采用预测策略的Earley算法

谷 波 李 茹 刘开瑛
(山西大学计算机与信息技术学院 太原 030006)
(计算智能与中文信息处理教育部重点实验室 太原 030006)

摘 要 在自然语言处理中,句法分析主要有基于统计的方法和基于规则的方法。Earley算法是一种基于规则的方法,可以分析任意上下文无关文法(CFG),而不需要对文法进行修改。详细分析了Earley算法的特点。在通常的Earley算法中增加了多种预测机制,这些预测机制借鉴了LL,LR以及SLR等确定性分析算法的一些思想,并对这几种不同的预测机制及其组合在相同条件下进行了中文句法分析实验。结果显示,引入这些预测机制通常可以减少产生项目的数量,从而节省存储空间,减少运行时间。

关键词 上下文无关文法,句法分析,Earley算法
中图法分类号 TP391 **文献标识码** A

Earley Algorithm Using Prediction Strategies

GU Bo LI Ru LIU Kai-ying
(School of Computer and Information Technology, Shanxi University, Taiyuan 030006, China)
(Key Laboratory of Ministry of Education for Computation Intelligence and Chinese Information Processing, Taiyuan 030006, China)

Abstract There are two kinds of parsing algorithms in nature language processing: one based on statistics and the other based on grammar rules. Earley algorithm is based on grammar rules. It can parse any context free grammar (CFG) without changing the grammar. This paper used several predictive strategies in Earley algorithm. These strategies come from LL, SLR and LR algorithms. Experiments were made for these strategies and their combinations. Results indicate that these strategies can usually reduce number of items and make parsing faster.

Keywords Context free grammar, Parsing, Earley algorithm

1 引言

句法分析是自然语言处理领域中很重要的一个环节。它前承分词、词性标注,后接语义分析、篇章分析和信息抽取等。在中文信息处理中,中文分词和词性标注技术已经趋于成熟,语义分析和篇章分析也有许多学者在进行研究。但是中文句法分析难度较大且目前还不很成熟,因此已经成为制约语义分析和篇章分析等后续信息处理的瓶颈。句法分析的方法可以分为两类:一类是基于统计的方法[1],另一类是基于规则的方法。目前有许多新的统计机器学习方法应用于句法分析,如SVM[2],ME[3],MEMM[4]等。单纯使用统计方法难以将句子处理成嵌套形式的短语结构,因此这些方法通常进行的是浅层句法分析。基于规则的方法可以很方便地将短语组织成嵌套形式,从而实现完全句法分析。对于基于规则的方法而言,存在的困难主要有以下两个方面的因素。

第一,中文句法分析可使用的短语语料库比较匮乏。中文句法规则可以由语言专家人工给出,也可以从人工标记短语的语料库中进行自动抽取。前者实现起来比较困难,很难覆盖短语的组成情况,也很难获得其他一些有用的统计信息。后者实现相对容易,当语料库规模较大时,可以很好地覆盖实际中的短语构成情况,并且能够从中获取相关统计信息。但短语语料库除了分词和标注词性之外还要进行多层嵌套的短语标注,这对于标注人员来说是一件费时费力的事情。目前国内很难获得一个规模比较大的、可用的中文短语语料库。

第二,句法分析算法本身效率不高。许多高效的确定性算法都是针对上下文无关文法(缩写为CFG)的一些特定子集的,通常自然语言无法用这些子集来描述。确定性算法在其特定的CFG子集上的时间复杂度可以达到$O(n)$,其中n为输入字符串中字符的个数,如LR算法[5]等。在自然语言处理中使用的是一些针对CFG的通用句法分析算法,主要有CKY算法[6]、GLR算法[7]和Earley算法[8]等。这些算法的特点是能处理任意的CFG,但其时间复杂度较高,一般情况下为$O(n^3)$。CKY算法可从句子的任意位置开始进行分析,但要求文法必须是乔姆斯基范式(CNF)。虽然可以将任意一个CFG改为CNF,但在自然语言中这样做有可能改变整个短语类型标记体系和原有的短语结构。GLR算法是一种改进的LR算法,当LR分析表中出现冲突时,建立多条并行分析路径使得分析继续进行。GLR算法使用图栈和共享节点

