«Теория принятия решений»

Тема 4. **Оптимизация решений**

Лекция. **Динамическое программирование**

Цель: Показать сущность задач динамического программирования и дать основные методы их решения.

Время - 2 часа

Учебные вопросы:

1. *Многошаговый процесс принятия решений*
2. *Алгоритмы решения задач динамического программирования*
3. *Марковские процессы принятия решений*

Введение. Динамическое программирование связано с многошаговыми процессами принятия решений. Многошаговый процесс принятия решений можно определить как деятельность, при которой принимаются последовательные решения, направленные на достижение цели, Многошаговые процессы принятия решений встречаются в различных ситуациях. Динамическое программирование можно определить как набор математических процедур, используемых при анализе многошаговых процессов принятия решений.

1. Многошаговый процесс принятия решений

Динамическое программирование представляет собой математический аппарат, разработанный с целью повышения эффективности вычислений при решении некоторого класса задач математического программирования путем их разложения (декомпозиции) на относительно небольшие и, следовательно, менее сложные подзадачи. Характерным для динамического программирования является подход к решению задачи по **этапам**, с каждым из которых ассоциирована одна управляемая переменная. Набор рекуррентных вычислительных процедур, связывающих различные этапы, обеспечивает получение оптимального решения ***в целом*** при достижении последнего этапа.

Фундаментальным принципом, положенным в основу теории динамического программирования, является **принцип оптимальности**. По существу, он определяет порядок поэтапного решения допускающей декомпозицию задачи с помощью рекуррентных вычислительных процедур.

Расчеты на некотором этапе осуществляются на базе *сводной информации* о максимальном выигрыше, полученном на ***всех*** предыдущих этапах. при этом отдельные решения, найденные на предшествующих этапах, не представляют существенного интереса. Действительно, все последующие решения строятся некоторым оптимальным образом независимо от решений, полученных на предшествующих этапах. Это отражает сущность принципа оптимальности.

Пусть имеется некоторая операция, которую можно разделить на m этапов. Эффективность операции оцениваем некоторой функцией Φ, которую назовем «выигрышем». Положим, что выигрыш за всю операцию складывается из выигрышей на отдельных этапах: $\Phi = \sum_{i=1}^{m} \varphi_i$, где φ_i - выигрыш на ***i***-м этапе (***i***=1,...,***m***).

Операция представляет собой управляемый процесс, то есть предполагается, что можно выбирать какие-то параметры, влияющие на ход и исход каждого этапа и операции в целом. Поэтому на каждом этапе следует выбирать такие параметры операции, которые влияют не только на выигрыш на данном этапе, но и на суммарный выигрыш за операцию.

Таким образом, задача динамического программирования может быть записана в виде $\Phi = \sum_{i=1}^{m} \varphi_i(\vec{x}_i) \Rightarrow \max_{x \in G}(\min)$, где $\vec{x}_i$-управление на ***i***-м шаге.

Каждый многошаговый процесс принятия решений представляет собой развитие следующей задачи: найти кратчайший (или самый длинный) путь в направленной ациклической сети.

Опишем направленную сеть. Она состоит из множества узлов ***S*** и подмножества ***T*** множества ***S***×***S***. Элементы подмножества ***T*** называют *дугами* (ребрами) сети. Наличие дуги (***i***,,***j***) указывает на возможность движения от узла ***i*** к узлу ***j***. *Путем* (i_1,i_2,...,i_n) называют *конечную последовательность узлов*, таких, что (i_k,i_{k+1}) является направленной дугой для ***k***=1,2,...,***n***-1. Путь у которого $i_1 = i_n$, $n \geq 2$ называется циклом. Сеть называется *циклической*, если она содержит один или более циклов. В противном случае сеть называется *ациклической*.

В направленной ациклической сети всегда можно пометить узлы целыми числами от 1 до N таким образом, что для каждой дуги (***i***,,***j***) справедливо неравенство ***i*** < ***j*** Для примера рассмотрим сеть, изображенную на рис.1.

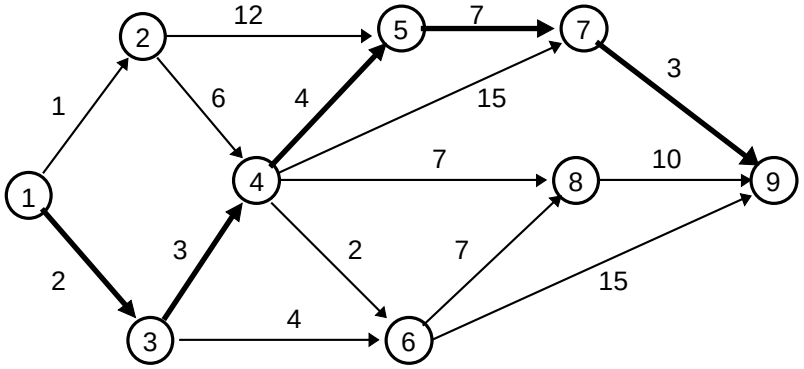

Рис.1. Графическое изображение задачи о кратчайшем пути

У каждой дуги (***i***,***j***) указана ее длина t_{ij}. В качестве длины пути выступает сумма длин входящих в него дуг.

