语义知识挖掘通用平台的设计与实现

张立邦

院 （系）：计算机科学与技术　　专　　业：计算机科学与技术

学　　号：1080310212　　指导教师：关毅

2012 年 **6** 月

哈爾濱工業大學

毕业设计（论文）

题　　目　语义知识挖掘通用平台的设计与实现

专　　业　计算机科学与技术

学　　号　1080310212

学　　生　张立邦

指 导 教 师　关毅

答 辩 日 期　2012 年 7 月 2 日

哈尔滨工业大学毕业设计（论文）评语

姓名：张立邦 学号：1080310212 专业：计算机科学与技术

毕业设计（论文）题目：语义知识挖掘通用平台的设计与实现

工作起止日期：2012 年 3 月 19 日起 2012 年 6 月 26 日止

指导教师对毕业设计（论文）进行情况，完成质量及评分意见：

指导教师签字： 指导教师职称：

评阅人评阅意见：

评阅教师签字： 评阅教师职称：

答辩委员会评语：

_____

_____

_____

根据毕业设计（论文）的材料和学生的答辩情况，答辩委员会作出如下评定：

学生 _____ 毕业设计（论文）答辩成绩评定为：_____

对毕业设计（论文）的特殊评语：

_____

_____

_____

_____

_____

_____

_____

_____

_____

答辩委员会主任（签字）：_____ 职称：_____

答辩委员会副主任（签字）：_____ _____

答辩委员会委员（签字）：_____ _____ _____ _____

_____ _____ _____ _____ _____

_____ _____ _____ _____ _____

_____

年 月 日

哈尔滨工业大学毕业设计（论文）任务书

姓　　名：张立邦	院　（系）：计算机科学与技术
专　　业：计算机科学与技术	班　　号：0803102
任务起至日期：　　2012　　年 3　月 19 日至　　2012 年　7 月　2　日	

毕业设计（论文）题目：语义知识挖掘通用平台的设计与实现

立题的目的和意义：

随着语义搜索的不断发展，从语料资源中自动挖掘语义信息，建设大规模的语义知识资源已成为一个热门的研究课题。其中，根据特定的规则，从语料中抽取出符合要求的语义知识，是最常用的方法之一。然而，不同的挖掘任务通常会使用具有不同形式、不同内容的规则，因而采用不同的实现方法，导致模块的可扩展性较差且难以维护。

本文从语义知识挖掘的两个实际案例即相关实体挖掘和实体语义标签挖掘入手，总结常用规则的内容和特点。据此，对规则进行了重新定义，并将规则的表达形式统一化，进而设计并实现了语义知识挖掘通用平台。该平台支持百度 query log、query session 和检索结果三种输入语料，为了完成不同的挖掘任务，用户只需在配置文件中对规则进行修改，而不必调整程序，从而极大地提高了挖掘效率，降低了维护成本，且使平台具有较好的可扩展性。

本文将平台划分为 9 个功能模块。其中，配置文件解析模块利用规则表达形式的特点，通过递归的方式对规则进行快速解析。挖掘模块采用了“先枚举所有可能的结果，再验证其是否符合规则”的抽取策略，从而避免了对复杂规则的直接演绎。针对“在大规模语料库中，规则的支持语句非常稀少”这一实际情况，根据规则的定义，提出了基于动态规划算法的过滤策略，在开始挖掘之前，将所有无用语料剔除，从而保证了挖掘模块的运行速度。

最后，针对相关实体挖掘和实体语义标签挖掘两个案例进行了实验，并以覆盖率、准确率和执行效率为指标对语义知识挖掘通用平台的挖掘效果进行了评估。评估结果表明，该平台达到了其预定的设计目标。

技术要求与主要内容：

技术要求：

（1） 所有模块使用 C++语言和 shell 脚本编码实现

（2） 部分模块需要在分布式计算系统上运行

（3） 使用百度抓取平台获得检索结果的前 N 条

主要内容：

从实验中总结语义知识挖掘任务中的常见需求，据此对规则的内容进行了归纳。对规则进行了重新定义，实现了规则表达形式的统一化，进而为实现语义知识挖掘通用平台打下基础。

根据通用平台的基本流程，将其划分为 9 个不同的功能模块，即初始配置文件生成模块、配置文件解析模块、配置文件解析模块（A）、配置文件解析模块（R）、用户词表预处理模块、语料预处理模块、NE 识别结果校正模块、挖掘模块和结果统计模块。并依次给出了各个模块的详细设计方案及实现方法。其中配置文件解析模块（A）、（R）抓住了规则表达形式上的特点，采用递归的方法对语义知识单元向量 A 和关系向量 R 的规则进行快速解析；挖掘模块使用了逆向思维，即先枚举可能的挖掘结果，再验证其是否满足规则，从而避免直接对复杂的规则进行演绎，设计并是实现了基于动态规划算法的过滤策略，将大部分无用语料直接过滤掉，从而保证了挖掘的效率。

最后，使用 C++语言和 shell 脚本完成了语义知识挖掘通用平台的编码实现，并针对相关实体挖掘和实体语义标签挖掘这两个任务进行了实验，实验结果表明该平台具有良好的通用性和较高的挖掘效率，达到了预期的设计目标。

进度安排：

2009/03/04 – 2010/03/15：完成关于相关实体挖掘的实验

2009/03/16 – 2010/03/31：完成关于实体语义标签挖掘的实验

2009/04/01 – 2010/04/30：完成语义知识挖掘通用平台的详细设计

2009/05/01 – 2010/05/31：完成配置文件解析模块、语料预处理模块的实现

2009/06/01 – 2010/06/15：完成挖掘模块、结果统计模块的实现

2009/06/16 – 2010/06/30：完成平台的功能测试，及详细设计文档

同组设计者及分工：

本项目由张立邦独自一人设计并实现。

指导教师签字_____

年　　月　　日

教研室主任意见：

教研室主任签字_____

年　　月　　日

摘　要

随着语义搜索的不断发展，从语料资源中自动挖掘语义信息，建设大规模的语义知识资源已成为一个热门的研究课题。其中，根据特定的规则，从语料中抽取出符合要求的语义知识，是最常用的方法之一。然而，不同的挖掘任务通常会使用具有不同形式、不同内容的规则，因而采用不同的实现方法，导致模块的可扩展性较差且难以维护。

本文从语义知识挖掘的两个实际案例即相关实体挖掘和实体语义标签挖掘入手，总结常用规则的内容和特点。据此，对规则进行了重新定义，并将规则的表达形式统一化，进而设计并实现了语义知识挖掘通用平台。该平台支持百度 query log、query session 和检索结果三种输入语料，为了完成不同的挖掘任务，用户只需在配置文件中对规则进行修改，而不必调整程序，从而极大地提高了挖掘效率，降低了维护成本，且使平台具有较好的可扩展性。

本文将平台划分为 9 个功能模块。其中，配置文件解析模块利用规则表达形式的特点，通过递归的方式对规则进行快速解析。挖掘模块采用了“先枚举所有可能的结果，再验证其是否符合规则”的抽取策略，从而避免了对复杂规则的直接演绎。针对“在大规模语料库中，规则的支持语句非常稀少”这一实际情况，根据规则的定义，提出了基于动态规划算法的过滤策略，在开始挖掘之前，将所有无用语料剔除，从而保证了挖掘模块的运行速度。

最后，针对相关实体挖掘和实体语义标签挖掘两个案例进行了实验，并以覆盖率、准确率和执行效率为指标对语义知识挖掘通用平台的挖掘效果进行了评估。评估结果表明，该平台达到了其预定的设计目标。

关键词：语义知识挖掘；通用平台；规则；统一化；动态规划算法

Abstract

With the continuous development of semantic search, automatic extraction of semantic information from the corpus resource and the establishment of large-scale semantic knowledge resources has become a hot research topic. Which, according to specific rules, extracting semantic knowledge from the sentences meet the constraint conditions, is one of the most commonly used methods. However, different data mining tasks usually has rules with various contents and expressions, and therefore different implementation methods. The modules have poor scalability and difficult to maintain.

This article from the semantic knowledge mining of two actual cases that the related entities mining and entity semantic labels mining, summarizes the content and characteristics of common rules. Accordingly, the rules were redefined, and will rule expression unification, then the design and implementation of semantic knowledge mining general platform. The platform supports query log, query session and search results from Baidu as three kinds of input data, in order to complete the various mining task, the user only need to change the rules in configuration file without adjustment of the code, thereby greatly improving the mining efficiency, reducing maintenance cost, and the platform has good scalability.

The platform is divided into 9 functional modules. Among them, the configuration file parsing module captures the characteristics of rules' expression form, then analyses them through a recursive algorithm. In order to avoid direct deduction of complex rules, the mining module will enumerate all the possible results firstly, then judge whether they conform to the rules. In large corpora, sentence supports the rules is very scarce. So we develop a filter strategy based on dynamic programming algorithm to eliminate all useless sentences.

Finally, we make experiments on cases of entities mining and entity semantic labels mining and evaluate our platform using index such as cover rate, accuracy and efficiency. The results of evaluation show that, the platform has reached its expected target.

Keywords: semantic knowledge, universal platform, rules, dynamic programming

目　录

第1章 绪 论

1.1 项目背景

随着互联网技术的飞速发展，信息以前所未有的速度不断地产生并传播着。这使得网络上的信息资源变得更加丰富、多样，但同时也出现了信息质量参差不齐、信息垃圾泛滥等不良现象。因此，基于简单的关键字匹配策略的传统搜索引擎，其准确率已无法很好地满足用户的检索需求，取而代之的将是从用户搜索意图本身入手的语义搜索引擎。百度推出的框计算、谷歌推出的知识谱图等，都是在语义搜索方面做出的探索与尝试。

知识库是语义搜索引擎进行推理和响应的基础和关键。语义知识库通常由两大部分组成：（1）Ontology（本体），即一种能在语义和知识层次上描述系统的概念模型，其目的在于以一种通用的方式来描述领域中的知识，提供对领域中概念的共同一致的理解，从而实现知识在不同的应用程序和组织之间的共享和重用[1]；（2）时效性资源，即内容和结构会随着时间的推移而不断发生明显变化的语义资源，如相关实体、实体的语义标签、新词资源等。如果把整个语义知识库看作地球，那么 Ontology 就相当于地球上的海洋和陆地，其构造在较长时间内是基本不变的；相对地，时效性资源更像是漂浮于地球表面上的大气层，是需要实时积累和更新的。两者都是不可或缺的，它们在作用上是相辅相成的。

大规模语义词典和知识库通常依靠人工构建完成，但在建设和维护过程中都要耗费大量的人力物力。因此，从语料资源中自动抽取语义信息，建设大规模的语义知识资源已成为一个热门的研究课题[2]。

1.2 项目来源

该项目来源于百度时代网络技术（北京）有限公司自然语言处理部语义知识挖掘组，主要针对时效性语义资源的自动挖掘进行研究。百度已成为全球用户数量最多的中文搜索引擎，每天有数以亿计的检索 query，这些 query 及与其对应的检索结果中，都含有丰富的时效性语义资源。以用户检索 query 和百度搜索引擎返回的检索结果为原始语料，根据公司内部的对技术积累和实际应用的需求，本文先后进行了基于 query log 的相关实体挖掘、基于 query session 的相关实体挖掘、音乐类相关实体挖掘和基于语义标签的实体推荐等多次语义知识挖掘方面的实验，并从这些实验中析取和总结需求，进而设计并实现了语义知识挖掘通用平台。

