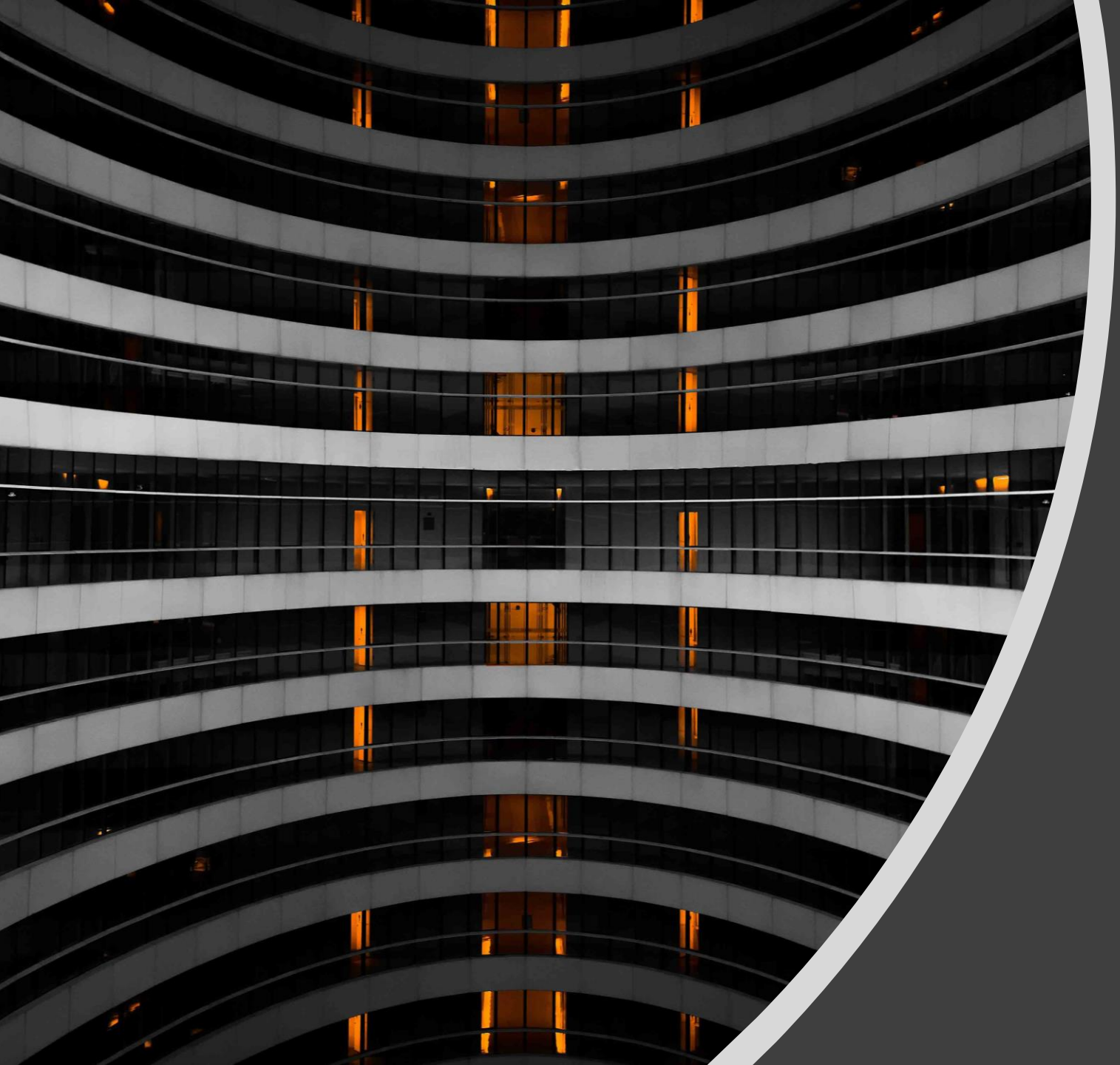


Machine Learning

Лекция #4

Поиск наборов объектов.
Суммирование наборов объектов.


Часто встречающиеся наборы
и ассоциативные правила.

Наборы объектов и наборы тидов

Пусть $\mathfrak{I} = \{x_1, x_2, \ldots, x_m\}$ – множество элементов, называемых *объектами*.

Множество $X \subseteq \mathfrak{I}$ – *набор объектов*

k-набор – набор мощности (размера)

Обозначим $\mathfrak{I}^{(k)}$ множество всех k-наборов, то есть множество всех подмножеств $\mathfrak{I}$ размера k

Наборы объектов и наборы тидов

Пусть $\mathfrak{T} = \{t_1, t_2, \ldots, t_n\}$ – другое множество элементов, называемых ***идентификаторами транзакций или тидами*** (англ. transaction identifiers).

Множество $T \subseteq \mathfrak{T}$ назовём *набором тидов*.

Транзакция представляет из себя кортеж вида $\langle t, X \rangle$, где $t \in \mathfrak{T}$ – уникальный идентификатор транзакции, а X – набор объектов.

Представление базы данных

Представление базы данных

- Пусть **D** – это двоичное отношение над набором объектов и идентификаторов транзакций, то есть, $\mathbf{D} \subseteq \mathfrak{T} \times \mathfrak{I}$.
- Будем говорить, что идентификатор $t \in \mathfrak{T}$ *содержит* объект $x \in \mathfrak{I}$ тогда и только тогда, когда $(t, x) \in \mathbf{D}$.
- Другими словами, $(t, x) \in \mathbf{D}$ тогда и только тогда, когда $x \in X$ входит в кортеж $\langle t, X \rangle$. Будем говорить, что идентификатор t содержит набор $X = \{x_1, x_2, \ldots, x_k\}$ тогда и только тогда, когда $(t, x_i) \in \mathbf{D}$ для всех $i = 1, 2, \ldots, k$.

D	*A*	*B*	*C*	*D*	*E*
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

(a) Binary database

Представление базы данных

Для множества X обозначим через 2^X множество степеней X, то есть множество всех подмножеств X. Пусть $\mathbf{i} : 2^{\mathfrak{T}} \to 2^{\mathfrak{I}}$ - функция, определённая следующим образом:

$$\mathbf{i}(T) = \{x \mid \forall t \in T, t \text{ содержит } X\}, (8.1)$$

где $T \subseteq \mathfrak{T}$, а $\mathbf{i}(T)$ – множество объектов, общих для всех транзакций в наборе T. В частности, $\mathbf{i}(t)$ – набор объектов, содержащихся в транзакции $t \in \mathfrak{T}$.

Представление базы данных

Пусть $t : 2^{\mathfrak{I}} \to 2^{\mathfrak{T}}$ – функция, определённая следующим образом:

$$\mathbf{t}(X) = \{t \mid t \in T \text{ и } t \text{ содержит } X\}, (8.2)$$

где $X \subseteq \mathfrak{I}$, а $\mathbf{t}(X)$ – набор идентификаторов транзакций, который содержит все объекты из в наборе X. В частности, $\mathbf{t}(x)$ – набор идентификаторов, которые содержат единственный объект $x \in \mathfrak{I}$.

Поддержка и часто встречающиеся наборы элементов. Ассоциативные правила.

Поддержка и часто встречающиеся наборы элементов

Поддержка набора X в наборе данных $\mathbf{D}$, обозначаемая как $sup(X, \mathbf{D})$ – это число транзакций в $\mathbf{D}$, которые содержат X:

$$sup(X, \boldsymbol{D}) = |\{t \mid \langle t, \mathbf{i}(t)\rangle \in \mathbf{D} \text{ и } X \subseteq \mathbf{i}(t)\}| = |\mathbf{t}(X)|.$$

Относительная поддержка X – это доля транзакций, содержащих X:

$$rsup(X, \boldsymbol{D}) = \frac{sup(X, \boldsymbol{D})}{|\boldsymbol{D}|}$$

Это оценка *совместной вероятности* элементов из X.

Поддержка и часто встречающиеся наборы элементов

Набор X считается частым в $\boldsymbol{D}$, если $sup(X, \boldsymbol{D}) \geq minsup$, где $minsup$ – определяемый пользователем *минимальный уровень поддержки*.

Если $minsup$ определяется дробью, предполагается относительная поддержка. Мы будем обозначать $\mathfrak{F}$ множество всех часто встречающихся наборов объектов, а $\mathfrak{F}^k$ – множество всех часто встречающихся k-наборов.

Ассоциативные правила

Ассоциативное правило – это выражение вида: $X \overset{s,c}{\rightarrow} Y$, где X и Y – непересекающиеся наборы объектов, то есть, $X, Y \subseteq \Im$ и $X \cap Y = \emptyset$. Обозначим набор $X \cap Y$ как XY. *Поддержкой* правила называется число транзакций, в которых X и Y встречаются вместе как подмножества:

$$s = sup(X \rightarrow Y) = |t(XY)| = sup(XY).$$

Относительная поддержка правила

Относительная поддержка правила – доля транзакций, в которые входят X и Y, и она даёт оценку совместной вероятности X и Y:

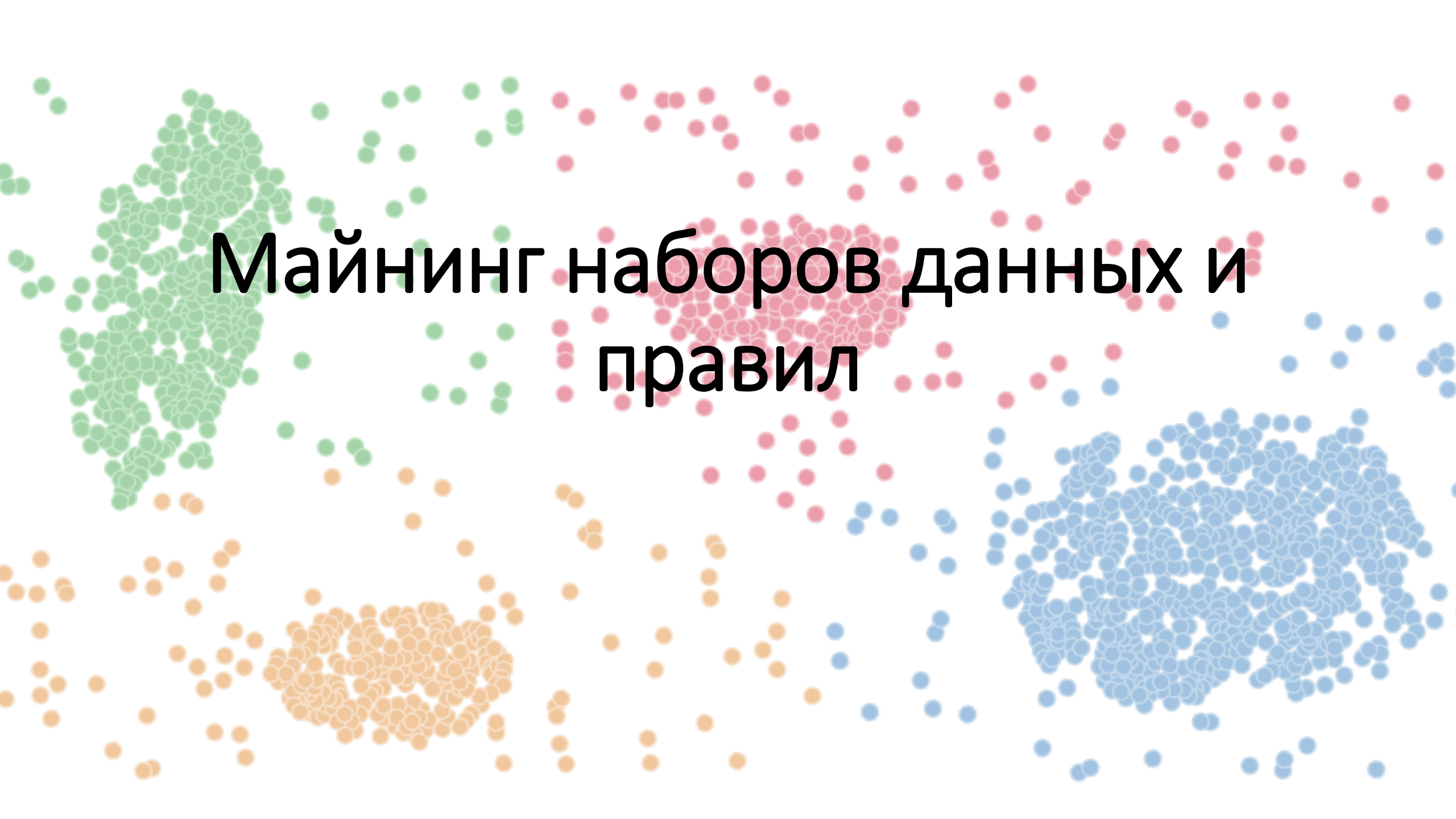
$$rsup(X \to Y) = \frac{sup(XY)}{|\boldsymbol{D}|} = P(X \wedge Y).$$

