Table 3 Experimental results on random instances ⟨30,*n*,10⟩

表 **3** 随机实例⟨30,*n*,10⟩的实验结果

Instances	IKCHER		DKCHER		UKCHER		C2E		KCER	
	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)
⟨30,50,10⟩	29 675	19.385	33 778	21.246	102 847	1.372	45 363	0.693	80 336	14.699
⟨30,60,10⟩	12 114	9.696	18 752	10.500	142 109	2.149	46 133	0.671	150 186	13.362
⟨30,70,10⟩	6 096	3.553	7 082	3.220	123 242	2.840	41 903	0.562	123 717	17.251
⟨30,80,10⟩	6 260	3.553	4 853	2.301	90 857	3.228	50 169	0.690	136 676	19.738
⟨30,90,10⟩	2 032	1.115	2 231	1.088	91 499	3.361	55 943	0.684	169 235	19.233
⟨30,100,10⟩	1 381	1.038	1 052	0.609	80 883	3.573	52 454	0.680	170 060	22.176
⟨30,110,10⟩	750	0.861	1 053	0.748	55 830	2.978	58 068	0.720	101 370	19.556
⟨30,120,10⟩	335	0.810	204	0.850	89 962	3.742	48 422	0.611	130 408	16.250

基于表 1~表 3 能够观察出以下现象:(1) 当变量数固定时,IKCHER 算法和 DKCHER 算法的编译结果的规模和编译所需时间会随着子句数的增加而减少;(2) 当子句集规模较小时,UKCHER 算法、C2E 算法和 KCER 算法的编译结果的规模和编译所需时间随着子句数的增加而增加;(3) 当子句集规模大于某一临界值时,UKCHER 算法、C2E 算法和 KCER 算法的编译结果的规模会维持不变,甚至减少;(4) 由于样例随机生成,因此可能会存在少量的波动.例如表 2 中,当 n<90 时,UKCHER 算法、C2E 算法和 KCER 算法的编译结果的规模和编译所需时间随着子句数的增加而逐渐增加(现象(2));当 n≥90 时,UKCHER 算法、C2E 算法和 KCER 算法的编译所需时间仍然随着子句数的增加而逐渐增加,其编译结果的规模却逐渐稳定下来(现象(3));当 n≥90 时,UKCHER 算法、C2E 算法和 KCER 算法编译结果的规模均上下无规律波动(现象(4)).类似的现象同样出现在表 1 和表 3 中.

从编译规模来看,表 1~表 3 中,45.8%的实例下,IKCHER 算法的编译质量是最优的;41.7%的实例下,DKCHER 算法的编译质量是最优的;12.5%的实例下,C2E 的编译质量是最优的.由于 DKCHER 算法和 IKCHER 算法的编译质量的差值较小,因此可以认定 IKCHER 算法和 DKCHER 算法的编译质量相当,二者在大多数情况下是最优的.表 1 中,当 n<90 时,C2E 算法的编译规模会随着子句数的增加而增加;当 n≥90 时,C2E 算法的编译规模会稳定在 2 100 左右.同样的现象出现在 KCER 算法中,不过整体来看,KCER 算法的编译规模大于 C2E 算法的编译规模.表 1 中,当 n<50 时,IKCHER 算法和 DKCHER 算法的编译规模大于 C2E 算法的编译规模;然而随着子句数的增加,当 n≥50 时,IKCHER 算法和 DKCHER 算法的编译规模远远小于 C2E 和 KCER 的编译规模.甚至当 n=100 时,C2E 算法的编译规模是 IKCHER 算法的 107 倍,是 DKCHER 算法编译规模的 128 倍;KCER 算法编译规模是 IKCHER 算法的 193 倍,是 DKCHER 算法编译规模的 254 倍.由此可知,求交知识编译算法 IKCHER 对于变量数固定、子句数规模较大的子句集能够取得很好的编译效果.UKCHER 算法的编译规模始终大于 C2E 算法的编译规模,而和 KCER 算法的编译规模接近,但是小于后者.从表 2 和表 3 可以得到和表 1 相同的结论.

从编译效率来看,表 1 中,当 n≤50 时,UKCHER 算法的效率是最高的,C2E 算法居于第 3 位,而 IKCHER 算法和 DKCHER 算法的效率最差.这是由于 IKCHER 算法和 DKCHER 算法的扩展过程中都需要保存较多的中间结果,其中很多无效的中间结果会参与到算法扩展过程中,影响了整体执行效率.当 n>50 时,DKCHER 算法的编译效率最高,IKCHER 算法的编译效率排在第 2 位,但是与 DKCHER 算法的编译效率相差不大;同时,随着子句数的增加,UKCHER 算法和 KCER 算法所需编译时间也随之大幅度增加,二者的编译效率最终均差于 C2E 算法.表 3 中,IKCHER 算法的编译效率始终差于 C2E 算法的编译效率.整体来看,对于随机长度的 SAT 问题,IKCHER 算法适用于变量数较小、子句数较大的子句集.

