1 程序设计课程语料库构建

本章首先对程序设计课程语料库构建面临的问题进行了分析，然后按照语料库构建流程详细叙述了基于规则的程序设计课程小型实体库的构建过程、自动化语料标注过程以及适配 BERT 的标注语料格式转换方法。

1.1 问题分析

目前对于程序设计课程的实体提取研究比较匮乏，没有人工标注的程序设计课程语料数据集，也没有完备的程序设计课程知识实体库，因此需要从零构建程序设计课程语料库。此外，人工标注的方式在有限的时间中标注的语料数量也十分有限，直接使用有限的数据来进行实体提取模型的训练可能会导致模型精确度较差，因此考虑通过构建小型实体库的方式来实现语料的自动化批量标注。

大部分成熟的命名实体识别都是面向人名、地名、机构名的，无法提取出程序设计课程目标实体，因此考虑使用基础的规则匹配方式来构建小型实体库。由于规则匹配的精确度较低，对于规则匹配的结果还需要进行人工审核。为了减轻人工审核的工作量，可以创建一个黑名单，记录下所有被丢弃的结果。对于规则匹配得到的候选实体，优先和黑名单以及实体库匹配，对于匹配成功的词无需进行重复的人工审核，从而减轻人工审核对于重复出现的实体部分的工作量。

程序设计课程大纲是构建小型实体库的最佳数据来源，因为相比起其他程序设计课程相关文本，课程大纲的知识实体密度是最大的。在文本中出现的目标实体密度越小，获取相同数量目标实体所需要审核的候选结果就越多，使用程序设计课程大纲可以减轻人工审核负担。但是通过网络上收集的得到的数据大纲大部分为 pdf 文件以及 png 图像文件，无法直接使用，需要格式处理获取其中的文本内容。

在确定了数据来源以及大致方案后，程序设计课程知识实体库构建流程总结如图 3.1 所示。当小型实体库搭建完毕后，便可以根据实体库来对文本进行匹配标注，完成语料标注。