Уровень доверия правила

Уровень доверия правила – условная вероятность того, что транзакция содержит Y, при условии, что она содержит X:

$$c = conf(X \to Y) = P(Y|X) = \frac{P(X \wedge Y)}{P(X)} = \frac{sup(XY)}{sup(X)}.$$

Правило считается *часто встречающимся*, если часто встречается набор объектов XY, то есть, $sup(XY) \geq minsup$, и *сильным*, если $conf \geq minconf$, где $minconf$ – заданный пользователем минимальный уровень доверия.


Майнинг наборов данных и
правил

Майнинг наборов данных и правил

Формально, имея двоичную базу данных $\boldsymbol{D}$ и заданный пользователем минимальный уровень поддержки $minsup$, задача поиска часто встречающихся наборов объектов сводится к перечислению **всех частых наборов**, то есть, таких, у которых **уровень поддержки не ниже $\boldsymbol{minsup}$**.

Далее, имея множество часто встречающихся наборов $\mathfrak{F}$ и минимальный уровень доверия $minconf$, задача поиска ассоциативных правил сводится к нахождению всех часто встречающихся сильных правил.


Алгоритмы майнинга набора данных.
Наивный (brute-force) алгоритм

Наивный (brute-force) алгоритм

Наивный (brute-force) алгоритм перебирает всевозможные $X \subseteq \mathcal{I}$, и для каждого подмножества определяет его поддержку в наборе данных $\boldsymbol{D}$.

Алгоритм состоит из двух шагов:

1. генерация кандидатов
2. вычисление поддержки.

Шаг 1: Генерация кандидатов

Этот шаг генерирует все подмножества $\mathcal{I}$, называемые *кандидатами*, т.к. каждый набор объектов (itemset) потенциально – кандидат в частый образец. Пространство поиска кандидатов экспоненциально, т.к. есть $2^{|\mathcal{I}|}$ потенциально частых образцов.

Множество всех наборов объектов составляет решетчатую структуру, где два набора X и Y связаны, если X есть *непосредственное подмножество* Y, т.е.

$$X \subseteq Y \text{ и } |X| = |Y| - 1$$

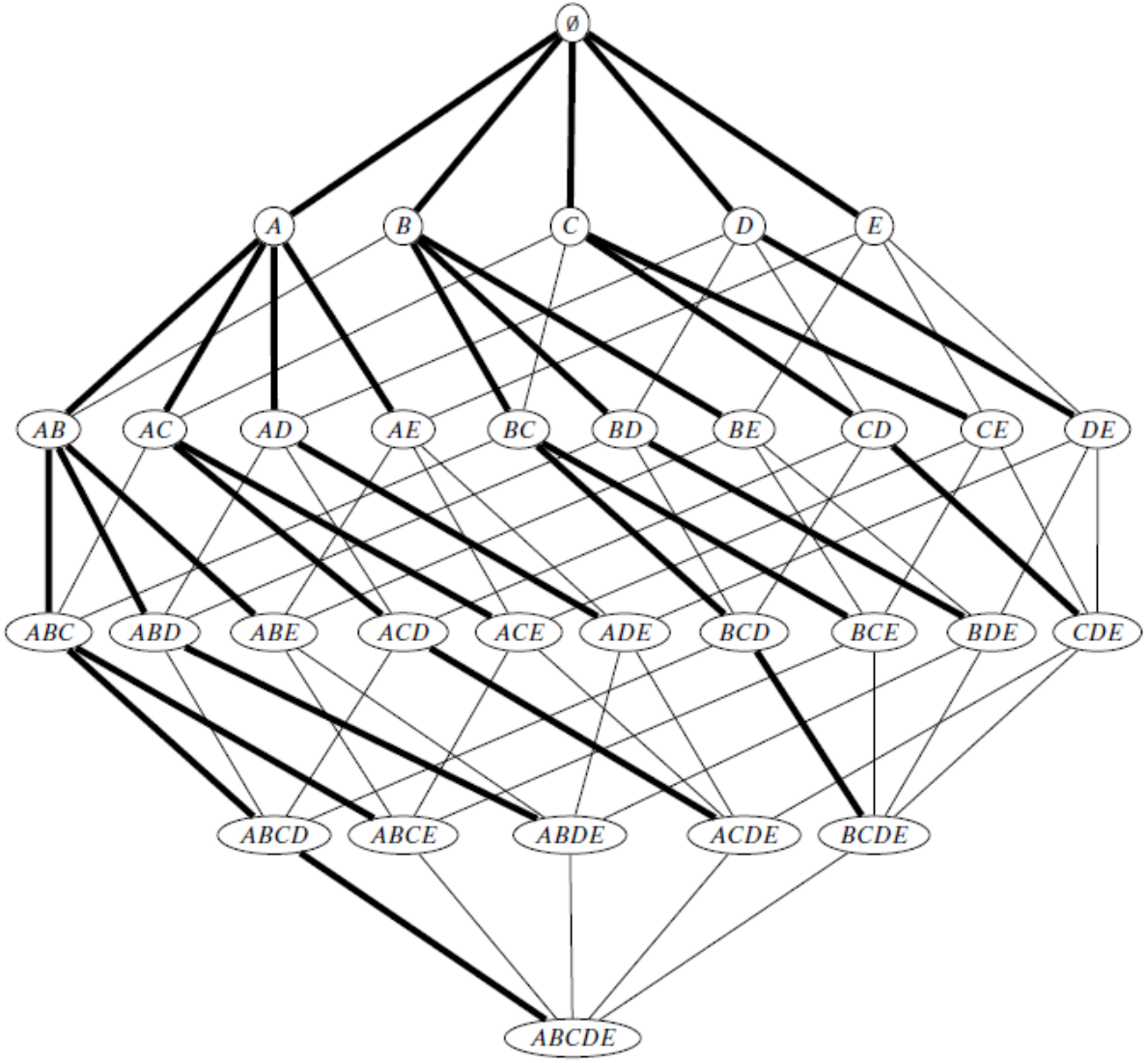


Figure 8.2. Itemset lattice and prefix-based search tree (in bold).

Пространство поиска кандидатов экспоненциально, т.к. есть $\mathbf{2}^{|\mathcal{I}|}$ потенциально частых образцов.

$$\mathbf{2}^{|\mathcal{I}|} = \mathbf{2}^{|\mathbf{5}|} = \mathbf{32}$$

Шаг 1: Генерация кандидатов. Поиск

С точки зрения стратегии поиска, можно перебрать наборы в решетке через **поиск в ширину (BFS, breadth-first)** или в **глубину (DFS, depth-first)** на *префиксном дереве*, X и Y связаны, если X – непосредственное подмножество Y и префикс Y.

Таким образом, можно перебрать наборы начиная с пустого множества, добавляя по одному на шаге.

Шаг 2: Вычисление поддержки

На этом шаге вычисляется поддержка каждого кандидата X и определяется, является ли X частым образцом. Для каждой транзакции $\langle t, \boldsymbol{i}(t) \rangle$ в базе определяется, является ли X подмножеством $\boldsymbol{i}(t)$. Если да, поддержка X увеличивается на 1.

Происходит перечисление наборов объектов $X \subseteq \mathcal{I}$, и для каждого набора объектов вычисляется поддержка проверкой $X \subseteq \boldsymbol{i}(t)$ для каждого $t \in \mathcal{T}$.

Brute-force алгоритм

BRUTEFORCE (D, $\mathcal{I}$, *minsup*):

```
1  F ← ∅ // set of frequent itemsets
2  foreach X ⊆ I do
3      sup(X) ← COMPUTESUPPORT (X, D)
4      if sup(X) ≥ minsup then
5          F ← F ∪ {(X, sup(X))}
6  return F
```

COMPUTESUPPORT (X, D):

```
7  sup(X) ← 0
8  foreach ⟨t, i(t)⟩ ∈ D do
9      if X ⊆ i(t) then
10         sup(X) ← sup(X) + 1
11 return sup(X)
```


Вычислительная сложность

Вычислительная сложность

Вычисление поддержки в худшем случае $O(|\mathcal{I}| \cdot |\boldsymbol{D}|)$, и поскольку есть $O\left(2^{|\mathcal{I}|}\right)$ возможных кандидатов, вычислительная сложность метода brute-force

$$O\left(2^{|\mathcal{I}|} \cdot \mathcal{I} \cdot |\boldsymbol{D}|\right).$$

Даже для небольших пространств наборов этот подход вычислительно трудный, тогда как на практике решетка $\mathcal{I}$ может быть очень большой (например, в супермаркете обычно тысячи товаров.

Способы систематического улучшения brute-force алгоритма.

