Информационные технологии автоматизированного проектирования
Часть 1

Лектор: доцент кафедры ЭТТ БГУИР

Бондарик Василий Михайлович

Лекция 4
Методы математического программирования при автоматизации конструирования ЭА

1 Основные классы задач математического программирования. Линейное программирование в ИТАП

2 Нелинейное программирование в ИТАП

3 Целочисленное программирование в ИТАП

4 Динамическое программирование в ИТАП

Вопрос 1 Основные классы задач математического программирования. Линейное программирование в ИТАП

Основные классы задач математического программирования

1. Класс линейного программирования
2. Класс нелинейного программирования
3. Класс целочисленного программирования
4. Класс динамического программирования

Линейное программирование

Класс линейного программирования - используется в случае, когда выходные параметры системы можно записать в виде линейных равенств и неравенств (хотя бы в приближении – кусочно-линейными аналогами)

Математическая формулировка:

определить такие значения переменных X^*, удовлетворяющих системе ограничений, при которой достигается экстремум (максимум или минимум) целевой функции $F(X)$.

Прикладные задачи линейного программирования

Транспортная задача - (задача прикрепления поставщиков к потребителям) связана с нахождением наиболее рационального прикрепления пунктов отправления грузов к пунктам их назначения, при котором общая стоимость всех перевозок минимальна.

Задача о назначениях

частный случай транспортной задачи, который используется при проектировании ЭА

Задача о назначениях

Пусть имеется n вакантных видов работ, на которые претендует n работников, причем на каждом виде работ может быть использован только один из n претендентов. Эффективность использования *i*-го работника на *j*-ом виде работ равна c_{ij}.

Можно построить квадратную матрицу C_{ij} , в которой каждый элемент *i*-й строки соответствует эффективности использования *i*-го работника на каждом из n видов работ, а в *j*-ом столбце записывается эффективность использования каждого из n работников на *j*-ом виде работ.

Требуется так распределить n работников на n вакантных видах работ, чтобы суммарная эффективность их использования была максимальной.

Задача о назначениях

Определим матрицу

где

Необходимо максимизировать функцию *F*

При ограничениях

(на каждом j-м виде работ может использоваться только один работник); за каждым i-м работником может быть закреплено только одно вакантное место)

Задача о назначениях

Требование целочисленности переменных Xji, усложняет решение задачи. Однако доказано, что в задаче о назначениях условие Xij = {0, 1} можно заменить на более простое Xij ≥ 0 (обычная транспортная задача), но в результате автоматически приходят к целочисленным значениям Xij.

Особенностью такой задачи является то, что в ней все значения матрицы = 1. Это позволяет решать ее с помощью более простого алгоритма (**венгерский метод**) по сравнению с методами, применяемыми при решении общей транспортной задачи.

При проектировании ЭА – задача размещения конструктивных элементов

a_{ij} – элементы матрицы смежности, d_{ij} – элементы матрицы расстояний

Симплекс-метод

осуществляется направленное движение по опорным планам до получения оптимального решения

Алгоритм:

1) По определенному правилу находим какую-либо вершину, принадлежащую множеству допустимых решений. Проверяем, не соответствует ли данная вершина оптимальному значению целевой функции. Если да, то задача решена.

2) Если задача не решена, то проверяем, нельзя ли на данном шаге утверждать, что целевая функция не ограничена сверху (снизу) на множестве допустимых решений при отыскании максимума (минимума) функции. Если да, то задача не имеет решения.

3) Если задача имеет решение, то находим новую вершину, в которой целевая функция имеет более оптимальное значение. Далее решение осуществляем в соответствии с пунктом 1, принимая в качестве исходной вновь выбранную вершину.

Симплекс-метод

Алгоритм:

Для решения задачи линейного программирования этим методом необходимо, чтобы ее математическая модель была задана в канонической форме (y + Ax = B), т.е. все переменные должны быть положительными, а ограничения иметь вид равенств.

Если этого нет, то изменением начала координат всегда можно добиться положительности всех переменных, а за счет введения дополнительных неосновных переменных yi перейти от неравенств к равенствам.

Венгерский метод

Разработал венгерский математик Е. Эгервари задолго до возникновения теории линейного программирования

ограничения

1. Задачи линейного программирования эквивалентны, если их оптимальные управляемые параметры совпадают.
2. Преобразования, переводящие задачу линейного программирования в эквивалентную задачу называются эквивалентными.
3. Две матрицы C и D эквивалентны, если

α = const, β = const,

т.е. если к *i* строке, либо *j* столбцу прибавить любое постоянное число, то получится эквивалентная исходной матрица

Венгерский метод

Основан на трансформации *системы независимых нулей.*

Систему нулевых элементов матрицы, обладающую тем свойством, что никакая пара из них не лежит в одной строке или одном столбце, называют системой независимых нулей.

