УДК 004.8

А.А. Фильченков, К.В. Фроленков, А.В. Сироткин, А.Л. Тулупьев

СИСТЕМА АЛГОРИТМОВ СИНТЕЗА ПОДМНОЖЕСТВ МИНИМАЛЬНЫХ ГРАФОВ СМЕЖНОСТИ

Фильченков А.А., Фроленков К.В., Сироткин А.В., Тулупьев А.Л. **Система алгоритмов синтеза подмножеств минимальных графов смежности.**

Аннотация. Предложена двухэтапная схема синтеза подмножеств минимальных графов смежности, предполагающая построение трех множеств (стереосепараторов, их владений и обязательных ребер) по множеству подалфавитов и построение по этим четырем множествам множеств жил определенного вида для каждого стереосепаратора. Систематизированы алгоритмы, реализующие оба этапа, и дана оценка их сложности.

Ключевые слова: алгебраическая байесовская сеть, минимальный граф смежности, вторичная структура, глобальное обучение.

Filchenkov A.A., Frolenkov K.V., Sirotkin A.V., Tulupyev A.L. **Minimal join graph subsets synthesis system.**

Abstract. Two-stage scheme for minimal join graph subsets synthesis that involves the construction of three sets (stereoseparators, their possessions and necessary edges) over given subalphabet set with subsequent construction a set of a certain kind of sinews for each stereoseparator. Systematized algorithms, implementing both stages, are systematized and their complexity is estimated.

Keywords: algebraic Bayesian network, minimal join graph, secondary structure, global learning.

1. Введение. Граф смежности — граф со строго определенными свойствами [4, 12, 13, 25], который выступает в качестве вторичной структуры алгебраической байесовской сети (АБС) [3–12, 22]. Графы смежности строятся над первичной структурой АБС, которая в рамках соответствующей задачи формализуется как набор взаимно непоглощающих подалфавитов некоторого алфавита атомарных пропозициональных формул (атомов) [14, 21]. Над заданной первичной структурой в общем случае можно построить множество графов смежности. Особый интерес представляют минимальные графы смежности (МГС), т. е. графы смежности, число ребер которых минимально [25]. Такие графы обладают рядом важных характеристик, которые позволяют их использовать для логико-вероятностного вывода.

На сегодняшний день известно большое число подходов и алгоритмов построения как отдельного МГС, так и всего множества таких графов [1. 15–18, 25]. Помимо этого, в рамках одних и тех же подходов различаются конкретные шаги построения того или иного вспомогательного элемента (например, элементов третичной полиструктуры

АБС), что значительно увеличивает число самих алгоритмов и усложняет их систематизацию.

Все существующие алгоритмы синтеза МГС (как произвольного МГС, так и множества МГС) можно разделить на два класса. Алгоритмы первого класса явно или неявно опираются на структурные особенности графов смежности, формализованные через теорему о множестве минимальных графов смежности [25, 28], которая будет приведена в данной работе. Такие алгоритмы используют сепараторы и строят МГС путем добавления ребер. Алгоритмы из второго класса — жадные — опираются на совпадение множеств минимальных и нередуцируемых графов смежности (т. е. таких графов смежности, их которых нельзя удалить ребро так, чтобы результирующий граф также был графом смежности) [2, 26]. Алгоритмы второго класса строят граф смежности путем удаления ребер из максимального по числу ребер графа смежности [1].

В работе будет предложена систематизация алгоритмов первого класса на основе приведения их к единой функциональной структуре, а также даны оценки сложности этих алгоритмов.

2. Элементы глобальных структур АБС. В данном разделе будет приведен ряд понятий теории АБС, необходимых для описания алгоритмов синтеза МГС. Изложение будет согласовано с терминологией, введенной в работах [21, 25, 28].

Граф — пара $\langle V, E\rangle$, где V — множество вершин графа, а E — множество ребер, каждое из которых является неупорядоченной парой $\{v_i, v_j\}$, $i \neq j$, $v_i, v_j \in V$. Через V и E будем обозначать функции от графа, возвращающие множество его вершин и множество его ребер соответственно: $V(G') = V'$; $E(G') = E'$, где $G' = \langle V', E'\rangle$.

Алфавит — множество атомарных пропозициональных формул: $A = \{x_1, \ldots, x_n\}$. Под *первичной структурой* АБС будем понимать набор взаимно непоглощающих подалфавитов A. Следует указать, что в теории АБС первичной структурой выступает набор максимальных (по включению) фрагментов знаний, фрагментом знаний выступает идеал конъюнктов с заданными над его элементами оценками истинности. При изучении глобальных структур АБС внутренняя организация фрагментов знаний не принимается во внимание, вместо самих фрагментов знаний рассматриваются подалфавиты, над которыми построены их идеалы. Соответственно, в качестве первичной структуры в рассматриваемом случае подразумевается именно набор подалфавитов, а не сами фрагменты знаний.

Для краткости изложения зафиксируем первичную структуру АБС, все вводимые далее объекты будем определять для нее.

Нагруженным графом называется тройка $\langle G, A, W\rangle$, где G — ненаправленный граф, A — алфавит, W — функция нагрузки, заданная на вершинах и ребрах G, принимающая значения из множества 2^A. *Сепаратором* двух вершин в нагруженном графе называется пересечение нагрузок соответствующих вершин: $\mathrm{Sep}(v,u) = W(v) \cap W(u)$. *Согласованным нагруженным графом* называется нагруженный граф, для которого нагрузка каждого ребра $e = \{v,u\}$ совпадает с сепаратором его концов:

$$W(e) = \mathrm{Sep}(v,u).$$

Первичная структура АБС однозначно соотносится с парой A и W: A — это алфавит, над которым задана первичная структура, а значения W для вершин — это подалфавиты, над которыми построены фрагменты знаний. Поскольку первичная структура зафиксирована, согласованный нагруженный граф будем ассоциировать с первым элементом тройки (собственно, графом), явно не указывая остальные ее элементы.

Две вершины называются *сочлененными*, если их сепаратор непуст. *Граф максимальных фрагментов знаний* (*граф МФЗ*) — согласованный лексический граф, в котором ребра возможны только между сочлененными вершинами (т. е. нагрузка любого ребра непуста). *Магистральный путь* между двумя вершинами u и v в согласованном нагруженном графе — такой путь $(u = w_0, w_1, \dots, w_{n-1}, w_n = v)$, что нагрузка любой вершины содержит сепаратор u и v: $\forall w_i\ \mathrm{Sep}(u,v) \subset W(w_i)$. Согласованный лексический граф *магистрально связен*, если между каждой парой сочлененных вершин существует магистральный путь.

Граф смежности — магистрально связный граф МФЗ. В графе смежности возможны ребра только между сочлененными вершинами и между любой парой сочлененных вершин существует магистральный путь. *Минимальный граф смежности* (*МГС*) — граф смежности, число ребер которого минимально. *Максимальный граф смежности* $G_{\max}$ — граф смежности, число ребер которого максимально, такой граф единственен [25]. *Нередуцируемый граф смежности* — граф смежности, который при удалении любого ребра теряет магистральную связность.

Сужение $G \downarrow U$ согласованного лексического графа G на подалфавит U — это ненаправленный граф, в который входят только те вершины и ребра исходного графа G, нагрузки которых содержат или равны U:

$$G \downarrow U = \langle \{V_i | V_i \in V(G), U \subseteq W(V_i)\}, \{E_i | E_i \in E(G), U \subseteq W(E_i)\} \rangle.$$

Под *сужением на сепаратор* U без указания сужаемого графа будем понимать $G_{\max} \downarrow U$.

Сильное сужение $G \Downarrow U$ — сужение $G \downarrow U$, из которого удалили все ребра с нагрузкой U:

$$G \Downarrow U = \langle \{V_i | V_i \in V(G), U \subseteq W(V_i)\}, \{E_i | E_i \in E(G), U \subset W(E_i)\} \rangle.$$

Под *сильным сужением на сепаратор* U будем понимать $G_{\max} \Downarrow U$. Сильное сужение на сепаратор разбивается на компоненты связности, которые называются его *владениями* соответствующих сепараторов.

Рассмотрим произвольный МГС G и произвольный сепаратор U. *Жила с нагрузкой* U — набор ребер с нагрузкой U, число которых равно числу компонент сильного сужения $G_{\max} \Downarrow U$, уменьшенного на единицу, а граф, полученный добавлением к $G_{\max} \Downarrow U$ ребер жилы, является связным. Другими словами, ребра G, нагрузка которых равна U, образуют дерево на владениях сильного сужения $G_{\max} \Downarrow U$. Набор ребер, образующих дерево на владениях сильного сужения $G_{\max} \Downarrow U$, называется *жилой с нагрузкой* U.

Рассмотрим кортеж жил, где для каждого сепаратора встречается ровно одна жила с соответствующей нагрузкой. Объединение элементов такого кортежа называется *пучком*. Известна следующая теорема:

Теорема 1 (о множестве МГС) [25, 28]: Множество минимальных графов смежности совпадает с множеством пучков.

Теорема о множестве МГС используется для построения как множества МГС, так и произвольного МГС. В первом случае следует строить все возможные жилы над владениями $G_{\max} \Downarrow U$, во втором случае — либо произвольные, либо обладающие конкретными свойствами. Подробнее это будет рассмотрено в следующем разделе.

3. Классификация сепараторов. Введем классификацию сепараторов, опираясь на работы [27, 29], значительно переработав ее в сторону упрощения и с учетом изменений в терминологии. Вводимая классификация потребуется в приводимых в следующих разделах алгоритмах.

Моносепаратор — сепаратор, у которого ровно одно владение. Такому сепаратору не соответствует никаких жил.

Моносепаратор с одним ребром (*моносепаратор типа* 1) — это моносепаратор, для которого существует ровно одна пара вершин, для которых он выступает сепаратором.

Дисепаратор — сепаратор, у которого ровно два владения.

Полисепатор — сепаратор, у которого больше, чем два владения.

Сепаратор называется *бездетным*, если нет сепараторов, которые его содержат. С введением понятия родительского графа в следующем разделе определение 5 станет интуитивно понятным.

Утверждение 1. Любой моносепаратор не является бездетным.

Доказательство. Если моносепаратор бездетен, то нагрузка всех ребер, соединяющих вершины в сужении на этот сепаратор, в точности совпадает с моносепаратором, следовательно, каждая вершина будет составлять компоненту связности в сильном сужении. Однако таких вершин не менее двух, тогда как у моносепаратора по определению ровно одно владение.

Обязательное ребро — ребро, которое встречается в каждом минимальном графе смежности.

Утверждение 2. Множество обязательных ребер и множество жил бездетных дисепараторов совпадают.

Доказательство. Для бездетного дисепаратора существует единственная жила, состоящая из единственного ребра, которое соединяет две его вершины, следовательно, эта жила входит в каждый кортеж жил, выбранных по одной для каждого сепаратора, а соответствующее ей ребро входит в каждый МГС.

Рассмотрим произвольное обязательное ребро $\{v, w\}$ и тор U, равный его нагрузке. Пускай v входит в компоненту связности C_v сильного сужения $G_{\max} \downarrow U$, а w — в компоненту связности C_w. Если существует хотя бы одна компонента связности $C' \neq C_w$ то существует жила, в которой у всех ребер один из концов лежит в C'. Такая жила не будет содержать ребра $\{v, w\}$. Если в какой-либо из жил C_v или C_w больше одной вершины — пусть C_w содержит также вершину w', то ребро $\{v, w'\}$ также является жилой. Таким образом, в сужение на сепаратор U входит ровно две вершины, следовательно, U является бездетным дисепаратором.

Замечание. Сужение на бездетный дисепаратор состоит ровно из двух вершин.

Бисепаратор — бездетный дисепаратор.

Стереосепаратор — сепаратор, которому соответствует более одной жилы.

