2 MW 策略和 LC&MW 策略

在算法#ER[19]中,每次选择规约子句对极大项空间进行规约时,没有考虑子句之间的相互关系,机械地采用顺序选择策略,没有考虑启发式信息.针对同一子句集而言,选择不同的子句作为规约子句会对极大项空间的规模产生较大的影响,若极大项空间不能快速地进行规约,会造成空间消耗过大、递归调用次数多等问题,导致算法的时空效率降低.

例 1:给定子句集 $T=\{x_1\vee x_2,\neg x_1\vee x_2\vee x_3,x_1\vee\neg x_3\}$,变量集 $X=\{x_1,x_2,x_3\}$.假设选择 $x_1\vee x_2$ 作为规约子句,则 $T'=\{\neg x_1\vee x_2\vee x_3,x_1\vee\neg x_3\}$,#ER 算法的输入为子句集和变量集,此时的返回结果为 $M(T,X)=M(T',X)-M(T',X-\{x_1,x_2\})$;假设选择$\neg x_1\vee x_2\vee x_3$ 作为规约子句,则 $T'=\{\neg x_1\vee x_2\vee x_3,x_1\vee\neg x_3\}$,此时的返回结果为 $M(T,X)=M(T',X)-M(T',X-\{x_1,x_2,x_3\})$.

观察例 1 中两种情况的返回结果,$M(T',X-\{x_1,x_2,x_3\}$明显较$M(T',X-\{x_1,x_2\}$更容易进行计算,前者的极大项空间为 2^0,而后者的极大项空间为 2^1.选择$\neg x_1\vee x_2\vee x_3$ 作为规约子句所产生的单文字是 3 个,而选择 $x_1\vee x_2$ 作为规约子句产生的单文字是 2 个,子句和变量规模较大时,产生的单文字个数越多,越有利于单文字规则的使用,极大项空间的规模也能迅速降低.尤其当子句和变量规模较大时,选择规约子句的顺序会对算法的执行效率造成较大影响.因此,本文设计了两种启发式策略 MW 和 LC&MW,并基于这两种策略分别设计了算法 CER_MW 和 CER_LC&MW 来指导选择合适的规约子句,从而显著地减小极大项空间的规模,减少递归调用次数,提高求解效率.

在介绍 MW 策略和 LC&MW 策略之前,首先给出变量权值和子句权值的定义,以此来确定选择规约子句时的优先顺序.

定义 2(变量权值). 给定子句集 $T=\{C_1,...,C_n\}$和变量集 $X=\{x_1,...,x_m\}$,x 为变量集 X 中的任意一个变量,记变量 x_i 的权值为 $w(x_i)$,$w(x_i)$的大小为变量 x_i 在子句集 T 中出现的次数.

定义 3(子句权值). 给定子句集 $T=\{C_1,...,C_n\}$和变量集 $X=\{x_1,...,x_m\}$,C 为子句集 X 中的任意一个子句,记子句 C 的权值为 $w(C)$,$w(C)$的大小为子句 C 包含的所有变量的权值之和,即

$$w(C)=\sum_{i=1}^{k}w(x_i),\ k=|C| \tag{4}$$

定义 4(单文字规则). 若一个子句集中包含单文字 l,则可将 l 指派为真,对除 l 外的任意子句 C:

1) 若 C 包含 l,则从子句集中删除 C;
2) 若 C 包含$\neg l$,则将$\neg l$ 从 C 中删除;
3) 若 C 不包含 l 或$\neg l$,则不作处理.

2.1 MW策略

MW 策略的主要思想:在基于扩展规则的#SAT 求解算法中,每次选择规约子句来规约极大项空间时,优先选择子句权值最大的子句作为规约子句.若存在两个子句的权值相同,则随机地进行选择.

直观来讲,MW 策略具有明显优势,因为 MW 策略在选择规约子句时主要倾向于选择以下两种情况的子句:

1) 子句长度较大,包含的变量数多;
2) 子句不一定是最长的,但是所包含的变量在子句集中出现的次数较为频繁.

下面我们对这两种情况的子句进行分析.

