Эвристический перебор на Лиспе

Приведем текст лисповской функции HEURISTIC_SEARCH, реализующей эвристический перебор в пространстве состояний с использованием эвристической оценочной функции EST (зависящей от конкретной поисковой задачи). Вычисляемая для каждого состояния эвристическая оценка хранится как четвертый элемент списка-описания состояния, а третий элемент этого списка является глубиной состояния в дереве поиска. Заметим, что на шаге 3, выбирая из списка Open первый элемент-вершину, мы тем самым выбираем вершину с минимальной оценкой, поскольку список Open всегда упорядочен по неубыванию оценок вершин-состояний, хранящихся в нем (это обеспечивается вспомогательной функцией MERGE).

```
   (defun HEURISTIC_SEARCH(StartState)
     (prog (Open  Closed  Current
            Deslist ;список дочерних вершин;
            Reflist ;список указателей;
            Depth ;глубина текущей вершины;)
;Шаг 1:;    (setq Open (list (list 'S0  StartState  0
                                  (EST (list StartState 0)))))
;Шаг 2:; HS(cond((null Open) (return())))
;Шаг 3:;    (setq Current (car Open))
           (setq Open (cdr Open))
           (setq Closed (cons Current Closed))
           (setq Depth (caddr Current))
;Шаг 4:;    (cond((IS_GOAL Current)
                 (return (SOLUTION Current Reflist))))
;Шаг 5:;    (setq Deslist (OPENING Current))
; Исключение повторных вершин-состояний;
           (setq Deslist (RETAIN_NEW Deslist))
           (cond((null Deslist) (go HS) ))
;Шаг 6:;    (setq Open (MERGE (SORT(ADD_DEPTH_EST(add1 Depth) Deslist))
                             Open))
           (setq Reflist(append(CONNECT Deslist Current)
                               Reflist))
           (go HS) ))
```

Данная функция использует вспомогательные функции OPENING, SOLUTION, IS_GOAL, CONNECT, EST (которые зависят от конкретной поисковой задачи), а также три вспомогательные функции: RETAIN_NEW, ADD_DEPTH_EST и MERGE.

Вспомогательная функция RETAIN_NEW оставляет в списке дочерних состояний Dlist только те, которые не порождались ранее – тем самым исключается зацикливание при поиске в произвольном графе. Вспомогательная функция SORT упорядочивает (по неубыванию эвристической оценки) список дочерних вершин, к элементам которого добавлена информация о глубине в дереве поиска и об оценке. Определения этих двух функций мы не рассматриваем.

Функция ADD_DEPTH_EST устанавливает глубину дочерних вершин и вычисляет их эвристическую оценку:

```
   (defun ADD_DEPTH_EST (Dn Slist)
      (cond((null Slist) ())
           (t (cons(list (caar Slist) (cadar Slist) Dn
                         (EST(list (cadar Slist) Dn)) )
                   (ADD_DEPTH_EST Dn (cdr Slist)) ) )) )
```

Функция MERGE выполняет слияние двух упорядоченных (по неубыванию эвристической оценки) списков состояний в результирующий упорядоченный список:

```
   (defun MERGE (L1 L2)
      (cond((null L1) L2)
           ((null L2) L1)
           ((gt (car (cdddar L1)) (car (cdddar L2)))
                (cons(car L2) (MERGE L1 (cdr L2))))
           (t   (cons(car L1) (MERGE (cdr L1) L2))) ))
```

Игра в восемь (Лисп)

Граф-пространство состояний для головоломки-игры в восемь достаточно велик (в игре в пятнадцать фишек он на порядок больше), поэтому хотя в принципе применимы алгоритмы слепого поиска, предпочтителен все же эвристический поиск. Опишем нужные для `HEURISTIC_SEARCH` вспомогательные лисп-функции: `IS_GOAL, EST, OPENING, SOLUTION, CONNECT` - для игры в восемь. Состояние задачи (конфигурация игры) представляется списком из следующих элементов:

- идентификатор состояния (используются атомы `S1, S2, S3, …` , генерируемые встроенной лисп-функцией `gensym`);
- собственно описание состояния – список из номеров фишек, записанных последовательно по рядам квадрата;
- число-глубина состояния-вершины в дереве перебора;
- числовая эвристическая оценка состояния.

