口的增大，窗口中需要进行匹配对比操作的记录数

目增加了。但是执行时间的增长率仍然较为平缓可控。

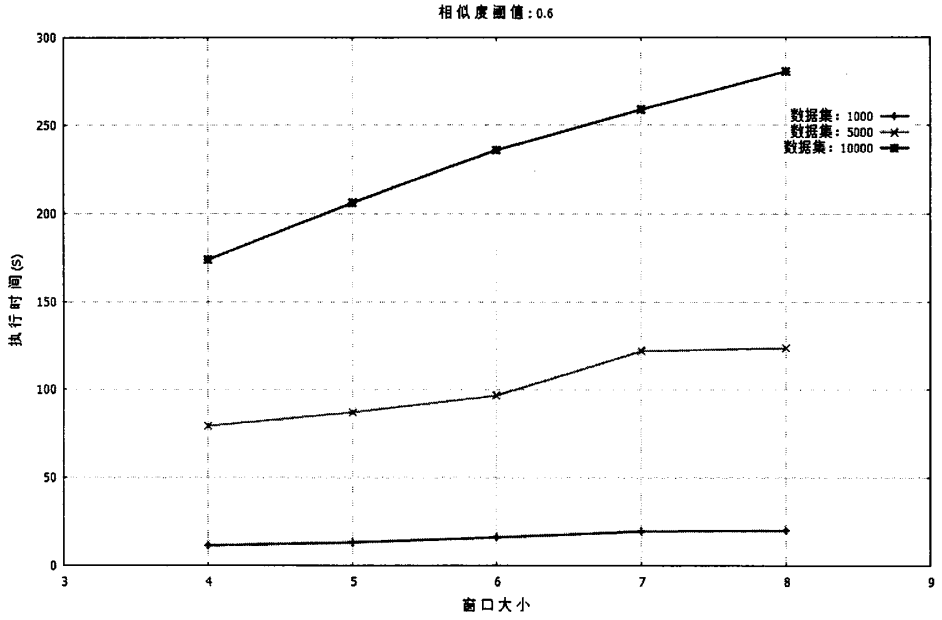


图 5-9 不同窗口大小(*w*)的算法执行时间(Execution Time)

图 5-10 不同相似度阈值下，不同数据集的算法执行时间，其中滑动窗口的大小 *w* 为 6。从图中可以看出，随着相似度阈值的增大，算法执行时间随之减少，并趋于平稳。

