第7卷第4期 软 件 学 报 Vol. 7, No. 4
1996年4月 JOURNAL OF SOFTWARE Apr. 1996

求解SAT问题的分级重排搜索算法*

刘 涛 李国杰

(国家智能计算机研究开发中心 北京 100080)

摘要 局部搜索法在SAT问题上的成功运用已引起越来越广泛的重视,然而,它在面对不可满足问题例时的局限性不能不被考虑.分级重排搜索算法MSRA(multi-stage search rearrangement algorithm)正是为克服局部搜索法的不完备性而提出的,准确地讲,它是几种算法在思想上的集成,但为明确起见,把其最典型的分级重排过程作为名称.分级重排搜索算法在求解SAT问题时,能表现出优于单一求解策略(如局部搜索法或回溯算法)的明显特性.由于可根据约束条件的强弱来估计SAT问题例的可满足性,因此能够以此来确定更有效的求解策略.

关键词 3-SAT问题,局部搜索,回溯算法,分级重排,D-P算法.

SAT(satisfiability)问题及其求解算法在人工智能、VLSI设计和计算理论等领域有着广泛的应用背景,有关实用算法的研究已成为国内外研究的热点之一.

一般说来,SAT问题有2大类常见的求解算法:局部搜索法[1~4]和回溯算法.[5]其中,局部搜索算法成功的一个主要原因就是它能简单而有针对性地利用问题的本质特性.如文献[3]就是依据了某类数学物理模型,提出了一种求解SAT问题的"拟物算法".

众所周知,对于一个SAT问题例而言,求解算法只需回答满足与否.事实上,当面对的是一个可满足问题例时,直接寻找一个解似乎比证明有解更容易;而当面对的是一个不可满足问题例时,证明其无解又似乎比寻找一个"最优解"(大于零的冲突数,即不可满足的子句个数)更容易.

分级重排搜索算法正是基于上述事实而选择了如下的总体策略:

假定所给问题例可满足,运用局部搜索法寻找解.如果估计问题例无解或在容许的时间内找不到解,那么设法证明问题例无解,证明的方法属于回溯算法之列.

尽管对这样的求解系统难以给出准确的理论分析,但如下的事实保证了算法的总体性能,即当问题有解时,几乎所有的问题例都可以用局部搜索法在很短的时间内找到一个解.

"分级重排"有几层含义:

(1)它是指上述的2阶段策略.如果SAT问题的约束较弱,可直接使用局部搜索法来求解该问题;而当问题的约束较强时,则直接使用回溯算法来求解该问题.问题约束的强弱可

---

* 作者刘涛,1965年生,博士生,主要研究领域为优化算法,计算复杂性,人工智能.李国杰,1943年生,研究员,博士导师,院士,主要研究领域为并行处理,人工智能,搜索技术,组合优化.

本文通讯联系人:李国杰,北京100080,国家智能计算机研究开发中心

本文1995-02-21收到修改稿

根据经验(如:子句数与变量数之比值的大小)来估计.

(2)在"定理证明"过程中,分多层对约束进行简化.由于D-P(davis-putnam)算法及其变种在处理SAT问题时十分有效,我们将其与回溯算法结合起来,允许算法在回溯搜索树中的几个层位上进行D-P过程.在MSRA算法中,我们使用了更广泛的D-P简化方法.

(3)回溯算法中最强约束变量优先分枝(扩展)策略被认为是有效的.然而,对于按均匀分布生成的随机SAT问题只对变量作静态(一次性)排序显然是不够的.在回溯过程进行的最初阶段(目的是将SAT问题分解成若干子问题),我们选择了一种动态排序方法,即每次都扩展当前(被认为的)受约束"最强"的变量.当然,约束强弱的衡量只能是一种近似,我们使用的标准与一个变量所涉及的约束条件个数有关.

(4)重排过程还包含有对子问题表示进行简化,并使用静态排序方法对变量做一次性排序.我们知道分枝过程在算法的初始阶段效率很高,但随着分枝变量数的增加,一些不必要的查找操作浪费了许多时间.这主要是因为一些约束条件已被满足,而其中的未扩展变量实际上已没有考虑的必要,这样只要将对应于未扩展变量的检索表简化就可以大大加快搜索过程.