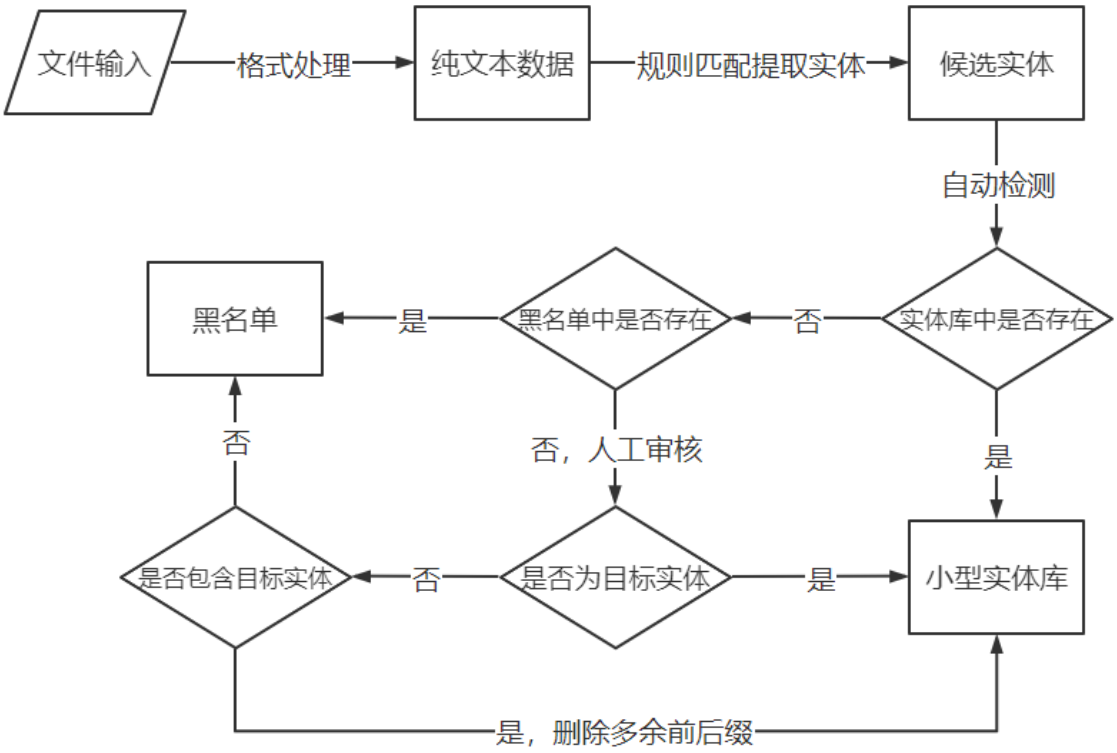


图 3.1 语料库构建流程图

1.2 基于规则的小型实体库构建

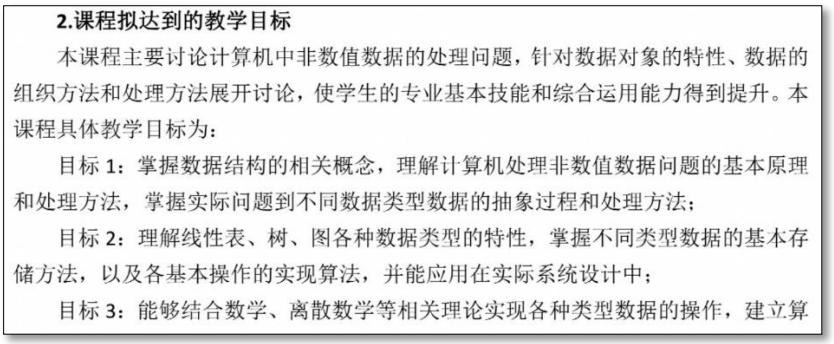
1.2.1 数据格式处理

课程大纲的主流文件格式包括 pdf、word 以及 png 图像格式。

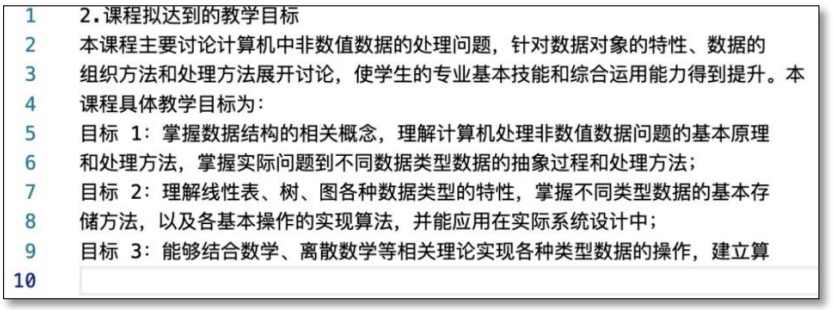
对于 pdf 文件，首先使用 Python 的 PyMuPDF 模块将其转换成 png 图像文件。MuPDF 是一个轻量级的 PDF 查询编辑器，支持包括 PDF 在内的多种文档格式，而 PyMuPDF 则是 MuPDF 的 Python 绑定，可以通过 Python 代码接口实现 MuPDF 的各种功能。PyMuPDF 模块可以保证以近乎无损地精度将 pdf 文件转换为 png 图像格式，在转换时需要设置好 png 图像文件的缩放比例，以提高图片清晰度，否则后续流程中文本检测将无法定位到文字。

对于 png 图像文件，使用百度飞桨(PaddlePaddle)的中文光学字符识别套件来获取图像的文本信息，其中文本检测使用的是 DBNet 算法，文本识别使用的是 CRNN 模型。对于 png 图像文件中的中文文字，该套件拥有较高的准确率和效果，但是对于英文符号，识别准确率较低。同时识别结果是按行保存的，如图 3.2 所示，导致文本本身的段落格式以及表格格式无法被复原，同一段文本会被

保存成多行文本，无法判断前后行的文本内容是否为连贯内容，因此对于图像文件识别后的内容需要进行清洗才能使用。对于识别出的每一行文本内容，只将两个相邻结束符(如句号、感叹号等)之间的文本内容提取出来，从而保证句子内容的正确性。



(a) 图像原始文件



(b) OCR 识别结果

图 3.2 图像文件 OCR 识别结果

对于 word 文件，使用 Python 的 document 模块来解析。与图像文件识别结果相比，使用 document 模块解析 Python 可以按段、表格格式获取文本内容，保留了原始结构，同时也不会出现一句话被拆分的情况。因此对于每一段、每一单元格的文本只需要按照结束符拆分出句子内容即可，对于较短的段甚至无需拆分，后续任务支持多句话同时输入。

对于上述输入文件，最后统一以句子为单位保存成纯文本格式，便于后续处理。

1.2.2 规则匹配

通用领域中的命名实体识别的规则匹配，是多位语言学专家选用了文本中的多个特征后手工构造的，主要包括标点符号、关键字、方位词、指示词以及文本统计信息等。这类系统的构建依赖于知识库和词典，以模式和字符串匹配为主要