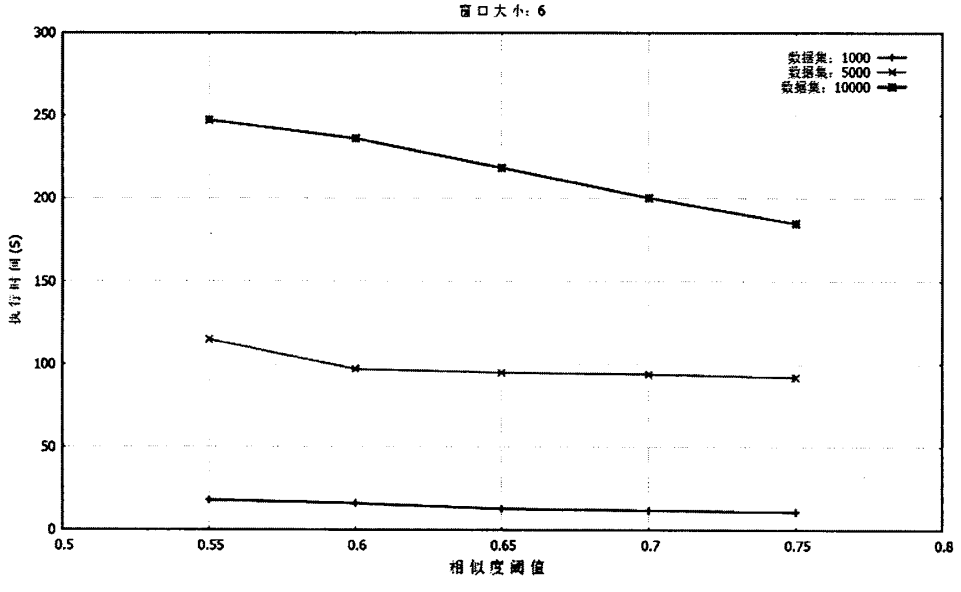
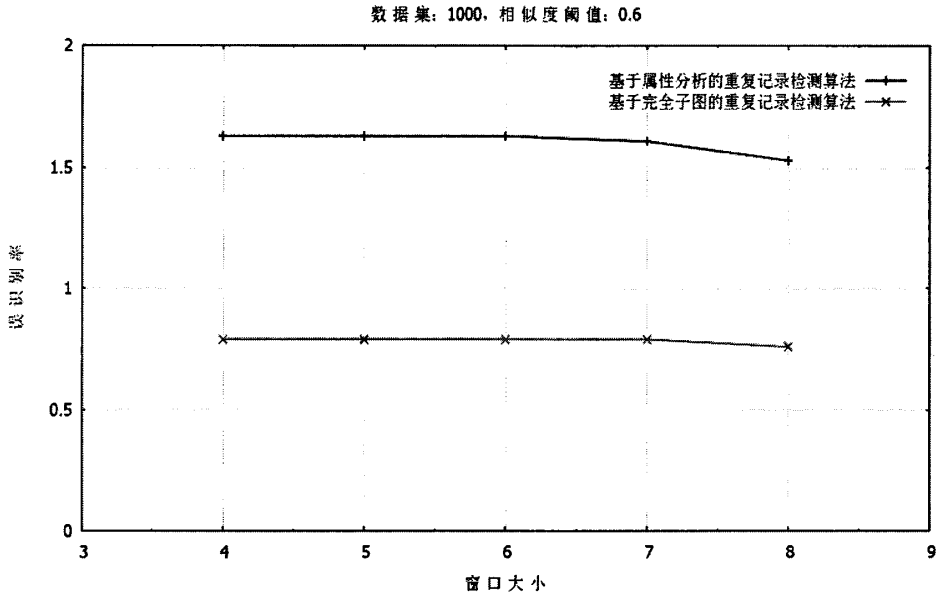
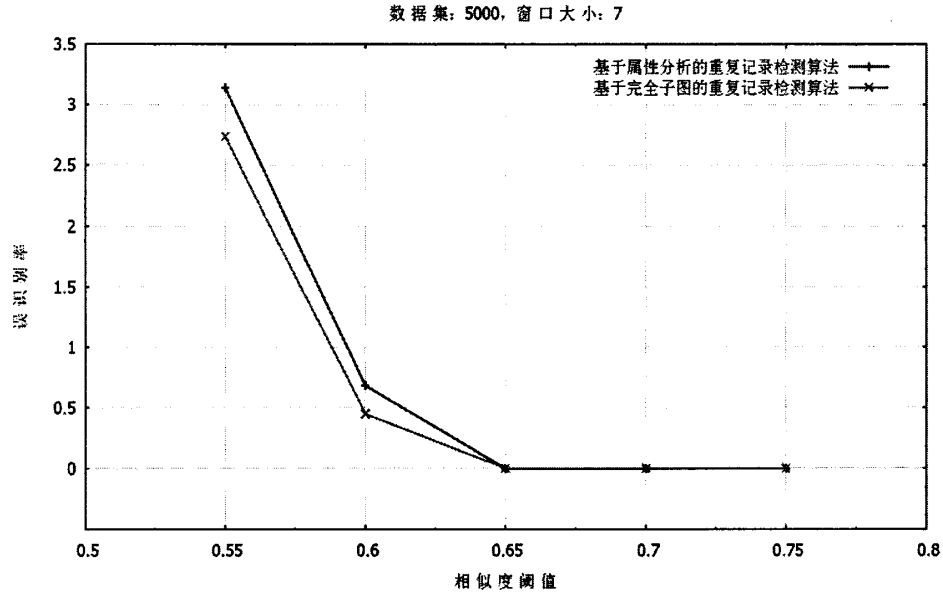


图 5-10 不同相似度阈值(*LowThreshold*)的算法执行时间(Execution Time)