Замечание. Любой сепаратор является либо моносепаратором, либо бисепаратором, либо стереосепаратором.

4. Схема алгоритмов синтеза подмножеств МГС. В данной работе будут рассматриваться алгоритмы синтеза произвольного МГС, всего множества МГС, а также подмножества МГС, содержащего графы смежности с минимальным диаметром. Обобщая, речь пойдет об

алгоритмах синтеза подмножеств МГС, где подмножества могут как совпадать со всем множеством МГС, так и быть одноэлементными. Подобное обобщение позволит компактно изложить материал статьи и привести результаты работы алгоритмов к единому виду.

Через ModelKit будем обозначать четверку $\langle \text{Workloads}, \text{Stereoseparators}, \text{Stereoholding}, \text{NecessaryEdge} \rangle$, где Workloads — множество нагрузок, составляющих первичную структуру, Stereoholding — множество компонент связности сильных сужений на каждый стереосепаратор:

$$\text{Stereoholding} = \{\textbf{COMPONENTS}(G_{\max} \downarrow U) | U \in \text{Stereoseparators}\},$$

где **COMPONENTS** возвращает множество компонент связности графа, NecessaryEdge — множество обязательных ребер:

$$\text{NecessaryEdge} = \bigcup_{U \in \text{BSep}} E(G_{\max} \downarrow U),$$

где BSep — множество бисепараторов.

ModelKit используется как структура, по которой осуществляется синтез как элементов или подмножеств множества МГС, так и самого такого множества. Далее в этом разделе будут описаны алгоритмы синтеза каждого из указанных объектов, а в следующих разделах приведены алгоритмы синтеза самой ModelKit.

Схема алгоритмов синтеза МГС сводится к следующему: сначала одним из возможных способов строится ModelKit (рассмотрению этих способов будут посвящены следующие разделы статьи), затем строятся подмножества МГС, каждый элемент которого синтезируется последовательным перебором всех стереосепараторов, построением жилы для каждого сепаратора и объединением всех построенных жил с множеством обязательных ребер.

Для корректного описания указанных алгоритмов необходимо ввести дополнительные понятия и алгоритмы [31].

CHOOSERANDOM равновероятно возвращает элемент заданного непустого множества. Будем считать, что сложность работы равна $O(1)$.

COMPONENTS для заданного графа возвращает множество подграфов, являющихся его компонентами связности. Сложность работы равна $O(V + E)$, где V — число вершин, а E — число ребер в графе.

REVERSEDCOMPONENTS для заданного графа возвращает множество подграфов, являющихся компонентами связности для обратного графа (т.е. графа, в котором смежны те и только те вершины, которые не смежны в исходном). Сложность работы равна $O(V^2)$, где V — число вершин в графе.

PRÜFERTREES для заданного набора вершин возвращает множество возможных деревьев над этими вершинами. Реализуется на основе алгоритма Прюфера [32]. Всего для V вершин можно построить V^{V-2} деревьев, сложность работы алгоритма — $O(V^{V-1})$.

Объединяющей алгебраической сверткой набора множеств $S = \{S_i\}_{i=1,..,n}$, элементы которых также являются множествами, будем называть следующее множество:

$$\text{UnitingAlgebraicFolding}(S) = \left\{ \bigcup_{i=1,..,n} s_{m_i} \middle| \forall i \; s_{m_i} \in S_i \right\}.$$

Иначе говоря, объединяющая алгебраическая свертка состоит из множества элементов, которые являются объединением элементов кортежей декартового произведения набора множеств.

UNITINGALGEBRAICFOLDING для заданного набора множеств множеств возвращает объединяющую алгебраическую свертку этого набора. Рассмотрим n множеств, в которых $c_1, c_2, \ldots, c_n$ элементов соответственно, тогда сложность построения объединяющей алгебраической свертки для такого набора равна $O\left(n \prod_{i=1,\ldots,n} c_i\right)$, поскольку для построения каждого элемента свертки требуется объединить n элементов из множеств, а всего элементов в свертке ровно $\prod_{i=1,\ldots,n} c_i$.

SINEWS — абстрактная функция, которая для заданного типа жил T и множества владений одного сепаратора (и, возможно, дополнительных аргументов) строит множество жил указанного типа соответствующего сепаратора. Варианты реализации функции **SINEWS** будут приведены далее.

В подразделе 5.3 вместе с порядком на множестве сепараторов будет введен алгоритм **SORTSEPARATORS**, сложность которого равна $O(ms \ln s)$, где m — максимальное число атомов во фрагменте знаний, а s — число непустых сепараторов.

UNITEDISJOINTSETS объединяет два непересекающихся множества. Сложность работы равна $O(1)$.

ISCHILDLESS для заданного сепаратора и заданного множества непустых сепараторов определяет, истинно ли, что у первого нет детей (что то же самое — потомков) среди указанного множества. Сложность работы равна $O(ms)$, где m — максимальное число атомов во фрагменте знаний, а s — число непустых сепараторов, поскольку в худшем случае придется сравнить за $O(m)$ заданный сепаратор со всеми сепараторами множества.

Будем предполагать, что все множества задаются хэш-таблицами, в которые добавление и удаление элемента осуществляются за $O(1)$,

пересечение, объединение и разность двух множеств — за длину меньшего из множеств. Множества не упорядочены, сортировка требует $O(n \ln n)$, где n — размер множества.

Алгоритмы синтеза подмножеств МГС по ModelKit будем называть *алгоритмами сборки*. Алгоритм приведен на листинге 1.

```
MJGAssembling
Require: ⟨Workloads, Stereoseparators, Stereoholding, NecessaryEdge⟩, T
Ensure: Stereoseparators ∪ NecessaryEdge ≠ ∅
 1: for all u ∈ Stereoseparators do
 2:     Sinews[u] ←Sinews(T, Stereoholding[u])
 3: end for
 4: EdgeSets ←UnitingAlgebraicFolding(Sinews)
 5: MJGSubset ← ∅
 6: for all E ∈ EdgeSets
 7: MJGSubset ←
        MJGSubset ∪ {⟨Workloads, UniteDisjointSets(E, NecessaryEdge)⟩}
 8: end for
 9: if EdgeSets = ∅ do
10:     MJGSubset ← {⟨Workloads, NecessaryEdge⟩}
11: end if
12: return MJGSubset
```

Листинг 1. Алгоритм сборки подмножества МГС.

Алгоритм по ModelKit и типу жил T строит множество жил указанного типа для заданного непустого сепаратора. Для этого для каждого стереосепаратора строятся все жилы указанного типа (1–3), а затем строится объединяющая алгебраическая свертка построенных множеств (4). Элементы подмножества минимальных графов смежности строятся за счет построения графов, вершинами которого являются нагрузки первичной структуры, а ребра являются объединением ребер каждого элемента объединяющей алгебраической свертки и множества обязательных ребер (6–8). Если же множество стереосепараторов (а вслед за ним и результат объединяющей алгебраической свертки) оказалось пусто, то множество минимальных графов смежности состоит из единственного графа, множество ребер которого совпадает с множеством обязательных ребер (9–11).

Утверждение 3. Сложность работы алгоритма сборки подмножества МГС равна $\sum_{u \in \mathrm{Sep}} S_u^T + O\left(s \prod_{u \in \mathrm{Sep}} M_u^T\right)$, где S_u^T — сложность построения множества жил указанного типа T для сепаратора u, s —

число непустых сепараторов, а M_u^T — мощность множества жил типа T.[1].

Доказательство. Сложность работы шага 4 равна $O\left(s\prod_{u\in\mathrm{Sep}} M_u^T\right)$ по определению алгоритма объединяющей алгебраической свертки, так как число элементов этой свертки равно $\prod_{u\in\mathrm{Sep}} M_u^T$. Сложность работы шага 7 равна $O(1)$, поскольку такова сложность **UNITEDISJOINTSETS** и добавления элемента к множеству, поэтому сложность цикла 6–8 равна числу его шагов, которых ровно $\prod_{u\in\mathrm{Sep}} M_u^T$. Наконец, сложность работы условного оператора 9–11 равна $O(1)$.

В дальнейшем для алгоритмов синтеза подмножеств МГС будем выражать оценку их сложности через мощности множеств жил соответствующего типа.

В качестве функции **SINEWS** могут выступать, в том числе, и три алгоритма, приводимые ниже.

Алгоритм построения множества всех жил по заданному множеству владений строит все возможные жилы соответствующего сепаратора. Алгоритм опирается на алгоритмы, приведенные в работах [15–18]. Алгоритм приведен на листинге 2.

SINEWSALL
Require: Holdings
Ensure: Holdings $= \{C_0, \dots, C_{h-1}\}; h > 0;\ |c_i| > 1\ \ \forall i \in 0, \dots, h-1$
```
 1: Sinews ← ∅
 2: Homages ← PRÜFERTREES(Holdings)
 3: for all H ∈ Homages do
 4:    for i ← 0 to |Holdings| − 2
 5:       TreeRealisationEdge[i] ← ∅
 6:    end for
 7:    i ← 0
 8:    for all (U,V) ∈ E(H) do
 9:       for all u ∈ Holdings[U], v ∈ Holdings[V] do
10:          TreeRealisationEdge[i] ← TreeRealisationEdge[i] ∪ {{u, v}}
11:       end for
12:       i ← i + 1
13:    end for
```

[1]Следует отметить, что в каждом случае мощность синтезируемого подмножества известна до начала работы алгоритмы, поэтому ее можно отнести к исходным параметрам.

```
14: Sinews ← UniteDisjointSets(Sinew, UnitingAlgebraicFolding(TreeRealisationEdge))
15: end for
16: return Sinews
```

Листинг 2. Алгоритм построения множества всех жил для заданного сепаратора.

Получая на входе множество владений Holdings, алгоритм строит множество всевозможных деревьев над владениями (2), затем для каждого дерева H перебирает все его ребра (8–13) и для каждого ребра такого дерева перебирает все его реализации на владениях (9–11). Применение объединяющей алгебраической свертки на шаге 14 строит из множества таких ребер множество жил, реализующих соответствующее дерево на владениях, это множество добавляется к множеству уже построенных жил рассматриваемого стереосепаратора.

Утверждение 4. Сложность работы алгоритма **SinewsAll** принадлежит классу

$$O((hc^2)^{h-1}),$$

где h — число владений сепаратора, а c — максимальное число вершин во владении.

Доказательство. Шаг (2) имеет сложность $O(h^{h-1})$. Всего деревьев на владениях ровно h^{h-2}. Для каждого дерева осуществляется цикл 4–6, требующий $h-2$ шага, а также цикл 8–13, в котором для каждого ребра дерева (которых $h-1$) перебираются все возможные реализации этого ребра как пары вершин из соответствующих владений. Всего во владении не более c вершин, поэтому для каждого ребра существует не более c^2 реализаций, как следствие, для каждого дерева — не более $(c^2)^{h-1} = c^{2h-2}$ реализаций. Следовательно, сложность одного прохода цикла 3–13 равна $O(c^{2h-2} + h)$. Сложность выполнения для каждой реализации дерева **UnitingAlgebraicFolding** шага 14 равна $O(hc^{2h-2})$, поскольку для каждого ребра существует не более c^2 реализаций, следовательно, объединяющая алгебраическая свертка применяется к $h-1$ множествам, в которых не более c^2 элементов. Объединение выполняется за $O(1)$. Следовательно, сложность выполнения цикла 3–15 для одного дерева равна $O(hc^{2h-2})$, всего таких деревьев h^{h-2}, поэтому сложность выполнения всего алгоритма равна

$$O((hc^2)^{h-1}).$$

Следствие 1. $S_u^{T_{\mathrm{All}}} = O\left(hM_u^{T_{\mathrm{All}}}\right)$, где T_{All} обозначает тип жил, соответствующих всем жилам.