3.2 对于标准**3-SAT**子句集上**IKCHER**算法的测试

为了更全面地展示本文提出的知识编译算法的特性,本文还随机生成了变量数固定、子句数改变的标准 3-SAT 子句集,表 4~表 6 分别给出了⟨20,*n*⟩,⟨25,*n*⟩和⟨30,*n*⟩这 3 种不同 3-SAT 子句集实例的样例进行测试,测试结

果为 50 次实验的平均值.

Table 4 Experimental results on random 3-SAT instances ⟨20,*n*⟩

表 4 随机 3-SAT 实例⟨20,*n*⟩的实验结果

Instances	IKCHER		DKCHER		UKCHER		C2E		KCER	
	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)
⟨20,46⟩	1 493	0.049	1 601	0.047	6 199	0.105	2 972	0.093	6 199	0.114
⟨20,56⟩	745	0.028	857	0.032	6 726	0.153	3 101	0.091	6 726	0.160
⟨20,66⟩	329	0.026	299	0.029	6 945	0.205	3 250	0.087	6 945	0.203
⟨20,76⟩	132	0.025	129	0.028	6 108	0.251	3 241	0.092	6 118	0.245
⟨20,86⟩	54	0.025	63	0.026	6 008	0.261	3 010	0.101	5 859	0.282
⟨20,96⟩	17	0.027	22	0.026	6 556	0.337	3 457	0.093	6 812	0.351
⟨20,106⟩	6	0.028	5	0.025	6 512	0.337	3 303	0.097	6 943	0.376
⟨20,116⟩	3	0.025	3	0.024	6 961	0.343	3 075	0.091	6 709	0.414

Table 5 Experimental results on random 3-SAT instances ⟨25,*n*⟩

表 5 随机 3-SAT 实例⟨25,*n*⟩的实验结果

Instances	IKCHER		DKCHER		UKCHER		C2E		KCER	
	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)
⟨25,57⟩	8 012	0.846	8 031	1.009	41 680	0.859	17 861	0.230	41 680	1.612
⟨25,67⟩	4 386	0.453	4 237	0.473	43 660	1.182	21 320	0.268	43 660	1.940
⟨25,77⟩	1 967	0.234	1 849	0.274	42 422	1.537	18 908	0.244	42 422	2.160
⟨25,87⟩	613	0.175	645	0.209	43 344	1.933	17 566	0.249	43 344	2.688
⟨25,97⟩	258	0.198	347	0.190	44 898	2.394	18 826	0.259	45 470	3.101
⟨25,107⟩	92	.0189	145	0.206	43 106	2.711	19 763	0.251	43 360	3.285
⟨25,117⟩	36	0.212	34	0.216	42 009	2.568	21 979	0.273	40 233	3.745
⟨25,127⟩	8	0.180	16	0.218	44 833	2.801	18 243	0.243	43 504	3.933

Table 6 Experimental results on random 3-SAT instances ⟨30,*n*⟩

表 6 随机 3-SAT 实例⟨30,*n*⟩的实验结果

Instances	IKCHER		DKCHER		UKCHER		C2E		KCER	
	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)
⟨30,69⟩	26 436	6.774	20 548	5.854	230 468	6.160	112 513	1.272	235 802	28.009
⟨30,79⟩	12 453	3.014	13 999	3.401	219 648	6.114	107 060	1.145	–	–
⟨30,89⟩	6 697	2.342	7 721	2.074	–	–	103 951	1.181	–	–
⟨30,99⟩	2 949	1.535	4 116	1.947	–	–	112 619	1.198	–	–
⟨30,109⟩	1 152	1.536	1 671	1.623	–	–	109 678	1.129	–	–
⟨30,119⟩	425	1.796	345	1.501	–	–	122 524	1.303	–	–
⟨30,129⟩	163	1.835	107	1.664	–	–	115 610	1.244	–	–
⟨30,139⟩	53	1.646	58	1.567	–	–	106 545	1.146	–	–

基于表 4~表 6 能够得出:(1) 对于 3-SAT 子句集,当变量数固定、子句数改变时,UKCHER、C2E 和 KCER 这 3 种知识编译算法的编译规模均随着变量数的增加而增加;(2) IKCHER 算法和 DKCHER 算法的编译规模会随着子句数的增加而减少,而二者的编译质量相差不大.然而,其中 62.5%的实例下,IKCHER 算法的编译质量是最优的;37.5%的实例下,DKCHER 算法的编译质量是最优的,因此从编译质量角度考虑,在编译 3-SAT 子句集时,应尽可能地选择 IKCHER 算法.从编译效率上来看,DKCHER 算法的编译效率在大多数情况下优于 IKCHER 算法的编译效率,但是二者相差不大.

表 6 中,由于变量数过大,UKCHER 算法和 KCER 算法会经常内存溢出,因此,对于表 6 中的很多实例,UKCHER 和 KCER 是无法解决的.显然,二者不适用于变量数较多的 3-SAT 问题.

为了验证 IKCHER 算法并行化的可行性,本文进一步研究了如何实现 IKCHER 算法的并行化.

4 求交知识编译算法的并行化

本文将求解多个 EPCCL 理论所能扩展出的极大项集简称为 EPCCL 理论求交,EPCCL 理论求交对于实现 IKCHER 算法的并行化是一个关键问题.