- 针对情况 1),长度较大的子句所包含的变量数较多,因此计算 $M(T'\wedge\neg C_i)$时所获得的单文字个数就越多,使用单文字规则的次数可能会增加,进而减少了计算 $M(T'\wedge\neg C_i)$的过程中的递归调用次数,同时使得极大项空间的规模减小;
- 针对情况 2),规约子句所包含的变量在子句集中出现的次数较为频繁,这样在使用单文字规则时,可能会有更多的子句的长度减小,进而有利于子句集产生更多的单文字或短子句,也减小了极大项空间的规模,可能会使求解效率提高.

下面描述 MW 策略的实现框架.

算法 1. *Init*(*T*).

1) 初始化变量权值数组 *wv*、子句权值数组 *wc*
2) **for** (*C*∈*T*)
3) **for** (*x*∈*C*)
4) *wv*(*x*)+=1;
5) **for** (*C*∈*T*)
6) **for** (*x*∈*C*)
7) *wc*(*C*)+=*wv*(*x*).

算法 1 用于完成变量权值数组和子句权值数组的初始化工作,在整体算法执行前完成.数组 *wv* 用来保存每一个变量的权值大小,数组 *wc* 用来保存每一个子句的权值大小.算法 1 首先对数组 *wv* 和 *wc* 进行初始化为 0,随后依次遍历每个子句中的每一个变量,完成变量权值的计算过程,最后通过累加变量权值得到每一个子句的权值大小.

算法 2. *MW*(*T*).

输入:子句集 *T*.

输出:规约子句.

1) **for** (*C*∈*T*)
2) **if** (*wc*(*C*)>Max) **then**
3) Max=*wc*(*C*);
4) *C*′=*C*;
5) **return** *C*′.

算法 1 已经得到了每个子句的权值数组,算法 2 在算法 1 的基础上执行,遍历子句集中的每一个子句,返回具有最大子句权值的子句 *C*′.

下面对基于 MW 策略的#SAT 求解算法 CER_MW 进行描述.

算法 3. *CER_MW*(*T*,*X*).

输入:子句集 *T*、变量集 *X*.

输出:子句集 *T* 的模型数.

1) **if** (*T*=∅) **then**
2) return $2^{|X|}$;
3) **if** (∅∈*T*) **then**
4) **return** 0;
5) **if** (*T* 包含单元子句{*l*},*x* 是 *l* 的变量) **then**
6) **return** *CER_MW*({*C*−{¬*l*}|*C*∈*T*,*l*∉*C*},*X*−{*x*});
7) *C*′=*MW*(*T*),x_1,...,x_k 都是子句 *C*′中的变量;
8) *T*′=*T*;
9) **for** (*C*∈*T*′)
10) **if** (*C*′与 *C* 互补) **then**
11) 将 *C* 从 *T*′中删除;
12) **else if** (*C*′≠*C*) **then**
13) 删除子句 *C* 中与 *C*′相同的文字;
14) **return** *CER_MW*(*T*−{*C*′},*X*)−*CER_MW*(*T*′,*X*−{x_1,...,x_k}).

算法 3 的基本过程如下.

对子句集进行必要的判定操作:若子句集为空,则直接返回模型数为 $2^{|X|}$;若子句集包含空,则直接返回模型

数为 0;若子句集中存在单文字,使用单文字规则对子句集进行一定的规约,递归调用规约后的子句集作为返回值.利用 MW 策略选择出规约子句 C',复制一个子句集 T 的副本 T',利用规约子句将 T'的极大项空间进行规约.对 T'中的任一子句 C,若 C'与 C 互补,将 C 从 T'中删除;否则,删除子句 C 中与 C'相同的文字.该过程相当于对 $\neg C'$所产生的单文字全部使用了单文字规则.利用公式(3)的特性进行递归计算,得到子句集 T 的模型数.

2.2 LC&MW策略

我们考虑到在选择规约子句时,选择长度较大的子句作为规约子句可能更为重要,因为子句长度越长,相当于一次规约所减少的变量数越多.利用这一特征规约极大项空间更为稳定,而考虑规约子句长度的同时,也应尽可能的考虑变量出现的频率,一旦可以使用单文字规则,考虑这一因素有可能会使得子句集获得更多的单文字或短子句来减小极大项空间.因此,本文提出了 LC&MW 策略.