Звездой называется дерево, в котором степень всех вершин, кроме одной, равна единице (а степень последней, соответственно, на едини-

цу меньше общего числа вершин в дереве). Жилы в форме звезды используются для построения минимальных графов смежности, позволяющие эффективнее искать минимальные графы смежности с минимальным диаметром [23].

Локально звездчатым графом смежности называется граф смежности, в котором каждая жила является звездой.

Звездчатым графом смежности называется локально звездчатый граф смежности, в котором для каждого сепаратора вершины соответствующей жилы являются доменными вершинами или центрами звезд жил, соответствующих сыновьям этого сепаратора.

Множество звездчатых графов смежности содержит минимальные графы смежности с минимальным диаметром [23], соответственно, локально звездчатые графы также содержат минимальные графы смежности с минимальным диаметром. Локально звездчатые графы смежности отличаются от звездчатых тем, что выбор жил производится независимо, тогда как выбор жил при построении звездчатых графов осуществляется последовательно и зависит от выбора предыдущих жил. В рамках настоящей работы мы рассмотрим алгоритмы синтеза всего множества локально звездчатых графов смежности. Алгоритм построения звездчатых графов смежности несложно получить за счет добавления таблицы, в которой для каждой жилы хранится ее центр и последовательным перебором стереосепараторов.

Алгоритм построения множества звезд-жил по заданному множеству владений строит все возможные жилы соответствующего сепаратора, являющиеся звездами. Алгоритм приведен на листинге 3.

SinewsStars
Require: Holdings
Ensure: Holdings $= \{C_0, \dots, C_{h-1}\}; h > 0;\ |C_i| > 1\ \ \forall i \in 0, \dots, h-1.$

```
 1: Sinews ← ∅
 2: for all C ∈ Holdings do
 3:   for all u ∈ C do
 4:     EdgeRealisation ← ∅
 5:     for all C′ ∈ Holdings, C ≠ C′ do
 6:       EdgeRealisation[C′] ← ∅
 7:       for all v ∈ C′ do
 8:         EdgeRealisation[C′] ← EdgeRealisation[C′] ∪ {(u, v)}
 9:       end for
10:     end for
11:     Sinews ← Sinews ∪UnitingAlgebraicFolding(EdgeRealisation)
```

```
12:   end for
13: end for
14: return Sinews
```

Листинг 3. Алгоритм построения множества жил–звезд для заданного сепаратора.

Получив на входе множество владений Holdings, алгоритм перебирает все владения, затем для каждого выбранного владения перебирает все вершины, которые в него входят (3–12), и для каждой выбранной вершины v перебирает все остальные владения (5–10), для каждого из которых строит множество EdgeRealisation всевозможных ребер, соединяющих v с вершинами этого владения (7–9). Множество жил строится как объединение объединяющих алгебраических сверток множеств EdgeRealisation для каждой вершины, выбранной как центр звезды (11).

Утверждение 5. Сложность работы алгоритма **SINEWSSTARS** принадлежит классу $O(h^2c^h)$, где h — число владений сепаратора, а c — максимальное число вершин во владении.

Доказательство. Цикл 2–13 выполняется для каждого владения C (всего их h), цикл 3–12 выполняется для каждой вершины v владения C (всего их не более c). Для каждого владения, отличающегося от C (их ровно $h-1$) в цикле 5–10 перебираются все пары ребер, соединяющих v с вершиной этого владения (таких ребер не более c). Таким образом, будет построено $h-1$ множество с не более чем c элементами в каждом, поэтому объединяющая алгебраическая свертка на шаге (11) имеет сложность $O(hc^{h-1})$. Следовательно, сложность работа алгоритма равна $O(h^2c^h)$.

Следствие 2. $S_u^{T_{\text{Star}}} = O\left(hM_u^{T_{\text{Star}}}\right)$, где T_{Star} обозначает тип жил, соответствующих жилам–звездам.

Алгоритм рандомизированного синтеза минимального графа смежности по множеству владений сепаратора строит реализацию случайного минимального графа смежности, причем каждая реализация может быть построена с ненулевой вероятностью [24]. Алгоритм приведен на листинге 4.

SINEWSRANDOMONE

Require: Holdings

Ensure: Holdings $= \{c_0, \dots, c_{h-1}\}$; $h > 0$; $|c_i| > 1 \quad \forall i \in 0, \dots, h-1$.

```
1:   C ← CHOOSERANDOM(Holdings)
2:   NotYetConnectedH ← Holdings\{C}
3:   AlreadyConnectedV ← V(C)
```

```
 4:    Edges ← ∅
 5:    while NotYetConnectedH ≠ ∅ do
 6:       C ← ChooseRandom(Holdings)
 7:       v ← ChooseRandom(AlreadyConnectedV)
 8:       u ← ChooseRandom(C)
 9:       Edges ← Edges ∪ {v, u}
10:       NotYetConnectedH ← NotYetConnectedH\{C}
11:       AlreadyConnectedV ← AlreadyConnectedV ∪ V(C)
12:    end while
13: return {Edges}
```

Листинг 4. Алгоритм случайного построения жилы для заданного сепаратора.

По множеству владений Holding алгоритм строит два множества: множество еще не соединенных владений NotYetConnectedH, в которое изначально входят все владения, кроме случайно выбранного C (1–2), а также множество уже соединенных вершин, в которое изначально входят только вершины из C (3). Затем последовательно из множества еще не соединенных владений произвольным образом выбирается владение (6), из которого случайным образом выбирается вершина (8), которая, в свою очередь, соединяется ребром со случайно выбранной вершиной из множества уже соединенных вершин (7). Это ребро добавляется к множеству уже построенных ребер жилы (9). Таким образом, выбранное владение оказывается в том же компоненте связности, что и выбранное в самом начале владение C, поэтому оно удаляется из множества еще не соединенных владений (10), а его вершины добавляются к множеству уже соединенных вершин (11). Цикл из описанных шагов (5–12) работает до тех пор, пока не будут перебраны и соединены все владения.

Утверждение 6. Сложность работы алгоритма **SinewsRandomOne** принадлежит классу $O(ch)$, где h — число владений сепаратора, а c — максимальное число вершин во владении.

Доказательство. В цикле (5–12) перебираются все элементы множества NotYetConnected, которое изначально (2) было инициализировано Holdings\{C}, т.е. в нем $h - 1$ элемент. На каждом шаге цикла из указанного множества удаляется один элемент (10), поэтому число шагов в цикле равно числу элементов множества. Все шаги в цикле, кроме (11), имеют сложность $O(1)$, тогда как шаг (11) имеет сложность $O(c)$, потому что мощность $V(C)$ — вершин в компоненте связности — ограничена c. Таким образом, сложность работы алгоритма равна $O(ch)$.

Следствие 3. $S_u^{T_{\text{rand}}} = O\left(chM_u^{T_{\text{rand}}}\right)$, где T_{rand} обозначает реализацию случайной жилы.

Поскольку строится всего один минимальный граф смежности, а сложность построения превосходит $O(h)$, алгоритм не является эффективным, однако поскольку само подмножество одноэлементно, это не является критичным для скорости синтеза такого подмножества.

5. Вспомогательные алгоритмы синтеза.

5.1. Общая схема синтеза ModelKit. Используемые алгоритмы синтеза множества МГС явно опираются на синтез ModelKit [15–18]. Однако известны и иные алгоритмы синтеза множества МГС, а также алгоритмы синтеза произвольного графа смежности, которые не используют ModelKit [25]. Все такие алгоритмы могут быть сведены к алгоритмам, строящим ModelKit и использующим приведенные в предыдущем разделе алгоритмы сборки. Для этого необходимо уже после построения множества всех жил для каждого сепаратора за один проход удалить все те сепараторы, для которых это множество пусто или содержит ровно одну жилу. Для второго случая все такие жилы необходимо объединить в множество обязательных ребер. Сложность такого дополнительного построения предполагает единичный проход по множеству всех построенных сепараторов, поэтому линейно зависит от его размера.

Для каждого описываемого далее алгоритма входными данными является набор нагрузок Workloads, который соответствует подалфавитам первичной структуры. Отметим, что такой набор как первичная структура должен состоять из максимальных по включению подалфавитов:

$$\forall w_i, w_j \in \text{Workloads}, i \neq j \Rightarrow w_i \not\subseteq w_j.$$

Семейство таких множеств обозначим как **PrimaryStructures**.

Всевозможные алгоритмы синтеза элементов ModelKit состоят из следующих шагов:

1. Построение подмножества непустых сепараторов — Separators.
2. Построение множеств вершин, входящих в сужение на каждую нагрузку из Separators — $\text{Vertices}[s], s \in \text{Separators}$ и построение множества значимых ребер NecessaryEdge.
3. Построение компонент связности сильных сужений на каждую нагрузку из Separators — $\text{Holdings}[s], s \in \text{Separators}$ и построение множества стереосепараторов Stereoseparators.

При этом в разных алгоритмах, приведенные выше, шаги могут следовать в иной последовательности или выполняться одновременно — цель таких изменений состоит в ускорении работы алгоритмов за счет уменьшения числа необходимых шагов. Однако подобная вариативность усложняет описание всего семейства алгоритмов.

5.2. Вспомогательные алгоритмы. Введем вспомогательные алгоритмы, основываясь на работах [15–20].

Прежде всего, рассмотрим алгоритм построения *множества непустых сепараторов*, приведенный на листинге 5, который по множеству нагрузок, соответствующих первичной структуре, строит множество непустых сепараторов, а также, дополнительно, таблицу, в которой для каждой пары нагрузок хранится их сепаратор.

SeparatorSet
Require: Workloads
Ensure: Workloads ∈ **PrimaryStructure**
1: Separators ← ∅
2: **for all** $w, w' \in$ Workloads, $w \neq w'$ **do**
3: SeparatorTable$[w, w']$ ← $w \cap w'$
4: **if** $w \cap w' \neq \emptyset$ **do**
5: Separators ← Separators ∪ SeparatorTable$[w, w']$
6: **end if**
7: **end for**
8: **return** ⟨Separators, SeparatorTable⟩

Листинг 5. Алгоритм построения множества непустых сепараторов.

Получив на входе множество нагрузок Workloads, алгоритм перебирает все пары нагрузок (2–7). Пересечение каждой пары записывается в таблицу SeparatorTable (3), и если оно не пусто, то соответствующий сепаратор добавляется к множеству сепараторов (4–6).

Утверждение 7. Сложность работы алгоритма **SeparatorSet** принадлежит классу $O(mw^2)$, где w — число фрагментов знаний в первичной структуре, а m — максимальное число атомов во фрагменте знаний.

Доказательство. Всего пар вершин $\frac{w(w-1)}{2}$,. Для каждой пары вершин на шаге 3 берется их пересечение, что имеет сложность $O(m)$, на шаге 4 производится сравнение за $O(1)$ и на шаге 5 —добавление элемента к множеству, что также производится за $O(1)$, таким образом, сложность равна $O(mw^2)$.

Вслед за этим рассмотрим *алгоритм построения множества вершин, входящих в сужение на заданный непустой сепаратор*, представленный на листинге 6.

```
NarrowedVertices
Require: s, Workloads
Ensure: s ∈ Separators
 1: NarrowedVertices ← ∅
 2: for all w ∈ Workloads do
 3:   if s ⊂ w do
 4:     NarrowedVertices ← NarrowedVertices ∪ {w}
 5:   end if
 6: end for
 7: return NarrowedVertices
```

Листинг 6. Алгоритм построения множества вершин сужения.

Получив на вход сепаратор s и множество нагрузок, алгоритм для каждой нагрузки проверяет, содержится ли в ней сепаратор, и если да, то она добавляется в множество вершин, входящих в сужение на соответствующий сепаратор (4).

Утверждение 8. Сложность работы алгоритма **NarrowedVertices** принадлежит классу $O(mw)$, где m — максимальное число атомов во фрагменте знаний, а w — число фрагментов знаний в первичной структуре.

