ATLAS 语言实现中的设备分配算法研究

郭德贵,刘 磊,金 英,程 斌
(吉林大学计算机科学与技术学院,吉林长春 130012)

摘 要: ATLAS 是一种专业领域测试语言,其特有的设备分配给该语言的实现带来一定困难.本文提出了一种基于剥夺的启发式双重回溯搜索静态设备分配算法;并针对实际应用中 ATE 的特点,给出了两个实用的启发函数指导设备分配过程.通过若干实例表明,该算法能够以较高的效率实现静态设备分配.

关键词: 设备分配; 启发函数; 双重回溯搜索算法; 设备分配请求
中图分类号: TP31 **文献标识码:** A **文章编号:** 0372-2112 (2007) 11-2205-06

Device Allocation Algorithm of ATLAS Language Implementation

GUO De-gui, LIU Lei, JIN Ying, CHENG Bin
(*College of Computer Science and Technology, Jilin University, Changchun, Jilin* 130012, *China*)

Abstract: As a test language of special domain, ATLAS (Abbreviated Test Language for All Systems) is difficult to be implemented for its proper device allocation. This paper proposes a kind of static device allocation algorithm based on the privative heuristic duple-back-tracing searching. Furthermore, according to the character of practical ATE (Automatic Test Equipment), two practicable heuristic functions are provided to guide the process of device allocation. Finally some instances are shown to indicate the algorithm can implement the static device allocation with better efficiency.

Key words: device allocation; heuristic function; duple-back-tracing searching algorithm; device allocation request

1 引言

ATLAS(Abbreviated Test Language for All Systems)是一个被广泛应用于军事和电子测试的通用标准测试语言.用该语言编写的测试程序不依赖于任何特殊的被测系统,并且它能在自动测试设备(ATE)上运行[1~3].设备分配是实现 ATLAS 语言编译器的核心模块之一,其功能是按照用户程序提出的申请信息为每个被测试设备经过开关网络分配合适的测试设备.该模块不但在具体实现上具有较大的难度,而且其效率难以保证,有时甚至会成为系统的瓶颈,严重影响整个系统的运行效率.

本文提出了一种基于剥夺的启发式双重回溯的静态设备分配算法.实验表明,该算法能够正确地进行静态设备分配,并提高了测试程序的运行效率.

2 设备分配问题的抽象描述

在自动测试系统(ATE)中,被测试设备和相应的测试设备之间是通过开关网络实现互连的,图 1 是一个简单的 ATE 系统示例.

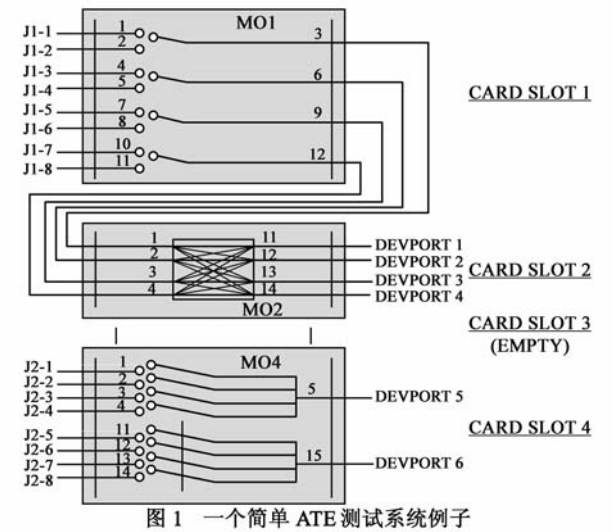

图 1 一个简单 ATE 测试系统例子

图 1 左部的 J1-1,J1-2,…,J3-4 是被测设备的引脚,图右部的 DEVPORT1,…,DEVPORT6 则是测试设备的引脚,两者之间是由多种不同类型的开关组成的开关网络.设备分配的目的是按照用户的请求寻找被测设备到测试设备之间的路径.