Поуровневый подход: алгоритм Apriori

Поуровневый подход: алгоритм Apriori

Наивный (brute-force) алгоритм перечисляет все возможные наборы объектов, что приводит к лишним вычислениям, т.к**. многие кандидаты не могут не являться частыми.**

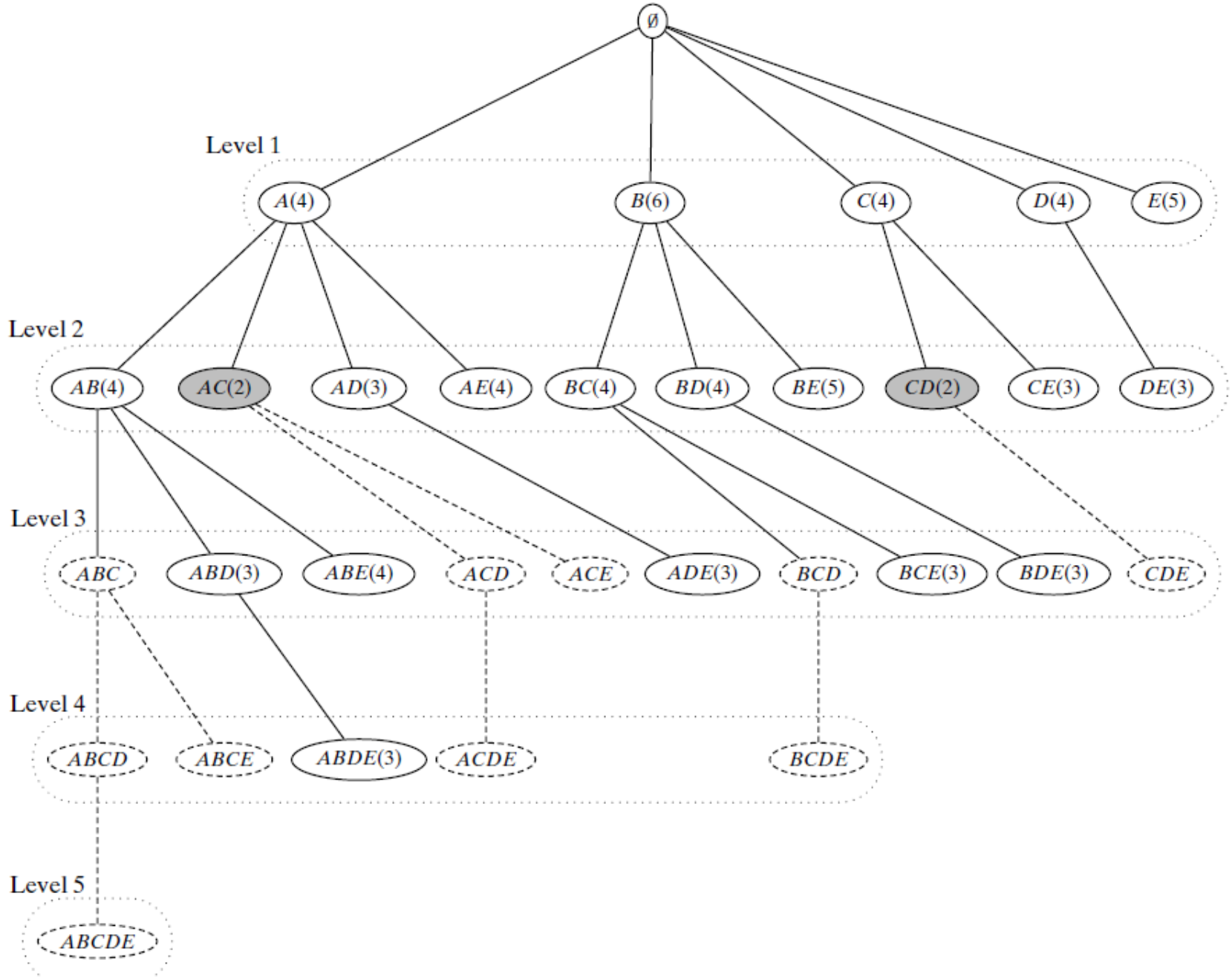
Пусть $X, Y \subseteq \mathcal{I}$ - два любых набора. Заметим, что если $X \subseteq Y$, то $\sup(X) \geq \sup(Y)$, что приводит к следующим наблюдениям:

1. если X – частый образец, то любое подмножество $Y \subseteq X$ тоже часто,
2. если X не часто, то любое надмножество $Y \supseteq X$ не может быть часто.

Поуровневый подход: алгоритм Apriori

Алгоритм Apriori осуществляет поуровневый (в ширину) поиск по пространству поиска наборов, удаление всех надмножеств нечастых кандидатов и генерацию кандидатов с нечастыми подмножествами.

Вместо вычисления поддержки **одного набора объектов** исследует **префиксное дерево** в ширину и вычисляет поддержку всех допустимых кандидатов размера k, которые составляют уровень ***k префиксного дерева***.


Ø
Level 1
A(4)
B(6)
C(4)
D(4)
E(5)
Level 2
AB(4)
AC(2)
AD(3)
AE(4)
BC(4)
BD(4)
BE(5)
CD(2)
CE(3)
DE(3)
Level 3
ABC
ABD(3)
ABE(4)
ACD
ACE
ADE(3)
BCD
BCE(3)
BDE(3)
CDE
Level 4
ABCD
ABCE
ABDE(3)
ACDE
BCDE
Level 5
ABCDE

```
APRIORI (D, I, minsup):
1  F ← ∅
2  C(1) ← {∅} // Initial prefix tree with single items
3  foreach i ∈ I do Add i as child of ∅ in C(1) with sup(i) ← 0
4  k ← 1 // k denotes the level
5  while C(k) ≠ ∅ do
6      COMPUTESUPPORT (C(k), D)
7      foreach leaf X ∈ C(k) do
8          if sup(X) ≥ minsup then F ← F ∪ {(X, sup(X))}
9          else remove X from C(k)
10     C(k+1) ← EXTENDPREFIXTREE (C(k))
11     k ← k + 1
12 return F(k)

COMPUTESUPPORT (C(k), D):
13 foreach ⟨t, i(t)⟩ ∈ D do
14     foreach k-subset X ⊆ i(t) do
15         if X ∈ C(k) then sup(X) ← sup(X) + 1

EXTENDPREFIXTREE (C(k)):
16 foreach leaf Xa ∈ C(k) do
17     foreach leaf Xb ∈ SIBLING(Xa), such that b > a do
18         Xab ← Xa ∪ Xb
           // prune candidate if there are any infrequent subsets
19         if Xj ∈ C(k), for all Xj ⊂ Xab, such that |Xj| = |Xab| − 1 then
20             Add Xab as child of Xa with sup(Xab) ← 0
21     if no extensions from Xa then
22         remove Xa, and all ancestors of Xa with no extensions, from C(k)
23 return C(k)
```

```
APRIORI (D, I, minsup):
1  F ← ∅
2  C(1) ← {∅} // Initial prefix tree with single items
3  foreach i ∈ I do Add i as child of ∅ in C(1) with sup(i) ← 0
4  k ← 1 // k denotes the level
5  while C(k) ≠ ∅ do
6      COMPUTESUPPORT (C(k), D)
7      foreach leaf X ∈ C(k) do
8          if sup(X) ≥ minsup then F ← F ∪ {(X, sup(X))}
9          else remove X from C(k)
10     C(k+1) ← EXTENDPREFIXTREE (C(k))
11     k ← k + 1
12 return F(k)

COMPUTESUPPORT (C(k), D):
13 foreach ⟨t, i(t)⟩ ∈ D do
14     foreach k-subset X ⊆ i(t) do
15         if X ∈ C(k) then sup(X) ← sup(X) + 1
```

1. Пусть $C^{(k)}$ обозначает префиксное дерево из всех кандидатов длины k. Начало метода – **заполнение уровня $C^{(1)}$ единичными объектами** (префиксное дерево изначально пусто).

2. В цикле while (строчки 5-11) происходит **вычисление поддержки для текущего набора кандидатов на уровне k** через процедуру

3. **ComputeSupport**, которая генерирует k-подмножества каждой транзакции в базе D, и для каждого такого подмножества увеличивает поддержку соответствующего кандидата в $C^{(k)}$, если такой существует

```
APRIORI (D, I, minsup):
1  F ← ∅
2  C(1) ← {∅} // Initial prefix tree with single items
3  foreach i ∈ I do Add i as child of ∅ in C(1) with sup(i) ← 0
4  k ← 1 // k denotes the level
5  while C(k) ≠ ∅ do
6      COMPUTESUPPORT (C(k), D)
7      foreach leaf X ∈ C(k) do
8          if sup(X) ≥ minsup then F ← F ∪ {(X, sup(X))}
9          else remove X from C(k)
10     C(k+1) ← EXTENDPREFIXTREE (C(k))
11     k ← k + 1
12 return F(k)

EXTENDPREFIXTREE (C(k)):
16 foreach leaf Xa ∈ C(k) do
17     foreach leaf Xb ∈ SIBLING(Xa), such that b > a do
18         Xab ← Xa ∪ Xb
           // prune candidate if there are any infrequent subsets
19         if Xj ∈ C(k), for all Xj ⊂ Xab, such that |Xj| = |Xab| − 1 then
20             Add Xab as child of Xa with sup(Xab) ← 0
21     if no extensions from Xa then
22         remove Xa, and all ancestors of Xa with no extensions, from C(k)
23 return C(k)
```

4. Затем, происходит **удаление нечастых** кандидатов (строчка 9).

5. Пережившие это листья префиксного дерева составляют множество частых наборов объектов длины k $\mathcal{F}^{(k)}$, которые используются, чтобы **сгенерировать множество кандидатов длиной $k+1$** для следующего уровня (строчка 10).

6. Процедура **ExtendPrefixTree** применяет расширение на основе префикса для генерации кандидатов

7. Для двух частых наборов длины k $\mathrm{X_a}$ и X_b с общим префиксом длиной $k-1$ (т.е. это два листа с общим родителем), создается кандидат длиной $(k+1)$ - $X_{ab} = X_a \cup X_b$.

8. Кандидат принимается, если он не содержит нечастого подмножества. Наконец, если набор длины k X_a не имеет расширений, он рекурсивно удаляется из префиксного дерева вместе со своими предками, тоже не имеющими расширения.

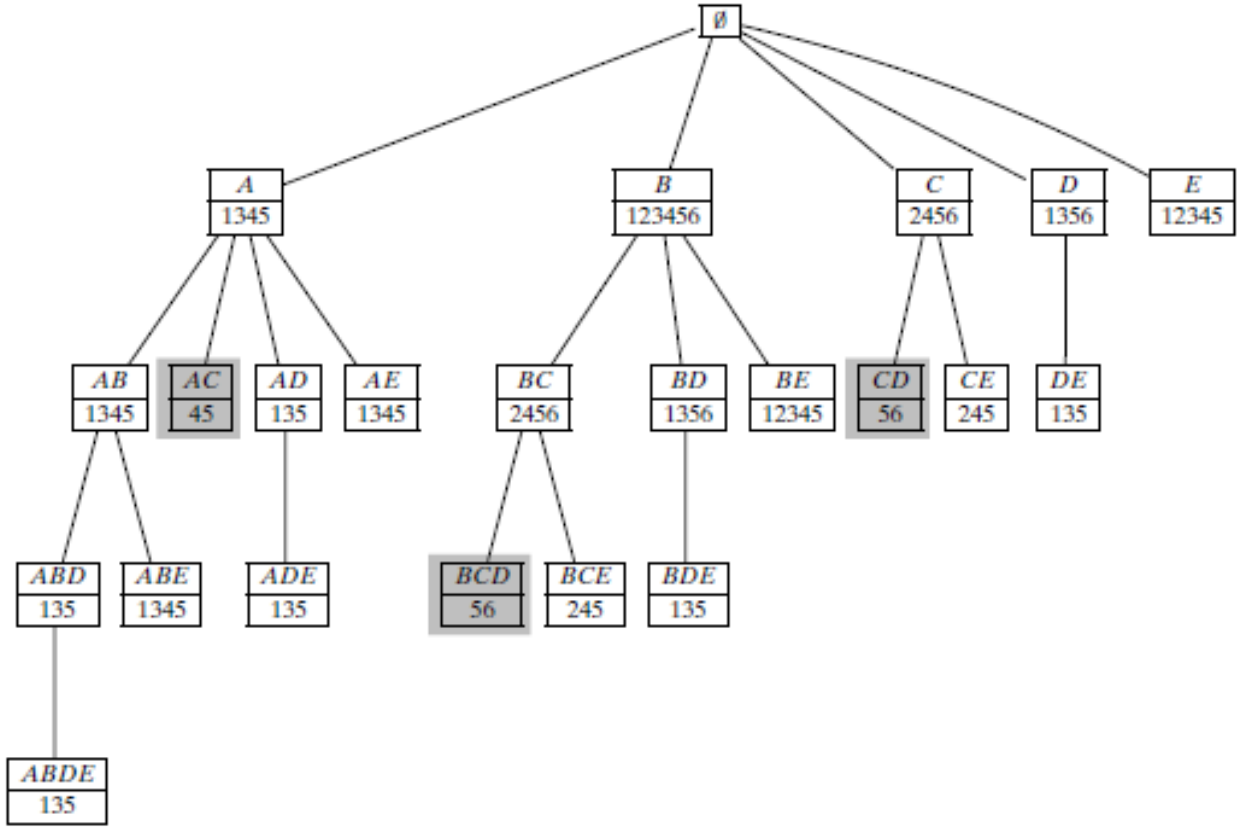
Если на уровне были добавлены новые кандидаты, процесс повторяется для следующего уровня.