Доказательство. Для каждой нагрузки на шаге (3) выполняется проверка, содержит ли одно множество другое. Поскольку размер каждого из множеств не больше m, то одна проверка выполняется за $O(m)$. Добавление элемента к множеству (4) выполняется на $O(1)$. Число шагов в цикле в точности равно w, таким образом, сложность алгоритма равна $O(mw)$.

Два приведенных выше алгоритма применяются последовательно. Одно из улучшений работы алгоритмов состоит в том, чтобы строить множества вершин одновременно с множеством сепараторов. В связи с этим возникает понятие основной вершины сепаратора s — вершины инцидентной ребру нагрузки s в максимальном графе смежности. Множество таких вершин обозначим как MainVertices:

$$\mathrm{MainVertices}(s) = \{w | \exists w' : \mathrm{Sep}(w, w') = s\}.$$

Множество основных вершин обладает особым свойством, формализованным в утверждении 9, которое позволяет использовать это множество в алгоритмах синтеза ModelKit:

Утверждение 9. Множество основных вершин стереосепараторов и бисепараторов совпадает с множеством вершин сужения на соответствующий сепаратор.

Доказательство. Рассмотрим сепаратор s, для которого множество его основных вершин не совпадает с множеством вершин, входящих в сужение на s. Рассмотрим произвольную вершину v, такую, что $s \subset W(v)$ и v не является основной вершиной сепаратора s. Поскольку для любой другой вершины u $Sep(v,u) \neq s$, то сильное сужение на s содержит ребро (v,u), то есть любая вершина в сильном сужении связна с v, поэтому сильное сужение состоит из одной компоненты связности и не может быть ни бисепаратором, ни стереосепаратором.

Алгоритм построения множества непустых сепараторов и множеств их основных вершин по множеству нагрузок, соответствующих первичной структуре, приведен на листинге 7. Основная идея алгоритма состоит в том, чтобы сократить число обращений к каждой вершине.

```
SeparatorSetAndMainVertices
Require: Workloads
Ensure: Workloads ∈ PrimaryStructure
 1: Separators ← ∅
 2: MainVertices ← ∅
 3: for all w, w′ ∈ Workloads, w ≠ w′ do
 4:    s ← w ∩ w′
 5:    SeparatorTable[w, w′] ← s
 6:    if s ≠ ∅ do
 7:       if s ∉ Separators do
 8:          Separators ← Separators ∪ {s}
 9:          MainVertices[s] ← ∅
10:       end if
11:       MainVertices[s] ← MainVertices[s] ∪ {w}
12:       MainVertices[s] ← MainVertices[s] ∪ {w′}
13:    end if
14: end for
15: return ⟨Separators, MainVertices, SeparatorTable⟩
```

Листинг 7. Алгоритм построения множества непустых сепараторов и множеств их основных вершин.

Получив на вход множество нагрузок Workloads, алгоритм перебирает все пары нагрузок из Workloads (3–14). Пересечение нагру-

зок — s, добавляется в таблицу SeparatorTable (5). Далее s добавляется к множеству непустых сепараторов (7–10), а нагрузки добавляются к множеству основных вершин s (11–12).

Утверждение 10. Сложность работы алгоритма **SeparatorSetAndMainVertices** принадлежит классу $O(mw^2)$.

Доказательство. Для каждой пары нагрузок на шаге 4 берется их пересечение, что имеет сложность $O(mw^2)$. Проверка, является ли множество пустым (7), добавление элемента к множеству (8, 11–12), имеют сложность $O(1)$, поэтому сложность алгоритма равна $O(mw^2)$.

Другим важным вспомогательным алгоритмом является *алгоритм построения сильного сужения* по множеству вершины сужения, представленный на листинге 8.

```
StrongNarrowing
Require: s, NarrowedVertices, SeparatorTable
Ensure: ∀w ∈ NarrowedVertices  s ⊂ w;  s ∈ Separators
 1: Edges ← ∅
 2: for all w, w′ ∈ NarrowedVertices, w ≠ w′ do
 3:   if s ⊂ SeparatorTable[w, w′] do
 4:     Edges ← Edges ∪ {{w, w′}}
 5:   end if
 6: end for
 7: return ⟨NarrowedVertices, Edges⟩
```

Листинг 8. Алгоритм построения сильного сужения сепаратора.

Получив на входе NarrowedVertices — множество вершин, входящих в сужение некоторого сепаратора s — а также таблицу, в которую записаны пересечения нагрузок вершин, алгоритм перебирает все пары вершин из NarrowedVertices и проверяет, если пересечение этих нагрузок строго содержит сепаратор s (3), и в этом случае добавляет ребро между этим вершинами к множеству ребер строгого сужения (4).

Утверждение 11. Сложность работы алгоритма **StrongNarrowing** принадлежит классу $O({v_s}^2)$, где v_s — число вершин, входящих в сужение на сепаратор s.

Доказательство. В алгоритме перебираются все пары вершин из NarrowedVertices, и для каждой пары осуществляется проверка на шаге 3 за $O(1)$ и добавление ребра к множеству на шаге 4 за $O(1)$. Следовательно, сложность алгоритма равна $O({v_s}^2)$.

Алгоритм построения множества основных вершин и графа, обратного сильному сужению, позволяет по исходному набору нагрузок, соответствующих первичной структуре, одновременно построить множество непустых сепараторов, множества основных вершин каждого непустого сепаратора и множество графов, обратных сильному сужению каждого непустого сепаратора. Этот алгоритм приведен на листинге 9.

```
SsMVsSNNEs
Require: Workloads
Ensure: Workloads ∈ PrimaryStructure
 1: Separators ← ∅
 2: MainVertices ← ∅
 3: for all w, w′ ∈ Workloads, w ≠ w′ do
 4:   s ← w ∩ w′
 5:   if s ≠ ∅ do
 6:     if s ∉ Separators do
 7:       Separators ← Separators ∪ {s}
 8:       MainVertices[s] ← ∅
 9:       StrongNarrowingNoEdges[s] ← ∅
10:     end if
11:     MainVertices[s] ← MainVertices[s] ∪ {w}
12:     MainVertices[s] ← MainVertices[s] ∪ {w′}
13:     StrongNarrowingNoEdges[s] ←
                          StrongNarrowingNoEdges[s] ∪ {{w, w′}}
14:   end if
15: end for
16: return ⟨Separators, MainVertices⟩
```

Листинг 9. Алгоритм построения множества основных вершин и графа, обратного сильному сужению на сепаратор.

По заданному множеству нагрузок Workloads алгоритм перебирает все пары нагрузок (3–15). Пересечение нагрузок записывается в s (4) и если оно непусто, то s добавляется к множеству непустых сепараторов (6–10). Рассматриваемая пара вершин добавляется к множеству основных вершин сепаратора s, а ребро между ними добавляется к множеству ребер графа, обратному сильному сужению на этот сепаратор.

Утверждение 12. Сложность работы алгоритма **SsMVsSNNEs** принадлежит классу $O(mw^2)$.

Доказательство. Алгоритм похож на алгоритм **SeparatorSetAndMainVertices**, отличаясь от него наличием шага 13, который состоит в добавлении элемента к множеству и делается не более w^2 (точнее, $\frac{w(w-1)}{2}$) раз, поэтому сложность алгоритма также равна $O(mw^2)$.

5.3. Алгоритмы синтеза третичной структуры. *Родительский граф* над Separators — это направленный граф, вершинами которого выступают непустые сепараторы, а ребро проведено от сепаратора s к сепаратору s' тогда и только тогда, когда $s' \subset s$ и $\nexists s'' \in \text{Separator}: s \subset s'' \subset s'$. Другими словами, родительский граф задается диаграммой Хассе для множества Separators с отношением частичного порядка, индуцированным отношением включения. Начало любого ребра родительского графа будем называть *родителем* конца этого ребра, а конец этого ребра — *сыном* начала этого ребра.

Указанный частичный порядок можно расширить до полного порядка. Так, будем говорить, что сепаратор s *больше* сепаратора s', если выполняется одно из следующих условий:

1) $|s| > |s'|$;
2) $|s| = |s'|$ и s больше s' лексикографически (т.е. в соответствии с порядком, заданным на алфавите).

Из первого пункта следует, что $s \subset s' \Rightarrow s > s'$, поэтому введенное отношение порядка содержит частичный порядок.

В дальнейшем мы будем пользоваться алгоритмом сортировки множества Separators **SortSeparators**, который по заданному множеству сепараторов возвращает то же множество, упорядоченное по возрастанию. Сложность алгоритма, как уже было сказано в первом разделе, равна $O(ms \ln s)$, где s — число непустых сепараторов, а m — максимальное число атомов во фрагменте знаний, поскольку на сравнение двух сепараторов тратится не более m операций (т.к. размер каждого сепаратора строго меньше m), а для сортировки требуется порядка $s \ln s$ сравнений. В последующих алгоритмах множества непустых сепараторов, упорядоченные этим алгоритмом, можно перебирать последовательно.

Использование родительского графа в алгоритмах синтеза множества минимальных графов смежности позволяет эффективно строить множество вершин, входящих в сужение на сепаратор: множество вершин сужения на сепаратор U является подмножеством вершин сужения на сепаратор любого из его родителей.

Приведем алгоритмы синтеза родительского графа над множеством непустых сепараторов — *третичной структуры* АБС, рассмотренные в работе [19]. В данной работе мы приведем оценки сложности, в которых уменьшим число определяющих параметров. Кроме того, далее мы предложим еще один алгоритм синтеза третичной структуры.

Для построения ModelKit, рассматриваемой в следующем разделе, может использоваться как один из приводимых ниже алгоритмов, так и иной, более эффективный — все они являются реализацией абстрактного **SONS**.

Алгоритм построения родительского графа при помощи потомков (листинг 10) по заданному множеству непустых сепараторов строит родительский граф последовательным построением множества потомков каждого сепаратора, с последующим удалением из этого множества потомков, которые являются потомками потомков (т.е. по определению не являются сыновьями).

SONSBYDESCENDANTS
Require: Sepatators
1: Descendands ← ∅
2: **for all** $s, s' \in$ Separators, $s' \not\subset s$ **do**
3: **if** $s \subset s'$ **do**
4: Descendands[s] ← Descendands[s] ∪ {s'}
5: **end if**
6: **end for**
7: **for all** $s \in$ Separators **do**
8: PraDescendants ← ∅
9: **for all** $s' \in$ Descendands[s] **do**
10: PraDescendants ← PraDescendants ∪ Descendants[s']
11: **end for**
12: Sons[s] ← Descendants[s]\PraDescendants
13: **end for**
14: **return** Sons

Листинг 10. Алгоритм построения родительского графа при помощи потомков [19].

Утверждение 13. Сложность работы алгоритма **SONSBYDESCENDANTS** принадлежит классу $O(ms^2 + s^3)$.

Доказательство. Шаг 4 требует сравнения всех пар сепараторов. Одно сравнение имеет сложность $O(m)$, поэтому цикл 2–6 имеет сложность $O(ms^2)$. В цикле 7–13 перебираются все непустые сепара-

торы, которых s, в цикле 9–11 перебираются все потомки сепаратора, которых меньше s, и для каждого потомка на шаге 10 происходит объединение двух множеств, каждый из которых имеет размер не больше s. Таким образом, сложность цикла 9–11 равна $O(s^2)$, сложность шага 12 равна $O(s)$, поэтому сложность цикла 7–13 равна $O(s^3)$, а сложность работы всего алгоритма — $O(ms^2 + s^3)$.

```
SonsBottomUp
Require: Sepatators
 1: SortedSeparators ← SortSeparators(Sepatators)
 2: for i = |SortedSeparators| − 1 downto 0 do
 3:    s ← SortedSeparators[i]
 4:    Sons[s] ← ∅
 5:    for j = i + 1 to |SortedSeparators| do
 6:       s′ ← SortedSeparators[j]
 7:       SepIncluding[s, s′] ← (s ⊂ s′)
 8:       if SepIncluding[s, s′] = TRUE do
 9:          is_grson ← FALSE
10:          for all s″ ∈ Sons[s] do
11:             if SepIncluding[s″, s′] = TRUE do
12:                is_grson ← TRUE
13:                break
14:             end if
15:          end for
16:          if is_grson = FALSE do
17:             Sons[s] ← Sons[s] ∪ {s′}
18:          end if
19:       end if
20:    end for
21: end for
22: return Sons
```

Листинг 11. Алгоритм построения родительского графа снизу–вверх [19].

