石油传输网络最少增压器问题的回溯与分支限界解法*)

毕迎春　王相海

(辽宁师范大学计算机与信息技术学院　大连116029)　(信息安全国家重点实验室　北京100039)

摘　要　回溯法和分支限界法是用于解决诸多问题的重要而有效的方法。本文首先提出石油传输网络中的最少增压器问题,然后介绍了基于回溯法和分支限界法的两种有效算法,最后对这两种算法进行了比较和讨论。实验结果验证了算法的有效性。

关键词　回溯法,分支限界法,石油传输网络,复杂度

The Backtracking Algorithm and the Branch-and-Bound Algorithm for the Least Supercharger Problem of Petroleum Transmission Network

BI Ying-Chun　WANG Xiang-Hai

(School of Computer and Information Technology, Liaoning Normol University, Shenyang 116029)

(State Key Lab of Information Security, Beijing 100039)

Abstract　Backtraking algorithm and branch-and-bound algorithm are important and efficient methods to many problems. In this paper, the least supercharger problem of petroleum transmission network is brought up firstly. And then this paper presents two efficient algorithms based on backtracking algorithm and branch-and-bound algorithm. Finally, backtraking algorithm and branch-and-bound algorithm are compared and discussed. Simulation results show they are effective.

Keywords　Backtraking algorithm, Branch and bound algorithm, Petroleum transmission network, Complexity

1　引言

随着石油在人们日常生活中的广泛的应用,石油公司需要通过管道输送大量的石油。目前,中国油气管道正呈现出蓬勃发展的势头,已成为我国第五大运输业[1]。而在石油传输网络的设计中通常会遇到最少增压器问题。本文首先对石油传输网络最少增压器的问题进行了说明,进而给出了解决这一问题的回溯解法和分支限界解法,并对所提出算法的时间复杂度进行了分析和讨论,最后对两种算法进行了比较。

2　问题的提出

所谓石油传输网络是指石油公司以本公司为油源通过管道将石油输送到其他多个城市石油公司的网络结构。在该网络结构中,各个石油公司为网络的结点。网络中的油压随距离增大而减小,为了保证整个输油网络正常工作,需要维持网络中的最低油压 P_{min}。为此需要在网络的某些或全部顶点处设置增压器,在设置增压器的顶点处油压可升至最大值 P_{max}。油压从 P_{max} 减至 P_{min} 可使石油传输的距离至少为 d。该石油公司要设计一个增压器的最优放置方案,使得用最少的增压器保证石油运输畅通。本文将该问题简称为石油传输网络最少增压器问题。

3　问题的回溯解法

回溯法是基本的算法策略之一,它是在问题的解空间中,按照深度优先的策略从根结点出发搜索解空间树,同时根据实际问题确定相应的剪枝函数,在搜索最优解的过程中根据剪枝函数对解空间进行剪枝,以提高搜索速度[2~4]。

3.1　问题的解空间

对于具有 n 个结点的石油传输网络,其解空间由长度为 n、元素为0或1的向量组成,每一个向量表示问题的一个可能解,其中元素0和1分别表示在相应的结点处不设和设置增压器,我们将石油传输网络最少增压器问题的解空间用子集树表示(图1给出了当 n 为3时的解空间子集树)。

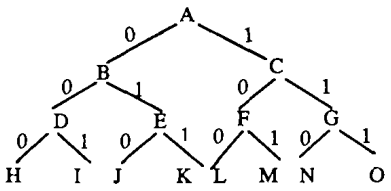

图1　问题的解空间

3.2　问题的回溯算法设计

(1) *存储结构的确定*　我们将石油传输网络表示为一个非循环带权有向图。图中每条边上的权表示该边连接两个城市(以下我们用顶点表示)间管道的距离。我们采用邻接表来存储一非循环带权有向图,具体存储结构定义如下:

```
typedef struct node
{ int adjvex; //邻接点域
  int weights; //权值
  struct node *next; //链域
}edgenode; //边表结点
typedef struct
```