4.1 EPCCL理论的求交操作

事实上,IKCHER 算法中用到了两个 EPCCL 理论的求交操作,算法执行过程中需要不断地对 F_2 和 S_i 进行求交操作.定理 2 给出了两个 EPCCL 理论求交的具体过程,任意两个 EPCCL 理论 $E_1=\{R_1,\ldots,R_p\}$ 和 $E_2=\{T_1,\ldots,T_q\}$ 求交之后,所得结果为一个 EPCCL 理论:$S=J_M(E_1)\cap J_M(E_2)\equiv\{I_1,\ldots,I_{p\times q}\}$,其中,任意 $I_i(1\leq i\leq p\times q)$为 $J_M(R_{(i-1)/q+1})\cap J_M(T_{(i-1) \bmod q+1})$的计算结果.显然,上述过程可以并行分解,由于任意 I_i 和 I_j 的求解过程之间互不影响,因此可以并行求解$\{I_1,\ldots,I_{p\times q}\}$.我们可以将 E_1 拆分成 k 个子集 $E_1^1,\ldots,E_1^k$,然后并行求得 $\bigcup_{1\leq i\leq k}(E_1^i\cap E_2)$,所得结果即为 E_1 和 E_2 所能扩展出的极大项集的交集.

由于两个 EPCCL 理论求交之后仍然是一个 EPCCL 理论,因此可以采用增量式的思想并行求得任意多个 EPCCL 理论所能扩展出的极大项集的交集.根据上述过程,我们设计了多个 EPCCL 理论的并行求交算法,并称其为 PIAE.该算法描述如下.

算法 **2**. PIAE.

1. 输入:s 个 EPCCL 理论$\{E_1,\ldots,E_s\}$
2. $E=E_1$, $h=2$
3. **While** $h\leq s$
4. **Divide** E_h **to** $\{E_h^1,\ldots,E_h^k\}$
5. **Parallel**
6. **Foreach** E_h^v $(1\leq v\leq k)$
7. $d=1$
8. **While** $d\leq|E_h^v|$
9. $T_v=ICE(E,C_d)$ $(C_d\in E_h^v)$
10. d++
11. **End parallel**
12. $E=T_1\cup\ldots\cup T_k$
13. h++
14. **Return** E

Sub-procedure ICE (CNF $F=\{C_1,\ldots,C_n\}$, clause C)

1. 令 $F_1=\varnothing$, $i=j=1$
2. **While** $i\leq|F|$
3. **If** C 与 C_i 互补 **Then skip**
4. **Else if** $C\vDash C_i$ **Then** $F_1=F_1\cup\{C_i\}$
5. **Else if** $C_i\vDash C$ **Then** $F_1=F_1\cup\{C\}$
6. **Else** $F_1=F_1\cup\{C_i\vee(C-C_i)\}$
7. j++
8. $j=1$
9. i++

10. **Return** F_1

算法 2 实现了 k 个 EPCCL 理论进行的并行求交操作,并将结果保存为 EPCCL 理论.该算法利用第 3 行的 while 循环实现 EPCCL 理论的增量式求交操作;在第 4 行,将已合并的 EPCCL 理论分为 k 份,然后和下一个未操作的 EPCCL 理论进行并行求交操作;第 5 行~第 11 行是算法的并行操作部分;第 12 行将求交结果合并.该算法在第 9 行调用了子程序 ICE.ICE 的功能是求解一个子句与一个子句集所能扩展出的极大项集的交集,若输入子句集是一个 EPCCL 理论,则输出的子句集为 EPCCL 理论,否则为普通子句集.

定理 **6**. 给定一个 EPCCL 理论 $E=\{R_1,\ldots,R_p\}$和一个子句集 $F=\{T_1,\ldots,T_q\}$,则 E_1 和 F 在 M 上所能扩展出的极大项集的差集 $S=J_M(E)-J_M(F)\equiv\bigcup_{1\le i\le p}(J_M(R_i)\cap J_M(\neg T_1)\cap\ldots\cap J_M(\neg T_q))$,且 S 为一个 EPCCL 理论.

证明:根据定理 1,$J_M(\neg T_1)\cap\ldots\cap J_M(\neg T_q)$能够表示 F 所不能扩展出的极大项集,

则任意 $J_M(R_i)\cap J_M(\neg T_1)\cap\ldots\cap J_M(\neg T_q)$能够表示 R_i 与 F 在 M 上所能扩展出的极大项集的差集,

因此, $S=J_M(E)-J_M(F)\equiv\bigcup_{1\le i\le p}(J_M(R_i)\cap J_M(\neg T_1)\cap\ldots\cap J_M(\neg T_q))$ 成立.

又由于 $J_M(R_i)\cap J_M(\neg T_1)\cap\ldots\cap J_M(\neg T_q)\subseteq J_M(R_i)$且 $E=\{R_1,\ldots,R_p\}$是一个 EPCCL 理论,

因此,S 是一个 EPCCL 理论. □

根据定理 6,PIAE 算法中,若能知道输入 EPCCL 理论对应与其所不能扩展出的极大项集等价的普通子句集,则可以直接对普通子句集操作.通常,当普通子句集的子句数与变量数的比值较小时,求解其所不能扩展出的极大项集所得 EPCCL 理论的规模要远远大于输入子句集,因此,直接对普通子句集操作能够加快合并过程.基于此,我们设计了新的并行 EPCCL 理论求并算法,称其为 imp-PIAE,描述如下.

