北京信息科技大学

**毕业设计（论文）附录**

**题 目：** 特定词汇语义关系挖掘系统的设计与实现

**学 院：** 计算机学院 \_\_

**专 业：** 计算机科学与技术 \_\_\_

**学生姓名：** 高泽坤 **班级/学号** 1403/2014011397 \_\_\_

**指导老师/督导老师：** 蒋玉茹\_\_\_\_\_\_\_\_\_\_\_\_\_

**起止时间：\_** 2018年2月26日 至 2018年6月22日\_\_

目 录

附件1 开题报告 共3页

附件2 计算机程序 共20页

附录3 外文文献译文 共8页

附录4外文文献原文 共9页

## 附件1 开题报告

特定词汇语义关系挖掘系统的设计与实现

**开题报告**

**班级（学号） 计科1403(2014011397) 姓名 高泽坤**

**指导教师 蒋玉茹**

1. **综述**

**1、题目背景和意义**

随着互联网的兴起和普及，现在有大量的信息可以通过互联网进行获取。而跟据研究，所有词汇的语义关系可以大致分为同义、反义、上下位、整体-部分等关系，如果可以利用互联网上已有的信息，例如百度百科中相关词汇板块的资源，配合爬虫、网页分析技术以及目前正在兴起的人工智能和大数据技术，从中提取出有效信息并加以利用，进而从互联网中提取大量的相关词汇关系并进行分析，将会对将来的词典构建较为完善的语义关系体系将会有很大的帮助。除此之外，该系统的实现对于其他语种的同义词自动获取方面同样具有实用性。在未来的研究中，还可以应用于进一步实现模式的自动定义、完善抽词词典、有效排除噪音数据并构建能真实反映语义关系的词汇矩阵。

**2.研究的现状及已有成果**

针对语义关系抽取的研究早已开始进行，在传统情况中，就已经有了两种方法，分别为：

1. 基于模式匹配的方法：运用人工或自动抽取的规则发现词汇语义关系，即两个名词之间的特定词汇句法模式预示着特定的语义关系。基于模式的方法从语料库中直接抽取语义关系, 准确率较高, 但是模式的适用性与语料本身密切相关, 移植到新领域成本比较高。
2. 基于统计的方法：根据统计词语和相近词语联合出现的概率，判断两个词语是否有语义关系。通过对语句进行句法分析, 将句法分析树切分成子树, 并将句法分析树的子树作为特征,构建特征向量, 利用感知器算法对英文句子进行语义关系抽取。基于统计的方法适合处理大规模数据, 移植性好、易扩展,但是准确性比基于模式的抽取方法低。

目前, 国内已经有了一些中文同义词资源[1 ] , 例如:《同义词词林》(梅家驹)、《同义词词林扩展版》(哈工大信息检索研究室)、《中文概念词典CCD》(北京大学计算语言学研究所)、中文WordNet 等。《同义词词林》以及扩展版收录的主要是日常用语中出现的普通同义词, 其收词数量有限, 词典更新滞后, 若将其直接作为同义词词典使用, 显然不能满足实际的需要, 尤其是在信息检索领域, 对于检索科技文献或其它专门学科资料的目的而言, 《同义词词林》并不能适用。《中文概念词典CCD 》和中文Word-Net 是以英文WordNet 为基础, 结合英汉翻译等半自动方式创建的面向中文信息处理的中文语义词典, 这两部语义词典的开发在一定程度上缓解了同义词获取的困难, 但其所涵盖的概念,尤其是常用的科技类术语, 还是远远不够的, 在新知识、新术语持续激增的网络信息时代, 它们并不能很好地用于解决同义词问题。

目前随着百科资源的丰富，网络上的知识性百科资源为词汇语义关系的发现和获取提供了非常丰富的语料。例如在百度百科对词条的注释当中，通常会使用与同义词等对词汇进行解释，这对我们利用该词条进行获取信息。除此之外，百科系统对概念的解释以及html网页的代码格式也有利于我们对词汇语义关系进行提取并加以利用。因此本项目主要基于百科资源进行实现。

**二、研究内容**

**1、研究方向**

课题以百度百科中的语料作为基础，设计并实现一个词汇语义关系挖掘系统。系统将对以百度百科为基础，扩展到整个互联网中的相关词汇语义关系进行自动获取，并转化为固定格式，利用D3JS可视化技术转化为图片以直观的形式进行展示。

**2、研究内容**

1)基于网页标签的获取方法

基于网页标签的方法，以百度百科的生物板块为例，首先利用爬虫技术爬取网页，根据分析网页中固定的网页标签，直接可以获取具有相应语义关系的词汇。提取出数据后加工成D3JS可直接转化为树状图的格式即可。

2)基于模式的获取方法

这里主要借鉴了DIPRE利用模式和关系之间的二元性，使用自举的模式匹配方法，在自然语言的文本上抽取关系实例。首先与方法一近似，首先对百度百科的语料进行提取，然后以已有的语料作为种子，提取出固定模式，成为可信模式，最后利用可新模式获取网络中所存在的新的同义词。在处理过程中，可能使用到大数据相关的处理方式对大量的数据进行处理。

**3、系统功能**

该系统对百度百科中的有效数据进行提取，并进一步实现从更多的网页中分析并提取语义关系的功能。主要方法包括两项，其一为利用爬取下来的html网页中有效信息所在的部分的标签格式，对有效信息进行直接提取；其二为利用已有的信息作为种子，去学习词汇语义关系所在的句子格式，并利用学习到的模式实现在更多语料中自动提取有效信息。

**三、实现方法及预期目标**

**1、初步方案**

1）利用爬虫技术，针对百度百科的网页进行爬取

2）分析爬取下来的网页，寻找有效信息所在部分的特殊代码格式

3）利用固定百度百科网页的固定格式，提取出其中有用的数据，其中一部分可直接转化为有效数据

4）另一部分作为DIPRE技术所用的种子进行固定格式的探索

5）利用之前提取出来的数据，利用DIPRE技术实现从固定格式的语料中自动提取有效数据。

**2.重点与难点**

爬取网页、分析网页、提取数据、按固定格式输出、利用D3JS进行展示的一整套流程，以及在应用方法2实现的过程中对于DIPRE技术的应用，将会涉及到人工智能和大数据的相关应用。

**3、环境**

1）硬件环境：硬件环境：主流的PC机

2）开发环境：PyCharm，Eclipse

3）编程语言（运行环境）：python和java

1. **对进度的具体安排**

第1～3周 确定毕业设计任务，收集资料，完成开题报告

第4周 完成需求分析工作

第5～7周 总体设计，构造系统框架及主要模块

第8～9周 系统功能源代码设计

第10～11周 调试程序与修改工作

第12～13周 完善系统、整体测试，撰写、提交论文

第14～15周 修改论文、准备答辩PPT

第16～17周 参加答辩

1. **参考文献**

[1] Sergey Brin.Extracting Patterns and Relations from the World Wide Web

[2]陆勇,章成志,侯汉清.基于百科资源的多策略中文同义词自动抽取研究[J].中国图书馆学报,2010,36(01):56-62.

[3] 宋文杰,顾彦慧,周俊生,孙玉杰,严杰,曲维光.多策略同义词获取方法研究[J].北京大学学报(自然科学版),2015,51(02):301-306.

[4] Marti A. Hearst. Automatic Acquisition of Hyponyms ~om Large Text Corpora

[5] Deepak Ravichandran，Eduard Hovy. Learning Surface Text Patterns for a Question Answering SystemLearning Surface Text Patterns for a Question Answering System

[6]赵君.垂直搜索引擎爬虫系统DIPRE算法及改进[J].软件导刊,2016,15(08):30-32.

[7]伍盛. 基于词义相似度的文本推荐系统的研究与实现[D].电子科技大学,2015.

[8]宋文杰. 汉语词汇上下位关系获取及其应用研究[D].南京师范大学,2015.

[9]马海昌. 基于大规模语料库的中文同义词抽取方法研究[D].西北师范大学,2014.

[10]陆勇. 面向信息检索的汉语同义词自动识别[D].南京农业大学,2005.