手段，而程序设计课程实体库还需构建，知识库与词典都不存在，因此并不适用。

通过实验观察发现，根据词性标注后的结果来制定规则，可以筛选出包括目标实体的候选实体。词性标注任务目前已经发展得相当成熟，因此可以使用已有的自然语言处理平台，而无需重复训练词性标注模型，在本论文中使用的是哈尔滨工业大学研发的语言技术平台(Language Technology Platform, LTP)。LTP 平台支持中文分词、词性标注、命名实体识别、句法分析等常见自然语言处理基础任务，提供 Python 接口以及网络服务(Web Service)。该平台的分词和词性标注任务都使用的是 Chinese-ELECTRA 模型(中文预训练模型。在谷歌与斯坦福大学共同研发的预训练模型 ELECTRA 的基础上使用中文数据训练得到。其参数为 BERT 的 1/10，却拥有相当的效果。)，在人民日报 1998 年 1 月数据测试集上 F1 分数都达到了 98.5，拥有优秀的性能。为了更好的介绍具体匹配规则，首先需要了解词性标注的结果种类。LTP 词性标注结果种类如表 3.1 所示：

表 3.1 词性标注集

标注符号	代表含义	标注符号	代表含义
a	形容词	ni	组织名
b	其他修饰词	nl	地名
c	连词	ns	地理名
d	副词	nt	时间名词
e	感叹词	nz	专有名词
g	词素	o	拟声词
h	前缀词	p	介词
i	成语	q	量词
j	缩写	r	代词
k	后缀词	u	助动词
m	数字	v	动词
n	普通名词	wp	符号
nd	方位词	ws	外来词汇
nh	人名	x	非词语成分
		z	描述词

由于存在人工审核，所以可以制定较为宽泛的规则，尽可能挖掘出潜在的实体，从而快速构建小型实体库。具体的规则可以总结归纳成主语、修饰语+主语以及独立外来词汇三种。其中，主语和修饰语分别为主语成分和修饰词的最长连续组合。

主语成分包括名词(n)和动词(v)。由于大部分的目标实体均为专业名词，因此将名词视为主语的主要构成成分。同时在词性标注时，有部分动名词会被标注成动词成分，而它实际上是作为名词使用的，所以将动词也纳入主语成分。如“快速排序”中的“排序”会被标注成动词成分，而实际上“快速排序”也可以作为“快速排序算法”的省略表达，是名词短语。此外，一些完整的专业名词会被拆分成多个名词或动词，因此还需要考虑主语成分的连续组合，如“数据结构”一词被拆分成了“数据”(n)和“结构”(n)，“排序算法”一词被拆分成了“排序”(v)和“算法”(n)。

相比起主语成分，修饰词的范围则更加宽泛，因为在词性标注结果中很多词性的词语都可以作为修饰词。修饰词主要包括形容词(a)、其他修饰词(b)、副词(d)、符号(wp)、数字(m)、量词(q)、方位词(nd)以及外来词汇(ws)八种成分。

外来词汇本身是修饰词中的一种，但是同时也能独立作为目标实体存在。如果一个外来词汇的前后词既不属于主语成分也不属于修饰词，那么将其视为独立外来词汇，作为候选实体选出。

具体规则对应匹配的目标实体如表 3.2 所示：

表 3.2 目标实体与具体规则对应表

目标实体	构成规则	所属类别
选择/语句	v n	主语
逻辑/表达式	n n	主语
单分支	n	主语
分支/嵌套	n v	主语
散列/函数	v n	主语
深度/优先/搜索	n v v v	主语
If/语句	ws n	修饰语+主语
双/分支	m n	修饰语+主语

最/短/距离	d a n	修饰语+主语
单/步/执行	b q v	修饰语+主语
外部/排序	nd v	修饰语+主语
B/+/树	ws wp n	修饰语+主语
Dijkstra	ws	独立外来词汇
Kruskal	ws	独立外来词汇

在匹配主语和修饰语时都是按照最长连续串来匹配的，这样才能保证不会出现目标实体被截断的情况，但是会引起多余前缀或者后缀的问题。一些常见的多余前后缀会被记录下来，在输出候选实体时被自动清除，如“掌握”、“重点”、“难点”等等。剩余的非常见多余前后缀则在人工审核时被手动清除，保留出目标实体部分。

上述的规则较为宽泛，候选实体中存在大量非目标实体。在构建小型实体库初期时这是比较必要的，这样可以保证尽可能挖掘出更多的实体，快速构建出实体库。

1.2.3 黑名单匹配及优化

为了减轻人工审核的工作量，除了实体库外还额外构建了黑名单集合。实体库与黑名单皆为字典结构，分别记录了目标实体与其词频以及不符合要求的候选实体与其词频，其中实体为键，词频为对应值。对于规则匹配的结果将和实体库、黑名单做匹配，对于已经出现过的实体将更新其词频，并且无需后续人工审核。

