فصل پنجم

بازگشت به عقب

مقدمه

روش بازگشت به عقب (پسگرد، عقبگرد، backtrack) در مسائل تصمیم گیری استفاده می‌شود. این روش در کاهش مرتبه زمانی تاثیر مثبت نمی‌گذارد، اما با کاهش حالت‌های قابل بررسی سعی می‌کند زمان اجرای الگوریتم را برای n کوچک، قابل قبول کند. برخی از ویژگی‌های مسائلی که به روش پسگرد حل می‌شوند و ویژگی‌های خود روش عبارتند از:

۱) مرتبه زمانی نامعقول. اکثر مسائلی که به روش بازگشت به عقب حل می‌شوند ذاتاً مسائل سختی هستند و مرتبه زمانی نامعقولی (2^n، $n!$ و 3^n و ...) دارند. روش بازگشت به عقب نیز از کاهش مرتبه زمانی آنها و معقول (چند جمله‌ای) کردن آنها عاجز است، اما با کاهش حالت‌ها تنها زمان اجرا را برای n کوچک، کاهش می‌دهد.

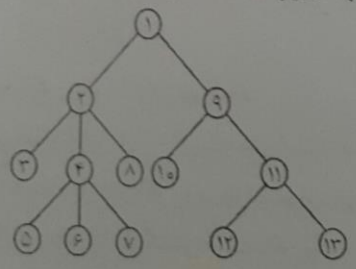
۲) روش پسگرد از اصول و فنون درخت‌ها استفاده می‌کند. بخصوص از روش پیمایش preorder درخت.

۳) مسائلی که به روش پسگرد حل می‌شوند، اغلب مسائل تصمیم گیری هستند. تصمیمات، مرحله به مرحله اتخاذ می‌گردد. مجموعه تصمیم‌ها به صورت یک درخت قابل نمایش است که به آن، درخت تصمیم (Decisiontree) می‌گویند.

۴) چنانچه در مرحله ای از الگوریتم، کلیه انتخاب‌های ممکن (تصمیم‌ها) بررسی گردد و هیچ کدام قابل قبول نباشند، باید تصمیم مرحله قبل را تغییر داد. این بدان معنی است که باید از یک گره درخت، به گره پدر آن، بازگشت کنیم.

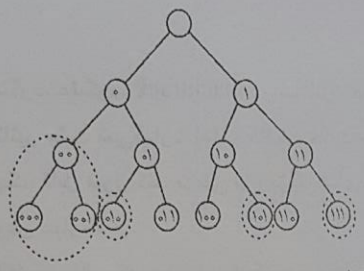
۵) چنانچه مساله بیش از یک جواب داشته باشد، همه جواب‌ها را باید پیدا کنیم.

یادآوری: گره‌های درخت مقابل به ترتیب preorder شماره‌گذاری شده‌اند:



به عنوان مثال یک قفل رمزی دیجیتالی را در نظر بگیرید. فرض کنید رمز قفل یک عدد n بیتی است. اگر هیچ اطلاعاتی راجع به رمز نداشته باشیم باید 2^n حالت را امتحان کنیم. در حالیکه قفل فقط با یک حالت باز می‌شود.

اگر n=۳ باشد باید ۸ حالت را آزمایش کنیم ولی اگر بدانیم که رمز شامل دو رقم یک بوده، می‌توان سریع تر به جواب رسید. درخت زیر که به درخت فضای حالت (State spacetree) معروف است تمام حالت‌های مختلف را نشان می‌دهد:



در درخت فوق، قسمتهایی که نقطه چین مشخص شده را بررسی نمی‌کنیم. یعنی هرگاه به گره ای رسیدیم که مطمئن بودیم ادامه آن به جواب نمی‌رسد به گره پدر باز می‌گردیم مثلاً در درخت فوق وقتی به گره ۰۰ می‌رسیم، ادامه آن به جواب نخواهد رسید و به گره ۰ باز می‌گردیم و مسیری دیگر را ادامه می‌دهیم.

یک گره را ناامید بخش گوییم اگر ادامه آن منجر به جواب نشود. مثلاً گره ۰۰ در درخت فوق، ناامیدبخش است. در غیر این صورت گره را امیدبخش گویند.