В описание состояния включается также – в качестве первого элемента списка – обозначение того оператора движения пустой клетки, который привел к данному состоянию. Этот элемент нужен, чтобы исключить тривиальные повторы состояний при раскрытии вершин. Операторы будут обозначаться соответственно именами-атомами `right, left, up, down`; а "пустышка" (пустая клетка) – символом #. Отметим, что эвристическая оценка используется только в алгоритме эвристического перебора, а глубина вершины – в алгоритмах эвристического перебора и ограниченного перебора вглубь.

Для примера, приведенного на рис. 1 в основной части текста ***День 10, лекции № 19, № 20***, начальное состояние `S0` имеет такой вид: `(? 2 8 3 1 6 4 7 # 5)`. Символ «?» означает, что оператора, который привел к данному состоянию, нет. При обращении к функции `HEURISTIC_SEARCH` с таким параметром на Шаге 1 в описание состояния `S0` будут добавлены: этот идентификатор, информация о глубине (0) и эвристической оценке (4), затем будет сформирован список `Open`:

```
((S0 (? 2 8 3 1 6 4 7 5 #) 0 4))
```

Дочерние вершины `S0`:

```
(S1 ('right 2 8 3 1 6 4 7 5 #))
(S2 ('left  2 8 3 1 6 4 # 7 5))
(S3 ('up    2 8 3 1 # 4 7 6 5))
```

будут построены (именно в таком порядке) на Шаге 5 функцией `OPENING`. Список этих вершин станет значением переменной `Deslist`.

На шаге 6 в эти описания будет добавлена информация о глубине вершин (1) и их эвристических оценках (6, 6, 4) – в функции `ADD_DEPTH_EST`.

Описанные ниже лисп-функции для игры в восемь пригодны и для игры в пятнадцать, так как размер стороны игрового квадрата, равный 3, используется в них как глобальная переменная (переменная `Size`). Другая глобальная переменная `(Goalstate)`, хранит описание целевого состояния (без идентификатора, глубины и эвристической оценки). Значение оценочной функции `EST` – сумма числа фишек, стоящих не на «своих» местах, и длины пути (глубины) оцениваемого состояния в дереве поиска.

```
(defun IS_GOAL(State)
       (equal (cdadr State) Goalstate ))

(defun EST (S)
   (prog(Len G  N  E1 E2)
       (setq Len (cadr S)) (setq N 0)
       (setq S (cdar S)) (setq G Goalstate)
;одновременный просмотр списков-описаний;
;     заданного и целевого состояний;
    ES (cond((null S) (return (+ (- N 1) Len)) ))
       (setq E1 (car S)) (setq S (cdr S))
       (setq E2 (car G)) (setq G (cdr G))
       (cond((neq E1 E2) (setq N (add1 N))
            ((eq  E1 '#) (setq N (add1 N)))
       (go ES) ))
```

```
(defun OPENING (State)
   (prog(Op St  Dlist K  I J El)
 ; выделение составных элементов описания состояния;
       (setq St (cadr State))
       (setq Op (car St))  (setq St (cdr St))
       (setq State St) (setq K 0)
 ; поиск порядкового номера K "пустышки" в списке;
    OP (setq El (car St))
       (setq K (add1 K))
       (cond ((neq El '#) (setq St(cdr St)) (go OP) ))
 ; вычисление номера ряда и номера столбца "пустышки";
       (setq I(add1(mod K Size))) (setq J (rem K Size))
 ; поочередно проверка возможности движения «пустышки»;
 ; вправо/влево/вверх/вниз; за счет анализа оператора Op;
 ; исключаем в дереве поиска тривиальные циклы, т.е. возврат;
 ; после применения двух операторов в исходное состояние;
       (cond((and (neq Op 'left) (< J Size))
                  (ADD_STATE 'right K (add1 K)) ))
       (cond((and (neq Op 'right) (> J 1))
                  (ADD_STATE 'left K (sub1 K)) ))
       (cond((and (neq Op 'down) (> I 1))
                  (ADD_STATE 'up K (- K Size)) ))
       (cond((and (neq Op 'up) (< I Size))
                  (ADD_STATE 'down K (+ K Size)) ))
       (return Dlist) ))
```