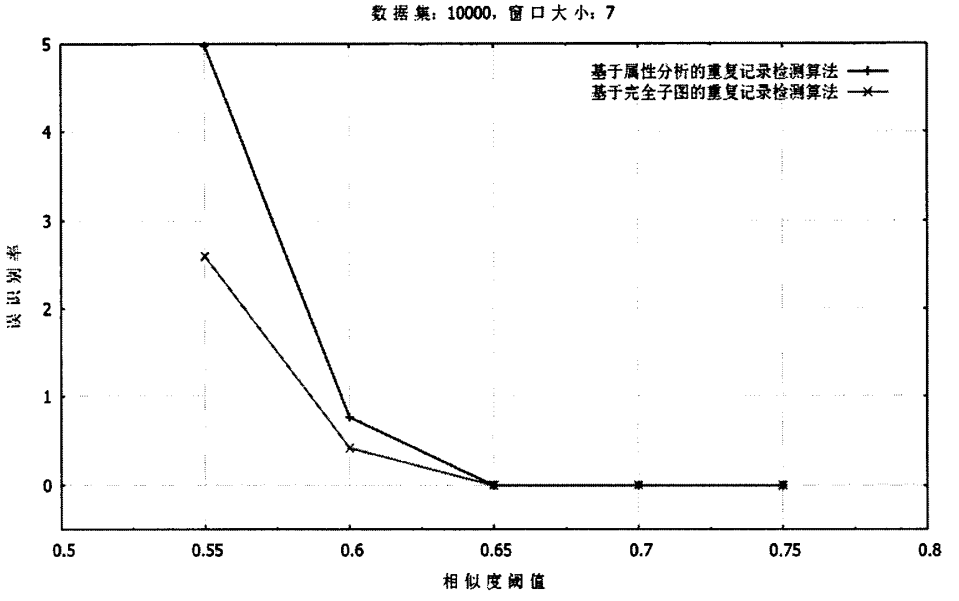
下面我们就误识别率这一评价标准对本章的基于完全子图的算法同上一章基于传递闭包的方法进行比较。结果如下所示。



(a)



(b)



(c)

图 5-11 不同记录集合下基于完全子图与基于传递闭包的算法误识别率对比

图 5-11(a)为基于完全子图的算法与基于传递闭包的算法在数据集数目为1000，相似度阈值为 0.6 的情况下，不同滑动窗口的误识别率比较。图 5-11 (b)为基于完全子图的算法与基于传递闭包的算法在数据集数目为 5000，滑动窗口大小为 7 的情况下，不同相似度阈值的误识别率比较。图 5-11(c)为基于完全子图的算法与基于传递闭包的算法在数据集数目为 10000，滑动窗口大小为 7 的情况下，不同相似度阈值的误识别率比较。运用控制变量的方法，在同等的条件下，基于完全子图的方法的误识别率比基于传递闭包的方法的误识别率要低。实验结果表明，本章提出的基于完全子图的算法能够有效的解决因为传递闭包而造成的误识别的问题。从(a)中可以看出，当相似度阈值 *LowThreshold* 固定时，窗口越大，误识别率越低。这是因为随着窗口的增大，在窗口中进行相互匹配与识别的记录数目越多，识别出的相似重复记录增加。从(b)、(c)中可以看出，当窗口大小 w 固定时，相似度阈值越大，误识别率越低。这是因为随着相似度阈值的增大，判定记录相似性的要求更加严格。不论是采用固定窗口大小亦或是固定相似度阈值的比较方式，都可以看到，基于完全子图算法的误识别率要低于基于传递闭包算法的误识别率，性能较好。