Утверждение 14. Сложность работы алгоритма **SonsBottomUp** принадлежит классу $O(ms^2 + e_S s)$, где e_S — число ребер в третичной структуре.

Доказательство. Сортировка на шаге 1 имеет сложность $O(ms \ln s)$. Сравнение на шаге 7 будет осуществляться для всех пар сепараторов и имеет сложность $O(ms^2)$. Цикл 10–15 осуществляется

для каждой пары сепараторов (u, u'), и в нем перебираются все сыновья u, для каждого сына сложность выполнения операций 11–14 равна $O(1)$. Сумма числа сыновей каждого сепаратора равна числу ребер в родительском графе, таким образом, сложность цикла 10–15 равна $O(e_S s)$. Поэтому сложность работы алгоритма равна $O(ms^2 + e_S s)$.

Введем новый алгоритм синтеза третичной структуры. Идея алгоритма следующая: будем перебирать сепараторы по возрастанию, добавляя их к родительскому дереву следующим образом: рассматриваемый сепаратор s пытается стать сыном каждого сепаратора, начиная с корня, инициализированного пустой нагрузкой. Для этого проверяется, содержит ли рассматриваемый сепаратор c сепаратор s, который алгоритм пытается пристроить к нему в сыновья. Если содержит, то проверяется, нет ли у c детей, которые содержат s. Если таких детей не нашлось, то s становится сыном c, а если такие дети нашлись, то s пытается стать сыном каждого из сыновей c, его содержащих.

Сначала опишем вспомогательный алгоритм **ADOPT**, который для двух сепараторов пытается сделать первый — s — сыном второго — c (листинг 12).

```
ADOPT
Require: s, c
 1: if LastAdoptTry[c] ≠ s do
 2:    b ← TRUE
 3:    LastAdoptTry[c] ← s
 4:    for all c' ∈ Sons[c] do
 5:       if LastAdoptTry[c'] ≠ s
 6:          if c' ⊂ s do
 7:             b ← FALSE
 8:             ADOPT(s,c')
 9:             Contains[c'] ← s
10:          end if
11:       else
12:          b ← b ∨ (Contains[c'] = s)
13:       end if
14:    end for
15:    if b = TRUE
16:       Sons[c] ← Sons[c] ∪ {s}
17:       Sons[s] ← ∅
18:       LastAdoptTry[s] ← ∅
```

```
19:   end if
20: end if
21: return
```

Листинг 12. Алгоритм построения множества основных вершин и графа, обратного сильному сужению на сепаратор.

По двум сепараторам, s и c, алгоритм рекурсивно пристраивает s в сыновья к c. Первым шагом алгоритм проверяет, не совпадает ли s с одним из элементов LastAdoptTry[c], что говорило бы о том, что мы уже пытались пристроить s в сыновья c. В этом случае ничего не происходит, поскольку все необходимые действия были проделаны до этого. Если же условие не выполняется, то полю LastAdoptTry[c] присваивается значение s (3), переменная b, характеризующая отсутствие сыновей c, которые содержат s, инициализируется истиной (2). Алгоритм перебирает сыновей c (4–14), также проверяя, не пытались ли мы пристроить s в сыновья к соответствующему сыну. Если не пытались, то есть если рассматриваемый сын содержит s (6–10), то b перестает быть истинной (7), и s пристраивается в сыновья уже к рассматриваемому сыну (8), а ячейке массива Contains с индексом c' присваивается значение s (9), что говорит о том, что c содержит s. Если же LastAdoptTry[c'] оказалось равно s (другими словами, мы уже пытались пристроить s в сыновья к c'), то значение b изменяется с учетом значения Contains. Если после перебора всех уже построенных сыновей (4–14) b все же осталось истиной (15), то s добавляется к множеству сыновей c, множество сыновей s инициализируется пустым множеством (17), как и соответствующий s элемент множества LastAdoptTry (18).

Алгоритм построения родительского графа усыновлением по заданному множеству сепараторов строит родительский граф над множеством сепараторов, дополненному пустым сепаратором (листинг 13).

```
SONSADOPTATION
Require: Sepatators
 1: Sons[∅] ← ∅
 2: SortedSeparators ← SORTSEPARATORS(Sepatators)
 3: for i ← 0 to |SortedSeparators| − 1 do
 4:    s ← SortedSeparators[i]
 5:    ADOPT(s, ∅)
 6: end for
```

```
7: return Sons
```

Листинг 13. Алгоритм построения родительского графа усыновлением.

Получив на входе множество сепараторов Separators, алгоритм инициализирует родительское дерево пустым сепаратором, множество сыновей которого изначально пусто (1). Затем, перебирая сепараторы по возрастанию, пытается пристроить каждый сепаратор в сыновья к пустому сепаратору (3–6).

Утверждение 15. Алгоритм построения родительского графа усыновлением строит родительский граф над множеством сепараторов, дополненным пустым сепаратором.

Доказательство. Прежде всего заметим, что, благодаря тому, что сепараторы перебираются по возрастанию, сепаратор будет добавляться к дереву уже после того, как были добавлены всего его родители (и, более того, все его предки).

Покажем, используя индукционное рассуждение, что для любого значения i между 0 и $|\text{SortedSeparators}| - 1$ после выполнения **ADOPT**(SortedSeparators$[i], \emptyset$) будет построено родительское дерево над множеством сепараторов $\{\text{SortedSeparators}[j] | 0 \le j \le i\}$, пополненным пустым сепаратором. База индукции, $i = 0$. После выполнения первого шага единственный сепаратор будет добавлен как сын пустого сепаратора (потому что пустой сепаратор содержит любой другой сепаратор, а множество его сыновей было инициализировано пустым множеством), что описывает родительский граф, построенный над соответствующим множеством.

Пускай теперь для всех сепараторов, меньших s, в ходе работы алгоритма был построен родительский граф над соответствующим множеством. Покажем, что и после выполнения **ADOPT**$(s, \emptyset)$ пополненный граф будет родительским.

Заметим, что если **ADOPT**(s, c) было вызвано, то $c \subset s$. Покажем, что **ADOPT**(s, c) будет вызвано для любого c, такого, что $c \subset s$. Предположим обратное, тогда выберем такое c, что $c \subset s$, **ADOPT**(s, c) не было вызвано и расстояние до c от вершины родительского графа минимально. Тогда существует родитель c, которого мы обозначим как c', для которого происходил вызов **ADOPT**(s, c') (потому что $c' \subset c \subset s$ и расстояние от вершины до c' на единицу меньше, чем до c). В первый вызов **ADOPT**(s, c') значение LastAdoptTry$[c']$ не было равно s, поэтому выполнялись все шаги цикла 4–14, а значит, перебирались все сыновья c', в том числе c, которое по предположению содержится в s, и, следовательно, для c шаг 8 достигался, что приводит нас к противоречию.

Таким образом, **ADOPT**(s, c) было запущено для всех c, таких, что $c \subset s$, и только для них.

Для каждого s' — родителя s — множество $\mathrm{Sons}[s']$ по предположению индукции не содержало элементов s'', таких, что $s'' \subset s$, поэтому при вызове **ADOPT**(s, s') сепаратор s был бы добавлен к множеству $\mathrm{Sons}[s']$. С другой стороны, у сепараторов, не являющихся родителями s, соответствующее множество Sons содержало такие элементы s'', поэтому s не было бы добавлено в эти множества. Таким образом, s будет добавлено к множествам $\mathrm{Sons}[s']$ для каждого своего родителя s'.

При этом алгоритм закончит работу, поскольку он обходит вершины родительского дерева не более одного раза, так как после посещения любой вершины c значение $\mathrm{LastAdoptTry}[c]$ меняется на s и не изменится до тех пор, пока не будет запущен **ADOPT** с другим сепаратором в качестве первого аргумента.

Утверждение 16. Сложность работы алгоритма **SONSADOPTATION** принадлежит классу $O(ms^2 + e_s s)$.

Доказательство. Рассмотрим сложность добавления произвольного сепаратора к уже построенному родительскому графу. Как было показано, он добавляется ко всем своим предкам и только к ним (шаг 8), поэтому сравнение на шаге 6 осуществляется число раз, равное числу предков сепаратора. Это число меньше s, а сложность сравнения равна $O(m)$. Кроме того, этот сепаратор сравнивается со всеми своими предками, которых не больше s, и каждое сравнение требует не более $O(m)$ операций. Таким образом, все сравнения имеют сложность $O(ms^2)$. Кроме того, при запуске алгоритма для каждого стереосепаратора (5) на шаге 4 алгоритма **ADOPT** будут перебираться все построенные к тому моменту сыновья стереосепараторов, содержащих рассматриваемый стереосепаратор. Таких сыновей не больше, чем ребер в третичной структуре (то есть сыновей всех стереосепараторов), поэтому оценка сложности этого перебора лежит в классе $O(e_s s)$. Следовательно, оценка сложности работы алгоритма равна $O(ms^2 + e_s s)$.

Вариантом модификации алгоритма (который мы не будем рассматривать в этой работе) является построение дерева снизу вверх (тогда нам нужно кроме сыновей хранить и родителей. С одной стороны, это потребует в среднем больше времени на вычисление того, содержится ли один сепаратор в другом (которое мы оцениваем по минимальной длине сепараторов), с другой стороны, предполагается, что такой подход позволит минимизировать число тех сепараторов, к ко-

торым в сыновья в процессе работы алгоритма пристраивается каждый стереосепаратор.

5.4. Алгоритмы синтеза четвертичной структуры. *Полусиблинговым графом* для данного сепаратора s называется граф, вершинами которого выступают сыновья s, а ребра проведены между теми сыновьями, сужения на которые пересекаются. Полусиблинговые графы также используются для алгоритмов синтеза множества минимальных графов смежности за счет эффективного построения компонент связности сильного сужения (т.е. владений). *Алгоритм построения множества ребер полусиблингового графа* приведен на листинге 14.

```
HALFSIBLINGGRAPHEDGES
Require: s, Sons, Vertices
Ensure: s ∈ Separators
 1: Fellows ← ∅
 2: for all c, c′ ∈ Sons[s] do
 3:    if Vertices[c] ∩ Vertices[c′] ≠ ∅ do
 4:       Fellows ← Fellows ∪ {{c, c′}}
 5:    end if
 6: end for
 7: return Fellows
```

Листинг 14. Алгоритм построения множества ребер полусиблингового графа

Получив на входе сепаратор, а также множества сыновей и вершин (входящих в сужения) каждого сепаратора, алгоритм перебирает все пары сепараторов (2–5), и если вершины соответствующих сепараторов пересекаются, то ребро двумя сепараторами добавляется к множеству ребер (4).

Утверждение 17. Сложность работы алгоритма **HALFSIBLINGGRAPHEDGES** относится к классу $O(c_u {s_u}^2)$, где s_u — число сыновей сепаратора u, а c_u — максимальное число вершин во владениях сепаратора u.

Доказательство. В цикле 2–6, перебирающем каждую пару сыновей исходного сепаратора, множество вершин этих сепараторов пересекаются за $O(c_u)$, поскольку в каждом из этих множеств не более c_u вершин. Следовательно, сложность алгоритма равна $O(c_u {s_u}^2)$.

