Приближенные алгоритмы для NP-полных задач

Арина Афанасьева
СПбГУ ИТМО
2010

Введение

Понятие NP-полной задачи

- **Задача поиска** (search problem) задаётся алгоритмом **C**, который получает на вход условие $\mathbf{I}$ и кандидата на решение $\mathbf{S}$ и имеет время работы, ограниченное полиномом от $|\mathbf{I}|$. $\mathbf{S}$ называется решением (solution), если и только если $\mathbf{C}(\mathbf{S}, \mathbf{I}) = \mathbf{true}$.

- **NP** — класс всех задач поиска.
- **P** — класс задач поиска, решение для которых может быть быстро найдено (за полиномиальное время).
- Большинство исследователей считают, что $\mathbf{P} \neq \mathbf{NP}$.

Введение

Понятие NP-полной задачи

- Задача поиска называется **NP-полной** (NP-complete), если к ней сводятся все задачи поиска.

- В предположении **$P \neq NP$ не существует полиномиальных алгоритмов для NP-полных задач.**

Введение
Понятие приближенного алгоритма

- Многие задачи, представляющие практический интерес – NP-полные.
- Для них маловероятно найти точный алгоритм с полиномиальным временем работы.

- При **небольшом объеме входных данных** может подойти алгоритм, время работы которого выражается показательной функцией.
- Иногда удается выделить **важные частные случаи**, разрешимые в течение полиномиального времени.

- Можно найти в течение полиномиального времени решение, близкое к оптимальному.
- Алгоритм, возвращающий решения, близкие к оптимальным, называется **приближенным алгоритмом.**

Методы решения NP-полных задач

- **Приближенные и эвристические методы** - применение эвристик для выбора элементов решения.
- **Псевдополиномиальные алгоритмы** - подкласс динамического программирования.
- **Метод локальных улучшений** - поиск более оптимального решения в окрестности некоторого текущего решения.
- **Метод ветвей и границ** - отбрасывание заведомо неоптимальных решений целыми классами в соответствии с некоторой оценкой.
- **Метод случайного поиска** - представление выбора последовательностью случайных выборов.

Оценка качества приближенных алгоритмов

- Говорят, что алгоритм обладает **коэффициентом аппроксимации ρ(n)**, если

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \le \rho(n)$$

n – размер входных данных
C – стоимость решения
C* - стоимость оптимального решения

- **ρ(n) - приближенный алгоритм** – алгоритм, в котором достигается коэффициент аппроксимации **ρ(n)**.

- **Рассматриваемые примеры:**
  - 2-приближенный алгоритм решения задачи о вершинном покрытии.
  - 2-приближенный и $3/2$-приближенный алгоритмы решения задачи о коммивояжере.
  - $(\mathrm{Ln}|n| + 1)$-приближенный алгоритм решения задачи о покрытии множества.

Оценка качества приближенных алгоритмов

- **Схема аппроксимации** – приближенный алгоритм, входные данные которого включают в себя такое $\varepsilon > 0$, что для любого фиксированного ε эта схема является $(1+\varepsilon)$ - приближенным алгоритмом.
- **Схема аппроксимации с полиномиальным временем выполнения** – схема аппроксимации, работа которой при любом фиксированном $\varepsilon > 0$ завершатся в течение времени, выраженного полиномиальной функцией от размера n входных данных.
- **Схема аппроксимации с полностью полиномиальным временем работы** – схема аппроксимации, время работы которой выражается полиномом от $1/\varepsilon$ и размера входных данных задачи n.
- **Рассматриваемый пример:** алгоритм решения задачи о сумме подмножества – $(1+\varepsilon)$-приближенный с временем работы $O(Polynom(4n*\ln(t)/\varepsilon))$

Задача о вершинном покрытии

Определения и постановка

- **Определение:** **Вершинное покрытие** неориентированного графа $G = (V, E)$ – такое подмножество $V' \subseteq V$, что если (u, v)- ребро графа G, то либо $u \in V'$, либо $v \in V'$ (могут выполняться и оба соотношения).

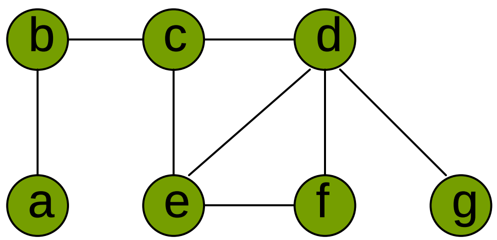
  Размер вершинного покрытия – количество содержащихся в нем вершин.

- **Задача:** найти для заданного неориентированного графа вершинное покрытие минимального размера.

Задача о вершинном покрытии
Алгоритм

- **2-приближенный алгоритм**

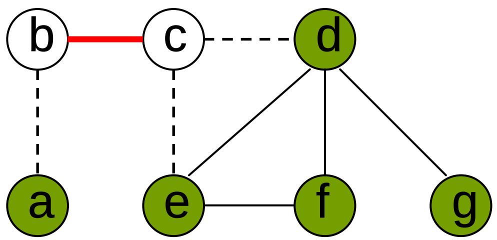
```
Approx_Vertex_Cover(G)
1. C ← Ø
2. E′ ← E[G]
3. While E′ ≠ Ø
4.      do  (u, v) – произвольное
                     ребро из E′
5.              C ← C ∪{u, v}
6.        Удаляем из множества E′ все
          ребра, инцидентные вершинам
          u или v
7. Return C
```



Задача о вершинном покрытии
Алгоритм

- **2-приближенный алгоритм**

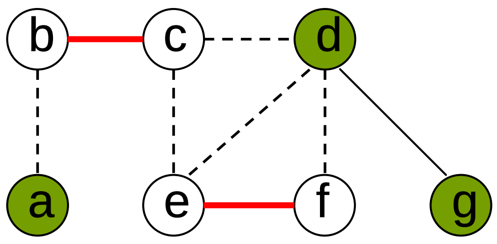
```
Approx_Vertex_Cover(G)
1. C ← ∅
2. E′ ← E[G]
3. While E′ ≠ ∅
4.      do  (u, v) –  произвольное
                      ребро из E′
5.             C ← C ∪{u, v}
6.       Удаляем из множества E′ все
         ребра, инцидентные вершинам
         u или v
7. Return C
```