接下来对针对特殊情况下的算法进行实验分析，即基于两个完全子图只有共

点，没有共边的假设的前提下的算法改进。具体实验结果如下图所示。

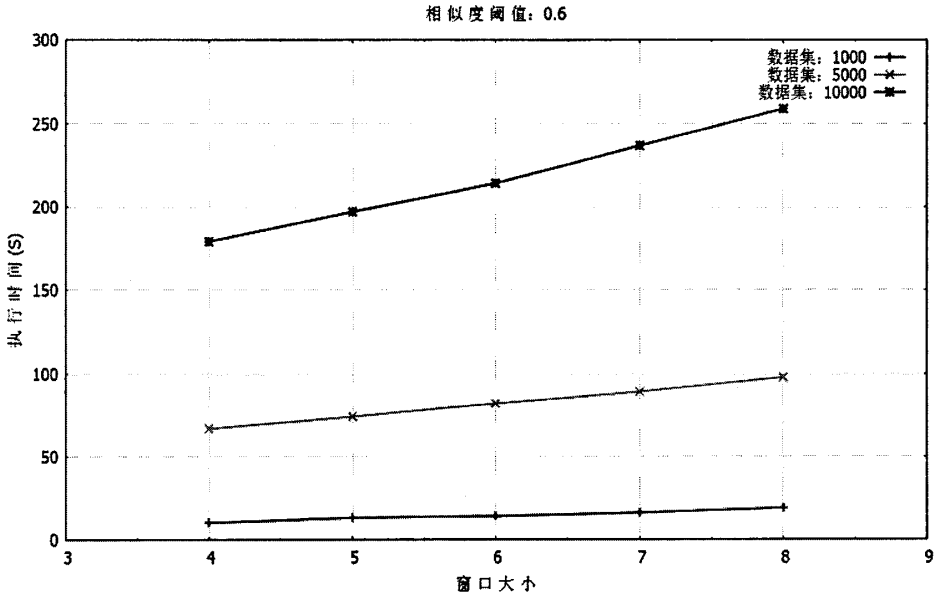


图 5-12 不同窗口大小(*w*)的算法执行时间(Execution Time)

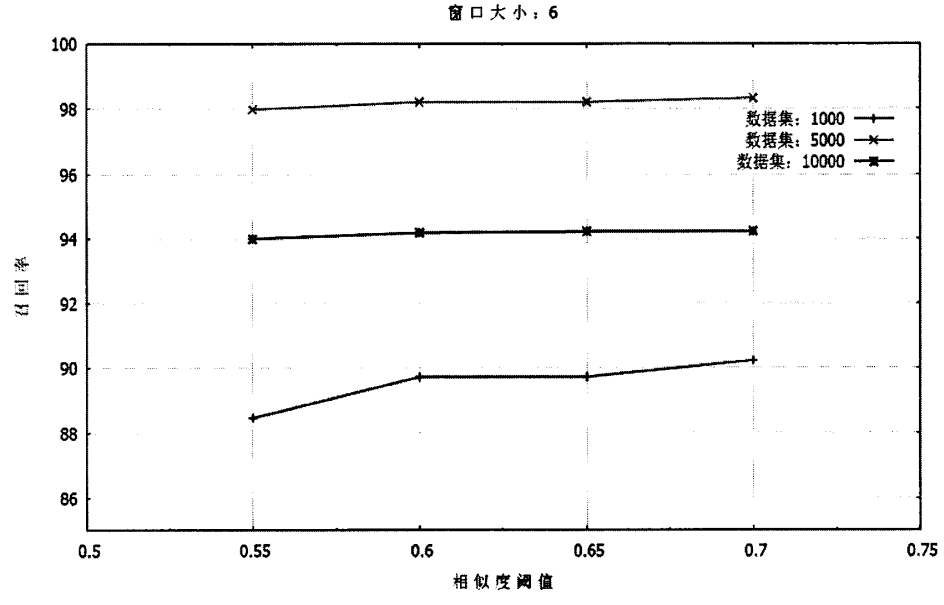


图 5-13 不同相似度阈值(*LowThreshold*)的算法召回率(Recall Rate)