**指导教师：**（签署意见并签字） 年 月 日

**督导教师：**（签署意见并签字） 年 月 日

**领导小组审查意见：**

审查人签字： 年 月 日

## 附件2 计算机程序

|  |
| --- |
| 数据提取：Biology.py |
| # \_\*\_ coding: utf-8 \_\*\_  #!/user/bin/python  #!conding=utf-8  import json  import re  import os  import time  from html.parser import HTMLParser  class MyHTMLParser(HTMLParser):  def \_\_init\_\_(self):  HTMLParser.\_\_init\_\_(self)  self.a\_t= False  self.b\_t = False  self.c\_t = False  self.d\_t = False  self.keys = []  self.values = []  self.check = []  #处理开始标签，比如<xx>  def handle\_starttag(self, tag, attrs):    for (variable, value) in attrs:  #if tag == "div" and variable == "class" and value == "para":  #self.c\_t = True  if tag == "span" and variable == "class" and value == "biTitle":  self.a\_t = True  elif tag == "div" and variable == "class" and value == "biContent":  self.b\_t = True  else:  pass  #处理<xx>data</xx>中间的那些数据  def handle\_data(self, data):  #if self.c\_t is True:  #self.check.append("".join(data.split()))  #if "生物" in self.check:  #self.d\_t = True  if self.a\_t is True:  self.keys.append("".join(data.split()))  if "中文学名" in self.keys:  self.d\_t = True  elif self.b\_t is True:  self.values.append("".join(data.split()))  else:  pass  def handle\_endtag(self, tag):  self.a\_t=False  self.b\_t=False    g = os.walk(r'C:\Users\\Zekun Gao\Desktop\演示\\biology')  keys = []  values = []  for path,d,filelist in g:  for filename in filelist:  htmlfile = open(path + '/' + filename, 'r', encoding='utf-8')  htmlpage = htmlfile.read()  p=MyHTMLParser()  p.feed(htmlpage)  if p.d\_t is False:  continue  p.close  keys = p.keys  values = p.values  results = dict(zip(keys, values))  print(results)  ret\_path = path + '/result/'  if os.path.exists(ret\_path) == False:  os.mkdir(ret\_path)  fw = open(ret\_path + filename.split('.')[0] + '.json', 'w',encoding='utf-8')  a = json.dumps(results,indent=4,ensure\_ascii=False)  fw.write(a)  fw.close  os.system("pause") |

|  |
| --- |
| 格式处理：Biology2.py |
| # -\*- coding: utf-8 -\*-  import os  import json  levels = ["门", "纲","目","科","属",]  names = ["中文学名", "拉丁学名", "别称", "英文名"]  def get\_json\_files(direction):  res = []  for root, dirs, files in os.walk(direction):  for file in files:  if file.endswith(".json"):  res.append(os.path.join(root, file))  return res  def dump\_into\_json1(files):  animal = {"name": "动物界", "children": []}  for file\_name in files:  with open(file\_name, "r",encoding='utf-8') as f:  biology = json.load(f)  if biology.get("界") == "动物界":  process\_json(animal, biology)  with open("result1.json", "w",encoding='utf-8') as f:  json.dump(animal, f, indent=4, ensure\_ascii=False)  f.write("\n")  def dump\_into\_json2(files):  plant = {"name": "植物界", "children": []}  for file\_name in files:  with open(file\_name, "r",encoding='utf-8') as f:  biology = json.load(f)  if biology.get("界") == "植物界":  process\_json(plant, biology)  with open("result2.json", "w",encoding='utf-8') as f:  json.dump(plant, f, indent=4, ensure\_ascii=False)  f.write("\n")  def process\_json(world, bio\_json):  for l in levels:  found = False  for child in world.get("children"):  if child.get("name") == bio\_json.get(l, " "):  world = child  found = True  break  if not found:  cur = {  "name":bio\_json.get(l, " "),  "children":[],  }  world.get("children").append(cur)  world = cur  for name in names:  if bio\_json.get(name):  if name == "别称" and bio\_json[name].count("、"):  inner\_names = bio\_json[name].split("、")  for i in range(len(inner\_names)):  world["children"].append({  "name": name+str(i+1),  "children": [  {  "name": inner\_names[i]  }  ]  })  continue  world["children"].append({  "name" : name,  "children": [  {  "name":bio\_json[name]  }  ]  })  if \_\_name\_\_ == "\_\_main\_\_":  dump\_into\_json1(get\_json\_files("./result"))  dump\_into\_json2(get\_json\_files("./result")) |

|  |
| --- |
| 结果显示（动物）：display1.html |
| <html>  <head>  <meta charset="utf-8">  <title>Tree</title>  </head>    <style>    .node circle {  fill: #fff;  stroke: steelblue;  stroke-width: 1px;  }    .node {  font: 12px sans-serif;  }    .link {  fill: none;  stroke: #ccc;  stroke-width: 1px;  }    </style>  <body>  <script src="http://d3js.org/d3.v3.min.js"></script>  <script>    var width = 2000,  height = 2000;    var tree = d3.layout.tree()  .size([width, height-200])  .separation(function(a, b) { return (a.parent == b.parent ? 1 : 2) / a.depth; });    var diagonal = d3.svg.diagonal()  .projection(function(d) { return [d.y, d.x]; });    var svg = d3.select("body").append("svg")  .attr("width", width)  .attr("height", height)  .append("g")  .attr("transform", "translate(40,0)");        d3.json("result1.json", function(error, root) {    var nodes = tree.nodes(root);  var links = tree.links(nodes);    console.log(nodes);  console.log(links);    var link = svg.selectAll(".link")  .data(links)  .enter()  .append("path")  .attr("class", "link")  .attr("d", diagonal);    var node = svg.selectAll(".node")  .data(nodes)  .enter()  .append("g")  .attr("class", "node")  .attr("transform", function(d) { return "translate(" + d.y + "," + d.x + ")"; })    node.append("circle")  .attr("r", 4.5);    node.append("text")  .attr("dx", function(d) { return d.children ? -16 : 16; })  .attr("dy", 5)  .style("text-anchor", function(d) { return d.children ? "end" : "start"; })  .text(function(d) { return d.name; });  });    </script>    </body>  </html> |

|  |
| --- |
| 结果显示（植物）：display2.html |
| <html>  <head>  <meta charset="utf-8">  <title>Tree</title>  </head>    <style>    .node circle {  fill: #fff;  stroke: steelblue;  stroke-width: 1px;  }    .node {  font: 12px sans-serif;  }    .link {  fill: none;  stroke: #ccc;  stroke-width: 1px;  }    </style>  <body>  <script src="http://d3js.org/d3.v3.min.js"></script>  <script>    var width = 2000,  height = 2000;    var tree = d3.layout.tree()  .size([width, height-200])  .separation(function(a, b) { return (a.parent == b.parent ? 1 : 2) / a.depth; });    var diagonal = d3.svg.diagonal()  .projection(function(d) { return [d.y, d.x]; });    var svg = d3.select("body").append("svg")  .attr("width", width)  .attr("height", height)  .append("g")  .attr("transform", "translate(40,0)");        d3.json("result2.json", function(error, root) {    var nodes = tree.nodes(root);  var links = tree.links(nodes);    console.log(nodes);  console.log(links);    var link = svg.selectAll(".link")  .data(links)  .enter()  .append("path")  .attr("class", "link")  .attr("d", diagonal);    var node = svg.selectAll(".node")  .data(nodes)  .enter()  .append("g")  .attr("class", "node")  .attr("transform", function(d) { return "translate(" + d.y + "," + d.x + ")"; })    node.append("circle")  .attr("r", 4.5);    node.append("text")  .attr("dx", function(d) { return d.children ? -16 : 16; })  .attr("dy", 5)  .style("text-anchor", function(d) { return d.children ? "end" : "start"; })  .text(function(d) { return d.name; });  });    </script>    </body>  </html> |

|  |
| --- |
| 种子提取程序：words.py |
| # \_\*\_ coding: utf-8 \_\*\_  # time 2018/3/28  # target url 'http://jinyici.xpcha.com/'  # use requests lxml  # 多进程 反义词和近义词同时爬取  # 使用Manager 数据共享  import json  import requests  import sys  from lxml import etree  from multiprocessing import Process, Manager  def headers():  return {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) "  "Chrome/62.0.3202.94 Safari/537.36"}  def get\_url(url="http://jinyici.xpcha.com/list\_0.html"):  try:  response = requests.get(url, headers=headers())  except Exception as f:  print("出现错误1")  print(f.args)  sys.exit(1)  if response.status\_code != 200:  print(url)  print("出现错误2")  sys.exit(1)  html = response.text  etree\_html = etree.HTML(html)  all\_a = etree\_html.xpath("//dl[@class='shaixuan\_5']/dd/a")  head = url.split("com")[0] + "com/"  words\_link = {a.xpath("./text()")[0]: head + a.xpath('./@href')[0] for a in all\_a}  return words\_link  def write\_json(words\_link):  dict\_store = dict()  for word, link in words\_link.items():  print(word, link)  try:  response = requests.get(link, headers=headers())  except Exception as f:  print("出现错误3")  print(f.args)  sys.exit(1)  if response.status\_code != 200:  print("出现错误4")  sys.exit(1)  etree\_html = etree.HTML(response.text)  all\_span = etree\_html.xpath("//dl[@class='shaixuan\_1']/dd/span")  all\_words = [span.xpath("./text()")[0].strip("：") for span in all\_span]  dict\_store[word] = all\_words  return dict\_store  def jyc(words\_store, maxPage):  words\_list = []  for page in range(1, maxPage):  url = "http://jinyici.xpcha.com/list\_0\_{}.html".format(page)  words\_link = get\_url(url=url)  dict\_store = write\_json(words\_link)  words\_list.append({'page{}'.format(page): dict\_store})  words\_store['近义词'] = words\_list  def fyc(words\_store, maxPage):  words\_list = []  for page in range(1, maxPage):  url = "http://fanyici.xpcha.com/list\_0\_{}.html".format(page)  words\_link = get\_url(url=url)  dict\_store = write\_json(words\_link)  words\_list.append({'page{}'.format(page): dict\_store})  words\_store['反义词'] = words\_list  if \_\_name\_\_ == "\_\_main\_\_":  multiprocessing.freeze\_support()  all\_words = Manager().dict()  count = input("Please input the number of the pages:")  count = int(count)  count = count+1  #count = 2 # 5为4页以此类推 反义词和近义词共同的页数  p\_jyc = Process(target=jyc, args=(all\_words, count))  p\_fyc = Process(target=fyc, args=(all\_words, count))  p\_fyc.start()  p\_jyc.start()  p\_fyc.join()  p\_jyc.join()  all\_words = dict(all\_words)  with open("words.txt", "wb+") as f:  f.write(json.dumps(all\_words, ensure\_ascii=False).encode("utf-8"))  print("\n\n完成!!!!") |