到稿日期:2009-02-25 返修日期:2009-05-06 本文受国家863计划(2006AA01Z142),国家自然科学基金项目(60873128)资助。
谷 波(1978—),男,博士生,讲师,主要研究方向为自然语言处理等,E-mail:gubo@sxu.edu.cn;李 茹(1963—),女,教授,主要研究方向为人工智能、自然语言处理等;刘开瑛(1931—),男,教授,博士生导师,主要研究方向为自然语言处理等。

包的方式来节省存储空间，但是当需要从共享节点包回溯找出所有的句法分析树时，仍需要额外花费 $O(n^3)$ 数量级的时间。相比之下，Earley 算法易于实现，不需要改造文法，也不需要事先建立 LR 分析表。

本文分析了 Earley 算法的特点，采用一些预测策略避免了许多无用项目的出现。实验结果显示，这样做不但节省了存储空间，而且减少了算法的查找时间，从而提高了 Earley 算法的分析效率。本文第 2 节对 Earley 算法做简要描述，并引出一些术语；第 3 节从原理上对 Earley 算法和确定性 LR 算法以及 LL 算法进行分析比较；第 4 节提出改进后的 Earley 算法；第 5 节是实验结果和分析。

2 Earley 算法概述

对文中术语做一些约定。称 CFG 的规则右部任意位置加入一个点为一个 LR(0)项目，形如 A→…●…(“…”表示任意由终结符和非终结符组成的串且可为空串)；形如[LR(0)项目，状态编号] 的一个二元组称为一个 Earley 项目(简称项目)；一个 Earley 状态(简称状态)是 Earley 项目的集合。状态从 0 开始编号，并称之为初始状态，之后每分析过一个终结符产生新的状态且状态编号加 1。

Earley 算法有 3 个动作，描述如下(VN 表示非终结符集，VT 表示终结符集，S_i 表示编号为 i 的状态)。

1)预测(Predictor)：若$[A\rightarrow X_1\cdots\bullet C\cdots X_m, j]\in S_i$，且 $C\in VN$，则对于文法中每个形如 $C\rightarrow Y_1\cdots Y_n$ 的规则，在状态 S_i 中增加新的形如$[C\rightarrow\bullet Y_1\cdots Y_n, i]$ 的项目(点在规则右部最左端的项目称为起始项目，包含下面的初始项目)；

2)完成(Completer)：若$[A\rightarrow X_1\cdots X_m\bullet, j]$(这种点在规则右部最右端的项目称为归约项目)$\in S_i$，且有$[B\rightarrow X_1\cdots\bullet A\cdots X_m, k]\in S_j$，则在 S_i 中增加新的形如$[B\rightarrow X_1\cdots A\bullet\cdots X_m, k]$的项目，这个新的项目称为 completed 项目；

3)扫描(Scanner)：若项目$[A\rightarrow X_1\cdots\bullet a\cdots X_m, j]\in S_i$，$a\in VT$，且当前读入的句子的符号 $x=a$，则在 S_{i+1} 中增加新的形如$[A\rightarrow X_1\cdots a\bullet\cdots X_m, j]$的项目。

文献[8]在提出 Earley 算法时，考虑了类似于 LR(k)的 look-ahead(称为展望符集)。在实际中为了简化算法的实现，通常没有考虑展望符集。自然语言的短语文法中通常没有形如 $A\rightarrow\varepsilon$(ε 为空串，A 为导空符号)的规则，因而本文不考虑对这种规则的处理。下面是 Earley 算法的描述，其中 S 是文法初始符，S'是新增非终结符，ϕ 表示空集。

给定输入字符串：$X=x_1\cdots x_n$($x_i\in VT$，$i=1,2,\cdots,n$)；

①$S_0=\{[S'\rightarrow\bullet S, 0]$(初始项目)$\}$；

②For $i=0$ to n do

顺序处理每一个项目 $s\in S_i$，对 s 依次使用如下可行的操作，直至不产生新的项目(新增项目在最后)；

预测(向 S_i 添加新的项目)

完成(向 S_i 添加新的项目)

扫描(向 S_{i+1} 添加新的项目)