*)国家自然科学基金(60372071)、辽宁省自然科学基金(20032105)和大连市科技基金项目资助。毕迎春　硕士研究生,研究方向为算法分析与设计及多媒体信息处理。王相海　博士,教授,主要研究领域为CG、CAGD、多媒体信息处理及算法与设计。

```
{ int vertex; //顶点信息
  int id; //入度
  struct node *link; //边表头指针
}vexnode; //顶点表结点
vexnode ga[n]; //全程量邻接表
int dig[n]; //记载顶点的拓扑序列号
```

(2) *确定拓扑序列* 因为各顶点间存在着先后关系，为计算各顶点处的油压值，需建立顶点间的拓扑序列。约定将得到的拓扑序列(对应顶点在 ga 中的下标值)存入数组 *dig*[*n*]中。

实现拓扑排序的算法如下：

```
void toposort( ) //拓扑排序
{ int i,j,k,t=0,m=0,top=-1;
edgenode *p;
  for(i=0;i<n;i++)
//建立入度为0的顶点链栈
  if (ga[i].id==0)
    { ga[i].id=top;
      top=i;
    }
  while(top!=-1) //栈非空
{ j=top;
top=ga[top].id; //第j+1个顶点退栈
dig[t++]=j; //退栈顶点存入数组 dig 中
m++; //为输出顶点计数
p=ga[j].link; //p 指向 vj+1的出边表结点指针
while(p) //删去所有以 vj+1为起点的出边表
    {k=p->adjvex;
     ga[k].id--; //vk+1入度减1
     if(ga[k].id==0) //将新入度为0的顶点入栈
      { ga[k].id=top; top=k; }
     p=p->next; //找 vj+1的下一条边
    }
  }
if(m<n) //输出顶点数小于 n，有回路存在
  { tag=1;
    printf("\nthe network has a cycle\n");
  }
}
```

(3) *剪枝函数的确定* 约定在子集树的第 *j*+1层的结点 *z* 处，*num* 记录当前的增压器数，*best* 记录当前已得到的最少增压器数，则在该顶点处将会有如下剪枝情况发生：

① 当 *num*>*best* 时，以结点 *z* 为根的子树中所有结点都不满足约束条件，因而该子树中的解均为不可行解，故可将该子树剪去；

② 如果所对应的第 *j* 个待处理顶点处的油压初值不小于 P_{max}，则该顶点处不用设置增压器，不必考虑以 *z* 为根的右子树，故可将其右子树剪去；

③ 如果所对应的第 *j* 个待处理的顶点处的油压初值小于 P_{min}，则该顶点处必须设置增压器，不必考虑以 *z* 为根的左子树，可将其左子树剪去；

④ 如果所对应的第 *j* 个待处理的顶点沿管道将油输送至与其相邻的其他顶点时油压小于 P_{min}，则该顶点处必须设置增压器，不必考虑以 *z* 为根的左子树，可将其左子树剪去。

(4) *确定 backtrack 函数* *backtrack*(*i*,*num*)的功能是搜索子集树中第 *i*+1层子树，在主函数中调用以 *backtrack*(0,0)形式实现对整个解空间的回溯搜索。具体过程如下：

① 当 *i*>*n*−1时，已得到一个解(因为最后一个顶点必不用设置增压器，故只需判断 *n*−1个结点即可)，若此时获得的增压器数 *num* 小于当前最优解 *bestx*[*n*]所对应的增压器数 *best*，则表示当前解优于迄今所找到的最优解，故需更新 *best* 和 *bestx*[*n*]；

② 当 *i*<*n*−1时，当前扩展结点 *z* 是子集树中的一个内部结点，该结点有 *x*[*i*]=1和 *x*[*i*]=0两个分支，其左儿子分支表示 *x*[*i*]=0的情形，这样仅当从第 *i*+1个顶点将油输送到其他各顶点时油压都大于等于 P_{min}时进入左子树，递归地对左子树进行搜索；其右儿子结点表示 *x*[*i*]=1的情形，由于可行结点的右儿子结点总是可行的，故进入右子树时不需要检查可行性。