算法 **3**. imp-PIAE.

1. 输入:s 个 EPCCL 理论$\{E_1,\ldots,E_s\}$,$F_1,\ldots,F_s$ 是分别与 $E_1,\ldots,E_s$ 所不能扩展出的极大项集等价的普通子句集
2. $E=E_1$, $h=2$
3. **While** $h\le s$
4. **Divide** E **to** $\{E_h^1,\ldots,E_h^k\}$
5. **Parallel**
6. **Foreach** E_h^k $(1\le v\le k)$
7. $d=1$
8. **While** $d\le|E_h^k|$
9. $T_v=CIF(F_h,C_d)$ $(C_d\in E_h^v)$
10. d++
11. **End parallel**
12. $E=T_1\cup\ldots\cup T_k$
13. h++
14. **Return** E

Sub-procedure CIF (CNF $F=\{C_1,\ldots,C_n\}$, clause C)

1. 令 $F_1=\{C\}$, $F_2=\varnothing$, $i=j=k=1$
2. **While** $i\le|F|$
3. $S_i=\{\neg C_i$ 的互补展开$\}$ //$J_M(\square)-J_M(C_i)$,变量集 M 包含了 F 中出现的所有变量
4. **While** $j\le|F_1|$
5. **If** C_i 与 C_j 互补 **Then skip**
6. **Else**
7. **While** $k\le|C_i|$

8. **If** $S_i[k]$与 C_j 互补 **Then skip**
9. **Else if** $S_i[k]\models C_j$ **Then** $F_2=F_2\cup\{C_j\}$
10. **Else if** $C_j\models S_i[k]$ **Then** $F_2=F_2\cup\{S_i[k]\}$
11. **Else** $F_2=F_2\cup\{C_j\vee(S_i[k]-C_j)\}$
12. k++
13. j++
14. k=1
15. $F_1=F_2$
16. $F_2=\varnothing$
17. i++
18. j=1
19. **Return** F_1

算法 3 的执行过程与算法 2 大致相同.不同之处在于:算法 3 第 9 行利用了子程序 CIF,而算法 2 调用了子程序 ICE.子程序 CIF 的功能是计算任意子句与任意子句集所不能扩展出的极大项集的交集,输出结果为一个 EPCCL 理论.CIF 算法的过程是 IKCHER 算法的子过程,其正确性可参照定理 5 的证明.

4.2 两种并行的IKCHER算法

从 IKCHER 算法的求解过程中不难看出,可以对求交知识编译过程进行拆分以实现并行执行.对于输入子句集 $F=\{C_1,\ldots,C_n\}$,有两种并行方式.

1) 利用互补展开求得 F 中 n 个子句分别所不能扩展出的极大项集$\{\neg C_1\},\ldots,\{\neg C_n\}$,然后利用 PIAE 并行合并这 n 个 EPCCL 理论,所得结果为 F 所不能扩展出的极大项集.再次利用上述过程即可求得与 F 等价的 EPCCL 理论,这种方式至少需要 n 次并行化,因此,并行编译过程拆分和合并需要较大开销.
2) 对于任意子句集 $F=\{C_1,\ldots,C_n\}$,将 F 划分为 k 个子集 $F_1,\ldots,F_k$,并行求出任意 F 子集 F_i 所不能扩展出的极大项集,所得结果用 EPCC 理论 E_i 保存,然后利用 PIAE 并行合并 F 子集对应的 k 个 EPCCL 理论.再次利用上述过程即可求得与 F 等价的 EPCCL 理论,这种方式只需要 4 次并行化,因此,本文结合 PIAE 采用该方式进行 IKCHER 算法的并行化.

算法 **4**. P-IKCHER.

1. 输入:子句集 $F=\{C_1,\ldots,C_n\}$
2. t=1
3. **Divide** F **to** $\{F_1,\ldots,F_k\}$
4. **Parallel**
5. **Foreach** $F_h(1\leq h\leq k)$
6. $E_h=CKCHER(F_h)$
7. **End parallel**
8. $E=PIAE(E_1,\ldots,E_k)$
9. **If** t=1 **Then**
10. t−−
11. $F=E$
12. **Goto** 3
13. **Else return** E

Sub-procedure CKCHER (CNF $F=\{C_1,\ldots,C_n\}$)