Как только число независимых нулей становится равным n, задача назначения считается решенной: **оптимальный план определяется местоположением независимых нулей в последней из матриц, эквивалентных начальной**.

Венгерский метод. Алгоритм

1. Подготовительный этап

Для каждого из столбцов j исходной матрицы C_{ij} эффективности отыскивается максимальный элемент

Новая матрица C* формируется по следующему правилу: каждый элемент c^*_{ij} матрицы C* равен разности между максимальным элементом d_j данного **столбца** исходной матрицы и соответствующим элементом c_{ij}:

В новой матрице C*, состоящей из неотрицательных элементов, в каждом столбце по крайней мере один нуль.

Венгерский метод. Алгоритм

в каждой **строке** матрицы C^* отыскивается минимальный элемент t_i, который вычитается из всех элементов соответствующей строки матрицы. В результате получаем эквивалентную матрицу C_o, элементы которой подсчитываются по формуле

Венгерский метод. Алгоритм

Образуем первоначальную **систему независимых нулей.**

С этой целью отыскиваем и помечаем звездочкой произвольный нуль в первом столбце матрицы Со. Такой нуль обязательно найдется. Просматриваем элементы второго столбца, и если обнаруживаем нуль, не лежащий в одной строке с ранее отмеченным звездочкой нулем, то его также помечаем звездочкой. Такие операции осуществляем для всех последующих столбцов матрицы.

Полученная система помеченных звездочкой нулей является исходной для последующего решения задачи и содержит по крайней мере два независимых нуля.

На этом подготовительный этап заканчивается.

Венгерский метод. Алгоритм

2. Подсчитываем число независимых нулей (k) полученной матрицы

3. Если k = n, то решение задачи получено, в противном случае переходим к блоку **4**.

4. Столбцы, содержащие нуль со звездочкой, выделим знаком плюс. Так как k < n, то не все столбцы матрицы оказываются выделенными.

5. Проверяем, есть ли среди невыделенных столбцов матрицы хотя бы один нуль. Если да, то переходим к блоку **6**, в противном случае — к блоку **7**.

6. Проверяем, содержит ли строка с невыделенным нулем также и нуль со звездочкой. Если да, то переходим к блоку **8**, в противном случае — к блоку **9**.

Венгерский метод. Алгоритм

7. Формируем новые невыделенные нули. Для этого среди невыделенных элементов матрицы C* выбираем минимальный и вычитаем его из элементов, расположенных в невыделенных строках, и прибавляем к элементам, лежащим в выделенных столбцах. Получаемая при этом матрица Ck* является эквивалентной матрице C*. Переходим к блоку **6**.

8. Найденный невыделенный нуль отмечаем штрихом (0'), а содержащую этот нуль строку — знаком «+». Снимаем знак выделения «+» над столбцом, в котором расположен нуль со звездочкой, лежащий в только что выделенной строке. Переходим к блоку **5**.

9. Невыделенный нуль отмечаем (0').

10. Подготавливаем информацию для оценки возможности увеличения числа независимых нулей в матрице эффективности (нулей со звездочками).

Венгерский метод. Алгоритм

Для этого, начиная с нуля со штрихом, в одной строке с которым нет нуля со звездочкой, осуществляем построение цепочки элементов матрицы C* по следующему правилу:
выбираем исходный нуль со штрихом (0'); в цепочку включаем нуль со звездочкой (0*), лежащий с 0' в одном столбце (если такой найдется); к нему прибавляем опять нуль со штрихом (0'), лежащий в одной строке с предшествующим 0*, и т.д. Построение цепочки нулей по описанному правилу осуществляется однозначно. Число нулей в такой цепочке всегда нечетно, причем 0' находятся на нечетных местах, а 0* — на четных. Может случиться, что в одном столбце с исходным 0' нет 0*. Тогда цепочка нулей получается вырожденной и состоит из одного исходного элемента.

Венгерский метод. Алгоритм

11. Меняем знаки у нулей в построенной цепочке, причем звездочки уничтожаем, а штрихи заменяем на звездочки. Так как такая цепочка обязательно должна заканчиваться 0', а число элементов в ней нечетно, то при замене 0' на 0* происходит увеличение числа последних на единицу.

Возвращаемся к блоку 2.

Венгерский метод. Пример

Задана исходная матрица эффективности

где Ai - рабочие; Bi - различные виды работ. Каждый элемент матрицы соответствует тому эффекту, который может быть получен при использовании конкретного рабочего на определенном виде работ.