收稿日期:2006-09-25;修回日期:2007-08-30
基金项目:吉林省科技发展计划项目基金(No.20050527);博士学科点专项科研基金(No.20060183044)

定义 1 设备分配图 设备分配图 G 是一个无向图,用一个二元组$\langle V,E\rangle$来表示,其中 V 为顶点集合,E 为边的集合.顶点集合 $V=J\cup S\cup D$,其中 J 为被测单元引脚集合,S 为开关触点集合,D 为测试设备引脚集合;边集合 $E=\{(v_i,v_j)\mid v_i$ 和 v_j 在 ATE 系统中有连线或 v_i 和 v_j 分别为开关(矩阵)左右两侧的触点,$v_i,v_j\in V\}$.

定义 2 设备分配请求 DR 设 $G=\langle V,E\rangle$为一个设备分配图,$V=J\cup S\cup D$.一个设备分配请求 DR 是 G 中一个有序顶点对$\langle a,b\rangle$,$a\in J$,$b\in J\cup D$,表示需要在 G 中寻找一条 a 和 b 之间的连通路径.此时,称 a 为 DR 的源节点,称 b 为 DR 的目标节点.

定义 3 k 元设备分配请求集合 DR-k 设 $G=\langle V,E\rangle$为一个设备分配图,$V=J\cup S\cup D$.k 元设备分配请求集合就是由 k 个设备分配请求组成的集合$\{DR_1,\cdots,DR_k\}$($DR_i=\langle a_i,b_i\rangle$,$i=1,\cdots,k$),并且要求对于任意 i 和 j 有 a_i,a_j,b_i,b_j 互不相等.k 元设备分配请求集合可记为 DR-k.

定义 4 一个设备分配请求的解 设 $G=\langle V,E\rangle$为一个设备分配图,$V=J\cup S\cup D$,DR 是针对 G 的一个设备分配请求($DR=\langle a,b\rangle$),则称 G 上的一条路径 p 为 DR 的一个解,其中 $p=\langle a,b\rangle$,或者 $p=\langle a,s_1,s_2,\cdots,s_n,b\rangle$,$s_1,s_2,\cdots,s_n\in S$,$a\in J$,$b\in J\cup D$.

定义 5 k 元设备分配请求集合的解 设 $G=\langle V,E\rangle$为一个设备分配图,$V=J\cup S\cup D$,DR-k 为一个 k 元设备分配请求集合(DR-$k=\{DR_1,\cdots,DR_k\}$,$DR_i=\langle a_i,b_i\rangle$,$i=1,\cdots,k$),则称 G 上的 k 条路径组成的集合$\{p_1,\cdots,p_k\}$是 DR-k 的解,如果满足:

(1)p_i 是 DR_i 的解;

(2)对于任意 i 和 j,p_i 和 p_j 无公共顶点.

通过上面的定义,设备分配问题抽象为这样一个问题:对于给定设备分配图 G 和 k 元设备分配请求集合 DR-k,在 G 中求 DR-k 的解.

3 启发式双重回溯剥夺算法

在图中查找顶点不相交路径问题是一个基本的计算问题,它有很多具体应用.对任意图,该问题是一个 NP 完全问题,并且即使输入为一个平面图,该问题仍然是 NP 问题[4].

问题既然是 NP 难解的,通常的做法是分析输入图 G 的某些特征并加入适当的限制条件进而得到一个 P 问题.然而图的特征分析以及条件的寻找往往比较困难且需要复杂严格的证明,得到的算法往往结构很复杂,而且对图的特征依赖很大,导致算法的适用范围很窄.Robertson 和 Seymour 在文献[5]中就指出,当输入 k 确定时,该问题是 P 问题.文献[6]给出了 $k=2$ 时的一个更为有效的算法.