Пусть f_i - длина кратчайшего пути от узла 1 до узла ***i***. По определению f_1=0. Из определения f_i следует, что f_i+ t_{ij} равняется длине кратчайшего пути от узла 1 до узла f_j при условии, что последней дугой пути является дуга (***i***,***j***). Кратчайший путь от узла 1 до узла ***j*** должен содержать некоторую дугу в качестве конечной, и поэтому

$$f_j = \min_{i|(i,j)} \left\{ f_i + t_{ij} \right\}.$$

Говорят, что дуга (***i***,***j***) *выходит* из узла ***i*** и *входит* в узел ***j***. Отметим, что для всех дуг (***i***,***j***), входящих в узел j, имеет место неравенство ***i*** < ***j***. Последнее означает, что приведенное выражение можно использовать для последовательного вычисления f_j при ***j***=2,3,...,N. Например, дуги (4,7) и (5,7) входят в узел 7, и f_7 можно вычислить по этому соотношению, как только станут известны f_4 и f_5.

Такое положение позволяет суммировать рассуждения в виде вычислительных процедур (алгоритмов).

2. Алгоритмы решения задач динамического программирования

Алгоритм прямой прогонки.

Шаг 1. Положить v_1=0 и v_k=∞ для ***k***=2,...,N (9). Положить ***i***=1.

Шаг 2. Для каждой дуги (***i***,***j***), исходящей из узла ***i***, положить

$$v_j \leftarrow \min\left\{ v_j, v_i + t_{ij} \right\}.$$

Шаг 3. Остановиться, если ***i***=N-1(8). В противном случае ***i***←***i***+1 и вернуться на шаг 2.

Пример реализации алгоритма прямой прогонки для задачи (рис.1).

Значение i	Дуга	Длина пути v_j
1	(1,2)	$v_2=(v_1+t_{12})=(0+1)=1$ *
	(1,3)	$v_3=(v_1+t_{13})=(0+2)=2$ *
2	(2,4)	$v_4=(v_2+t_{24})=(1+6)=7$
	(2,5)	$v_5=(v_2+t_{25})=(1+12)=13$
3	(3,4)	$v_4=(v_3+t_{34})=(2+3)=5$ *
	(3,6)	$v_6=(v_3+t_{36})=(2+4)=6$ *
4	(4,5)	$v_5=(v_4+t_{45})=(5+4)=9$ *
	(4,6)	$v_6=(v_4+t_{46})=(5+2)=7$
	(4,7)	$v_7=(v_4+t_{47})=(5+15)=20$
	(4,8)	$v_8=(v_4+t_{48})=(5+7)=12$ *
5	(5,7)	$v_7=(v_5+t_{57})=(9+7)=16$ *
6	(6,8)	$v_8=(v_6+t_{68})=(6+7)=13$
	(6,9)	$v_9=(v_6+t_{69})=(6+15)=21$
7	(7,9)	$v_9=(v_7+t_{79})=(16+3)=19$ *
8	(8,9)	$v_9=(v_8+t_{89})=(12+10)=22$

Длина кратчайшего пути (9-7-5-4-3-1) = 19.

Алгоритм носит название *прямой прогонки* поскольку вычисления ведутся в естественном порядке следования этапов

Алгоритм обратной прогонки.

Шаг 1. Положить $v_1=0$ и $v_k=\infty$ для $k=2,...,N$ (9). Положить $j=2$.

Шаг 2. Для каждой дуги (i,j), входящей в узел j, положить

$$v_j \leftarrow \min\left\{ v_j, v_i + t_{ij} \right\}.$$

Шаг 3. Остановиться, если $j=N$ (9). В противном случае $j \leftarrow j+1$ и вернуться на шаг 2.

Пример реализации алгоритма обратной прогонки для задачи (рис.1).