需要提醒注意的是,在求解具体SAT问题例时,多级重排算法的各种策略还需根据具体情况来灵活使用.一般说来,对于规模较小的问题例(如满足均匀分布的随机3-SAT问题中变量数 $m\leqslant 150$),基本的回溯算法就足以解决任何难解问题例.对于规模较大而且较难的问题例,则需要使用全部策略.

尽管在某些情况下MSRA将"退化"为局部搜索算法或回溯算法,由于局部搜索和回溯过程是MSRA的基本组成部分,且本文所谈及的局部搜索算法和回溯算法与常见的算法略有不同,我们将专门对其进行描述.

1 SAT问题的几种表示方式

SAT问题的一般性描述如下:

给定一组布尔变量 $x=\{x_1,x_2,\cdots,x_m\}$,它们的某个合取范式(CNF)是否可满足?

实际上,如果我们能找到这一组布尔变量 x 的某种赋值使布尔表达式取真,问题就是可满足的;否则问题不可满足.

按一般习惯,用0对应于"假",1对应于"真",SAT问题就变成了寻找 x 的某个0/1赋值使

$$f(x)=C_1\cdot C_2\cdot\cdots\cdot C_n=1$$

其中 C_i 称为子句(Clause),它由几个文字(Literals)的和("或")组成.一个文字既可以是一个布尔变量 x_i 也可以是它的非($\overline{x_i}$),子句的长度即是该子句中所包含的文字个数.

显然,只有在每个子句都取真值(1)时,SAT问题才可以满足.

如果将布尔表达式 $f(x)$ 求反,就可以得到SAT问题的第3种表示形式:

$$g(x)=M_1+M_2+\cdots+M_n=0?$$

其中,M_i 称为一个布尔单项式,它由几个文字的积("与")组成.注意 $g(x)$ 中的文字与 $f(x)$ 中的文字互反,这样,SAT问题就变成了一个布尔方程求根的问题.我们称一个可满足

的 SAT 问题有解,称不可满足的 SAT 问题无解.

在本文中,我们将主要讨论均匀分布情况下的 SAT 问题,即每个变量及其"否定"在各个子句中出现的概率相同.此外,我们还要求每个句子中不允许出现重复的变量,所有的子句长度都相同.

2 "弱"约束情况下的局部搜索算法

J. Gu,李未,黄文奇和 B. Selman 等人的工作[1~4]已经证实了优化方法求解 SAT 问题的有效性.我们在这里将算法的适用范围限定在约束较弱(足以使问题有解)的问题例上,然后给出几种新的求解策略.

运用优化方法的首要一步是建立目标函数.如前所述,我们将冲突数(未满足子句的个数)作为目标函数,由于每个变量 $x_i(i=1,2,\cdots,m)$ 只能取二值(0 或 1),其可行解空间大小为 2^m.尽管保证找到全局最优解不是一件容易的事情,而对 SAT 问题,其目标函数的最小值是零,一旦某个局部最优解使冲突数变零,算法就可以停止并输出一个解.

本文所使用的求解 SAT 问题的局部搜索过程描述如下:

```
Procedure Find Solution
Input: g(x) / * 布尔函数 * /
Output: True or Local Minimum Conflicts / * "真"或局部最小冲突数 * /
begin
    x = initial_solution();
    for K = 1 to Number of loops
        if Conflicts(x) == 0 then return true;
        for j = 1 to n
            if M_j ≠ 0 then
                w(M_j) = w(M_j) + 1; / * M_j 的权加 1 * /
                with probability P(j) = kw(M_j)
                    x' = testswap(x_i); / * x_i ∈ M_j * /
                    if Conflicts(x') ≤ Conflicts(x)
                    then x = x';
            endif
        endfor
    endfor
end Procedure
```

在这里我们使用了 SAT 问题的第 3 种表示形式.其中 x 代表一个可行解的形式,x_i 是单项式 M_j 中的一个变量.$P(j)$的大小由 M_j 的权值来决定,权值越大,$P(j)$越大.

testswap 函数的功能是将 x_i 代之以 $\overline{x_i}(=1-x_i)$,并返回一个新的解矢量 x'.函数 Conflicts 计算当前变量赋值的冲突数,注意实际上只需计算由于 x_i 的"翻转"而引起的冲突数变化.