بنابراین روش پسگرد، عبارت از جستجوی در عمق در درخت فضای حالت، بررسی امیدبخش بودن آن، و در صورت امیدبخش نبودن گره، بازگشت به گره پدر، است. این فرآیند را هرس کردن درخت فضای حالت گویند. الگوریتم عمومی روش پسگرد به شکل زیر است:

```
void bt (node v)
 {
  if (promising (v))
    if (there is a solution at v)
      write the solution;
    else
      for (each child u of v)
        bt (u);
 }
```

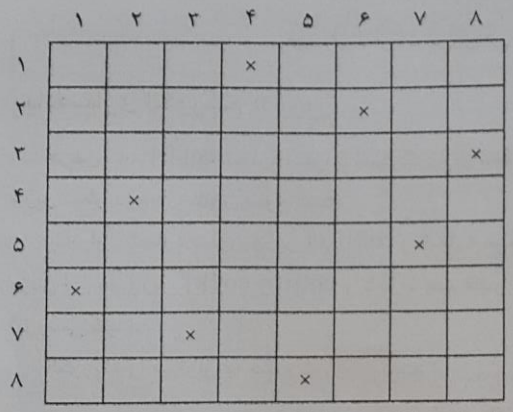
ریشه درخت فضای حالت به الگوریتم bt داده می‌شود.

تابع promising بررسی می‌کند که آیا گره v امیدبخش است یا خیر. اگر امیدبخش بود، بررسی می‌شود که آیا در آن گره جواب هست یا خیر اگر بود جواب چاپ می‌شود و در غیر این صورت فرزندان آن گره بررسی می‌شوند. چون الگوریتم bt بازگشتی است، اگر یک گره امیدبخش نبود، خود به خود عقب گرد کرده و کار را با فرزند دیگر پدر آن گره ادامه می‌دهد.

مساله n وزیر

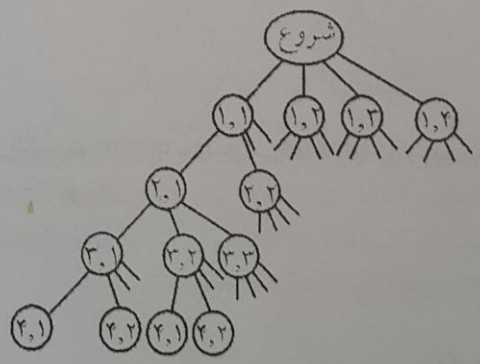
می خواهیم n وزیر را در صفحه شطرنجی n×n طوری قرار دهیم که هیچ دو وزیری یکدیگر را تهدید نکنند (سطر، ستون و قطر آنها یکسان نباشد).

به عنوان مثال به ازای n=۸ یک جواب در شکل زیر آمده است:

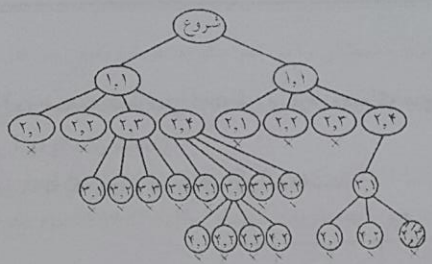


فرض کنید به ازای n=۴ می‌خواهیم مسئله را حل کنیم. با توجه به اینکه هیچ دو وزیری نمی‌توانند در یک سطر قرار بگیرند، برای هر وزیر ۴ حالت وجود دارد بنابراین تعداد کل حالات ممکن ۲۵۶ = ۴×۴×۴×۴ می‌باشد. این ۲۵۶ حالت را می‌توان با درخت فضای حالت نشان داد.

در درخت، گره‌ها با زوج (i , j) نشان داده می‌شوند که i وزیر iام است که در سطر iام قرار دارد و j شماره ستونی است که وزیر در آن قرار دارد. مثلاً زوج (۱ و ۲) یعنی وز یر سطر دوم در ستون یک قرار دارد:



اولین مسیر در درخت حالت فوق {(۱ و ۴) (۱ و ۳) (۱ و ۲) (۱ و ۱)} می‌باشد مسیر دوم در درخت {(۲ و ۴) (۱ و ۳) (۱ و ۲) (۱ و ۱)} و ... می‌باشد. بررسی همه مسیرها معقول نیست و می‌توان با استفاده از علائمی بررسی را کاهش داد و درخت را هرس کرد. مثلاً هیچ دو وزیری نباید در یک ستون یا هم قطر باشند. شکل زیر بخشی از درخت هرس شده را نشان می‌دهد. گره‌ای که امیدبخش نیست با × مشخص شده.