Значение j	Дуга	Длина пути v_j
2	(1,2)	$v_2=(v_1+t_{12})=(0+1)=1$ *
3	(1,3)	$v_3=(v_1+t_{13})=(0+2)=2$ *
4	(2,4)	$v_4=(v_2+t_{24})=(1+6)=7$
	(3,4)	$v_4=(v_3+t_{34})=(2+3)=5$ *
5	(2,5)	$v_5=(v_2+t_{25})=(1+12)=13$
	(4,5)	$v_5=(v_4+t_{45})=(5+4)=9$ *
6	(3,6)	$v_6=(v_3+t_{36})=(2+4)=6$ *
	(4,6)	$v_6=(v_4+t_{46})=(5+2)=7$
7	(4,7)	$v_7=(v_4+t_{47})=(5+15)=20$
	(5,7)	$v_7=(v_5+t_{57})=(9+7)=16$ *
8	(4,8)	$v_8=(v_4+t_{48})=(5+7)=12$ *
	(6,8)	$v_8=(v_6+t_{68})=(6+7)=13$
9	(6,9)	$v_9=(v_6+t_{69})=(6+15)=21$
	(7,9)	$v_9=(v_7+t_{79})=(16+3)=19$ *
	(8,9)	$v_9=(v_8+t_{89})=(12+10)=22$

Длина кратчайшего пути (9-7-5-4-3-1) = 19.

Алгоритм обратной прогонки позволяет начинать расчеты в любом конце сети и двигаться в *обратном* направлении к другому концу сети.

Шаг 1. Положить $v_N=0$ и $v_k=\infty$ для $k=N-1,...,1$. Положить $j=N$.

Шаг 2. Для каждой дуги (i,j), входящей в узел j, положить

$$v_j \leftarrow \min\left\{ v_j, v_i + t_{ij} \right\}.$$

Шаг 3. Остановиться, если $j=2$. В противном случае $j \leftarrow j$ -1 и вернуться на шаг 2.

Пример реализации алгоритма обратной прокрутки для задачи (рис.1).

Значение j	Дуга	Длина пути v_j
9	(6,9)	$v_6=(v_9+t_{69})=(0+15)=15^*$
	(7,9)	$v_7=(v_9+t_{79})=(0+3)=3$
	(8,9)	$v_8=(v_9+t_{89})=(0+10)=10$
8	(4,8)	$v_4=(v_8+t_{48})=(10+7)=17$
	(6,8)	$v_6=(v_8+t_{68})=(10+7)=17$
7	(4,7)	$v_4=(v_7+t_{47})=(3+15)=18$
	(5,7)	$v_5=(v_7+t_{57})=(3+7)=10$
6	(3,6)	$v_3=(v_6+t_{36})=(15+4)=19$
	(4,6)	$v_4=(v_6+t_{46})=(15+2)=17$
5	(2,5)	$v_2=(v_5+t_{25})=(1+12)=13$
	(4,5)	$v_4=(v_5+t_{45})=(10+4)=14^*$
4	(2,4)	$v_2=(v_4+t_{24})=(14+6)=20$
	(3,4)	$v_3=(v_4+t_{34})=(14+3)=17$
3	(1,3)	$v_1=(v_3+t_{13})=(17+2)=19^*$
2	(1,2)	$v_1=(v_2+t_{12})=(20+1)=21$

Длина кратчайшего пути (1-3-4-5-7-9) = 19.

Может возникнуть вопрос о целесообразности применения алгоритма обратной прогонки, особенно в связи с тем, что алгоритм прямой прогонки представляется более ясным и логичным. В обоих алгоритмах предусматривается одно сложение и одно сравнение на каждую дугу. Следовательно, оба алгоритма обладают одинаковым быстродействием.

Для рассмотренного примера вычислительные схемы алгоритмов прямой и обратной эквивалентны. Однако возможны ситуации, когда различие между алгоритмами оказывается существенным Это проявляется при решении задач пошагового принятия решений, в которых нумерация этапов, соответствующих последовательным промежуткам времени, осуществляется в хронологическом порядке.

Опыт практического применения методов динамического программирования показывает, что процедура обратной прогонки более эффективна. В большинстве научных работ, посвященных динамическому программированию, изложение ведется в соответствии с методом обратной прогонки независимо от вычислительных аспектов. Алгоритм прямой прогонки используется лишь для сравнения, а также в случаях, когда он имеет явное преимущество перед алгоритмом обратной прогонки.

Возвращаясь к рассмотренному примеру, следует отметить, что, пользуясь обоими алгоритмами, легко найти самый длинный путь в ациклической направленной сети: следует заменить min на max в расчетных соотношениях и на первом шаге алгоритмов положить $-\infty$ вместо $+\infty$.

С математической точки зрения только что рассмотренная задача совершенно тривиальна. Несмотря на это, именно к ней сводятся многие задачи динамического программирования, возникающие в процессе исследования операций.

3. Марковские процессы принятия решений