УДК 512.25/26

ВЫЧИСЛИТЕЛЬНЫЕ АСПЕКТЫ КОНЕЧНОГО ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Буй Дык Тхык, И.В.Романовский

I. Введение. Ричард Беллман в своих многочисленных работах (мы сошлемся на основные монографии [1,2]) предложил замечательный метод решения оптимизационных задач рекуррентного типа, который в принципе применим к широкому классу практических задач.

Этот метод, как известно, заключается в решении функционального уравнения для экстремального значения целевой функции от параметров задачи соответственно уравнения Беллмана и функции Беллмана. Для численного нахождения функции Беллмана было предложено несколько вычислительных схем, таких как метод последовательных приближений [1]; монотонные приближения в пространстве поведений [1,3]; схема последовательных приближений Р.Ховарда [4]; метод последовательного анализа вариантов В.Михалевича [5]; недавно примененный Д.Кнутом в задачах полиграфии [6] встречавшийся у нескольких авторов метод монотонного вычисления, для которого характерно последовательное нахождение значений функции Беллмана во всех состояниях процесса, начиная от терминальных, с последовательным определением состояний, в которых вычисленное значение функции уже окончательное. Типичным примером этого последнего является метод Дейкстры [7] для задачи о кратчайшем пути в ориентированном графе. Интересно, что даже в этой простейшей задаче продолжается развитие алгоритмов и эффективность численных методов повышается (см.[8], нельзя не упомянуть также обзор [9]).

В этой статье мы хотели бы сравнить перечисленные подходы с неожиданно простой идеей организации расчетов, которая с учетом современных тенденций построения программного обеспечения представляется весьма эффективной. Для обсуждения выбрана самая простая из схем динамического программирования, а возможные обобщения обсуждаются в конце статьи.

2. Простейшая модель. Рассмотрим процесс динамического программирования с конечным множеством состояний S, в котором невозможны циклы, т.е. никакое состояние не может встретиться в одной и той же траектории дважды.

Этот процесс может быть описан в терминах графа переходов, с множеством вершин S, в котором дуги, выходящие из вершины $s \in S$, соответствуют решениям, принимаемым в состоянии s. Таким образом, каждому $s \in S$ отвечает множество дуг Q_s, выходящих из s. Каждой дуге $q \in Q_s$ сопоставляется новое состояние $j(q) \in S$ и затраты $c(q)$ (для простоты функция затрат на траектории принимается аддитивной).

Состояния, в которых множество решений пусто, называются терминальными, в них процесс останавливается. Множество терминальных состояний обозначим через S_+. Каждому состоянию $s \in S_+$ сопоставлены терминальные затраты $g(s)$.

Требуется по данному начальному состоянию s_0 найти траекторию, т.е. последовательность решений, приводящих процесс в какое-либо терминальное состояние, на которой суммарные затраты от принятых решений и этого терминального состояния были бы минимальны.

Обозначая через $v(s)$ минимальные затраты при начальном состоянии $s_0 = s$, получим функциональное уравнение Беллмана

$$v(s) = \min \{ c(q) + v(j(q)) \mid q \in Q_s \}, \quad s \in S \setminus S_+, \quad (1)$$

$$v(s) = g(s), \quad s \in S_+. \quad (2)$$

Обычные методы решения этого уравнения связаны с построением сходящейся последовательности функций

$$v_n(s) = \min \{ c(q) + v_{n-1}(j(q)) \mid q \in Q_s \}, \quad s \in S \setminus S_+, \quad (3)$$

для $n \geqslant 1$, функция $v_0(s)$ задается. Это и есть метод последовательных приближений.

При особом выборе начального приближения v_0 эта последовательность приобретает дополнительные свойства. Например,

выбор в качестве v_* решения уравнения

$$v_*(s) = c(q_s) + v_*(j(q_s)), \quad s \in S \setminus S_+, \qquad (4)$$

где $\{q_s\}$ – некоторое поведение, т.е. набор управлений для всех $s \in S \setminus S_+$, обеспечивает монотонное неубывание v_n с ростом n (метод приближений в пространстве поведений).

Для рассматриваемой модели, как было давно отмечено Р. Беллманом, можно последовательно находить значение функции $v(s)$ слоями, по максимальному числу шагов до терминального множества S_+.