上述 Find Solution 过程与一般局部搜索算法有 2 个重要区别.首先,我们在这里使用的不等式为"≤",而不是"<".这样做的一个直观解释是要与目标函数的选取协调起来.我们知道,冲突数的最大值只能是子句的个数 n.在大多数情况下,局部搜索算法可以很快使冲突数大幅度下降.此时,冲突值都分布在值较小的范围里,算法中使用的不等式"≤"可以扩大算法的搜索空间,减少算法落入局部最优点的次数.此外,我们在算法中使用"权"来避免算法过早陷入局部最优.在这里有必要强调的是本文所用策略与文献[4]中的"加权"技术

不同,在文献[4]中实际上要修改目标函数.经验还告诉我们,子句的权值越大,表明该子句"经常"得不到满足,实质上是因为这些子句中包含了某些强约束变量.对于原始的随机(均匀分布)SAT问题而言,各变量的约束强弱比较接近,所以在搜索过程中,有时是不满足子句的个数较小,但它们又不是固定的几个子句.在这种情况下,该问题例多半是无解的,然而,要证明它无解却是相当困难的,当问题例的规模稍大(变量个数大于300)时甚至不能在容许的时间内证明它无解.我们把这一困难归因于Co-NP问题的难解性.

上述局部搜索算法的收敛速度较快,但这也难以避免算法陷入局部最优.为此,我们提供了如下2种简单而实用的策略:

(a)当冲突数在有限步之内不再降低时,重新初始化矢量x,再进行一次寻解过程.

(b)当已得到一个局部最优解时,强行改变部分(或全部)变量的赋值,再进行寻解过程.

3 "强"约束情况下求解小规模SAT问题的回溯算法

关于具体SAT问题例的约束强弱只能是一个估计,因为无法预知该问题例中约束条件是否强到包含矛盾的程度.目前,还没有证据表明可以完全摆脱回溯过程来说明一个SAT问题例是无解的.在这方面,许多改进的回溯算法可以使我们乐观地处理这一问题.

不失一般性地,我们将着重讨论3-SAT问题的求解算法.

对于规模较小的3-SAT问题(变量个数$m \leqslant 150$的任何随机问题例),基于约束传播的回溯算法被证实为一种十分有效的算法.尽管基本回溯过程只是作为多级重排搜索算法(MSRA)的基本组成部分,但由于它在求解某些小规模的SAT问题时十分有效,所以,我们在此首先给出MSRA中所使用的回溯算法描述:

```
Procedure CPBA
Input: f(x) / * 布尔表达式 * /
Output: true or false / * 如果有解返回 true,否则返回 false * /
begin
    PosN=0;
    v_i=select_var(); / * 选择一个(约束最强)的 v_i * /
    Value(v_i)=1;
    branch_var=v_i; / * 标记 v_i 为扩展变量 * /
    free(v_i)="true"; / * 标记 v_i 为自由变量 * /
    Tos=1;
    isat(v_i);
    while PosN<n and Tos>0 do
        if v_i=select_unfreevar(); / * 存在一个未经扩展的非自由变量 v_i * /
        then branch_var=v_i and unfree(v_i)="true"; / * 标记 v_i 为扩展变量和非自由变量 * /
        else v_i=select_var(); / * 选择一个未经扩展(约束较强)的变量 v_i * /
            branch_var=v_i and free(v_i)="true"; / * 并将其标记为扩展变量及自由变量 * /
        endif
        Tos=Tos+1;
        isat(v_i);
    endwhile
    if PosN==n return true;
    if Tos==0 return false;
end Procedure
```