实现 backtrack 的具体算法如下：

```
void backtrack(int i,int num)
{ int j,k,v,flag=0;
  edgenode *p;
  if(i>=n-1) //已搜索到一个解
  { if (num<best) //更新最优值及最优解
      { best=num;
        for(k=0;k<n;k++) bestx[k]=x[k];
      }
    return;
  }
  //搜索子树
  v=dig[i]; yy[v]=0;
  if (i==0&&(s<Pmin||s>=Pmax)) //剪枝回溯
    { yy[v]=Pmax;
      if (s<Pmin)
        { x[0]=1; backtrack(1,1);} //剪去左子树
      if(s>=Pmax)
            { x[0]=0; backtrack(1,0);} //剪去右子树
}
else
  { if(i==0) yy[v]=s;
    else
    for(k=0;k<i;k++) //求顶点 ga[v]处初始油压值
     { p=ga[dig[k]].link;
       while(p)
       {if(p->adjvex==v)
        yy[v]=yy[v]+yy[dig[k]]-ave*p->weights;
        p=p->next;
        }
     }
    for(j=0;j<2;j++) //x[i]的可选值仅为0和1
      { if(yy[v]>=Pmax) //剪枝回溯
         { yy[v]=Pmax; x[i]=0;
           backtrack(i+1,num);//剪去右子树
           break;
         }
        if(yy[v]<=Pmin)//剪枝回溯
        { yy[v]=Pmax; x[i]=1;num++;
          if(num>=best) return;
            else
             { backtrack(i+1,num);//剪去左子树
               break;
             }
        }
       if(j==0) //检查将油传至相邻各顶点时油压
                //是否小于 Pmin
         { p=ga[v].link;
           while(p)
           if(yy[v]-ave*p->weights<Pmin)
                { flag=1;break; } //剪枝回溯
           else p=p->next;
         }
      }
  if (flag==1||j==1) //此时 x[i]必为1
    { x[i]=1;num++; yy[v]=Pmax;
      if(num>=best) return;//剪枝回溯
      else { backtrack(i+1,num);break;}
     }
    else
    { x[i]=0; backtrack(i+1,num);}//进入左子树
    }
  }
}
```

3.3 时间复杂度分析

对于有 *n* 个顶点和 *e* 条边的石油传输网络，建立网络的邻接表的算法 creatdjlist 所需要的时间复杂度为 $O(n+e)$，拓扑排序 toposort 所需要的时间复杂度也为 $O(n+e)$，函数 backtrack 在每个结点处时间主要花费在求第 *i* 个待处理顶点未经过处理前的油压值 yy[dig[i]]的语句上，最坏情况下该语句需要执行 $O(2(n-1)$的时间，此时共有 $2n-1$个结点需要搜索，故 backtrack 所需的时间为 $O(2(n-1)*(2n-1)$，算法最坏情况下的时间复杂度为 $O(n^2)$。

4 问题的分支限界解法

与回溯法相比，分支限界法也采用子集树来表示问题的解空间，但每一个活结点只有一次机会成为扩展结点，活结点

一旦成为扩展结点，就一次性产生所有儿子结点，并把那些导致不可行解或非最优解的儿子结点舍弃，其余儿子结点加入到活结点表中，此后，从活结点表中取下一结点成为当前扩展结点，这个过程一直持续到找到所需的解或活结点表为空时为止。从活结点表中选择下一扩展结点的不同方式导致了不同的分支限界法，最常见的有队列式(FIFO)分支限界法和优先队列(PQ)式分支限界法两种[2,5]。以下我们采用优先队列式分支限界法来解决石油传输网络最少增压器问题。

4.1 问题的优先队列式分支限界解法

在问题的优先队列式分支限界算法中，问题的解空间、拓扑排序和剪枝函数与第3节中基本相同，选用了最小优先队列来表示解空间的活结点表。

(1) *存储结构的确定* 石油传输网络的存储结构也采用与3.2相类似的邻接表，此外还定义了子集树的元素类型和最小堆的元素类型：

```
struct bnode //子集树的元素类型
{ struct bnode *parent; //指向父结点的指针
  int lchild; //左儿子结点标志
  int yy; //该结点处的油压值
};
struct bnode *E; //初始扩展结点
struct heap //最小堆的元素类型
{ int level; //活结点在子集树中所处的层序号
  int num; //活结点优先级
  struct heap *next; //链域
  struct bnode *ptr; //指向活结点在子集树相应结//点的指针
};
struct heap *head; //最小堆的头结点
```