1.2.1 相关实体挖掘

相关实体挖掘在很多方面有着潜在应用，具体包括（1）百度应用中的相关应用推荐；（2）百度百科词条关系化构建；（3）百度百科词条内链效果改善；（3）时效性阿拉丁（展现动态变化的实体间关系）；（4）实体类 query 检索结果展现等。在挖掘出相关实体的同时，还需要同时抽取出每对相关实体的关系描述，如“刘德华”和“朱丽倩”的关系描述“结婚”，以及“刘德华”和“张学友”的关系描述“四大天王”等，以便于用户更好地了解相关实体的具体关系类型是什么。

本文假设在同一个句子中共现的两个实体具有一定的相关性。为了保证时效性，程序需要每天定时挖掘并积累数据，主要包括以下四个步骤：

（1）语料预处理：将语料库中的语料拆分成句，并进行包括分词、词性标注和 NE（命名实体）识别在内的一系列处理

（2）候选实体对及其关系描述词的抽取：以句子为单位，从中抽取出任意一对共现的实体作为候选相关实体对，并根据词性、距离限定等规则抽取出它们的关系描述词

（3）对候选相关实体对进行统计、整合和排序：将内容相同的实体对合并去重，并将其频次累加，再根据实际需求，按照关键字和频次对其进行排序

（4）对整合后的相关实体对及其关系描述词进行过滤：包括在词形及词性上的过滤、基于候选词表的过滤、基于停用词表的过滤和基于抓取平台的过滤等

其流程图如下：

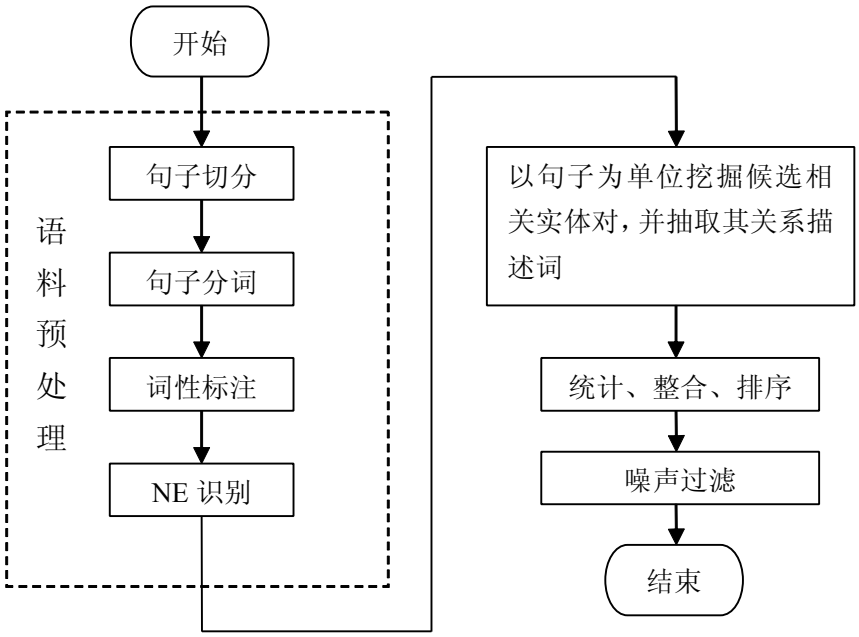

图 1-1 相关实体挖掘流程图

根据原始语料和需求类型的不同，相关实体挖掘又分为“基于 query log 的相关实体挖掘”、“基于 query session 的相关实体挖掘”和“音乐类相关实体挖掘”。

（一）基于 query log 的相关实体挖掘

该部分实验以 query log 为语料。由于 query 的长度很短，在一条 query 中同时出现的两个实体往往具有一定的相关性。因此，适合从 query log 中抽取长度较短的相关实体。在原始语料中，每条 query 是独占一行的，因此，在进行语料预处理（分词、词性标注、NE 识别）之后，便可以直接以行为单位进行候选相关实体对及其关系描述词的抽取。表 1-1 给出了 query log 预处理和候选相关实体对抽取的示例。

表 1-1 query log预处理与候选相关实体对抽取示例

query	刘德华和朱丽倩已经结婚
分　　词	刘德华　和　朱丽倩　已经　结婚
词性标注	刘德华[nr]　和[c]　朱丽倩[nr]　已经[d]　结婚[v]
NE 识别	刘德华[nr][PER]　和[c][NOR]　朱丽倩[nr][PER] 已经[d][NOR]　结婚[v][NOR]
抽取结果	相关实体对：（刘德华，朱丽倩），关系描述词：结婚

根据对实验结果的评估，为了保证准确率，抽取之后还需要进行以下几步噪声过滤：（1）过滤单字 NE；（2）过滤包含标点符号的 NE，但会保留外国人名中常见的符号“.”和“-”；（3）过滤以数字开头的人名、地名；（4）对类型为 PER、ORG、LOC 的实体，只保留词性为名词相关以及缩略词的 NE，即词性标注结果为“n”、“nr”、“ns”和“j”的 NE。

对于这部分实验，本文以人名为轴，展开相关实体的挖掘，包括 PER-LOC（人名-地名）、PER-PER（人名-人名）、PER-ORG（人名-机构名）、PER-NVL（人名-小说名）、PER-SNG（人名-歌曲名）、PER-VDO_MVE（人名-电影名）、PER-VDO_TV（人名-电视剧名）和 PER-VDO_TVSHOW（人名-电视节目名）类型的相关实体对。

（二）基于 query session 的相关实体挖掘

该部分实验以 query session（一个 session 为一个用户在一定时间内连续搜索的 query 的集合）为语料。对用户行为的分析表明，在同一个 query session 中出现的同类型实体，它们之间通常具有较强的关联。这是因为用户经常会连续地搜索同一系列的电影、风格相近的歌曲、类型相同的游戏和功能互补的软件等。当然它们也可能仅仅是同一时期的热门，针对这种 case，需要在排序结果的时候对其

进行适度的打压。在原始语料中，每个 session 独占一行。由于本文所使用的分词、词性标注和 NE 识别工具具有动态识别的功能，同一个语句在不同的上下文中会有不同的处理结果，所以如果直接对 session 进行预处理，会导致结果偏离 session 中每条独立 query 原本的语义特征。因此，需要先将 session 中的每条 query 切分成独立的一行，进行预处理之后，再将其结果合并为一行，然后再进行候选相关实体对的抽取。表 1-2 给出了 query log 预处理和候选相关实体对抽取的示例。

表 1-2 query session预处理与候选相关实体对抽取示例

query session	唐伯虎点秋香下载\t 武状元苏乞儿下载
session 切分	唐伯虎点秋香下载 武状元苏乞儿下载
预处理	唐伯虎点秋香[n][VDO_MVE]　下载[v][NOR] 武状元苏乞儿[n][VDO_MVE]　下载[v][NOR]
session 还原	唐伯虎点秋香[n][VDO_MVE]　下载[v][NOR]　武状元苏乞儿[n][VDO_MVE]　下载[v][NOR]
抽取结果	相关实体对：（唐伯虎点秋香，武状元苏乞儿）

由于 NE 识别错误，抽取出来的相关实体存在噪声，因此还需要对结果进行过滤。过滤的方法主要包括两部分：（1）基于停用 NE 表过滤：在抽取结果统计实体的频次，得到高频的停用词；（2）基于词性的过滤：观察实验结果，发现人名容易被识别为小说、歌曲等，所以将词性为“nr”的 NE 过滤掉。

在这部分实验中，本文主要针对 NVL-NVL（小说名-小说名）、SNG-SNG（歌曲名-歌曲名）、VDO_MVE-VDO_MVE（电影名-电影名）、VDO-TV- VDO-TV（电视剧名-电视剧名）、SFT-SFT（软件名-软件名）和 GME-GME（游戏名-游戏名）这六种类型的相关实体进行挖掘。

（3）音乐类相关实体挖掘

人们在聆听音乐时，当一首歌结束之后，往往希望可以继续试听该歌手的其他热门歌曲，也可能想要尝试与该歌手相关的其他歌手的音乐作品。因此，在音乐类 APP 中，实现同一歌手的歌曲按热度排序以及不同歌手之间的相关推荐是非常必要的。针对这一实际应用中的需求，本文在“基于 query log 的相关实体挖掘”的基础上设计并实现了音乐类相关实体的线上挖掘模块，只挖掘“歌手名-歌手名”和“歌手名-歌曲名”类型的相关实体对。其应用情况如图 1-2 和图 1-3 所示：

图 1-2 音乐类相关实体挖掘应用示意图1

图 1-3 音乐类相关实体挖掘应用示意图2

在图 1-2 中，用户正在使用百度音乐收听歌手 Eminem 的歌曲《Fubba Cuba》，其右侧推荐的相关应用为 Eminem 近期的热门歌曲《Love The Way Lie》、《Lighters》、《Stan》、《Without Me》和《Lose Yourself》等。

在图 1-3 中，在用户收听 Eminem 的歌曲的同时，应用界面的右侧推荐了其相关歌手 50Cent、Lil wayne、Dr Dre、Lady Gaga 和 2Pac 等。其中 50Cent、Lil wayne、Dr Dre、2Pac 与 Eminem 都是世界著名的说唱歌手。不仅如此，50Cent 曾由 Eminem 提拔并走红，Dr Dre 是 Eminem 的恩师，Lil wayne 曾多次与 Eminem 在歌曲中合作，2Pac 与 Eminem 被公认为两个时代的说唱天王，而 Lady Gaga 的一些歌曲曾由 Eminem 重新制作和混音，而且在格莱美颁奖典礼上，Eminem 也曾作为嘉宾为 Lady Gaga 颁奖。由此可见，相关应用的推荐结果是非常合理的。

由于百度音乐的资源有限，并不是所有的抽取结果都应该在推荐栏展示出来，需要根据应用的资源列表对相关实体的挖掘结果进行过滤，仅保留那些在实际应用中有对应资源的歌手和歌曲的名称。此外，还需要使用停用词表来过滤掉少数特殊 cases 导致的噪声。

1.2.2 基于语义标签的实体推荐

实体相关的 query 在搜索引擎的全部 query 中占有很大比重。其中的一类 query 即是搜索满足属性约束的某种类型的实体。如“战争 电影”，即是搜索战争（限定）题材的电影（实体类型）；“澳大利亚 动物”是搜索澳大利亚（限定）的动物物种（实体类型）；“减肥 蔬菜”则是搜索能够减肥（限定）的蔬菜（实体类型）等。这类 query 的共同特点是用户想搜索满足特定需求的一类实体，但并不明确要搜索的实体具体是什么，而这时倘若搜索引擎可以直接推荐出满足用户需求的实体列表，则将显著提高用户获取信息的效率，从而改善用户体验。具体示例如图 1-4 所示：

图 1-4 特定属性实体推荐示例

实现该功能的一个直接的想法是将其看作自动问答的问题，即从一个大规模的语料库（如互联网）中匹配用户的 query，然后抽取出满足用户需求的实体类型的实体。这种做法的主要问题是召回率较低。假设基于搜索引擎来实现上述过程，且我们想搜索的是“说唱歌手”，则我们可以用“说唱歌手”作为 query，利用搜索引擎返回含有“说唱歌手”的所有网页，进而从中挖掘 query 周围出现的人名。然而遗憾的是，由于搜索引擎通常对返回结果的数量进行了限制（如至多 N 条），采用这种方法所得到的结果的覆盖率将会受到很大的局限。当然，也有少数的热门 query，如“惊悚电影”、“伤感歌曲”等，网络上已经有整理得比较全面的列表，并不需要再去自动汇总。