注意上述算法中一个自由变量可允许0/1两种取值,如果一个变量只允许一种取值则

该变量为非自由变量.算法中 $PosN$ 为已被满足的子句个数,Tos 为已扩展变量("分枝变量")的个数.$isat()$函数的主要作用是检查当前扩展变量取值是否与已扩展变量取值矛盾.如果没有矛盾,修正 $PosN$ 的值,并返回一组新的非自由变量及其取值(如果存在的话);否则,回到这之前的第一个自由变量并将该变量标记为只能取唯一值的非自由变量.由于篇幅所限,我们不可能给出 $isat()$函数的详细描述,但在这里我们有必要介绍一下具体程序中所使用的一种标记方法.例如对于 3−SAT 问题,程序中采用了如下的标记:

$$Stateofclause[C_i]=\begin{cases}0, & \text{子句中不存在已扩展变量}\\ 1, & \text{子句中的第 1 个扩展文字取真}\\ 2, & \text{子句中只有第 2 个扩展文字取真}\\ 3, & \text{子句中只有第 3 个扩展文字取真}\\ -1, & \text{子句中第 1 扩展文字取假}\\ -2, & \text{子句中前 2 个扩展文字取假}\end{cases}$$

在这里,我们使用了扩展文字的概念.扩展文字的真假可以由该文字的形式决定,它或者是某个变量或者是该变量的非.

这样,我们在进行回溯操作时,如果相应子句 C_i 的标记号小于零时,只需将子句标号加 1;否则子句 C_i 的标号变为$(1-Stateofclause[C_i])$,并将 $PosN$ 减去 1.

很明显,CPBA 是一个完备算法,只要稍作改动,就可将其变为一个求全部解的算法.

当运用上述算法求解具体问题时,有 2 个因素是必须考虑的.首先是如何选择一个自由变量进行扩展,我们所使用的策略是最强约束变量优先策略,变量的约束强弱主要是以变量所占子句个数来决定的,即如果一个变量出现于较多的子句之中,则该变量约束较强.在这里我们只选择静态排序.另一个至关重要的因素是如何选择一个非自由变量(即此未扩展变量出现于某一子句中,而该子句中所有其它变量的取值都未能使该子句为真).在我们的程序中使用的是一种自然序,每次都选择编号最大的非自由变量作为当前扩展变量.

最后,需要指出的是对于不可满足的问题例而言,考虑某一扩展变量的取值(真或假)顺序似乎是没有必要的,因为目前还不存在这样一种完备算法,而它不必一一遍历扩展变量的所有可能取值.

4　"强"约束情况下求解大规模 SAT 问题的分级重排搜索算法

前述回溯算法对于更大规模的难解 SAT 问题例效率很低,为此我们必须综合运用多级重排搜索算法的各种策略.

对于"弱"约束情况下的 SAT 问题,局部搜索算法就可以直接解决问题;在"强"约束情况下,算法 MSRA 认为该问题例是不可满足的并试图证明它无解,这就促使我们引用广义的 D−P 过程来将布尔函数(目标函数)进行化简以突出某些"强"约束变量.

一般的 D−P 过程包含有如下 3 个消解(Resolution)规则[3]:

规则 1:单位消解(Unit Resolution)——如果有某个子句只包含一个文字 y,则该文字 y 只能取真,我们可以消去所有包含有 y 的子句并以 C'_i 代替所有子句 $C_i=C'_i+\bar{y}$,如果 C'_i 为空,原问题无解.如果子句数变为零,则原问题有解(可满足).

规则2:纯消解(Pure Resolution)——纯文字是只以 x_i(或$\bar{x}_i$)的形式出现于所有子句中的文字.如果在一个问题例的布尔表达式中出现了这样一个文字,可以将该文字赋真值,并将包含有该文字的所有子句消去.

规则3:句间消解——存在2个子句 $C_i=C'_i+x$, $C_j=C'_j+\bar{x}$,且 C'_i 与 C'_j 之间没有其它的互补变量(如 x),那么可以在原布尔表达式上加入新的子句($C'_i+C'_j-C'_{ij}$),其中 C'_{ij} 为 C'_i 和 C'_j 的共同文字.

上述3个消解过程与一个分枝过程结合就成为一个求解SAT问题的完备算法——D—P算法.这一算法特别适用于子句长度非固定的随机SAT问题,然而,当它求解诸如3—SAT这样的子句长度相同的SAT问题时,一般需借助于分枝过程.

分枝过程是有选择地对某些变量赋值的过程,相当于回溯算法中的变量扩展过程,它的一个主要作用是创造了更多的机会来使用D—P过程.由此可见,分级重排算法的第2层含义(约束化简)和第3层含义(变量扩展策略)之间的密切关系.