Задача о вершинном покрытии
Алгоритм

- **2-приближенный алгоритм**

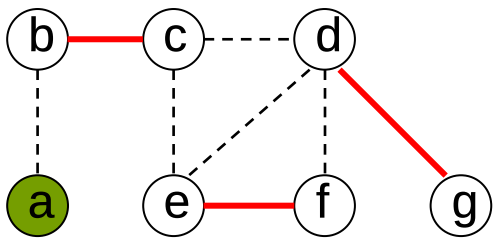
```
Approx_Vertex_Cover(G)
1. C ← ∅
2. E′ ← E[G]
3. While E′ ≠ ∅
4.      do  (u, v) –  произвольное
                      ребро из E′
5.            C ← C ∪{u, v}
6.       Удаляем из множества E′ все
         ребра, инцидентные вершинам
         u или v
7. Return C
```



Задача о вершинном покрытии
Алгоритм

- **2-приближенный алгоритм**

```
Approx_Vertex_Cover(G)
1. C ← Ø
2. E′ ← E[G]
3. While E′ ≠ Ø
4.      do  (u, v) –  произвольное
                      ребро из E′
5.            C ← C ∪{u, v}
6.       Удаляем из множества E′ все
         ребра, инцидентные вершинам
         u или v
7. Return C
```

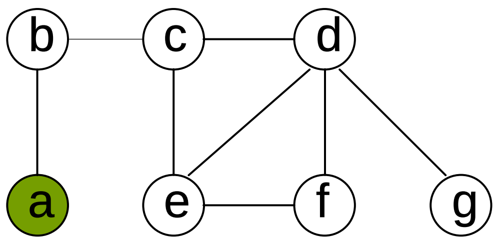


Задача о вершинном покрытии
Алгоритм

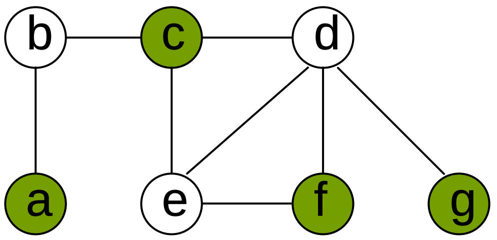
- **2-приближенный алгоритм**

```
Approx_Vertex_Cover(G)
1. C ← ∅
2. E′ ← E[G]
3. While E′ ≠ ∅
4.      do  (u, v) – произвольное
                     ребро из E′
5.           C ← C ∪{u, v}
6.      Удаляем из множества E′ все
        ребра, инцидентные вершинам
        u или v
7. Return C
```

Результат:



Оптимальное решение:



Задача о вершинном покрытии
Теорема о точности алгоритма

- **Теорема:**
  - Approx_Vertex_Cover является $\mathbf{2}$–приближенным алгоритмом с полиномиальным временем работы.

- **Доказательство**
  - **время работы**

    $\mathbf{O(V + E)}$ при использовании списков смежных вершин.
  - **корректность**

    множество вершин **C**, возвращаемое алгоритмом – вершинное покрытие, т.к. алгоритм не выходит из цикла, пока каждое ребро $\mathbf{E[G]}$ не будет покрыто некоторой вершиной из множества **C**.

Задача о вершинном покрытии
Теорема о точности алгоритма

- **точность**

  A – множество выбранных ребер

  C^* – оптимальное покрытие

  C – возвращаемое алгоритмом покрытие

  C^* содержит хотя бы одну конечную точку каждого ребра из A. Никакие два ребра из A не покрываются одной и той же вершиной из C^*. Следовательно

$$|C^*| \geq |A|$$

  Каждый раз выбирается ребро, ни одна из конечных точек которого не вошла в C. Поэтому

$$|C| = 2|A|$$

  Получаем

$$|C| = 2|A| \leq 2|C^*|$$
$$\rho(n) = C/C^* = 2$$

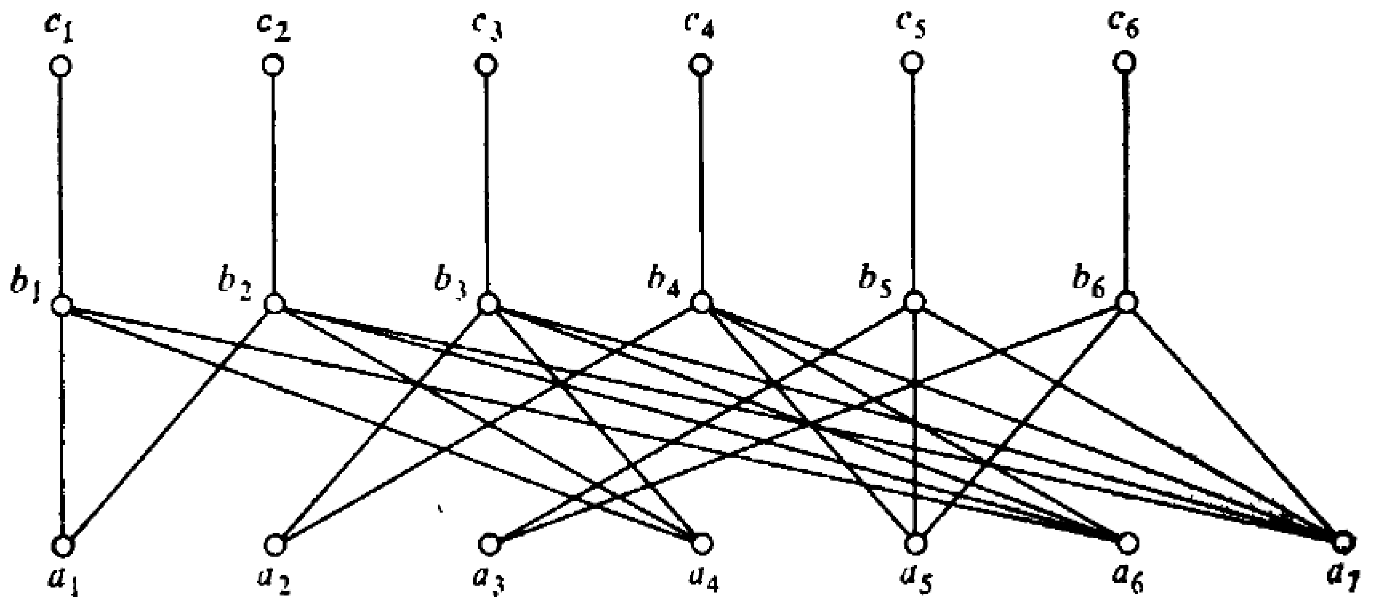
Задача о вершинном покрытии
Другие алгоритмы

- Существует множество других решений этой задачи, однако значение всех коэффициентов аппроксимации **не меньше 2 – o(1)**

- Использование эвристики **выбора вершин с максимальной кратностью** приводит к алгоритму, не имеющему фиксированной границы для получаемой относительной ошибки.