1. h=1
2. 初始化:$F_2=\{\neg C_1\}$, $F_3=\varnothing$, i=2, j=k=1

3. **While** $i \le |F_1|$
4. S_i={¬C_i 的互补展开} //$J_M(\square)-J_M(C_i)$,变量集 M 包含了 F 中出现的所有变量
5. **While** $j \le |F_2|$
6. **If** C_i 与 C_j 互补 **Then skip**
7. **Else**
8. **While** $k \le |C_i|$
9. **If** $S_i[k]$与 C_j 互补 **Then skip**
10. **Else if** $S_i[k] \models C_j$ **Then** $\Sigma_3=\Sigma_3\cup\{C_j\}$
11. **Else if** $C_j \models S_i[k]$ **Then** $\Sigma_3=\Sigma_3\cup\{S_i[k]\}$
12. **Else** $\Sigma_3=\Sigma_3\cup\{C_j\vee(S_i[k]-C_j)\}$
13. k++
14. j++
15. k=1
16. $F_2=F_3$
17. $F_3=\varnothing$
18. i++
19. j=1
20. **Return** F_2

算法 4 采用 PIAE 策略进行 EPCCL 理论的并行求交知识编译.该算法中,第 6 行的 CKCHER 子程序用来计算任意子句集所不能扩展出的极大项集,所得结果为 EPCCL 理论,该过程为 IKCHER 算法的子过程;第 8 行的 PIAE 子程序的具体过程参照第 3 节中的算法 2.P-IKCHER 算法采用的策略是:先将输入子句集分割成 k 个子集,体现在算法 4 的第 3 行;然后并行求得各个子集所不能扩展出的极大项集,所得结果为 EPCCL 理论,体现在算法 4 的第 4 行~第 7 行;最后,利用 PIAE 算法将所有的 EPCCL 理论求交,所得结果就是与输入子句集整体所不能扩展出的极大项集等价的 EPCCL 理论.另外,该算法中用 t 控制求解出与原子句集等价的 EPCCL 理论.

在利用扩展规则对子句集进行编译的过程中,子句的长度会随着编译的过程不断增加,子句的长度增加后,其所能扩展出的极大项数会减少,扩展能力会降低.P-IKCHER 算法在对输入子句集的子集编译后,采用 PIAE 求交策略对子任务所得的 EPCCL 理论进行合并.求交过程中,保留了大量子任务输出 EPCCL 理论中的长子句.这些长子句扩展能力低,因此会需要更多的长子句来得到与输入子句集等价的 EPCCL 理论.而且并行算法使用的核数越多,得到的长子句就越多,进而编译效果和编译效率会越差.因而可预见的是:利用 P-IKCHER 算法并行编译所得结果规模会增加,编译效率相对于串行程序会有所降低.

算法 **5**. impP-IKCHER.

1. 输入:子句集 $F=\{C_1,\ldots,C_n\}$
2. t=1
3. **Divide** F **to** $\{F_1,F_2\}$
4. $E=CKCHER(F_1)$
5. $F=imp\text{-}PIAE(E,F_2)$
6. **If** t=1 **Then**
7. t––
8. $F=E$
9. **Goto** 3
10. **Else return** F

算法 5 采用 imp-PIAE 策略进行 EPCCL 理论的并行求交知识编译.该算法中,CKCHER 子程序与算法 4 中

的 CKCHER 子程序执行过程相同,第 5 行的 imp-PIAE 子程序的具体过程参照第 3 节中的算法 3.在算法 5 中,首先求得输入子句集中部分子句所不能扩展出的极大项集,并将结果保存为 EPCCL 理论;然后对利用 imp-PIAE 算法对所得到的 EPCCL 理论和剩余子句集进行并行求交操作,得到输入子句集所不能扩展出的极大项对应的 EPCCL 理论;然后对该 EPCCL 理论进行相同的操作,得到与输入子句集等价的 EPCCL 理论.同样地,该算法中,用 t 控制求解出与原子句集等价的 EPCCL 理论.

impP-IKCHER 算法利用 imp-PIAE 策略对所得的 EPCCL 理论进行求交操作,根据定理 6 以及 IKCHER 算法的执行过程不难看出,impP-IKCHER 算法是将 IKCHER 算法中不影响最终结果的可以并行的操作并行化执行.因此可以预见的是,imp-IKCHER 算法不会改变 IKCHER 算法的编译结果的规模,而且还能提升 IKCHER 算法的编译效率.

5 IKCHER 算法并行效果的测试

UKCHER 算法是另一种可并行的知识编译算法,然而其并行化的工作尚未实现,并且其编译效率和编译质量远差于 IKCHER 算法(实验结果见第 3 节),因此本文仅对比测试了 impP-IKCHER 算法、P-IKCHER 算法和 IKCHER 算法的编译效果,以此验证 EPCCL 理论并行编译的可行性.本文按照子句数/变量数≈4.3 的比例生成了若干在相变点附近的 3-SAT 实例.并行算法用多核算法设计实现,用 k 表示任务的划分粒度.

由于 P-IKCHER 算法是本文设计的一种值得借鉴的有瑕疵的并行算法,该算法所使用的核数越多,效果越差,因此本节对该算法只测试 k=2 的并行效果.实验结果为 50 次实验的平均值,结果见表 7.