Hochbaum 在文献[7]、Broder 在文献[8]、Shamir 和 Upfal 在文献[9]中也研究了在任意随机图模型 G 中顶点不相交路径问题,提出了一些高效的算法,但是他们都对图模型或者顶点对数目进行了限定.类似的研究和应用可以参见文献[10~13].

本文提出的启发式双重回溯剥夺设备分配算法不是对 k 元设备分配请求和设备分配图增加限制条件,而是采用一种合理的剥夺方式搜索路径,并且引入了启发函数来指导设备分配的进行.

3.1 算法思想

启发式双重回溯剥夺算法是对输入 k 元设备分配请求集合 DR-k,在任意设备分配图 G 中求解满足这 k 元设备分配请求集合的顶点不相交的连通路径.

算法使用带启发信息的双重回溯过程.第一层回溯发生在寻找一个分配请求的连通路径的搜索过程中;第二层回溯发生在一个分配请求剥夺其他请求的连通路径上的顶点时.剥夺发生时,首先暂停当前请求的搜索转而在被剥夺顶点处继续搜索被剥夺请求的连通路径,如果被剥夺请求在剥夺后仍能被满足,则剥夺发生,否则回溯到剥夺发生之前进而剥夺其他节点.不论普通的搜索还是剥夺,都在启发信息指导下进行.

3.2 相关抽象域定义

为了便于算法描述,首先给出相关的抽象域定义.

(1)ATE 系统中的三类引脚:

J 表示被测设备引脚集合;

S 表示开关引脚集合;

D 表示测试设备引脚集合;

(2)ATE 系统中的所有引脚集合:$V=J\cup S\cup D$

(3)设备分配图抽象定义:

$G\in DevAG=NODE^*\times EDGE^*$

$NODE=PinNum\times F1Val\times F2Val$

$PinNum\in V=J\cup S\cup D$,

$F1Val\in REAL$,$F2Val\in REAL$ 分别表示两个启发函数值;

$EDGE=NODE\times NODE$

(4)k 元设备分配请求集合表示为:

$DR\text{-}k=DR_1\times DR_2\times\cdots\times DR_k$

$DR_i=J\times(J+D)$

(5)k 元设备分配请求集合的解:

$Stack\text{-}k=Stack_1\times Stack_2\times\cdots\times Stack_k$

$Stack_i=NODE^*$

(6)设置一个栈(称作剥夺回溯栈)来存储剥夺了其他设备分配请求节点的请求编号:

$BStack=N^*$,$N\in[1,2,\cdots,k]$;

(7)设置 k 个队列(称作回溯点队列)分别用于记

录 k 个设备分配请求在搜索过程中的回溯节点:

$$BQueue\text{-}k = BQueue_1 \times BQueue_2 \times \cdots \times BQueue_k;$$

$$BQueue_i = NODE^*;$$

3.3 算法描述

定义 6 设备分配图 G 中未被任何设备分配请求占用的节点称为**空闲节点**.

定义 7 在设备分配过程中,需要对节点进行扩展,若节点的某个可扩展节点已被其他分配请求占用,则称该可扩展的节点为**冲突节点**.

算法具体描述见图 2、图 3 和图 4.

Algorithm 1:HDevA(G,DR-k)

输入:G 表示设备分配图;DR-k 表示 k 元设备分配请求集合.
输出:Stack-k 表示 k 元设备分配请求集合的解.

```
1.Stack-k←null;BStack←null;BQueue-k←null;
2.f1(G);f2(G);/*计算每个顶点的启发函数值*/
3.for each i do Stack[i]←left(DR[i]);end for
4.for each i do Search(G,Stack[i],DR[i]);end for
5.return(Stack-k);
```

图 2　启发式双重回溯剥夺静态设备分配算法 HDevA

Search(G,Stack[i],DR[i])