Задача о вершинном покрытии
Выбор вершин с максимальной степенью

- **Контрпример** для эвристики выбора вершин с максимальной степенью. Коэффициент приближения равен $\mathbf{L(n)/n + 1}$, где $\mathbf{L(n)}$ – число вершин типа **a.**



n	6	10	30	100	1000	...
$L(n)/n$ (%)	117	160	267	380	600	...

Задача о коммивояжере
Определения и постановка

- **Дано:**
  - полный неориентированный граф $G = (V,E)$
  - каждому ребру $(u, v) \in E$ сопоставляется целочисленная стоимость $c(u, v) \geq 0$
  - $c(u, v)$ удовлетворяет **неравенству треугольника**, т.е. для всех $u, v, w \in V$

$$c(u, w) \leq c(u, v) + c(v, w)$$

- **Определение:** **гамильтонов цикл** – простой цикл, содержащий все вершины множества V.

- **Задача:** найти гамильтонов цикл минимальной стоимости.

Задача о коммивояжере
Алгоритм

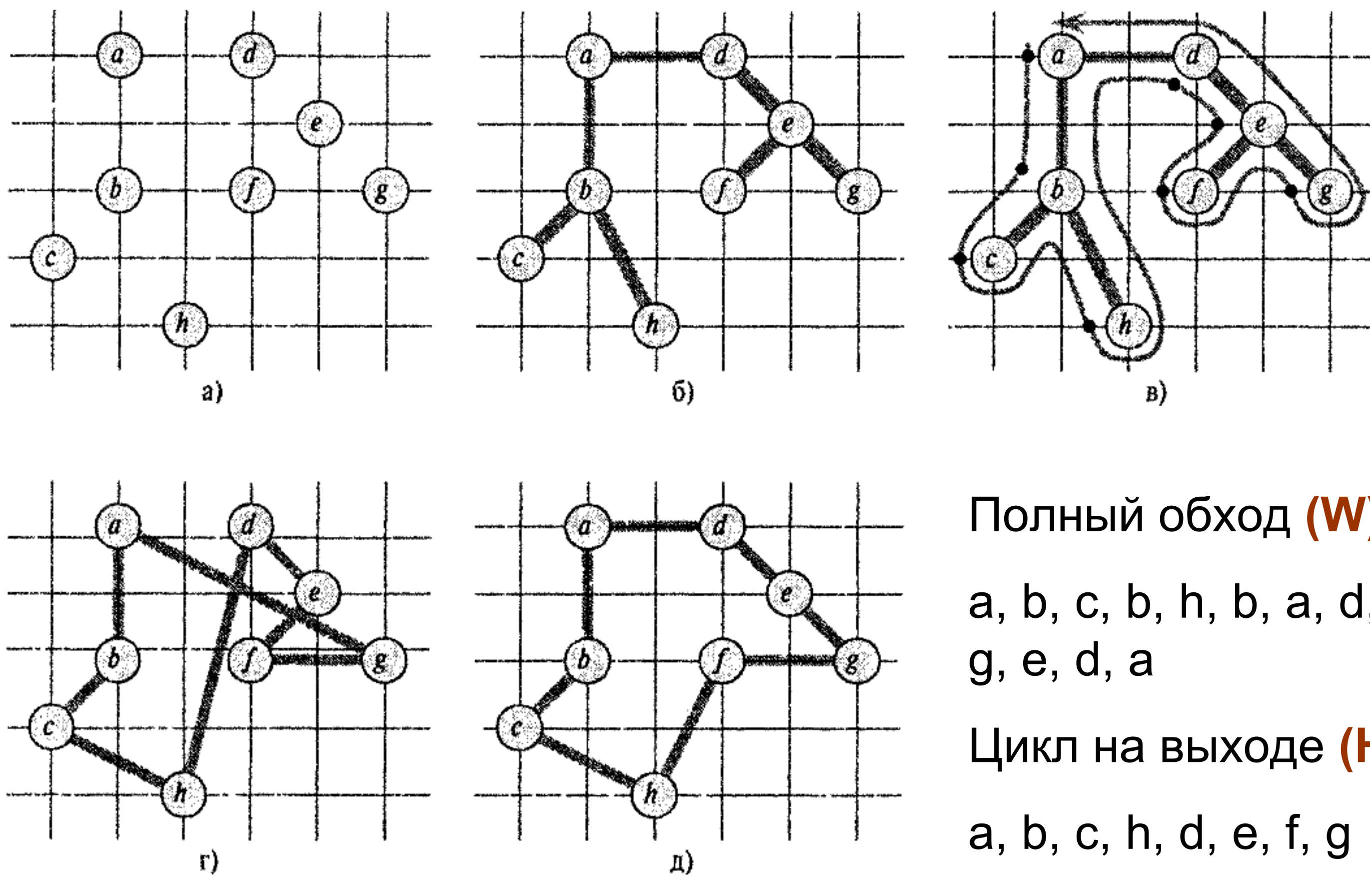
- **2-приближенный алгоритм**

Approx_TSP_Tour(G, c)

1. **Выбирается вершина r ∈ V[G], которая будет «корневой»**
2. **Из корня r с помощью алгоритма MST-Prim(G, c, r) строится минимальное остовное дерево T для графа G**
3. **Пусть L – список вершин, которые посещаются при обходе вершин дерева T в прямом порядке**
4. **return Гамильтонов цикл H, который посещает вершины в порядке перечисления в списке L**

Задача о коммивояжере
Алгоритм

- **Пример**



Полный обход **(W):**

a, b, c, b, h, b, a, d, e, f, e, g, e, d, a

Цикл на выходе **(H):**

a, b, c, h, d, e, f, g

Задача о коммивояжере
Теорема о точности алгоритма

- **Теорема:** Approx_TSP_Tour является 2-приближенным алгоритмом с полиномиальным временем работы.