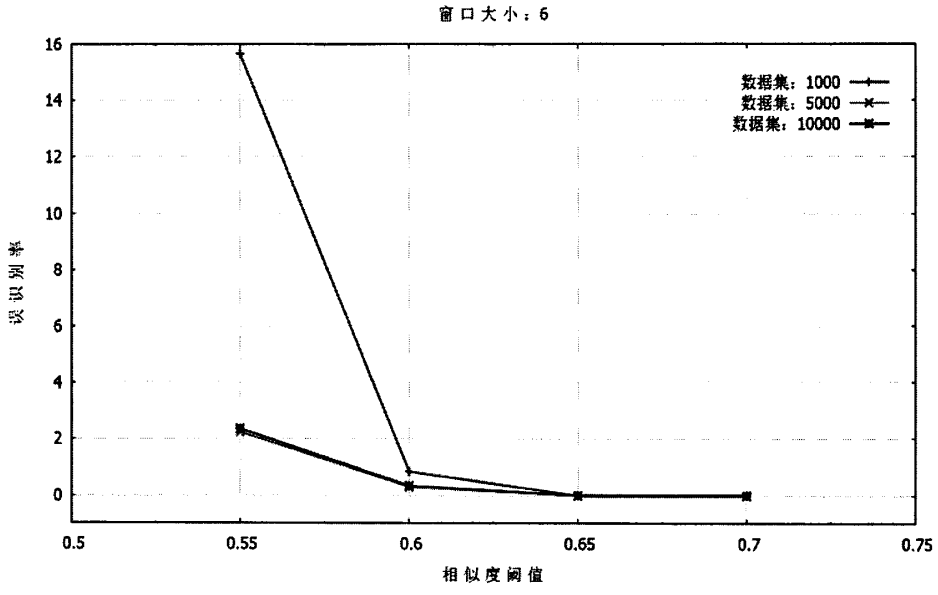


图 5-14 不同相似度阈值(*LowThreshold*)的算法误识别率(False Merge Rate)

图 5-12 为不同窗口大小情况下的算法执行时间，相似度阈值 *LowThreshold* 为 0.65。随着窗口增大，算法的执行时间相对较为平缓。且与图 5-9 进行对比发现，算法的执行时间得到了有效降低，也用实验验证了上一节中算法复杂度分析的正确性。

图 5-13 为不同相似度阈值情况下的算法召回率，滑动窗口大小固定为 6。从图中可以看出，当相似度阈值增大时，召回率基本保持稳定，并维持在较高的水平。

图 5-14 为不同相似度情况下的算法误识别率，滑动窗口大小固定为 6。从图中可以看出，随着相似度阈值的增大，误识别率逐渐降低，因为算法对于相似性的定义越来越严格，最后对于所有的记录集合误识别率都降为 0。

基于特殊情况的改进算法的实验结果表明，该算法具有较高的召回率以及较低的误识别率，即在保持算法性能的基础上，有效的降低了算法的执行时间。有利于处理特殊但普遍存在的这一数据集情况。

5.6 本章小结

本章提出的基于完全子图的重复记录检测算法，能够有效的解决合并已检

测的重复记录时传递闭包引起的误识别的问题。算法的解决方法是将相似记录集视为一个完全子图，因此合并相似重复记录的问题就转换为在连通图中寻找完全子图。我们的方法是在每个滑动窗口中根据记录的相似性，建立连通图并寻找完全子图。算法将窗口中的第一条记录所对应的点视为一个可能的完全子图的第一个点，并将与该可能成为完全子图中的点全连接的点加入该完全子图中，重复如上步骤直到所有的记录都被处理完为止。同时，算法能够有效的避免已检测的完全子图的一部分被重复检测的情况。

算法在基于任意两个重复记录集没有两个或以上相同的记录的假设前提下，即任意两个完全子图只有共点，没有共边的情形，提出了针对这一特殊情况的改进算法。该算法的研究意义在与此情形具有普遍性。最后通过实验验证，算法在保持性能的基础上，减少运行时间，提高了算法效率。

第六章 总结与展望

6.1 总结

在信息爆炸的时代，人们渴求通过数据挖掘获取真实有效并能为决策提供支持的数据。而数据清洗作为数据挖掘技术中的重要环节，清洗质量的效果如何直接决定了数据挖掘的效果和质量。“脏数据”按其不同的表现形式可具体概括为不完整数据、相似重复数据和错误数据三种类型。其中由于多数据源合并而造成的信息重复是最关键的问题，因此重复信息的检测和清除是数据清洗的研究热点。本文主要做了以下几个方面的工作。

（1）介绍数据挖掘方面的相关知识。包括数据清洗的原理、方法、基本流程和主要工具，重点介绍重复记录检测相关知识和目前研究现状，包括字段匹配算法和重复记录清除算法。

（2）提出基于传递闭包的重复记录检测算法。在 SNM 算法的基础上提出如下改进：在排序步骤通过多趟排序使相似重复记录在物理位置上尽可能相近；通过引入判断机制，适时中止多趟排序的步骤，减少算法的复杂度；在记录匹配步骤，引入特定权值和有效权值的概念，提高记录匹配的准确率；最后在合并步骤，通过传递闭包对不同窗口中检测出的重复记录进行合并。

（3）提出基于属性分析的重复记录检测算法。在基于传递闭包的重复记录检测算法的基础上，提出了两个方面的改进，一是进行属性分析，并按照属性权重的高低对属性进行排序，二是引入过滤机制，减少字段匹配的次数。在保证正确率的同时，有效的提高了算法的效率，减少算法的运行时间。

