СТРАТЕГИИ НЕИНФОРМИРОВАННОГО ПОИСКА

Поиск на основе данных и от цели

Поиск в пространстве состояний можно вести в двух направлениях: от исходных данных задачи к цели и в обратном направлении от цели к исходным данным. При *поиске на основе данных (data-driven search* — поиск, управляемый данными), который иногда называют *прямой цепочкой (forward chaining),* исследователь начинает процесс решения задачи, анализируя ее условие, а затем применяет допустимые ходы или правила изменения состояния. В процессе поиска правила применяются к известным фактам для получения новых фактов, которые, в свою очередь, используются для генерации новых фактов. Этот процесс продолжается до тех пор, пока мы, если повезет, не достигнем цели. Возможен и альтернативный подход. Рассмотрим цель, которую мы хотим достичь. Проанализируем правила или допустимые ходы, ведущие к цели, и определим условия их применения. Эти условия становятся новыми целями, или подцелями, поиска. Поиск продолжается в обратном направлении от достигнутых подцелей до тех пор, пока (если повезет) мы не достигнем исходных данных задачи. Таким образом, определяется путь от данных к цели, который на самом деле строится в обратном направлении. Этот подход называется *поиском от цели*, или *обратной цепочкой*.

Подведем итоги: *поиск на основе данных* начинается с условий задачи и выполняется путем применения правил или допустимых ходов для получения новых фактов, ведущих к цели. *Поиск от цели* начинается с обращения к цели и продолжается путем определения правил, которые могут привести к цели, и построения цепочки подцелей, ведущей к исходным данным задачи. В обоих случаях исследователь работает с одним и тем же графом пространства состояний, однако порядок и число состояний в процессе поиска могут различаться. Какую стратегию поиска предпочесть, зависит от самой задачи. При этом следует учитывать сложность правил, "форму" пространства состояний, природу и доступность данных задачи. Все это может изменяться от задачи к задаче.

Процесс поиска от цели рекомендован в следующих случаях.

1. Цель поиска (или гипотеза) явно присутствует в постановке задачи или может быть легко сформулирована. Например, если задача состоит в доказательстве математической теоремы, то целью является сама теорема. Многие диагностические системы рассматривают возможные диагнозы, систематически подтверждая или отвергая некоторые из них способом поиска от цели.
2. Имеется большое число правил, которые на основе полученных фактов позволяют продуцировать возрастающее число заключений или целей. Своевременный отбор целей позволяет отсеять множество возможных ветвей, что делает процесс поиска в пространстве состояний более эффективным. Например, в процессе доказательства математических теорем число используемых правил вывода теоремы обычно значительно меньше количества, формируемого на основе полной системы аксиом.

3. Исходные данные не приводятся в задаче, но подразумевается, что они должны быть известны решателю. В этом случае поиск от цели может служить руководством для правильной постановки задачи. В программе медицинской диагностики, например, имеются всевозможные диагностические тесты. Доктор выбирает из них только те, которые позволяют подтвердить или опровергнуть конкретную гипотезу о состоянии пациента.

Таким образом, при поиске от цели подходящие правила применяются для исключения неперспективных ветвей поиска.

Поиск на основе данных применим к решению задачи в следующих случаях.

1. Все или большинство исходных данных заданы в постановке задачи. Задача интерпретации состоит в выборе этих данных и их представлении в виде, подходящем для использования в интерпретирующих системах более высокого уровня. На стратегии поиска от данных основаны системы анализа данных определенного типа.
2. Существует большое число потенциальных целей, но всего лишь несколько способов применения фактов и представления информации о конкретном примере задачи.
3. Сформировать цель или гипотезы очень трудно.

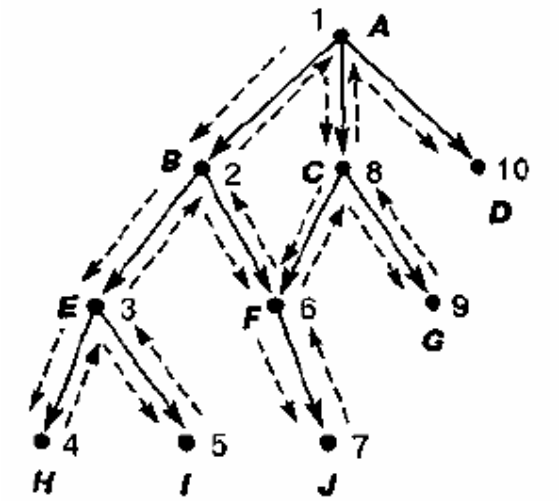
При поиске на основе данных знания и ограничения, заложенные в исходной постановке задачи, используются для нахождения пути к решению.

Реализация поиска на графах

При решении задач путем поиска на основе данных либо от цели требуется найти путь от начального состояния к целевому на графе пространства состояний. Последовательность дуг этого пути соответствует упорядоченной последовательности этапов решения задачи. Модуль решения задачи должен безошибочно двигаться прямо к цели, запоминая путь движения. *Поиск с возвратами {backtracking)* — это метод систематической проверки различных путей в пространстве состояний.

Алгоритм поиска с возвратами запускается из начального состояния и следует по некоторому пути до тех пор, пока не достигнет цели либо не упрется в тупик. Если цель достигнута, поиск завершается, и в качестве решения задачи возвращается путь к цели. Если же поиск привел в тупиковую вершину, то алгоритм возвращается в ближайшую из пройденных вершин и исследует все ее вершины-братья, а затем спускается по одной из ветвей, ведущих от вершины-брата. Алгоритм работает до тех пор, пока не достигнет цели либо не исследует все пространство состояний.