При раскрытии используется вспомогательная функция ADD_STATE, добавляющая в список дочерних вершин список-описание новой вершины из 2 элементов: идентификатора этой вершины (его генерирует функция gensym) и списка номеров фишек. В ADD_STATE применяется встроенная лисп-функция NTH, выбирающая из заданного списка элемент с указанным порядковым номером.

```
(defun  ADD_STATE (Op K1 K2)
 (push(list(gensym)
       (cons Op
        (cond((< K1 K2)
              (EXCHANGE State 1 '# K1 (NTH K2 State)K2))
         (t (EXCHANGE State 1(NTH K2 State)K2 '# K1))) ))
       Dlist))
```

Рекурсивная функция EXCHANGE, используемая в ADD_STATE, формирует новое состояние игры путем перестановки заданных элементов исходного состояния-списка List (переменная K служит для просмотра этого списка, при обращении к функции ее значение равно 1).

```
(defun EXCHANGE (List K Elem1 K1 Elem2 K2)
  (cond((eql K K1) (cons Elem2
         (EXCHANGE(cdr List)(add1 K)Elem1 K1 Elem2 K2) ))
       ((eql K K2)(cons Elem1 (cdr List)) )) )
```

Функция CONNECT формирует список указателей от текущей вершины-состояния Curr к заданным (в списке Dlist) дочерним вершинам-состояниям. Каждый указатель есть трехэлементный список из идентификатора родительской вершины, идентификатора дочерней вершины и названия связывающего их оператора.

```
(defun CONNECT (Dlist Curr)
   (prog (D Di Ci Rlist)
     C (setq D (car Dlist)) (setq Dlist (cdr Dlist))
       (setq Di (car D))  (setq Ci (car Curr))
       (setq Rlist(cons (list Ci Di (caadr D))Rlist))
       (cond ((null Dlist) (return Rlist) ))
       (go C) ))
```

Функция SOLUTION, выделяя решающий путь, строит последовательность (названий) операторов, преобразующих начальное состояние в целевое (ее аргумент Reflist – список всех указателей-связей между состояниями). Для поиска указателя к очередной вершине решающего пути функция использует вспомогательную функцию LOOK_FOR.

```
(defun SOLUTION (Goal Reflist)
  (prog (Sollist ;список, в котором строится решение;
           Gi Edge)
     (setq Gi (car Goal))
   S (cond((eq Gi 'S0) (return Sollist) ))
     (setq Edge (LOOK_FOR Gi Reflist))
     (setq Sollist (cons(caddr Edge) Sollist))
     (setq Gi (car Edge))
     (go S) ))

(defun LOOK_FOR (Id List)
      (cond((null List) (quote error))
           ((eq ID (cadar List)) (car List))
           (t(LOOK_FOR Id (cdr List)) )) )
```

В заключение приведем блок, инициализирующий эвристический поиск решения игры в восемь. В самом его начале устанавливаются значения параметров встроенной функции gensym.

```
(prog(Goalstate Initstate Size)
      (setq *gensym-prefix* 'S)
      (setq *gensym-count* 1)
      (setq Size 3)
      (setq Goalstate '(1 2 3 8 # 4 7 6 5))
      (setq Initstate '(? 2 8 3 1 6 4 7 # 5))
      (HEURISTIC_SEARCH Initstate))
```