Согласно теореме о классификации владений [30], компоненты связности сильного сужения на произвольный сепаратор u представляют собой либо *доменные вершины* — т.е. вершины, которые входят в сужение на сепаратор u и не входят ни в какое сужение на любой из

его сыновей, либо *владения* и *братства* — множества вершин, входящие в сужение на каждый сепаратор из компоненты связности, одноэлементной и многоэлементной соответственно.

Это позволяет предложить альтернативный способ построения владений, идея которого состоит в поиске компонент связности полусиблингового графа. Эта идея реализуется в *алгоритме построения владений по четвертичной структуре* (листинг 15).

```
HoldingsByFourthStructure
Require: s, Sons, Vertices
Ensure: s ∈ Separators
 1: Brotherhoods ←
        Components(⟨Sons[s], HalfsiblingGraphEdges(s, Sons, Vertices)⟩)
 2: Holdings ← ∅
 3: Demesne ← Vertices[s]
 3: for all B ∈ Brotherhoods do
 4:    HoldingVertices ← ∅
 5:    for all c ∈ B do
 6:       HoldingVertices ← HoldingVertices ∪ Vertices[c]
 7:    end for
 8:    Demesne ← Demesne\HoldingVertices
 9:    Holdings ← UniteDisjointSets(Holdings, {HoldingVertices})
10: end for
11: for all d ∈ Demesne do
12:    Holdings ← Holdings ∪ {d}
13: end for
14: return Holdings
```

Листинг 15. Алгоритм построения владений по четвертичной структуре

Получив на вход сепаратор s, множества сыновей и вершин (входящих в сильные сужения) сепараторов, алгоритм на шаге 1 строит множество братств Brotherhoods за счет поиска компонент связности в полусиблинговом графе. На шаге 3 множество доменных вершин Demesne инициализируется вершинами сепаратора s. Затем в цикле 3–10 для каждого братства из множества братств (которое является графом) строятся соответствующие им владения, для чего перебираются все вершины братства (5–7), которые по определению являются сыновьями s, и для каждой такой вершины к множеству вершин владения добавляются вершины сужения на c (6). Эти же вершины на шаге 8 вычитаются из множества доменных вершин. Затем построенное мно-

жество вершин, входящих во владение, на шаге 9 добавляет к множеству владений (поскольку множества вершин каждого владения по определению дизъюнктны, используется **UNITEDISJOINTSETS**).

После перебора всех братств, в цикле 11–13 перебираются все доменные вершины, которые добавляются как одноэлементные компоненты связности к множеству владений (12).

Утверждение 18. Сложность работы алгоритма **HOLDINGSBYFOURTHSTRUCTURE** принадлежит классу $O(c_u {s_u}^2 + v_u)$, где s_u — число сыновей сепаратора u, а c_u — максимальное число вершин во владениях сепаратора u, а v_u — число вершин в сужении на u.

Доказательство. Сложность выполнения шага 1 равна $O(c_u {s_u}^2)$, поскольку именно такую сложность согласно утверждению 17 имеет работа алгоритма **HALFSIBLINGGRAPHEDGES**, а построение компонент связности полусиблингового графа имеет сложность $O({s_u}^2)$. Далее в цикле 5–7 для каждой вершины каждого братства (то есть для каждого сына c рассматриваемого стереосепаратора) происходит объединение двух множеств, одно из которых есть множество вершин из сужения на c. Следовательно, сложность каждого такого шага можно оценить как $O(v_c)$. Поскольку $\sum_{c \in \mathrm{Sons}(u)} v_c \le v_u$, то сложность выполнения всех таких объединений равна $O(v_u)$. Поэтому сложность работы алгоритма равна $O(c_u {s_u}^2 + v_u)$.

6. Система алгоритмов синтеза ModelKit. Далее в оценках сложности приводимых алгоритмов будем обозначать сложность работы **SONS** как S^*. В предыдущем разделе было показано, что она лежит в классе $O(ms^2 + e_S s)$, где m — максимальное число атомов во фрагменте знаний, s — число непустых сепараторов, а e_S — число ребер в третичной структуре.

В этом разделе мы приведем шесть различных алгоритмов построения ModelKit, основанных на работах [15–18]. Для алгоритмов будет указано, где описан алгоритм, который лег в их основу.

Идея первого алгоритма — **MK1**, основанного на алгоритме построения множества МГС при помощи самоуправляемых клик [15] — состоит в следующем: сначала строится множество непустых сепараторов, затем для каждого сепаратора перебором всех нагрузок строится множество вершин, входящих в соответствующее сужение. Для построенного множества вершин определяется, является ли сепаратор бисепаратором (число вершин равно двум) или является ли он стереосепаратором (не бисепаратор и владений больше одного). В первом случае единственное ребро добавляется к множеству обязательных

ребер, во втором случае владения строятся как компоненты связности сильного сужения. Алгоритм приведен на листинге 16.

```
MK1
Require: Workloads
Ensure: Workloads ∈ PrimaryStructure
 1: Stereoseparators ← ∅
 2: NecessaryEdge ← ∅
 3: Stereoholdings ← ∅
 4: ⟨Separators, SeparatorTable⟩ ← SeparatorSet(Workloads)
 5: for all s ∈ Separators do
 6:    Vertices ← NarrowedVertices(s, Workloads)
 7:    if |Vertices[s]| = 2 do
 8:       NecessaryEdge ←
                          NecessaryEdge ∪ (Vertices[s][0], Vertices[s][1])
 9:    else
10:       Holdings ← Components(StrongNarrowing
                                  (Vertices[s], s, SeparatorTable))
11:       if |Holdings| > 1 do
12:          Stereoseparators ← Stereoseparators ∪ {s}
13:          Stereoholding[s] ← Holdings
14:       end if
15:    end if
16: end for
17: return
          ⟨Workloads, Stereoseparators, Stereoholding, NecessaryEdge⟩
```

Листинг 16. Алгоритм построения ModelKit MK1, основанный на алгоритме построения множества МГС при помощи самоуправляемых клик [15].

Утверждение 19. Сложность работы алгоритма **MK1** принадлежит классу

$$O\left(mw^2 + msw + \sum\nolimits_{u \in \mathrm{NotBiSep}} ({v_u}^2)\right),$$

где s_u — число сыновей сепаратора u, а c_u — максимальное число вершин во владениях сепаратора u, а v_u — число вершин в сужении на u.

Доказательство. По утверждению 7, сложность шага 4 равна $O(mw^2)$, а сложность построения сужения на шаге 6, согласно утверждению 8, равна $O(msw)$. Для сепаратора u, не являющегося бисепа-

ратором, сложность построения сильного сужения на шаге 10, согласно утверждению 11, равна $O({v_u}^2)$, такова же сложность поиска компонент связности, поэтому сложность выполнения всех шагов 10 равна

$$\sum_{u \in \text{NotBiSep}} ({v_u}^2).$$

Суммируя приведенные слагаемые, получим итоговый результат.

Идея второго алгоритма — **MK2**, развивающего алгоритм построения множества МГС на основе клик владений [16], состоит в следующем: сначала строится множество непустых сепараторов, затем над построенным множеством строится родительский граф. После этого последовательно перебираются сепараторы и для каждого из них строится множество вершин, входящих в соответствующее сужение. Как и в предыдущем случае, бисепараторы отсеиваются по числу вершин сужения (жилы добавляются к множеству обязательных ребер). Отличие от предыдущего алгоритма состоит в том, что владения строятся на основе алгоритма **HOLDINGSBYFOURTHSTRUCUTRE**, то есть за счет поиска компонент связности в четвертичной структуре. Построенные владения обрабатываются так же: неодноэлементное множество владений добавляется к множеству владений стереосепараторов, а соответствующий сепаратор добавляется к множеству стереосепараторов. Алгоритм приведен на листинге 17.

```
MK2
Require: Workloads
Ensure: Workloads ∈ PrimaryStructure
 1: Stereoseparators ← ∅
 2: NecessaryEdge ← ∅
 3: Stereoholdings ← ∅
 4: ⟨Separators, SeparatorTable⟩ ← SEPARATORSET(Workloads)
 5: Sons ← SONS(Separators)
 6: SortedSeparators ← SORTSEPARATORS(Sepatators)
 7: for i ← 0 to |SortedSeparators| − 1 do
 8:    s ← SortedSeparators[i]
 9:    Vertices[s] ← NARROWEDVERTICES(s, Workloads)
10:    if |Vertices[s]| = 2 do
11:       NecessaryEdge ←
                    NecessaryEdge ∪ {(Vertices[s][0], Vertices[s][1])}
12:    else
13:       Holdings ← HOLDINGSBYFOURTHSTRUCUTRE(s, Sons, Vertices)
```

```
14:       if |Holdings| > 1 do
15:          Stereoseparators ← Stereoseparators ∪ {s}
16:          Stereoholdings[s] ← Holdings
17:       end if
18:    end if
19: end for
20: return
                ⟨Workloads, Stereoseparators, Stereoholding, NecessaryEdge⟩
```

Листинг 17. Алгоритм построения ModelKit MK2, основанный на алгоритме построения множества МГС при помощи клик владений [16].

Утверждение 20. Сложность алгоритма лежит в классе

$$O\left(mw^2 + ms\ln s + S_3 + msw + \sum_{u\in \mathrm{NBSep}} (c_u {s_u}^2 + v_u)\right),$$

где m — максимальное число атомов во фрагменте знаний, w — число фрагментов знаний, s — число непустых сепараторов, c_u — максимальное число вершин во владении сепаратора u, s_u — число сыновей сепаратора u, v_u — число вершин, входящих в сужение на сепаратор u, а NBSep — множество сепараторов, не являющихся бисепараторами.

Доказательство. По утверждению 7, сложность шага 4 равна $O(mw^2)$, сложность сортировки на шаге 6 — $O(ms\ln s)$. Сложность построения сужения на шаге 9, согласно утверждению 8, равна $O(msw)$. Для каждого сепаратора (не являющегося бисепаратором) u сложность работы шага 13 равна, согласно по утверждению 16, $O(c_u {s_u}^2 + v_u)$, поэтому работа всех шагов 13 равна

$$\sum_{u\in \mathrm{NBSep}} (c_u {s_u}^2 + v_u).$$

Суммируя приведенные слагаемые, получим итоговый результат.

Следующий алгоритм основан на предыдущем алгоритме, основное отличие состоит в построении вершин сужений. Эти вершины строятся за счет сужения не всех нагрузок, а только нагрузок родителя (если родитель существует). Алгоритм приведен на листинге 18.

```
MK2+
Require: Workloads
Ensure: Workloads ∈ PrimaryStructure
 1: Stereoseparators ← ∅
 2: NecessaryEdge ← ∅
```

```
 3: Holdings ← ∅
 4: ⟨Separators, SeparatorTable⟩ ← SeparatorSet(Workloads)
 5: Sons ← Sons(Separators)
 6: SortedSeparators ← SortSeparators(Sepatators)
 7: for i ← |SortedSeparators| − 1 downto 0 do
 8:    s ← SortedSeparators[i]
 9:    if |Vertices[s]| = 0 do
10:       Vertices[s] ← NarrowedVertices(s, Workloads)
11:    end if
12:    if |Vertices[s]| = 2 do
13:       NecessaryEdge ←
                    NecessaryEdge ∪ {{(Vertices[s][0], Vertices[s][1])}}
14:    else
15:       for all c ∈ Sons[s]
16:          if |Vertices[c]| = 0 do
17:             Vertices[c] ← NarrowedVertices(c, Vertices[s])
18:          end if
19:       end for
20:       Holdings ← HoldingsByFourthStructure(s, Sons, Vertices)
21:       if |Holdings| > 1 do
22:          Stereoseparators ← Stereoseparators ∪ {s}
23:          Holdings[s] ← Holdings
24:       end if
25:    end if
26: end for
27: return
          ⟨Workloads, Stereoseparators, Stereoholding, NecessaryEdge⟩
```

Листинг 18. Алгоритм построения ModelKit MK2.