|  |
| --- |
| 模式提取和词对获取程序：run.py |
| # -\*- coding: utf-8 -\*-”  import os  import re  import codecs  import math  from pyquery import PyQuery  know\_pattern\_lines = set() # store the known pair words  unknow\_pattern\_words = set() # store the unknow pair words  patterns = set() # store patterns  middles = []  prefixs = []  suffixs = []  def read\_word\_pairs(words\_file):  """  read word pairs from file  :param words\_file: the file path  :return:  """  word\_pairs = []  with codecs.open(words\_file, "r", "utf-8") as f:  for line in f:  tokens = line.strip().split(u":")  w1 = tokens[0].strip().split(u"\"")[1]  #print 'w1'+w1  for w in tokens[-1].strip().split(u","):  if w is not None and len(w.strip()) > 0:  w2 = w.split(u"\"")[1]  #print 'w2'+w2  word\_pairs.append((w1, w2))  return word\_pairs  def get\_patterns(word\_pairs):  """  find patterns  :return:  """  print u"find patterns"  middle1 = []  prefix1 = []  suffix1 = []  middle\_number = []  prefix\_number = []  suffix\_number = []  middle\_choose = []  prefix\_choose = []  suffix\_choose = []  pattern = []  display = []  words\_number = []  w11 = []  w22 = []  order = []  rank = []  pattern\_number = []  pattern\_words\_number = []  number = []  rp = []  i = -1  k = 0  j = 0  #x = 1  m = 0  for doc in os.listdir(u"./datas"):  if u"html" not in doc:  continue  doc\_file = u"./datas/" + doc  content = PyQuery(open(doc\_file).read()).text() # delete the html tags  #content = chr(content)  #content.encode('utf-8')  #print content  #f = codecs.open('./test.txt', 'w', 'utf-8')  #f.write(content)  lines = re.split(u'。|，|：|？|\?|;|,|!|\n', content.strip())  for w1, w2 in word\_pairs:  m += 1  for line in lines: # to find pattern line by line  line = line.strip()  if len(line):  f = codecs.open('./test1.txt', 'w', 'utf-8')  f.write(line)  if line is None or len(line) < 1:  continue  for w1, w2 in word\_pairs:  words\_number.append(0)  pos1 = line.find(w1)  pos2 = line.find(w2)  if pos1 == -1 or pos2 == -1: # the words not in a line at the same time  continue  print w1, w2  i += 1  words\_number[i] += 1  if pos1 < pos2:  print doc\_file    pattern\_number.append(0)  w11.append(w1)  w22.append(w2)  middle = line[pos1 + len(w1): pos2]  prefix = line[0:line.rfind(w1)]  t =line.find(w2)+len(w2)  suffix = line[t:]  order.append("True")  #print i  # if i >= 1:  #for j in range (0,i+1):  # if middle in middle1:  #if prefix in prefix1:  #if suffix in suffix1:  #x = 0  if middle.strip() == '':  middle1.append('')  else:  middle1.append(middle)  if prefix.strip() == ' ':  prefix1.append('')  else:  prefix1.append(prefix)  if suffix.strip() == '':  suffix1.append('')  else:  suffix1.append(suffix)  print "middle"+middle1[i]  print "prefix"+prefix1[i]  print "suffix"+suffix1[i]    #display.append((prefix,w1,middle,w2,suffix))  #print display[i]  print "display:"+prefix+w1+middle+w2+suffix+" order:True"  else:  print doc\_file  i += 1  pattern\_number.append(0)  w11.append(w1)  w22.append(w2)  middle = line[pos2 + len(w2): pos1]  prefix = line[0:line.rfind(w2)]  suffix = line[-1:line.find(w1)]  order.append("False")  if middle.strip() == '':  middle1.append('')  else:  middle1.append(middle)  if prefix.strip() == ' ':  prefix1.append('')  else:  prefix1.append(prefix)  if suffix.strip() == '':  suffix1.append('')  else:  suffix1.append(suffix)  print "middle"+middle1[i]  print "prefix"+prefix1[i]  print "suffix"+suffix1[i]  #for j in range (0,i+1):  #if middle in middle1[j]:  #if prefix in prefix1[j]:  #if suffix in suffix1[j]:  #del middle1[middle]  #del prefix1[prefix]  #del suffix1[suffix]  #x = 0    print "display:"+prefix+w2+middle+w1+suffix+" order:False"  for line in lines:  if middle in middle1:  if prefix in prefix1:  if suffix in suffix1:  same1 = line.find(middle1[i])  same2 = line.find(prefix1[i])  same3 = line.find(suffix1[i])  if same1 == -1 or same2 == -1 or same3 == -1:  continue  else:  pattern\_number[i] += 1  if middle.strip() < 1:  continue  if middle.strip() in [u"。", u'，', u'；', u'：', u'!', u'、', u'.', u',']:  continue  #print "0"+prefix1[0]  #print "1"+prefix1[1]    j = 0  #for j in range (0,i+1):  #print pattern\_number[j]  j = 0  for j in range (0,i+1):  middle\_number.append(0)  prefix\_number.append(0)  suffix\_number.append(0)  k = 0  for k in range (0,i+1):  if middle1[j] == middle1[k]:  middle\_number[j] += 1  if prefix1[j] == prefix1[k]:  prefix\_number[j] += 1  if suffix1[j] == suffix1[k]:  suffix\_number[j] += 1  k += 1  x = middle\_number[j]  x = float(x)  y = prefix\_number[j]  y = float(y)  z = suffix\_number[j]  z = float(z)  q = min(x,y,z)  pattern\_words\_number.append(q)  #print pattern\_words\_number[j]  #print words\_number[j]  #print middle\_number[j]  #print prefix\_number[j]  #print suffix\_number[j]  j += 1  j = 0  j = 0  for j in range (0,i+1):  x = pattern\_number[j]  x = float(x)  #print x  y = words\_number[j]  y = float(y)  # print y  z = middle\_number[j]  z = float(z)  #print z  q = math.log(z/(x\*y))    if q == 0:  q = -1  print q    k = 0    for k in range (0,j):  t = number[k]  t = float(t)  if q > t:  max1 = q  if q == t:  max1 = q  #print max1  #number[k] = number[k] + t  #number.append(0)  #j += 1  number.append(q)  #print max1  j = 0  for j in range (0,i+1):  if max1 == 0:  max1 = -1  x = float(number[j])  #print x  y = x/max1  #print y  t = y/m  #print t  rp.append(t)  print rp[j]  #print i  count = input("Please input the times:")  count = int(count)  for j in range (0,i+1):  if middle\_number[j] >= count:  if prefix\_number[j] >= count:  if suffix\_number[j] >= count:  suffix\_choose.append(suffix1[j])  middle\_choose.append(middle1[j])  prefix\_choose.append(prefix1[j])  j += 1  #y = 0  #for j in range (0,i+1):  # t = middle\_number[j]  # t = int(t)  # if      t = len(middle\_choose)  j = 0  if len(middle\_choose):  for j in range (0,t):#display  #if len(middle\_choose[j]) > 5:  #if middle\_choose[j] in middle\_choose:  #continue  #print "choose middle"+middle\_choose[j]  #print "choose prefix"+prefix\_choose[j]  #print "choose suffix"+suffix\_choose[j]  middles.append(middle\_choose[j])  prefixs.append(prefix\_choose[j])  suffixs.append(suffix\_choose[j])  print "display:"  print "nature:synonymy"  print "prefix:"+prefix\_choose[j]  print "w1:"+w11[j]  print "middle:"+middle\_choose[j]  print "w2:"+w22[j]  print "suddix:"+suffix\_choose[j]  #print "order:"+order[j]  print "order:"+order[j]  #print "rank:"+rank  #t = middle\_number[j]  #t = int(t)  print "times:",  #print 1  #j += 1  #print "display:"  #print "nature:"+"synonymy"  #print "prefix:"+prefix\_choose[0]  #print "middle:"+middle\_choose[0]  #print "suffix:"+suffix\_choose[0]  #print "word:"  j = 0  for j in range (0,i+1):#make pattern-1  #print u"find pattern [" + middle\_choose[j] + u"]"    j += 1  know\_pattern\_lines.add(line)  def find\_new\_word\_pairs():  """  find new word pairs in datas  :return:  """    print u"find new word pairs with patterns"  for doc in os.listdir(u"./datas"):  if u"html" not in doc:  continue  doc\_file = u"./datas/" + doc  print doc\_file  content = PyQuery(open(doc\_file).read()).text() # delete the html tags  lines = re.split(u'。|，|：|？|\?|;|,|!|\n', content.strip())  for line in lines: # to find pattern line by line  i = 0  line = line.strip()  if line is None or len(line) < 1:  continue  for mid in middles:  mids = line.find(middles[i])  pres = line.find(prefixs[i])  sufs = line.find(suffixs[i])  if mids != -1 and pres!= -1 and sufs != -1:  # if line in know\_pattern\_lines:  # continue    w1 = line[:mids]  w2 = line[mids + len(middles[i]):]  if len(prefixs[i]) != 0:      t = len(w1) - len(prefixs[i])  x = -t  w1 = w1[-t:]  if len(suffixs[i]) != 0:    t = len(w2) - len(suffixs[i])  w2 = w2[0:t]    if len(w1) >4:  continue  if len(w2) >4:  continue  if len(w1) > 1 and len(w2) > 1:  unknow\_pattern\_words.add(frozenset((w2, w1)))  i += 1  def main():  word\_pairs = read\_word\_pairs("words.txt")  get\_patterns(word\_pairs)  print u"#" \* 50  find\_new\_word\_pairs()  for tup in unknow\_pattern\_words:  tup = list(tup)  print "(" + tup[0] + ", " + tup[1] + ")"  if \_\_name\_\_ == "\_\_main\_\_":  main() |