If $S_{i+1}=\phi$ 退出循环；

End for

③If $i=n$ and $[S'\rightarrow S\bullet, 0]\in S_{n+1}$ 接受 X

Else 拒绝 X

3 Earley 算法分析

从 Earley 算法的描述中，可以看出 Earley 算法是一种自顶向下与自底向上相结合的分析算法。它从文法初始符出发，进行自顶向下的预测(Predictor)，并根据当前读入的终结符对项目进行自底向上的选择(Scanner)。另外 Predictor 是从左向右的推导过程，而 Completer 是从左向右的归约过程。

Predictor 是一种自顶向下预测推导的过程，因此这一过程可以结合 LL(k)[9]方法的预测筛选的思想。执行 Predictor 时，根据当前读入的符号对起始项目进行筛选。LL(k)分析算法对 LL(k)文法每次预测时都有唯一的一个规则被选出，因此可根据文法事先构造一张 LL(k)分析表，然后直接由这张表对句子进行确定的 LL(k)分析。Earley 算法分析的是任意的 CFG，但它具备了自顶向下预测规则的能力，因而 LL(k)算法的预测机制可以应用在 *Earley* 算法中。这虽然不会改变 *Earley* 算法的时间复杂度，但是许多情况下有可能极大地减少分析过程中状态产生的项目数目，这样做不但节省了存储空间，而且可以减少后续状态执行 *Completer* 操作时查找前面状态中的项目所花费的时间。

另一方面，*Earley* 算法和 *LR* 算法有很多相似之处。首先，*Earley* 项目在 *LR* 项目的基础上增加了引出该项目的项目所在的状态编号，而其他地方与 *LR* 项目完全一样。其次，*Earley* 状态的产生方式与 *LR* 项目集的产生相似。当前的 *Earley* 状态移过一个终结符后，产生下一个状态，然后对这个新产生的状态中的项目进行预测或归约，这一点与 *LR* 项目集产生方式相似。所不同的是 *Earley* 算法对于当前状态中的归约项目会直接进行 *Completer*，并在当前状态中增加新项目；*LR* 算法不在当前状态中进行归约，而是在算法执行时进行归约。因此在对归约项目进行 *Completer* 操作时，除了可以使用 *Earley* 最初提出的采用展望符集的方式进行预测外，还可以采用 *SLR* 算法[9]中的 *Follow* 集来进行预测。本文在 *Earley* 算法中考虑了 *LL*(1)、*LR*(1)和 *SLR*(1)这 3 种预测机制的使用。

4 采用预测的 Earley 算法

自然语言的句法分析通常不只是判定句子是否合法，获得其分析树也是很重要的。为了方便得到分析树，在算法中为每个项目增加了两个指针：“左指针”指向点移过一个非终结符(Completer)或终结符(Scanner)后产生该项目的项目(即左指针总是由当前项目指向点在前一个位置的项目)；“上指针”由 completed 项目(见第 2 节)指向产生该项目的归约项目(上指针总指向归约项目)。起始项目没有这两个指针。

对 Earley 算法的改进主要是采用了几种预测机制来减少无用项目的产生。由前面的分析得知，当 Earley 算法进行 Predictor 时，可以利用 LL 预测的思想来减少一些无用起始项目的出现；在进行 Completer 时，可以使用展望符集或 Follow 集来限制一些无用的归约。自然语言文法中通常没有导空符号，所以在求解和使用 First 集和 Follow 集时不用考虑导空符号。因为每向前多看一个就使得计算量成指数级增加，所以仅向前看一个输入的终结符。下面是使用的 3 种预测策略。

第一种策略，结合 LL 预测分析的思想，在 Predictor 操作

时对起始项目进行限制，称这种策略为 LL 策略。预先求出文法每个非终结符的 First 集，当进行 Predictor 操作时，检查非终结符所导出的每个规则，根据不同的情况来判断是否在当前状态中添加这个规则产生的起始项目。具体方法是：假设从输入句子中读到的当前终结符是 x，如果规则的右部第一个是终结符且是 x，或者规则的右部第一个是非终结符且该终结符的 First 集中含有 x，则对此规则产生新的起始项目并加入到当前状态中，否则不进行。