- **Доказательство**
  - **Время работы $O(V^2)$**
  - **Точность**

    H^* - оптимальный тур

    T – минимальное остовное дерево

    $$c(T) \le c(H^*) \qquad (1)$$

    W – список вершин, посещаемых при полном обходе T

    $$c(W) = 2c(T) \qquad (2)$$

    $$(1), (2) \Rightarrow c(W) \le 2c(H^*) \qquad (3)$$

Задача о коммивояжере
Теорема о точности алгоритма

H – цикл, возвращаемый алгоритмом

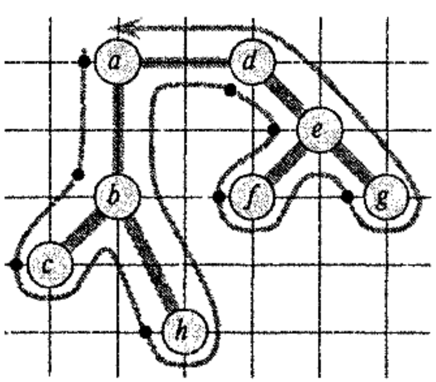
H получается из **W** путем исключения повторных вхождений каждой вершины. Согласно неравенству треугольника, стоимость при этом не возрастает.

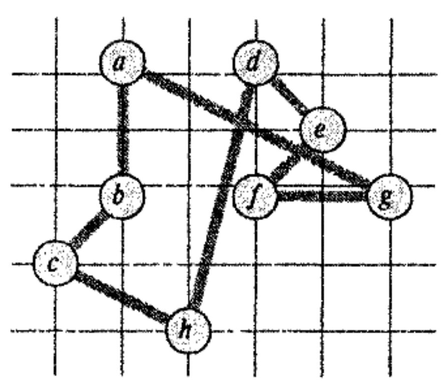
$$c(H) \le c(W) \quad (4)$$

$$(3), (4) \Rightarrow c(H) \le 2c(H^*)$$

$$\rho = c(H)/c(H^*) = 2$$

Пример:





Полный обход **(W):**

a, b, c, b, h, b, a, d, e, f, e, g, e, d, a

Цикл на выходе **(H):**

a, b, c, h, d, e, f, g

Задача о коммивояжере
3/2 – приближенный алгоритм

`Approx-TSP-Improved(G)`

1. Построить минимальное остовное дерево **`T`** графа **`G`**
2. Найти минимальное полное паросочетание всех вершин дерева **`T`** нечетной степени
3. Добавить найденные ребра в дерево **`T`** и найти в полученном графе эйлеров цикл
4. Убрать из полученного цикла все повторения вершин и вернуть полученный цикл

- **Теорема:** Approx_TSP_Tour является 3/2-приближенным алгоритмом с полиномиальным временем работы.

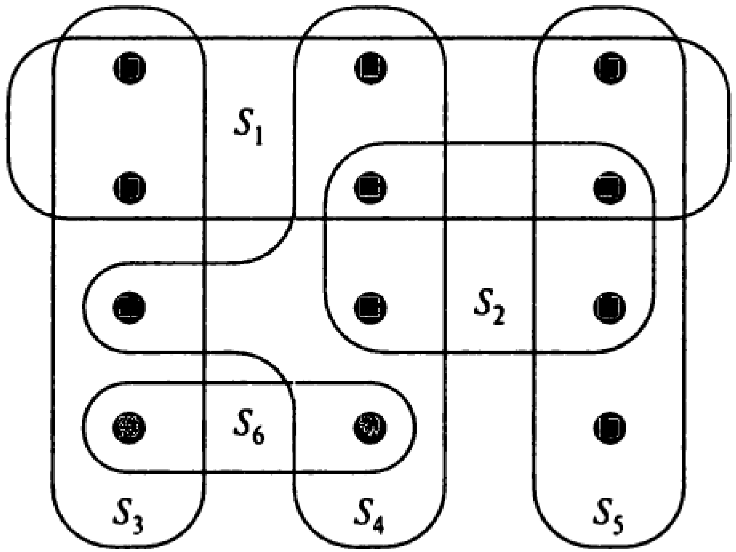
Задача о коммивояжере
3/2 – приближенный алгоритм

- **Доказательство 3 / 2-приближенности:**
  - Стоимость построенного цикла не провосходит **c(T) +c(P),** где **c(P)** — вес минимального паросочетания вершин нечетной степени дерева **T**
  - Нужно показать, что **c(P) ≤ c(H*)/2**
  - Обозначим через **A** множество всех вершин нечетной степени дерева **T**
  - Рассмотрим такой гамильтонов цикл на вершинах множества **A**: вершины множества **A** в нем будут встречаться в такой последовательности, в какой они идут в **H*.**
  - Разбив вершины этого цикла на четные и нечетные, мы получим два паросочетания.
  - Вес хотя бы одного из них будет не более **c(H*)/2**
  - Значит, и вес минимального паросочетания не превосходит **C(H*)/2**

Задача о покрытии множества
Постановка

- **Дано**:
  - конечное множество **X**
  - семейство **F** подмножеств множества **X** такое, что каждый элемент **X** принадлежит хотя бы одному подмножеству.

  Говорят, что $\mathbf{S} \in \mathbf{F}$ покрывает содержащиеся в нем элементы.
- **Задача:**
  - найти подмножество $\mathbf{C} \subseteq \mathbf{F}$ минимального размера, члены которого покрывают все множество **X**: $X = \bigcup_{S \in C} S$



X состоит из 12 черных точек

$F = \{S_1, S_2, S_3, S_4, S_5, S_6\}$

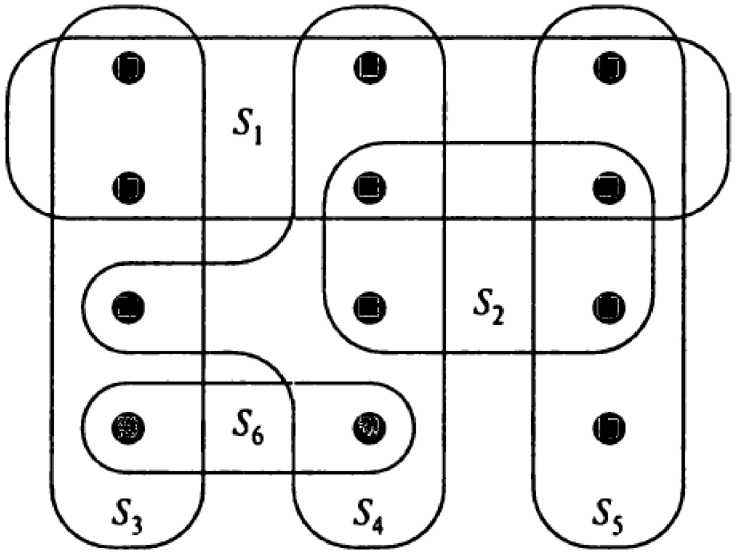
Покрытие минимального размера $C^* = \{S_3, S_4, S_5\}$.