Утверждение 21. Сложность алгоритма принадлежит к классу

$$O\left(mw^2 + ms\ln s + S_3 + ms_\emptyset w + \sum_{u\in \mathrm{NBSep}} (ms_u v_u + c_u {s_u}^2 + v_u)\right),$$

где $s_\emptyset$ — число сепараторов, у которых нет родителей в третичной структуре.

Доказательство. По утверждению 7, сложность шага 4 равна $O(mw^2)$, сложность сортировки на шаге 6 — $O(ms\ln s)$. Сложность построения третичной структуры на шаге 5 равна S_3.

Для оценки сложности работы цикла 7–26 воспользуемся следующими рассуждениями. Алгоритм **NarrowedVertices** будет запущен для каждого сепаратора одним из его родителей либо самим сепаратором. Введем в рассмотрение пустой сепаратор, который будет единственной вершиной родительского графа и который будет родителем тех сепараторов, которые в третичной структуре не имеют родителей. Сложность работы построения сужений, таким образом, оказывается не больше, чем сложность работы построения для каждого сепаратора, не являющегося бисепаратором, множества вершин сужения на каждого его сына.

Таким образом, сложность выполнения всех шагов 10 и 17 равна

$$\sum_{u \in \mathrm{NBSep} \cup \{\emptyset\}} (ms_u v_u),$$

поскольку для каждого сына c каждого сепаратора u из множества $\mathrm{NBSep} \cup \{\emptyset\}$ строится сужение $G_{\max} \downarrow u$ на c, а сложность каждого такого шага согласно утверждению 8 равна $O(mv_u)$.

Сложность построения владений каждого непустого сепаратора u, не являющего бисепаратором, на шаге 20 согласно утверждению 16 равна $O(c_u {s_u}^2 + v_u)$, поэтому сложность выполнения всех таких шагов равна

$$\sum_{u \in NBSep} (c_u {s_u}^2 + v_u).$$

Сложив приведенные слагаемые, предварительно вынеся пустой сепаратор из суммирования, получим указанную в утверждении оценку. Отметим, что s_u может быть равно 0 для всех рассматриваемых сепараторов, поэтому утверждение о том, что v_u принадлежит классу $O(ms_u v_u)$, в общем случае неверно.

Идея четвертого алгоритма — **MK3**, базирующегося на алгоритме построения множества МГС на основе самоуправляемых кликсобственников [17], состоит в следующем: вместо последовательного построения множеств непустых сепараторов и множеств вершин сужений, одновременно строится множество непустых сепараторов и множества их основных вершин. Затем для каждого построенного сепаратора проверяется, является ли он бисепаратором (для этого у него должно быть всего две основные вершины и он должен быть бездетным), жила таких сепараторов соответствующим образом обрабатывается. Если у бисепаратора более двух основных вершин, строятся его владения за счет поиска компонент в сильном сужении, и если их бо-

лее одного, то такой сепаратор и его владения обрабатываются как стереосепараторы. Алгоритм приведен на листинге 19.

```
MK3
Require: Workloads
Ensure: Workloads ∈ PrimaryStructure
 1: Stereoseparators ← ∅
 2: NecessaryEdge ← ∅
 3: Holdings ← ∅
 4: ⟨Separators, MainVertices, SeparatorTable⟩ ←
                          SeparatorSetAndMainVertices(Workloads)
 5: for all s ∈ Separators do
 6:   if |MainVertices[s]| = 2 do
 7:     if isChildless(s) = TRUE do
 8:       NecessaryEdge ← NecessaryEdge ∪
                          {(MainVertices[s][0], MainVertices[s][1])}
 9:     end if
10:   else
11:     Holdings ← Components(StrongNarrowing
                          (MainVertices[s], s, SeparatorTable))
12:     if |Holdings| > 1 do
13:       Stereoseparators ← Stereoseparators ∪ {s}
14:       Stereoholding[s] ← Holdings
15:     end if
16:   end if
17: end for
18: return
          ⟨Workloads, Stereoseparators, Stereoholding, NecessaryEdge⟩
```

Листинг 190. Алгоритм построения множества МГС при помощи самоуправляемых клик-собственников [17].

Утверждение 22. Сложность алгоритма лежит в классе

$$O\left(mw^2 + ms(s - |\text{NB\&M1Sep}|) + \sum_{u \in \text{NB\&M1Sep}} ({v_u}^2)\right),$$

где m — максимальное число атомов во фрагменте знаний, w — число фрагментов знаний, s — число непустых сепараторов, v_u — число вершин, входящих в сужение на сепаратор u, а NB&$M1Sep$ — множе-

ство сепараторов, не являющихся бисепараторами и моносепараторами типа 1.

Доказательство. Сложность шага 4, согласно утверждению 10, равна $O(mw^2)$. Сложность шага 7 равна $O(ms)$, и он выполняется для всех бисепараторов и моносепараторов типа 1, поэтому в совокупности сложность этого шага равна $O(ms(s - |\text{NB\&M1Sep}|))$. Сложность шага 11 равна

$$O\left(\sum_{u \in \text{NB\&M1Sep}} (v_u{}^2)\right),$$

поскольку компоненты связности строятся для всех сепараторов из множества NB&M1Sep. Сложив приведенные значения, получим итоговую оценку.

Пятый алгоритм — **MK3+**, — отличается от предыдущего тем, что одновременно с множеством непустых сепараторов и множествами основных вершин этих сепараторов строит граф, обратный сильному сужению. Алгоритм представлен на листинге 20.

```
MK3+
Require: Workloads
Ensure: Workloads ∈ PrimaryStructure
 1: Stereoseparators ← ∅
 2: NecessaryEdge ← ∅
 3: Holdings ← ∅
 4: ⟨Separators, MainVertices, StrongNarrowingNoEdges⟩ ←
                                        SsMVsSNNEs(Workloads)
 5: for all s ∈ Separators do
 6:   if |MainVertices[s]| = 2 do
 7:     if IsChildless(s) = TRUE do
 8:       NecessaryEdge ← NecessaryEdge ∪
                        {(MainVertices[s][0], MainVertices[s][1])}
 9:     end if
10:   else
11:     Holdings ← ReversedComponents
                        (MainVertices[s], StrongNarrowingNoEdges))
12:     if |Holdings| > 1 do
13:       Stereoseparators ← Stereoseparators ∪ {s}
14:       Stereoholding[s] ← Holdings
```

```
15:     end if
16:   end if
17: end for
18: return
```

⟨Workloads, Stereoseparators, Stereoholding, NecessaryEdge⟩

Листинг 20. Алгоритм построения множества МГС при помощи самоуправляемых клик-собственников.

Утверждение 23. Сложность алгоритма лежит в классе

$$O\left(mw^2 + ms(s - |\text{NB\&M1Sep}|) + \sum_{u \in \text{NB\&M1Sep}} ({v_u}^2)\right),$$

где m — максимальное число атомов во фрагменте знаний, w — число фрагментов знаний, s — число непустых сепараторов, v_u — число вершин, входящих в сужение на сепаратор u, а NB&$M1Sep$ — множество сепараторов, не являющихся бисепараторами и моносепараторами типа 1.

Доказательство. Сложность работы шага 4, согласно утверждению 12, равна $O(mw^2)$. Сложность шага 11 равна $O(v_u)$ для каждого сепаратора u, для которого не выполняется условие 6, то есть являющего элементом множества NB&M1Sep. Таким образом, сложность работы алгоритма совпадает со сложностью работы алгоритма **MK3**.

Наконец, последний алгоритм построения ModelKit, рассматриваемый в этой работе — алгоритм **MK4**, приведенный на листинге 21, основан на алгоритме построения множества МГС при помощи клик–собственников владений [18], однако, в отличие от предыдущих алгоритмов, он содержит ряд изменений.

Идея MK4 состоит в следующем: сначала строится множество непустых сепараторов одновременно с множествами основных вершин таких сепараторов, затем строится третичная структура. Сепараторы перебираются по возрастанию, если у сепаратора u две основных вершины и нет детей, то он является бисепаратором и обрабатывается соответствующим образом. Если же у него есть дети, то он является моносепаратором типа 1, и тогда множество его вершин строится за счет объединения вершин его сыновей с его основными вершинами. В том же случае, если у сепаратора оказывается более двух основных вершин, то множество его владений синтезируется за счет построения полусиблингового графа и поиска компонент связности в нем. Если владений больше одного, то сепаратор обрабатывается как стереосепа-

ратор. В том же случае, если владение одно, то такой сепаратор является моносепаратором, но не моносепаратором первого типа, и множество его вершин строится тем же способом, что и для моносепаратора типа 1. Такая обработка моносепараторов обоих типов нужна для того, чтобы при построении владений родителей моносепараторов учитывались все вершины моносепаратора, а не только те, которые являются его основными (согласно утверждению 9, множество основных вершин совпадает с множеством вершин, входящих в сужение, только для бисепараторов и стереосепараторов, но не для моносепараторов).

```
MK4
Require: Workloads
Ensure: Workloads ∈ PrimaryStructure
 1: Stereoseparators ← ∅
 2: NecessaryEdge ← ∅
 3: Holdings ← ∅
 4: ⟨Separators, MainVertices, SeparatorTable⟩ ←
                          SeparatorSetAndMainVertices(Workloads)
 5: Sons ← Sons(Separators);
 6: SortedSeparators ← SortSeparators(Sepatators)
 7: for i ← 0 to |SortedSeparators| − 1 do
 8:    s ← SortedSeparators[i]
 9:    Vertices[s] ← MainVertices[s]
10:    if |MainVertices[s]| = 2 do
11:       if Sons[s] = ∅ do
12:          NecessaryEdge ← NecessaryEdge ∪
                          {(MainVertices[s][0], MainVertices[s][1])}
13:       else
14:          for all c ∈ Sons[s] do
15:             Vertices[s] ← Vertices[s] ∪ Vertices[c]
16:          end for
17:       end if
18:    else
19:       Holdings ← HoldingsbyfourthstrucutRe(s, Sons, Vertices)
20:       if |Holdings| > 1 do
21:          Stereoseparators ← Stereoseparators ∪ {s}
22:          Holdings[s] ← Holdings
23:       else
24:          for all c ∈ Sons[s] do
```

```
25:            Vertices[s] ← Vertices[s] ∪ Vertices[c]
26:          end for
27:        end if
28:      end if
29: end for
30: return
```

⟨Workloads, Stereoseparators, Stereoholding, NecessaryEdge⟩

Листинг 21. Алгоритм построения множества МГС при помощи клик–собственников [18].

Утверждение 24. Сложность алгоритма лежит в классе

$$O\left(mw^2 + ms\ln s + S_3 + \sum_{u\in \mathrm{MSep}} (c_u s_u) + \sum_{u\in \mathrm{NB\&M1Sep}} (c_u {s_u}^2 + v_u)\right),$$

где m — максимальное число атомов во фрагменте знаний, w — число фрагментов знаний, s — число непустых сепараторов, c_u — максимальное число вершин во владении сепаратора u, s_u — число сыновей сепаратора u, v_u — число вершин, входящих в сужение на сепаратор u, а NB&M1Sep — множество сепараторов, не являющихся бисепараторами и моносепараторами типа 1, а MSep — множество моносепараторов.