输入:G 表示设备分配图;Stack[i]表示第 i 个设备分配请求的解;DR[i]表示第 i 个设备分配请求.
输出:boolean 值表示求第 i 个解是否成功,1 表示成功找到解,0 表示无解.
说明:该函数求解第 i 个设备分配请求的解.变量 node_kind 标志得到的扩展节点的种类:值 1 表示该节点是空闲节点;值 2 表示该节点是冲突节点.变量 top 为堆栈节点元素类型,存储搜索栈的栈顶元素.变量 next 为可以扩展的所有节点的链表.

```
1.top←null;next←null;
2.if Stack[i]=null then return(0);
3.top←gettop(Stack[i])
4.if top≠right(DR[i]) then      //不是目标节点
5.   (next,node_kind)=GetNext(G,DR[i],top);//求邻接节点
6.   if next=null then top=pop(Stack[i]);push(BQueue[i],top);goto 2;
7.   else if node_kind=1 then push(Stack[i],next);goto 2;   //占用next 节点
8.   else if node_kind=2 then   //需要剥夺,next 为可供剥夺节点列表
9.      first←gethead(next);
10.     push(BStack,i);
11.     for each j do bs[j]←Stack[j];bb[j]←BQueue[j];end for
12.     if(∃x)first∈Stack[x] then do node=pop(Stack[x]);until node=first;endif   //剥夺请求 DR[x]的节点 first,x 是唯一的
13.     push(Stack[i],first);   //当前请求占用 first 节点
14.     bRet←Search(G,Stack[x],DR[x]);
          //从 first 前一个节点开始重新搜索被剥夺请求 DR[x]的解
15.     if bRet=0 then      //DR[x]搜索失败
16.        for each j do Stack[j]←bs[j];BQueue[j]←bb[j];end for
17.        next←delfirst(next,first);pop(BStack,i);//将 first 从 next 中删除
18.        if next!=null then goto 6;
19.        else top←pop(Stack[i]);push(BQueue[i],top);goto 2;end if
20.     else pop(BStack,i);goto 2;endif //DR[x]搜索成功,继续 DR[i]搜索
21.  end if
22.end if
23.top←gettop(Stack[i]);
24.if top=right(DR[i]) then return(1) else return(0) end if
```

图 3　设备分配函数 Search

GetNext(G,DR[i],head)

输入:G 表示设备分配图;DR[i]表示第 i 个设备分配请求集合;head 表示图 G 中的一个顶点.
输出:(next,kind),next 表示 head 节点的邻接点链表,kind 表示邻接点的种类.
说明:该函数用于在图 G 中得到节点 head 的可扩展节点.算法中用 I 表示图 G 中当前空闲节点集合,C 表示当前冲突节点集合.

```
1.for all ⟨head,node⟩∈E do
2.   if node∉Stack[i]&node∉BQueue[i] then list=list+node;end if
3.end for
4.f_free←MAX;next←NULL;idle_node←NULL;//f_free 表示启发函数值
5.for all node∈list do
6.   if node∈I & f(node)<f_free then idle_node←node;f_free←f(node);end if
7.end for   //函数 Idle 判断节点是否为空闲节点,f 求节点的启发函数值
8.if idle_node≠NULL then next←idle_node;return(next,1);//有可扩展的空闲节点
9.else if list≠NULL then next←Sort(list);return(next,2);
        //无空闲节点但有可扩展冲突节点,Sort 表示按照启发函数好坏降序排列
10.else return(NULL,_);   //无可扩展节点
11.end if
```

图 4　求邻接节点函数 GetNext

3.4 启发函数

算法中使用两个启发函数指导搜索过程的进行,分别如下定义:

定义 8 启发函数 $f1(node)=\sum W_i(1\leqslant i\leqslant k)$;其中 $W_i=1$ 如果从请求 DR_i 的源节点能够到达节点 $node$,否则 $W_i=0$.

定义 9 启发函数 $f2[i](node)$,节点 $node$ 到达请求 DR_i 的目标节点的最短路径长度.