Покрытие, возвращаемое алгоритмом $C = \{S_1, S_4, S_5, S_3\}$

Задача о покрытии множества
Алгоритм

- **p(n)-приближенный алгоритм**

```
Greedy_Set_Cover(X, F)
1.  U ← X
2.  C ← Ø
3.  while U ≠ Ø
4.      do выбирается подмн. S∈F,
        максимизирующее |S ∩ U|
5.      U ← U – S
6.      C ← C ∪ {S}
7.  return C
```



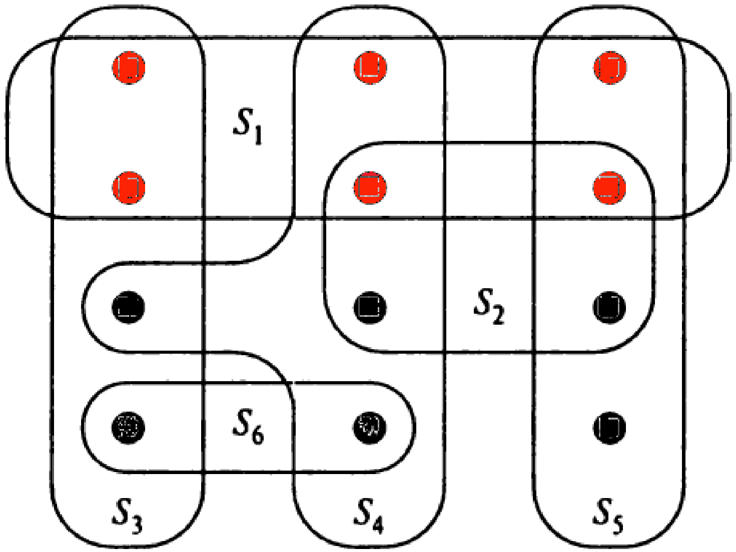
C = {Ø}

Задача о покрытии множества

Алгоритм

- **p(n)-приближенный алгоритм**

```
Greedy_Set_Cover(X, F)
1.  U ← X
2.  C ← Ø
3.  while U ≠ Ø
4.      do выбирается подмн. S∈F,
        максимизирующее |S ∩ U|
5.      U ← U – S
6.      C ← C ∪ {S}
7.  return C
```

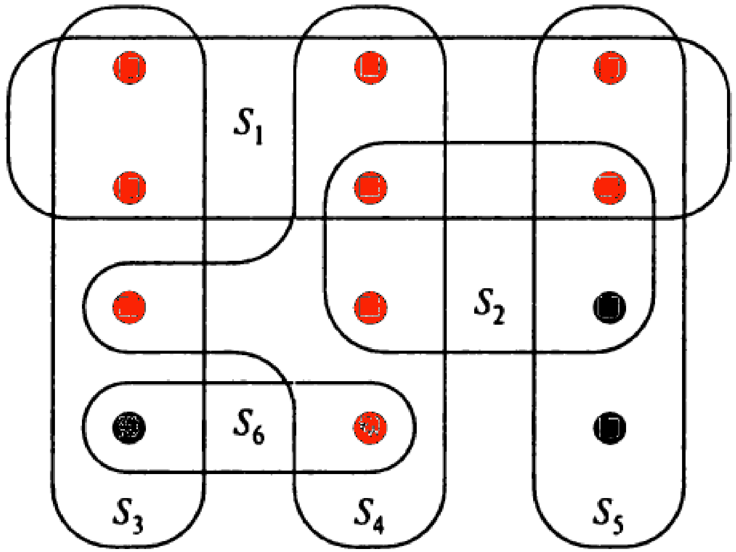


$C = \{S_1\}$

Задача о покрытии множества
Алгоритм

- **p(n)-приближенный алгоритм**

```
Greedy_Set_Cover(X, F)
1.  U ← X
2.  C ← Ø
3.  while U ≠ Ø
4.      do выбирается подмн. S∈F,
        максимизирующее |S ∩ U|
5.      U ← U – S
6.      C ← C ∪ {S}
7.  return C
```

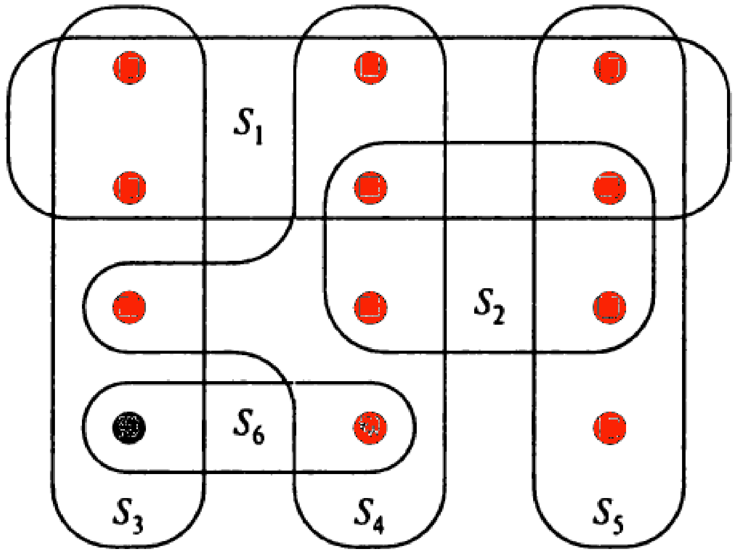


$C = \{S_1, S_4\}$

Задача о покрытии множества
Алгоритм

- **p(n)-приближенный алгоритм**

```
Greedy_Set_Cover(X, F)
1.  U ← X
2.  C ← Ø
3.  while U ≠ Ø
4.      do выбирается подмн. S∈F,
        максимизирующее |S ∩ U|
5.      U ← U – S
6.      C ← C ∪ {S}
7.  return C
```



