2016 Контрольная работа.

Тематики задач и способы их решения.

Составитель: *avasite*

Можно печатать 4 страницы на одну страницу А4.

То, что здесь приведено может местами различаться с точными определениями из теории, это не теормин, а описание того, как решаются конкретные задачи из курса.

1. Задачи из раздела «Антагонистические игры»
2. Антогонистические игры

***Седловая точка*** – это такая точка (i0, j0), что F(i, j0) <= F(i0, j0) <= F(i0, j), где F (i, j) – платёжная функция (может быть задана матрицей, а может быть задана функцией)

Iн = max (по i) min (по j) F(i, j) – нижняя цена игры, i0­ – оптимальная стратегия 1-го игрока

Iв = min (по j) max (по i) F(i, j) – верхняя цена игры, j0 – оптимальная стратегия 2-го игрока

* 1. Найти **все** седловые точки заданной функции F(x, y), x \in [a, b], y \in [c, d]

F(x, y) – имеет частные производные

Способы решения задачи:

* (плохой метод, лектор рассчитывает не на это)

Ищем седловые точки почти по определению.

Найдём нижнюю (maxmin) и верхнюю (minmax) цену игры, если они будут ==, то оптимальные стратегии, при которых они достигаются – будут являться седловыми точками.

* Поиск седловой точки в матрице:

Найдём нижнюю и верхнюю цену игры, если они будут равны, то оптимальные стратегии, при которых они достигаются – будут являться седловыми точками:

* + Обозначим звёздочкой максимальный элемент в каждом столбце
  + Обозначим кружочком минимальный элемент в каждой строке
  + Возьмём все точки в матрице, которые обозначены звёздочкой и кружочком
    - (**Если** Iв == Iн, то это цена игры, а все индексы в матрице, при которых она достигается являются оптимальными стратегиями.)
    - Если не равно, то этот метод нахождения седловой точки не подходит (кажется, тогда седловых точек просто не будет).
* Поиск седловой точки для функции F

Рассмотрим систему уравнений:

F`x(x, y) (x-a)(x-b) = 0

F`y(x, y) (y-c)(y-d) = 0

По ней находим все решения (xi, yi)

Строим таблицу k x k:

|  |  |
| --- | --- |
|  | y1, y2, … yk |
| x1, x2, …, xk | F(yi, xi) |

Была теорема, что если x0, y0 – седловая точка в исходной игре, то она же и седловая точка в приведённой выше матрице, –>

ищем все седловые точки матрицы и каждую проверяем, что она седловая точка F(x, y)

Проверка седловой точки происходит по определению:

F(x, y0) <= F(x0, y0) <= F(x0, y) для любых x, y из прямоугольной области

*По теореме Фон-Неймана у выпукло (по x\y) – вогнутой (по y\x) функции седловая точка всегда есть и если у матрицы седловая точка единственна – то это и есть седловая точка F(x, y).*

* 1. Матричные игры (игроки играют несколько раз, стратегия игрока – это вектор - pi – вероятность применения стратегии i)

Дана матрица:

* Вспомнить про доминирование строк
* Найти одно решение в смешанных стратегиях (т.е. найти оптимальную стратегию и значение игры)

В зависимости от выбранного метода на к.р. – нужно подписать, какой метод используется

О.с.с. – оптимальная смешанная стратегия (вектор вероятностей выбора той или иной стратегии игроком) ((p1, …, pn), где pi >= 0, pi <= 1, и сумма всех pi = 1)

Способы решения задачи:

* Предварительное доминирование столбцов и строк (уменьшение размера матрицы):

Доминирование - если одна строка меньше либо равна выпуклой (коэффициенты a1 + … + an-1 = 1, ai >= 0) линейной комбинации других (n-1) строк.

Если одна строка доминируется (поэлементно меньше либо равна) линейной комбинацией других строк, то её можно вычеркнуть. (*вычёркивание минимальной строчки*)

Если один столбец доминируется (поэлементно больше либо равен) линейной комбинацией других столбцов, то егоможно вычеркнуть. (*вычёркивание максимального столбца*)

Из доминирования следует, что

Если одна строка меньше другой строки, то меньшую можно вычеркнуть.

Если один столбец больше другого столбца, то больший можно вычеркнуть.