LC&MW 策略的主要思想:在基于扩展规则的#SAT 求解算法中,每次选择规约子句来规约极大项空间时,优先选择子句长度最长的子句作为规约子句.一般情况下,子句长度的取值范围较小,所选择的最长子句一般存在多个.若存在多个最长子句,在其中选择权值最大的子句作为规约子句.

下面描述 LC&MW 策略的实现框架.

算法 4. *LC&MW*(T).

输入:子句集 T.

输出:规约子句.

1) $\Sigma=\{C|C\in T,\neg\exists(C'\in T)L(C')>L(C)\}$;
2) **return** $MW(\Sigma)$.

算法 4 中,用 $L(C)$表示任一子句 C 的长度.对子句集 T 中的任一子句 C,若不存在 T 中的其他子句 C',使得 $L(C')>L(C)$,则将子句 C 加入集合Σ.此时,Σ为长度最大的一个或多个子句的集合.再将子句集Σ调用算法 2 来选择规约子句.通过该算法选择的规约子句在首先保证子句长度的同时,也尽可能地使得权值更大.

下面对基于 LC&MW 策略的#SAT 求解算法 CER_LC&MW 进行描述.

算法 5. *CER_LC&MW*(T,X).

输入:子句集 T、变量集 X.

输出:子句集 T 的模型数.

1) **if** ($T=\varnothing$) **then**
2) **return** $2^{|X|}$;
3) **if** ($\varnothing\in T$) **then**
4) **return** 0;
5) **if** (T 包含单元子句$\{l\}$,x 是 l 的变量) **then**
6) **return** $CER\_LC\&MW(\{C-\{\neg l\}|C\in T,l\notin C\},X-\{x\})$;
7) $C'=LC\&MW(T)$,$x_1,...,x_k$ 都是子句 C'中的变量;
8) $T'=T$;
9) **for** ($C\in T'$)
10) **if** (C'与 C 互补) **then**
11) 将 C 从 T'中删除;
12) **else if** ($C'\neq C$) **then**
13) 删除子句 C 中与 C'相同的文字;
14) **return** $CER\_LC\&MW(T-\{C'\},X)-CER\_LC\&MW(T',X-\{x_1,...,x_k\})$.

算法 5 的执行流程与算法 3 基本相同,仅在第 7)行选择规约子句时改变了启发式策略,利用 LC&MW 策略来选择规约子句.

3 实验部分

在本节中,将本文的 CER_MW 算法、CER_LC&MW 算法与目前最优的几个基于扩展规则的#SAT 求解算法进行了对比,主要包括#ER[19],RCER[20]和 CDCER[21].实验第 1 部分测试长度随机的 SAT 测试用例,第 2 部分测试长度固定的 SAT 测试用例.本文的实验平台如下:Windows 8.1 操作系统,CPU Intel(R) Core(TM) i7-4790 3.60GHz,RAM 8GB.

3.1 随机子句长度的SAT用例测试

本文用随机产生器生成了子句长度不固定的测试用例,随机产生器的结果为包含 3 个参数⟨*m*,*n*,*e*⟩的子句集,其中,*m* 为变量个数,*n* 为子句个数,*e* 为每个子句的最大长度.*m* 个变量对应 2×*m* 个文字,为了保证所生成测试集的质量,随机产生器在生成每个文字时,2×*m* 个文字出现的概率是相同的,同时控制生成过程,避免生成重言式.为增加子句集的求解难度,我们将子句的最小长度设定为 3.目前,基于扩展规则的求解方法都不能解决规模大的问题,表 1~表 3 中分别给出了⟨20,*n*,10⟩,⟨30,*n*,10⟩和⟨40,*n*,10⟩这 3 种变量数固定、子句数不同的随机测试用例的实验结果.实验结果为 10 次实验的平均值,time 表示运行时间(单位为 s),"–"表示在 1 000s 内无法对问题进行求解.