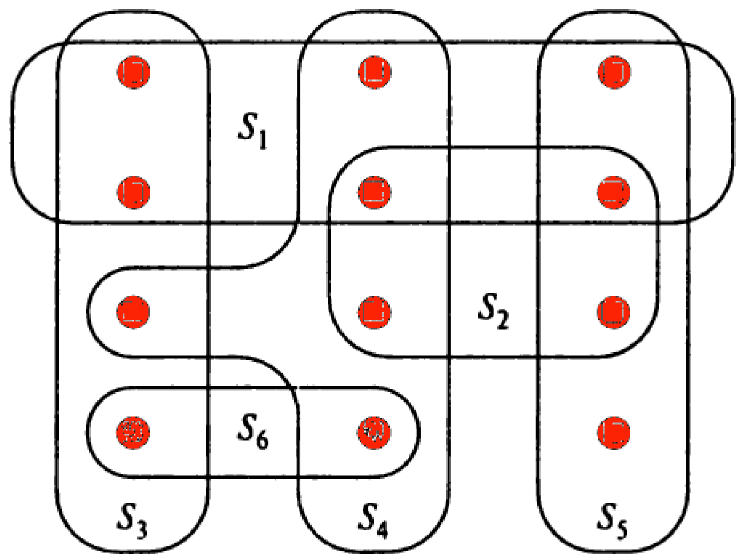
$C = \{S_1, S_4, S_5\}$

Задача о покрытии множества

Алгоритм

- **p(n)-приближенный алгоритм**

```
Greedy_Set_Cover(X, F)
1.  U ← X
2.  C ← ∅
3.  while U ≠ ∅
4.      do выбирается подмн. S∈F,
        максимизирующее |S ∩ U|
5.      U ← U – S
6.      C ← C ∪ {S}
7.  return C
```



$C = \{S_1, S_4, S_5, S_3\}$

Задача о покрытии множества
Теорема о точности алгоритма

- **Обозначение:**

$$H(d) = \sum_{i=1}^{d} 1/i$$ d-ое гармоническое число

- **Теорема:** Greedy_Set_Cover - $\rho(n)$ - приближенный алгоритм с полиномиальным временем работы, где

$$\rho(n) = H(\max\{|S| : S \in F\})$$

Задача о покрытии множества

Теорема о точности алгоритма

- **Доказательство:**

  S_i – i-ое подмножество, выбранное алгоритмом.

  Присвоим S_i стоимость 1.

  Если элемент $x \in X$ впервые покрывается подмножеством S_i, то

$$c_x = \frac{1}{|S_i - (S_1 \bigcup S_2 \bigcup \ldots \bigcup S_{i-1}|}$$

Задача о покрытии множества
Теорема о точности алгоритма

- На каждом шаге алгоритма присваивается единичная стоимость, поэтому

$$|C| = \sum_{x \in X} c_x$$

- Стоимость, присвоенная оптимальному покрытию, равна

$$\sum_{S \in C^*} \sum_{x \in S} c_x$$

- Т.к. каждый **x ∈ X** принадлежит хотя бы одному множеству **S∈C***

$$\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x$$

- Получаем

$$|C| \leq \sum_{S \in C^*} \sum_{x \in S} c_x \qquad (1)$$

Задача о покрытии множества
Теорема о точности алгоритма

- Для любого **S ∈ F** выполняется (доказательство см. в [1])

$$\sum_{x \in S} c_x \le H(|S|) \qquad (2)$$

- Из (1) и (2) следует

$$|C| \le \sum_{S \in C^*} H(|S|) \le |C^*| \times H(\max\{|S| : S \in F\})$$

- **Теорема:** Greedy_Set_Cover является **(Ln|X| + 1)-приближенным алгоритмом** с полиномиальным временем работы.

- **Доказательство:** следует из предыдущей теоремы, с учетом того, что

$$\sum_{k=1}^{n} \frac{1}{k} \le \ln n + 1$$

Задача о сумме подмножества
Постановка

- **Дано:** пара (S, t), где
  - S – множество $\{x_1, x_2, \dots, x_n\}$ положительных целых чисел
  - t – положительное целое число.
- **Задача:** найти подмножество множества S, сумма элементов которого принимает максимально возможное значение, не большее t.

Задача о сумме подмножества
Точный алгоритм (экспоненциальный)

```
Exact_Subset_Sum(S, t)
1. n ← |S|
2. L_0 ← <0>
3. for i ← 1 to n
4.      do L_i ← Merge_Lists(L_i -1,
                 L_i - 1 + x_i)
5.         Из списка L_i удаляются
           все элементы, большие t
6. return Максимальный элемент из
   списка L_n
```

Пояснение:

P_i – множество всех значений, которые можно получить, выбрав подмножество $\{x_1, x_2, \ldots, x_i\}$ и просуммировав его элементы.

$S = \{1, 4, 5\}$

$P_1 = \{0, 1\}$

$P_2 = \{0, 1, 4, 5\}$

$P_3 = \{0, 1, 4, 5, 6, 9, 10\}$

$\mathbf{P_i = P_{i-1} \cup (P_{i-1} + x_i)}$

L_i – отсортированный список, содержащий все $\mathbf{x} \in P_i : x \leq t$

$|L_i| <= 2^i$

Задача о сумме подмножества
Приближенный алгоритм

- На основе точного алгоритма разработаем **схему аппроксимации с полностью полиномиальным временем работы.**

- **Идея**: если два значения в списке **L** мало отличаются друг от друга, то для получения приближенного решения нет смысла явно обрабатывать оба эти значения.

- **Сократить список L** по параметру **δ** значит удалить из **L** максимальное количество элементов так, чтобы в полученном **L’** для каждого удаленного из **L** элемента **y** содержался **z**, аппроксимирующий **y**:

$$\frac{y}{1+\delta} \le z \le y$$

Задача о сумме подмножества
Приближенный алгоритм

- **Алгоритм сокращения списка**

```
Trim (L, δ)
1. m ← |L|
2. L′← <y₁>
3. last ← y₁
4. for i ← 2 to m
5.      do if yᵢ > last * (1 + δ)
        //yᵢ ≥ last т.к. список L
        отсортирован
6.           then last ← yᵢ
7. return L′
```

$$\frac{y_i}{1+\delta} \le last \le y_i$$

y_i добавляется в L', если

$$y_i \ge last * (1+\delta)$$

Пример:

$\delta = 0{,}1$

L = <10, 11, 12, 15, 20, 21, 22, 23, 24, 29>

L' = <10, 12, 15, 20, 23, 29>

Задача о сумме подмножества
Приближенный алгоритм

- **Входные данные:** $S = \{x_1, x_2, \ldots, x_n\}; t; 0<\varepsilon<1$
- **Возвращаемое значение** z отличается от оптимального не более, чем в $1 + \varepsilon$ раз.

```
Approx_Subset_Sum(S, t, ε)
1. n ← |S|
2. L_0 ← <0>
3. for i ← 1 to n
4.      do     L_i ← Merge_Lists(L_{i-1}, L_{i-1} + x_i)
5.             L_i ← Trim(L_i, ε/2n)
6.             Из списка L_i удаляются все элементы, большие t
7. Пусть z* - максимальное значение в списке L_n
8. return z*
```

Задача о сумме подмножества
Теорема о точности алгоритма

- **Теорема:** Approx_Subset_Sum - схема аппроксимации с полностью полиномиальным временем работы, позволяющая решить задачу о сумме подмножества.