Если использовалось уменьшение матрицы посредством доминирования, то потом после получения вектора оптимальной смешанной стратегии, нужно соответствующим образом его подправить, нарастив до о.с.с. исходной матрицы. (нужно добавить нули в соответствующие позиции)

При нестрогом доминировании – при выкидывании стр. или стб. могут потеряться некоторые оптимальные смешанные стратегии (что нас не будет волновать, если нужно найти хотя бы одно), при строгом доминировании – вычёркивание не приведёт к потере оптимальных смешанных стратегий.

* Для матрицы размера 2 x n – ***графический метод решения***:

|  |  |
| --- | --- |
| p | a11 … a1n |
| 1-p | a21 … a2n |

Строим линии lj: y = p\* a1j + (1-p)\*a2j

Рисуем их на графике y(p), после чего ищем min по всем j в каждой точке p, получаем ломаную кривую, после чего max y по всем p на отрезке [0, 1] (т.е. берётся минимум по всем lj и выбирается самая высокая точка по всем p из [0, 1])

Особые случаи:

* + Если p0 == 0 или з0 == 0, то на позиции kj в смешанной стратегии будет стоять 1 игрока 1 (p), и стоять 1 для игрока 2 (q).
  + Если p0 – это любая точка (попался горизонтальный участок ломаной кривой в качестве максимума), то 1-й игрок может взять в качестве оптимальной стратегии любое значение из горизонтального диапазона, а для рассчёта оптимальной стратегии 2-го игрока, нужно взять любую из крайних точек горизонтального участка (нужны kj1, kj2)

Найденный y0 = max y – это ***значение игры***

Если p0, для которого достигается y0, и допустим это на границе прямой lj1 и lj2, тогда

(0, …, 0, p0, 0, …, 0, 1-p0, 0, …, 0) (на позициях j1 и j2) – это ***оптимальная смешанная стратегия 1-го игрока***.

Если kj1, kj2 – коэффициенты прямых lj1 и lj2, тогда ищем q0, решение уравнения

kj1\*q + kj2 \* (1-q) = 0, тогда

(0, …, 0, q0, 0, …, 0, 1-q0, 0, …, 0) (на позициях j1 и j2) – это ***оптимальная смешанная стратегия 2-го игрока***.

* Для матрицы размера n x 2:

Тогда рассматриваем матрицу -AT, после чего решаем задачу для матрицы 2 x n, после чего меняем стратегии 1-го и 2-го игрока местами, а значение игры берём с минусом.

* Проверка, что угаданная стратегия действительно является о.с.с.:

поэлементный max(A\*q) <= константа V (цена игры) <= поэлементный min (p\*A)

* (нахождение всех возможных о.с.с)(очень долго) Можно решать задачу для матрицы с использованием теоремы об оптимальных смешанных стратегиях через поиск по квадратным подматрицам
  + Перебираем все квадратные невырожденные подматрицы
  + Находим простое решение каждой из невырожденных подмариц

Если просто решение существует и определитель матрицы != 0, то простое решение единственно и равно

p = V I`A-1, q=VA-1I, V = 1 / (I`A-1I)

* + Получив простое решение проверяем, дополняем его нулями до о.с.с. для размера исходной матрицы и проверяем, что оно действительно является о.с.с. исходной матрицы A
  + *о.с.с. для матриц размера 1x1 – это седловая точка, и их можно найти алгоритмом из задачи 1.1 для матриц.*
  1. Найти **все** оптимальные смешанные стратегии 1-го или 2-го игрока

(либо через метод «крайние оптимальные смешанные стратегии», либо через «геометрический метод»)

cм. задачу 1.2, особенно последние 2 её пункта

1. Задачи из раздела «Потоки в сетях»
2. Потоки в сетях

s – исток

t – сток

fij – поток по ребру сети

* 1. Дана потоковая сеть, найти максимальный поток по Форду-Фалкерсону (нужно записывать увеличивающий путь и дельту (размер потока по этому пути))

В ответе необходимо указать величину потока и сам поток.

Найти минимальный разрез (с обоснованием)

Алгоритм Форда-Фалкерсона:

* берём любой начальный поток (начальное приближение, например, = 0)
* ищем увеличивающий путь
* меняем поток согласно увеличивающему пути, если такого нету, то break, иначе снова идём искать увеличивающий путь, …

***Теорема о максимальном потоке и минимальный разрез*** – разрез у которого сумма пропускных способностей дуг (не величина потока через них, а именно сумма пропускной способности!), равна величине максимального потока через сеть, есть минимальный разрез.