在表 1、表 2 中,CER_MW 和 CER_LC&MW 求解速度都快于 RCER,CDCER 和#ER,其中,CER_MW 比 RCER 快 4 倍~64.7 倍,比 CDCER 快 5.5 倍~93.5 倍,比#ER 快 1.4 倍~4.3 倍;CER_LC&MW 比 RCER 快 4 倍~69 倍,比 CDCER 快 5.5 倍~100.2 倍,比#ER 快 1.4 倍~4.5 倍.表 1 中,二者求解速度相当,其原因可能是因为子句集本身的规模较小,求解时间相差不大,也可能是因为每次使用 MW 策略所选择的最大权值子句同时也是最长子句;表 2 中,CER_LC&MW 比 CER_MW 求解速度更快,说明在选择规约子句的过程中,子句长度这一因素发挥了更重要的作用.在表 3 中,RCER 和 CDCER 的求解已经非常吃力,大部分问题已经无法求解,因此我们仅给出了#ER,CER_MW 和 CER_LC&MW 的对比结果.在表 3 的 9 个测试用例中,#ER 仅能求解前 5 个测试用例,而 CER_MW 和 CER_LC&MW 可以求解所有测试用例,在求解能力上较 RCER,CDCER 和#ER 有较大的提高.

Table 1 Experimental results on random instances ⟨20,*n*,10⟩

表 1 随机用例⟨20,*n*,10⟩的实验结果

Instances	Complementary factor	Algorithm				
		RCER(s)	CDCER(s)	#ER(s)	CER_MW(s)	CER_LC&MW(s)
⟨20,30,10⟩	0.756 322	0.001	0.001	0.001	**<0.001**	**<0.001**
⟨20,40,10⟩	0.741 026	0.002	0.002	0.001	**<0.001**	**<0.001**
⟨20,50,10⟩	0.693 878	0.004	0.006	0.004	**0.001**	**0.001**
⟨20,60,10⟩	0.676 271	0.010	0.011	0.004	**0.002**	**0.002**
⟨20,70,10⟩	0.685 300	0.016	0.019	0.005	0.003	**0.002**
⟨20,80,10⟩	0.678 797	0.033	0.036	0.007	**0.004**	0.005
⟨20,90,10⟩	0.679 151	0.062	0.070	0.010	**0.005**	**0.005**
⟨20,100,10⟩	0.657 980	0.148	0.190	0.016	**0.008**	**0.008**

Table 2 Experimental results on random instances ⟨30,*n*,10⟩

表 2 随机用例⟨30,*n*,10⟩的实验结果

Instances	Complementary factor	Algorithm				
		RCER(s)	CDCER(s)	#ER(s)	CER_MW(s)	CER_LC&MW(s)
⟨30,60,10⟩	0.499 435	0.065	0.097	0.021	0.015	**0.013**
⟨30,70,10⟩	0.518 427	0.172	0.214	0.045	0.026	**0.024**
⟨30,80,10⟩	0.514 557	0.380	0.479	0.089	0.048	**0.040**
⟨30,90,10⟩	0.510 362	0.928	1.285	0.217	0.100	**0.096**
⟨30,100,10⟩	0.529 495	1.675	1.997	0.345	0.133	**0.119**
⟨30,110,10⟩	0.509 758	3.765	5.940	0.683	0.253	**0.234**
⟨30,120,10⟩	0.527 451	6.510	9.013	0.985	0.327	**0.299**
⟨30,130,10⟩	0.515 206	16.883	29.040	1.890	0.560	**0.510**
⟨30,140,10⟩	0.518 499	36.443	60.088	2.897	0.843	**0.757**
⟨30,150,10⟩	0.518 479	65.097	94.448	4.280	1.006	**0.943**