## 附录3 外文文献译文

**从万维网中提取模式和关系**

**摘要：**万维网是一个巨大的信息资源库。与此同时，它的分布极其广泛。一种特定类型的数据，比如餐馆列表，可能会分散在成千上万的独立的信息源中。在本文中，我们考虑了从所有数据源中自动提取相应类型数据的关系的问题。我们提出了一种利用模式与关系的对偶性，从一个小样本开始增长目标关系的技术。为了测试我们的技术，我们使用它从万维网中提取(作者、标题)的关系对。

1. **介绍**

万维网提供了几乎所有类型的海量信息，从DNA库到简历再到最受欢迎的餐馆列表。然而，这些信息通常是分散在许多web服务器和主机上，并且使用许多不同的格式。如果这些信息可以从万维网中提取并集成到一个结构化的表单中，它们将形成前所未有的信息来源。它将包括最大的国际人口目录，最大和最多样化的产品数据库，最伟大的学术文献书籍和许多其他有用的资源。

使用特殊编码的封装器或过滤器(Tsi,MOS97)去整合大量的信息源是分量相当大的工作。但是，创建和通常使用数十个而不是数千个信息源是非常浪费时间的。在本文中，我们解决了从成千上万个在万维网中可能有微小关系的数据来源中提取关系的问题。我们的目标是发现信息来源，并从它们中要么完全自动要么利用非常少的人为干预来提取相关信息。

在本文中，我们考虑了从网络中提取图书的(作者、标题)关系对的问题。直观上，我们的解决方案如下所示。我们从一组(作者、标题)关系对开始并把它作为种子(在测试中我们只使用了一套五本书)。然后我们在网上找到所有出现这些书的地方。从这些出现的地方我们可以识别出这些书被引用的固定格式。然后我们在网上搜索这些模式并找到新的书籍。然后我们可以把这些书找出来并找到他们的出处，从中寻找到更多的模式。然后我们再利用这些模式找到更多的书籍，以此类推。最终我们可以得到一个长长的书单以及可以寻找到它们的模式。

1. **模式和关系的对偶性**

我们提出的方法被称为双反迭代模式关系。它依赖于我们下面解释的模式和关系之间的二元性。

**2.1 问题**

在这里我们更加正式的定义问题：

将D定义为一个庞大的非结构化信息数据库，例如万维网。让R = r1……rn是目标关系。每一个在R中的元组t都在D中出现了一次或多次。每一次这样的出现就构成了t的域，我们把它描绘成字符串，他们彼此在D当中的出现都靠的非常近（在网络中，这就意味着每个域之间在同一个网页中离的非常近）。

在本文我们研究的测试问题中，目标关系R是网络中出现的图书(作者、标题)这个关系对。显然，这并不是很好定义。然而，如果有一个潜在的作者和标题，以及他们在网上被提到的地方，一个人通常可以判断这是一本合法的书。

如果我们计算一个近似值，R’在R中的覆盖率为并且错误率为。我们的目标是最大化覆盖率并最小化错误率。然而，低错误率比高覆盖率要重要得多。对于一个效率很高的大数据库D，20%的撤销是可以被接受的。但是，超过10%的错误率对于许多应用程序来说可能是无用的。

代表性的，我们无法计算R，因此，我们不能不知道覆盖率和错误率的精确值。但是，我们可以通过让用户检查R0中的随机元素来检测错误率。覆盖的估计则要困难得多。

* 1. **模式**

直观地说，一个模式匹配一个特定的目标关系元组的出现格式。理想情况下，这个模式是足够特殊可以使他不去匹配到不在关系范围内的元组，但是在实践中一些错误的结果可能会出现。模式可能有各种各样的表现形式。在我们的工作中，我们使用了非常有限的正则表达式，更真实地说：

以p最为一种模式，M（D）p是在D中与p相匹配的一系列元组，|p|D是M（D）p中元素的数量。这样p的覆盖范围并且p的错误率为。

对于一组模式，，我们定义。我们将CD（P,R）和ED(P,R)进行类比。对于MD(P)的替代否定可能需要一个元组来匹配多个模式(参见第6节)。

* 1. **模式二元性的关系**

一个重要的观察是给定一组模式，P伴随着高覆盖率和低错误率，我们可以通过找到所有匹配的模式来构造一个很好的近似R。因此，给定一组好的模式，我们可以构建一组好的元组。但是，我们也希望拥有相反的属性——给定一组好的元组，我们可以构建一组好的模式。我们可以通过查找D中所有的元组的出现并在事件中发现相似点来做到这一点。从元组的模式和模式中找到元组的能力的组合给了我们很大的力量，也是我们在本文中提出的技术的基础。

1. **双迭代模式关系提取。**

双迭代模式关系提取- DIPRE是一种提取关系的技术，利用模式关系的二元性。它的工作原理如下:

1. R’<——样本

从一个小样本开始，R’的目标关系。这个示例是由用户提供的，可以非常小。在我们的测试中，我们使用了五本书的作者。

1. O<——发现出现集(R’,D)

然后，在我们的实验中，发现所有的R’元组在D中的出现。在我们的实验中，这些都是作者附近出现的，并且是一本书的标题。随着元组的发现，保持每个事件的上下文(url和周围的文本)。

1. P<——情报模式（O）

根据事件集合生成模式。这是算法的难点。粗略地说，这个例程必须为具有类似上下文的集合生成模式。模式需要有一个低的错误率，所以它们不太一般是很重要的。模式的覆盖率越高越好。但是，一个较低的覆盖率可以通过一个更大的数据库得到补偿

1. R’<——MD(P) 在数据库中搜索与任何模式匹配的元组。
2. 如果R’足够大，返回，并进行步骤2

**3.1 控制扩张**

上面的过程并不一定非常稳定，可能会偏离R，特别是MD(P)中的几个伪元组会在下一次迭代中导致P的一些伪模式。这反过来会导致大量的伪元组。出于这个原因，GenPatterns例程必须小心地减少潜在的伪元组(或多个小元组)所造成的损害。另一种安全措施是更严格地执行MD(P)，以便在P中需要元组来匹配多个模式。在我们所做的测试中，这第二项措施没有必要，但在将来的测试中可能是必要的。最后，随着关系的扩展，各种阈值可能需要波动。

1. **发现标题和作者**

在我们的实验中，我们选择了从万维网(World Wide Web)中计算(作者、标题)对的关系。这个问题特别适合DIPRE，因为在许多网站上都有许多著名的书籍。许多web站点在整个站点上遵循一种合理统一的格式。

**4.1 书的模式**

为了使用DIPRE来查找图书，有必要找出其中的模式。模式的定义很大程度上决定了DIPRE的成功。然而，在我们的测试中，我们使用了一个非常简单的模式。它需要进一步的调查，以确定更复杂的模式否认是否更好。

我们将模式视为一个元组(order, urlprex, prex, middle, sux)，其中order是一个布尔值，其他属性是字符串。如果order是true，那么(author,title)对匹配模式，如果在集合(WWW)中有一个与urlprefix\*匹配的URL，并且该URL与正则表达式匹配的文本匹配:



作者限于：



标题限于：



如果顺序是假的，那么标题和作者就被交换了。

* 1. **事件**

我们还必须指出一个事件的结构，因为它应该与一个模式的出现相对应。一个(作者，标题)对的出现包括一个7元组:



该顺序对应于标题和作者在文本中出现的顺序。url是它们所发生的文档的url。前缀包含了作者之前的m字符(在测试m中是10)。中间是作者和标题之间的文本，后缀由标题(或作者)后面的m字符组成。

* 1. **生成书的模式**

DIPRE过程的一个重要组成部分是GenPatterns例程，它采用一系列的书籍，并将它们转换为一个模式列表。这是一个非常重要的问题，整个模式识别都致力于解决这个问题的一般版本。但是，出于我们的目的，我们使用一组简单的启发式方法来生成来自事件的模式。只要有很少的假阳性(产生非书籍的模式)，这是有先见之明的。每个模式只需要很小的覆盖率，因为web是巨大的，并且有许多信息来源，所以所有模式的覆盖范围仍然是巨大的。

假设我们给定了一组事件，我们希望构造一个特定的模式，以匹配所有这些事件。我们可以这样做:

1. 验证所有事件的order和middle是相同的。如果没有，就不可能生成匹配它们的模式。设置outpattern.order和outpattern.middle让order和middle分别排列。
2. 找到所有url的最长匹配前缀。设置outpattern.urlprefix前缀。
3. 设置outpattern.prefix的最长匹配后缀的出现的前缀。
4. 设置outpattern.suffix为出现的后缀的最长匹配的前缀。