В самом деле, для слоя $S_0 = S_+$ функция $v(s)$ определена в (2). Пусть

$$T_1 = \{ s \mid j(q_s) \in S_0 \quad \text{для всех} \quad q_s \in Q_s \}. \qquad (5)$$

Значения $v(s)$ находятся для $s \in T_1$ непосредственно по соотношению (I). Положим $S_1 = S_0 \cup T_1$. Далее, обозначая

$$T_2 = \{ s \mid j(q_s) \in S_1 \text{ для всех } q_s \in Q_s, \; s \notin S_1 \}, \qquad (6)$$

устанавливаем, что $v(s)$ вычислимо на T_2; полагаем $S_2 = S_1 \cup T_2$ и т.д. За конечное число шагов функция $v(s)$ будет вычислена во всех состояниях процесса.

Этот вариант направленного вычисления $v(s)$ мы и примем за основу. Хотя он представляется наиболее экономным по количеству "минимизаций", т.е. вычислений по формуле (I), его реализация может приводить к трудности, например при опознании принадлежности состояний тому или иному слою T_i. Обсудим это более подробно.

3. Обсуждение трудностей. Прежде всего об опознании слоя. На самом деле, и это хорошо известно, нет необходимости знать номер слоя. Важно лишь, чтобы к моменту, когда будет вычисляться значение $v(s)$ для данного s, были известны $v(s')$ для всех s', которые могут встретиться в правой части уравнения (I). Точнее, к моменту, когда значение $v(s)$ будет фиксироваться как окончательное, должны быть найдены и использованы для вычисления $v(s')$ все нужные $v(s')$. В некоторых специальных случаях это требование легко выполнить.

Например, есть задачи, где состояния процесса естественно упорядочены и переход возможен только в соответствии с этим упорядочением, скажем, от старших состояний к младшим. В таких задачах нужное свойство автоматически обеспечивается

при просмотре состояний в направлении от младших к старшим. Такая организация характерна для методов последовательного анализа вариантов [5,6,10], в котором появляется специфическое множество "рабочих состояний", находящихся в режиме продолженного вычисления минимума. К таким методам относится и метод Дейкстры для нахождения кратчайшего пути в ориентированном графе, где, впрочем, окончательность вычисленного значения устанавливается несколько иначе, с использованием неотрицательности длины дуг графа. Для общего случая в развитие метода Дейкстры может быть организован подсчет для каждого $s \in S$ количества невычисленных компонент в фигурных скобках в (I) с выделением тех вершин, для которых это число обратилось в ноль, и вычисление минимума тем самым завершено. Так можно вычислять, например, критический путь в сетевом графике (см. [II]).

Однако большое число практически важных задач не подходит под описанные случаи, и в них задача распознания состояний с уже вычисленными значениями функции Беллмана так просто не решается.

Основная же трудность численного решения задач динамического программирования лежит в больших размерах множества S, и на преодоление "проклятия размерности" и должны быть направлены основные усилия.

Часто в задачах с большим числом состояний возможности перехода процесса из состояния в состояние оказываются сильно ограниченными. В этом случае значительная часть работы по вычислению значений $v(s)$ оказывается напрасной, и можно достичь успеха, если ограничиться в вычислениях лишь теми состояниями, которые нужны для решения поставленной задачи. Именно этот подход и рассматривается ниже.

4. Тривиальная рекурсия. Если ставить целью ограничение вычислений только необходимыми значениями $v(s)$, то уравнение (I) дает способ непосредственного вычисления $v(s_0)$ в рекурсивной процедуре такого типа (в некоторой смеси алгоритмических языков):

```
I.  v:  proc(s) returns (real);
2.          if    s<конечный пункт> then  min:=g(s);
3.              else      min:= infinity;
4.          do    d ∈ q(s);
```

```
5.                ctq:= c(d) + v(j(d));
6.                if min > ctq  then min:=ctq;
7.      end;
8.   return (min);
9.  end v;
```

Условный оператор в строках 2-7 состоит из двух ветвей, соответствующих (2) и (I). В строке 5 процедура обращается к самой себе с другим значением аргумента. Таким образом, для вычисления $v(s)$ потребуются значения $v(j(q))$ для всех $q \in Q_s$ с соответствующими вызовами процедуры v. Очевидный дефект этого наивного алгоритма в том, что если одно и то же состояние встречается в нескольких траекториях, по которым идет вычисление, произойдет "цепная реакция" нарастания вычислений, от которой и защищают нас методы Беллмана.