Требуется так распределить рабочих по местам, чтобы общая эффективность их использования была max, т. е. необходимо решить задачу выбора или назначения для матрицы С.

Венгерский метод. Пример

Подготовительный этап.

Отыскиваем максимальные элементы в каждом из столбцов матрицы C

Вычитаем каждый элемент матрицы C из максимального элемента соответствующего столбца: получаем матрицу C*

Венгерский метод. Пример

Подготовительный этап.

Отыскиваем минимальные элементы в каждой из строк матрицы C*:

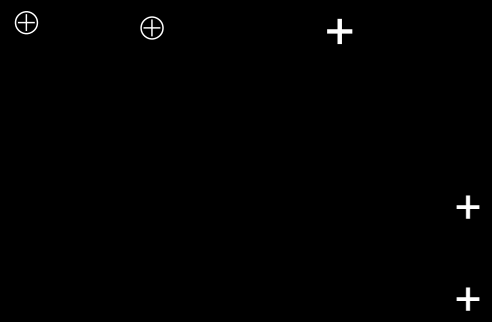
A1 =0, A2 = 4, A3 = 0, A4 = 4.

Вычитаем из каждого элемента матрицы C* минимальный элемент соответствующей строки: находим матрицу Co:

Примечание. Операция уничтожения знака «+» над столбцом содержащим 0* условно обозначается обведением этого знака кружком

Венгерский метод. Пример

Образуем первоначальную систему независимых нулей, отмечая их звездочками:



Первая итерация. Подсчитываем число независимых нулей **k**. Так как **k = 3 < n = 4**, то выделяем столбцы матрицы, содержащие 0* (1, 2, 4)

Среди невыделенных элементов матрицы Co (лежат в столбце j = 3) имеется невыделенный нуль (c_{33}). В строке t = 3, где находится невыделенный нуль имеется 0*.

Поэтому переходим к блоку 8.

В результате преобразований блока 8 получаем матрицу Co:

Венгерский метод. Пример

Просматриваем нули первого столбца. Нуль на месте C_{31} выделять нельзя, так как он находится в ранее выделенной строке (строка помечена знаком «+». Нуль на месте c_{41} можно выделить, пометив его штрихом. Так как в четвертой строке матрицы Co нет нуля со звездочкой, то выделение нулей на этом заканчивается (нет больше непомеченных столбцов, в которых можно было бы искать невыделенные нули). Получаем матрицу :

⊕ ⊕ +

+

+

Венгерский метод. Пример

Построение цепочки нулей начинаем с последнего выделенного нуля со штрихом. Такой нуль находится на месте c_{41}. От этого нуля строим дугу в том же столбце матрицы Co к нулю со звездочкой, который располагается на месте c_{21}. От него по той же строке строим дугу цепочки до c_{22}, далее по столбцу к нулю со звездочкой, находящимся на месте c_{32}, а затем по строке к нулю со штрихом на месте c_{33}.

Здесь цепочка обрывается и ее построение заканчивается. В результате цепочка нулей состоит из следующей последовательности элементов матрицы

Венгерский метод. Пример

Уничтожаем пометки нулей со звездочками, нули со штрихами помечаем звездочками. Убираем все пометки строк и столбцов матрицы Co:

Подсчитываем число независимых нулей (со звездочками). Так как **k = 4 = n**, то задача решена, т. е. получаем оптимальный план, который можно представить в виде матрицы X*:

Венгерский метод. Пример

Наложив этот план на значения исходной матрицы эффективности C, получим соответствующее значение целевой функции :

Вопрос 2 Нелинейное программирование в ИТАП

Нелинейное программирование

В задачах нелинейного программирования, в большинстве случаев, нелинейность связана с эмпирическими соотношениями, такими, как выход годной продукции, показатели качества изделий и т. п., а также с необходимостью учета случайных факторов, описываемых их функциями распределения.

Математическая формулировка:

определить такие значения переменных $X^* = \{x^*_1, x^*_2, ..., x^*_n\}$, удовлетворяющие системе $n + m$ ограничений

при которых достигается экстремум (максимум или минимум) целевой функции $F(X)$

Нелинейное программирование. Прикладные задачи

Задача выбора оптимальных параметров технологического процесса

В нелинейном программировании существует несколько методов оптимизации, например:

-Метод ГАУССА—ЗАЙДЕЛЯ

-Метод ГРАДИЕНТА

-Метод крутого восхождения (БОКСА—УИЛСОНА)

и др.