一般说来,对于较难的3—SAT问题例($n/m=4\sim5$, $m\geqslant200$),这一步极少有机会使用单位消解规则.使用频率最高的是规则3,它有可能使布尔表达式变得过于庞大.为此,我们在规则3中加入了如下限制条件:

合并后的子句长度不得超过原来最长子句的长度.

下面是对3—SAT问题例运用规则3时可能出现的几种情况,其中"→"表示的是等价关系.

(1) $(x_i+x_{j1}+x_k)(x_i+x_{j2}+\bar{x}_k)\rightarrow(x_i+x_{j1}+x_k)(x_i+x_{j2}+\bar{x}_k)(x_i+x_{j1}+x_{j2})$

(2) $(x_i+x_j+x_k)(x_i+x_j+\bar{x}_k)\rightarrow(x_i+x_j)$

(3) $(x_i+x_{j1}+x_k)(x_{j2}+\bar{x}_k)\rightarrow(x_i+x_{j1}+x_k)(x_{j2}+\bar{x}_k)(x_i+x_{j1}+x_{j2})$

(4) $(x_i+x_j+x_k)(x_i+\bar{x}_k)\rightarrow(x_i+x_j)(x_i+\bar{x}_k)$

(5) $(x_i+x_k)(x_j+\bar{x}_k)\rightarrow(x_i+x_k)(x_j+\bar{x}_k)(x_i+x_j)$

(6) $(x_i+x_k)(x_i+\bar{x}_k)\rightarrow x_i$

为增强D—P算法的能力,我们将消解规则扩充至6个.

规则4:异或消解——如果布尔表达式中同时出现有2个子句(x_i+x_j)、$(\bar{x}_i+\bar{x}_j)$,那么用$\bar{x}_i$代替 x_j 并由此消去变量 x_j.

规则5:同值消解——如果布尔表达式中出现有2个子句$(x_i+\bar{x}_j)$,$(\bar{x}_i+x_j)$,那么用 x_i 代替 x_j,并由此消去变量 x_j.

规则6:广义纯消解——如果存在有某个变量子集,其中各变量的某种赋值能够使所有包含这些变量的子句为真,那么我们可以将这些已满足的子句消去.

对于较难的问题例,即使运用了所有上述6种规则也只能保证充分简化了布尔函数.所需的分枝过程是由上一节中描述的回溯算法来实现的.

直接应用回溯算法来实现分枝过程的一个明显好处是避免了对原布尔表达式进行不必要的修改操作,而且只使用标记操作,这就使分枝过程变得更为有效.

虽然一般回溯算法可以完成分枝过程,但还必须考虑所谓分枝策略问题.

第1个问题是如何确定扩展(分枝)变量的优先级.我们知道,选择最强约束变量作为当

前的扩展变量可以最大限度地缩小搜索树.为此,我们选择了如下的简单标准来将变量由大到小排序:　　$M_p \cdot min(p_k,q_k)+p_k+q_k$

其中 M_p 是某个正整数,常取 5～10;

$p_k=p_{k1}+p_{k2}$, $q_k=q_{k1}+N_q \cdot q_{k2}$;

p_{k1}是包含有文字 x_i 的长度为 3 的子句个数;

q_{k1}是包含有文字$\bar{x}_i$ 的长度为 3 的子句个数;

p_{k2}是包含有文字 x_i 的长度为 2 的子句个数(其中长度为 2 的子句是由 D－P 消解过程和子问题化简过程而来的;

q_{k2}是包含有文字$\bar{x}_i$ 的长度为 2 的子句个数;

N_q 为 2～4 之间的某个数.

在求解大规模问题时,在分枝过程的初始阶段(即问题分解过程)使用变量动态排序方法是较合理的.而后续分枝过程(即子问题求解过程)中,我们则用快速排序算法将所有变量只进行一次排序(静态排序).

第 2 个与分枝策略有关的问题如何选定子问题个数(或者子问题大小).我们使用了如下的经验标准:保证经分枝后子问题的布尔表达式中长度是 2 的子句数与长度是 3 的子句数之比近似为 1:10.例如,当 $n/m\approx4.3$ 时,如果 $m=200$,则分枝变量数约为 7 而子问题数一般不大于 100;如果 $m=300$,则分枝变量数约为 13 而子问题数一般小于 500.