Способы систематического улучшения brute-force алгоритма.

Подход пересечения наборов тидов: алгоритм Eclat

Алгоритм Eclat

- Алгоритм Eclat настраивает наборы тидов непосредственно для вычисления поддержки. Основная идея – поддержка каждого набора-кандидата может быть вычислена пересечением наборов тидов для правильно выбранных подмножеств. В общем, если есть $\boldsymbol{t}(X)$ и $\boldsymbol{t}(Y)$ для двух любых частых наборов X и Y, имеем
- $\boldsymbol{t}(XY) = \boldsymbol{t}(X) \cap \boldsymbol{t}(Y)$
- Поддержка кандидата XY есть мощность $\boldsymbol{t}(XY)$, т.е. $\sup(XY) = |\boldsymbol{t}(XY)|$. Eclat пересекает наборы тидов, только если частые наборы объектов имеют общий префикс, и проходит префиксной дерево поиском в глубину, обрабатывая наборы объектов с общим префиксом, также называемый *префиксным классов эквивалентности*.



Вычисления сложность Eclat в худшем случае

$O(|\boldsymbol{D}| \cdot 2^{|\mathcal{I}|})$, поскольку может быть $2^{|\mathcal{I}|}$ частых наборов объектов, и пересечение двух наборов тидов требует максимум $O(|\boldsymbol{D}|)$ времени.

Сложность ввода-вывода Eclat тяжелее определить, т.к. она зависит от размера промежуточных наборов тидов.

Если t – средний размер наборов, изначальный размер базы $O(t \cdot |\mathcal{L}|)$, и общий размер всех промежуточных наборов тидов $O\left(t \cdot 2^{|\mathcal{I}|}\right)$.

Таким образом, Eclat требует $\frac{t \cdot 2^{|\mathcal{I}|}}{t \cdot |\mathcal{I}|} = O\left(\frac{2^{|\mathcal{I}|}}{|\mathcal{I}|}\right)$ сканирований БД в худшем случае.

// Initial Call: $\mathcal{F} \leftarrow \emptyset, P \leftarrow \{\langle i, \mathbf{t}(i)\rangle \mid i \in \mathcal{I}, |\mathbf{t}(i)| \geq minsup\}$

ECLAT (P, $minsup$, $\mathcal{F}$):

```
1 foreach ⟨X_a, t(X_a)⟩ ∈ P do
2     F ← F ∪ {(X_a, sup(X_a))}
3     P_a ← ∅
4     foreach ⟨X_b, t(X_b)⟩ ∈ P, with X_b > X_a do
5         X_ab = X_a ∪ X_b
6         t(X_ab) = t(X_a) ∩ t(X_b)
7         if sup(X_ab) ≥ minsup then
8             P_a ← P_a ∪ {⟨X_ab, t(X_ab)⟩}
9     if P_a ≠ ∅ then ECLAT (P_a, minsup, F)
```

Diffset-ы: разница между наборами тидов

Алгоритм dEclat как оптимизация алгоритма Eclat

Пусть $X_k = \{x_1, x_2, \dots, x_{k-1}, x_k\}$ – набор объектов длины k.

Определим diffset от X_k как множество тидов, содержащих префикс $X_{\{k-1\}} = \{x_1, x_2, \dots, x_{k-1}\}$, но не содержащих объект x_k:

$$\boldsymbol{d}(X_k) = \boldsymbol{t}(X_{k-1}) \backslash \boldsymbol{t}(X_k)$$

Рассмотрим два набора объектов длины k:

$X_a = \{x_1, \dots, x_{k-1}, x_a\}$ и $X_b = \{x_1, \dots, x_{k-1}, x_b\}$, разделяющие префикс

$X = \{x_1, x_2, \dots, x_{k-1}\}$. $\boldsymbol{d}(X_{ab} = X_a \cup X_b = \{x_1, \dots, x_{k-1}, x_a, x_b\})$ определен как:

$$\boldsymbol{d}(X_{ab}) = \boldsymbol{t}(X_a) \backslash \boldsymbol{t}(X_{ab}) = \boldsymbol{t}(X_a) \backslash \boldsymbol{t}(X_b)$$

Заметим что

$$\boldsymbol{t}(X_a)\backslash\boldsymbol{t}(X_b) = \boldsymbol{t}(X_a) \cap \overline{\boldsymbol{t}(X_b)}$$

и, поскольку $\boldsymbol{t}(X) \cap \overline{\boldsymbol{t}(X)} = \emptyset$, можно выразить $\mathbf{d}(X_{ab})$ через $\boldsymbol{d}(X_a)$ и $\boldsymbol{d}(X_b)$ следующим образом:

$$\begin{aligned}
\boldsymbol{d}(X_{ab}) &= \boldsymbol{t}(X_a)\backslash\boldsymbol{t}(X_b) \\
&= \boldsymbol{t}(X_a) \cap \overline{\boldsymbol{t}(X_b)} \\
&= \left(\boldsymbol{t}(X_a) \cap \overline{\boldsymbol{t}(X_b)}\right) \cup \boldsymbol{t}(X) \cap \overline{\boldsymbol{t}(X)} \\
&= \left(\left(\boldsymbol{t}(X_a) \cup \boldsymbol{t}(X)\right) \cap \left(\overline{\boldsymbol{t}(X_b)} \cup \overline{\boldsymbol{t}(X)}\right)\right) \cap \left(\boldsymbol{t}(X_a) \cup \overline{\boldsymbol{t}(X)}\right) \cap \left(\overline{\boldsymbol{t}(X_b)} \cup \boldsymbol{t}(X)\right) \\
&= \left(\boldsymbol{t}(X) \cap \overline{\boldsymbol{t}(X_b)}\right) \cap \overline{\left(\boldsymbol{t}(X) \cap \overline{\boldsymbol{t}(X_a)}\right)} \cap \mathcal{T} \\
&= \boldsymbol{d}(X_b)\backslash\boldsymbol{d}(X_a)
\end{aligned}$$

Вывод:

$\mathbf{d}(X_{ab})$ может быть получен через diffset-ы подмножеств X_a и X_b, что означает, что все операции пересечения в алгоритме Eclat можно заменить на операции с diffset-ами.

Тогда поддержка набора объектов-кандидата может получена:

$$\sup(X_{ab}) = \sup(X_a) - |\boldsymbol{d}(X_{ab})|$$

Вход алгоритма – все частые единичные объекты $i \in \mathcal{I}$ вместе с их $\boldsymbol{d}(i)$, которые вычисляются как

$$\boldsymbol{d}(i) = \boldsymbol{t}(\emptyset) \backslash \boldsymbol{t}(i) = \mathcal{T} \backslash \boldsymbol{t}(i)$$

Алгоритм 8.4. dEclat

// Изначальный вызов: $\mathcal{F} \leftarrow \emptyset$,

$P \leftarrow \{\langle i, d(i), \sup(i)\rangle | i \in \mathcal{I}, d(i) = \mathcal{T} \backslash t(i), \sup(i) \geq minsup\}$.

***d*Eclat**$(P, minsup, \mathcal{F})$:

1. **foreach** $\langle X_a, d(X_a), \sup(X_a)\rangle \in P$ **do**
2. $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, \sup(X_a)\}$
3. $\quad P_a \leftarrow \emptyset$
4. $\quad$ **foreach** $\langle X_b, d(X_b), \sup(X_b)\rangle \in P$, с $X_b > X_a$ **do**
5. $\quad\quad X_{ab} = X_a \cup X_b$
6. $\quad\quad d(X_{ab}) = d(X_b) \backslash d(X_a)$
7. $\quad\quad \sup(X_{ab}) = \sup(X_a) - |d(X_{ab})\|$
8. $\quad\quad$ **if** $\sup(X_{ab}) \geq minsup$ **then**
9. $\quad\quad\quad P_a \leftarrow P_a \cup \{\langle X_{ab}, d(X_{ab}), \sup(X_{ab})\rangle\}$
10. $\quad$ **if** $P_a \neq \emptyset$ **then Eclat**$(P_a, minsup, \mathcal{F})$

Алгоритм 8.4. dEclat

// Изначальный вызов: $\mathcal{F} \leftarrow \emptyset$,

$P \leftarrow \{\langle i, d(i), \sup(i)\rangle | i \in \mathcal{I}, d(i) = \mathcal{T} \backslash t(i), \sup(i) \geq minsup\}$.

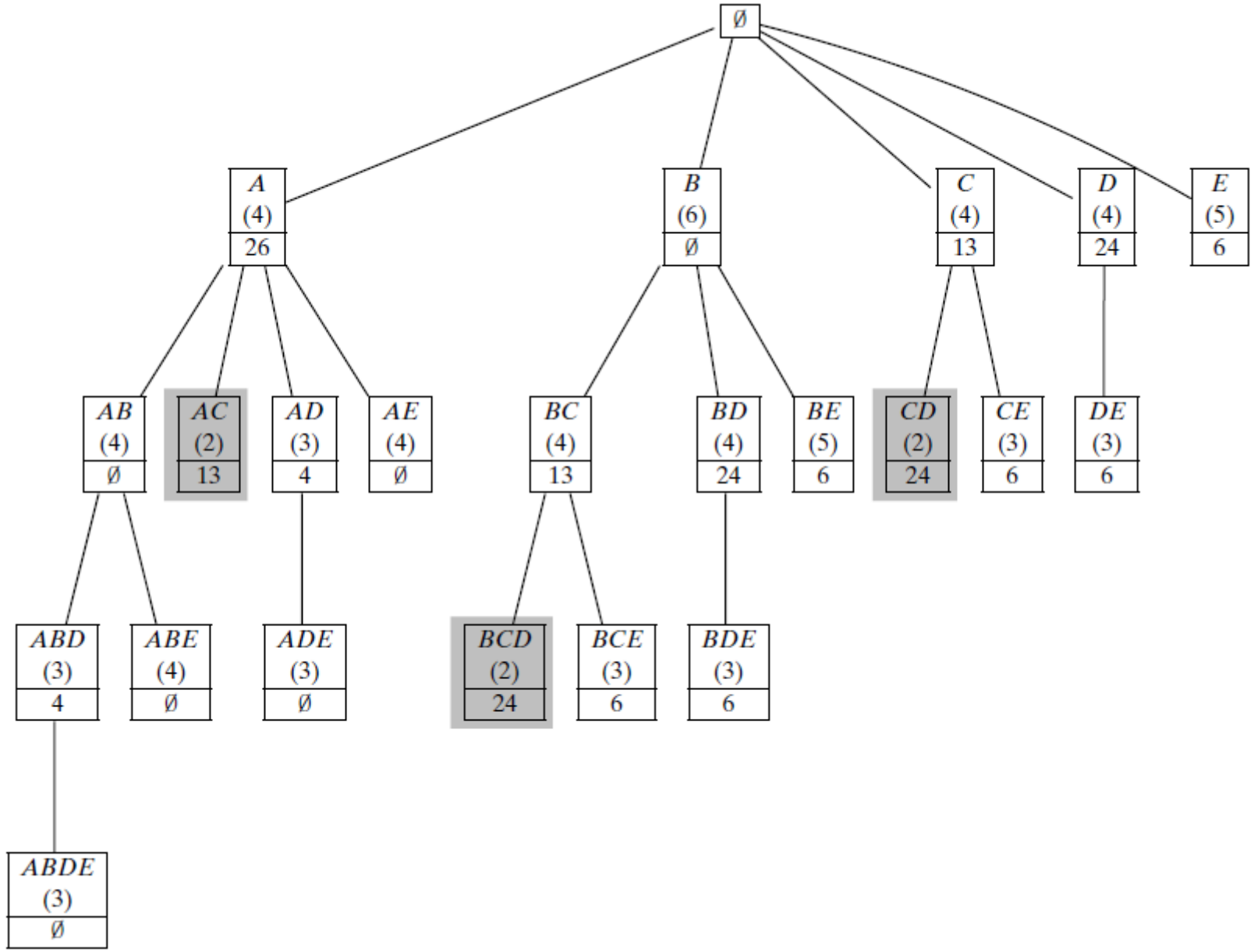
dEclat$(P, minsup, \mathcal{F})$:

1. **foreach** $\langle X_a, d(X_a), \sup(X_a)\rangle \in P$ **do**
2. $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, \sup(X_a)\}$
3. $\quad P_a \leftarrow \emptyset$
4. $\quad$ **foreach** $\langle X_b, d(X_b), \sup(X_b)\rangle \in P$, с $X_b > X_a$ **do**
5. $\quad\quad X_{ab} = X_a \cup X_b$
6. $\quad\quad d(X_{ab}) = d(X_b) \backslash d(X_a)$
7. $\quad\quad \sup(X_{ab}) = \sup(X_a) - |d(X_{ab})\|$
8. $\quad\quad$ **if** $\sup(X_{ab}) \geq minsup$ **then**
9. $\quad\quad\quad P_a \leftarrow P_a \cup \{\langle X_{ab}, d(X_{ab}), \sup(X_{ab})\rangle\}$
10. $\quad$ **if** $P_a \neq \emptyset$ **then Eclat**$(P_a, minsup, \mathcal{F})$

строчки 6-7:

Имея класс эквивалентности P, для каждой отличной пары наборов объектов X_a и X_b генерируют образец-кандидат $X_{ab} = X_a \cup X_b$ и проверяют, является ли он частым с использованием diffset-ов

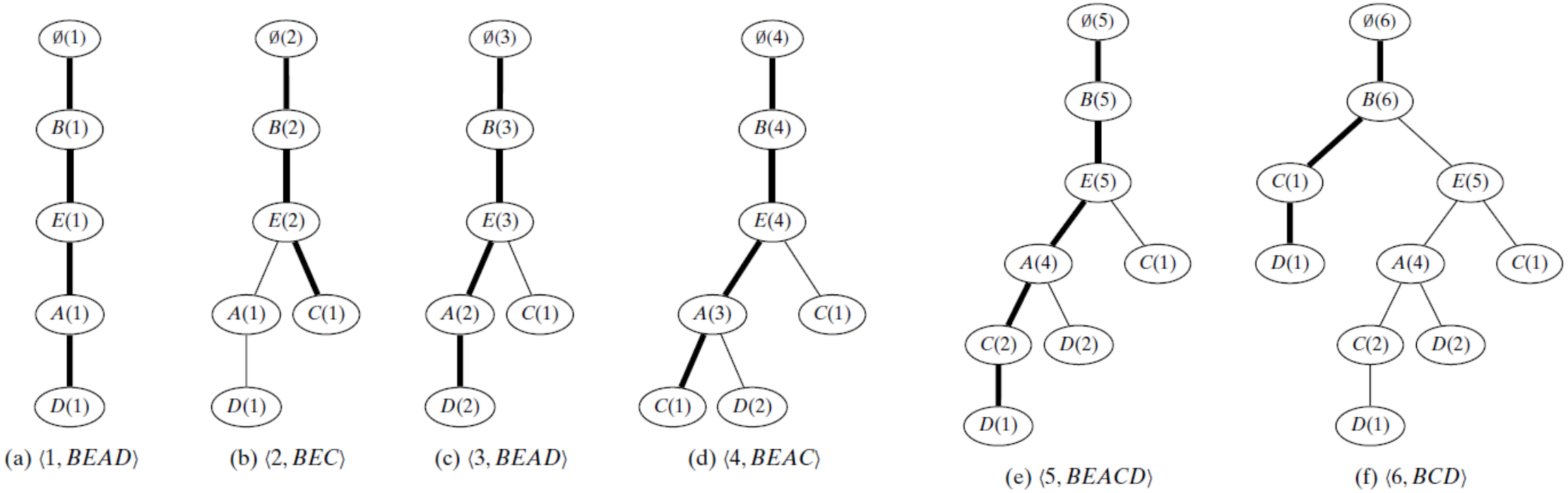
Переключение с наборов тидов на diffset-ы может быть сделано на любом рекурсивном вызове метода.


Ø
A (4) 26
B (6) Ø
C (4) 13
D (4) 24
E (5) 6
AB (4) Ø
AC (2) 13
AD (3) 4
AE (4) Ø
BC (4) 13
BD (4) 24
BE (5) 6
CD (2) 24
CE (3) 6
DE (3) 6
ABD (3) 4
ABE (4) Ø
ADE (3) Ø
BCD (2) 24
BCE (3) 6
BDE (3) 6
ABDE (3) Ø

Подход дерева частоты образцов: алгоритмов FPGrowth

Алгоритм FPGrowth индексирует БД для быстрого вычисления поддержки с использованием расширенного префиксного дерева, называемого *дерево частых образцов* (frequent pattern tree, FP-дерево).

Каждый узел дерева хранит один объект и информацию о наборе объектов, составляющем объекты по пути от корня до этого узла.



(a) ⟨1, *BEAD*⟩ (b) ⟨2, *BEC*⟩ (c) ⟨3, *BEAD*⟩ (d) ⟨4, *BEAC*⟩ (e) ⟨5, *BEACD*⟩ (f) ⟨6, *BCD*⟩

Изначально, дерево содержит пустой элемент $\emptyset$ как корень. Затем, для каждого кортежа $\langle t, X \rangle \in \boldsymbol{D}$,

где $X = \boldsymbol{i}(t)$, мы вставляем наборов объектов X в FP-дерево, увеличивая число на узлах по пути, представляющем X

Если X разделяет префикс с какой-либо ранее вставленной транзакцией, то X будет следовать по тому же пути на протяжении всего общего префикса.

Для оставшихся объектов будут созданы новые узлы с числами, установленными в 1. FP-дерево завершено, когда все транзакции вставлены.

Алгоритм 8.5. Алгоритм FPGrowth

FPGrowth(R, P, $\mathcal{F}$, minsup):

1. Удалить нечастые объекты из R
2. **if** isPath(R) **then** // надо вставить подмножества R В $\mathcal{F}$
3. foreach Y $\subseteq$ R do
4. $X \leftarrow P \cup Y$
5. $\sup(X) \leftarrow \min_{x \in Y}\{cnt(x)\}$
6. $\mathcal{F} \leftarrow F \cup \{(X, \sup(X))\}$
7. else // обработать спроецированные FP-деревья для каждого частого i
8. foreach $i \in R$ в порядке возрастания sup (i) do
9. $X \leftarrow P \cup \{i\}$
10. $\sup(X) \leftarrow \sup(i)$ // сумма $cnt(i)$ для всех вершин с меткой i
11. $\mathcal{F} \leftarrow F \cup \{(X, \sup(X))\}$
12. $R_x \leftarrow \emptyset$ // проецируемое FP-дерево для X
13. **foreach** $path \in$ PathFromRoot(i) **do**
14. $cnt(i) \leftarrow$ число для i в $path$
15. Вставить $path$ без i в FP-дерево R_x с числом $cnt(i)$
16. if $R_x \neq \emptyset$ **then** FPGrowth(R_x, X, $\mathcal{F}$, minsup)

FPGrowth упорядочивает элементы **в порядке убывания поддержки**.

Чтобы спроецировать R на объект i, надо найти все вхождения i в дереве, и для каждого вхождения определить путь от корня до i **(строчка 13).**

Число в элементе i на данном пути записано в $cnt(i)$ **(строчка 14),** и путь вставляется в проецируемое дерево R_X, где X – набор объектов, полученный расширением префикса P объектом i.

При вставке пути, числа на каждом узле в R_x по данному пути увеличиваются на $cnt(i)$, а сам объект i опускается из пути. Полученное FP-дерево – проекция набора объектов X, которая составляет текущий префикс, расширенной объектом i **(строчка 9).**

Затем происходит рекурсивный вызов FPGrowth от R_x и нового X **(строчка 16).**

Когда дерево R – единичный путь, такое дерево обрабатывается перечислением всех наборов объектов, являющихся подмножествами пути, и рекурсивный вызов не происходит.


Генерация правил
ассоциации

Генерация правил ассоциации

Имея коллекцию частых наборов объектов $\mathcal{F}$, для генерации правил ассоциации можно провести итерацию по всех наборам объектов $Z \in \mathcal{F}$, и вычислить уверенность для различных правил, которые могут быть потом выведены из набора объектов. Формально, имея частый набор объектов $Z \in \mathcal{F}$, мы смотрим на все строгие подмножества $X \subset Z$, чтобы вычислить правила вида

$$X \xrightarrow{s,c} Y, \text{ где } Y = Z \backslash X$$

где $\mathrm{Z} \backslash X = Z - X$. Правило должно быть частым, поскольку

$$\mathrm{s} = \sup(XY) = \sup(Z) \geq minsup$$

Таким образом, достаточно проверить, удовлетворяет ли уверенность правила пределу $minconf$. Уверенность вычисляется следующим образом:

$$c = \frac{\sup(X \cup Y)}{\sup(X)} = \frac{\sup(Z)}{\sup(X)}$$

если $c \geq minconf$, то правило – сильное. С другой стороны, если $conf(X \to Y) < c$, то $conf(W \to Z \backslash W) < C$ для всех подмножеств $W \subset X$, где $\sup(W) \geq \sup(X)$. Таким образом, можно избежать проверки подмножеств X.