在该部分实验中，本文转换了一下思路，即不是从 query 出发去搜索满足要求的实体，而是事先对语料（搜索引擎返回的检索结果）进行离线处理，为实体打上语义标签（如“运动员”类型的实体“林丹”的语义标签包括“羽毛球”、“奥运”、“亚运会”、“苏迪曼杯”等，其中的每一个语义标签都可以看作是对“林丹”的限定或者描述），进而形成实体知识库，然后在搜索阶段直接从实体知识库中匹配给定类型且满足 query 限定的实体。

可见，本实验的关键在于实体语义标签的挖掘，因为这些标签的质量的好坏

将直接影响实体推荐的效果。实体语义标签的挖掘主要包括以下四个步骤：

（1）语料预处理

包括对原始语料的断句、分词、词性标注和 NE 识别。

（2）挖掘给定实体类型的候选语义标签

设 E 为静态实体资源中的一个 NE 类型（如“运动员”），其中包括 n 个实体 $\{e_1, e_2, ..., e_n\}$，则需要对上述每个 e_i 抽取语义标签。严格上讲，任何词都有可能作为一个语义标签，然而并不是所有可能的语义标签都是有抽取价值的。本实验所面向的应用是实体推荐，即推荐满足给定语义限定和实体类型的实体列表。因此在本实验中，只需使用那些被用户所关心的实体限定词作为给定实体类型的候选语义标签。于是，需要通过挖掘用户数据来分析对于类型 E，哪些语义限定是用户最感兴趣的。本文使用 query log 作为语料库，从中搜索满足“*E”形式并且分词后 term 个数不多于 4 个的 query，其中“E”为实体类型词，如“电影”，“*”则表示任意其他词。我们将 query 中 E 前面的那些词作为该实体类型的候选语义标签，并使用一定的规则对其进行过滤。

（3）挖掘给定实体的候选语义标签

对于给定的静态目标实体 e_i，首先以 e_i 为 query 在百度中进行搜索，并通过抓取平台获得去重后的前 100 条检索结果（网页的标题和摘要）；再从每一条标题和摘要中，查找与 e_i 共现的、实体类型 E 的候选语义标签 c_j，并统计其频次；最后，将与实体 e_i 共现次数大于 1 的标签 c_j 抽取出来，作为 e_i 的候选语义标签。由此得到 e_i 的候选语义标签集合 $C_i = \{c_1, c_2, ..., c_m\}$。

然而，有一些语义标签难以从实体自身的检索结果中抽取出来，如“动作电影”中的“动作”。针对这种情况，本文采用如下策略来对实体 e_i 的候选标签集合进行补充：将“标签 实体类型”（如“动作 电影”）作为 query 在百度中进行搜索；保留去重后的前 100 条检索结果中同时含有“标签”和“实体类型”的标题和摘要，如摘要“热门动作电影：白蛇传说 变形金刚 3 大闹天宫 大武生 功夫熊猫 2 赤裸羔羊 龙门飞甲”；若 e_i 和 c_j 在其中共现，则将 c_j 加入到 e_i 的候选语义标签集合 C_i 中作为补充。

（4）实体候选语义标签的过滤

采用上述方法抽取出的句子尽管都含有“实体-标签”对 (e_i, c_j)，但是其中含有大量噪声，即很多的 (e_i, c_j) 只是恰好出现在相同的句子里面，二者之间并不具有限定的语义关联。因此，还需要对实体的候选标签进行过滤。但由于该过滤策略不在本文的研究范文之内，故不再赘述。

实体语义标签挖掘的整体流程如图 1-5 所示：

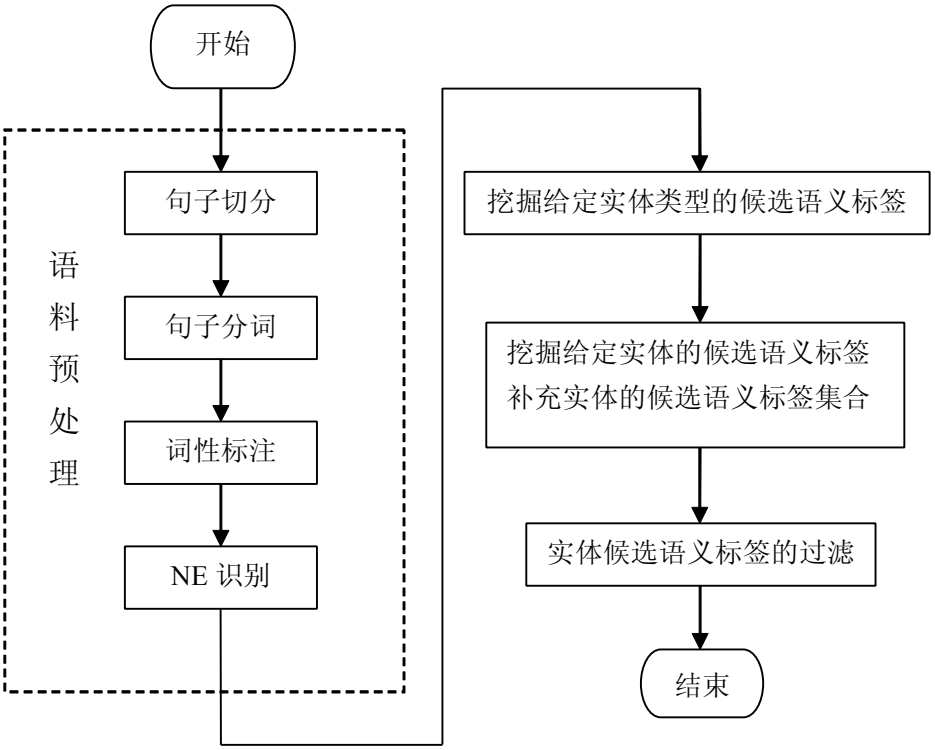

图 1-5 实体语义标签挖掘流程图

目前，本文仅针对游戏、漫画、电影和音乐这四类实体进行了语义标签挖掘的实验。

1.2.3 语义知识挖掘通用平台

从前面的实验内容不难看出，无论是相关实体的挖掘，还是实体语义标签的挖掘，其方法都是基于规则的。这些规则的主要包括：共现、词性、NE 类型、词的字面内容（如前缀、后缀、子串）、词的长度、词的相对位置、词的枚举（如候选词表、停用词表）和词的上下文内容等。然而，在不同的应用中，所用到的具体规则是不同的，而这些规则又分别被写在了不同的程序中，导致各个模块的可扩展性较差，且不便于维护。因此，将规则的表达形式统一化，进而开发出具有通用性的、高效的语义知识挖掘平台，将大大降低今后开发和维护类似模块所需的时间和成本。

1.3 平台的设计目标

（1）程序运行所需的环境变量、工作路径和数据位置可以在配置文件中修改

（2）所有的常用规则能够使用统一的形式进行表达并可以在配置文件中修改

（3）程序能够正确地读取、解析配置文件，对拼写错误和格式错误给出提示
（4）支持基于多种语料库的挖掘，包括 query log、query session 和检索结果
（5）支持不同维度（1-9）的语义知识单元向量（参见第 2 章第 1 节）的挖掘
（6）对于同一种语料，程序运行一次可以同时完成多个不同的数据挖掘任务如：同时从 query log 中挖掘音乐类相关实体和给定实体类型的语义标签
（7）可以正确地完成之前的四个实验中所有的挖掘任务并在功能上有所扩展
（8）为了实现平台的通用性，其挖掘效率与原来的每一个独立模块相比必然有所降低，但不能低于原模块效率的 80%
（9）平台具有较好的稳定性和健壮性，运行中发生异常时可立即退出并报错
（10）程序运行时不应占用过多的系统资源，如内存、外存、分布式节点等

1.4 本文主要研究内容及章节安排

本文提出了语义知识挖掘通用平台的设计方案和实现方法，并给出了针对实际应用案例的实验结果。具体的章节安排如下：

第一章为绪论，简要介绍了项目背景、项目进行之前所开展的调研工作以及项目的设计目标和意义。

第二章给出了该平台对规则的定义、规则的具体内容，进而给出了规则的表达形式，实现了其统一化，为实现平台的通用性打下了基础。

第三章为平台的总体设计，根据基本使用流程，将平台划分为 9 个功能模块，简要介绍了各个模块在系统中的作用，并给出了其目录结构。

第四章为平台的详细设计，给出了各个模块的详细设计方案及实现方法，并在挖掘模块中提出了基于动态规划算法的过滤策略，对无用语料进行过滤。

第五章为实验与评价，针对绪论中的相关实体挖掘和实体语义标签挖掘进行了实验，并以覆盖率、准确率和执行效率对平台的挖掘效果进行评估。

最后，在结论中对平台的设计方案、实现方法和评估结果进行了总结。

第 2 章规则表达形式的统一化

2.1 规则的定义

设挖掘任务 T 在对句子 S 进行分词之后，需要从中抽取出 n 个共现的词，它们在 S 中自左向右依次为 $a_1, a_2, ..., a_n$，定义 $A=(a_1, a_2, ..., a_n)$ 为 n 维的语义知识单元向量；设 S 中与 A 对应的上下文自左向右依次为 $r_0, r_1, r_2, ..., r_n$，定义 $R=(r_0, r_1, r_2, ..., r_n)$ 为 n 维的关系向量。则有 $S=r_0a_1r_1a_2r_2...a_nr_n$，其中 $a_i(1\le i\le n)$ 必须是 S 的分词结果中的一个词，而 $r_i(0\le i\le n)$ 可能是连续的若干个词的组合，也可能为空。例如：令句子 S =“刘德华和朱丽倩结婚了吗”，其分词结果为“刘德华 / 和 / 朱丽倩 / 结婚 / 了 / 吗”，若 $A=$(刘德华, 朱丽倩)，则 $R=(\phi$, 和, 结婚了吗)，其中 ϕ 代表 r_0 为空，即“刘德华”位于句子 S 的开端；令句子 S 为“好像刘德华朱丽倩结婚了”，其分词结果为“好像 / 刘德华 / 朱丽倩 / 结婚 / 了”，若 $A=$(刘德华, 朱丽倩)，则 $R=(好像, \phi$, 结婚了)，而 $r_1=\phi$ 则表示“刘德华”和“朱丽倩”这两个词在句子 S 中是紧邻的。

本文将规则定义为挖掘任务 T 对向量 A 和 R 的约束条件。挖掘模块需要在大规模语料库中选择同时满足针对向量 A 和 R 的规则的句子 S 作为 support sentence（支持语句），并将与 S 对应的向量 A 作为结果输出。

2.2 规则的内容

通过分析绪论中所提到的实验内容，可以得出对向量 A 和 R 的约束条件主要分为两大类：（1）对字面内容的约束，包括对字符串内容、长度、前缀、后缀和子串的限制等（2）对语义特征的约束，包括对分词后 term 的个数、词性和 NE 类型的限制等。而对词的相对位置的约束，已经被包含在向量 A 和 R 本身的定义当中。向量 A 和 R 的具体约束条件如表 2-1 和表 2-2 所示：