我们表示这个为常规的GenOnePattern(O)。

模式特异性：上面生成的模式可能太笼统或太具体。我们并不担心它太过特殊，因为会有许多模式产生并结合在一起会有很多书。然而，这种模式可能过于笼统，可能会产生许多非书籍。

为了解决这一问题，我们试图测量该模式的特殊性。模式p的特殊性大致对应于，其中X是一些均匀分布在R的元组上的随机变量。为了快速计算，我们使用了一个模式特异性的公式（|s|为s的长度）：



我们拒绝任何具有过低的特异性的模式，这样就不会产生过于通用的模式。更具体地说，我们坚持认为specificity(p)n > t，其中n是支持模式p的事件的数量，t是一个阈值。这将确保模式的所有字符串都是非空的(否则特性为零)。我们还要求n > 1，因为基于一个例子的模式非常容易出错。

算法生成模式：这里，我们给出了GenPatterns(O)的算法。它利用了第4.3节引入的算法GenOnePattern(O)。

1. 按order和middle的的顺序将所有在O中出现的o按顺序排列，让结果的组是
2. 对于每个Oi组，。如果p满足特殊要求，则输出p，否则:

如果Oi中的所有o都有相同的URL，则拒绝Oi。

否则，将Oi中出现的事件与在其url中按字符分组的子组分开，这是一个过去的p.urlprefix。对这些子组重复步骤2中的过程。

这个例程使用一个基于url的简单的进一步细分，当生成的模式不是很明确的时候。你也可以想象使用前缀或后缀。

我们描述了一种简单的技术，可以从出现的书籍列表中生成模式。人们可以想象更复杂的技术，这是进一步研究的主题。然而，正如结果(第5节)所指出的，即使这个简单的方案也很有效。

* 1. **性能问题**

有两种非常苛刻的任务:一长串的书和找到模式匹配的书的出现。这两个操作都必须在一个非常大的Web文档数据库中进行。

对于第一个任务，查找书籍的出现，我们通过两个fgrep过滤器传递数据。一个只通过包含有效作者的行，而另一个只通过包含有效标题的行。在此之后，它是用Python编写的一个程序的任务，它实际上是检查行中是否有匹配的作者和标题，识别它们并生成作为输出的事件。在Flex和Python中有一些涉及大型正则表达式的替代方法，但它们很快就超出了各种内部界限。

对于第二个任务，我们只使用Python程序。每个模式都被转换成一对正则表达式，一个用于URL，一个用于实际发生。每个URL都首先经过测试，看看哪些模式适用于它。然后程序测试相关正则表达式的每一行。这种方法非常缓慢，需要改进。未来的版本可能使用Flex或rex C库。这一任务可以通过定位与模式匹配的URL而变得更容易，我们也尝试过这样做。无论如何，数据的结构都不是那么简单，我们希望我们开发的技术足够一般化，能够在URL上不受限制。

在这个时间点上，发生的模式的生成并不是一个性能问题，因为只有成千上万的事件发生。随着更大的测试运行，这将变得更加重要。目前，这些事件按order和middle顺序进行排序。然后，Python程序将读取结果列表并生成如第4.3节所述的模式。

**5.实验**

虽然到目前为止所进行的实验是非常有限的，由于时间的限制，它们产生了非常积极的结果。更多的实验正在进行中。

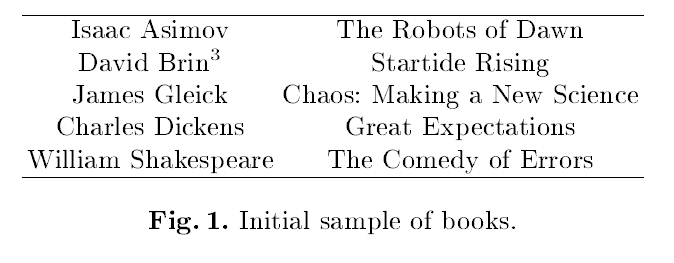
* 1. **用于实验的网络数据**

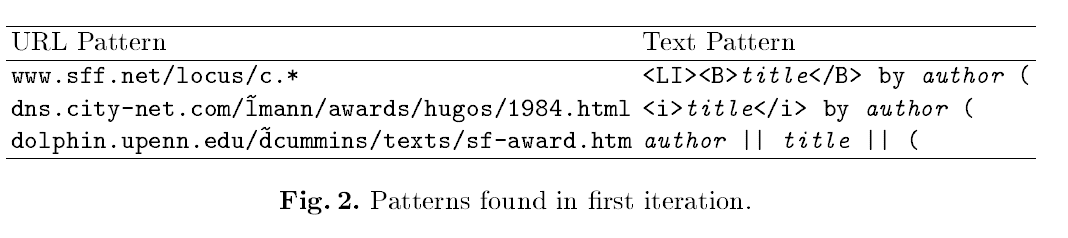
对于数据，我们使用了一个24,000,000个web页面的存储库，总计147gb。该数据是斯坦福网络基地的一部分，用于谷歌搜索引擎[BP]和其他研究项目。作为搜索引擎的一部分，我们构建了整个存储库的反向索引。

存储库跨越了许多磁盘和几台机器。即使不进行任何实质性的处理，也要花相当长的时间才能将数据传递给数据。因此，在这些中，我们只在任何给定的迭代中对存储库的子集进行传递。

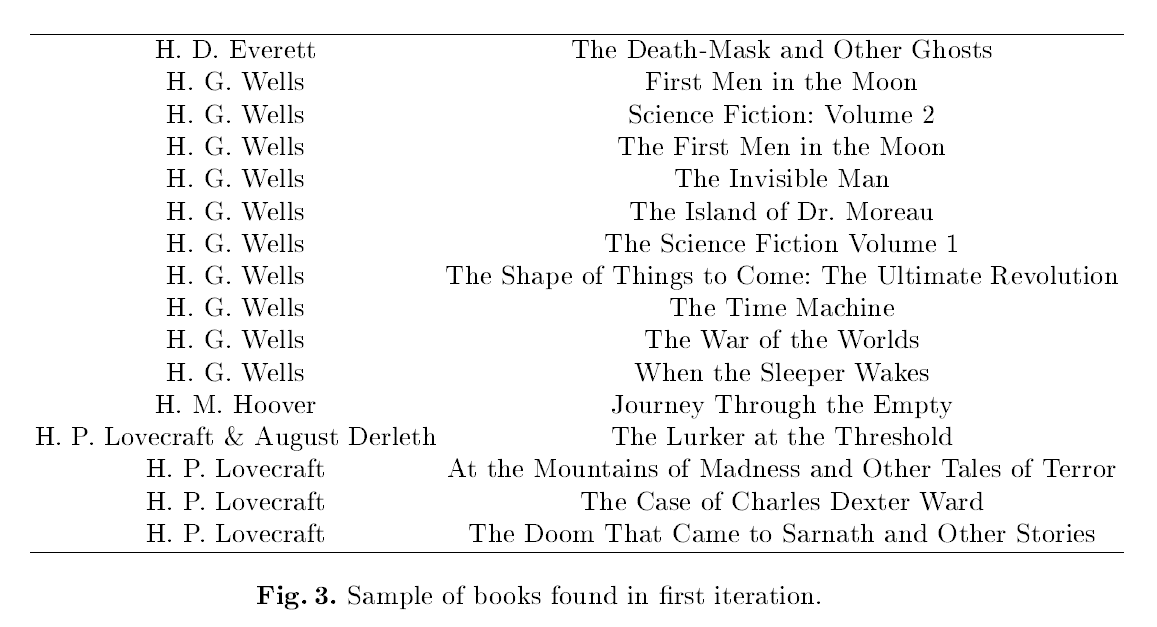
这个项目的一个重要提示是，存储库几乎不包含来自Amazon [Ama]的web页面。这是因为它们自动生成的url使爬行变得困难。

* 1. **扩张模式关系**





我们用5本书开始了这个实验(见图1)。这些书产生了199次，产生了3种模式(见图2)。有趣的是，这五本书中只有前两本书产生了这些模式，因为它们都是科幻小说。在匹配的URL上运行这些模式会产生4047个惟一的(作者、标题)对。它们大多是科幻小说，但也有例外。(见图3。



搜索了大约500万个网页，发现了3972个此类书籍。这个数字有点令人失望，因为它并不是第一次迭代中发生的大爆炸。但是，至少要花几天时间才能在整个存储库上运行，所以我们没有尝试生成更多。这些事件产生了105个模式，其中24个有url前缀，这些都不是完整的url。一个超过几百万个url的传递产生了9369个独特的(作者，标题)对。不幸的是，在这些书中有一些伪造的书。其中242个是合法的标题，但有一个“结论”的作者。我们把这些从名单上删除了。这是整个过程中唯一的人工干预。在未来的实验中，我们会很有趣地看到，这些东西是否会产生大量的垃圾。

、对于最终的迭代，我们选择使用包含工作簿的存储库子集。这包括大约15.6万份文件。扫描剩下的9127本书，产生了9938次。这些依次为346种模式。扫描同一组文件，生成了15257本独特的书籍，并没有多少伪造的数据。(见图4)