第二种策略，采用 Follow 集对 Completer 操作进行归约上的限制，称为 F 策略。事先求出每个非终结符的 Follow 集。当对当前状态中的归约项目进行 Completer 时，假设当前归约项目形如$[A\rightarrow X_1\cdots X_m\bullet,i]$，且当前读入的终结符为 x，若 A 的 Follow 集中含有 x，则对这个归约项目进行 Completer 操作，否则不进行。

第三种策略，采用展望符集对 Completer 操作进行归约上的限制，称为 L 策略。虽然展望符集的计算相对 Follow 集而言更为复杂，但展望符集更为精确地指出了归约之后非终结符后面可能出现的终结符。原始的 Earley 算法中，从每个起始项目开始就计算其展望符集；移过非终结符或终结符时，将展望符集传给后面的项目；遇到归约项目时，根据归约项目的展望符集中是否有当前输入的终结符决定是否对其实行 Completer。通过分析对展望符集的计算过程做了如下一些改进。

非归约项目的展望符集对其自身并没有作用，它们只是起到向后面的项目传递展望符集的作用。这些展望符集一方面会浪费存储空间，另一方面每次的传递也会增加时间上的开销。所以并不是从一开始就计算每个起始项目的展望符集，而是将展望符集的计算推迟到遇到形如$[A\rightarrow X_1\cdots\bullet X_m,i]$的项目时才进行，称这种项目为“次归约项目”。这样，当进行归约时，通过归约项目，左指针查找其对应的次归约项目即可获得展望符集。之所以求次归约项目的展望符而不是直接去求归约项目的展望符，是因为如果遇到归约项目时再去计算展望符集，可能导致展望符集不能直接求得。例如，当遇到项目$t=[A\rightarrow X_1\cdots X_m\bullet,j]$时计算展望符集，那么要考虑 S_j 状态中所有点后面是 A 的项目。把这些项目分为两类：一类形如项目$s=[B\rightarrow\cdots\bullet AC\cdots,i]$，另一类形如$r=[B\rightarrow\cdots\bullet A,i]$。因为自然语言文法通常不含导空符号，所以对于前一类，如果 C 是终结符，就将 C 加入到项目 t 的展望符集；如果 C 是非终结符，就将 First(C)并入项目 t 的展望符集。对于后一类情况，应该将 r 的展望符集并入 t 的展望符集，但此时 r 的展望符集没有事先计算出来，所以这时候无法继续去求 t 的展望符集，这说明次归约项目 r 的展望符集应该提前求出。因此，将展望符集的计算推迟到出现次归约项目时再求次归约项目的展望符集，就不存在上述求不到的问题。若次归约项目有了展望符集，那么归约项目就可以根据 left 指针直接找到次归约项目，获得其展望符集。下面是改进后的求展望符集的算法。

1)对于初始项目$[S'\rightarrow\bullet S,0]$，其展望符集为{\$}，\$ 表示句子末尾；

2)对于新产生的次归约项目$t=[A\rightarrow X_1\cdots\bullet C,j]\in S_i$，按如下方式计算其展望符集：

a)若$[B\rightarrow X_1\cdots\bullet AX_m\cdots,k]\in S_j$，且 $X_m\in VN$，则将 X_m 的 First 集并入 t 的展望符集；若 $X_m\in VT$，则将 X_m 添加到 t 的展望符集；

b)若项目$s=[B\rightarrow X_1\cdots\bullet A,k]\in S_j$，则将 s 的展望符集并入 t 的展望符集。

上面第一种策略修改了 Predictor 操作，第二种和第三种策略需要修改 Completer 操作。改进后的操作如下。

1)Predictor：设$[A\rightarrow X_1\cdots\bullet C\cdots X_m,j]\in S_i$，$C\in VN$，当前读入的终结符为 x。对文法中每个形如 $C\rightarrow Y_1\cdots Y_n$ 的规则，若 Y_1 是终结符 x，或 $Y_1\in VN$ 且 Y_1 的 First 集中含有 x，则在状态 S_i 中增加新的形如$[C\rightarrow\bullet Y_1\cdots Y_n,i]$的项目；

2)Completer：设$[A\rightarrow X_1\cdots X_m\bullet,j]\in S_i$，当前读入的终结符为 x。若 A 的 Follow 集(策略二)或该项目左指针指向的次归约项目的展望符集(策略三)含有 x，且$[B\rightarrow X_1\cdots\bullet A\cdots X_m,k]\in S_j$，则在 S_i 中增加$[B\rightarrow X_1\cdots A\bullet\cdots X_m,k]$。