پیاده‌سازی الگوریتم n وزیر

فرض کنید col [i] شماره ستونی است که وزیر سطر iام در آن قرار دارد. مثلاً اگر col[۲] = ۳ یعنی وزیر سطر دوم در ستون سوم است.

بنابراین شماره ستون وزیر iام col[i] و شماره ستون وزیر kام col[k] می‌باشد. شرط هم ستون بودن این دو وزیر col[i] = col[k] و شرط هم قطر بودن آنها $|col[i] - col[k]| = |i - k|$ می‌باشد (بررسی کنید).

الگوریتم را می‌توان به صورت زیر نوشت:

```
void queens (i)
 {
   if (promising (i))
     if (i= = n)
       printf (col [1] through col [n])
else
                          // بررسی اینکه آیا وزیر سطر i+۱ را می‌توان در یکی از ستون‌ها قرار داد
       for (j=1 to n)
         {
           Col [i+1]=j;
           queens (i+1);
         }
   }
```

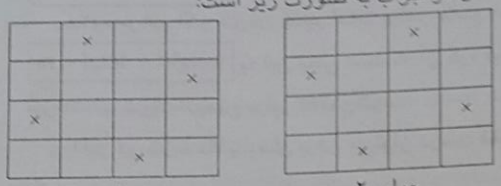
تابع promising در الگوریتم queens بررسی می‌کند که آیا دو وزیر هم ستون یا هم قطر هستند یا خیر. اگر بودند یعنی امیدبخش نیست.

```
bool promising (i)
 {
   k=1;
   switch= true;
   while (k<i) and switch do
    {
     if (col [i]= col [k]) or (abs (col [i]-col [k])=abs (i-k))
        Switch = false;
     k=k+1;
    }
    return switch;
 }
```

همانطور که قبلاً اشاره شد، الگوریتم پسگرد، تمام جواب‌ها را چاپ می‌کند البته می‌توان آن را طوری تغییر داد که فقط تعدادی جواب را چاپ کند. برای اجرا باید تابع queens را به صورت queens (0) صدا بزنیم. به ازای $n = ۴$ خروجی به صورت زیر است:

۳ و ۱ و ۴ و ۲
۲ و ۴ و ۱ و ۳

یعنی اینکه برای n=۴ دو جواب وجود دارد که شکل دو جواب به صورت زیر است:



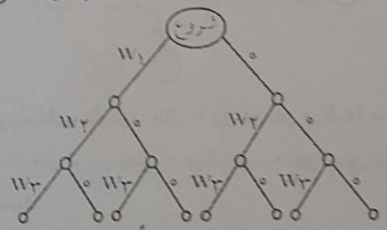
جواب ۱ جواب ۲

برای n=۵، ۱۰ جواب و برای n=۳ هیچ جوابی وجود ندارد.

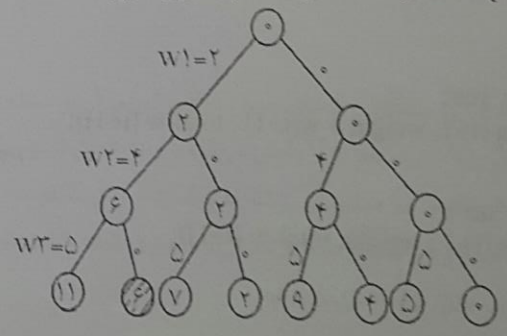
مجموع زیر مجموعه‌ها

فرض کنید $W = \{w_1, w_2, ..., w_n\}$ یک مجموعه n عضوی از اعداد صحیح و مثبت است و می‌خواهیم همه زیر مجموعه‌های W را بیابیم که مجموع اعضای هر یک m باشد. مثلاً اگر $W = \{w_1 = ۵, w_2 = ۱۳, w_3 = ۷, w_4 = ۸\}$ و m=۲۰ آنگاه دو زیر مجموعه $\{w_2 = ۱۳, w_3 = ۷\}$ و $\{w_1 = ۵, w_3 = ۷, w_4 = ۸\}$ جمع اعضایشان ۲۰ می‌باشد.