(2) *确定优先级* 活结点 x 在优先队列中的优先级定义为从根结点到结点 x 的路径所相应的增压器数 num，优先队列中活结点的优先级为 $x.num$。

(3) *确定 Addlivenode 函数* 函数 *Addlivenode* 的功能是将一个新产生的活结点以结点元素类型 bnode 的形式加入到子集树中，并将这个新结点以结点元素类型 heap 的形式插入到表示活结点优先队列的最小堆中。具体算法如下：

```
void addlivenode(struct bnode *E,int lev,
                 int ch, int yy,int num)
{ struct bnode *b;
  struct heap *s,*p,*qq;
  s=malloc(sizeof(struct heap));
  b=malloc(sizeof(struct bnode));
            //将新产生的结点 b 加入到子集树中
  b->parent=E;
  b->lchild=ch;
  b->yy=yy;
      //将新产生的结点 s 插入到优先队列的最小堆中
  s->level=lev;
  s->num=num;
  s->ptr=b;
  if(head->next==NULL)
  { head->next=s; s->next=NULL; }
    else //按优先值 num 的值由小到大的顺序将 s
         //插入到队列最小堆中
      { p=head->next;qq=head;
        while(p->num<=s->num)
          { qq=p;p=p->next;
            if(p==NULL)break;
          }
        s->next=qq->next;
        qq->next=s;
      }
}
```

(4) *确定 fenzhi 函数* 函数 *fenzhi* 具体实施对解空间的优先队列式分支限界搜索。算法以 $i=0$ 和 $sum=0$ 开始，子集树的根结点是扩展结点。具体过程如下：

① 如果当前扩展结点的左儿子结点是可行结点，即从它所相应的顶点输送至其他各顶点时油压都大于等于 P_{min}，则将它加入到子集树的第 $i+1$ 层上，并插入最小堆；

② 扩展结点的右儿子结点必是可行的，故直接插入子集树和最小堆中；

③ 算法从最小堆中取出最小元作为下一扩展结点，如果该扩展结点的位于第 $n-1$ 层，则算法终止，它相应的可行解为最优解，否则重复前面的过程。

具体算法如下：

```
void fenzhi()
{ struct heap *H;
  struct bnode *ee;
  edgenode *p;
  int i=0,j,k,sum=0,y,flag=0;
  while(i<n-1) //搜索子集空间树
  { y=0; //y 记录当前扩展结点处的油压值
    if(i==0&&s<Pmin) //剪去左子树
      addlivenode(E,1,1,Pmax,1);
    else
    if(i==0&&s>=Pmax) //剪去右子树
      addlivenode(E,1,0,Pmax,0);
    else
      { if(i==0) y=s;
        else //求当前扩展结点处的油压初值
        { for(k=0;k<i;k++)
          { p=ga[dig[k]].link;
            while(p)
            { if(p->adjvex==dig[i])
              { ee=E;
                for(j=i-1;j>k;j--)
                ee=ee->parent;
                s=ee->yy;
                y=y+s-ave*p->weights;
              }
              p=p->next;
            }
          }
        }
          //检查当前扩展结点的儿子结点
      if (y>=Pmax) //剪去右子树
          addlivenode(E,i+1,0,Pmax,sum);
        else
        { p=ga[dig[i]].link;
          //检查当前扩展结点将油传到与其相邻的结
          //点时油压值是否小于 Pmin
            while(p)
              { if (y-ave*p->weights<Pmin)
                  { flag=1;break; }
                else p=p->next;
              }
          if (flag==0) //左右儿子结点均可行
            { addlivenode(E,i+1,0,y,sum);
              addlivenode(E,i+1,1,Pmax,sum+1);
            }
          else//剪去左子树
            addlivenode(E,i+1,1,Pmax,sum+1);
        }
      }
      H=head->next; //取下一扩展结点
      if (H==NULL) return;
        else
        { i=H->level;
          sum=H->num;
          E=H->ptr;
          flag=0;
          head->next=head->next->next;
        }
      }
}
```