每种策略都可以单独使用。第一种策略可以结合第二种或者第三种策略，第二种和第三种策略不能同时使用。另外需要说明的是，LL 策略仅需要求 First 集，求 Follow 集时需要用到 First 集，这两个集合可以根据文法直接求得而与待分析的句子无关。展望符集的计算也需要用到 First 集，而且需要在分析句子的时候根据状态的变化逐步求出，不能事先求得。我们在实验中计算算法运行时间的时候，将其中各种集合的计算时间都包含在内。

5 实验结果及分析

在两个不同的中文短语文法上进行了实验。第一个文法来自中科院计算所，共有 376 条规则。其中的规则比较长，最长的规则右部有 7 个文法符号。在此感谢中科院计算所提供的文法。另一个文法是我们自己写的一个文法，其中共有 169 条规则，规则右部长度不超过 3 个文法符号。使用这两个文法在实际分析句子时，大多数句子都会产生多个分析结果。通常句子越长，分析结果越多，导致算法运行时间特别长。为了减少分析时间，没有在长的句子上进行实验，仅在词数为 2 个至 6 个的短句或短语上进行了实验，每个词数下的句子数不尽相同。所有算法都在相同的硬件和软件环境下运行，并且算法均采用 Perl 实现。表 1 列出了每个文法上不同词数下的句子个数。

表 1 实验句子数

句数	2 词	3 词	4 词	5 词	6 词
文法 1	127	48	36	10	5
文法 2	168	100	80	39	34

为了检验这几种预测机制对算法的影响，分别测试了采用 LL 策略(LLE)，LL＋F(LLFE)，LL＋L(LLLE)，Follow(FE)，L(LE)的 Earley 算法以及没有采用任何预测策略的 Earley 算法(E)。实验结果平均到每个句子。

图 1 和图 2 分别是两个文法集上不同词数下平均每个句子产生的项目数。由于项目占据了主要的存储空间，项目数可以反映出算法在空间上的效率。

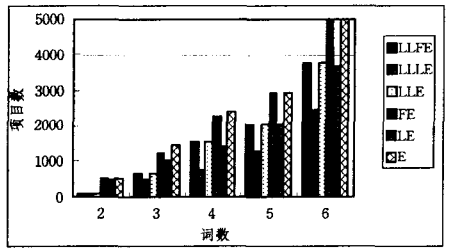

图 1　文法 1 产生的项目个数

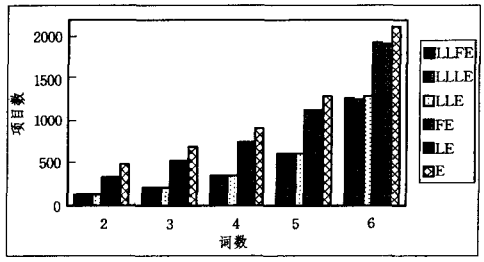

图 2　文法 2 产生的项目个数

表 2 和表 3 是两个文法集上不同词数下平均分析每个句子所花费的时间(单位是秒),行表示不同词数,列表示不同算法。运行时间反映出了算法的时间效率。

表 2　文法 1 花费的时间

秒	LLFE	LLLE	LLE	FE	LE	E
2	0.11	0.11	0.12	0.73	1.31	1.53
3	0.81	0.85	0.81	3.88	3.69	6.67
4	3.50	2.22	5.14	9.43	7.72	35.92
5	3.60	3.30	6.40	41.60	15.40	41.60
6	5.00	4.00	8.00	67.00	24.00	67.00

表 3　文法 2 花费的时间

秒	LLFE	LLLE	LLE	FE	LE	E
2	0.05	0.06	0.05	0.29	0.31	0.50
3	0.09	0.11	0.09	0.46	0.52	0.70
4	0.14	0.20	0.14	0.70	0.80	1.10
5	0.28	0.44	0.28	1.28	1.31	2.10
6	1.59	1.78	0.97	2.76	3.26	7.51