Алгоритм 8.6. Алгоритм AssociationRules

AssociationRules($\mathcal{F}, minconf$):

1. **foreach** $Z \in \mathcal{F}$, т.ч. $|Z| \geq 2$ **do**
2. $\mathcal{A} \leftarrow \{X | X \subset Z, X \neq \emptyset\}$
3. **w**hile $\mathcal{A} \neq \emptyset$ **do**
4. $X \leftarrow$ максимальный элемент в $\mathcal{A}$
5. $\mathcal{A} \leftarrow \mathcal{A} \backslash X$ // удалить X из A
6. $c \leftarrow \frac{\sup(Z)}{\sup(X)}$
7. **if** $c \geq minconf$ *then*
8. print $\mathrm{X} \rightarrow \mathrm{Y}, \sup(\mathrm{Z}), \mathrm{c}$
9. **else**
10. $\mathcal{A} \leftarrow \mathcal{A} \setminus \{W | W \subset X\}$ // удалить все подможества X и $\mathcal{A}$


Подведение итогов

Подведение итогов

- Пространство поиска для часто встречающихся наборов элементов обычно очень велико и растет экспоненциально с увеличением количества элементов.
- Альтернативный подход, изучаемый в этой главе, состоит в том, чтобы определить сжатые представления часто встречающихся наборов элементов, которые суммируют их основные характеристики.
- Далее рассмотрены три представления: замкнутые, максимальные и невыводимые наборы элементов.

$$\prod_{m=1}^{n-1}(2m-1) = 1 \times 3 \times 5 \times 7 \times \cdots \times (2n-3) = (2n-3)!! \quad (14.1)$$


МАКСИМАЛЬНАЯ И ЗАМКНУТАЯ
ЧАСТОТА НАБОРОВ

Для двоичной базы данных $\mathbf{D} \subseteq \mathcal{T} \times \mathcal{I}$, над тидами $\mathcal{T}$ и элементами $\mathcal{I}$, пусть $\mathcal{F}$ обозначает набор всех частых наборов элементов, то есть

$$\mathcal{F} = \{X | X \subseteq \mathcal{I} \text{ и } \sup(X) \geq \text{minsup}\}$$

Максимально частые наборы элементов

Частое множество $X \subseteq \mathcal{F}$ называется максимальным, если оно не имеет частых надмножеств. Пусть $\mathcal{M}$ будет набором всех максимально частых наборов элементов, заданных как

$$\mathcal{M} = \{X | X \in \mathcal{F} \text{ и } \nexists Y \supset X, \text{такого что } Y \in \mathcal{F}\}$$

Множество $\mathcal{M}$ является сжатым представлением набора всех часто встречающихся элементов $\mathcal{F}$, потому что мы можем определить, является ли какой-либо набор элементов X частым или нет, используя $\mathcal{M}$. Если существует максимальный набор элементов Z такой, что $X \subseteq Z$, то X должен

Замкнутые часто встречающиеся наборы

Функция $\mathbf{t}: 2^{\mathcal{I}} \longrightarrow 2^{\mathcal{T}}$ отображает наборы элементов в наборы тидов, и функция $\mathbf{i}: 2^{\mathcal{T}} \longrightarrow 2^{\mathcal{I}}$ отображает наборы тидов в наборы элементов. То есть, для $T \subseteq \mathcal{T}$ и $X \subseteq \mathcal{I}$, получаем

$$\mathbf{t}(X) = \{t \in \mathcal{T} | \; t \text{ содержит } X\}$$

$$\mathbf{i}(T) = \{x \in \mathcal{I} | \; \forall t \in T, t \text{ содержит } x\}$$

Определим $\mathbf{c}: 2^{\mathcal{I}} \longrightarrow 2^{\mathcal{T}}$ – оператор замыкания, заданный как

$$\mathbf{c}(X) = \mathbf{i} \circ \mathbf{t}(\mathrm{X}) = \mathbf{i}(\mathbf{t}(X))$$

Оператор замыкания **c** отображает наборы элементов в наборы элементов и удовлетворяет следующим трем свойствам:

Экстенсивность: $X \subseteq \mathbf{c}(X)$

Монотонность: Если $X_i \subseteq X_j$, то $c(X_i) \subseteq c(X_j)$

Идемпотентность: $\mathbf{c}\big(\mathbf{c}(X)\big) = \mathbf{c}(X)$

Набор элементов X называется замкнутым, если c(X)=X, то есть если X является неподвижной точкой оператора замыкания c. С другой стороны, если X≠c(X), то X не замкнут, но множество c(X) называется его замыканием.

X, и $\mathbf{c}(X)$ имеют **одинаковый набор тидов**. => X замкнуто, если все надмножества X имеют строго меньшую поддержку, то есть $\sup(\mathrm{X}) \geq \sup(Y)$ для всех $Y \supset X$.

$$\mathcal{C} = \{X | X \in \mathcal{F} \text{ и } \nexists Y \supset X, \text{такого, что } \sup(X) = \sup(Y)\} (9.1)$$

Набор всех замкнутых часто встречающихся наборов элементов $\mathcal{C}$ представляет собой сжатое представление, поскольку мы можем определить, является ли набор элементов X частым, а также точную поддержку X, используя только $\mathcal{C}$. Набор X является частым, если существует замкнутый частый набор элементов $Z \in \mathcal{C}$, такой что $X \subseteq Z$.

$$\sup(X) = \max\{\sup(Z) | Z \in \mathcal{C}, X \subseteq Z\}$$

Следующие отношения поддерживаются между набором всех, замкнутых и максимально частых наборов элементов:

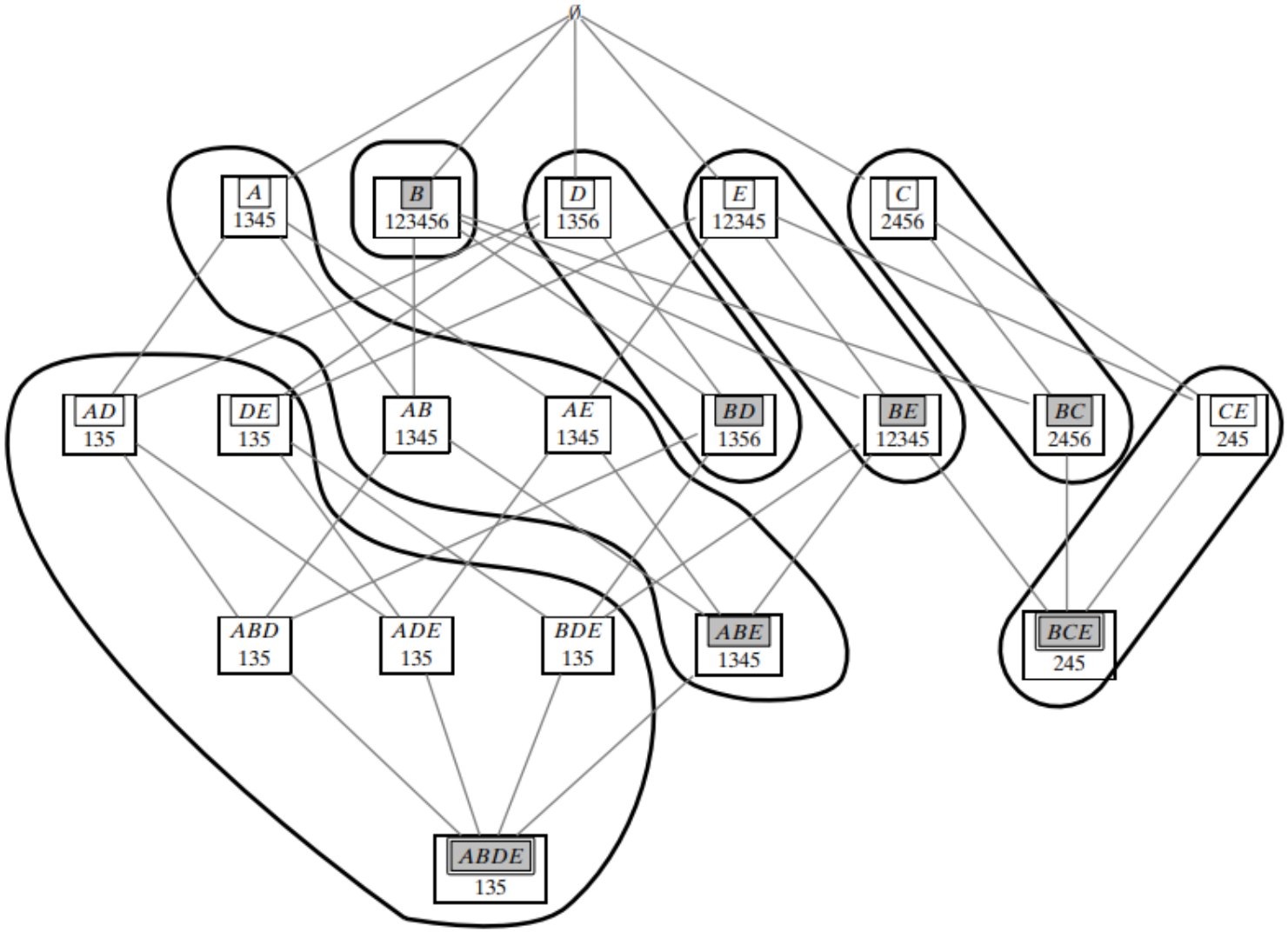
$$\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{F}$$


Минимальные генераторы

Частый набор элементов X является минимальным генератором, если у него нет подмножеств с такой же поддержкой:

$$\mathcal{G} = \{X | X \in \mathcal{F} \text{и } \nexists Y \subset X, \text{такого, что } \sup(X) = \sup(Y)\}$$

Другими словами, все подмножества X имеют строго более высокую поддержку, то есть$\sup(\mathrm{X}) < \sup(Y)$ для всех $Y \subset X$. Понятие минимального генератора тесно связано с понятием замкнутых наборов элементов. Учитывая класс эквивалентности наборов элементов, которые имеют один и тот же набор тидов, замкнутый набор элементов является единственным максимальным элементом класса, а минимальные генераторы – минимальными элементами класса.



Наборы элементов, заключенные в рамки и закрашенные, замкнуты, тогда как наборы внутри рамок (но не закрашенные) являются минимальными генераторами; максимальные наборы элементов показаны двойными линиями.

Из замкнутых наборов элементов максимальными являются $ABDE$ и BCE. Рассмотрим набор элементов AB. Используя $\mathcal{C}$, мы можем определить, что

$$\sup(AB) = \max\{\sup(ABE) \,|\, \sup(ABDE)\} = \max\{4,3\} = 4$$

ПОИСК МАКСИМАЛЬНОЙ ЧАСТОТЫ НАБОРА: АЛГОРИТМ GENMAX

Предположим, что набор максимально частых наборов элементов изначально пуст, то есть $\mathcal{M} = \emptyset$, каждый раз, когда мы генерируем новый набор часто встречающихся элементов X, мы должны выполнять следующие проверки максимальности

Проверка подмножества: $\nexists Y \in \mathcal{M}$, такого что $X \subset Y$. Если такой Y существует, очевидно что X не максимальный. В противном случае добавляем X в $\mathcal{M}$, как потенциально максимальный набор элементов.

Проверка надмножества: $\nexists Y \in \mathcal{M}$, такого что $Y \subset X$. Если такой Y существует, то Y не может быть максимальным, и мы должны исключить его из $\mathcal{M}$.

Проверки на максимальность **занимают время** $\boldsymbol{O}(|\boldsymbol{\mathcal{M}}|)$ => важно следить за эффективностью

Идея

- Любой из часто используемых алгоритмов интеллектуального анализа наборов элементов **может быть расширен** для поиска максимально частых наборов элементов **путем добавления шагов проверки максимальности**.

- Рассмотрим метод GenMax, который основан на подходе пересечений наборов тидов Eclat. Он никогда не вставляет не максимальный набор элементов в $\mathcal{M}$. Таким образом, он исключает проверки надмножества и требует только проверки подмножества для определения максимальности.

- Первоначальный вызов принимает в качестве входных данных набор часто встречающихся элементов вместе с их наборами тидов, $\langle i, \mathbf{t}(i) \rangle$, и изначально пустой набор максимальных наборов элементов, $\mathcal{M}$. Для данного набора пар наборов элементов и наборов тидов, называемых IT-парами, вида $\langle X, \mathbf{t}(X) \rangle$.