فرض کنید $W = \{w_1, w_2, w_3\}$ باشد درخت فضای حالت را برای W می‌توان به شکل زیر کشید:



یال چپ یعنی انتخاب و یال راست یعنی عدم انتخاب. مثلاً اگر از ریشه به چپ برویم یعنی w_1 را انتخاب کرده‌ایم و سطوح بعدی، انتخاب یا عدم انتخاب w_2 و w_3 را نشان می‌دهند اگر داخل گره‌ها جمع وزن‌های انتخاب شده تا آن گره را بنویسیم در این صورت با فرض m=۶ و $W = \{w_1 = ۲, w_2 = ۴, w_3 = ۵\}$ درخت فضای حالت به صورت زیر است:



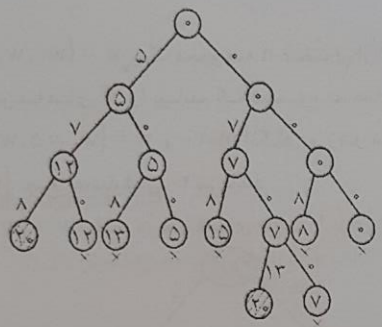
این مساله فقط یک جواب دارد $\{w_1, w_2\}$.

اگر فرض کنیم w_i ها صعودی هستند و اگر weight حاصل جمع w_i ها تا سطح iام باشد در این صورت اگر انتخاب w_{i+1} باعث شود weight (یعنی جمع اعضای انتخاب شده) از m بیشتر شود، بدیهی است که آن گره امیدبخش نیست و باید پسگرد کنیم پس یکی از شرایط غیر امیدبخش بودن: $\boxed{weight + w_{i+1} > m}$ می‌باشد.

علاوه بر این اگر فرض کنیم total برابر جمع همه اعداد باقی مانده است در این صورت اگر $\boxed{weight + total < m}$ به این معنی است که آن گره نیز امیدبخش نیست زیرا اگر همه اعداد باقی مانده نیز انتخاب شوند، مجموع برابر m نمی‌شود.

با این دو شرط ناامیدبخش بودن می‌توان درخت فضای حالت را هرس کرد و همه حالت را بررسی نکرد.

مثال: فرض کنید $w = \{w_1 = 5, w_2 = 7, w_3 = 8, w_4 = 13\}$ و m=۲۰. درخت فضای حالت هرس شده به شکل زیر است:



توجه کنید که برخی الگوریتم‌های عقبگرد، مثل n وزیر، در برگ‌ها به جواب می‌رسند. ولی برخی مثل جمع زیر مجموعه‌ها، ممکن است قبل از رسیدن به برگ نیز به جواب برسند.

الگوریتم به صورت زیر است:

```
void sum-of-subset (index i , int weight, int total)
 {
  If (promising (i))
    if (weight = = m)
      printf (include [1] thrugh include [i]);
   else {                                          //w[i+1] انتخاب

        include [i+1]= 'yes';
        sum-of-subsets (i+1, weight + w[i+1] , total-w [i+1]);
                                                   //w[i+1] عدم انتخاب

        include [i+1]="no";
        sum-of-subsets (i+1 , weight, total-w [i+1]);
       }
}
```

در تابع فوق n و w و m و include سراسری هستند. خروجی این الگوریتم تمامی زیر مجموعه‌هایی است که جمعشان m می‌شود. تابع promising به شکل زیر است:

```
bool promising (index i)
 {
   return (weight+ total >=m) and ((weight = = m) or (weight + w [i+1]<=m));
}
```

تابع اصلی باید به صورت $sum-of-subsets(0,0,total)$ صدا زده شود که در ابتدا $total = w[1] + w[2] + ... + w[n]$.

در بدترین حالت تعداد گره‌هایی که در درخت فضای حالت الگوریتم، بررسی می‌شوند برابر است با:

$$1 + 2 + 2^2 + ... + 2^n = 2^{n+1} - 1$$

هر چند که در موارد زیادی تعداد واقعی گره‌های مورد بررسی از حداکثر فوق کمتر است ولی به هر حال الگوریتم فوق از مرتبه نمایی است و ممکن است در شرایطی لازم باشد همه گره‌ها بررسی شوند مثلاً در حالتی که داشته باشیم:

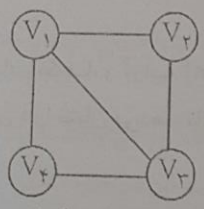
$$w_1 + w_2 + ... + w_{n-1} < m \ , \ w_n = m$$

در این صورت تنها جواب مسئله $\{w_n\}$ است که برای یافتن آن باید همه گره‌ها بررسی شوند.