表 2-1 向量A和R的约束条件列表1

约束条件类型	约束条件表达式	约束条件内容
针对向量A和R的分量的字面内容的约束条件	string = 'parameter'	字符串必须等于 parameter
	string != 'parameter'	字符串不能等于 parameter
	prefix = 'parameter'	前缀必须是 parameter
	prefix != 'parameter'	前缀不能是 parameter
	suffix = 'parameter'	后缀必须是 parameter
	suffix != 'parameter'	后缀不能是 parameter
	substring = 'parameter'	必须包含子串 parameter
	substring != 'parameter'	不能包含子串 parameter
	text_length = '[a, b]'	长度必须大于等于 a 且小于等于 b
	candidate_list = 'parameter'	必须在候选列表 parameter 中
	stop_list = 'parameter'	不能在停用列表 parameter 中
	prefix_candidate_list = 'parameter'	至少有一个前缀在候选列表 parameter 中
	prefix_stop_list = 'parameter'	任何前缀都不能在停用列表 parameter 中
	suffix_candidate_list = 'parameter'	至少有一个后缀在候选列表 parameter 中
	suffix_stop_list = 'parameter'	任何后缀都不能在停用列表 parameter 中
	substring_candidate_list = 'parameter'	至少有一个子串在候选列表 parameter 中
	substring_stop_list = 'parameter'	任何子串都不能在停用列表 parameter 中

表 2-2 向量A和R的约束条件列表2

约束条件类型	约束条件表达式	约束条件内容
针对向量A和R的分量的语义特征的约束条件	term_count = '[a, b]'	分词后的 term 数必须 大于等于 a 且小于等于 b
	pos_candidate_set = '[c1,c2,...,ck]'	组成该分量的所有词的词性 必须在 c1,c2,...,ck 的范围之内
	pos_stop_set = '[c1,c2,...,ck]'	组成该分量的所有词的词性 不能是 c1,c2,...,ck 中的任何一个
	ner_candidate_set = '[c1,c2,...,ck]'	组成该分量的所有词的 NE 类型 必须在 c1,c2,...,ck 的范围之内
	ner_stop_set = '[c1,c2,...,ck]'	组成该分量的所有词的 NE 类型 不能是 c1,c2,...,ck 中的任何一个
针对向量A的子向量的约束条件	candidate_list = '[(c1,c2,...,ck), (parameter)]'	由 A 的第 c1、c2、...、ck 分量构成的子向量 必须在候选列表 parameter 中
	stop_list = '[(c1,c2,...,ck), (parameter)]'	由 A 的第 c1、c2、...、ck 分量构成的子向量 不能在候选列表 parameter 中

对向量A的上下文的整体约束	支持针对向量 A 和 R 的分量的所有约束条件的表达式	上下文整体为 R 的各个分量的和，其约束内容与单个分量是相同的

2.3 规则的表达

将向量 A 或 R 的分量的规则用如下形式进行表达：

```
a(r) = {
    0 : constraint0;
    1 : constraint1;
    ......
    LE : logical expression;
}
```

图 2-1 分量的规则表达形式

其中 0、1 为约束条件的序号；LE 代表约束条件的与或逻辑表达式，例如：若该要求分量必须满足条件 0，且满足条件 1，则表示为“LE : (0 & 1);”，若要求该分量满足条件 0，或满足条件 1，则表示为“LE : (0 | 1)”，也可以通过括号的嵌套来描述约束条件之间更加复杂的逻辑关系，如‘LE : ((0 & 1) | (2 & 3))”。

进一步地，可以得出向量 A 和 R 的规则的表达方式，分别如图 2-2 和图 2-3 所示：

```
A = {

    am = {
        ......
    }

    a1 = {
        ......
    }

    a2 = {
        ......
    }

    ......

    an = {
        ......
    }

}
```

```
R = {

    rm = {
        ......
    }

    r0 = {
        ......
    }

    r1 = {
        ......
    }

    ......

    rn = {
        ......
    }

}
```

图 2-2 A的规则表达形式　　图 2-3 R的规则表达形式

在图 2-2 中，“am”是一个虚拟分量，am 表示所有对 A 的子向量的约束的集合，它的表达形式和 A 的实分量相同，但只支持“candidate_list = '[(c1,c2,...,ck), (parameter)]'”和“stop_list = '[(c1,c2,...,ck), (parameter)]'”两种约束条件表达式。

根据平台的设计目标（2）和（6），配置文件中可能写入多个 A 和 R 的规则，将不同的 A 依次编号为 1，2，.....，R 同理。此外，不同的 A 和 R 之间还必须形成映射关系，这样才能支持多个任务的同时挖掘。为此，在 A 中增加一个虚拟分量“at”，它的表达形式如图 2-4 所示：

```
at = {
    to = '[id1, id2, ... ]';
}
```

图 2-4 虚拟分量at的规则表达形式

它表示当前的 A 向量是与编号为“id1、id2、... ”的 R 向量相对应的。每一个向量 A 至少与一个向量 R 对应。更进一步地，便得出语义知识单元向量 A 和关系向量 R 的规则的最终表达形式分别如图 2-5 和图 2-6 所示：

```
A1 = {

    at = {
        ......
    }

    am = {
        ......
    }

    a1 = {
        ......
    }

    ......

    an = {
        ......
    }

}

A2 = {

    at = {
        ......
    }

    am = {
        ......
    }

    a1 = {
        ......
    }

    ......

    an = {
        ......
    }

}

......
```

图 2-5 向量A的最终规则表达式

```
R1 = {

    rm = {
        ......
    }

    r0 = {
        ......
    }

    r1 = {
        ......
    }

    ......

    rn = {
        ......
    }

}

R2 = {

    rm = {
        ......
    }

    r0 = {
        ......
    }

    r1 = {
        ......
    }

    ......

    rn = {
        ......
    }

}

......
```

图 2-6 向量R的最终规则表达式

第 3 章平台的总体设计

3.1 平台的基本流程

根据平台的设计目标，平台的运行过程主要包括以下五个步骤：

（1） 运行程序，生成初始配置文件

（2） 用户根据自己的需求，手动修改配置文件

（3） 运行程序，程序对配置文件进行读取和解析，如果发现错误，转至（2）

（4） 程序对大规模语料库进行断句、分词、词性标注、NE 识别等预处理

（5） 程序从预处理结果中挖掘符合规则的语义知识单元向量，并输出结果

其流程如图 3-1 所示：

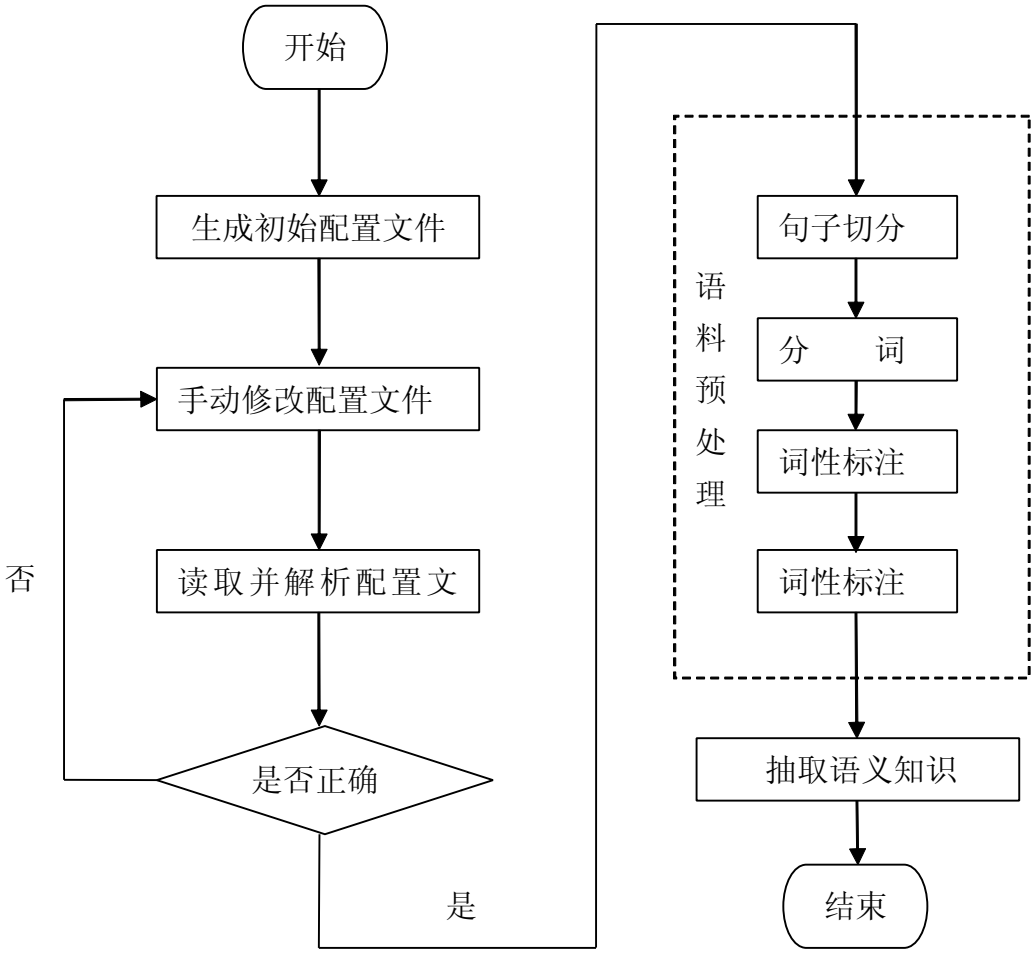

图 3-1 平台基本流程示意图

3.2 平台的模块设计

分析平台的基本流程可知，平台至少需要由初始配置文件生成模块、配置文件解析模块、语料预处理模块和语义知识挖掘模块四部分组成。

为了方便用户区分和程序的解析，将配置文件分为三个：（1）config，用于添加和修改程序运行所需的环境变量、工作路径、语料来源、数据存放的位置等；（2）config_of_r，用于添加和修改针对向量 R 的规则；（3）config_of_a，用于添加和修改针对向量 A 的规则。因此，配置文件生成模块需要同时生成这三个初始配置文件，而配置文件的解析应该由三个不同的模块来分别完成。

由于现有的 NE 识别技术其效果并不是非常理想，在某些特定的语境中，一些常见的实体名会被切散，如电影名“不能说的秘密”可能被切分成“不能说的 / 秘密”，也可能未被识别出到底是什么类型的实体，如识别结果为“不能说的秘密[n][NOR]”，这将会导致与命名实体有关的挖掘任务的召回率偏低，例如：在音乐类相关实体挖掘中，如果某些热门的歌手名、歌曲名没有被成功召回，将严重影响其相关推荐的效果。因此，增加一个对 NE 识别结果进行校正的模块，使得用户可以根据自己的实际需求，保证某些重要的实体名被召回，是非常必要的。

另外，规则中有很多关于用户词表的操作，如“candidate_list”、“stop_list”等，如果在挖掘模块开始运行时才发现有些用户词表的路径书写错误，这时再退出程序，无疑会对时间和资源造成很大的浪费。因此，还需要增加一个用户词表预处理模块，在语料预处理模块开始运行之前排除用户词表路径书写错误的问题。

最后，需要对挖掘的结果进行去重、整合和统计等操作，这部分工作需要由单独的结果统计模块来完成。

综上所述，除总控程序之外，语义知识挖掘通用平台共有 9 个不同的功能模块，即“初始配置文件生成模块”、“配置文件解析模块”、“配置文件解析模块（A）”、“配置文件解析模块（R）”、“用户词表预处理模块”、“语料预处理模块”、“NE 识别结果校正模块”、“挖掘模块”和“结果统计模块”。据此得出该平台的更为详细的运行流程，如图 3-2 所示：