2 способа нахождение минимального разреза:

* Теорема о максимальном потоке и минимальном разрезе
* 1 итерация алгоритма расстановки пометок

Алгоритм расстановки пометок (алгоритм поиска увеличивающего пути):

* пп – помечен и просмотрен
* пн – помечен и не просмотрен
* н – не помечен
* s – пн, остальные – н
* 2: если нету пн, то STOP, увеличивающего пути нету
* выбрать любой узел i помеченный как пн
  + для j: (i, j) \in A, fij < uij, тогда пометим j – пн
  + для j: (j, i) \in A, fij > 0, тогда пометим j – пн
  + пометим i – пп
* Если t не помечена => goto 2

Если t – помечен, то построить увеличивающий путь по меткам в обратном направлении

* 1. Дана потоковая сеть, найти максимальный поток с помощью алгоритма Корзанова начиная с 0-го потока в качестве начального приближения

Надо указать остаточную сеть, слоистую сеть и тупиковый поток (просто указать, взяв его с потолка) в слоистой сети.

Алгоритм Корзанова:

* Строим G(f) – остаточную сеть
  + дуга (i, j) её пропускная способность fij, поток через неё aij
  + если f < a, то в остаточную сеть дуга входит с пропускной способностью a-f
  + если f > 0, то в остаточную сеть вводится обратная дуга с пропускной способностью f
* В сети G(f) – ищем прямой путь из истока к стоку (визуально)
  + Если такого пути нету, то максимальный поток уже построен, break
  + Иначе идём строить слоистую сеть
* Строим слоистую сеть G\*(f) (сеть включает в себя все прямые кратчайшие пути из источника в сток в сети G). Будем строить слои вершин:
  + вершина s
  + все дуги, исходящие из истока s, соответствующие вершины включаются в 1-й слой
  + берём все дуги из 1-го слоя, но ведущие в вершины, которые ранее не использовались.
  + …
  + в k-м слое появится вершина t, и ещё несколько вершин.
  + удаляем все висячие вершины из сети (удаляем итерационно, начиная с k-го слоя) (по сути удаляем те вершины в построенной сети, которые не ведут в t)
  + Слоистая сеть построена
* Ищем тупиковый поток (по путям нельзя пустить бОльший поток) в слоистой сети G\*(f) и по нему модифицируем поток через G(f)
* goto beginning
  1. Поиск допустимого расписания с прерываниями

*Алгоритм упаковки*: построить сеть –> построить поток –> расписание

m – число процессоров (все процессоры одинаковы)

N – число работ

директивные интервалы работ - bi, fi

длительность работы - ti

Построение потоковой сети:

* Отмечаем на временной оси t все bi, fi, теперь они все последовательность y1, y2, … yp
* Строим сеть:

s(исток)

–{m\*dj; длинна интервала dj на оси t}–>

Ij (узел – соответствующий интервал на временной оси t)

–{dj; если работа W может быть выполнена в интервале I}–>

Wi (узел – работа, которую нужно выполнить)

–{ti, время выполнения работы W}–>

t(сток)

Строим максимальный поток через есть (любым способом)

Если максимальный поток насытил все дуги Wi –> t, значит расписание существует.

Далее восстанавливаем расписание по сети и максимальному потоку в ней (фактически сеть указывает, в каком интервале выполняется какая работа)

1. Задачи по МетОптам
2. Методы оптимизации

П – задача, ответ на которую да/нет (напр., существует ли маршрут?)

ПО – оптимизационная задача, ответ ***не*** односложный (напр., найти **минимальный** маршрут)

D(П) – множество индивидуальных задач I

YП – множество индивидуальных задач I с ответом «да»

M(I) – функция максимума – равна максимальному числу в индивидуальной задаче I

l(I) (**размер задачи**) – длинна входа задачи (битовая, напр.)

Класс **P** – класс задач, для которых есть полиномиальный алгоритм решения.

Класс **NP** – класс задач, для которых есть полиномиальный проверяющий алгоритм.