在算法完成了初始分枝过程之后就进入如下的第 3 步,即开始对各个子问题分别求解.在正式对各子问题进行搜索之前,同样使用已扩充的 D－P 规则来化简表达式并使用相同的子句长度限制.

在子问题布尔表达式的化简过程中应该考虑避免重复操作.例如,$(x_i+x_j+x_k)(x_i+\bar{x}_k)$可以由$(x_i+x_j+x_k)(x_i+x_{j1}+\bar{x}_k)$经分枝过程得到.假设原来的 D－P 消解过程是充分的,则没有必要再检查这种消解的可能性.同样道理,$(x_i+x_k)(x_i+\bar{x}_k)$也是不可能出现的.为此,我们选择如下的方案来执行 D－P 过程:

(1)将子问题中的子句按长度 2 和 3 分为 2 组.只考虑 2 组内执行消解规则的可能性,由此生成的"新子句集"将单独存储起来.

(2)在"新子句集"内消去重复子句并进行句间消解,然后将化简了的"新子句集"附加到原布尔表达式上去.

(3)执行附加子句与原子句间的可能消解过程,并将生成的"新子句集"存储起来.如果新生成的子句数小于 $N_c(>0)$,则终止此过程.

稍作注意就可以发现前述的子句长度限制条件可以在这里得到某种补偿.如$(x_i+x_k)(x_j+\bar{x}_k)$ 完全可能是由$(x_i+x_{j1}+x_k)(x_j+x_{j1}+\bar{x}_k)$ 经分枝过程得到,然而在分枝前此句间消解是不允许的.

当算法进入第 4 步时,可以完全用 CPBA 过程来求解各个子问题.但对于那些问题规模更大的难解问题例,则需要适当地重复算法中的第 2、3 步直至完全证明该问题无解或是找到一个解.

一般情况下,对于经过最后一级分枝过程所得到的子问题,我们只进行一次变量扩展序

重排.这是因为子问题已变得相当易解,多余的操作只能是浪费时间.在这里,我们也取消了消解过程,完全使用CPBA过程来实现小的子问题求解.

简单地讲,算法的最后一级分枝过程的结果只是将最后的子问题"复制"一遍,那么分枝与扩展的等价关系是否使这一步变得没有必要?回答是我们可能因此而成倍地提高算法的执行速度,其理由是省去了一些不必要的检索操作.注意这一复制过程丝毫未改变搜索树的形状.

至此,分级重排搜索算法(MSRA)的详细过程就介绍完毕.为简明起见,将其总结为如下几个关键步骤:

(1)视具体情况首先用局部搜索法找问题的一个解.如果成功则停止计算;否则,继续以下各步;

(2)运用已扩充的D—P过程化简布尔表达式;

(3)由分枝过程将SAT问题分解为若干子问题,如果子问题过大,返回2;否则,继续以下各步;

(4)扩展所有非自由变量并"复制"子问题;

(5)用回溯算法(CPBA)求解所有子问题.

注意在2~5中的任何一步都可能出现子问题有解的情况,一旦这一情况发生立刻停止运算,说明原问题可满足;否则需求解完所有子问题.

5 算法性能比较

我们用C语言实现了多级重排搜索算法和采用同样分枝策略(最强约束优先)的D—P算法.表1为所作的实验性能比较.注意这里所研究的SAT问题例都是按均匀分布生成的随机3—SAT问题例.由于篇幅所限,我们只公布那些被认为[4]是困难问题例($n/m=4.3$)的算法性能比较数据.

一般情况下,当$n/m\leqslant 4$时,局部搜索算法就可很快找到解,而当$n/m\geqslant 5$,则问题基本上无解.