- **Доказательство:**
  - Оптимальное решение $y^* \in P_n$
  - $z^* \leq y^*$
  - Нужно показать, что $y^*/z^* \leq 1 + \varepsilon$
  - Также нужно показать, что время работы алгоритма выражается полиномиальной функцией и от $1/\varepsilon$, и от размера входных данных.

Задача о сумме подмножества
Теорема о точности алгоритма

- Для каждого $\mathbf{y \le t,\ y \in P_i}$ существует $\mathbf{z \in L_i}$:
$$\frac{y}{(1+\varepsilon/2n)^i} \le z \le y$$
- Это неравенство должно выполняться для $\mathbf{y^* \in P_n}$, поэтому существует элемент $\mathbf{z \in L_n}$:
$$\frac{y^*}{(1+\varepsilon/2n)^n} \le z \le y^*$$
$$\frac{y^*}{z} \le \left(1+\frac{\varepsilon}{2n}\right)^n$$
$$\forall z \in L_n, z^* \ge z \Rightarrow$$
$$\frac{y^*}{z^*} \le \left(1+\frac{\varepsilon}{2n}\right)^n \le e^{\frac{\varepsilon}{2}} \le 1+\frac{\varepsilon}{2}+\left(\frac{\varepsilon}{2}\right)^2 \le 1+\varepsilon$$

Задача о сумме подмножества
Теорема о точности алгоритма

- Оценим границу длины списка L_i .
- Рассмотрим последовательные элементы z и z' в списке L_i после сокращения: $z'/z > 1 + \varepsilon/2n$.
- Каждый список содержит 0, возможно, 1 и до $\lfloor \log_{1+\varepsilon/2n} t \rfloor$ дополнительных значений.
- Количество элементов в каждом списке L_i не превышает

$$\log_{1+\varepsilon/2n} t + 2 = \ln t / (\ln(1 + \varepsilon/2n)) + 2 \le$$

$$\le (2n(1 + \varepsilon/2n) * \ln t)/\varepsilon + 2 \le$$

$$\le 4n * \ln t/\varepsilon + 2$$

- Поскольку время выполнения процедуры Approx_Subset_Sum выражается **полиномиальной функцией от длины списка** L_i, эта процедура является **схемой аппроксимации с полностью полиномиальным временем выполнения.**

Рандомизированные приближенные алгоритмы

- **Рандомизированный ρ(n) приближенный алгоритм** имеет коэффициент аппроксимации ρ(n):

$$\max\left(\frac{E(C)}{C^*}, \frac{C^*}{E(C)}\right) \le \rho(n)$$

n – размер входных данных
E(C) – мат. ожидание стоимости решения
C* - стоимость оптимального решения

MAX-3-CNF выполнимость
Постановка задачи

- **CNF** – конъюнктивная нормальная форма.
- В случае **3-CNF** в каждых скобках содержится ровно три различных литерала.
- Пример:

$$(x_1 \vee \neg x_1 \vee x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

- **Задача:** найти присваиваемые переменным значения, при которых максимальное количество подвыражений в скобках принимает значение **1**.

MAX-3-CNF выполнимость

Рандомизированный алгоритм

- **Идея алгоритма:** каждой переменной независимо с вероятностью ½ присваивается значение **1** и с вероятностью ½ – значение **0**.

- **Теорема:** Для заданного экземпляра задачи о **MAX-3-CNF** выполнимости с **n** переменными и **m** выражениями в скобках такой алгоритм является **рандомизированным 8 / 7 – приближенным алгоритмом.**

MAX-3-CNF выполнимость

Рандомизированный алгоритм

- **Доказательство:**

$Y_i = I$ {i-е подвыражение в скобках выполняется}

$\Pr$ {i-е подвыражение не выполняется} $= (1/2)^3 = 1/8$

$\Pr$ {i-е подвыражение выполняется} $= 1 - 1/8 = 7/8$

$$E[Y_i] = 7/8$$

$$E[Y] = E[\sum_{i=1}^{m} Y_i] = \sum_{i=1}^{m} E[Y_i] = \sum_{i=1}^{m} 7/8 = 7m/8$$

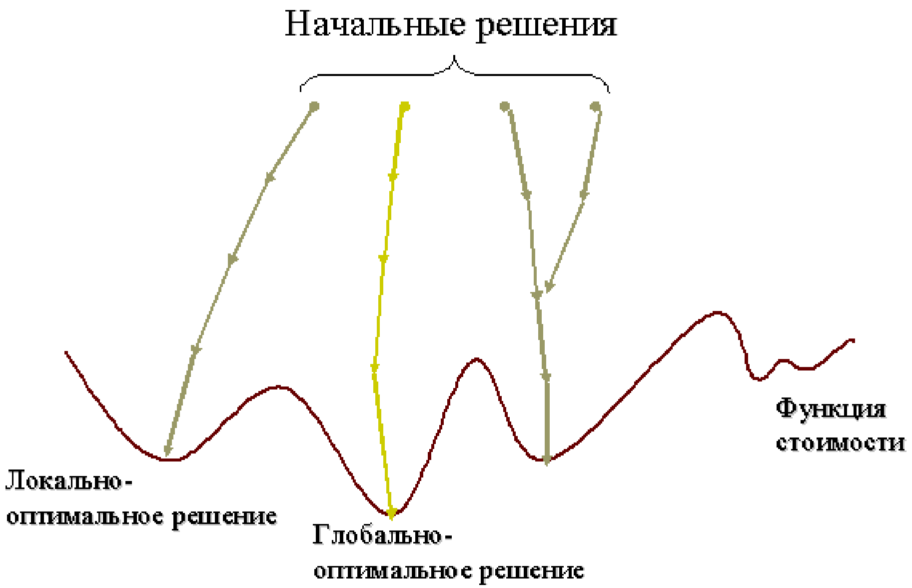
- $\rho(n) = m/(7m/8) = 8/7$

Метод локальных приближений
Идея

- Начать с **произвольного решения**.
- Для улучшения текущего решения применить к нему какое-либо **преобразование** из заданной совокупности преобразований. Это улучшенное решение становится текущим решением.
- Повторять указанную процедуру до тех пор, **пока** ни одно из преобразований в заданной совокупности **не позволит улучшить** текущее решение.