启发函数 $f1(node)$的信息反映的是对于分配请求 $DR_1,\cdots,DR_k$ 的路径占用节点 $node$ 的可能性.值为 0 时表示节点 $node$ 不可能在任何一个分配请求的可能路径上,因而该节点也就不会被搜索过程扩展;节点的 $f1$ 值越大则节点被多个分配请求共享的几率越大,亦即在该节点冲突的可能性越大.搜索过程在扩展时将优先选择值小的节点,即优先选择冲突可能性小的节点.

启发函数 $f2[i](node)$的信息反映的是节点 $node$ 到达请求 DR_i 的目标节点的最短路径长度.在函数 $f1$ 的基础之上搜索过程将优先选择 $f2$ 值小的节点进行扩展,使不扩大冲突几率的情况下尽量得到优良的路径.

4 存在请求动态释放情况下算法的推广

算法 HDevA 认为每个设备分配请求对一条路径的占用是永久的,一个开关被某个设备分配请求占用后

就不能再被其他的请求使用直到整个测试程序执行完毕.事实上,任何一个分配请求都是在测试程序执行过程的某个时刻提出申请,在这个设备分配请求相关的测试动作执行完成之后就立即释放其所占用的路径以及设备,而其他在该请求释放之后才提出申请的分配请求可以使用前面所释放的资源.

另外,在现实的系统中,我们总是倾向于使用最少的资源来完成尽量多的测试任务,只要测试任务在时间上不存在冲突,其相应的分配请求就可以占用相同的节点.

综上所述,设备分配请求的释放实际上是一个动态的过程,因此需要在完成静态设备分配过程的算法 HDevA 基础上模拟出动态的效果.

4.1 设备分配请求的时间性质

算法 HDevA 中设备分配请求 DR 是图 G 中节点的有序对$\langle s,d\rangle$,代表图 G 中一条从节点 s 到节点 d 的一条连通路径,下面给出其新的定义.

定义 10 设备分配请求 DR 设 $G=\langle V,E\rangle$为一个设备分配图,$V=J\cup S\cup D$.一个设备分配请求 DR 是 G 中一个有序顶点对$\langle a,b,t_start,t_end\rangle$,$a\in J$,$b\in J\cup D$,$t\_start$ 和 t_end 分别代表该分配请求占用和释放链路的相对时刻(相对于测试程序开始执行时刻).此时,称 a 为 DR 的源节点,称 b 为 DR 的目标节点.

在 t_start 之前该分配请求没有提出资源占用申请;在 t_end 之后该分配请求释放所占用的资源以供其他请求使用.由于时间是单一连续的,两个分配请求的(t_start_i,t_end_i)、(t_start_j,t_end_j)在时间轴上是否存在交集就能反映出这两个请求在时间上冲突与否,如图 5 所示:

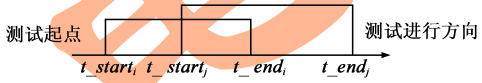

图 5 设备分配请求时间性质

定义 11 k 元设备分配请求集合 DR-k 设 $G=\langle V,E\rangle$为一个设备分配图,$V=J\cup S\cup D$.k 元设备分配请求集合就是由 k 个设备分配请求组成的集合$\{DR_1,\cdots,DR_k\}$($DR_i=\langle a_i,b_i\rangle$,$i=1,\cdots,k$),并且要求对于任意 i 和 j 有 a_i,a_j,b_i,b_j 互不相等.k 元设备分配请求集合可记为 DR-k.

定义 12 设备分配请求 DR 的生命周期是从 t_start 到 t_end 这段时间.

通过如上定义,存在请求动态申请与释放的设备分配可以抽象为给定 k 元设备分配请求集合 DR-k,在设备分配图 G 中求 k 个设备分配请求解的集合 $P=\{p_i\mid p_i$ 是分配请求$\langle s_i,d_i,t_start_j,t_end_j\rangle$在图 G 中的连通路径 $1\leqslant i\leqslant k$,且对任意 i,j,若 DR_i 和 DR_j 的生命周期有交集则要求 p_i 和 p_j 无公共顶点$\}$;

4.2 对算法 HDevA 的修改

定义 13 动态空闲节点:不被其他请求占用的节点或者被其他请求占用但是占用该节点的所有请求和当前请求的生命周期不冲突的节点.