（4）提出基于完全子图的重复记录检测算法。针对合并步骤传递闭包易引起误识别的问题。算法将相似记录集视为一个完全子图，将合并相似重复记录的问题转换为在连通图中寻找完全子图的问题。通过建立连通图并寻找完全子图来合并相似重复记录。算法在有效避免重复检测问题的同时，解决传递闭包引起的误识别问题，降低了算法的误识别率。

6.2 展望

由于研究时间和条件的限制，本文虽然对相似重复记录清洗问题提出了一些自己的改进算法，但仍然存在以下的不足之处：

（1）本文的记录检测算法能够较好的识别字符串缩写，字符串拼写错误和处理字符串顺序颠倒的情形，但是不能直接用于中文中，这一方面也有待进一步的研究。

（2）对基于完全子图的重复记录检测算法，因为构成完全子图的条件较为严苛，所以可能会增加一些不必要的比较操作，如何能够在保持现有检测效果的同时，减少比较的操作，仍是一个值得继续研究的问题。

参考文献

[1] 方幼林，杨冬青，唐世渭. 数据仓库中数据质量控制研究[J]. 计算机工程与应用，2003，39(13): 1-4.

[2] Rahm E, Do H H. Data Cleaning: Problems and Current Approaches [J]. IEEE Data Engineering Bulletin, 2000, 23(4): 3-13.

[3] Mong Li Lee, Wynne Hsu, Vijay Kothari. Cleaning the Spurious Links in Data [J]. IEEE Intelligent Systems, 2004, 19(2): 28-33.

[4] Wasito I, Mirkin B. Nearest Neighbour Approach in the Least-Squares Data Imputation Algorithms [J], Information Sciences, 2005, 169(1-2): 1-25.

[5] Galhardas H, Florescu D, Shasha D. Declarative Data Cleaning: Language, Model and Algorithms[C]. In: Proceedings of the 27th International Conference on Very Large Data Bases, Rome, Italy. 2001: 371- 380.

[6] Bitton D, De Witt D J. Duplicate Record Elimination in Large Data Files [J]. ACM Transactions on Database Systems, 1983, 8 (2):255- 265.

[7] Elma garmid A K, Ipeirotis P G, Verykios V S. Duplicate Record Detection: A Survey [J]. IEEE Transactions on Knowledge and Data Engineering, 2007, 19 (1): 1- 16.

[8] Panse F, Keulen M V, Keijzer A D and N Ritter. Duplicate Detection in Probabilistic Data[C]. In: Proceedings of the 26th International Conference on Data Engineering Workshop. 2010: 179-182.

[9] Batin C, Scannapieca M. Data Quality: Concepts, Methodologies and Techniques [M]. Springer, 2006: 230-235.

[10] Elfeky M G, Verykios V S. On Search Enhancement of the Record Linkage Process [C]. In: Proceedings of the KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation. 2003: 31-33.

[11] Lee M. L, Ling T. W, Lwo W. L. IntelliClean: A Knowledge-based Intelligent Data Cleaner [C]. In: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Ming. Boston: ACM Press, 2000: 290-294.

[12] Levenshtein I V. Binary Codes Capable of Correcting Spurious Insertions and Deletions of Ones [J]. Problems of Information Transmission, 1965, 1(1): 8 - 17.

[13] M.S. Waterman, T.F. Smith and W.A. Beyer. Some Biological Sequence Metrics [J]. Advances in Math. 1976, 20(4): 367-387.

[14] T.F. Smith and M.S. Waterman. Identification of Common Molecular Subsequences [J]. Journal Molecular Biology. 1981, 147(1): 195-197.

[15] M.A. Jaro. Unimatch: A Record Linkage System: User's Manual [M]. United States. Bureau of the Census, 1978: 34-36.

[16] J. R. Ullmann. A Binary n-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion, and Reversal Errors in Words [J]. The Computer Journal. 1977, 20(2): 141-147.

[17] E. Ukkonen. Approximate String Matching with Q-Grams and Maximal Matches [J]. Theoretical Computer Science, 1992, 92(1): 191-211.

[18] A.E. Monge and C.P. Elkan. The Field Matching Problem: Algorithms and Applications[C]. In: Proceeding of Second International Conference of Knowledge Discovery and Data Mining, 1996: 267-270.