Table 7 Experimental results on random 3-SAT instances with n/m≈4.3

表 7 n/m≈4.3 的随机 3-SAT 实例测试

Instances	impP-IKCHER (k=4)		impP-IKCHER (k=2)		IKCHER		DKCHER		P-IKCHER (k=2)	
	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)	Size	Time (s)
⟨26,111⟩	149	0.192	149	0.223	149	0.274	157	0.268	161	0.396
⟨27,116⟩	141	0.150	141	0.298	141	0.412	141	0.402	158	0.532
⟨28,120⟩	113	0.349	113	0.554	113	0.718	111	0.711	133	0.832
⟨29,124⟩	198	0.503	198	0.782	198	0.998	202	0.990	222	1.224
⟨30,128⟩	96	0.538	96	1.068	96	1.311	97	1.303	108	1.678
⟨31,133⟩	223	1.116	223	1.882	223	2.079	220	2.068	245	2.326
⟨32,137⟩	207	1.623	207	2.112	207	2.914	205	2.896	237	3.995
⟨33,141⟩	130	2.003	130	3.169	130	3.931	132	3.924	144	5.465
⟨34,146⟩	159	2.798	159	4.582	159	5.185	162	5.173	170	6.365
⟨35,150⟩	168	4.362	168	6.004	168	7.216	170	7.209	181	9.332
uf20-01	79	0.014	79	0.024	79	0.029	78	0.020	92	0.041
uf20-02	103	0.011	103	0.015	103	0.018	100	0.014	114	0.032
uf20-03	18	0.009	18	0.013	18	0.015	20	0.011	22	0.018
blockworld-anomaly	47	0.037	47	0.149	47	0.157	48	0.153	56	0.256
pigeon-hole-6	1	3.068	1	4.328	1	5.533	1	5.528	3	7.362
par8-1-c	62	0.026	62	0.033	62	0.034	64	0.033	72	0.051

从编译质量来看,表 7 中,对于所有实例,impP-IKCHER(k=4),impP-IKCHER(k=2)的编译质量与 IKCHER 算法均一样;P-IKCHER 算法相对于 IKCHER 算法,编译质量有所下降.这是由于 impP-IKCHER 算法先将部分子句集编译为 EPCCL 理论,然后在合并过程采用 imp-PIAE 对已编译的 EPCCL 理论以及剩余未编译的原始子句集进行并行求交,这一系列过程与 IKCHER 算法的执行流程类似,不同之处在于,impP-IKCHER 算法是将 IKCHER 算法中不影响最终结果的可以并行的操作并行化执行;P-IKCHER 算法先将原始子句集分为若干子集,然后求得这些子集所不能扩展出的 EPCCL 理论,再将这些 EPCCL 理论并行求交,再求得子集所不能扩展出的极大项后会改变原始子句集的结构,因此在合并之后很难得到与 IKCHER 算法相同的结果.而且由于 EPCCL 理论中子句的长度普遍要长于原始子句集中子句的长度,因此合并之后,P-IKCHER 算法包含了较多的长子句,这些长子句所能扩展出的极大项集的规模较小,因此需要更多的长子句来表示原有子句集,从而 P-IKCHER 算法的编译

规模会大于 impP-IKCHER 算法和 IKCHER 算法的编译规模.

从编译效率上看,表 7 中,impP-IKCHER(k=4)和 impP-IKCHER(k=2)均起到了一定的加速效果,然而加速效果并不明显,当 k=4 时,最大加速比仅为 2 倍(如表 1 中,⟨27,116⟩,⟨30,128⟩,⟨34,146⟩和 blockworld-anomaly 等实例),这是由于本文分配任务的策略采用的是按子句数划分,这种划分策略并未考虑到子句集本身的内在结构,因此负载均衡较差;P-IKCHER(k=2)未起到加速效果,反而降低了编译效率,其原因正在于它采用 PIAE 算法直接合并多个 EPCCL 理论,极大地增加了编译过程所需时间开销.

整体来看,impP-IKCHER 能够提高 P-IKCHER 算法的编译效率.原始 IKCHER 算法的编译效率稍弱于 DKCHER 算法的编译效率,但由于 impP-IKCHER 算法采用了合适的并行合并策略,使求交知识编译方法的编译效率得到了提高(当 k=2 时,impP-IKCHER 算法的编译效率已经明显高于 DKCHER 算法的编译效率).因此,采用合适的并行合并策略之后,IKCHER 算法的并行化是可行的,这一点能够为其他目标编译语言的并行编译提供借鉴.

6 结论与展望

本文提出了一种新的 EPCCL 理论编译算法——求交知识编译算法 IKCHER,并设计了两种 EPCCL 理论的并行求交算法 PIAE 和 imp-PIAE,其中,imp-PIAE 利用了输入 EPCCL 理论对于所不能扩展出极大项集的原始子句集.基于 PIAE 算法和 imp-PIAE 算法,提出了两种并行求交知识编译算法 P-IKCHER 和 impP-IKCHER.实验结果表明,IKCHER 算法有最优的编译质量以及接近最优的编译效率,而 impP-IKCHER 能够提高 IKCHER 算法的编译效率,P-IKCHER 算法的编译效率相对于 IKCHER 算法反而有所降低,说明了合适的合并策略对于知识编译算法的并行化至关重要.本文的研究成果验证了并行知识编译是可行的,该成果同时能够为其他目标语言编译算法的并行化提供借鉴.