ALGORITHM 9.1. Algorithm GENMAX

```
// Initial Call: M ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, sup(i) ≥ minsup}
GENMAX (P, minsup, M):
1  Y ← ⋃ X_i
2  if ∃Z ∈ M, such that Y ⊆ Z then
3      return // prune entire branch
4  foreach ⟨X_i, t(X_i)⟩ ∈ P do
5      P_i ← ∅
6      foreach ⟨X_j, t(X_j)⟩ ∈ P, with j > i do
7          X_ij ← X_i ∪ X_j
8          t(X_ij) = t(X_i) ∩ t(X_j)
9          if sup(X_ij) ≥ minsup then P_i ← P_i ∪ {⟨X_ij, t(X_ij)⟩}
10     if P_i ≠ ∅ then GENMAX (P_i, minsup, M)
11     else if ∄Z ∈ M, X_i ⊆ Z then
12         M = M ∪ X_i    // add X_i to maximal set
```

1–3 проверяем, можно ли обрезать всю текущую ветвь, проверяя, входит ли объединение всех наборов элементов, $Y = \bigcup X_i$, в некоторый максимальный паттерн $Z \in \mathcal{M}$(или содержится в нем).

Если это так, то из текущей ветви невозможно сгенерировать максимальный набор элементов, и он удаляется.

6–9 Если ветвь не отсечена, мы перекрещиваем каждую IT-пару $\langle X_i, \mathbf{t}(X_i)\rangle$ со всеми другими IT-парами $\langle X_j, \mathbf{t}(X_j)\rangle$, при $j > i$, для генерации новых кандидатов X_{ij} , которые добавляются к множеству IT-пар P_i

ALGORITHM 9.1. Algorithm GENMAX

```
// Initial Call: M ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, sup(i) ≥ minsup}
GENMAX (P, minsup, M):
1  Y ← ⋃ X_i
2  if ∃Z ∈ M, such that Y ⊆ Z then
3      return // prune entire branch
4  foreach ⟨X_i, t(X_i)⟩ ∈ P do
5      P_i ← ∅
6      foreach ⟨X_j, t(X_j)⟩ ∈ P, with j > i do
7          X_ij ← X_i ∪ X_j
8          t(X_ij) = t(X_i) ∩ t(X_j)
9          if sup(X_ij) ≥ minsup then P_i ← P_i ∪ {⟨X_ij, t(X_ij)⟩}
10     if P_i ≠ ∅ then GENMAX (P_i, minsup, M)
11     else if ∄Z ∈ M, X_i ⊆ Z then
12         M = M ∪ X_i    // add X_i to maximal set
```

Если P_i не пуст, выполняется рекурсивный вызов GenMax для поиска других потенциально частых расширений X_i.

Если P_i пусто, это означает, что X_i не может быть расширен, и он потенциально максимален. Тогда мы добавляем X_i к множеству $\mathcal{M}$ при условии, что X_i не содержится ни в одном ранее добавленном максимальном множестве $Z \in \mathcal{M}$ (**строка 12**).

По завершении GenMax набор M содержит последний набор всех максимально частых наборов элементов.

Поиск закрытых часто встречающихся наборов элементов: CHARM алгоритм.

Поиск часто встречающихся закрытых наборов элементов требует проверки $X = \boldsymbol{c}(X)$ (проверка закрытия)

Прямая проверка закрытия может быть очень дорогой операцией, так как мы должны были бы проверить, что X является самым большим набором элементов, общим для всех тидов в $\boldsymbol{t}(X)$, то есть, $X = \bigcap_{t \in \boldsymbol{t}(X)} \boldsymbol{i}(t)$.

Альтернативный вариант: метод пересечения вертикальных наборов тидов **CHARM**.
Учитывая совокупность IT-пар $\{\langle X_i , \boldsymbol{t}(X_i)\rangle\}$, выполняются следующие 3 свойства:

1.Если $\boldsymbol{t}(X_i) = \boldsymbol{t}(X_j)$, тогда $\boldsymbol{c}(X_i) = \boldsymbol{c}(X_j) = \boldsymbol{c}(X_i \cup X_j)$=> можем заменить каждое вхождение X_i на $X_i \cup X_j$ и отбросить ветвь под X_j, т.к. ее закрытие идентично закрытию $X_i \cup X_j$.

2. Если $\boldsymbol{t}(X_i) \subset \boldsymbol{t}(X_j)$, тогда $\boldsymbol{c}(X_i) \neq \boldsymbol{c}(X_j)$, но $\boldsymbol{c}(X_i) = \boldsymbol{c}(X_i \cup X_j)$, => можем заменить каждое вхождение X_i на $X_i \cup X_j$, но не можем отбросить X_j, потому что это порождает другое закрытие. Заметим, что если $\boldsymbol{t}(X_i) \supset \boldsymbol{t}(X_j)$, тогда мы просто меняем роль X_i и X_j.

3. Если $\boldsymbol{t}(X_i) \neq \boldsymbol{t}(X_j)$, то $\boldsymbol{c}(X_i) \neq \boldsymbol{c}(X_j) \neq \boldsymbol{c}(X_i \cup X_j)$. В этом случае мы не можем удалить ни X_i ни X_j, так как каждый из них создает свое закрытие.

ALGORITHM 9.2. Algorithm CHARM

```
// Initial Call: C ← ∅, P ← {⟨i, t(i)⟩ : i ∈ I, sup(i) ≥ minsup}
CHARM (P, minsup, C):
1  Sort P in increasing order of support (i.e., by increasing |t(X_i)|)
2  foreach ⟨X_i, t(X_i)⟩ ∈ P do
3      P_i ← ∅
4      foreach ⟨X_j, t(X_j)⟩ ∈ P, with j > i do
5          X_ij = X_i ∪ X_j
6          t(X_ij) = t(X_i) ∩ t(X_j)
7          if sup(X_ij) ≥ minsup then
8              if t(X_i) = t(X_j) then // Property 1
9                  Replace X_i with X_ij in P and P_i
10                 Remove ⟨X_j, t(X_j)⟩ from P
11             else
12                 if t(X_i) ⊂ t(X_j) then // Property 2
13                     Replace X_i with X_ij in P and P_i
14                 else // Property 3
15                     P_i ← P_i ∪ {⟨X_ij, t(X_ij)⟩}
16     if P_i ≠ ∅ then CHARM (P_i, minsup, C)
17     if ∄Z ∈ C, such that X_i ⊆ Z and t(X_i) = t(Z) then
18         C = C ∪ X_i   // Add X_i to closed set
```

Основан на алгоритме Eclat

Принимает в качестве входных набор данных всех часто встречающихся одиночных элементов вместе с их наборами тидов

Изначально множество всех замкнутых наборов элементов, $\mathcal{C}$, пусто

Учитывая любое множество IT-пар $P = \{\langle X_i , \boldsymbol{t}(X_i)\rangle\}$, метод сначала сортирует их в порядке возрастания поддержки

Для каждого набора элементов X_i мы пытаемся расширить его со всеми другими элементами X_j в отсортированном порядке и применяем вышеупомянутые три свойства для отбрасывания ветвей, где это возможно

ALGORITHM 9.2. Algorithm CHARM

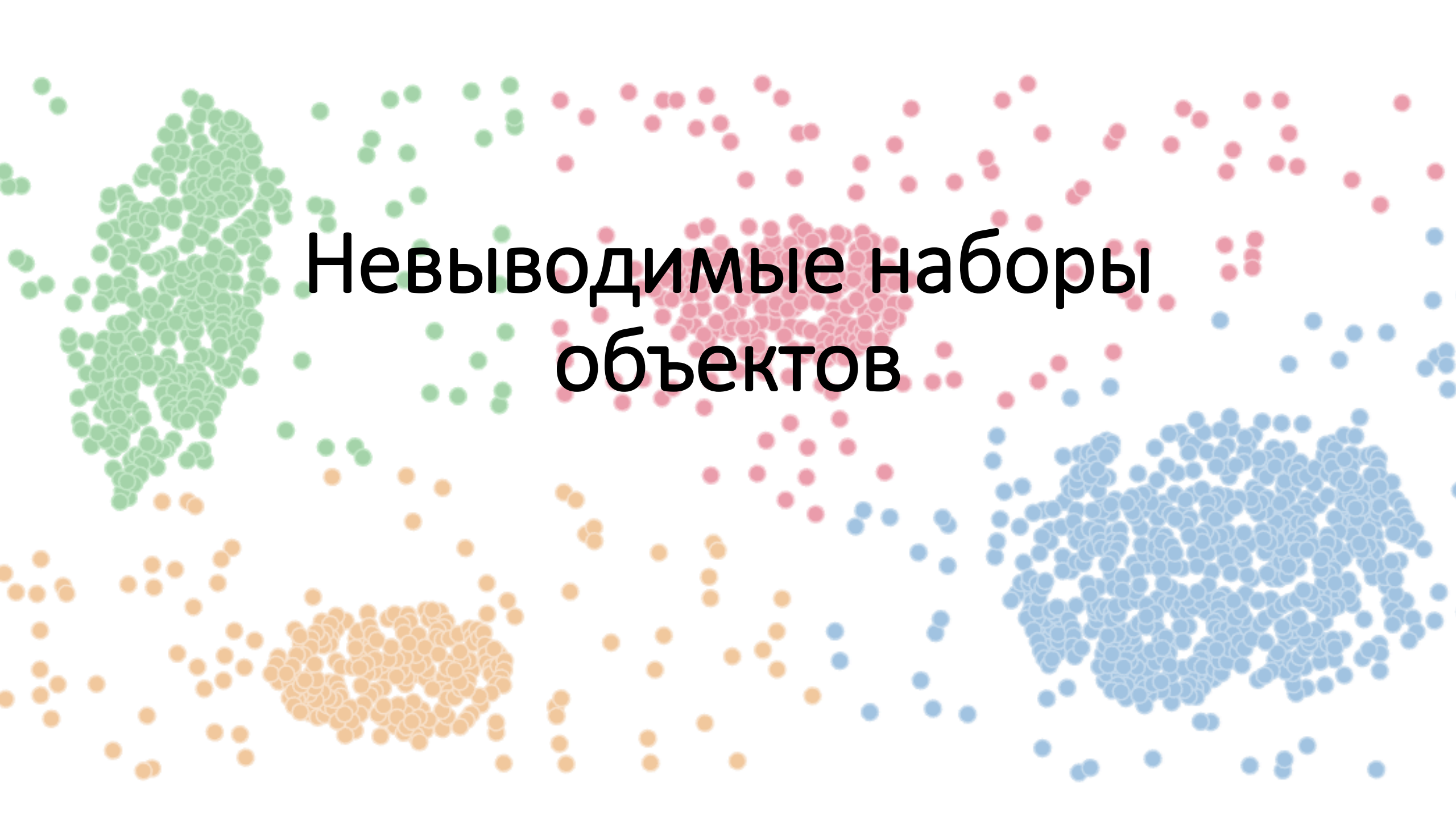
```
// Initial Call: C ← ∅, P ← {⟨i, t(i)⟩ : i ∈ I, sup(i) ≥ minsup}
CHARM (P, minsup, C):
1  Sort P in increasing order of support (i.e., by increasing |t(X_i)|)
2  foreach ⟨X_i, t(X_i)⟩ ∈ P do
3      P_i ← ∅
4      foreach ⟨X_j, t(X_j)⟩ ∈ P, with j > i do
5          X_ij = X_i ∪ X_j
6          t(X_ij) = t(X_i) ∩ t(X_j)
7          if sup(X_ij) ≥ minsup then
8              if t(X_i) = t(X_j) then // Property 1
9                  Replace X_i with X_ij in P and P_i
10                 Remove ⟨X_j, t(X_j)⟩ from P
11             else
12                 if t(X_i) ⊂ t(X_j) then // Property 2
13                     Replace X_i with X_ij in P and P_i
14                 else // Property 3
15                     P_i ← P_i ∪ {⟨X_ij, t(X_ij)⟩}
16     if P_i ≠ ∅ then CHARM (P_i, minsup, C)
17     if ∄Z ∈ C, such that X_i ⊆ Z and t(X_i) = t(Z) then
18         C = C ∪ X_i   // Add X_i to closed set
```

8,12 убедимся, что $X_{ij} = X_i \cup X_j$ является часто встречающимся, проверив мощность $\boldsymbol{t}(X_{ij})$. Если да, то мы проверяем свойства 1 и 2

14 когда свойство 3 выполняется, мы добавляем новое расширение X_{ij} к набору P_i

Если множество P_i не пусто, то мы делаем рекурсивный вызов Charm.