5. Модифицированный каскад. Выход из положения дает запоминание вычисленных значений. Образуется специальный программный механизм - информационная служба, которая хранит всю информацию о состояниях с известными $v(s)$, как терминальными, так и вычисленными, и выдает эту информацию по запросам. Схема вычислений из предыдущего пункта лишь слегка модифицируется:

```
I.    v:  proc(s) returns (real);
2.        if < v(s) уже вычислено> then min:= v(s);
3.                 else   min := infinity;
4.        do   d ∈ Q(s);
5.             ctq:= c(d) + v(j(d));
6.          if min > ctq then min:= ctq; end;
7.  <запомнить пару      (s, min) > ;
8.     return (min);
9.     end v;
```

Здесь только строки 2 и 7 содержат изменения, вызванные появлением информационной службы. Внешне эта служба имеет два очевидных режима: запоминание пары (s, v) (или тройки (s, v, q), если требуется знать оптимальное управление) и поиск информации о данном состоянии s.

Выделение специальной информационной службы позволяет легко варьировать ее реализацию в зависимости от условий конкретной задачи. Например, если количество состояний в множестве допускает хранение информации для всех точек, время

вычисления v несколько увеличится за счет усложнения доступа к вычисленным уже значениям, но сэкономится в той мере, насколько избыточно множество S .

Особый интерес представляет возможность организации в этой информационной службе вспомогательных вычислений функции Беллмана для тех точек, в которых она еще не найдена, но может быть точно или приближенно восстановлена по уже найденным значениям. Этот вопрос более подробно разбирается ниже.

6. Информационно-устойчивые модули. Схему, описанную в предыдущем пункте, удобно уточнить основываясь на понятии информационно-устойчивого модуля, введенного Д.Парнасом [I2], для которого в [I3,I4] мы предложили название "служба". Под службой понимается совокупность процедурных возможностей, объединенных информацией, недоступной непосредственно программному окружению. Поскольку специальная конструкция для описания служб (так называемые модули) имеется в языке Модула-2 [I5], дальнейшие схемы будут Модула-подобными.

Каскадная схема динамического программирования представляется тремя службами: службой вычисления v с режимами вычисления восстановления оптимальной траектории (эту часть мы опустим) и инициализации;службой процесса с режимами инициализации, перебора решений,возможных в данном состоянии процесса с формированием результатов очередного решения, перебора терминальных состояний; службой информации, режим которой описан выше.

Службы информации и процесса представлены описанием их интерфейсов. Так, для службы процесса получим (оставляя в стороне вопросы передачи из этой службы в службу информации набора терминальных значений)

```
DEFINITION  MODULE   dq;
  EXPORT  QUALIFIED   openqts, opendq, nextdec;
  TYPE  STATE;
  PROCEDURE   opendq;
  PROCEDURE   openqts (state : STATE);
  PROCEDURE   nextdec (state : STATE);
                      VAR    snew : STATE;
                      VAR    r : REAL;
                      VAR    cont:BOOLEAN)
END dq.
```

Служба информации имеет такое внешнее описание:

```
DEFINITION   MODULE   inf;
  EXPORT QUALIFIED    opinf, save, find;
  TYPE   STATE;
  PROCEDURE   opinf;
  PROCEDURE   save ( s: STATE;
                     VAR  v: REAL;
                     VAR  b: BOOLEAN)
END inf.
```

Служба каскада имеет с учетом этих внешних описаний следующий вид:

```
MODULE    cascade;
 IMPORT    dq, inf;
 TYPE STATE;
 CONST   infinity = 10 E 50; (* большее число   *)
 VAR    s, snew: STATE;
           min: REAL;
            v : REAL;
PROCEDURE  Bellman ( s: STATE;
                     v: REAL);
          VAR ex  : BOOLEAN;
              vnew: REAL;
   BEGIN
         inf. find (s, v, ex);
        IF    NOT    ex   THEN
 LOOP
      dq.nextdec (s, snew, ctq, ex);
      IF   NOT ex   THEN   EXIT   END;
      Bellman (snew, vnew);
      ctq:= ctq + vnew;
      IF  ctq < min   THEN  min:= ctq  END
 END  (*  конец цикла    *)
END;
 inf. save (s, min); v:= min
END  Bellman
END  cascade.
```

Для простоты описания в него не включен процесс инициализации служб и запоминание оптимальных управлений.

