31.10.25 **Динамическое программирование** Синицын Е.А.

Динамическое программирование впервые было описано Р. Беллманом в 1940-х г. для описания процесса нахождения решения задачи, где ответ на одну задачу м. б. получен только после реш. „предшест.“.

Динамическое программирование – один из разделов математического/оптимального програм-ия, в котором процесс принятия решения и управления может быть разбит на отдельные этапы (шаги).

Одним из основных методов ДП является метод рекуррентных соотношений, который основывается на использовании принципа оптимальности.

Постановка задачи. Функция Беллмана

S_i – состояние системы; $i=\overline{0,n}$

S_0 – начальное; S_n – конечное

$X=(x_1, x_2, \dots, x_n)$ – управление, переводящее систему из S_0 в S_n.

$F_k(S_{k-1}, x_k)$, где $k=\overline{1,n}$ – Функция Беллмана (ЦФ)

$$S_0 \xrightarrow[F_1(S_0,x_1)]{x_1} S_1 \xrightarrow[F_2(S_1,x_2)]{x_2} S_2 \longrightarrow \dots \longrightarrow S_{n-1} \xrightarrow[F_n(S_{n-1},x_n)]{x_n} S_n$$

Наша задача – подобрать X, переводящее систему за n шагов из S_0 в S_n, чтобы достичь экстремума ЦФ.

$$\boxed{Z(S_0, X) = F_1(S_0,x_1) + F_2(S_1,x_2) + \dots + F_n(S_{n-1},x_n) = \sum_{k=1}^{n} F_k(S_{k-1},x_k)}$$

Функциональное уравнение Беллмана

Ищем max ЦФ

шаг n: $Z_n(S_{n-1}) = \max\limits_{x_n} F_n(S_{n-1}, x_n)$

шаг $n-1$: $Z_{n-1}(S_{n-2}) = \max\limits_{x_{n-1}} \{F_{n-1}(S_{n-2}, x_{n-1}) + Z_n(S_{n-1})\}$

шаг $n-2$: $Z_{n-2}(S_{n-3}) = \max\limits_{x_{n-2}} \{F_{n-2}(S_{n-3}, x_{n-2}) + Z_{n-1}(S_{n-2})\}$

....

шаг k: $Z_k(S_{k-1}) = \max\limits_{x_k} \{F_k(S_{k-1}, x_k) + Z_{k+1}(S_k)\}$

То есть на k-м шаге надо так подобрать управление x_k, чтобы сумма выигрышей на k-м шаге $F_k(S_{k-1}, x_k)$ и на $n-k$ последующих шагах $Z_{k+1}(S_k(S_{k-1}, x_k)) \to max$

Общая схема решения задач ДП

- Определяем все состояния S_i, где $i = \overline{0, n}$
- Определяем вид ЦФ $F_k(S_{k-1}, x_k)$, где $k = \overline{1, n}$
- Определяем ф-ии перехода $S_k(S_{k-1}, x_k)$ из сост. S_{k-1} в S_k
- Из ур. Беллмана для $Z_k(S_{k-1})$ находим оптимальное управление на k-м шаге x_k^*. По S_0 и x_k^* определяем состояние системы после k-го шага: $S_k = S_k(S_{k-1}, x_k)$, $k = \overline{1, n}$

Многошаговый процесс „проходится" дважды:

1) от конца к началу → на всех шагах находятся условно оптимальное управление и условно оптимальный выигрыш.

2) от начала к концу → на всех шагах находятся оптимальные управления

Постановка задачи

∃ операция O представляет собой управляемый процесс, т.е. мы можем выбирать какие-то параметры, влияющие на его исход.

При чём на каждом шаге выбирается какое-то решение, от которого зависит выигрыш на данном шаге и выигрыш за операцию в целом.

Будем называть это решение „шаговым управлением"

$X = (x_1, x_2, \dots, x_n)$ — управление операцией.

Требуется найти управление X : $W = \sum_1^m w_i \to max$

$$W = \sum_1^m w_i$$

Суммарный выигрыш — Выигрыш на i-м шаге

Т.о. X^*: $W = \sum_1^m w_i$ достигает max, будем называть оптимальным управлением.

Оно состоит из совокупности оптимальных шаговых управлений: $W^* = \max\limits_x \{w(x)\}$

Свойства задач ДП:

1) Задача должна допускать интерпретацию как n-шаговый процесс принятия решений.

2) Задача должна быть определена для ∀ числа шагов и иметь структуру, не зависящую от их числа.

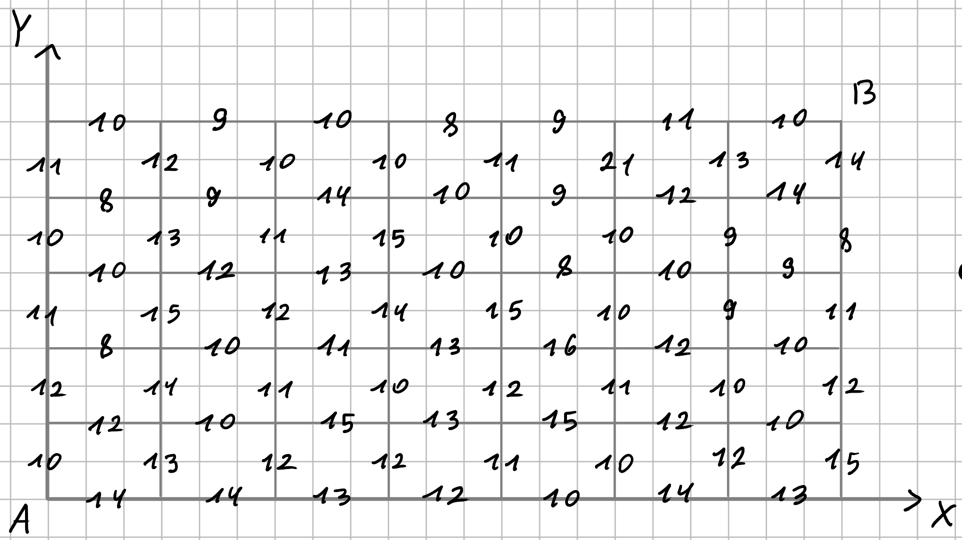
3) При рассмотрении k-шаговой задачи должно быть задано некоторое мн-во параметров, описывающих состояние системы, от которых зависят оптимальные значения переменных. При чём это мн-во не должно изменяться при увеличении числа шагов.