表1 MSRA与DPA求解3—SAT问题例的算法性能比较(时间单位:s)

问题例		平均运行时间		可满足比率	可满足例的平均运行时间	
m	n	MSRA	DPA	Ratio of Sat.	MSRA	DPA
50	215	0.12	0.5	62/100	0.02	0.4
70	301	0.40	1.4	57/100	0.03	1.0
100	430	0.85	5.6	51/100	0.05	4.2
120	516	2.46	27.6	44/100	0.08	8.1
140	602	3.90	86.3	39/100	0.12	11.5
150	645	8.62	114.2	5/10	0.52	30.2
170	731	20.04	252.6	5/10	0.84	43.9
200	860	70.5	>3600	4/10	1.35	58.7
250	1075	510	>3600	4/10	4.27	105
300	1290	2230	>3600	5/10	8.54	680

需要指明的是,当$n/m\leqslant 4$或$n/m\geqslant 5$时多级重排算法的性能会更好.此外,上述算法的

比较只是粗略进行的,因为我们所给出的实例并不多(m=50～140 时所用实例为 100 个,m=150～300时所用实例为 10 个),然而这足以看出 MSRA 的优越之处.

注意表 1 未反映出局部搜索法的"失误率".失误率是局部搜索法在某一固定循环次数之内未能从一个可满足问题例中找到解的概率.失误率一般与问题例的规模和难度成正比,对于表 1 中的问题例,失误率可以保证在 10%内.

6 结　论

从本文的实验结果可以看出:将局部搜索算法和基于回溯的搜索算法进行集成而形成的分级重排搜索算法(MSRA)在求解 SAT 问题时,能表现出优于单一求解策略的明显特性.由于已存在一种估计 SAT 问题例是否有解的简单方法[6](根据约束条件的强弱),我们就能够以此来确定更有效的求解策略.

对满足均匀分布的随机 3—SAT 问题而言,当子句数 n 与变量数 m 之比 (n/m) 小于 4 时,所要求解的 3—SAT 问题例一般是可满足的,通常只需用局部搜索算法就可很容易地找到一个解,因而其算法效率明显高于回溯算法;当 $n/m > 5$ 时,所要求解的 3—SAT 问题例一般是不可满足的,局部搜索算法在此时是无能为力的,我们因此选择了改进的回溯算法来证明问题例无解.实验结果表明:n/m =4～5 时,随机 3—SAT 问题例往往是一些困难的问题例,其可满足性几乎无法预测.因为局部搜索算法的不完备性,回溯过程就难以避免,从某种程度上说,本文可作为改进回溯算法效率的一种新的尝试.最后,有必要强调:尽管本文的讨论主要是针对 3—SAT 问题而言,但我们很容易将其推广至一般的 SAT 问题求解过程.

致谢　作者在此特别感谢白硕教授在算法设计和实现过程中提出的宝贵建议和意见.此外,感谢顾钧(J. Gu)教授所提供的资料和信息.

参考文献

1　Gu J. How to solve very large—scale satisfiability(VLSS) problems. Tech. Rept. 1988, 1988.

2　Gu J. Efficient local search for very large—scale satisfiability problem. SIGART Bulletin, ACM Press, 1992, **3(1)**: 8～12.

3　李未,黄文奇.一种求解 SAT 问题的数学物理算法.中国科学,1994, **24(11)**:1208～1217.

4　Selman B, Levesque H, Mitchell D. A new method for solving hard satisfiability problems. AAAI'92, San Jose, CA 1992, 1992. 440～446.

5　Davis M, Putnam H. A computing procedure for quantification theory. J. of ACM, 1960. 201～215.

6　Cheeseman H, Kanefsky B, Taylor W M. Where the really hard problem are. Proceedings IJCAI—91, 1991. 163～169.

MULTI—STAGE SEARCH REARRANGEMENT ALGORITHM FOR SOLVING SAT PROBLEM

Liu Tao　Li Guojie

(*National Research Center for Intelligent Computing System　Beijing*　100080)

Abstract Recently, local search method has been successfully applied to solve large scale SAT problem, but it will fail to find solution when an original SAT problem instance is unsatisfiable. MSRA(multi—stage search rearrangement algorithm) is just proposed to overcome the incompleteness of local search method. Properly speaking, MSRA is based on the "separate and conquer" strategy and is the integration of several algorithms. While solving the SAT problem, MSRA is more efficient than a single method, such as local search and backtracking method. Since the satisfiability of a SAT problem instance can be estimated by the strength of constrained conditions, the authors could find a more efficient strategy to solve the instance.

Key words 3—SAT problem, local search, backtracking algorithm, multi—stage search rearrangement, D—P algorithm.