رنگ‌آمیزی گراف

می‌خواهیم تمام راه‌های ممکن برای رنگ آمیزی رئوس یک گراف با استفاده از m رنگ مختلف را بیابیم به طوری که هیچ دو راس مجاوری همرنگ نباشند.

مثال: در گراف شکل زیر:

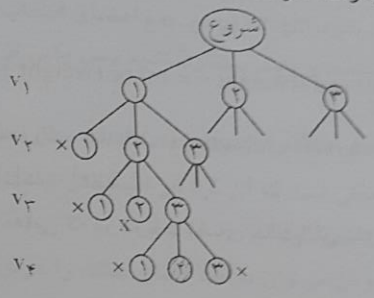


فرض کنید با ۳ رنگ ۳ و ۲ و ۱ می‌خواهیم رئوس را رنگ کنیم برای این مساله ۶ راه وجود دارد که ۲ تای آن عبارتند از:

V_1	V_2	V_3	V_4
۱	۲	۳	۲
۲	۱	۳	۱

در درخت فضای حالت این مساله، رنگ‌های ممکن برای راس V_1 در سطح ۱ امتحان می‌شوند، سپس رنگ‌های ممکن برای راس V_2 در سطح ۲ آزمایش می‌گردد و به همین ترتیب تا اینکه رنگ‌های ممکن برای V_n در سطح n امتحان می‌شود. هر مسیر از ریشه تا برگ یک حل کاندید است. یک گره در صورتی ناامیدبخش است که گره مجاورش قبلاً با همان رنگ، رنگ آمیزی شده باشد. شکل زیر بخشی از درخت فضای حالت را برای گراف مثال قبل نشان می‌دهد.

اولین حل در گره سایه‌دار است. این حل نشان می‌دهد که راس V_1 با رنگ ۱، راس V_2 با رنگ ۲، راس V_3 با رنگ ۳ و راس V_4 با رنگ ۲، رنگ شده‌اند.



الگوریتم به شکل زیر است:

```
void m_coloring (index i)
{
  int color;
  if (promising (i))
    if (i= = n)
      printf (vcolor [1] through vcolor [n]);
    else
                                  هر رنگ بعدی را برای گره بعدی آزمایش می‌کند//
      for (color=1 to m)
        {
          vcolor [i+1]= color;
          m_coloring (i+1);
        }
  }
```

در تابع فوق، n تعداد رئوس، m تعداد رنگ‌ها، و آرایه vcolor [1 .. n] خروجی الگوریتم است که vcolor [i] رنگ نسبت داده شده به راس i را نشان می‌دهد. تابع promising به شکل زیر است:

```
bool promising (index i)
 {
  index j=1;
  bool switch= true;
  while ((j<i) and switch) // check if an adjacent
                           // vertex is already this color
    {
    if (w [i] [j] and (vcolor [i]= = vcolor [j]))
      switch= false;
    j=j+1;
   }
  return switch
}
```

در تابع فوق، w ماتریس همجواری گراف است. اگر راس i و j مجاور باشند آنگاه w[i][j]=true در غیر این صورت false خواهد بود.

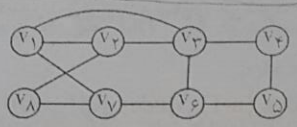
تابع را به صورت m _ coloring (۰) فراخوانی می‌کنیم. تعداد گره‌های درخت فضای حالت برابر است با:

$$۱+m+m^۲+...+m^n=\frac{m^{n+۱}-۱}{m-۱}$$

می‌توان برای حالت خاص m=۲ الگوریتمی نوشت که پیچیدگی آن در بدترین حالت نمایی نباشد. ولی برای $m \geq ۳$ تاکنون کسی الگوریتمی که نمایی نباشد، ابداع نکرده است.

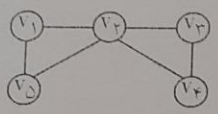
سیکل همیلتونی

سیکل همیلتونی، دوری است که از همه رئوس فقط و فقط یکبار عبور کند. مثلاً در گراف زیر



دنباله $[V_۱V_۲V_۸V_۷V_۶V_۵V_۴V_۳V_۱]$ یک سیکل همیلتونی است.

ولی در گراف شکل زیر سیکل همیلتونی وجود ندارد.



می‌خواهیم تمام سیکل‌های همیلتونی یک گراف غیر جهت دار را بیابیم.

فرض کنید مجموعه رئوس $V=\{۰,۱,۲,...,n-۱\}$ باشد.