定义 14 动态冲突节点:如果一个节点是冲突节点并且占用该节点的请求和当前请求的生命周期有交集,则称该节点为动态冲突节点.

修改主要在函数 Search 和函数 GetNext 中完成,分别见图 6 和图 7.

```
Search(G,Stack[i],DR[i])
1.top←null;next←null;
2.if Stack[i] = null return(0);
3.top←gettop(Stack[i])
4.if top≠right(DR[i]) then   //不是目标节点
5.   (next,node_kind) = GetNext(G,DR[i],top);
6.   if next = null then top = pop(Stack[i]);push(BQueue[i],top);goto
     2;
7.   else if node_kind = 1 then push(Stack[i],next);goto 2;   //占用
     next 节点
8.   else if node_kind = 2 then      //需要剥夺
9.      first←gethead(next);
10.     push(BStack,i);
11.     for each j do bs[j]←Stack[j];bb[j]←BQueue[j];end for
12.     if(∀x)first∈Stack[x] then do node = pop(Stack[x]);until node
        = first;endif
           /*剥夺请求 DR[x]的节点 first,可能有多个 x,设为集合
           P={m,⋯,n}*/
13.     push(Stack[i],first);//当前请求占用 first 节点
14.     for all x∈P do
15.       bRet←Search(G,Stack[x],DR[x]);//搜索被剥夺请求 DR
          [x]的解
16.       if bRet = 0 then break;
17.     end for
18.     if bRet = 0 then      //(∃x)DR[x]搜索失败
19.       for each j do Stack[j]←bs[j];BQueue[j]←bb[j];end for
20.       next←delfirst(next,first);pop(BStack,i);
21.       if next! = null then goto 6;
22.       else top←pop(Stack[i]);push(BQueue[i],top);goto 2;end if
23.     else pop(BStack,i);goto 2;end if   //(∀x)DR[x]搜索成功,
        继续 DR[i]搜索
24.  end if
25.end if
26.top←gettop(Stack[i]);
27.if top = right(DR[i]) then return(1) else return(0) end if
```

图 6 改进的设备分配函数 Search

```
GetNext(G,DR[i],head)
说明:算法中用 DI 表示图 G 中当前动态空闲节点集合,DC 表示当
     前动态冲突节点集合.
```

```
1. for all 〈head,node〉∈E do      //node 为求邻接节点
2.    if node∈Stack[i] ‖ node∈BQueue[i] then continue;
3.    else if node∈I & node∉DI[i]-I then
4.       IdleList = IdelList + node;   //建立空闲节点列表
5.    else if node∈DI[i]-I & node∉I then
6.       DIdleList = DIdleList + node;   //建立生命周期不冲突的非空
         闲节点
7.    else ConflictList = ConflictLlist + node;//建立动态冲突节点列表
8.    end if
9. end for
10. f_free←MAX;next←NULL;   //f_free 表示启发函数值
11. for all node∈DIdleList do   //在 DIdleList 中查找启发函数值最小
    的节点
12. if f(node) < f_free then
13.       next←node;f_free←f(node);
14.    end if
15. end for
16. for all node∈IdleList do   //在 Idle 中查找启发函数值最小的节点
17.    if f(node) < f_free then
18.       next←node;f_free←f(node);
19.    end if
20. end for
21. if next≠NULL then   //存在启发函数值最小的非动态冲突节点
    next
22.    return(next,1);
23. else if ConflictList≠NULL then   //不存在非动态冲突节点
24.    next←Sort(ConflictList);return(next,2);   //返回冲突节点列表
                //Sort 表示按照启发函数好坏降序排列
25. else    return(NULL,_);
26. end if
```