Table 3 Experimental results on random instances ⟨40,*n*,10⟩

表 3 随机用例⟨40,*n*,10⟩的实验结果

Instances	Complementary factor	Algorithm		
		#ER(s)	CER_MW(s)	CER_LC&MW(s)
⟨40,120,10⟩	0.323 810	169.462	**1.740**	2.061
⟨40,130,10⟩	0.290 865	175.103	14.148	**9.489**
⟨40,140,10⟩	0.271 285	116.810	3.812	**2.808**
⟨40,150,10⟩	0.320 000	581.076	**23.361**	27.629
⟨40,160,10⟩	0.289 007	583.150	10.966	**10.131**
⟨40,170,10⟩	0.251 082	–	**17.471**	17.931
⟨40,180,10⟩	0.320 608	–	57.742	**48.335**
⟨40,190,10⟩	0.292 509	–	**76.723**	125.369
⟨40,200,10⟩	0.324 724	–	**96.222**	96.443

#ER,CER_MW 和 CER_LC&MW 都是采用递归调用的方式来求解问题,MW 和 LC&MW 策略能找到合适的规约子句,使得极大项空间的规模快速减小,进而使得递归调用次数减少,提高了算法求解效率.下面我们给出#ER,CER_MW 和 CER_LC&MW 的递归调用次数对比图,图 1~图 3 中分别给出了在⟨20,*n*,10⟩,⟨30,*n*,10⟩和⟨40,*n*,10⟩这 3 种变量数固定、子句数不同的随机测试用例中,#ER,CER_MW 和 CER_LC&MW 的递归调用次数.