(5) *构造最优解*

为了构造最优解，在算法的搜索进程中已经保存了当前已构造出的部分解空间树，这样在算法确定了达到最优值的叶结点时，就可以在解空间树中从该叶结点开始向根结点回溯，构造出相应的最优解。具体算法如下：

```
for(j=n-2;j>=0;j--)
//最后一个需处理的顶点处必不用设置增压器，
//故 j 从 n-2开始，E 为最后一个扩展结点
{ bestx[j]=E->lchild;
  E=E->parent;
}
```

4.2 时间复杂度分析

对于有 n 个顶点和 e 条边的石油传输网络，建立网络的邻接表的算法 creatdjlist 所需要的时间复杂度为 $O(n+e)$，拓扑排序 toposort 所需要的时间复杂度也为 $O(n+e)$。算法的时间主要花费函数 *fenzhi* 的 while 循环体的语句中，该语句

在最坏情况下的计算时间为 $O(3(n-1)$，而 while 循环体在最坏情况下要执行 $2n-1$ 次，故最坏情况下函数 *fenzhi* 所需的时间复杂度为 $O(3(n-1)*(2n-1))$，故算法最坏情况下的时间复杂度为 $O(n^2)$。

5 两种算法的简单比较

(1) 尽管上面两种算法的最坏时间复杂度均为 $O(n^2)$，但回溯算法在搜索过程中即使找到了最优解也不能马上确定它就是最优的，只有把它所有的可行解全部求出后才能确定最后的最优解，而分支限界法则只要搜索到一个解，这个解必为最优的，所以总体而言采用分支限界法将会比回溯法检查更少的结点，算法的平均时间要比回溯法节省；

(2) 上述两种算法中，回溯法占用的内存是解空间的最大路径长度个单元，而分支限界法所占用的内存为整个解空间的大小，对于一个子集空间，回溯法通常需要 $O(n)$ 的内存空间，而分支限界法则需要 $O(2^n)$ 的空间；对于排列空间，回溯需要 $O(n)$ 的内存空间，分支限界法界需要 $O(n!)$ 的空间。

6 实验结果

当 $P_{max}=200$，$P_{min}=80$，$d=60$ 时，我们针对图2所示的网络图对两种算法进行了实验，输出结果如图3所示，其中 a 图是建立存储结构的输入界面，b 图是算法的输出结果。由于采用了相同的输出界面，所以两种算法的输出结果是相同的。

结束语 本文针对石油传输网络最少增压器问题分别采用了回溯法和分支限界法进行了算法设计，并对这两种算法的时间和空间复杂度进行了分析和比较，实验结果验证了算法的有效性。

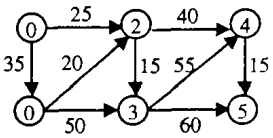

图2 石油传输网络图示

```
请输入6个顶点                    请输入源点 S 处的油压值
0 1 2 3 4 5                      s=120
请输入9条边——i,j,w               结果为
0 1 35                           best×[0]=1
0 2 25                           best×[1]=1
1 2 20                           best×[2]=0
1 3 50                           best×[3]=0
2 3 15                           best×[4]=0
2 4 40                           best×[5]=0
3 4 55
3 5 60
4 5 10
```

a　　b

图3 算法试验结果

参 考 文 献

1 苏士峰. 中国油气管道在创新中发展. http://www.gdb.net.cn/share.asp,2004

2 王晓东. 计算机算法设计与分析. 北京：电子工业出版社，2001

3 王岩冰，郑明春，刘弘. 回溯算法的形式模型. 计算机研究与发展，2001，38(9)：1066～1079

4 backtracking. http://www.ibrtses.com/delphi/backtracking.html,2003

5 分支限界问题分析. http://www.frontfree.net/articles/services/view.asp?id=669&page=1,2002,11