Будут подробнее рассмотрены в следующем семестре

Вопрос 3 Целочисленное программирование в ИТАП

Целочисленное программирование

все или часть переменных принимают только целочисленные значения

Математическая формулировка:

В общем виде задача целочисленного программирования формулируется аналогично задачам нелинейного программирования: требуется найти план X^*, соответствующий минимуму (максимуму) целевой функции n переменных $F(x)$ при ограничениях

x_i – целые числа

Целочисленное программирование

1 Метод отсечения

заключается в последовательном добавлении к исходной задаче линейных ограничений, которым удовлетворяют все целочисленные решения задачи, но не удовлетворяют нецелочисленные решения.

Путем отсечения на каждом шаге алгоритма оптимального нецелочисленного результата решение задачи целочисленного программирования сводится к решению последовательности задач линейного программирования.

Правила формирования ограничений были разработаны американским ученым Р. Гомори.

2 Комбинаторный метод

решение задачи сводится к направленному перебору.

Наиболее известными из этой группы являются метод ветвей и границ и различные его модификации, а также методы динамического дискретного программирования и последовательной оптимизации.

3 Приближенный метод

используются при решении задач большой размерности, для которых точные методы малоэффективны. Наиболее известным из этой группы является метод случайного поиска.

4 Метод статистической оптимизации (случайного поиска)

При использовании метода случайного поиска (метод Монте-Карло) производят случайный выбор плана задачи X и вычисляют значение целевой функции F(X). Новый план решения задачи выбирают произвольно и снова вычисляют значение F(X). Этот процесс многократно повторяют и из полученных планов решения задачи выбирают наилучший в смысле значения целевой функции.

Д: гарантирует нахождение глобального экстремума функции F(X),

Н: малоэффективен ввиду большой трудоемкости поиска (среднее число планов задачи, которые необходимо просмотреть, приближается к полному перебору).

Прикладные задачи

- задача о назначениях,
- задача о кратчайшем пути через n точек (задача о коммивояжере)
- задача о покрытии
- задача об укладке

Вопрос 4 Динамическое программирование в ИТАП

Динамическое программирование

Процесс поиска решения разбивается на отдельные этапы (шаги). На каждом шаге принимается одно из допустимого множества L решение, результатом которого является преобразование плана задачи. Управляющие операторы должны выбираться таким образом, чтобы максимизировать (минимизировать) F(N).

Преобразования, выполненные на последующих шагах, не должны оказывать никакого влияния на предыдущие шаги.

В основе теории динамического программирования лежит принцип оптимальности Р. Беллмана, согласно которому любой отрезок оптимальной траектории оптимален.

Любое правило поиска решения, которое дает допустимую последовательность решений, называют стратегией (политикой).

Динамическое программирование. Пример

Определить кратчайший путь между точками X1 и X2, соединенными сложной сетью (рис.). Длина каждой связи сети задана

Динамическое программирование. Решение

1. Разобьем весь процесс поиска на этапы.
2. Сначала найдем кратчайшие пути, соединяющие точки пересечения линии N — 1 с сетью и точку X2 **(этап IV).** Таких оптимальных путей два (отмечены на рис. штрихами).
3. Определим пути между точками пересечения N-2 и N-1 с сетью, которые приводят к минимальным суммарным путям, соединяющим точки на линии N- 2 и точку X2 **(этап III).**
4. В качестве продолжения данных путей необходимо выбирать пути, найденные на **IV этапе**. Таких оптимальных путей – 2.
5. На **этапе II** находим кратчайшие пути, соединяющие точки на линии N-3 и точку X2.
6. Перейдя к **этапу I**, находим минимальный суммарный путь из точки X1 в X2, длина которого равна **21**.

Динамическое программирование. Решение

Математически в примере решается следующее выражение:

где Q – управляющий оператор, с помощью которого осуществляется последовательный переход от одного состояния в другое, f – текущее значение (приращение) целевой функции на соответствующем этапе.

Из рассмотренного примера следует, что на каждом t-ом шаге перебираются не все пути, соединяющие точки на линии *N-i* и точку X2, а только оптимальные, уже отобранные на предыдущем шаге – преимущество динамического программирования.

При этом эффективность рекуррентного подхода оказывается огромной в случае, когда полный перебор практически неосуществим.

Динамическое программирование

Динамическое программирование является универсальным методом отыскания глобального экстремума в любых математических моделях, для которых справедлив принцип оптимальности Беллмана.

Н.: заключается в трудности сведения исходной задачи к математической модели, позволяющей использовать многошаговый процесс оптимизации, т. е. к модели, описываемой рекуррентными соотношениями.

Прикладная задача - волновой алгоритм Ли, положенный в основу многих известных программ трассировки печатных соединений.

Вопросы по прочитанному материалу?

Спасибо за внимание!