未来,我们将研究如何进一步提高 EPCCL 理论合并算法的效率,并使 impP-IKCHER 算法有更广的应用范围.本文中,对输入子句集的分割采用了等子句数分割,而分割结果会影响负载均衡及并行算法的整体执行效率,因此,任务分割的启发式选择也将是下一步的研究重点.

References:

[1] Cook SA. The complexity of theorem-proving procedures. In: Harrison MA, Banerji RB, Ullman JD, eds. Proc. of the 3rd Annual ACM Symp. on the Theory of Computing (STOC'71). Shaker Heights: ACM Press, 1971. 151−158. [doi: 10.1145/800157.805047]

[2] Mitchell DG, Selman B, Levesque HJ. Hard and easy distributions of SAT problems. In: Swartout WR, ed. Proc. of the 10th National Conf. on Artificial Intelligence (AAAI'92). San Jose: AAAI Press, The MIT Press, 1992. 459−465.

[3] Bai S, Bu DB. Analysis for phase transition of the 2-3-SAT problem. Ruan Jian Xue Bao/Journal of Software, 1998,9(11):828−832 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/9/828.htm [doi: 10.13328/j.cnki.jos.1998.11.006]

[4] Cai SW, Su KL. Local search for Boolean satisfiability with configuration checking and subscore. Artificial Intelligence, 2013,204: 75−98. [doi: 10.1016/j.artint.2013.09.001]

[5] Luo C, Su KL, Cai SW. More efficient two-mode stochastic local search for random 3-satisfiability. Applied Intelligence, 2014, 41(3):665−680. [doi: 10.1007/s10489-014-0556-7]

[6] Luo C, Cai SW, Wu W, Su KL. Double configuration checking in stochastic local search for satisfiability. In: Brodley CE, Stone P, eds. Proc. of the 28th AAAI Conf. on Artificial Intelligence (AAAI 2014). Québec: AAAI Press, 2014. 2703−2709.

[7] Cai SW, Su KL. Comprehensive score: Towards efficient local search for SAT with long clauses. In: Rossi F, ed. Proc. of the 23rd Int'l Joint Conf. on Artificial Intelligence (IJCAI 2013). IJCAI/AAAI Press, 2013. 489−495.

[8] Jeavons P, Petke J. Local consistency and SAT-solvers. Journal of Artificial Intelligence Research, 2012,43:329−351. [doi: 10.1613/jair.3531]

[9] Katsirelos G, Sabharwal A, Samulowitz H, Simon L. Resolution and parallelizability: Barriers to the efficient parallelization of SAT solvers. In: desJardins M, Littman ML, eds. Proc. of the 27th AAAI Conf. on Artificial Intelligence (AAAI 2013). Washington: AAAI Press, 2013.

[10] Audemard G, Simon L. Lazy clause exchange policy for parallel SAT solvers. In Sinz C, Egly U, eds. Proc. of the 17th Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT 2014). LNCS 8561, Vienna: Springer-Verlag, 2014. 197−205. [doi: 10.1007/978-3-319-09284-3_15]

[11] Sonobe T, Kondoh S, Inaba M. Community branching for parallel portfolio SAT solvers. In Sinz C, Egly U, eds. Proc. of the 17th Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT 2014). LNCS 8561, Vienna: Springer-Verlag, 2014. 188−196. [doi: 10.1007/978-3-319-09284-3_14]

[12] Val DA. On some tractable classes in deduction and abduction. Artificial Intelligence, 2000,116(1-2):297−313. [doi: 10.1016/S0004-3702(99)00088-0]

[13] Broeck GV, Darwiche A. On the role of canonicity in knowledge compilation. In: Bonet B, Koenig S, eds. Proc. of the 29th AAAI Conf. on Artificial Intelligence (AAAI 2015). Austin: AAAI Press, 2015. 1641−1648.

[14] Marquis P. Existential closures for knowledge compilation. In: Walsh T, ed. Proc. of the 22nd Int'l Joint Conf. on Artificial Intelligence (IJCAI 2011). Barcelona: IJCAI/AAAI Press, 2011. 996−1001. [doi: 10.5591/978-1-57735-516-8/IJCAI11-171]

[15] Fargier H, Marquis P. Disjunctive closures for knowledge compilation. Artificial Intelligence, 2014,216:129−162. [doi: 10.1016/j.artint.2014.07.004]

[16] Darwiche A. Compiling knowledge into decomposable negation normal form. In: Dean T, ed. Proc. of the 16th Int'l Joint Conf. on Artificial Intelligence (IJCAI'99). Stockholm: Morgan Kaufmann Publishers, 1999. 284−289.

[17] Darwiche A. Decomposable negation normal form. Journal of the ACM, 2001,48(4):608−647. [doi: 10.1145/502090.502091]

[18] Darwiche A, Marquis P. A knowledge compilation map. Journal of Artificial Intelligence Research, 2002,17:229−264. [doi: 10.1613/jair.989]

[19] Fargier H, Marquis P. Extending the knowledge compilation map: Krom, Horn, Affine and beyond. In: Fox D, Gomes CP, eds. Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI 2008). Chicago: AAAI Press, 2008. 442−447.