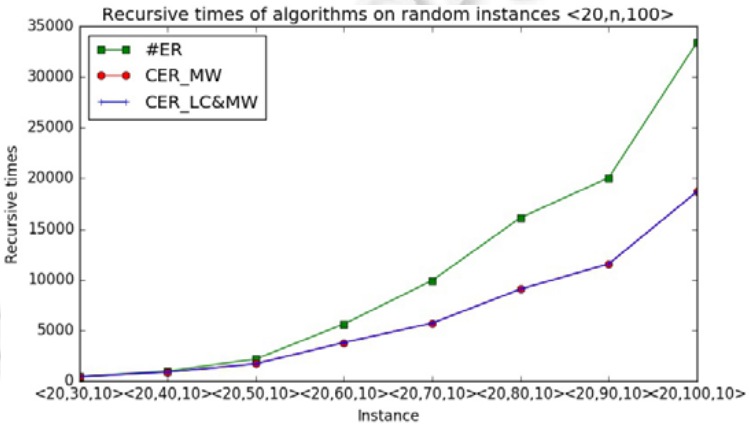

Fig.1 Recursive times of algorithms on random instances ⟨20,*n*,10⟩

图 1 随机用例⟨20,*n*,10⟩的算法递归调用次数

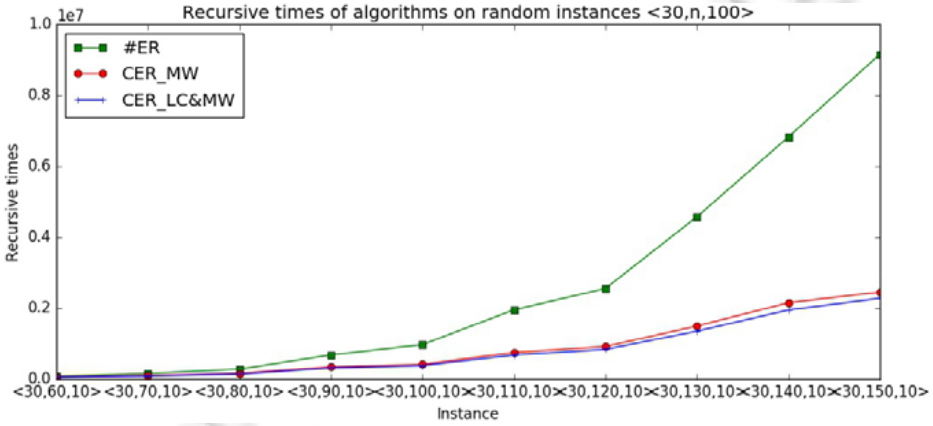

Fig.2 Recursive times of algorithms on random instances ⟨30,*n*,10⟩

图 2 随机用例⟨30,*n*,10⟩的算法递归调用次数

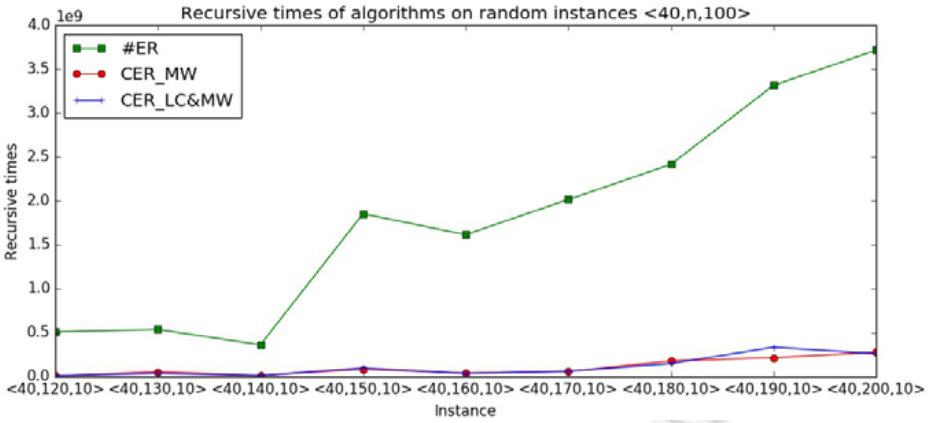

Fig.3 Recursive times of algorithms on random instances ⟨40,*n*,10⟩

图 3 随机用例⟨40,*n*,10⟩的算法递归调用次数

从图 1~图 3 可以看出,CER_MW 和 CER_LC&MW 的递归调用次数要远小于#ER 的递归调用次数.这是由于 MW 和 LC&MW 策略在选择规约子句时能选择到合适的规约子句,使算法充分发挥扩展规则和单文字规则的优势,迅速规约极大项空间,进而加速求解过程.

3.2 长度固定的SAT用例测试

本节针对长度固定的测试用例进行测试,其生成方法与上节相同,但限定子句的最大长度与最小长度都为 *n*,表 4、表 5 中分别给出了⟨30,100,*n*⟩和⟨40,200,*n*⟩这两种变量数固定、子句长度不同的测试用例的实验结果,表中参数与上节保持一致.

Table 4 Experimental results on random instances ⟨30,100,*n*⟩

表 4 随机用例⟨30,100,*n*⟩的实验结果

Instances	Complementary factor	Algorithm			
		RCER(s)	CDCER(s)	#ER(s)	CER_MW(s)
⟨30,100,10⟩	0.964 444	0.007	0.012	**0.001**	**0.001**
⟨30,100,9⟩	0.926 263	0.007	0.013	**0.002**	**0.002**
⟨30,100,8⟩	0.870 303	0.010	0.014	**0.002**	**0.002**
⟨30,100,7⟩	0.800 202	0.015	0.018	0.006	**0.005**
⟨30,100,6⟩	0.660 606	0.052	0.062	0.025	**0.021**
⟨30,100,5⟩	0.476 970	0.973	1.404	0.289	**0.261**
⟨30,100,4⟩	0.304 444	73.364	146.110	5.614	**4.137**
⟨30,100,3⟩	0.140 404	–	–	34.339	**30.564**

Table 5 Experimental results on random instances ⟨40,200,*n*⟩

表 5 随机用例⟨40,200,*n*⟩的实验结果

Instances	Complementary factor	Algorithm			
		RCER(s)	CDCER(s)	#ER(s)	CER_MW(s)
⟨40,200,10⟩	0.918 693	0.039	0.063	0.015	**0.014**
⟨40,200,9⟩	0.864 724	0.062	0.079	0.035	**0.032**
⟨40,200,8⟩	0.792 462	0.158	0.155	0.091	**0.083**
⟨40,200,7⟩	0.666 281	1.662	1.653	0.869	**0.746**
⟨40,200,6⟩	0.536 583	24.129	28.067	9.637	**7.986**
⟨40,200,5⟩	0.377 940	–	–	458.035	**310.905**
⟨40,200,4⟩	0.233 015	–	–	–	–
⟨40,200,3⟩	0.111 307	–	–	–	–

在本节的测试用例中,LC&MW 策略不再适用,因为测试用例中的子句长度都相等,LC&MW 与 MW 策略选择的规约子句一定相同.因此,我们仅给出了 RCER,CDCER,#ER 和 CER_MW 的对比结果.