图 7　改进的求邻接节点函数 GetNext

4.3　启发函数的改进

启发式双重回溯搜索算法的关键在于启发函数的好坏,因而这里考虑改进启发函数的方法.按照定义 8 所定义的启发函数,对于图 8 将使所有 8 个点的 $f1$ 值都是 2,显然这是不精确的.考虑修改 $f1$ 的定义如下:

定义 15　启发函数 $f1(node)=\sum W_i$ $(1\leqslant i\leqslant k)$;其中 $W_i=1$ 如果 $node$ 在请求 DR_i 的某条连通路径上,否则取 $W_i=0$.

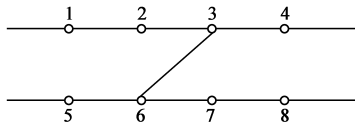

图 8　一个特殊的设备分配图

这样定义之后,图 8 的所有 8 个点的 $f1$ 的值都将是 1,显然更加精确.

5　实例分析

定义 16　渗透度 = L/N,其中 L 为一个设备分配请求的解中节点的个数,N 为求解这个设备分配请求时扩展的节点个数.

渗透度越高说明搜索过程朝着目标方向进行的程度越大,同时说明搜索树的宽度很小.在无限图搜索中如果渗透度很大,一旦搜索偏离了正确方向,则算法很难找到正确解或终止.因此渗透度反映了启发性信息是否能够很好地指导搜索朝着正确的目标进行.

本文针对 6 个实际 ATLAS 程序(中国海军航空工程飞行学院飞行仪器测试程序),把启发式双重回溯搜索算法与传统的穷举算法进行比较,实验结果列于表 1.

表 1　两种算法运行结果比较

		实例 1 (942,2055)	实例 2 (310,651)	实例 3 (292,639)	实例 4 (855,2014)	实例 5 (282,629)	实例 6 (327,1183)
	请求数	36	31	68	56	68	16
穷举算法	搜索扩展节点数	38801 (237)	30166 (112)	71861 (269)	138781 (392)	71661 (261)	24381 (56)
	平均扩展节点数	1078	973	1057	2478	1053	1523
	渗透度	0.006	0.003	0.004	0.003	0.004	0.002
启发式算法	启发函数开销	3444	569	4600	14751	4590	1598
	搜索扩展节点数	237 (237)	120 (112)	269 (269)	392 (392)	262 (261)	1586 (56)
	总扩展节点数	3681	689	4869	15143	4852	3184
	平均扩展节点数	102	22	72	270	71	199
	渗透度	1	0.933	1	1	0.996	0.035

注 1:每个实例括号内的数字分别表示设备分配图中节点和边的个数.

注 2:扩展节点数目括号内的数字表示所求解中节点的个数.

注 3:实例 6 中分配请求数为 16,但是只有 14 个分配请求能被满足,程序最后打印出能被满足的分配请求的路径并指出其中两个分配请求不能满足.

从表 1 可以看出启发式双重回溯搜索算法比穷举算法极大地减少了扩展节点的数目,从而提高了搜索路径的效率.存在解的情况下(如表 1 中的实例 1、2、3、4、5),启发式双重回溯搜索算法的渗透度较高,说明算法能够沿着正确的搜索方向前进;不存在解时(如表 1 中实例 6),算法的渗透度较低,但是比穷举算法要高很多.

另外,文献[14]分析了贪心算法在完全图中的近似性能,文献[15]研究包括完全图在内的几类特殊图中的不相交路径问题.这些文献的研究思路基本是对图或者请求数目进行限定或者简化,从而提高算法的效率;本文提出的算法对任意的分配图和请求,能够在启发函数指导下较快地找到解.