这个实验正在进行中，希望很快就会有一个更大的书籍列表。目前的一款是在线的[Bri]。

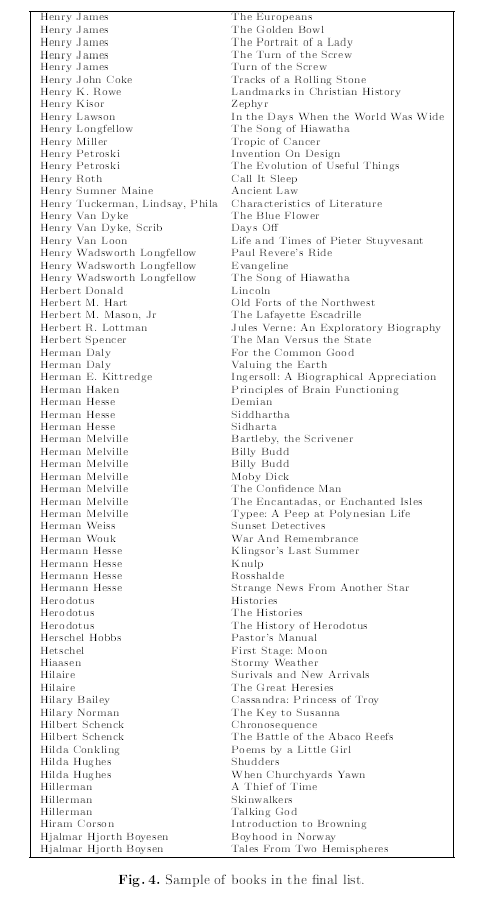
* 1. **结果的质量**

分析结果的质量,我们选择20个随机书籍的列表和试图确认他们实际的书通过在亚马逊上通过搜索(Ama),Visa购书指南 (Vis)、斯坦福大学网上图书馆目录,和Web.4测量结果的质量,20本中的19本都是真正的书。剩下的书实际上是一篇关于我为什么投票给一辆用户汽车的文章，由安德鲁·托拜厄斯(Andrew Tobias)著。

令人吃惊的是，除了网络之外，有些书没有在其他的资料中找到。其中一些是在线书籍;有些是模糊的或绝版的;有些网站根本没有在某些网站上列出原因。总共20本书中有5本不在亚马逊上，亚马逊声称拥有250万册图书。

除了上面提到的文章之外，数据还存在一些明显的问题。有些书因为小的差异，比如大写、间隔、作者的排列方式(例如，“E.R. Burroughs" 拼成 ”Edgar Rice Burroughs")。然而，幸运的是，作者们对于他们的名字是如何被列出的很特别，而且这些复制是有限的。在一些情况下，一些信息被附加到作者的名字，例如出版日期。

1. **结论**



我们的总体目标是能够利用它的广泛性，从整个万维网中提取结构化数据。在寻找书籍列表的简单例子中，DIPRE已经被证明是一个了不起的工具。它以一套5本书的样本开始，并将其扩展到一个相对高质量的列表，超过15000本书，并且很少有人干预。同样的工具也适用于其他一些领域，如电影、音乐、餐馆等等。这个工具的一个更复杂的版本可能能够提取人员目录、产品目录等等。

**6.1可延展性和稳态**

这种方法的可延展性有几个挑战。一种是在大型存储库上扫描大量模式和元组所需的每一项性能。基础算法和实现的改进很可能在不久的将来解决这个问题。

一个潜在的更大的障碍是，在扩展关系时，DIPRE是否可以避免偏离目标。例如，由于它只使用了种子样本中的两种科学功能书籍，为什么它没有产生大量的科学功能书籍。显然，它被所有书籍的汇编所吸引，甚至有一些零散的文章设法进入了这个关系。当关系展开时，保持这种效果是不平凡的，但有几种可能。

连接到单一的价值分解：一种可能是重新定义MD(P)，以要求多个模式匹配一个元组。一个更极端的版本是给每个元组和模式分配权重。匹配的元组根据匹配的模式的权重分配权重。生成的模式根据匹配它的元组的权重来分配权重。如果这是线性完成的，这个技巧将分解为tuple-pattern矩阵的奇异值分解(乘以它的转置)。这类似于在document-word矩阵上完成的潜在语义索引[DDF+90]。在这种情况下，最终的稳态是主要的特征向量。幸运的是，这与最初的样本无关，这显然是不可取的。尽管如此，与LSI的关系是引人注目的，并且需要进一步的调查。

即使不使用重量，从初始状态开始的稳定状态的独立性也可能是一个问题。有几种可能的解决方案。一个是只通过有限数量的迭代运行，就像我们在本文中演示的那样。另一种解决方案是确保元组转换成元组是非线性的，并且有一些依赖于初始状态的局部稳定状态。这可以通过在GenPatterns的计算中使用初始样本R’实现。在这种情况下，用户还可以提供一个R’，一个反例列表。

* 1. **自动提取的影响**

这个实验最令人惊讶的结果之一是找到了那些没有在主要的网上资料中列出的书，比如由道格拉斯·克拉克(Douglas Clark [Cla])出版的书，这本书是在网上出版的，或者是由杜兰·拉德福德(Dollie Radford)在1904年出版的一篇默默无闻的作品《少年园艺者的Kalendar》。如果图书列表可以扩展，并且如果几乎所有在线的图书都可以被提取，那么结果列表可能比任何现有的图书数据库都要完整。生成的列表将是数千个小型在线源的产品，而不是当前的图书数据库，后者是一些大型信息源的产品。信息流动的这种变化可能具有重要的社会影响。

## 附录4 外文文献原文

**Extracting Patterns and Relations from the World Wide Web**

**Abstract**. The World Wide Web is a vast resource for information. At the same time it is extremely distributed. A particular type of data such as restaurant lists may be scattered across thousands of independent information sources in many different formats. In this paper, we consider the problem of extracting a relation for such a data type from all of these sources automatically. We present a technique which exploits the duality between sets of patterns and relations to grow the target relation starting from a small sample. To test our technique we use it to extract a relation of (author, title) pairs from the World Wide Web.

**1 Introduction**

The World Wide Web provides a vast source of information of almost all types, ranging from DNA databases to resumes to lists of favorite restaurants. However, this information is often scattered among many web servers and hosts, using many different formats. If these chunks of information could be extracted from the World Wide Web and integrated into a structured form, they would form an unprecedented source of information. It would include the largest international directory of people, the largest and most diverse databases of products, the greatest bibliography of academic works, and many other useful resources.

There has been considerable work on integrating a number of information sources using specially coded wrappers or filters [Tsi, MOS97]. However, these can be time-consuming to create and are usually used for tens, not thousands of sources. In this paper, we address the problem of extracting a relation from the thousands of sources that may hold pieces of the relation on the World Wide Web. Our goal is to discover information sources and to extract the relevant in- formation from them either entirely automatically, or with very minimal human intervention.

In this paper, we consider the problem of extracting a relation of books (author, title) pairs from the Web. Intuitively, our solution works as follows. We begin with a small seed set of (author, title) pairs (in tests we used a set of just five books). Then we find all occurrences of those books on the Web. From these occurrences we recognise patterns for the citations of books. Then we search the Web for these patterns and find new books. We can then take these books and find all their occurrences and from those generate more patterns. We can use these new patterns to find more books, and so forth. Eventually, we will obtain a large list of books and patterns for finding them.

**2 The Duality of Patterns and Relations**

The method we propose is called DIPRE - Dual Iterative Pattern Relation Expansion. It relies on a duality between patterns and relations which we explain below.

**2.1 The Problem**

Here we define our problem more formally:

Let D be a large database of unstructured information such as the World Wide Web. Let R = r1,…..,rn be the target relation. Every tuple, t, of R occurs in one or more times in D. Every such occurrence consists of all the fields of t, represented as strings, occurring in close proximity to each other in D (in the case of the Web, this means all the fields are near each other, on the same Web page).

In the test problem we examine in this paper, the target relation R is the set of books (author, title) pairs that occur on the Web. Clearly, this is not well defined. However, given a potential author and title and where they are mentioned on the Web, a human can generally tell whether this is a legitimate book.

If we compute an approximation, R0 of R then the coverage is and the error rate is. Our goal is to maximize coverage and minimize the error rate. However, a low error rate is much more critical than high coverage. Given a sufficiently large database, D, a recall of just 20% may be acceptable. However, an error rate over 10% would likely be useless for many applications.

Typically, we cannot actually compute R. Therefore, we cannot not know the precise values of coverage and error rate. However, we can sample the error rate by having a user check random elements of R0. Coverage is much more difficult to estimate.

**2.2 Patterns**

Intuitively, a pattern matches one particular format of occurrences of tuples of the target relation. Ideally the pattern is specific enough not to match any tuples that should not be in the relation, however, in practice a few false positives may occur. Patterns may have various representations. In our work we used a very limited class of regular expressions. More formally:

Let p be a pattern. Then MD(p) is the set of tuples that match p in D and |p|D is the number of elements in MD(p). Then the coverage of p, and the error rate of p is 

For a set of patterns, , we define. We extend CD(P;R) and ED(P;R) analogously. Alternative definitions of MD(P) may require a tuple to match multiple patterns (see Section 6).