在表 4、表 5 中,对于固定长度的测试用例,其子句长度对该子句集的互补因子有着一定的影响,子句长度相对越大,子句两两之间互补的可能性就越大.从表中可以看出,子句长度与子句集的互补因子是正相关的,扩展规则推理方法擅长求解互补因子较高的测试用例,因此当互补因子较高时,4 种算法的求解速度较快;而随着互补因子的减小,受扩展规则方法本身的影响,算法的求解速度会变慢.从表中的数据可以看出,无论在互补因子较高的测试用例,还是在互补因子较低的测试用例中,CER_MW 的求解效率都高于 RCER、CDCER 和#ER.

4 总结与展望

选择规约子句的顺序对极大项空间的大小有着较大的影响,为此,本文提出了两种启发式策略:MW 策略和 LC&MW 策略.MW 每次选择具有最大权值的子句作为规约子句;LC&MW 每次优先选择最长子句作为规约子句来减小极大项空间,若最长子句存在多个,则在多个最长子句中选择具有最大权值的子句作为规约子句.本文根据 MW 策略和 LC&MW 策略分别设计了#SAT 求解算法 CER_MW 和 CER_LC&MW,实验结果表明,CER_MW 和 CER_LC&MW 相对于先前的#SAT 求解算法在求解效率和求解能力上都有显著的提高.在求解效率方面,CER_MW 和 CER_LC&MW 的求解速度是其他算法的 1.4 倍~100 倍;在求解能力方面,CER_MW 和 CER_LC&MW 在限定时间内可解的测试用例更多.

目前,虽然基于扩展规则的求解算法在互补因子较高的测试用例上优势明显,但受扩展规则本身的特性所限,其求解能力并不高,对于变量和子句规模较大的测试用例求解较为困难.未来将进一步研究如何通过启发式策略或局部搜索的思想来提升#SAT 求解算法的求解能力,使得基于扩展规则的方法适用于大规模的测试用例.

References:

[1] Biere A, Heule M, Maaren HV, Walsh T. Handbook of Satisfiability. IOS Press, 2009. 195−203.

[2] Cook SA. The complexity of theorem-proving procedures. In: Proc. of the 3rd Annual ACM Symp. on Theory of Computing. New York: ACM Press, 1971. 151−158.

[3] Cai S, Luo C, Lin J, Su K. New local search methods for partial MaxSAT. Artificial Intelligence, 2016,240:1−18.

[4] Ansótegui C, Gabàs J, Malitsky Y, Sellmann M. MaxSAT by improved instance-specific algorithm configuration. Artificial Intelligence, 2016,235:26−39.

[5] Frohlich A, Biere A, Wintersteiger CM, Hamadi AY. Stochastic local search for satisfiability modulo theories. In: Proc. of the 29th AAAI Conf. on Artificial Intelligence. Menlo Park: AAAI, 2015. 1136−1143.

[6] Zhang HT. An experiment with satisfiability modulo SAT. Journal of Automated Reasoning, 2016,56(2):143−154.

[7] Chakraborty S, Meel KS, Mistry R, Vardi MY. Approximate probabilistic inference via word-level counting. In: Proc. of the 30th AAAI Conf. on Artificial Intelligence. Menlo Park: AAAI, 2016. 3218−3224.

[8] Lagniez JM, Marquis P. On preprocessing techniques and their impact on propositional model counting. Journal of Automated Reasoning, 2017,58(4):413−481.

[9] Birnbaum E, Lozinskii EL. The good old Davis-Putnam procedure helps counting models. Journal of Artificial Intelligence Research, 1999,10(1):457−477.

[10] Bayardo RJ, Pehoushek JD. Counting models using connected components. In: Proc. of the 17th National Conf. on Artificial Intelligence and 20th Conf. on Innovative Applications of Artificial Intelligence. Menlo Park: AAAI, 2000. 157−162.