6　总结

本文针对 ATLAS 语言编译器实现中静态设备分配问题,提出了一种启发式双重回溯搜索算法;给出了两

个实际应用的启发函数定义,并讨论了启发函数的改进策略.本文还针对实际应用的 ATLAS 程序中的设备分配请求例子,给出了应用效果分析,通过这些实例表明,对任意给定的设备分配图 G 和 k 元设备分配请求 DR-k,该算法能够很快找到解或者没有解而退出.本文提出的算法已经在 ATLAS 语言编译器中实现,较好地解决了 ATLAS 程序中的设备分配请求问题.

参考文献:

[1] IEEE Std 771-1998, IEEE Guide to The Use of The ATLAS Specification[S].

[2] IEEE Std 716-1995, IEEE Standard Test Language for All Systems- Common/Abbreviated Test Language for All Systems (C/ATLAS)[S].

[3] IEEE SCC-20-1997, ATLAS-2000 Test Language Requirements Document[S].

[4] M R Garey, D S Johnson. Computers and Intractibility: A Guide to the Theory of NP-Completeness[M]. W H Freeman & Co New York, 1979.

[5] N Robertson, P D Seymour. Graph minors-XIII: The disjoint paths problem[J]. Journal of Combinatorial Theory B, 1995, 63(1): 65 – 110.

[6] Y Shiloach. A polynomial solution in the undirected two paths problem[J]. Journal of the ACM, 1980, 27(3): 445 – 456.

[7] D S Hochbaum. An exact sublinear algorithm for the max flow, vertex-disjoint paths and communication problems on random graphs[J]. Operations Research, 1992, 40(5): 923 – 935.

[8] A Z Broder, A M Frieze, S Suen, E Upfal. An efficient algorithm for the vertex-disjoint paths problem in random graphs [A]. Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms[C]. Society for Industrial and Applied Mathematics Philadelphia, 1996. 261 – 268.

[9] E Shamir, E Upfal. A parallel fast construction of disjoint paths in networks[A]. Foundations of computation theory[C]. New York: Elsevier North-Holland, 1985. 141 – 153.

[10] Stavros G Kolliopoulos, Clifford Stein. Approximating disjoint-path problems using packing integer programs[J]. Mathematical Programming, Springer Berlin/Heidelberg, 2004, 99(1): 63 – 87.

[11] C Chekuri, S Khanna, F B Shepherd. Edge-disjoint paths in planar graphs[A]. 45th Annual IEEE Symposium on Foundations of Computer Science [C]. IEEE Computer Society, 2004. 71 – 80.

[12] Matthew Andrews, Lisa Zhang. Hardness of the undirected edge-disjoint paths problem[A]. Proceedings of the thirty-seventh annual ACM symposium on Theory of computing [C]. Baltimore, MD, USA, 2005. 276 – 283.

[13] 陈跃泉,郭晓峰,曾庆凯,陈贵海. AMR:一个基于网络最大流的 Ad-Hoc 多路径路由算法[J]. 电子学报, 2004, 32(8): 1297 – 1301.
Chen Yue-quan, Guo Xiao-feng, Zeng Qing-kai, Chen Gui-hai. AMR: A multipath routing algorithm based on maximum flow in ad-hoc networks[J]. Acta Electronica Sinica, 2004, 32(8): 1297 – 1301. (in Chinese)

[14] P Carmi, T Erlebach, Y Okamoto. Greedy Edge-disjoint Paths in Complete Graphs[R]. ETH Zurich: TIK-Report155, Computer Engineering and Networks Laboratory (TIK), 2003. 143 – 155.

[15] Thomas Erlebach, Danica Vukadinovic. New results for paths problems in generalized stars, complete graphs, and brickWall graphs[J]. Discrete Applied Mathematics, 2006, 154(4): 673 – 683.

作者简介:

郭德贵 男,1972 年生于山东省诸城市,博士,研究领域为编译方法与技术、软件形式化方法、语义网. E-mail: guodg@jlu.edu.cn

刘 磊 男,1960 年生于吉林省白城市,教授/博士生导师,研究领域为编译方法与技术、软件形式化方法、语义网. E-mail: liulei@jlu.edu.cn