(上接第205页)

价标准集是整个评价工作的基础和关键。(2)部分工作[3,4]给出了一个内容庞杂、条目众多的标准集，却没有提供一种有效的、结构化方式来组织这些标准集，因而显得结构混乱而内容不清晰。(3)多数工作[3,4]所提出的评价标准只是一个抽象的概念，而不是一种具体机制，很难对是否符合进行判断，因而可操作性差，不易使用。

与已有工作相比较，本文提出的 SEF 具有多视角、易于操作、可扩展以及独立于应用等特点，能够更好地用于描述、分析和展示 MASDE 的基本特征，发现其优势和不足。SEF 由多个组构成，一个组代表了一个不同的视角，因此是多视角的；每个评价条目是小粒度的、底层的、具体的，因而是易于操作的；每个组的评价条目分为最小集和扩展集，扩展集可以根据需要进行扩展，因而是可扩展的；最后，所有评价条目的选择都是独立于具体应用的。综合来说，SEF 是一个完整、有效的 MASDE 的结构化评价框架。

基于 SEF，本文进一步对一组有代表性的 MASDE 进行了深入的剖析，得到了一系列重要的评价细节和结果，并具有以下特点：(1)覆盖面广，本文总共选择了十个典型的 MASDE 进行评价，基本上覆盖了各种类型的 MASDE。(2)具有分类评价结果和总体评价结果。

参 考 文 献

1 Schoepke S H. A Business View Regrading the Selection Of Agent Development Tookits. In: Proc. of the AAAI-98 Workshop on Software Tools for Developing Agents,1998

2 Ricordel P M,Demazeau Y. From analysis to deployment: a multi-agent platform survey. In:Proc. of the Workshop on Engineering Societies in the Agents' World,Springer-Verlag,2000

3 Eiter T, Mascardi V. Comparing Environments for Developing Software Agents, AI Communications,2002,15(4)

4 Garneau T,Delisle S. A New General, Flexible and Java-Based Software Development Tool for Multiagent Systems. ISE 2003, 2003

5 FIPA. FIPA Abstract Architecture Specification (FIPA0001). Available at http://www.fipa.org,2000

6 FIPA. FIPA Agent Management Specification (FIPA00023). Available at http://www.fipa.org, 2000

7 OMG. OMG Mobile Agent System Interoperability Facility (MASIF). Available at http://www.omg.org,1997

8 Tryllian. The Developer's Guide(ADK 2.1 version). Available at http://www.tryllian.com,2002

9 Regular Systems. AgentBuilder User's Guide Version 1.3. available at http://www.agentbuilder.com, 2000

10 Collier R W. Agent Factory: A Framework for the Engineering of Agent-Oriented Applications:[Ph. D. Thesis]. University College Dublin, Ireland, 2001

11 DeLoach S A. Analysis and Design using MaSE and agentTool. In:12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001), April 2001

12 Graham J R,Decker K S,Mersic M. DECAF - A Flexible Multi Agent System Architecture. Appearing in Autonomous Agents and Multi-Agent Systems,2003

13 Baumer C,Breugst M,Choy S,Magedanz T. Grasshopper--a universal agent platform based on OMG MASIF and FIPA standards. In:First Intl. Workshop on Mobile Agents for Telecommunication Applications (MATA'99), Oct. 1999

14 Busetta P,Ronnquist R, Hodgson A, Lucas A. Jack Intelligent Agents - Components for Intelligent Agents in Java. AgentLink News Letter, Janvier 1999

15 Bellifemine F, Poggi A, Rimassa G. JADE-A FIPA2000 Compliant Agent Development Environment. AGENTS'01,2001

16 Ferber J,Gutknecht O. A meta-model for the analysis and design of organizations in multiagent systems. In: Third Intl. Conf. on Multi-Agent Systems (ICMAS '98), IEEE Computer Society, 1998

17 Nwana, Ndumu,Lee,Collis. ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. Applied Artifical Intelligence Journal, 1999