درخت فضای حالت برای این مساله به این شرح است: راس شروع را در سطح ۰ درخت قرار می‌دهیم.

در سطح ۱ به بعد همه رئوس بجز راس شروع می‌توانند باشند.

با در نظر گرفتن ملاحظات زیر می‌توان در این درخت، عقبگرد کرد:

۱) راس iام از مسیر باید مجاور با راس (i-۱)ام از آن باشد.

۲) راس (n-۱) ام باید مجاور راس صفر (راس شروع) باشد.

۳) راس i ام نمی‌تواند هیچکدام از i-۱ راس قبلی باشد.

الگوریتم به صورت زیر است. راس $V_۱$ به عنوان راس شروع در نظر گرفته شده است. گراف با ماتریس همجواری w[۱۰۰n,۱۰۰n] نمایش داده شده است.

خروجی الگوریتم vindex[۰...n-۱] است که در آن vindex [i]، راس iام مسیر است. راس شروع، vindex[۰] است:

```
void Hamilton (i)
 {
  if promising (i) then
     if (i= = n-1)
  printf (vindex [0] through vindex [n-1])
     else
     for (j=2 to n-1) // try all vertices as next one
     {
       Vindex [i+1]=j;
       Hamilton (i+1);
       }
  }
```

که در آن تابع promising به صورت زیر پیاده سازی می‌شود:

```
bool promising (i)
  {
     if ((i = = n-1) and (w [vindex [n-1], vindex [0]]= =0))
     switch = false;                          // اولین راس باید با آخری مجاور باشد
   else
     if (i>0 and (w[vindex [i-1], vindex [i]]= =0)
      switch = false;                         //راس iام بایی با (i-1)ام مجاور باشد
     else
      {
      Switch=true;
      j=1;
      While ((j<i) and switch)                // بررسی اینکه راس قبلاً انتخاب شده
       {
        if (vindex [i]= = vindex [j])
          switch = false;
          j=j+1;
      }
     }
     return swich;
}
```

فراخوانی تابع Hamilton باید به صورت زیر باشد:

```
Vindex [0]=1;
Hamilton (0);
```

تعداد گره‌های درخت فضای حالت برابر است با:

$$1+(n-1)+(n-1)^2+\ldots+(n-1)^{n-1}=\frac{(n-1)^n-1}{n-2}$$

که بسیار بدتر از نمایی است. اگرچه تمام گره‌های فضای حالت بررسی نمی‌شوند، با این وجود، تعداد گره‌های بررسی شده، بدتر از نمایی است. مثلاً فرض کنید راس V_1 تنها به V_2 وصل است ولی بقیه رئوس غیر از V_1 همگی به هم وصل هستند، این گراف سیکل همیلتونی ندارد ولی الگوریتم، زمانی متوجه ن مطلب می‌شود که تقریباً همه گره‌ها را بررسی کرده باشد.

حل مسأله کوله‌پشتی صفر و یک با تکنیک عقبگرد

در کتاب مرجع مسأله کوله‌پشتی صفر و یک به شیوه عقبگرد نیز حل شده است که مطالعه آن را بر عهده دانشجویان قرار می‌دهیم. در اینجا فقط این نکته را یادآوری می‌کنیم که حل این مسأله با روش برنامه‌نویسی پویا از مرتبه $O(Min(۲^n, nW))$ بود و در روش عقبگرد در بدترین حالت $\theta(۲^n)$ گره را چک می‌کند. ممکن است به خاطر nW تصور شود که تکنیک پویا به تکنیک عقبگرد در این مسأله ارجحیت دارد ولی با اجرای واقعی روی کامپیوتر هورویتز و شانی در سال ۱۹۷۸ نشان دادند که در مسأله کوله‌پشتی صفر و یک روش عقبگرد معمولاً بازدهی بیشتری نسبت به روش پویا دارد.

همچنین هورویتز و شانی در سال ۱۹۷۴ با ترکیب روش تقسیم و حل و روش برنامه‌نویسی پویا الگوریتمی را طراحی کردند که در بدترین حالت به $O(۲^{n/۲})$ تعلق داشت و نشان دادند این الگوریتم اغلب بازدهی بیشتری نسبت به الگوریتم عقبگرد دارد.

توجه: تکنیک پیاده‌سازی دیگری برای الگوریتم‌ها شبیه تکنیک عقبگرد موسوم به Branch & Bound وجود دارد. در این تکنیک از پیمایش سطحی BFS استفاده می‌شود.