[19] Cohen, William W. Integration of Heterogeneous Databases without Common Domains Using Queries Based on Textual Similarity [J]. SIGMOD Record, ACM Special Interest Group on Management of Data. 1998, 27(2): 201-212.

[20] Guo zhimao, Zhou aoying. Research on Data Quality and Data Cleaning: a Survey [J]. Journal of Software, 2002, 13(11): 2076-2083.

[21] Alvaro E. Monge. Matching algorithms within a duplicate detection system[J]. IEEE Data Engineering Bulletin, 2000, 23(4): 14-20.

[22] Robert L. Taft. Name Search Techniques [M]. Bureau of Systems Development, New York State Identification and Intelligence System. 1970: 1-118.

[23] Leicester E. Gill. OX-LINK: The Oxford Medical Record Linkage System[C]. In: Proceeding of International Record Linkage Workshop and Exposition, 1997: 15-33.

[24] L. Philips. Hanging on the Metaphone[J]. Computer Language Magazine, 1990, 7(12): 39-44.

[25] Lawrence Philips. The Double Metaphone Search Algorithm [J]. C/C++ Users Journal. 2000, 18(6): 38-43.

[26] Sudipto Guha, Nick Koudas, Divesh Srivastava. Reasoning About Approximate Match Query Results [C]. In Proceedings of the 22nd International Conference on Data Engineering. 2006,

8.

[27] 韩京宇，徐立臻，董逸生. 一种大数据量的相似记录检测方法[J]. 计算机研究与发展. 2005, 42（12）：2206-2212.

[28] 张永，迟忠先，闫德勤. 数据仓库ETL中相似重复记录的检测方法及应用[J]. 计算机应用，2006, 26(4)：880-882

[29] 邱越峰，田增平，周傲英. 一种高效的检测相似重复记录的方法.计算机学报[J]，2001, 24(1)：69-75

[30] 周宏广. 异构数据源集成中清洗策略的研究及应用[D]. 中南大学, 2004.

[31] 唐懿芳, 钟达夫, 严小卫. 基于聚类模式的数据清洗技术[J].计算机应用, 2004, 24(5) : 116- 119.

[32] Mc Callum A, Nigam K, UngarL H. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching [C] . In: Proceeding of the 6th ACM SIGKDD International Conference on Know ledge Discovery and Data Mining, Boston, Massachusetts, USA. 2000: 169- 178.

[33] 戴东波, 汤春蕾, 熊赟. 基于整体和局部相似性的序列聚类算法[J] . 软件学报, 2010, 21 (4) : 702- 717.

[34] MaD Y, Zhang A D. An Adaptive Density - Based Clustering Algorithm for Spatial Database with Noise[C]. In Proceedings of the 4th International Conference on Data Mining, Brighton, UK. 2004: 467- 470.

[35] Ram A, Sharma A, Jalal A S, et al. An Enhanced Density Based Spatial Clustering of Applications with Noise[C]. In: Proceedings of the IEEE International Conference on Advance Computing, Patiala, India. 2009: 1475 - 1478.

[36] 曾东海, 米红, 刘力丰. 一种基于网格密度与空间划分树的聚类算法[J]. 系统工程理论与实践, 2008, 28 (7): 125 - 131.

[37] Huang M, Bian F L. A Grid and Density Based Fast Spatial Clustering Algorithm [C]. In: Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence, Shanghai China. 2009: 260 - 263.

[38] Meng L L, Ren J D, Hu C Z. CABGD: An Improved Clustering Algorithm Based on Grid-Density[C]. In: Proceedings of the 4th International Conference on Innovative Computing, Information and Control, Kaohsiung, Taiwan. 2009: 381 - 384.

[39] Chen J B, Sun J, Chen Y F. A New Ant- Based Clustering Algorithm on High Dimensional Data Space [C]. Complex Systems Concurrent Engineering, 2007: 605- 611.

[40] Madeiro S S, Bastos- Filho C JA, Neto F B L, et al. Adaptative Clustering Particle Swarm Optimization[C]. In: Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing, Rome, Italy. 2009: 2257- 2264.

[41] Hernandez M A, Stolfo S J. Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem [J]. Data Mining and Knowledge Discovery, 1998, 2(1): 9- 37.