Метод локальных приближений
Идея

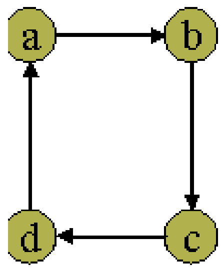
- Если заданная совокупность преобразований включает все преобразования (с помощью которых из любого решения можно получить любое другое), то мы получим точное (**глобально-оптимальное**) решение.
- На практике совокупность преобразований ограничивают. С помощью них из ряда произвольных решений получают **локально-оптимальные** решения и выбирают из них лучшее.

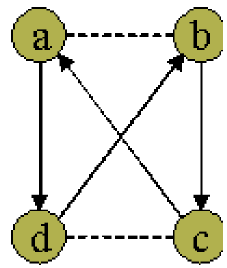


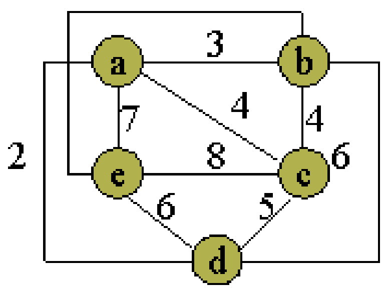
Метод локальных приближений

Пример: задача коммивояжера

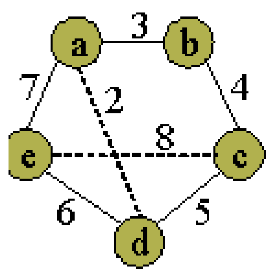
- «Двойной выбор»



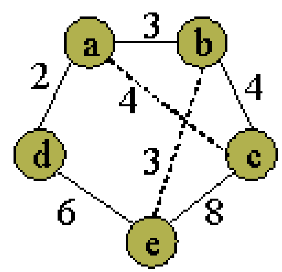




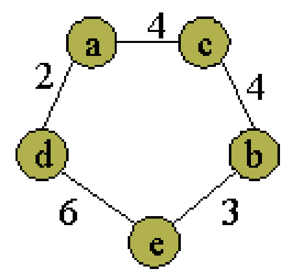
Граф



а



б



в

- k-выбор
- Выбор с переменной глубиной

Метод ветвей и границ

- Решая дискретную экстремальную задачу, разобьем множество всех возможных вариантов на **классы** и построим оценки для них.
- В результате становится возможным отбрасывать решения целыми классами, если их оценка хуже некоторого **рекордного значения**.

Метод ветвей и границ

- Дискретная экстремальная (на минимум) задача в общем виде:
  - Пусть задано дискретное множество $\mathbf{A}$ и определенная на нем функция $\mathbf{f}$. Обозначим минимум функции $\mathbf{f}$ на $\mathbf{X}$ как $\mathbf{F(X)}$.
  - Требуется найти $\mathbf{x_0 \in A: f(x_0) = F(A)}$

Метод ветвей и границ

- **Замечание 1**
  Пусть $A = A_0 \cup A_1 \cup \ldots \cup A_k, A_i \cap A_j = \emptyset, i \neq j.$
  Причем $F(A) < F(A_0)$, т. е. на A_0 минимум не достигается.
  Тогда справедливо следующее: $F(A) = \min\{ F(A_i) \mid i \in 1 : k \}$

- **Замечание 2**
  Пусть Φ — функция, заданная на совокупности подмножеств множества A так, что $\Phi(X) \leq F(X)\ \forall X \subset A$
  Пусть x^* — произвольный элемент A и пусть $f^* = f(x^*)$.
  Тогда справедливо следующее:
  $F(A) = \min\{ f^*, \min\{F(A_i) \mid i \in 1 : k, \Phi(A_i) \leq f^*\} \}$

Метод ветвей и границ

- Разобьем множество **A** на подмножества **A_i** и на каждом из них найдем нижнюю оценку **Φ**.
- Для элементов множества **A** будем вычислять значения функции **f** и запоминать наименьшее в качестве **рекордного значения** **f^***.
- Все подмножества, у которых оценка выше **f^***, объединим в подмножество **A_0**, чтобы в дальнейшем не рассматривать.
- Выберем какое-либо из множеств **$A_i, i > 0$**. Разобьем это множество на более мелкие подмножества. При этом мы будем продолжать улучшать рекордное значение **f^***.
- Этот процесс продолжается до тех пор, пока не будут просмотрены все множества **$A_i, i > 0$**.

Метод случайного поиска

- Обычно выбор решения можно представить **последовательностью выборов.**
- Если делать эти выборы с помощью какого-либо **случайного механизма,** то решение находится очень быстро, так что можно находить решение **многократно** и **запоминать "рекорд",** т. е. наилучшее из встретившихся решений.
- Этот наивный подход существенно улучшается, когда удается учесть в случайном механизме перспективность тех или иных выборов, т. е. **комбинировать случайный поиск с эвристическим методом и методом локального поиска.**
- Такие методы применяются, например, при составлении расписаний для Аэрофлота.

Источники

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 2-е издание. : Пер. с англ. – М. : "Вильямс", 2007.
2. Куликов А.С. Алгоритмы для NP-трудных задач [Online].

   Available: http://logic.pdmi.ras.ru/~infclub/?q=courses/npalgorithms [2010, March 4].
3. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. – М.: Мир, 1985.
4. Поликарпова Н., Герасименко А. Методы решения труднорешаемых задач [Online].

   Available: http://rain.ifmo.ru/cat/view.php/theory/unsorted/approx-2004 [2010, March 4].
5. Vijay V. Vazirani. *Approximation Algorithms.* Springer-Verlag, 2001.