**2.3 Pattern Relation Duality**

An important observation is that given a set of patterns, P with high coverage and low error rate, we can construct a very good approximation to R simply by finding all matches to all the patterns. Thus, given a good set of patterns, we can build a good set of tuples. However, we also wish to have the converse property - given a good set of tuples, we can build a good set of patterns. We can do this by finding all occurrences of the tuples in D and discovering similarities in the occurrences. The combination of the ability to find tuples from patterns and patterns from tuples gives us great power and is the basis for the technique we propose in this paper.

**3 Dual Iterative Pattern Relation Extraction**

Dual Iterative Pattern Relation Extraction - DIPRE is a technique for extracting relations which makes use of pattern-relation duality. It works as follows:

1. R’<——Sample

Start with a small sample, R0 of the target relation. This sample is given by the user and can be very small. In our tests, we used a list of five books with authors.

2. O<——FindOccurrences(R’,D)

Then, find all occurrences of tuples of R0 in D. In our experiments, these were nearby occurrences of the author and the title of a book in text. Along with the tuple found, keep the context of every occurrence (url and surrounding text).

3. P<——GenPatterns(O)

Generate patterns based on the set of occurrences. This is the tricky part of the algorithm. Roughly speaking, this routine must generate patterns for sets of occurrences with similar context. The patterns need to have a low error rate, so it is important that they are not overly general. The higher the coverage of the patterns the better. However, a low coverage can be compensated for with a larger database.

4. R’<——MD(P). Search the database for tuples matching any of the patterns.

5. If R’ is large enough, return. Else go to step 2.

**3.1 Controlling Expansion**

The above process is not necessarily very stable and may stray away from R. In particular, several bogus tuples in MD(P) can lead to several bogus patterns in P in the next iteration. This in turn can cause a whole slew of bogus tuples. For this reason the GenPatterns routine must be careful to minimize the amount of damage caused by a potential bogus tuple (or several small tuples). Another measure of safety is to define MD(P) more stringently so as to require tuples to match multiple patterns in P. This second measure has not been necessary in the tests we have performed but may be necessary in future tests. Finally, various threshholds may need to fluctuate as the relation expands.

**4 Finding Authors and Titles**

For our experiments we chose to compute a relation of (Author,Title) pairs from the World Wide Web. This problem lends itself particularly well to DIPRE bbecause there are a number of well-known books which are listed on many web sites. Many of the web sites conform to a reasonably uniform format across the site.

**4.1 Patterns for Books**

In order to use DIPRE to find books, it is necessary to define what patterns consist of. The definition of a pattern largely determines the success of DIPRE. However, for our tests we used a very simple definition of a pattern. It requires further investigation to determine whether more sophisticated definitions of patterns work better.

We defined a pattern as a five-tuple: (order, urlprefix, prefix, middle, suxffi) where order is a boolean value and the other attributes are strings. If order is true, an (author,title) pair matches the pattern if there is a document in the collection (the WWW) with a URL which matches urlprefix\* and which contains text that matches the regular expression:



The author is restricted to:



The title is restricted to:



If order is false, then the title and author are switched.

**4.2 Occurrences**

We also have to define how an occurrence is structured since it should have a correspondance to the definition of a pattern. An occurrence of an (author,title) pair consists of a seven-tuple:



The order corresponds to the order the title and the author occurred in the text. The url is the URL of the document they occurred on. The prefix consists of the m characters (in tests m was 10) preceding the author (or title if the title was first). The middle is the text between the author and title and the suffix consists of the m characters following the title (or author).

**4.3 Generating Patterns for Books**

An important component of the DIPRE procedure is the GenPatterns routine which takes a set of occurrences of books and converts them into a list of patterns. This is a nontrivial problem and there is the entire field of pattern recognition devoted to solving the general version of this problem. For our purposes, however, we use a simple set of heuristics for generating patterns from occurrences. As long as there are few false positives (patterns that generate nonbooks) this is sufficient. Each pattern need only have very small coverage since the web is vast and there are many sources of information so the total coverage of all the patterns can still be substantial.

Suppose we are given a set of occurrences and we wish to construct a specific a pattern as possible that matches all of them. We can do this as follows:

1. Verify that the order and middle of all the occurrences is the same. If not, it is not possible to generate a pattern to match them all. Set outpattern.order and outpattern.middle to order and middle respectively.
2. Find the longest matching prefix of all the urls. Set outpattern.urlprefix to that prefix.
3. Set outpattern.prefix to the longest matching suffix of the prefix's of the occurrences.
4. Set outpattern.suffix to the longest matching prefix of the suffix's of the occurrences.

We denote this routine GenOnePattern(O).

Pattern Specificity A pattern generated like the above can be too general or too specific. We are not concerned about it being too specific since there will be many patterns generated and combined there will be many books. However, the pattern may be too general and may produce many nonbooks.

To combat this problem we attempt to measure the specificity of the pattern. The specificity of a pattern p roughly corresponds to where X is some random variable distributed uniformly over the domain of tuples of R. For quick computation, we used the following formula for the specificity of a pattern (jsj denotes the length of s):



We reject any patterns with too low a specificity so that overly general patterns aren't generated. More specifically, we insist that specificity(p)n > t where n is the number of books with occurrences supporting the pattern p and t is a threshhold. This ensures that all the strings of a pattern are nonempty (other-wise the specificity is zero). Also we require that n > 1 since basing a pattern on one example is very error-prone.

Algorithm for Generating Patterns Here, we present the algorithm for GenPatterns(O). It takes advantage of the algorithm GenOnePattern(O) introduced in Section 4.3.

1. Group all occurrences o in O by order and middle. Let the resulting groups be O1,……,Ok.
2. For each group Oi, p<—GenOnePattern(Oi). If p meets the specificity requirements then output p. Otherwise:

If all o in Oi have the same URL then reject Oi.

Else, separate the occurrences o in Oi into subgroups grouped by the character in their urls which is one past p.urlprefix. Repeat the procedure in step 2 for these subgroups.

This routine uses a simple further subdivision based on the url when the pattern generated is not sufficiently specific. One can also imagine using the prefix or the suffix.

We have described a simple technique for generating patterns from lists of occurrences books. One can imagine far more sophisticated techniques and this is the subject of further research. However, as is indicated by the results (Section 5) even this simple scheme works well.

**4.4 Performance Issues**

There are two very demanding tasks DIPRE-finding occurrences of books given a long list of books and finding pattern matches given a list of patterns. Both of these operation must take place over a very large database of Web documents.

For the first task, finding occurrences of books, we first pass the data through two fgrep filters. One only passes through lines that contained a valid author and the other only passes through lines that contained a valid title. After this it is the task of a program written in Python to actually check that there are matching authors and titles in the line, identify them and produce occurrences as output. Several alternative approaches involving large regular expressions in Flex and in Python were attempted for this purpose but they quickly exceeded various internal bounds.

For the second task, we use just a Python program. Every pattern is trans-lated into a pair of regular expressions, one for the URL, and one for the actual occurrence. Every URL is first tested to see which patterns apply to it. Then the program tests every line for the relevant regular expressions. This approach is quite slow and needs to be improved. Future versions are likely to use Flex or the rex C library. This task can be made somewhat easier by targeting just the URL's which match the patterns and we made some attempt to do this. However, the data is not structured to make that completely trivial and we wish the techniques we develop to be general enough to be able to handle no restrictions on URL's.

The generation of patterns from occurrences is not much of a performance issue at this point in time because there are only thousands of occurrences generated. As larger tests are run this will become more important. Currently, the occurrences are sorted using gsort by order and middle. Then a Python program reads through the resulting list and generates patterns as described in Section 4.3.

**5 Experiments**

While the experiments performed so far have been very limited, due to time constraints they have produced very positive results. Many more experiments are in progress.

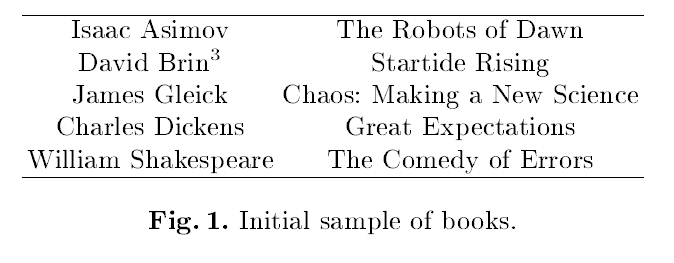
**5.1 Web Data Used in Experiments**

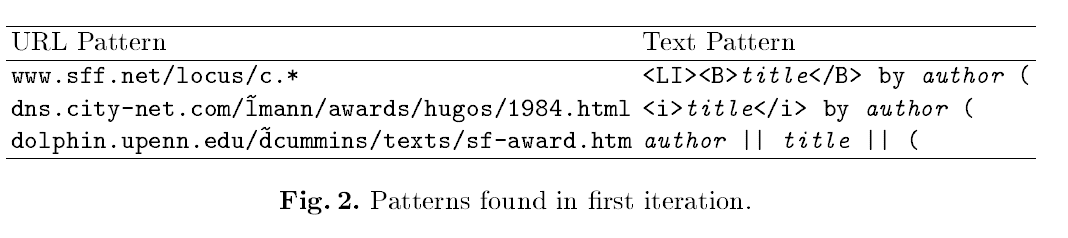
For data we used a repository of 24 million web pages totalling 147 gigabytes. This data is part of the Stanford WebBase and is used for the Google search engine [BP] and other research projects. As a part of the search engine, we have built an inverted index of the entire repository.