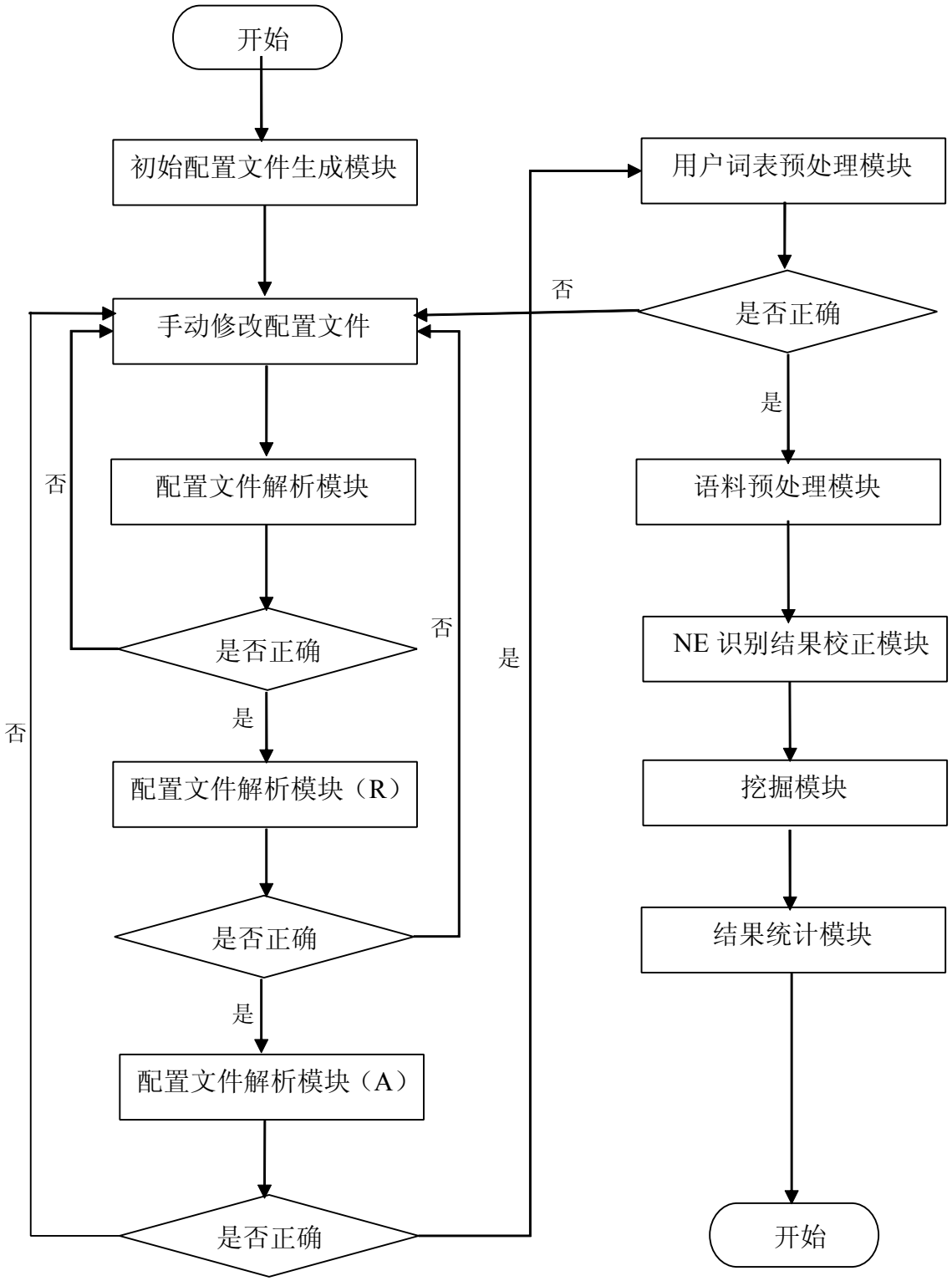

图 3-2 平台详细流程示意图

3.3 平台的目录结构

平台的目录结构如图 3-3 所示：

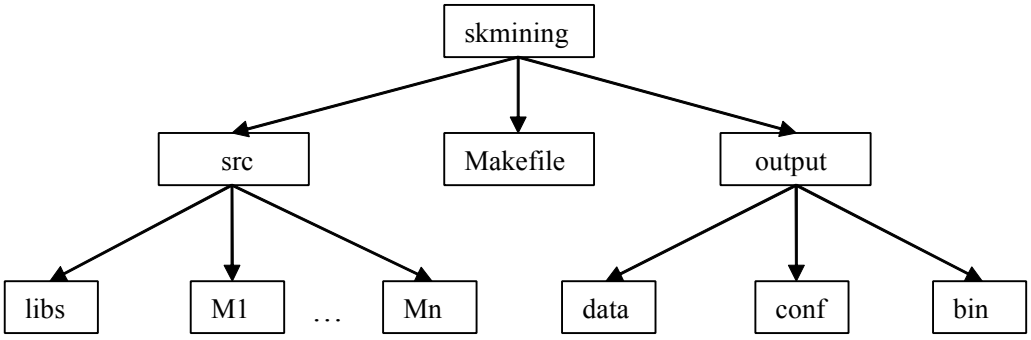

图 3-3 平台目录结构示意图

其中src/libs用于存放一些已经封装好的基本函数库，src/M1，…，src/Mn中存放各个模块的源代码；output/data中存放本地机器上的临时数据和中间数据，output/conf中可存放配置文件，output/bin中存放所有的可执行文件，包括脚本。在skmining目录下运行make命令，可以直接更新output/bin中的可执行文件。

3.4 平台的技术要求

该平台在 linux 环境下进行开发。由于输入数据为大规模语料库，为了保证执行效率，各个模块需要用 C++语言来编码实现，其中语料预处理模块、NE 识别结果校正模块和挖掘模块还需要通过 shell 脚本与分布式计算系统（本文采用 hadoop 分布式系统基础架构）进行交互。

第 4 章平台的详细设计与实现

4.1 总体逻辑控制模块

总体逻辑控制模块用于连接整个平台的各个功能模块，采用 C++语言进行编写，其生成的可执行文件命名为 skmining，它将直接与用户进行交互。其具体流程如下：

（1）当用户执行./skmining -H 时候，程序输出用户提示信息，给出参数说明

（2）当用户执行./skmining -S n PathOfConfig 时（其中 n 为向量 A 的维度，PathOfConfig 为配置文件所在的路径），程序调用初始配置文件生成模块，在目录 PathOfConfig 下生成三个初始配置文件 config、config_of_r、config_of_a，再由用户进行修改

（3）当用户执行./skmining PathOfConfig 时，程序调用配置文件解析模块，读取路径 PathOfConfig 下的 config，并解析，若有拼写或格式错误，退出程序并报错，否则将 config 存入内存；调用配置文件解析模块（R），读取并解析 config_of_r，若有拼写或格式错误，退出程序并报错，否则将 config_of_r 存入内存；调用配置文件解析模块（A），读取并解析 config_of_a，若有拼写或格式错误，退出程序并报错，否则将 config_of_a 存入内存；调用用户词表预处理模块，若某些词表路径填写错误，或文件不存在，则退出程序并报错；调用语料预处理模块对语料进行断句、分词、词性标注和 NE 识别；调用 NE 识别结果校正模块对上一个模块的输出进行调整；调用挖掘模块，完成挖掘任务；调用结果统计模块，对挖掘模块的输出进行统计、去重和整合，其输出为最终结果。

4.2 初始配置文件生成模块

该模块的作用是，根据用户输入的命令（命令./skmining –S n PathOfConfig），将事先存于内存中的初始配位置文件分别输出到 config、config_of_r 和 config_of_a 三个文件中，并保存到目录 PathOfConfig 下。该模块采用 C++语言实现。

4.2.1 config

config 中需包含向量 A 的维度，默认为 n（命令./skmining –S n PathOfConfig）；由于语料预处理模块、NE 识别结果校正模块、挖掘模块需要在分布式计算系统上运行，因此 config 中必须包含 hadoop 机群的客户端路径和用户在 hadoop 机群上的工作目录；为了使平台具有可移植性，即可以在不同的机器上、不同的目录下

运行，config 中需包含程序在本地机器上的位置；为了避免中间数据在 hadoop 机群上发生冲突，不同的任务需要有不同的名称，程序会自当将其加入到路径中，以保证各个任务的中间数据的存放位置没有重叠；在挖掘语义知识的过程中，规则的设计往往很难一次就满足要求，用户需要观察挖掘结果，不断地对规则进行改进，才能使准确率和召回率达到期望，所以 config 中需要包含一个 debug 选项，用户可以决定是否输出与挖掘结果对应的原文（默认情况下不输出），从而跟踪规则的工作情况；此外，config 中还必须包含校正 NE 识别结果所需的用户词典的位置、输入语料的位置（在本地或直接从 hadoop 机群上获取）、语料存放的路径、语料的类型（query log 或 query session 或检索结果的 title、snippet，若为检索结果，还需给出存放检索 query 的文件的路径和返回结果的条数上限）和挖掘结果的存放位置。综上所述，该模块生成的初始 config 如图 4-1 所示，其中，以“#”为第一个非空格字符的行是注释：

```
#semantic knowledge mining

# dimension of argument, integer from 1 to 9
[dimension] number = '2'

# name of your job
[job] name = ''

# path of hadoop client
[hadoop_client] path = ''

# hadoop dir of your work
[hadoop] dir = ''

# local dir of your work
[local] dir = ''

# position of your static dictionary
[user_dict] position = ''

# location of corpus, hadoop or local
[location] choice = ''

# type of input corpus, query_log or query_session
# or web_title or web_snippet or web_title_snippet
[corpus] type = ''

# position of query log on hadoop
[query_log] position = ''

# position of query session on hadoop
[query_session] position = ''
```

```
# position of self-defined corpus in local
[self-defined_corpus] position = ''

# position of queries to be searched
[queries] position = ''

# number of search results for each query, positive
integer
[search_results] number = ''

# output the support sentences, yes or no
[debug] choice = ''

# position of output
[output] position = ''
```

图 4-1 初始config示意图

4.2.2 config_of_r

程序需要根据 n 的不同，生成不同维度的关系向量 R 的初始配置文件，以 $n=2$ 为例，其内容如图 4-2 所示：

```
R1 = {

    rm = {

    }

    r0 = {

    }

    r1 = {

    }

    r2 = {

    }

}
```

图 4-2 初始config_of_r示意图1

若挖掘任务需要对向量 R 使用多种规则，则需要用户在 config_of_r 中手动添加 R2、R3 等。以需要对向量 R 实施两种不同的规则为例，如图 4-3 所示：

```
R1 = {
    rm = {
    }
    r0 = {
    }
    r1 = {
    }
    r2 = {
    }
}
R2 = {
    rm = {
    }
    r0 = {
    }
    r1 = {
    }
    r2 = {
    }
}
```

图 4-3 初始config_of_r示意图2

4.3.3 config_of_a

程序需要根据 n 的不同，生成不同维度的语义知识单元向量 A 的初始配置文件，以 $n=2$ 为例，其内容如图 4-4 所示：

```
A1 = {

    at = {
    }

    am = {
    }

    a1 = {
    }

    a2 ={
    }

}
```