4) Выбор решения на k-м шаге не должен оказывать влияния на предыдущие решения, кроме необходимого пересчёта переменных.

Классификация задач ДП:

Практические	Классические
• Распределение средств м/у предприятиями. • Распределение ресурсов. • Замены оборудования. • Складская задача • Найма работников.	• О вычислении чисел Фибоначчи. • О редакционном расстоянии. • О порядке перемножения матриц. • О „рюкзаке" и т.д.

Пример. (прокладка наивыгоднейшего пути)



$S_i = (x, y)$

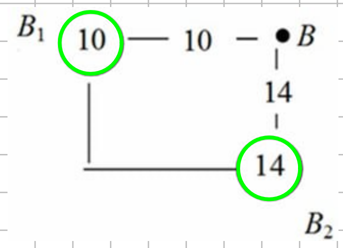
$a_i = \left[\begin{array}{l} \text{„с" – север} \\ \text{„в" – восток} \end{array} \right.$

↑ шаговое упр-ие

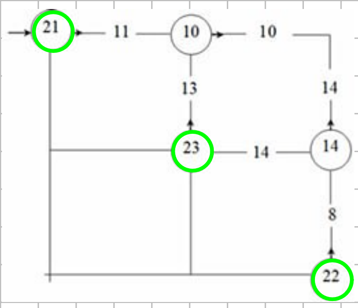
Процедуру условной оптимизации будем разворачивать от конца к началу.

Стоимость перемещения на:

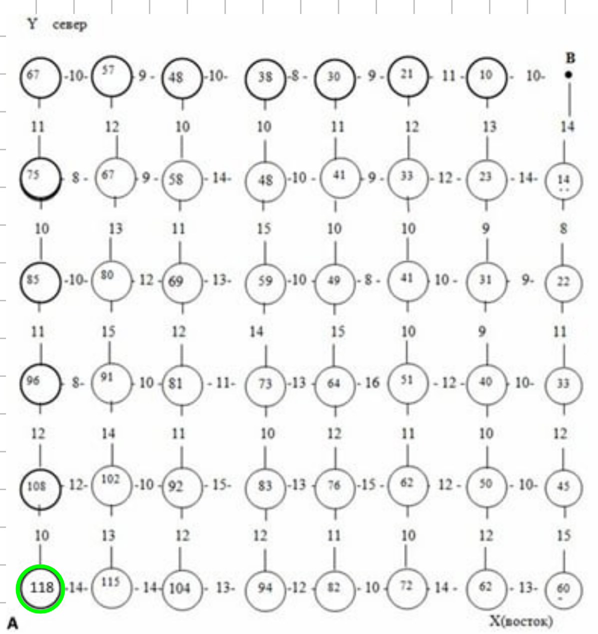
12-м шаге



11-м шаге



1-м шаге



Итого $W^* = 118$

Теперь строим безусловное оптимальное управление — траекторию, ведущую из A в B самым дешёвым способом.

Рекомендации по постановке задач ДП:

1) Выбрать параметры, характеризующие состояние S.
2) Расчленить операцию на этапы (шаги)
3) Выяснить набор шаговых управлений x_i
4) Определить выигрыш на i-м шаге: $W_i = f_i(S, x_i)$
5) Определить изменение состояния: $S_i = \varphi_i(S_{i-1}, x_i)$
6) Записать основное рекуррентное уравнение ДП
$W_i(S) = \max\{f_i(S, x_i) + W_{i+1}(\varphi_i(S, x_i))\}$
7) Провести условную оптимизацию последнего m-го шага
$W_m(S) = \max\{f_m(S, x_m)\}$
8) Провести условную оптимизацию шагов $\overline{m-1, 1}$ по формуле $W_i(S)$, где $i = \overline{m-1, 1}$, и для каждого из шагов указать условное оптимальное управление $x_i(S)$
9) Провести безусловную оптимизацию управления, „читая" соотв. рекомендации на каждом шаге.

Жадные алгоритмы.

Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.

Виды жадных алгоритмов:

Поиск кратчайшего пути:

- Алгоритм Дейкстры
- Алгоритм Флойда

Задача о минимальном покрывающем дереве:

- Алгоритм Прима
- Алгоритм Крускала

Сжатие информации:

- Алгоритм Хаффмена

Алгоритм Дейкстры:

1) Изначально все вершины получают метку a_i „∞", кроме начальной вершины с меткой нуль.
2) Вычисляется расстояние до каждого из соседей, как сумма текущей a_i и веса ребра до V_k.
   Если $a_i + L_{ik} < a_k \Rightarrow a_k = a_i + L_{ik}$
3) После прохода по всем соседям, текущая вершина отмечается как обработанная.
4) Ищется новая обрабатываемая вершина, ближайшая к уже обработанной, но ещё не использ-ая.
5) Обход till all вершины became обработанными.