7. Пример. Задача линейного раскроя. При решении задач серийного линейного раскроя [16] появляется экстремальная задача, которая давно уже служит для испытания различных методов динамического программирования.

Пусть заданы натуральное число m, набор положительных целых чисел $\ell_1, \ell_2, \ldots, \ell_m$ (длина деталей) и положительных чисел $c_1, c_2, \ldots, c_m$ (цен деталей). Требуется разрезать материал длины τ на детали заданных длин таким образом, чтобы цена отрезанных деталей была максимальна. Иными словами, требуется максимизировать $\sum_{i \in 1:m} c_i x_i$ при условиях

$$\sum_{i \in 1:m} \ell_i x_i \leq \tau , \qquad (7)$$

$$x_i \text{ – положительные и целые.} \qquad (8)$$

Как обычно, обозначая через $v(s,k)$ максимальную цену при $\tau = s$ и $m = k$, легко получить уравнение Беллмана в одной из следующих форм:

$$v(s,k) = \max \{ c_i + v(s-\ell_i, k) \mid i \in 1:k ,\ s \geq \ell_i \} , \qquad (9)$$

или

$$v(s,k) = \max \{ c_k \cdot d + v(s-\ell_k \cdot d, k-1) \mid d \in 0:\lceil s/\ell_k \rceil \} , \qquad (10)$$

или

$$v(s,k) = \max \{ v(s,k-1), c_k + v(s-\ell_k, k) \} . \qquad (11)$$

Мы опускаем здесь естественные граничные условия.

Интересно сопоставить основанные на этих формах вычислительные процедуры. Традиционное вычисление проводится по формулам (9) и (10). В случае (9) достаточно вычислять только $v(s,m)$, $s \in 1:\tau$, в то время как в (10) нужно вычислить m слоев функции. Тем не менее формула (10) оказывается вычислительно более технологичной. Еще удобнее в вычислительном отношении формула (11).

Кроме очевидного арифметического упрощения формула (11) дает еще и возможности сокращения требуемой памяти [10]. Именно, если воспользоваться тем, что при фиксированном k функция $v(s,k)$ монотонна и кусочно постоянна по s, то можно задавать и вычислять ее в виде последовательности скачков. Это соответствует методу последовательного анализа вариантов

[5]. Однако с увеличением k и ростом ς количество скачков увеличивается и эффективность такого подхода заметно падает.

Между тем, каскадное вычисление $v(\tau, m)$, например по формуле (3), позволяет уменьшить количество требуемых вычислений именно при больших τ. Эксперименты подтвердили правильность следующего подхода.

а) Путем обработки списка скачков готовится список скачков функции $v(\varsigma, m)$ для малых ς, приблизительно порядка $2 \min_i \ell_i$. Полученная информация передается службе информации при начальном заполнении.

б) Информационная служба запоминает интервалы постоянства функции $v(\varsigma, m)$ в той мере, в какой она обладает информацией о них.

в) Значение $v(\tau, m)$ вычисляется рекурсивно. При получении и запоминании двух состояний с одинаковыми значениями информационная служба запоминает весь интервал между ними и соответственно расширяет его при получении дополнительной информации.

В экспериментах была апробирована и некоторая полезная модификация этого алгоритма, заключающаяся в том, что при нахождении значения функции Беллмана вычислялась и некоторая область ее гарантированного постоянства, содержащая заданную точку. Естественно, что информационная служба в своих обменах информацией получала и сообщала при вызовах также области постоянства.

8. Пример. Задача размещения производства без транспортных издержек. Рассмотрим следующую экстремальную задачу.

Имеется m предприятий, которые должны производить k продуктов. Общий их объем производства задан положительным вектором $s[1:k]$. t-е предприятие имеет конечное множество Q_t допустимых планов производства. Каждый план $q \in Q_t$ характеризуется вектором выпуска $a_q[1:k]$ и затратами $c(t, q)$. Требуется найти такой набор $\{q(t)\}, q(t) \in Q_t$, для которого

$$\sum_{t \in 1:m} a_{q(t)}[1:k] = s[1:k]$$

и достигает минимума $\sum_{t \in 1:m} c(t, q(t))$.