图 4-4 初始config_of_a示意图

当针对向量 A 的规则有多个时，其修改方法与 R 相同，用户只需在 A1 下面继续添加 A2、A3 等。

4.3 配置文件解析模块

该模块的作用是，检查用户修改后的 config，若有拼写或格式错误，则退出程序并给出提示，否则将其内容读入到内存中。采用如图 4-5 所示的数据结构进存储，并通过 C++语言编码实现：

```
struct Config
{
    int dimension;                  //语义知识单元向量A的维度
    string hadoop_client;           //hadoop机群客户端位置
    string hadoop_dir;              //在hadoop机群上的工作目录
    string local_dir;               //在本地机器上的工作目录
    string job_name;                //当前任务的名称
    string user_dict;               //NE识别结果校正模块所需的词典
    string corpus_location;         //输入语料的位置，hadoop或local
    string corpus_type;             //语料库的类型
    string query_log;               //query_log在hadoop机群上的路径
    string query_session;           //query_session在hadoop机群上的路径
    string local_corpus;            //存放在本地机器上的语料库的位置
    string queries;                 //用于获取检索结果的queries
    string num_of_search_results;   //返回检索结果的条数的上限
```

```
    string debug;                    //是否输出与挖掘结果对应的支持语句
    string output;                   //输出数据在本地机器上的存放位置
};
```

图 4-5 存储config的数据结构

该平台要求 dimension 大于等于 1，小于等于 9；num_of_search_results 大于 0；corpus_type 只能是“query_log”或“query_session”或“web_title”或“web_snippet”或“web_title_snippet”；debug 只能是“yes”或“no”。

config中所有的语句由三部分组成，即标识符、操作符（config中的所有操作符均为“=”）和参数。以语句“[queries] position = '/data/queries'”为例，其标识符为“[queries]position”，操作符为“=”，参数为“data/queries”。解析config的流程如下：

（1） 初始化存储 config 的数据结构

（2） 读取 config 文件，并忽略所有以“#”作为第一个非空格字符的行

（3） 将每一行中的空格去掉，并记录是在原 config 中的第几行

（4） 将每一行划分为标识符、操作符和参数三个部分

（5） 检查所有标识符、操作符是否正确，参数是否符合要求（如1<=dimension<=9），标识符是否有缺失或重复出现等，若有异常，则保留错误信息

（6） 若错误信息不为空，则退出程序，并给出错误提示（定位到行）

4.4 配置文件解析模块（R）

该模块的作用是，检查用户修改后的 config_of_r，若有拼写或格式错误，则退出程序并给出提示，否则将其内容读入到内存中，采用 C++语言编码实现。关系向量 R 及其分量以及针对各个分量的约束条件是层层嵌套的关系。每一个约束条件都由三部分组成，即标识符、操作符和参数，其数据结构如图 4-6 所示：

```
struct Constraint
{
    string bar;          //标识符
    string operation;    //操作符
    string parameter;    //参数
    int id;              //约束条件的id
};
```

图 4-6 存储约束条件的数据结构

每一个分量的规则由针对该分量的若干约束条件及逻辑关系表达式组成，其

数据结构如图 4-7 所示：

```
struct relation
{
    vector<Constraint> C;     //约束条件
    string LE;                //逻辑关系表达式
    bool used;                //该分量是否具有规则
};
```

图 4-7 存储向量R的分量的规则的数据结构

关系向量R的规则又由其所有分量（包括虚拟分量）的规则所组成，其数据结构如图4-8所示：

```
struct Relation
{
    relation rm;        //向量R的虚拟分量，所有实分量的和
    relation r[16];     //向量R的实分量
};
```

图 4-8 存储向量R的规则的数据结构

最后，config_of_r由所有针对向量R的规则所组成，其数据结构如图4-9所示：

```
struct Config_R
{
    Relation R[1024];     //关系向量R的规则
    int count;            //R的规则的数量
};
```

图 4-9 存储config_of_r的数据结构

在解析config_of_r的过程中，首先要忽略所有的注释语句并将所有的空格去掉，然后将所有行的内容合并在一起，变成一个字符串，如图4-10所示：

```
Rl = {
    rm = {
    }
    r0 = {
        0 : stop_list = 'list0';
        LE : 0;
    }
    rl = {
    }
}
```

⇩

```
Rl={rm={}r0={0:stop_list='list0';LE:0;}rl={}}
```

图 4-10 config_of_r预处理示意图

观察图4-10不难发现，无论向量R还是其分量r，其形式都符合“R(r)={…}”，因此，可以用递归的方式对config_of_r进行解析：递归的第一层解析向量R，而递

归的第二层解析分量r。而分量r的内部都符合“0:c0;1:c1;…n:cn;LE:exp;”的形式，因此，先后以“;”和“:”为分割符对其进行切分后，便可以使用与解析config相同的方法，对config_of_r进行错误检查，并将其内容读入内存。

4.5 配置文件解析模块（A）

config_of_a 所涉及到的数据结构与 config_of_r 大体相同，但为了使向量 A 和 R 形成映射关系，向量 A 的数据结构中还需要增加一个域，以记录当前的向量 A 与哪些 R 相对应，如图 4-11 所示：

```
struct Argument
{
    vector<int> to;     //虚拟分量at中的内容，记录与A对应的R的id
    argument am;        //虚拟分量，包含针对A的子向量的规则
    argument a[16];     //向量A的实分量
}
```

图 4-11 存储向量A的规则的数据结构

由于向量 A 的规则的表达形式与向量 R 基本相同，因此 config_of_a 的解析方法也与 config_of_r 相差无几，故不再赘述。

4.6 用户词表预处理模块

该模块首先根据 config_of_r 和 config_of_a 中的 candidate_list、stop_list、prefix_candidate_list 、 prefix_stop_list 、 suffix_candidate_list 、 suffix_stop_list 、substring_candidate_list 和 substring_stop_list 类型约束条件所涉及到的用户词表的路径，对用户词表进行去重（有可能多个约束条件共用一个用户词表），然后将每一个不同的词表拷贝到临时目录 output/data/list_temp 中（该目录中的词表会被上传到 hadoop 机群以供挖掘模块使用），最后依次检查这些词表是否在 output/data/list_temp 中（若词表存在且路径书写正确，将被拷贝到该目录中），若不在则退出程序并报错。该模块采用 C++语言实现。

4.7 语料预处理模块

因为 query log、query session 和检索结果的数据格式各不相同，所以首先应将所有语料处理成相同的格式，以便后续程序统一处理。后续处理包括断句、分词、词性标注和 NE 识别，本文所采用的是百度现有的 base 工具，只需对重新封装即可使用，这里不在赘述。最终输出格式为“行号 *** 内容 *** 频次”，其中“ *** ”为列的分隔符。该模块将作为 mapper，在 hadoop 机群上运行。

4.8 NE 识别结果校正模块

该平台使用正向最大匹配算法，并基于用户词典来对 NE 识别结果进行校正，使用 C++语言编码实现，并作为 mapper，在 hadoop 机群上运行。其具体流程如下：

（1） 将指针 p 指向当前句子的第一个 term,指针 q 指向其最后一个 term

（2） 若 p 指向的是最后一个 term 之后，跳转至（8）

（3） 将以 p 为首，以 q 为尾的 terms 合并为一个 term

（4） 查找合并后的 term 是否在用户词表中

（5） 若不在且 p<q，则 q 向前移动一个位置，程序跳转至（3）；
若不在且 p=q，则跳转至（7）

（6） 若在，则将合并后的 term 的 pos_tag 和 ner_tag 按照词表进行修改，并跳转至（7）

（7） 并将 p 指向 q 后面的第一个 term，再将 q 指向最后一个 term，并跳转至（2）

（8） 退出程序

以句子“周杰伦[n][NOR] 不能说的[j][NOR] 秘密[n][NOR] 试听[v][NOR]”为例，若用户词典中包含“周杰伦 *** [n][PER]”、“不能说的秘密 *** [n][SNG]”，则校正结果为“周杰伦[n][PER] 不能说的秘密[n][SNG] 试听[v][NOR]”。

4.9 挖掘模块

挖掘模块的任务是，根据向量 A 和 R 的规则，从输入语料中挖掘出符合要求的语义知识。该模块采用 C++语言实现，并作为 mapper 在 hadoop 机群上运行。由于实现通用性之后，规则很可能会非常复杂，既包含对字符串内容的要求，又包括对词的词性和 NE 类型的限定，甚至还可能涉及到基于各类词表的约束，因此，普通的字符串匹配算法如正则表达式等，已经无法实现对规则的演绎。本文转换了一下思路，即先枚举所有可能的挖掘结果（即语义知识单元向量 A），再验证其是否满足各类规则的约束，而不是直接通过演绎规则来获得结果。

设输入数据为一个带有 pos_tag 和 ner_tag 的 term 序列 $S=t_1,t_2,...,t_m$，则挖掘任务转化为从 m 个 term 中任意选取 n 个（将 m<n 的输入直接过滤掉）作为候选语义知识单元向量 A 的 n 个分量，判断当前的向量 A 和 R 是否符合其规则的约束条件，若符合则将该向量 A 作为结果输出。最简单的方法是通过深度优先搜索算法递归地枚举所有可能的向量 A，再针对每一种情况进行规则的判定。假设判定规则的时间为常数，则算法的时间复杂度为 $O(C(m,n))$。当 m 较大时，如 m=100，

处理每一个 term 序列都将消耗数秒的时间，如此之低的挖掘效率将是无法承受的。原因有包括以下几个方面：

（1）对于规则：当向量 A 和 R 拥有多条规则时，很可能不同的规则之间有部分约束条件是重复的，这些约束条件事实上只需被判定一次即可

（2）对于递归的过程：先递归枚举结果，再判定规则，会导致部分 a 和 r 被重复判定了很多次，浪费了很多时间；

（3）对于语料库：根据绪论中所进行的实验可知，能够抽取出符合要求的语义知识的句子，在 query log 和 query session 中所占的比例通常不足 0.1%，而在检索结果中所占的比例往往仅在 5% 左右。因此，直接对所有输入的句子进行抽取，必将造成极大的浪费。

针对（1），应首先将规则中所有的约束条件重；针对（2），应预先判定输入 term 序列的所有连续子序列（包括空，即没有任何内容的 term，长度为 0，没有 ner_tag 和 pos_tag）是否满足向量 A 和 R 的各个规则，并将结果记录下来；针对（3），本文提出了基于动态规划算法的过滤策略，在枚举挖掘结果之前将所有不可能符合规则的句子过滤掉。从而得出挖掘模块的流程如图 4-12 所示：