**NP-полная (NPC)** П – если П \in NP и если любая П` \in NP полиномиально сводится к П.

**NP-полная в сильном смысле (NPCs) (сильно NP-полная задача)** – если П \in NP и существует полиномиальное сужение Пp \in NPC. (основные 7 задач, кроме Разб)

**NP-трудная (NPH)** П` - если существует П \in NPC, такая что П =>T П`

**NP-лёгкая (NPE)** П – если существует П` \in NP, такая что П =>T П`

**NP-эквивалентные задачи** – задачи \inNPH и \in NPE

*Псевдо-полиномиальный алгоритм* – алгоритм, сложность которого ограничивается полиномом p(l, M)

*Полиномиальное сужение Пp задачи П* – множество таких индивидуальных задач I, числовые параметры которых M(I) <= p(l(I))

*Задача Пp с числовыми параметрами* – если нету такого полинома, чтобы M(I) <= p(l(I)) для любой задачи I \in D(П)

*Сводимость по Тьюрингу П =>T П`* - если существует алгоритм А решения задачи П, использующий алгоритм А` решения задачи П`, таким образом, что если А` - полиномиальный, то и А – полиномиальный.

(фактически надо просто указать алгоритм, как из решения задачи П` получается решение задачи П за полином)

*Псевдо-полиномиальная сводимость П =>п/п П`* - если существует «f» отражающая индивидуальные задачи Дп -> Дп`:

* Однозначность ответов в задачах (да/нет)
* Сведение происходит за полиномиальное время и ограничено некоторым полиномом r(M(I), l(I))
* Существует q1: для любой I из DП: q1(l`(f(I))) >= l(I)
* Существует q2: для любой задачи I из DП: M`(f(I)) <= q2(M(I), l(I))
  1. Дана задача, показать, что она NP-полная (NPC)
* Покажем принадлежность к NP:
  + Привести проверяющий алгоритм и показать его полиномиальную сложность по количеству операций
  + Привести оценку сложности функции длинны (параметров задачи) (оценку сложности можно без обоснования, но она должна быть верной)
* Взять любую из 7-ми NPC задач и сводим её к выданной задаче
  + Показать, что сводимость имеет полиномиальную сложность
  + Показать, однозначность ответов этих 2-х задач

(теоретически сводимость можно показывать через несколько задач, т.е. П => П1 => П2 => … => Пn => П, но у нас такого быть не должно)

* 1. Доказать сильную NP-полноту выданной задачи П

1 способ (*аналогично доказательству NPC*):

* Показываем, что П принадлежит к NP
* Показываем, что некоторая задача П` \in NPCs сводится псевдополиномиально =>п/п к нашей задаче П
* без обоснования можно брать в качестве П` задачи:
  + 3-Разбиения
  + все задачи без числовых параметров (подходят все из основных 7-ми задач, кроме задачи разбиения)

2 способ (по определению):

* Строим полиномиальное сужение Пp (чтобы M <= p(l))
* Показываем, что Пp \in NPC
  1. Доказать, что выданная ПО задача NP-трудная и NP-легкая

Доказательство NP-трудности (NPH) задачи ПО:

* Нужно свести некоторую задачу П \in NPC (лучше брать ту же задачу, которую и дали, но в не оптимизационном варианте) =>Т сводим по Тьюрингу к ПО.

Доказательство NP-легкости (NPE) задачи ПО:

* Сводим нашу оптимизационную задачу ПО по Тьюрингу =>T к некоторой ДП \in NP (надо показать \in NP), алгоритм этой задачи нас не интересует

(лектор задумывал, что ДП – это некоторая «достройка к задаче»)

Покажем: ПО =>T ДП =>T П

* + П должна быть в NPC (тогда вторая сводимость Тьюринга следует из того, что задача П \in NPC)
  + Покажем, что ДП \in NP
  + Явным образом опишем, как ПО сводится по Тьюрингу к ДП

1. NP-задачи для метоптов:
   1. Семь основных задач \in NPC

Все задачи подразумеваются в целых числах.

Как исторически выводилась \in NPC:

ВЫП -> 3-ВЫП -> ВП -> Клика

-> ГЦ

-> 3-С -> Разб

1. **ВЫП** – задача выполнимости (Кук доказал \in NPC по определению)

Даны булевы переменные x1, …, xn

Заданы дизъюнкции c1, …, cm

Можно ли задать переменные true\false так, чтобы все дизъюнкции выполнялись?