[42] Hernandez M, Stolfo S. The Merge/Purge Problem for Large Databases [C]. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, San Jose, California. 1995: 127- 138.

[43] BroderA Z, Glassman S C, Manasse M S, et al. Syntactic Clustering of the Web [J]. Computer Networks and ISDN Systems, 1997, 29(8): 1157- 1165.

[44] Monge A, Elkan C. An Efficient Domain Independent Algorithm for Detecting Approximately Duplicate Database Records[C]. In: Proceedings of the SIGMOD Workshop on Data Mining and Knowledge Discovery, Tucson, Arizona. 1997: 23- 29.

[45] Codd E F, Codd S B, Salley C T. Beyond decision support [J]. Computer World, 1993, 27(30): 87-89.

[46] 王光宏，蒋平. 数据挖掘综述[J]. 同济大学学报. 2004, 32(2): 246-252.

[47] Pawlak Z. Rough set [J]. International Journal of Information and Computer Science, 1982, 11(5): 341-356.

[48] Eckerson W W. Data quality and the bottom line：achieving business success through a commitment to high quality data[R].The Data Warehousing Institute，2002.

[49] Eppler M J，Algesheimer R，Dimpfel M. Quality Criteria of Content-driven Websites and Their Influence on Customer Satisfaction and Loyalty: An Empirical Test of An Information Quality Framework [C]. In Proceeding of MIT Conference of Information Quality, 2003: 108-120.

[50] V. Raman, J. M. Hellerstein. Potter's Wheel: An Interactive Framework for Data Transformation and Cleaning [C]. In Proceedings of the 27th VLDB conference, Roma, Italy, 2001: 10- 49.

[51] Chaudhuri S, Ganjam K, Ganti V. Data Cleaning in Microsoft SQL Server 2005 [C]. In

Proceedings of the ACM SIGMOD Conference, 2005: 918-920.

[52] Helena Galhardas, Daniela Florescu, Dennis shasha, Eric Simon. AJAX: An Extensible Data Cleaning Tool [C]. In Proceedings of the ACM SIGMOD on Management of data, 2000: 590.

[53] M. Hernandez, S. Stolfo. A Generalization of Band Joins and Merge/Purge Problem [M]. Department of Computer Science, Columbia University, 1995.

[54] Helena G. Generative and Transformational Techniques in Software Engineering [M]. Data cleaning and transformation using the AJAX framework. Berlin：Springer，2006: 327-335.

[55] April Alain, Coallier Francois. Trillium: A Model for the Assessment of Telecom Software System Development and Maintenance Capability[C]. In Proceeding of the 2nd IEEE International Engineering Standards Symposium, 1995: 175-183.

[56] Raman, V., Hellerstein, J. Potter's wheel: An Interactive Data Cleaning System[C]. In: Proceedings of the 27th International Conference on Very Large Data Bases. Roma: Morgan Kaufmann, 2001: 381-390.

[57] Monge A，E. Matching Algorithm within A Duplicate Detection System[J]. IEEE Data Engineering Bulletin，2000, 23(4):14- 20.

[58] Jonathan I. Maletic Andrian Marcus. Data Cleansing: Beyond Integrity Analysis[C]. In Proceeding of the International Conferecne on Information Quality，2000: 200- 209.

[59] Minton S N, Nanjo C, Knoblock C A, et al. A Heterogeneous Field Matching Method for Record Linkage[C]. In: Proceeding of the 5th IEEE International Conference on Data Mining, Houston, Texas, USA. 2005: 314- 321.

[60] LEVENSHTEIN V I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals [J]. Soviet Physics Doklady, 1966, 10(8): 707 - 710.

[61] LI Jian, ZHENG Ning. Improvement on The Algorithm of Data Cleaning based on MPN [J]. Computer Applications and Software, 2008, 25(2):245-246.

[62] Alvaro E.Monge, Charles P.Elkan. An Efficient Domain-independent Algorithm for Detecting Approximately Duplicate Database Records [D]. University of California, 1997.

[63] Kosmas Karadimitriou, John M. Tyler. The Centroid Method for Compressing Sets of Similar Images [J]. Pattern Recognition Letters. 1998, 19(7): 585-593.

[64] Agus Pudjijono. Probabilistic Data Generation. Master Thesis, Australian National University, 2008.