Будем решать эту задачу методами динамического програм-

мирования. Обозначим через $v(t,\sigma)$ минимум затрат на производство в объеме $\sigma[1:k]$ первыми t предприятиями. Тогда $v(t,\sigma) = min\{c(t,q) + v(t-1,\sigma-a_q) \mid q \in Q_t\}$. При этом

$$v(0,\sigma) = \infty \quad \text{при} \quad \sigma \neq 0, \quad v(0,0) = 0.$$

Служба dq для этой задачи особого интереса не представляет. Опишем для нее вариант службы inf, использующий внешнюю память (набор прямого доступа в смысле ОС ЕС).

```
IMPLEMENTATION  MODULE  inf;
  IMPORT  bdam;
   (* импортируется модуль, реализующий метод прямого доступа с операциями open  для инициализации, create
      для образования новой задачи и search  для поиска записи *)
  TYPE KEY = ARRAY  0..6s  OF CHAR;
       ELEM = RECORD
                ky : KEY;
                min : REAL
              END;
       VAR   el : ELEM;
  PROCEDURE   stokey (s : STATE;
                      VAR k : KEY );
   (* зависит от конкретного типа STATE  *)
   PROCEDURE  opinf;
     BEGIN   bdam. open (64, 72)
   (* указываем длину ключа и длину записи *)
   (* далее следует запись терминальных точек *)
  END  opinf;
  PROCEDURE  save (s : STATE;
                   v: REAL );
     BEGIN
          stokey (s, el.ky);
          el.min := v;
          bdam.create (el)
  END  save;
 PROCEDURE  find (s : STATE;
                  v : REAL;
                  b : BOOLEAN );
            VAR   k : KEY;
```

```
BEGIN
        stokey (s, k);
        bdam.search (k, el, b);
        IF  b  THEN  v := el.min  END
END find
END inf.
```

9. Перебор k -оптимальных решений задачи динамического программирования. В некоторых случаях полезно иметь возможность перебирать траектории в порядке ухудшения значений целевой функции. Каскадный метод организации вычислений легко приспосабливается к этой задаче.

Начнем с уравнения Беллмана для k-го оптимального решения задачи динамического программирования. Пусть $v(s,k)$ - величина затрат на траектории, начинающейся в состоянии s и k-й по качеству. Пусть $\varkappa(q,k)$, где $q \in Q_s$ ("степень срабатывания" локального решения q) - целочисленная положительная характеристика, которая будет определена ниже, причем

$$\varkappa(q,1) \equiv 1 \quad , \quad q \in Q_s \quad , \quad s \in S . \tag{I2}$$

Рассмотрим уравнения

$$v(s,1) = g(s) \quad , \quad s \in S_+ \quad ; \tag{I3}$$

$$v(s,k) = +\infty \quad , \quad s \in S_+ \quad , \quad k > 1 \quad ; \tag{I4}$$

$$v(s,k) = \min \{ c(q) + v(j(q), \varkappa(q,k)) \mid q \in Q_s \}$$

$$\left(= \min_{d \in \ell} c(g(s,k)) + v(j(g(s,k)), \varkappa(g(s,k),k)), s \in S \setminus S_+ \right); \tag{I5}$$

$$\varkappa(q,k+1) = \begin{cases} \varkappa(q,k) + 1 & , \ q = g(s,k) \ ; \\ \varkappa(q,k) \text{ в противном случае.} \end{cases} \tag{I6}$$

ТЕОРЕМА. Значение $v(s,k)$, определенное из уравнений (I2)-(I6), является k-оптимальным решением уравнения Беллмана.