18 если X_i не является подмножеством любого замкнутого множества Z с той же поддержкой, мы можем смело добавить его к множеству замкнутых наборов элементов $\mathcal{C}$


Невыводимые наборы
объектов

Определение

Набор объектов называется **невыводимым,** если его поддержка не может быть выведена из поддержки его подмножеств.

Множество всех невыводимых наборов можно рассматривать как обобщённое или сжатое представление всех часто встречающихся наборов объектов. Кроме того, точная поддержка всех остальных часто встречающихся наборов может быть выведена из этого множества.

Обобщённые наборы элементов

Пусть $\boldsymbol{\mathcal{T}}$ – множество идентификаторов транзакций, $\boldsymbol{\mathcal{I}}$ – множество объектов, $\boldsymbol{X}$ – k-набор, то есть, $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_k}\}$. Рассмотрим набор тидов $\mathbf{t}(x_i)$ для каждого элемента $\boldsymbol{x_i} \in \boldsymbol{X}$.

Эти $\boldsymbol{k}$ наборов разбивают множество всех идентификаторов на $\boldsymbol{2^k}$ подмножеств, каждое такое подмножество содержит идентификаторы для некоторого подмножества объектов $\boldsymbol{Y} \subseteq \boldsymbol{X}$, но не содержит оставшиеся объекты $\boldsymbol{Z} = \boldsymbol{X} \backslash \boldsymbol{Y}$.

Каждое такое подмножество (область) является, таким образом, набором идентификаторов обобщённого набора объектов, включающего объекты из $\boldsymbol{X}$ и их отрицания. Этот обобщённый набор объектов можно записать как $\boldsymbol{Y}\overline{\boldsymbol{Z}}$, где $\boldsymbol{Y}$ состоит из обычных объектов, а $\boldsymbol{Z}$ состоит из отрицаний элементов.

Мы будем называть поддержкой обощённого набора объектов $\boldsymbol{Y}\overline{\boldsymbol{Z}}$ число транзакций, которые содержат все объекты из $\boldsymbol{Y}$, но не содержат объекты из $\boldsymbol{Z}$:

$$\boldsymbol{sup}(\boldsymbol{Y}\overline{\boldsymbol{Z}}) = |\{\boldsymbol{t} \in \boldsymbol{\mathcal{T}} \,|\boldsymbol{Y} \subseteq \boldsymbol{i}(\boldsymbol{t}) \text{ и } \boldsymbol{Z} \cap \boldsymbol{i}(\boldsymbol{t}) = \emptyset\}|.$$

Принцип включения-исключения

Принцип включения-исключения

Пусть $Y\bar{Z}$ – обобщённый набор объектов, а $X = Y \cup Z = YZ$. Принцип включения-исключения позволяет напрямую вычислить поддержку $Y\bar{Z}$ как комбинацию поддержки всех наборов объектов W, таких, что $Y \subseteq W \subseteq X$:

$$sup(Y\bar{Z}) = \sum_{Y \subseteq W \subseteq X} -1^{|W \setminus Y|} * sup(W) . (9.2)$$

Границы поддержки набора элементов

В формуле 9.2 для поддержки $Y\bar{Z}$ есть все подмножества между Y и $X = Y\bar{Z}$.

$$sup(Y\bar{Z}) = \sum_{Y \subseteq W \subseteq X} -1^{|W \backslash Y|} * sup(W) \,. (9.2)$$

Для данного k-набора X существует 2^k обобщённых наборов объектов в виде $Y\bar{Z}$, где $Y \subseteq X$ и $Z = X \backslash Y$, причём у каждого обобщённого набора есть условие для $sup(X)$ в уравнении включения-исключения: это верно для $W = X$.

Установим $sup(Y\bar{Z}) \geq 0$. Однако заметим, что в формуле (9.2) коэффициент при $sup(X)$ равен $+1$ всякий раз, когда $|X \backslash Y|$ чётно, и -1, когда $|X \backslash Y|$ нечётно.

Тогда из возможных 2^k подмножеств $Y \subseteq X$ мы можем вывести 2^{k-1} нижних границ и 2^{k-1} верхних границ для $sup(X)$, полученных после $sup(Y\bar{Z}) \geq 0$ перестановкой членов в уравнении включения-исключения так, что $sup(X)$ оказывается слева, а оставшиеся члены справа:

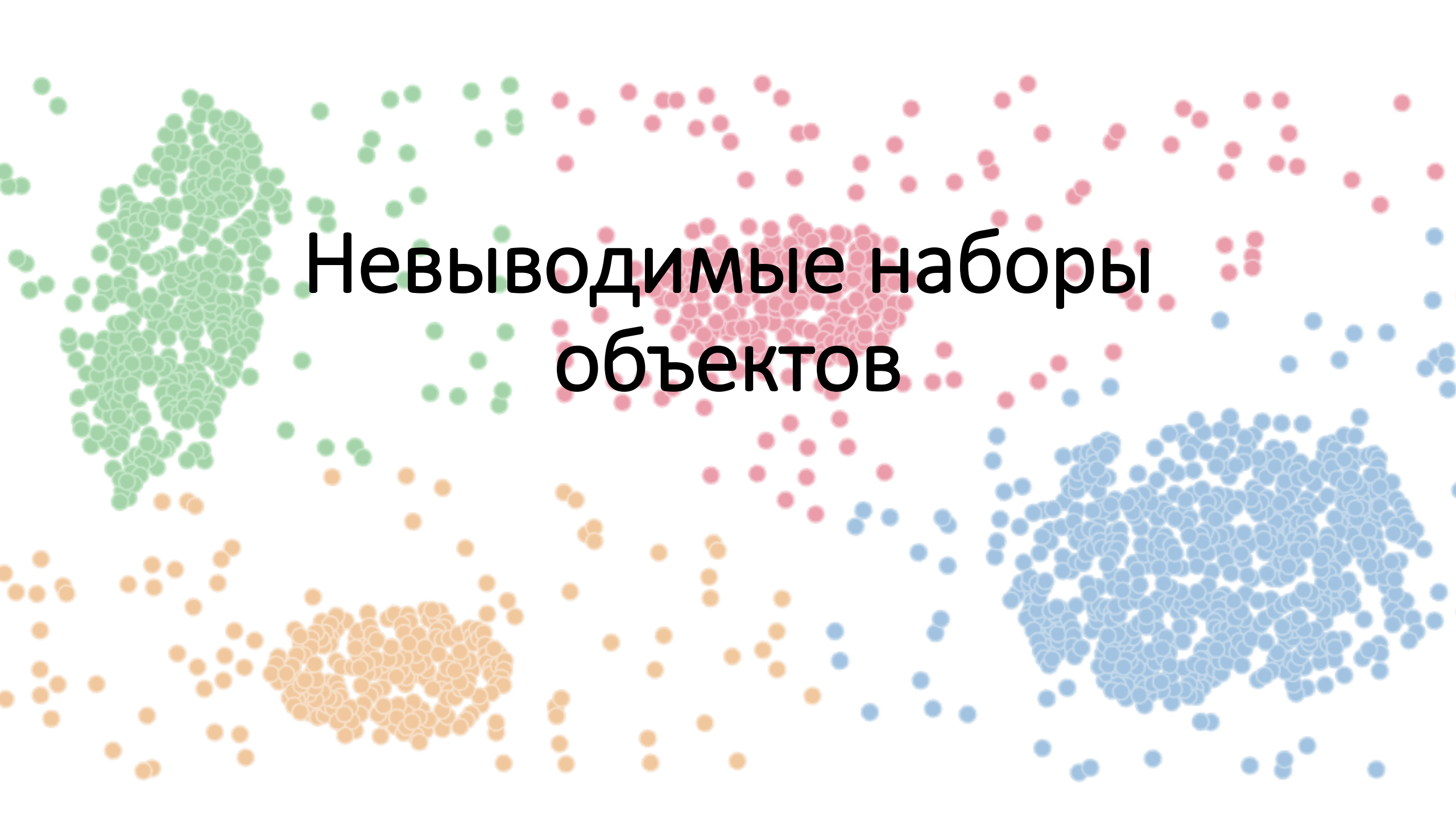
Верхние границы ($|X \backslash Y|$ нечётные):

$$sup(X) \leq \sum_{Y \subseteq W \subseteq X} -1^{(|X \backslash W|+1)} * sup(W)\,, (9.3)$$

Нижние границы ($|X \backslash Y|$ чётные):

$$sup(X) \geq \sum_{Y \subseteq W \subseteq X} -1^{(|X \backslash W|+1)} * sup(W)\,. (9.4)$$

Заметим, что разница между уравнениями (9.3) и (9.4) только в знаке неравенства, который зависит от начального подмножества Y.


Невыводимые наборы
объектов

Пусть дан набор объектов X, пусть $Y \subseteq X$. Обозначим через $IE(Y)$ сумму:

$$IE(Y) = \sum_{Y \subseteq W \subseteq X} -1^{(|X \setminus W|+1)} * sup(W).$$

Тогда множества всех верхних и нижних границ для поддержки $sup(X)$ будут иметь вид:

$$UB(X) = \{IE(Y) \mid Y \subseteq X, |X \setminus Y| \text{ не чётное}\}$$

$$LB(X) = \{IE(Y) \mid Y \subseteq X, |X \setminus Y| \text{ чётное}\}.$$

Набор объектов X называется *невыводимым*, если $\max\{LB(X)\} \neq \min\{UB(X)\}$, то есть, поддержка X не может быть выведена из значений поддержки подмножеств, можно лишь оценить диапазон возможных значений:

$$sup(X) \in [\max\{LB(X)\}, \min\{UB(X)\}].$$

Напротив, X – выводимый набор, если $sup(X) = \max\{LB(X)\} = \min\{UB(X)\}$, поскольку в этом случае поддержка X может быть выведена напрямую из поддержки его подмножеств. Таким образом, набор всех часто встречающихся невыводимых объектов может быть задан как:

$$\mathcal{N} = \{X \in \mathcal{F} \mid \max\{LB(X)\} \neq \min\{UB(X)\}\},$$

где $\mathcal{F}$ - набор всех часто встречающихся объектов.

Принцип включения-исключения