先从算法产生的项目数量上看,图 1 显示出 LFE 与没有使用预测策略的 Earley 算法(E)基本差不多,LLLE 结果最好,其它 3 种基本差不多,但都要明显优于 E;图 2 显示出 LLE,LLLE 和 LLFE 这 3 种算法效果都比较好且性能相近,FE 和 LE 效果不如前 3 种,但比 E 要稍好些。3 种策略中,LL 策略自顶向下限制无用起始项目的出现,使 L 策略和 F 策略可以自底向上限制对无用归约项目的规约。因为展望符集比 Follow 集精确,所以相同条件下使用 L 策略去掉的项目数肯定大于等于使用 F 策略去掉的项目数。一般情况下 LL 策略和 L 策略(或 F 策略)去掉的冗余项目不会完全一样,所以就"减少项目数"而言,LLLE 是不会比其它的算法差的。从实验结果来看,LLLE 产生的项目数平均为 E 的一半。

再从时间上看,表 2 的结果和图 1 的结果反映出了时间和项目数上的一致性,即产生的项目数量越少,算法花费的时间也越少;表 3 总体上也反映了这一情况,但当 LLE,LLLE 和 LLFE 在产生项目数量相差不多时,LLE 花费的时间最少,而 LLLE 花费时间最多。这说明对文法 2 来说,大部分项目已经由 LL 策略过滤,而 F 策略和 L 策略几乎不起作用,这时对归约项目进行限制等于花费了额外不必要的时间。从实验结果来看,表 2 中 LLLE 比 E 快 13 倍左右,而表 3 中 LLE 比 E 平均快 8 倍。

LL 策略是一种实现简单且效果较好的策略,虽然从理论上说即使不使用该策略,那些可以被其过滤的无用起始项目也会在 Scanner 时被过滤掉,但是后面状态进行 Completer 时会花费时间来扫描这些项目。如果这些无用起始项目数量众多,那么扫描花费的时间自然也就很多。自然语言的文法中非终结符数量不多,但其规则数目通常比较多,表明每个非终结符对应的规则特别多。而 LL 策略能够极大地限制某些规则对应的无用起始项目的出现。展望符集比 Follow 集更精确,使用 L 策略过滤掉的项目数不会少于使用 F 策略过滤掉的项目数。限制 Completer 只会限制当前的一个归约项目,而限制 Predictor 通常会限制一批由当前非终结符所导出的项目。这就解释了 LLE 在总体上要优于 LE 或 FE。

结束语　这些算法在时间和空间的效率表现上基本一致,但当某些算法产生的项目数相近时,算法在时空上的效率会有所差别。一方面说明项目数的减少意味着算法的查找时间会减少,另一方面说明对一些特定的文法或句子,不同算法产生的项目数相差无几时,策略本身所花费的时间开销会显现出来。实验结果显示,LL+L 策略在减少项目个数上是最好的,虽然单纯使用 LL 策略在时间效率上比较高,但综合考虑而言 LL+L 策略效果通常较好。

参 考 文 献

[1] 孟遥,李生,赵铁军,等. 基于统计的句法分析综述[J]. 计算机科学,2003,30(9):54-58

[2] 李珩,朱靖波,姚天顺. 基于 SVM 的中文组块分析[J]. 中文信息学报,2004,18(2):1-7

[3] 李素建,刘群,杨志峰. 基于最大熵模型的组块分析[J]. 计算机学报,2003,26(12):1722-1727

[4] Sun Guang-lu, Huang Chang-ning, Wang Xiao-long, et al. Chinese Chunking Based on Maximum Entropy Markov Models [J]. Computational Linguistics and Chinese Language Processing, 2006, 11(2): 115-136

[5] Aho A V, Johnson S C. LR Parsing [J]. Computing Survey, 1974, 6(2): 99-124

[6] Graham A L, Harrison M A, Ruzzo W L. An improved context-free recognizer[J]. ACM T PROGR LANG SYS, 1980: 415-462

[7] Tomita M. An Efficient Augmented-Context-free Parsing Algorithm[J]. Computational Linguistics, 1987, 13(1): 31-46

[8] Earley J. An efficient context-free parsing algorithm[J]. Comunications ACM, 1970, 13(2): 94-102

[9] Aho A V, Lam M S, Sathi R, et al. Compilers Principles Techniques and Tools(2nd)[M]. New York: Addison-Wesley, 1986: 191-300