ДОКАЗАТЕЛЬСТВО. Очевидно, что для $s \in S_+$ теорема доказана. Она очевидно верна и для $k=1$, так как в этом случае (I5) превращается в обычное уравнение Беллмана ввиду (I2). Пусть теперь $s \in S \setminus S_+$. В силу уравнения (I5)

$$v(s,k-1)=\min\{c(q)+v(j(q),\varkappa(q,k-1))\mid q\in Q_s\}$$
$$\Big(\underset{def}{=} c(\varsigma(s,k-1))+v(j(\varsigma(s,k-1)),\varkappa(\varsigma(s,k-1),k-1))\Big).$$

Тогда, согласно (I5) и (I6),

$$v(s,k)=\min\{c(q)+v(j(q),\varkappa(q,k))\mid q\in Q_s\}=$$
$$=\min\{c(\varsigma(s,k-1))+v(j(\varsigma(s,k-1)),\varkappa(\varsigma(s,k-1),k-1)+1),$$
$$c(q)+v(j(q),\varkappa(q,k-1))\mid q\in Q_s\setminus\varsigma(s,k-1)\}.$$

Отметим, что $v(s,k-1)$ достигает минимума на значении $q=\varsigma(s,k-1)$, поэтому

$$c(q)+v(j(q),\varkappa(q,k-1))\geqslant v(s,k-1),\quad q\in Q_s\setminus\varsigma(s,k-1).$$

Кроме того,

$$c(\varsigma(s,k-1))+v(j(\varsigma(s,k-1)),\varkappa(\varsigma(s,k-1),k-1)+1)\geqslant$$
$$\geqslant c(\varsigma(s,k-1))+v(j(\varsigma(s,k-1)),\varkappa(\varsigma(s,k-1),k-1))=v(s,k-1),$$

отсюда получаем неравенство

$$v(s,k)\geqslant v(s,k-1)\ ,\quad s\in S\setminus S_+\ ,\quad k>1.$$

Теорема доказана.

I0. Задача о кратчайшем пути. Рассмотрим ориентированный граф $\langle S,U\rangle$, не имеющий контуров. Пусть каждой дуге $u\in U$ сопоставлено неотрицательное число $c(u)$ – длина этой дуги. Заданы две вершины s_0 и $s_\ell\in S$. Требуется найти все пути из s_0 в s_ℓ в порядке возрастания длины. Обозначив через $v(s,k)$ минимальную длину пути при начальной вершине (состоянии) s, где k – индекс пути, мы получим функциональное уравнение Беллмана.

$$v(s,k)=\min\{c(u)+v(j(u),\varkappa(u,k))\mid u\in U_s^+\}\ ,\qquad (I7)$$

где значение $\varkappa(u,k)$ определяется аналогично (I2), (I6), причем $v(s_\ell,0)=0$, $v(s_\ell,k)=\infty$ при $k>1$.

При последовательном поиске k -оптимальных путей от s_0 до s_ℓ требуются только достигнутые значения величин $\varkappa(u)$. Поэтому для их хранения достаточно массива $\varkappa[U]$, элементы

которого первоначально принимаются равными I, а в дальнейшем, если на некотором u_0 достигается минимум при вычислении $v(s,k)$ в (I7), то $\varkappa[u_0]$ увеличивается на I.

Принципиальная схема вычислений выглядит таким образом:

```
I.   v : proc (s, k) returns (real);
2.        if< v(s,k) уже вычислено >  then min:=v(s,k);
3.           else   min:= infinity;
4.            loop  d  ч(s);
5.                  ctq:= c(d) + v(j(d), kappa(d));
6.                  if  min >ctq  then
7.                       min:=ctq; dmin:=d; end; end;
8.            Kappa (dmin):=kappa (dmin) + I;
9. <запомнить тройку (s, k, min )>;
I0.         end;
II.      return (min);
I2.   end V;
```

II. Второй подход к задаче о ранце. К задаче линейного раскроя очень близка задача о ранце, в которой на переменные x_j накладывается более жесткое условие $x_j \in 0:1$. В этой задаче применяется та же техника, и мы не будем ее рассматривать, перейдя прямо к более сложной задаче, называемой обычно многомерной задачей.

Пусть $b[M], c[N]$ – положительные векторы, $a[M,N]$ – неотрицательная матрица. Требуется найти вектор $x[N]$, удовлетворяющий условиям

$$a[M,N] \times x[N] \le b[M] \ , \qquad x[j] \in 0:1 \ , \ j \in N \tag{I8}$$

и максимизирующий

$$c[N] \times x[N] \ . \tag{I9}$$

Упростим эту задачу, заменив ограничения (I8) одним ограничением

$$(y[M] \times a[M,N]) \times x[N] \le y[M] \times b[M] \ , \tag{20}$$

где $y[M]$ – любой неотрицательный вектор. Положив

$$\gamma[N] = y[M] \times a[M,N] \ ,$$

$$\alpha = y[M] \times b[M] \ ,$$

получим следующую задачу.