На рисунке изображен процесс поиска с возвратами в гипотетическом пространстве состояний. Пунктирные стрелки в дереве указывают направление процесса поиска в пространстве состояний (вниз или вверх). Числа возле каждой вершины указывают порядок их посещения.



Ниже приведем алгоритм поиска с возвратами. В нем используются три списка, позволяющих запоминать путь от узла к узлу в пространстве состояний. *SL* (State List) — список исследованных состояний рассматриваемого пути. Если цель уже найдена, то *SL* содержит список состояний пути решения. *NSL* (New State List) — список новых состояний, он содержит вершины, подлежащие рассмотрению, т.е. список вершин, потомки которых еще не были порождены и рассмотрены. *DE* (Dead Ends) — список тупиков, т.е. список вершин, потомки которых уже были исследованы, но не привели к цели. Если состояние из этого списка снова встречается в процессе поиска, то оно обнаруживается в списке *DE* и исключается из рассмотрения. При описании алгоритма поиска с возвратами на графах общего вида необходимо учитывать возможность повторного появления состояний, чтобы избежать их повторного рассмотрения, а также петель, ведущих к зацикливанию алгоритма поиска пути. Это обеспечивается проверкой каждой вновь порожденной вершины на ее вхождение в один из трех вышеуказанных списков. Если новое состояние обнаружится хотя бы в одном из двух списков *SL* или *DE*, значит, оно уже рассматривалось, и его следует проигнорировать.

Обозначим текущее состояние при поиске с возвратами через *CS* (current state). Состояние *CS* всегда равно последнему из состояний, занесенных в список *SL*, и представляет "фронтальную" вершину на построенном в данный момент пути. Правила вывода, ходы в игре или иные соответствующие операторы решения задачи упорядочиваются и применяются к *CS*. В результате возникает упорядоченное множество новых состояний, потомков *CS*. Первый из этих потомков объявляется новым текущим состоянием, а остальные заносятся в список новых состояний *NSL* для дальнейшего изучения. Новое текущее состояние заносится в список состояний *SL*, и поиск продолжается. Если текущее состояние *CS* не имеет потомков, то оно удаляется из списка состояний *SL* (именно в этот момент алгоритм "возвращается назад"), и исследуется какой-либо из оставшихся потомков его предка в списке состояний *SL*.

```
functionbacktrack;

begin
  SL:=[Start];NSL:=[Start];DE:=[];CS:=Start;
%инициализация:
  while NSL≠[] do        %пока существуют неисследованные состояния
   begin
    if CS=goal(или удовлетворяет описанию цели)
     then return SL;      %при нахождении цели вернуть список
                          %состояний пути.
if CS не имеет потомков (исключая узлы, входящие в DE, SL, and NSL)
      then begin
        while SL не пуст и CS=первый элемент списка SL do
         begin
          добавить CS в DE;              %внести состояние в список
                                         %тупиков
          удалить первый элемент из SL;  %возврат
          удалить первый элемент из NSL;
          CS:= первый элемент NSL;
         end
        добавить CS в SL;
     end
     else begin
       поместить потомок CS(кроме узлов, уже содержащихся в DE,
                            SL или NSL) в NSL;
       CS:= первый элемент NSL;
       добавить CS в SL
    end
  end;
  return FAIL;
end.
```

Пример

SL=[A]; NSL=[A]; DE=[]; CS=A;

После итерации	Текущее состояние CS	Список состояний SL	Новые состояния NSL	Тупики DE
0	*A*	*[A]*	*[A]*	[ ]
1	*B*	*[B A]*	*[B C D A]*	[ ]
2	*E*	*[E B A]*	*[E F B C D A]*	**N**
3	*H*	*[H E B A]*	*[H I E F B C D A]*	[ ]
4	*I*	*[I E B A]*	*[I E F B C D A]*	*[H]*
5	*F*	*[F B A]*	*[F B C D A]*	*[E I H]*
6	*J*	*[J F B A]*	*[J F B C D A]*	*[E I H]*
7	*C*	*[C A]*	*[C D A]*	*[B F J E I H]*
8	*G*	*[G C A]*	*[G C D A]*	*[B F J E I H]*

Поиск с возвратами в данном случае является поиском на основе данных, при котором корень дерева связывается с начальным состоянием, а потомки узлов анализируются для построения пути к цели. Этот же алгоритм можно интерпретировать и как поиск от цели. Для этого целевую вершину следует взять в качестве корня дерева и анализировать совокупность предков для нахождения пути к начальному состоянию.

Поиск с возвратами (backtrack) — это алгоритм поиска на графе пространства состояний. Алгоритмы поиска на графах, которые будут рассматриваться далее, включая *поиск в глубину (depth-first), поиск в ширину (breadth-first)* и *поиск по первому наилучшему совпадению (bestfirst search),* используют идеи поиска с возвратами, в том числе следующие.

1. Формируется список неисследованных состояний *(NSL),* для того чтобы иметь возможность возвратиться к любому из них.
2. Поддерживается список "неудачных" состояний *(DE),* чтобы оградить алгоритм от проверки бесполезных путей.
3. Поддерживается список узлов *(SL)* текущего пути, который возвращается по достижении цели.
4. Каждое новое состояние проверяется на вхождение в эти списки, чтобы предотвратить зацикливание.