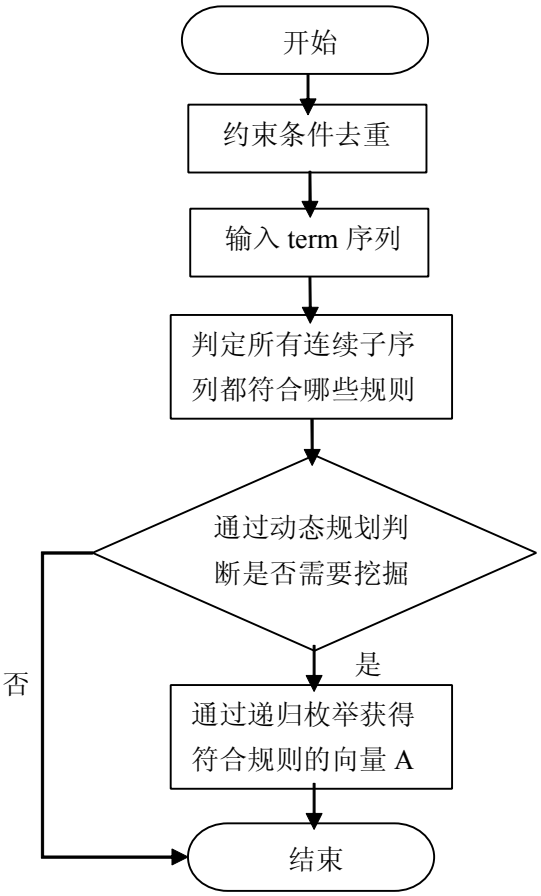

图 4-12 挖掘模块流程图

下面具体说明基于动态规划算法的过滤策略。经过预处理，输入 term 序列的任意连续子序列是否满足 $A_i.a_j$ 及 $R_i.r_j$ 的规则（因为针对虚拟分量 a_m 和 r_m 的规则只能在已知向量 A 的情况下判定，所以不包含在预处理结果中）为已知结果。观察输入句子 S 的表达形式 $S = r_0a_1r_1a_2r_2...a_nr_n(n \geq 1)$，不难发现 S 具有递归子结构。

在给定向量 A 及其对应的向量 R 的规则的前提下，句子 S 能否划分为形式 $r_0a_1r_1a_2...r_{n-1}a_n$ 与且仅与将 S' (S的前缀) 划分为形式 $r_0a_1r_1a_2...r_{k-1}a_k$ $(k < n)$ 的可能性相关。因此，判断句子 S 能否划分为形式 $r_0a_1r_1a_2...r_{n-1}a_n$ 的问题具有无后效性，可以使用动态规划算法来解决，然后再判断句子 S 划分为 $r_0a_1r_1a_2r_2...a_nr_n$ 的可能性。

令 $f(i,j)$ 表示以 S 中的第一个 term 为开头、第 i 个 term 为结尾的连续子序列划分为形式 $r_0a_1r_1a_2...r_{j-1}a_j$ 的可能性，$f(i,j) = true$ 表示可能，$f(i,j) = false$ 表示不可能，则有状态转移方程：

$$f(i,j) = \begin{cases} \bigcup\limits_{0 \leq k < i} (f(k,j-1) \& g(s(k+1,i-1), r_{j-1}) \& g(s(i,i), a_i)) \\ false, j > i \end{cases} \tag{4-1}$$

其中 $s(i,j)$ 表示以第 i 个 term 为开头、第 j 个 term 为结尾的句子 S 的连续子序列，若 $i > j$，则 $s(i,j) = \phi$；$g(s(i,j), a_k)$ 表示 $s(i,j)$ 能否满足向量 A 的第 k 分量的规则，$g(s(i,j), a_k) = true$ 表示满足，$g(s(i,j), a_k) = false$ 表示不满足，$g(s(i,j), r_k)$ 同理。初始化为 $f(0,0) = true,\ f(i,j) = false, i > 0, j > 0$，当 term 数为 m，向量 A 的维度为 n 时，计算 $f(i,j)$ 的时间复杂度为 $O(nm^2)$。而句子 $S = t_1, t_2, ..., t_m$ 划分为 $r_0a_1r_1a_2r_2...a_nr_n$ 的可能性为：

$$F(m,n) = \bigcup_{n \leq i \leq m} (f(i,n) \& g(s(i+1,m), r_n)) \tag{4-2}$$

计算 $F(m,n)$ 的时间复杂度为 $O(m)$，因此，基于动态规划算法的过滤策略的时间复杂度为 $O(nm^2) + O(m) = O(nm^2)$。当 m 较大且 $n \geq 2$ 时，其时间复杂度远小于深度优先搜索的时间复杂度 $O(C(m,n))$，通过该策略过滤掉大量的无用语料，将为挖掘任务节省很多的时间和资源。

4.10 结果统计模块

由于 hadoop 的 reduce 节点本身具有排序功能（默认情况下按字典序对每一行进行排序），所以，在挖掘模块的输出结果中，内容相同但频次不同的行会被排列到一起。因此，该模块只需要判断当前这一行内容是否与前一行相同，若相同则合并，并将其频次累加。该模块采用 C++来实现。

第 5 章实验与评价

5.1 评价指标

实现语义知识挖掘通用平台之后，本文针对相关实体挖掘和实体语义标签挖掘两个应用进行了实验，并对实验结果进行了评估。

相关实体挖掘的评价指标为覆盖率和准确率。对于实体 e，若挖掘结果中包含 e 的相关实体，则认为 e 是被覆盖的。根据经验，在最近 30 天的 querylog 中平均出现次数不少于 10 的实体为热门实体，这些实体往往是用户较为关心的。设某类热门实体的总数为 H，其中被覆盖的实体的个数为 h，则该类型相关实体挖掘的覆盖率为：

$$Coverate = \frac{h}{H} \tag{5-1}$$

从每一类热门实体中随机选取 10 个实体，并从挖掘结果中取出其相关实体（相关实体数超过 10 个的，按其共现频次保留 Top 10），再人工评估每一个相关实体对的正确性。设其中正确的对数为 e，总对数为 E，则该类型相关实体挖掘的准确率为：

$$Accuracy = \frac{e}{E} \tag{5-2}$$

实体语义标签挖掘的评价指标为准确率。从每一个类热门实体中随机选取 20 个，并对这些实体的语义标签抽取结果进行人工评估。设这些实体的标签的总数为 C，其中合理的标签有 c 个，则该类型实体语义标签挖掘的准确率为：

$$Accuracy = \frac{c}{C} \tag{5-3}$$

另外，还需要对通用平台的进行挖掘效率进行评估。设在占用相近系统资源的前提下，使用的单独模块完成某个挖掘任务的所需要的时间为 t，使用通用平台完成该任务所用的时间为 T，则根据设计目标（8），应满足：

$$\frac{t}{T} > 80\% \tag{5-4}$$

5.2 相关实体挖掘

5.2.1 基于 query log 的相关实体挖掘

根据绪论中的实验过程可以得出，完成该任务所需的配置文件 config、

config_of_a 和 config_of_r 分别如图 5-1、图 5-2 和图 5-3 所示：

```
# semantic knowledge mining

# target of mining, a(argument) or r(relation)
[mining] target = 'a'

# dimension of argument, integer from 1 to 9
[dimension] number = '2'

# name of your job
[job] name = 'relne_querylog'

# path of hadoop client
[hadoop_client] path = '/home/work/hadoop-client/hadoop/bin/'

# hadoop dir of your work
[hadoop] dir = '/app/st/nlp/zhanglibang'

# local dir of your work
[local] dir = '/home/work/zhanglibang/skmining/output/'

# position of your static dictionary
[user_dict] position = ''

# location of corpus, hadoop or local
[location] choice = 'hadoop'

# type of input corpus, query_log or query_session
# or web_title or web_snippet or web_title_snippet
[corpus] type = 'query_log'

# position of query log on hadoop
[query_log] position = '/log/20682/date/0000/querylog_date'

# position of query session on hadoop
[query_session] position = ''

# position of self-defined corpus in local
[self-defined_corpus] position = ''

# position of queries to be searched
[queries] position = ''

# number of search results for each query, positive integer
[search_results] number = ''

# output the support sentences, yes or no
[debug] choice = 'no'

# position of output
[output] position = '../data/result/relne_querylog'
```

图 5-1 基于query log的相关实体挖掘config示意图

```
# PER-PER
A1 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[PER]';
        1 : pos_candidate_set = '[n, nr, ns, j]';
        2 : text_length = '[3, 16]';
        3 : substring_stop_list = '../data/user_list/punctuation_list';
        4 : prefix_stop_list = '../data/user_list/digit_list';
        LE : (0 & 1 & 2 & 3 & 4);
    }

    a2 = {
        0 : ner_candidate_set = '[PER]';
        1 : pos_candidate_set = '[n, nr, ns, j]';
        2 : text_length = '[3, 16]';
        3 : substring_stop_list = '../data/user_list/punctuation_list';
        4 : prefix_stop_list = '../data/user_list/digit_list';
        LE : (0 & 1 & 2 & 3 & 4);
    }

}

#PER-LOC
A2 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[PER]';
        1 : pos_candidate_set = '[n, nr, ns, j]';
        2 : text_length = '[3, 16]';
        3 : substring_stop_list = '../data/user_list/punctuation_list';
        4 : prefix_stop_list = '../data/user_list/digit_list';
        LE : (0 & 1 & 2 & 3 & 4);
    }

    a2 = {
        0 : ner_candidate_set = '[LOC]';
        1 : pos_candidate_set = '[n, nr, ns, j]';
```

```
        2 : text_length = '[3, 16]';
        3 : prefix_stop_list = '../data/user_list/digit_list';
        LE : (0 & 1 & 2 & 3);
    }

}

#PER-ORG
A3 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[PER]';
        1 : pos_candidate_set = '[n, nr, ns, j]';
        2 : text_length = '[3, 16]';
        3 : substring_stop_list = '../data/user_list/punctuation_list';
        4 : prefix_stop_list = '../data/user_list/digit_list';
        LE : (0 & 1 & 2 & 3 & 4);
    }

    a2 = {
        0 : ner_candidate_set = '[ORG]';
        1 : pos_candidate_set = '[n, nr, ns, j]';
        2 : text_length = '[3, 16]';
        LE : (0 & 1 & 2);
    }

}

#PER-NVL,PER-SNG,PER-VDO_MVE,PER-VDO_TV,PER-VDO_TVSHOW
A4 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[PER]';
        1 : pos_candidate_set = '[n, nr, ns, j]';
        2 : text_length = '[3, 16]';
        3 : substring_stop_list = '../data/user_list/punctuation_list';
        4 : prefix_stop_list = '../data/user_list/digit_list';
        LE : (0 & 1 & 2 & 3 & 4);
    }

    a2 = {
```

```
        0 : ner_candidate_set = '[NVL, SNG, VDO_MVE, VDO_TV, VDO_TVSHOW]';
        1 : text_length = '[3, 16]';
        LE : (0 & 1);
    }

}
```

图 5-2 基于query log的相关实体挖掘config_of_a示意图

```
R1 = {

    rm = {

    }

    r0 = {

    }

    r1 = {

    }

    r2 = {
    }

}
```

图 5-3 基于query log的相关实体挖掘config_of_r示意图

人名的覆盖率为82.9%，以人名为轴的各类相关实体的准确率表5-1所示：

表 5-1 基于query log的相关实体挖掘的准确率

NE类型	人名	地名	机构名	小说	歌曲	电影	电视剧	节目
准确率	89%	70%	87%	85%	88%	83%	86%	95%

通用平台完成该任务平均用时28分钟左右，原独立模块完成该任务用时23分钟左右，由 $\frac{23}{28}=82\%>80\%$ 可知，通用平台的挖掘效率达到了预期的目标。