Найти максимум (I9) при условии

$$\gamma[N] \times x[N] \in \mathcal{A}, \qquad x[j] \in O:1 \quad , j \in N. \tag{21}$$

Решим эту задачу методом динамического программирования. Обозначим через $v(k,s,\delta)$ k-е максимальное значение целевой функции на состоянии (s,δ). Тогда для этой задачи уравнение Беллмана выглядит следующим образом:

$$v(k,s,\delta) = \max\{v(x^0(k,s,\delta), s-1, \delta), c[s] + v(x^1(k,s,\delta), s-1, \delta-\gamma[s])\}, \tag{22}$$

где $x^i(i,s,\delta)=i$ для (s,δ), $i \in O,1$. Пусть $v(k,s,\delta)$ достигает максимума на (s^*,δ^*)

$$x^i(k+1,s,\delta) = \begin{cases} x^i(k,s,\delta)+1 & \text{при } (s,\delta)=(s^*,\delta^*); \\ x^i(k,s,\delta) & \text{в противном случае,} \end{cases}$$

$i \in O:1$.

Будем последовательно искать k-оптимальные решения задачи (I9), (2I). Первое из этих решений, которое будет удовлетворять условиям (I8), будет, очевидно, оптимальным решением исходной задачи (I8),(I9).

Таким образом, каскадный метод дает не только хорошую возможность решения задачи динамического программирования, но и позволяет использовать те же подходы в более сложных задачах.

ЛИТЕРАТУРА

I. Беллман Р. Динамическое программирование. – М.: ИЛ, I960.

2. Беллман Р., Дрейфус С. Прикладные задачи динамического программирования. – М.: Наука, I965.

3. Bellman R. On computation problems in the theory of dynamic programming // Symposium on Numerical Methods / Amer. Math. Soc. – Santa Monica, I953.

4. Ховард Р. Динамическое программирование и марковские процессы. – М.:Сов. радио, I965.

5. Михалевич В.С. Последовательные алгоритмы оптимизации и их применение. Ч.I// Кибернетика. – I965. – №I. – С.45–56;
Ч.2// Кибернетика. – I965. – №2. – С.85–89.

6. Knuth D.E., Plass M. Breaking paragraphs into lines. - Software-Practice & Experience, I98I.
7. Dijkstra E.W. A note on two problems in connection with graphes // Numerische Math. - I959. - T.I. - P.269-27I.
8. Итоги конкурса программ построения матрицы кратчайших расстояний / Гарусов В.Н., Надольская Т.А., Романовский И.В., Тихомиров В.П., Черкасский Б.В. // Экономика и мат. методы. - I985. - Т.2I, № 3. - С.565-567.
9. Denardo E.V., Fox B.L. Shortest-route methods. I. Reaching, pruning and buckets // Operat. Res. - I979. - V.27, №I.- P.I6I-I66.
I0. Романовский И.В. Решение задачи гильотинного раскроя методом переработки списка состояний // Кибернетика. - I969. - № I. - С.I02-I03.
II. Романовский И.В. Алгоритмы решения экстремальных задач.- М.: Наука, I977.
I2. Parnas D.L. On the criteria to be used in decomposing systems into modules // Comm. ACM. - I972. - V.I5, №I2.- P.I053-I058.
I3. Воденин Д.Р., Романовский И.В. Программирование в терминах служб //Кибернетика. - I979. - № 5. - С.70-75.
I4. Романовский И.В. Анализ понятия "служба" // Кибернетика. - I98I. - № 4. - С.66-72.
I5. Вирт Н. Модула-2 // Языки программирования. - М.: Наука, I985.
I6. Канторович Л.В., Залгаллер В.А. Рациональный раскрой промышленных материалов. - Новосибирск: Наука, I97I.
I7. Емеличев В.А. Дискретная оптимизация. Последовательностные схемы решения. Ч.I // Кибернетика. - I97I. №6. - С.I09-I2I; Ч.2 // Кибернетика. - I972. - № 2. - С.92-I03.

Поступила в ред.-изд. отдел
06.05.I987 г.