当小型实体库中实体数量为 382 时，对于一篇新输入的文本内容，其规则匹配得到的候选实体中 25%的实体能够和实体库和黑名单匹配，剩余 75%的内容需要人工审核判断是否纳入实体库。

但是非目标的候选实体千奇百怪，导致黑名单的匹配成功率远远低于实体库，匹配占比大约为 1:4。为了进一步减少人工审核工作量，自动过滤掉更多的非候选实体，构建了黑名单部件和白名单部件两个集合，对黑名单机制进一步进行了优化。实体是由分词后的多个词语组合而成的，黑、白名单部件集合中则分别包括了黑名单和实体库中所有实体的构成词语。在对候选实体做匹配时，如果没有命中实体库和黑名单，则对候选实体的组成词语进行检查，若候选实体的组成词语都只在黑名单部件中出现且未在白名单部件中出现，则同样将该实体视为命中

黑名单。优化了黑名单匹配机制后，能够将匹配成功部分的平均占比从 25%提升至 40%，黑名单匹配平均占比从 5%提升至 20%，有效减少了人工工作量。

1.2.4 小型实体库介绍

本论文使用十篇数据结构、程序设计课程大纲作为小型实体库构建的原始数据，按照上述流程构造，最终获得了一个大小为 689 的小型实体库，部分实体库内容如表 3.3 所示。

表 3.3 部分实体库内容展示

实体	词频	实体	词频	实体	词频
栈	79	矩阵	30	线索化	81
队	1	广义表	100	连通性问题	2
数组元素	1	下标变换公式	2	拓扑排序	291
内存地址	104	三元组	13	邻接矩阵表示方法	1
指针数组	166	树	106	深度优先搜索遍历	15
函数指针参数	2	二叉树	1661	广度优先搜索遍历	15
库函数	4	森林	305	Prim 算法	1
命令行参数	2	哈夫曼树	201	拓扑序列	1
函数指针	2	逻辑特点	2	顺序查找法	1
指针参数	2	编码	576	半查找法	1
数组指针	98	遍历	2383	平衡二叉树	360
链表	1970	图	94	散列 Hash	1
类型别名	1	有向无环图	164	散列函数构造方法	3
动态链表	2	生成树	184	冲突解决方案	1
二进制格式	1	最小生成树	515	顺序查找	203

同时为了进一步帮助程序设计领域的知识图谱构建，本论文对实体库中的实体人工分类，将实体按照知识点类别分为数据结构实体、算法知识实体、程序设计知识实体、计算机通用知识实体这四类实体，并构建出知识实体分类集，其中部分内容见表 3.4。

表 3.4 部分知识实体分类集展示

数据结构实体	算法知识实体	程序设计知识实体	计算机知识实体
数组	哈夫曼编码	函数	编译器
指针	最小生成树算法	语法	源代码
字符串	最短路径算法	语义	进制
单链表	深度优先搜索遍历	符号	内存
字符	广度优先搜索遍历	转义符	数据
队列	插入排序	算术运算符	地址
顺序栈	快速排序	算术表达式	原码
矩阵	选择排序	变量值	反码
二叉树	归并排序	精度	补码
森林	基数排序	条件表达式	位运算
哈夫曼树	外部排序	参数	语法错误
哈希表	模式匹配算法	全局变量	注释规范
栈	矩阵转置算法	局部变量	IPO
队	拓扑排序	标准库函数	IO
B+树	递归算法	实参	程序逻辑
线性表	Prim 算法	形参	语法规则
数组指针	半查找法	作用域	软件工程思想

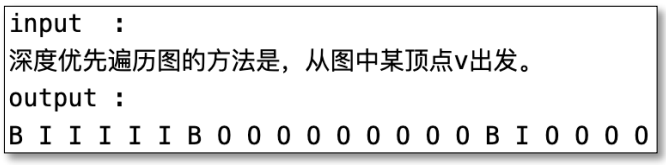
1.3 基于实体库的自动化语料标注

在构建了小型实体库后，便可以根据实体库中的实体来标注数据。在本论文中，使用 jieba 组件实现文本自动化标注。jieba 是专门针对中文分词的 Python 组件，支持加载自定义词典以及多种分词模式，包括精确模式（适合文本分析）、全模式（列出所有的词，不能解决歧义）、搜索引擎模式（对长词二次切割，适用于搜索引擎）以及 paddle 模式（使用 paddle 深度学习模型）。jieba 组件的分词算法采用基于词典的算法和统计学习模型相结合的方式，首先基于前缀词典生成一个由句子的所有可能分词情况所构成的有向无环图，然后使用动态规划算法找到其中的最大概率路径，即基于词频的最大切分组合。实体库中记录下词频也是