The repository spans many disks and several machines. It takes a considerable amount of time to make just one pass over the data even without doing any substantial processing. Therefore, in these we only made passes over subsets of the repository on any given iteration.

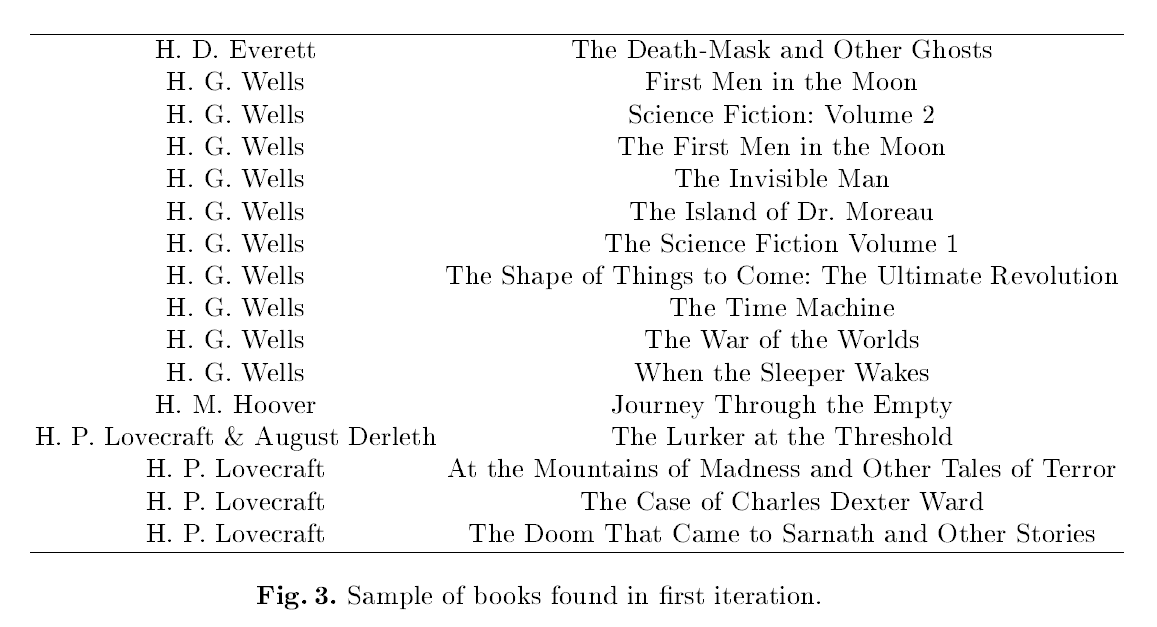
An important note for this project is that the repository contains almost no web pages from Amazon [Ama]. This is because their automatically generated urls make crawling difficult.

**5.2 Pattern Relation Expansion**





We started the experiment with just 5 books (see Figure 1). These produced 199 occurrences and generated 3 patterns (see Figure 2). Interestingly, only the first two of the five books produced the patterns because they were both science fiction books. A run of these patterns over matching URL's produced 4047 unique (author,title) pairs. They were mostly science fiction but there were some exceptions. (See Figure 3.



A search through roughly 5 million web pages found 3972 occurrences of these books. This number was something of a disappointment since it was not a large blowup as had happened in the first iteration. However, it would have at least a couple of days to run over the entire repository so we did not attempt to generate more. These occurrences produced 105 patterns, 24 of which had url prefixes which were not complete urls. A pass over a couple million urls produced 9369 unique (author, title) pairs. Unfortunately, there were some bogus books among these. In particular, 242 of them were legitimate titles but had an author of Conclusion". We removed these from the list. This was the only manual intervention through the whole process. In future experiments, it would be interesting to see whether leaving these in would produce an extraordinary amount of junk.

For the final iteration, we chose to use the subset of the repository which contained the work books. This consisted of roughly 156,000 documents. Scanning for the 9127 remaining books produced 9938 occurrences. These in turn generated 346 patterns. Scanning over the same set of documents produced 15257 unique books with very little bogus data. (See Figure 4)

This experiment is ongoing and hopefully, a larger list of books will be generated soon. The current one is available online [Bri].

**5.3 Quality of Results**

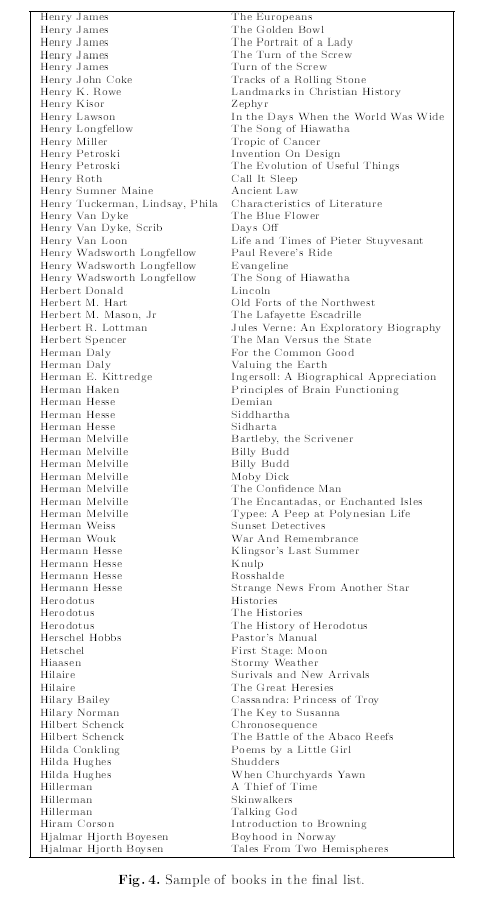
To analyse the quality of the results, we picked twenty random books out of the list and attempted to verify that they were actual books by searching on Amazon [Ama], the Visa Shopping Guide for books [Vis], the Stanford online library catalog, and the Web As a measure of the quality of the results, 19 of the 20 were all bonafide books. The remaining book was actually an article -“Why I Voted for a User Car", by Andrew Tobias.

The big surprise was that a number of the books were not found in some or all of the sources except for the Web. Some of these books were online books; some were obscure or out of print; some simply were not listed on some sites for no apparent reason. In total, 5 of the 20 books were not on Amazon which claims to have a catalog of 2.5 million books.

Other than the article mentioned above, there are a few visible problems with the data. Some books are mentioned several times due to small differences such as capitalization, spacing, how the author was listed (for example “E.R.Burroughs" versus”Edgar Rice Burroughs"). Fortunately, however, authors are quite particular about how their name is listed and these duplications are limited. In several cases, some information was appended to the author's name such as publication date.

**6 Conclusions**

Our general goal is to be able to extract structured data from the entire World Wide Web by leveraging on its vastness. DIPRE has proven to be a remarkable tool in the simple example of finding lists of books. It started with a sample set of 5 books and expanded it to a relatively high quality list of over 15,000 books with very minimal human intervention. The same tool may be applied to a number of other domains such as movies, music, restaurants, and so forth. A more sophisticated version of this tool is likely to be able to extract people directories, product catalogs, and more.



6.1 Scalability and Steady State

There are several challenges to the scalability of this method. One is the performance required to scan for large numbers of patterns and tuples over a huge repository. Improvements in the underlying algorithms and implementation are likely to solve this problem in the very near future.

A potentially more diffcult obstacle is whether DIPRE can be kept from diverging from the target as it expands the relation. For example, since it really used only the two science fiction books which were in the seed sample, why did it not produce a large list of science fiction books. Clearly, it gravitated to a compilation of all books and even a few scatterred articles managed to enter the relation. Keeping this effect under control as the relation expands is nontrivial but there are several possibilities.

Connection to Singular Value Decomposition One possibility is to redefine of MD(P) to require multiple patterns to match a tuple. A more extreme version of this is to assign a weight to every tuple and pattern. A matching tuple is assigned a weight based on the weights of the patterns it matches. A generated pattern is assigned a weight based on the weights of the tuples which match it. If this is done linearly, this technique breaks down to a singular value decomposition of the tuple-pattern matrix (multiplied by its transpose). This is analogous to Latent Semantic Indexing [DDF+90] which is done on the document-word matrix. In this case, the eventual steady state is the dominant eigenvector. Unfortunately, this is independent of the initial sample which is clearly not desirable. Nonetheless, the relationship to LSI is compelling and bears further investigation.

The independence of the steady state from the initial state above may also be a problem even without the use of weights. There are several possible solutions. One is to run only through a limited number of iterations like we demonstrated in this paper. Another solution is to make sure that the transformation of tuples to patterns to tuples is nonlinear and has some local steady states which depend on the initial state. This can be accomplished through the use of the initial sample R0 in the computation of GenPatterns. In this case, the user may also provide an R’, a list of counterexamples.

6.2 Implications of Automatic Extraction

One of the most surprising results of this experiment was finding books which were not listed in major online sources such as the book “Disbanded" by Douglas Clark [Cla] which is published online or “The Young Gardeners' Kalendar" by Dollie Radford [Rad04] an obscure work published in 1904. If the book list can be expanded and if almost all books listed in online sources can be extracted, the resulting list may be more complete than any existing book database. The generated list would be the product of thousands of small online sources as opposed to current book databases which are the products of a few large information sources. Such a change in information flow can have important social ramifications.