[20] Fargier H, Marquis P, Niveau A. Towards a knowledge compilation map for heterogeneous representation languages. In: Rossi F, ed. Proc. of the 23rd Int'l Joint Conf. on Artificial Intelligence (IJCAI 2013). IJCAI/AAAI Press, 2013. 877−883.

[21] Fargier H, Marquis P, Niveau A, Schmidt N. A knowledge compilation map for ordered real-valued decision diagrams. In: Brodley CE, Stone P, eds. Proc. of the 28th AAAI Conf. on Artificial Intelligence (AAAI 2014). Québec: AAAI Press, 2014. 1049−1055.

[22] Koriche F, Lagniez JM, Marquis P, Thomas S. Knowledge compilation for model counting: Affine decision trees. In: Rossi F, ed. Proc. of the 23rd Int'l Joint Conf. on Artificial Intelligence (IJCAI 2013). IJCAI/AAAI Press, 2013. 947−953.

[23] Lai Y, Liu DY, Wang SS. Reduced ordered binary decision diagram with implied literals: A new knowledge compilation approach. Knowledge and Information Systems, 2013,35(3):665−712. [doi: 10.1007/s10115-012-0525-6]

[24] Huang JB, Darwiche A. The language of search. Journal of Artificial Intelligence Research, 2007,29:191−219. [doi: 10.1613/jair.2097]

[25] Lin H, Sun JG, Zhang YM. Theorem proving based on the extension rule. Journal of Automated Reasoning, 2003,31(1):11−21. [doi: 10.1023/A:1027339205632]

[26] Yin MH, Lin H, Sun JG. Solving #SAT using extension rules. 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]

[27] 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).

[28] Li Y, Sun JG, Wu X, Zhu XJ. Extension rule algorithms based on IMOM and IBOHM heuristics strategies. Ruan Jian Xue Bao/Journal of Software, 2009,20(6):1521−1527 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3420.htm [doi: 10.3724/SP.J.1001.2009.03420]

[29] Sun JG, Li Y, Zhu XJ, Lü S. A novel theorem proving algorithm based on extension rule. Journal of Computer Research and Development, 2009,46(1):9−14 (in Chinese with English abstract).

[30] Lin H, Sun JG. Knowledge compilation using the extension rule. Journal of Automated Reasoning, 2004,32(2):93−102. [doi: 10.1023/B:JARS.0000029959.45572.44]

[31] Yin MH, Lin H, Sun JG. Counting models using extension rules. In: Proc. of the 22nd AAAI Conf. on Artificial Intelligence (AAAI 2007). Vancouver: AAAI Press, 2007. 1916−1917.

[32] Gu WX, Wang JY, Yin MH. Knowledge compilation using extension rule based on MCN and MO heuristic strategies. Journal of Computer Research and Development, 2011,48(11):2064−2073 (in Chinese with English abstract).

[33] Liu DY, Lai Y, Lin H. C2E: An EPCCL compiler with good performance. Chinese Journal of Computers, 2013,36(6):1254−1260 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.01254]

[34] Liu L, Niu DD, Li Z, Lü S. Dynamic online reasoning algorithm based on the hyper extension rule. Journal of Harbin Engineering University, 2015,36(12):1614−1619 (in Chinese with English abstract). [doi: 10.11990/jheu.201404055]

[35] Liu L, Niu DD, Lü S. Knowledge compilation methods based on the hyper extension rule. Chinese Journal of Computers, 2016, 39(8):1681−1696 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2016.01681]

附中文参考文献:

[3] 白硕,卜东波.2-3-SAT 问题相变现象剖析及其应用.软件学报,1998,9(11):828−832. http://www.jos.org.cn/1000-9825/9/828.htm [doi: 10.13328/j.cnki.jos.1998.11.006]

[26] 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]

[27] 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.

[28] 李莹,孙吉贵,吴瑕,朱兴军.基于 IMOM 和 IBOHM 启发式策略的扩展规则算法.软件学报,2009,20(6):1521−1527. http://www.jos.org.cn/1000-9825/3420.htm [doi: 10.3724/SP.J.1001.2009.03420]

[29] 孙吉贵,李莹,朱兴军,吕帅.一种新的基于扩展规则的定理证明方法.计算机研究与发展,2009,46(1):9−14.

[32] 谷文祥,王金艳,殷明浩.基于 MCN 和 MO 启发式策略的扩展规则知识编译方法.计算机研究与发展,2011,48(11):2064−2073.

[33] 刘大有,赖永,林海.C2E:一个高性能的 EPCCL 理论编译器.计算机学报,2013,36(6):1254−1260. [doi: 10.3724/SP.J.1016.2013.01254]

[34] 刘磊,牛当当,李壮,吕帅.基于超扩展规则的动态在线推理算法.哈尔滨工程大学学报,2015,36(12):1614−1619. [doi: 10.11990/jheu.201404055]

[35] 刘磊,牛当当,吕帅.基于超扩展规则的知识编译方法.计算机学报,2016,39(8):1681−1696. [doi: 10.11897/SP.J.1016.2016.01681]

牛当当(1990－),男,陕西西安人,博士生,CCF 学生会员,主要研究领域为智能规划,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.

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