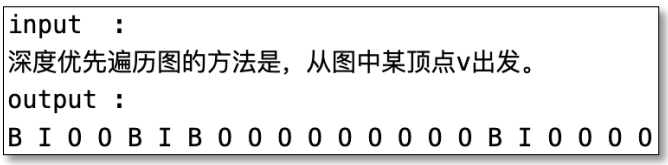
为了更好的分词效果。对于未登录词，则使用 HMM 模型与维特比(Viterbi)算法来处理。

本论文使用的是 jieba 组件的精确分词功能，并加载按照 1.2 节中方法构建的小型实体库作为词典，这样可以保证实体库中的实体被完整地拆分出来。标注方式采用的是经典的 BIO(B-begin，I-inside，O-outside)标注方式，在获得分词结果后，对所有词语遍历检查是否存在于实体库中，对于命中实体库的词语，第一个字符标注 B，剩余字符标注 I，而未命中实体库的字词全部标注 O。

加载实体库与不加载实体库的自动标注结果如图 3.3 所示。在不加载实体库进行分词时，“深度优先遍历”被拆分成了“深度/优先/遍历”，而“深度”、“遍历”都是实体库中的词语，因此被标注出来。“深度优先遍历”本身存在于实体库中，是一个完整的实体，但是却被错误的标注成“BIOOBI”序列。而加载实体库进行分词后，“深度优先遍历”可以被完整的分词出来，并且命中实体库，从而被正确的标注出来。



(a) 加载自建实体库进行分词的标注结果



(b) 不加载自建实体库进行分词的标注结果

图 3.3 自动化标注结果展示

1.4 适配 BERT 的标注语料格式处理

在本论文中，实体提取使用的大多数深度学习模型是基于中文 BERT 的(具体模型结构详见**错误!未找到引用源。**章)。由于模型使用中文 BERT，导致对于中英文混合的文本字符切割时会出现问题。在原本的标注方式中，对于文本中的英文字符与数字字符，会以单个英文字母和数字为粒度进行切割，中文字符也是按照单字进行切割。而在中文 BERT 的文本预处理过程中，对于中文文本中的中

文字符，会按照单字切割，但是对于非中文字符，包括英文以及数字，会连在一起被完整的切割出来。比如对于输入文本“第 13 章：迪杰斯特拉算法(Dijkstra)。”，原本标注方式中切割后的结果为“第/1/3/章/：/迪/杰/斯/特/拉/算/法/(/D/i/j/k/s/t/r/a/)/。/”，但是 BERT 文本预处理中的切割结果为“第/13/章/：/迪/杰/斯/特/拉/算/法/(/Dijkstra/)/。/”，数字 13 和英文 Dijkstra 都被度完整切割了出来。因此对于原本的标注结果还需要进行格式转换，将其转换为与中文 BERT 文本预处理一致的切割方式。

格式转换主要通过构建坐标转换数组来实现。对于长度为n的句子序列，构造一个长度为n的坐标转换数组D，其中$D[i]$记录了第 i 个字符在新标注序列中的坐标。有了坐标转换数组以后，可以轻松的实现原序列坐标至新序列坐标的转换。坐标转换数组的构造算法伪代码如下：

算法 1 坐标转换数组构造算法

```
# sentence 为输入的句子序列，类型为字符串
# D 为坐标转换数组指针，长度与 sentence 相同，均为 length
# is_en 用于标记前面一个字符是否是英语，is_num 用于标记前面一个字符是否是数字
# is_english 用于判断当前字符是否是英语，is_number 用于判断当前字符是否是数字
BUILD_TRANSARRAY(sentence, D, length)
    is_en = False
    is_num = False
    trans_length = 0
    for i in range(length):
        char = sentence[i]
        if is_english(char) and not is_en:
            is_en = True
            is_num = False
            D[i] = trans_length
            trans_length += 1
        elif is_english(char) and is_en:
            D[i] = trans_length – 1
        elif is_number(char) and not is_num:
            is_en = False
            is_num = True
            D[i] = trans_length
```

```
        trans_length += 1
    elif is_number(char) and is_num:
        D[i] = trans_length – 1
    else:
        is_en = False
        is_num = False
        D[i] = trans_length
        trans_length += 1
```