[11] Sang T, Beame P, Kautz H. Heuristics for fast exact model counting. In: Proc. of the Int'l Conf. on Theory and Applications of Satisfiability Testing. Berlin: Springer-Verlag, 2005. 226−240.

[12] Sang T, Bacchus F, Beame P, Kautz H, Pitassi T. Combining component caching and clause learning for effective model counting. In: Proc. of the Int'l Conf. on Theory and Applications of Satisfiability Testing. Berlin: Springer-Verlag, 2004. 20−28.

[13] Zhang L, Madigan CF, Moskewicz MH, Mailk S. Efficient conflict driven learning in a Boolean satisfiability solver. In: Proc. of the IEEE/ACM Int'l Conf. on Computer-Aided Design. New York: IEEE, 2001. 279−285.

[14] Thurley M. sharpSAT: Counting models with advanced component caching and implicit BCP. In: Proc. of the Int'l Conf. on Theory and Applications of Satisfiability Testing. Berlin: Springer-Verlag, 2006. 424−429.

[15] Davies J, Bacchus F. Using more reasoning to improve #SAT solving. In: Proc. of the 22nd AAAI Conf. on Artificial Intelligence. Vancouver: AAAI, 2007. 185−190.

[16] Lin H, Sun JG, Zhang YM. Theorem proving based on the extension rule. Journal of Automated Reasoning, 2003,31(1):11−21.

[17] Lin H, Sun JG. Knowledge compilation using the extension rule. Journal of Automated Reasoning, 2004,32(2):93−102.

[18] Yin MH, Lin H, Sun JG. Solving #SAT using extension rule. Ruan Jian Xue Bao/Journal of Software, 2009,20(7):1714−1725 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3320.htm [doi: 10.3724/SP.J.1001.2009.03320]

[19] Lai Y, Ouyang DT, Cai DB, Lü S. Model counting and planning using extension rule. Journal of Computer Research and Development, 2009,46(3):459−469 (in Chinese with English abstract).

[20] Jia FY, Ouyang DT, Zhang LM, Liu SG. Reconstructive algorithm based on extension rule for solving #SAT incrementally. Ruan Jian Xue Bao/Journal of Software, 2015,26(12):3117−3129 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4827.htm [doi: 10.13328/j.cnki.jos.004827]

[21] Ouyang DT, Jia FY, Liu SG, Zhang LM. An algorithm based on extension rule for solving #SAT using complementary degree. Journal of Computer Research and Development, 2016,53(7):1596−1604 (in Chinese with English abstract).

[22] Wang J, Yin M, Wu J. Two approximate algorithms for model counting. Theoretical Computer Science, 2017,657:28−37.

[23] Wei W, Erenrich J, Selman B. Towards efficient sampling: Exploiting random walk strategies. In: Proc. of the 19th National Conf. on Artificial Intelligence. AAAI, 2004. 670−676.

附中文参考文献:

[18] 殷明浩,林海,孙吉贵.一种基于扩展规则的#SAT 求解系统.软件学报,2009,20(7):1714−1725. http://www.jos.org.cn/1000-9825/3320.htm [doi: 10.3724/SP.J.1001.2009.03320]

[19] 赖永,欧阳丹彤,蔡敦波,吕帅.基于扩展规则的模型计数与智能规划方法.计算机研究与发展,2009,46(3):459−469.

[20] 贾凤雨,欧阳丹彤,张立明,刘思光.结合扩展规则重构的#SAT 问题增量求解方法.软件学报,2015,26(12):3117−3129. http://www.jos.org.cn/1000-9825/4827.htm [doi: 10.13328/j.cnki.jos.004827]

[21] 欧阳丹彤,贾凤雨,刘思光,张立明.结合互补度的基于扩展规则#SAT 问题求解方法.计算机研究与发展,2016,53(7):1596−1604.

王强(1994－),男,山西临汾人,硕士,主要研究领域为人工智能,智能规划与自动推理.

吕帅(1981－),男,博士,副教授,CCF 高级会员,主要研究领域为人工智能,智能规划与自动推理.

刘磊(1960－),男,教授,博士生导师,CCF 专业会员,主要研究领域为软件理论与技术.