1. **3-ВЫП** – задача три-выполнимости

Даны булевы переменные x1, …, xn

c1, …, cn – набор дизъюнкций (напр., x1 or !x5 or x9 or !x10, или x3, или …)

Существует ли выполняющий набор значений переменных для всех дизъюнкций, если мощность всех дизъюнкций |cj| = 3 (т.е. вид x1 or !x9 or x11)?

1. **3-С** – 3-х мерные сочетания

Даны множества W, X, Y, все размерности n не пересекаются.

Задано подмножество M декартова произведения W \* X \* Y.

Существует ли M` подмножество M, такое что размерность M` = n, и все элементы M` в тройках различны.

1. **ВП** – вершинное покрытие

G=(V, A)

K <= |V|

Существует ли вершинное покрытие V`, размерности K.

*Вершинное покрытие – это множество вершин, такое что, у каждого ребра графа, хотя бы один из концов входит в него*

1. **Клика** – задача Клика

G=(V, A)

J <= |V|

Существует ли Клика мощности J?

*Клика – это подмножество вершинного множества V: если мы берём любые i, j: i != j то (i, j) \in A.*

1. **ГЦ** – гамильтонов цикл (не задача с числовыми параметрами)

Существует ли в графе цикл из рёбер, проходящий через каждую вершину ровно 1 раз.

1. **РАЗБ** – задача о разбиении

a1, …, an – множество чисел

Можно ли разбить множество N={1, …, n} на N` \in N и N\N`, чтобы суммы соответствующих чисел были равны.

* 1. Просто задачи

1. **РМП - РМПС – расписание для многопроцессорной системы**

N = {1, … n} – работы

m процессоров (одинаковых)

ai, - длительность работ.

Работы нельзя прерывать и нельзя переносить с одного процессора на другой.

T – общий директивный срок выполнения всех работ

Существует ли расписание, чтобы задачи выполнились и завершились в срок? Т.е. существует ли разбиение N на Nj, чтобы суммы соответствующих ai каждой Nj были <= T

*(задача с числовыми параметрами)*

*(Разб \in NPC => РМПС):* Достаточно рассмотреть РМПС с 2-мя процессорами и директивным интервалом B/2 (B – это сумма ai из Разб)

1. **ЗК – Комм – задача коммивояжёра**

nj – пункты

dij – расстояние между пунктами

B – верхняя граница длинны маршрута

Существует ли маршрут коммивояжёра (замкнутый цикл, проходящий через каждую вершину ровно один раз) S <= B?

*(ГЦ \in NPC => ЗК):* Достаточно построить Комивояжёра на полном графе, и в качестве расстояний взять либо 1, если в гамильтоновом цикле такое ребро есть, либо 2, если нету, после чего нужно построить маршрут длинны |S| <= (B=|V|=длинна гамильтонова цикла)

1. **УВИ – упорядочивание внутри интервалов**

N={1, …, n}, задачи нельзя прерывать

m=1 (число процессоров = 1), ti <= fi - bi (директивные интервалы и время выполнения задач)

Можно ли упорядочить задачи так, чтобы в каждый момент времени выполнялась одна задача и всё было в своих директивных интервалах?

*(Разб \in NPC => УВИ):* достаточно построить УВИ, в которой все работы (числа из разбиения) – могут уложиться в любое место общего директивного интервала = B+1, но поцентру на B/2 впилить одну дополнительную работу с временем выполнения 1 ед., и жёсткими директивными сроками [B/2; B/2+1] – тогда она должна там выполниться, а до неё и после неё должны ваккурат лечь другие задачи как раз длинной в B/2 до и после.

1. **МП – множество представлений**

S = {1, …, n}

C = {c1, …, cm}

ci подмножество S

K > 0

Содержит ли S множество представлений для С мощности K (или эквивалентно: «мощности не больше K»)?

*Множество представлений – такое подмножество S` \in S, что для каждого сj существует элемент x: x \in S`.*

*(ВП \in NPC => МП):* Достаточно взять S как множество вершин, а каждое ребро представить, как ci, и тогда задача очевидным образом сведена.