5.2.2 基于 query session 的相关实体挖掘

根据绪论中的实验过程可以得出，config 中除了 output、job name、corpus type 和 query session 在 hadoop 上的路径以外，其他内容相同；config_of_r 文件与基于 query log 的相关实体挖掘相同，config_of_a 如图 5-4 所示：

```
# NVL-NVL
A1 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[NVL]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

    a2 = {
        0 : ner_candidate_set = '[NVL]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

}

#SNG-SNG
A2 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[SNG]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

    a2 = {
        0 : ner_candidate_set = '[SNG]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

}

#VDO_MVE-VDO_MVE
A3 = {

    at = {
        to = '[1]';
    }
```

```
    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[VDO_MVE]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

    a2 = {
        0 : ner_candidate_set = '[VDO_MVE]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

}


#VDO_TV-VDO_TV
A4 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[VDO_TV]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

    a2 = {
        0 : ner_candidate_set = '[VDO_TV]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

}

#SFT-SFT
A5 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[SFT]';
```

```
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

    a2 = {
        0 : ner_candidate_set = '[SFT]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

}

#GME-GME
A6 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[GME]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }

    a2 = {
        0 : ner_candidate_set = '[GME]';
        1 : pos_stop_set = '[nr]';
        LE : (0&1);
    }
}
```

图 5-4 基于query session的相关实体挖掘config_of_a示意图

各类型热门实体的覆盖率如表5-2所示：

表 5-2 各类型热门实体覆盖率

NE类型	小说	歌曲	电影	电视剧	游戏	软件
覆盖率	98.6%	92.6%	98.1%	87.2%	86.5%	82.3%

各类型相关实体的准确率评估结果如表5-3所示：

表 5-3 各类型相实体准确率

NE类型	小说	歌曲	电影	电视剧	游戏	软件
准确率	81.6%	72.6%	92.3%	83.2%	85.5%	90.3%

通用平台完成该任务平均用时65分钟左右，原独立模块平均用时60分钟左右，由$\frac{60}{65}=92.3\%>80\%$可知，通用平台的效率达到了预期的目标。

5.3.3 音乐类相关实体挖掘

根据绪论中的实验过程可知，完成该任务所需的位置文件 config 和 config_of_r 均与基于 query log 的相关实体挖掘相同（除 config 中的 job name 和 output 以外），只需对 config_of_a 做出修改，如图 5-5 所示：

```
#singer-singer
A1 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[PER]';
        1 : candidate_list = '../data/user_list/singer_list';
        2 : stop_list = '../data/user_list/music_stop_list';
        LE : (0 & 1 & 2);
    }

    a2 = {
        0 : ner_candidate_set = '[PER]';
        1 : candidate_list = '../data/user_list/singer_list';
        2 : stop_list = '../data/user_list/music_stop_list';
        LE : (0 & 1 & 2);
    }

}

#singer-song
A2 = {

    at = {
        to = '[1]';
    }

    am = {

    }

    a1 = {
        0 : ner_candidate_set = '[PER]';
        1 : candidate_list = '../data/user_list/singer_list';
        2 : stop_list = '../data/user_list/music_stop_list';
        LE : (0 & 1 & 2);
    }

    a2 = {
        0 : ner_candidate_set = '[SNG]';
        1 : candidate_list = '../data/user_list/song_list';
```

```
        2 : stop_list = '../data/user_list/music_stop_list';
        LE : (0 & 1 & 2);
    }

}
```

图 5-5 音乐类相关实体挖掘config_of_a示意图

由于目前百度音乐APP仅对相关歌手资源展开了应用，所以只对歌手-歌手类数据进行了评估，对APP资源列表中的歌手的覆盖率为90.7%，相关歌手的准确率为96.5%（相关歌手也必须在APP资源列表中）。效率评估结果与基于query log的相关实体挖掘相同。

5.3 实体语义标签的挖掘

根据绪论中的实验过程，当挖掘给定实体类型的候选语义标签时，该任务所需要的 config 文件与基于 query log 的相关实体挖掘所用到的 config 基本相同，只需要对 job name 和 output 进行修改；当挖掘给定实体的候选语义标签以及对给定实体的语义标签集合进行扩充时，只需将 corpus type 改为“web_title_snippet”，number of search results 改为“100”，并对 job name、dimension 和 output 进行修改。下面依次给出各个挖掘步骤所需的 config_of_a 和 config_of_r：

（1）挖掘给定实体类型的候选语义标签

```
#a1为候选语义标签，a2为给定的实体类型
A1 = {
    at = {
        to = '[1]';
    }
    am = {
    }
    a1 = {
        0 : stop_list = '../data/user_list/label_stop_list';
        LE : 0;
    }
    a2 = {
        0 : candidate_list = '../data/user_list/ne_type_list';
        LE : 0;
    }
}
```

图 5-6 挖掘给定实体类型的候选语义标签的config_of_a示意图

```
#通过R先限定query中包含的term数，以及实体类型必须位于query尾部
R1 = {

    rm = {
        0 : term_count = '[0, 2]';
        LE : 0;
    }

    r0 = {
    }

    r1 = {
    }

    r2 = {
        0 : text_length = '[0, 0]';
        LE : 0;
    }
}
```

图 5-7 挖掘给定实体类型的候选语义标签的config_of_r示意图

（2）挖掘给定实体的候选语义标签（以电影类为例）

```
A1 = {

    at = {
        to = '[1]';
    }

    am = {
    }

    a1 = {
        0 : candidate_list = '../data/user_list/movie_list';
        1 : ner_candidate_set = '[VDO_MVE]';
        LE : 0;
    }

    a2 = {
        0 : candidate_list = '../data/user_list/movie_label_list';
        LE : 0;
    }
}
```

```
A2 = {

    at = {
        to = '[1]';
    }

    am = {
    }

    a1 = {
        0 : candidate_list = '../data/user_list/movie_label_list';
        LE : 0;
    }

    a2 = {
        0 : candidate_list = '../data/user_list/movie_list';
        1 : ner_candidate_set = '[VDO_MVE]';
        LE : 0;
    }
}
```

图 5-8 挖掘给定实体的候选语义标签的config_of_a示意图

```
R1 = {

    rm = {
    }

    r0 = {
    }

    r1 = {
    }

    r2 = {
    }
}
```

图 5-9 挖掘给定实体的候选语义标签的config_of_r示意图

（3）对给定实体的候选语义标签集合的补充（以电影类为例）

```
A1 = {

    at = {
       to = '[1]';
    }

    am = {
       0 : candidate_list = '[(1,2),(../data/user_list/movie_lable_and_type)]';
        LE : 0;
    }

    a1 = {

    }

    a2 = {

    }

    a3 = {
       0 : candiate_list = '../data/user_list/movie_list';
        LE : 0;
    }
}
```

图 5-10 对给定实体的候选语义标签集合的补充的config_of_a示意图

```
R1 = {

    rm = {
    }

    r0 = {
    }

    r1 = {
    }

    r2 = {
    }

    r3 = {
    }

}
```

图 5-11 对给定实体的候选语义标签集合的补充的config_of_r示意图

各类实体的语义标签挖掘结果的准确率如表5-4所示：

表 5-4 实体语义标签挖掘准确率

NE类型	音乐	电影	游戏	漫画
准确率	98.3%	97.5%	95.2%	92.7%

通用平台完成该任务平均用时80分钟左右，原独立模块平均用时70分钟左右，由 $\frac{70}{80} = 87\% > 80\%$ 可知，通用平台的效率达到了预期目标。

结　论

本文在进行相关实体挖掘和实体语义标签挖掘的过程中发现，不同的规则通常会导致不同模块的实现方法不同，从而使得模块的可维护性和可扩展性较差。倘若可以为一些常用的规则设计统一的表达形式，并通过统一的方法对规则进行解析，再挖掘符合规则的语义知识，将节省很多用于重复开发的时间和资源。

本文从实验中总结语义知识挖掘任务中的常见需求，据此对规则的内容进行了归纳。对规则进行了重新定义，实现了规则表达形式的统一化，进而为实现语义知识挖掘通用平台打下基础。

根据通用平台的基本流程，将其划分为 9 个不同的功能模块，即初始配置文件生成模块、配置文件解析模块、配置文件解析模块（A）、配置文件解析模块（R）、用户词表预处理模块、语料预处理模块、NE 识别结果校正模块、挖掘模块和结果统计模块。并依次给出了各个模块的详细设计方案及实现方法。其中配置文件解析模块（A）、（R）抓住了规则表达形式上的特点，采用递归的方法对语义知识单元向量 A 和关系向量 R 的规则进行快速解析；挖掘模块使用了逆向思维，即先枚举可能的挖掘结果，再验证其是否满足规则，从而避免直接对复杂的规则进行演绎，设计并是实现了基于动态规划算法的过滤策略，将大部分无用语料直接过滤掉，从而保证了挖掘的效率。

最后，使用 C++语言和 shell 脚本完成了语义知识挖掘通用平台的编码实现，并针对相关实体挖掘和实体语义标签挖掘这两个任务进行了实验，实验结果表明该平台具有良好的通用性和较高的挖掘效率，达到了预期的设计目标。

参考文献

[1] 刘耀，惠志方，胡永伟. 领域 ontology 自动构建研究[N]. 北京：北京邮电大学学报，2006：1-1

[2] 李赟. 基于中文维基百科的语义知识挖掘相关研究[D]. 北京：北京邮电学博士学位论文，2009：1-1

[3] Shuqi Sun, Shiqi Zhao, Muyun Yang. Harvesting Related Entities with a Search Engine[J]. Harbin Institute of Technology, 2011

[4] Shiqi Zhao, Haifeng Wang, Ting Liu. Paraphrasing with Search Engine Query Logs[J]. Baidu Inc, 2010

[5] Kathrin Eichler, Holmer Hemsen, Günter Neumann. Unsupervised Relation Extraction from Web Documents[J]. In Proceedings of LREC, 2008

[6] Alpa Jain, Marco, Pennacchiotti. Open Entity Extraction from Web Search Query Logs[J]. In Proceedings of COLING, 2010

[7] Fei Wu, Daniel S. Weld. Open Information Extraction using Wikipedia[J]. In Proceedings of ACL, 2010

[8] Zaiqing Nie, Yunxiao Ma, Shuming Shi. Web Object Retrieval[J]. In Proceedings of WWW[J]. 2007

[9] Zaiqing Nie, Ji-Rong Wen, Wei-Ying Ma. Object-Level Vertical Search[R]. In Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research, 2007

[10] Mengqiu Wang. A Re-examination of Dependency Path Kernels for Relation Extraction[J]. In Proceedings of IJCNLP, 2008

致　谢

首先感谢指导教师关毅教授。关老师渊博的知识、扎实的专业基础、严谨的治学态度和对科学的献身精神，是我们一生学习的榜样。从关老师那里，我们学到的不仅仅是知识，更重要的是思维的方法和治学的态度。项目的完成凝聚了关老师的大量心血，我们的每一点进步都倾注着他的关心和帮助。在这里，衷心的感谢关老师的教导和培养。

感谢网络智能研究室的所有同学们，在实习一年里我从你们身上学到了很多东西。感谢你们对系统的各个部分所提出的建议，感谢你们和我一起度过了我快乐充实的大四时光。感谢杨锦锋师兄不辞劳苦，认真细致地审阅了论文全文，并在论文内容以及规范方面提出了宝贵的意见。

感谢百度时代网络技术（北京）有限公司为我提供的宝贵的实习机会和良好的工作环境，让我意识到了书本上的知识与实际应用的差别和联系，领悟到了“简单可依赖”的科研理念。每一位同事都很热心，这里不仅仅是一个工作和学习的地方，更是一个温暖的百度大家庭。

最后，特别感谢我的实习指导人赵世奇博士，我学到的不仅仅是新的技术，更重要的是发散思维的方法和与同事沟通交流的技巧。工作上的指导，生活上的关心，让我在科研之外，仍能感受到心中的温暖。