1. **ИП – Изоморфизм подграфов**

G1=(V1, A1), G2=(V2, A2)

|V1| = n1, |V2| = n2

Содержит ли G1 подграф, изоморфный G2?

*(Клика \in NPC => ИП):* Сводится очевидным образом, достаточно в качестве G2 взять полный подграф (Клику) соответствующей мощности J, запрошенной в задаче Клики.

1. **ОДОС – Оставное дерево ограниченной степени**

G=(V, A), |V| = n, K <= n-1

Существует ли остовное дерево степени K.

*(ГЦ \in NPC => ОДОС):* Сведение очевидно, так как гамильтонов путь – это оставное дерево степени K=2.

1. **ЗР – задача о рюкзаке**

N = {1, … n}

Si – стоимость вещи

Vi – объём вещи

Объём рюкзака = V

Необходимая стоимость = S

Можно ли забить рюкзак так, чтобы не превысить его объём, но достичь или превысить общую стоимость?

*(Разб \in NPC => ЗР):* Если взять V=S=B/2, то как раз получим для условия Рюкзака теже формулы, что и для задачи Разбиения.

1. **СДП – Самый длинный путь**

G=(V, A), K <= |V| -1

Существует ли простой путь из K рёбер?

*Простой путь (гамильтонов путь)* – путь, содержащий каждую вершину единожды.

*(ГЦ \in NPC => СДП):* ГЦ – это частный случай СДП, когда K = |V|-1

!!!??? – что за фигня, ведь путь и цикл – разные вещи?

1. **НОП – Наибольший общий подграф**

G1=(V1, A1), G2=(V2, A2), K

Существует ли G` - подграф G1 и G2, размера по количеству ребёр больше K?

*(Клика \in NPC => НОП):* Клика – это фактически частный случай задачи НОП, если G2 – это полный подграф размерности J (количество вершин Клики), а K = J\*(J-1)/2

1. **МСК** – Минимум суммы квадратов

a1, …, an, K, J

N = {1, …, n}

Существует ли разбиение N на K непересекающихся множеств Ni, такое что

sum по Ni (sum по aj из Ni ( aj\*aj )) <= J

*(Разб \in NPC => МСК):* МСК сводится к Разб, если K = 2, J = B2/2

1. **МВНЗ - Минимизация веса невыполненных заданий**

N = {1, …, n}

ti – время выполнения задания

wi – вес задания

fi – директивный срок i-го задания

K – некоторое число

Существует ли расписание, в котором суммарный вес опозданий не превышает K?

*(Разб \in NPC => МВНЗ):* Разб сводится к МВНЗ, если ti = wi = ai, fi = K = B/2

При такой подстановке получается, что в аккурат половина чисел должна уложиться в расписание, и ровно половина должна просрочиться.

1. **УК – упаковка контейнеров**

N = {1, …, n}

vi – объём элементов

K контейнеров объёмом V

Можно ли упаковать все элементы в K контейнеров объёма V?

*(Разб \in NPC => УК):* Разб является частным случаем УК, если в качестве V взять B/2, а контейнера всего 2.

1. **ИПК** – Интеграл произведения косинусов

Даны a1, …, an

Определить равен ли нулю определённый интеграл [0, 2\*π] произведения i=1,n cos(aix)

*(Разб \in NPC => ИПК):* Используя формулу раскрытия cos x \* cos y можно подобрать частный случай для решения задачи Разб.

1. **3-Разб** - задача 3-разбиения

N = {1, …, 3n}

a1, …, a3n

B: B/4 < ai < B/2, sum(ai) = nB

Можно ли разбить N на n непересекающихся подмножеств Ni, чтобы

sum (aj \in Ni) = B для любого Ni

*(3-Разб =>п/п РМПС):* Достаточно задать для РМПС количество процессоров m=n, директивные сроки ti = ai, T = B. Далее показать все 4 пункта из определения псевдо-полиномиальности.

1. **КПМ** – К-е по порядку множество (вопрос принадлежности к NP – открыт; Разб =>T КПМ \in NPH)

N = {1, …, n}

a1, …, an – натуральные числа

0 < с <= sum (ai)

K <= 2n

Существует ли не менее K различных Nj подмножеств N, таких чтобы для каждого j sum (ai \ in Nj) <= C

*(Разб =>Т КПМ):* Тут нечто хитрое с бинарными разбиениями - !!!???