Доказательство. Согласно утверждению 10, сложность шага 4 равна $O(mw^2)$. Сложность шага 5 равна $O(ms\ln s)$, сложность построения третичной структуры обозначаем как S_3. Для всех сепараторов, не являющихся ни бисепараторами, ни моносепараторами типа 1 (то есть не прошедших проверку на шаге 10) выполняется шаг 19, сложность которого, согласно утверждению 18, равна $O(c_u {s_u}^2 + v_u)$. Для всех моносепараторов выполняется цикл 14–16 (или эквивалентный ему цикл 24–26), объединяющий вершины всех сыновей сепаратора. Сложность выполнения такого цикла для сепаратора равна $O(c_u s_u)$, потому что в каждом сыне вершин не больше, чем во владении. Сложив перечисленные слагаемые, получим итоговый ответ.

7. Заключение. В работе предложена двухэтапная система построения множества минимальных графов смежности — на первом шаге строится ModelKit, затем применяется один из алгоритмов сборки, который, перебирая жилы особого вида, строит некоторое подмножество МГС.

Все приведенные в работе подмножества отличались тем, что жилы для них строятся независимо. Это условие выполняется не для всех графов. Так, например, для *звездчатых графов смежности* [23] требу-

ется, чтобы центр жилы сепаратора совпадал с центром жилы одного из детей этого сепаратора, если они у него есть. Определенные таким образом жилы не могут быть построены независимо, напротив, их построение требует перебора всех стереосепараторов в определенном порядке (в рассматриваемом случае — по возрастанию) и хранения дополнительных сведениях о жилах, построенных на предыдущих шагах.

Соответственно, для некоторых видов минимальных графов смежности подмножество таких графов может быть синтезировано за счет независимого построения жил определенного вида; для некоторых других видов минимальных графов смежности подмножество таких графов может быть синтезировано за счет построения жил с определенными свойствами при некотором порядке перебора сепараторов. Отнесение того или иного вида МГС к одной из выделенных категорий позволит оценить сложность построения подмножества таких МГС.

Следует также отметить, что оценки сложности, приведенные в работе, опираются на очень большое число параметров. Для уменьшения числа параметров необходимо выявить зависимости между этими параметрами. Так, можно указать, в каких границах может изменяться значение любого из этих параметров в зависимости от любого другого, однако использование подобных зависимостей приведет к завышению оценки сложности. Вместо этого необходимо выявить зависимости между большим числом параметров, что позволит точнее выразить сложность алгоритмов. Эта проблема представляет самостоятельный интерес с точки зрения комбинаторики.

Одной из частных задач в рамках этой проблемы является выражение числа непустых сепараторов через мощность алфавита, число подалфавитов и размер максимального подалфавита.

Литература

1. *Опарин В.В., Тулупьев А.Л.* Синтез графа смежности с минимальным числом ребер: формализация алгоритма и анализ его корректности // Труды СПИИРАН. СПб.: Наука, 2009. Вып. 11. С. 142–157.
2. *Опарин В.В., Фильченков А.А., Тулупьев А.Л., Сироткин А.В.* Матроидное представление семейства графов смежности над набором фрагментов знаний // Научно-технический вестник Санкт- Петербургского государственного университета информационных технологий, механики и оптики. 2010. Вып. 4. С. 73–76
3. *Сироткин А.В.* Модели, алгоритмы и вычислительная сложность синтеза согласованных оценок истинности в алгебраических байесовских сетях // Информационно-измерительные и управляющие системы. 2009. №11. С. 32–37.
4. *Тулупьев А.Л.* Алгебраические байесовские сети: глобальный логико-вероятностный вывод в деревьях смежности: Учеб. пособие. СПб.: СПбГУ; ООО Издательство «Анатолия», 2007. 40 с. (Сер. Элементы мягких вычислений).

5. *Тулупьев А.Л.* Алгебраические байесовские сети. Логико-вероятностный подход к моделированию баз знаний с неопределенностью. СПб.: СПИИРАН, 2000. 292 с.
6. *Тулупьев А.Л.* Алгебраические байесовские сети: локальный логико-вероятностный вывод: Учеб. пособие. СПб.: СПбГУ; ООО Издательство «Анатолия», 2007. 80 с. (Сер. Элементы мягких вычислений).
7. *Тулупьев А.Л.* Ациклические алгебраические байесовские сети: логико-вероятностный вывод // Нечеткие системы и мягкие вычисления: Научный журнал Российской ассоциации нечетких систем и мягких вычислений. 2006. Том 1, № 1. С. 57–93.
8. *Тулупьев А.Л.* Байесовские сети: логико-вероятностный вывод в циклах. СПб.: Изд-во С.-Петербургского ун-та, 2008. 140 с. (Элементы мягких вычислений).
9. *Тулупьев А.Л.* Согласованность данных и оценка вероятности альтернатив в цикле стохастических предпочтений // Известия высших учебных заведений: Приборостроение. 2009. №7. С. 3–8.
10. *Тулупьев А.Л., Николенко С.И., Сироткин А.В.* Байесовские сети: логико-вероятностный подход. СПб.: Наука, 2006. 607 с.
11. *Тулупьев А.Л., Сироткин А.В.* Алгебраические байесовские сети: принцип декомпозиции и логико-вероятностный вывод в условиях неопределенности // Информационно-измерительные и управляющие системы. 2008. № 10. т. 6. С. 85–87.
12. *Тулупьев А.Л., Сироткин А.В., Николенко С.И.* Байесовские сети доверия: логико-вероятностный вывод в ациклических направленных графах. СПб.: Изд-во С.-Петерб. ун-та, 2009, 400 с.
13. *Тулупьев А.Л., Столяров Д.М., Ментюков М.В.* Представление локальной и глобальной структуры алгебраической байесовской сети в Java-приложениях // Труды СПИИРАН. 2007. Вып. 5. СПб.: Наука, 2007. С. 71–99.
14. *Тулупьев А.Л., Фильченков А.А., Вальтман Н.А.* Алгебраические байесовские сети: задачи автоматического обучения // Информационно-измерительные и управляющие системы. 2011. № 11, т. 9. С. 57-61.
15. *Фильченков А.А.* Алгоритм построения множества минимальных графов смежности при помощи самоуправляемых клик // Труды СПИИРАН. 2010. Вып. 1 (12). С. 119–133.
16. *Фильченков А.А.* Алгоритм построения множества минимальных графов смежности при помощи самоуправляемых клик-собственников // Труды СПИИРАН. 2010. Вып. 3 (14) С. 150–169.
17. *Фильченков А.А.* Алгоритм построения множества минимальных графов смежности при помощи клик владений // Труды СПИИРАН. 2010. Вып. 2 (13). С. 119–133.
18. *Фильченков А.А.* Алгоритм построения множества минимальных графов смежности при помощи клик-собственников владений // Труды СПИИРАН. 2010. Вып. 4 (15). С. 193–212.
19. *Фильченков А.А.* Алгоритмы построения третичной структуры алгебраической байесовской сети // Труды СПИИРАН. 2011. Вып. 2(17). С. 197–218.
20. *Фильченков А.А.* Алгоритмы построения элементов третичной полиструктуры алгебраической байесовской сети // Труды СПИИРАН. 2011. Вып. 3 (18). С. 237–266.
21. *Фильченков А.А.* Иерархия глобальных структур алгебраической байесовской сети как система графов и гиперграфов // Научно-технический вестник информационных технологий, механики и оптики. 2013. Вып. 1(83). С. 75–80.
22. *Фильченков А.А.* Меры истинности и вероятностные графические модели для представления знаний с неопределенностью // Труды СПИИРАН. 2012. Вып. 4(23). С. 254–295.

23. *Фильченков А.А.* Субоптимальная звездчатая структура алгебраической байесовской сети // Информационно-управляющие системы. 2013. Вып. 2. С. 13–17.
24. *Фильченков А.А., Мусина В.Ф., Тулупьев А.Л.* Алгоритм рандомизированного синтеза минимального графа смежности // Труды СПИИРАН. 2013. Вып. 2(35). С. 221–234.
25. *Фильченков А.А., Тулупьев А.Л.* Структурный анализ систем минимальных графов смежности // Труды СПИИРАН. 2009. Вып. 11. С. 104–127.
26. *Фильченков А.А., Тулупьев А.Л.* Совпадение множеств минимальных и нередуцируемых графов смежности над первичной структурой алгебраической байесовской сети // Вестник Санкт-Петербургского государственного университета. Серия 1. Математика. Механика. Астрономия. 2012. Вып. 2. С. 65–74.
27. *Фильченков А.А., Тулупьев А.Л., Сироткин А.В.* Компаративный анализ клик минимальных графов смежности алгебраических байесовских сетей // Труды СПИИРАН. 2010. Вып. 2 (13). С. 87–105.
28. *Фильченков А.А., Тулупьев А.Л., Сироткин А.В.* Особенности анализа вторичной структуры алгебраической байесовской сети // Труды СПИИРАН. 2010. Вып. 1 (12). С. 97–118.
29. *Фильченков А.А., Тулупьев А.Л., Сироткин А.В.* Ребра графов смежности в контексте компаративного анализа клик минимальных графов смежности алгебраических байесовских сетей // Труды СПИИРАН. 2010. Вып. 3 (14). С. 132–149.
30. *Фильченков А.А., Тулупьев А.Л., Сироткин А.В.* Структурный анализ клик минимальных графов смежности // Вестн. Тверск. гос. ун-та. Сер.: Прикладная математика. 2011. №20. С. 139–151.
31. *Cormen T., Leiserson C., Rivest R., Stein C.* Introduction to Algorithms Cambridge, MAThe MIT Press; 3rd edition, 2009. 1312 p.
32. *Prüfer H.* Neuer Beweis eines Satzes über Permutationen // Arch. Math. Phys. 1918. No. 27. S. 742–744.

Фильченков Андрей Александрович — аспирант кафедры информатики математико-механического факультета С.-Петербургского государственного университета (СПбГУ), младший научный сотрудник лаборатории теоретических и междисциплинарных проблем информатики СПИИРАН. Область научных интересов: автоматическое обучение вероятностных графических моделей. Число научных публикаций — 100. aaafil@mail.ru, СПИИРАН, 14-я линия В.О., д. 39, г. Санкт-Петербург, 199178, РФ; р.т. +7(812)328-3337, факс +7(812)328-4450. Научный руководитель — А.Л. Тулупьев.

Filchenkov Andrey Alexandrovich — PhD student of Computer Science Department, SPbGU, junior researcher, Theoretical and Interdisciplinary Computer Science Laboratory, SPIIRAS Research area: machine learning of probabilistic graphical models. The number of publications — 100. aaafil@mail.ru, SPIIRAS, 14-th line V.O., 39, St. Petersburg, 199178, Russia; office phone +7(812)328-3337, fax +7(812)328-4450. Scientific advisor — A.L. Tulupyev.

Фроленков Константин Владиславович — аспирант кафедры информатики математико-механического факультета С.-Петербургского государственного университета (СПбГУ), младший научный сотрудник лаборатории теоретических и междисциплинарных проблем информатики СПИИРАН. Область научных интересов: машинное обучение, вероятностный вывод. Число научных публикаций — 8. frolenk@mail.ru. Научный руководитель — А.Л. Тулупьев.

Frolenkov Konstantin Vladislavovich — PhD student of Computer Science Department, SPbGU, junior researcher, Theoretical and Interdisciplinary Computer Science Laboratory, SPIIRAS. Research area: machine learning, probabilistic inference. The number of publications — 8. frolenk@mail.ru. Scientific advisor — A.L. Tulupyev.

Сироткин Александр Владимирович — к.ф.-м.н., научный сотрудник лаборатории теоретических и междисциплинарных проблем информатики, СПИИРАН. Область научных интересов: алгебраические байесовские сети: вычислительные аспекты логико-вероятностного вывода в условиях неопределенности. Число научных публикаций – 60. avs@iias.spb.su; СПИИРАН, 14-я линия В.О., д. 39, г. Санкт-Петербург, 199178, РФ; р.т. +7(812)328-3337, факс +7(812)328-4450.

Sirotkin Alexander Vladimirovich — PhD, researcher, Theoretical and Interdisciplinary Computer Science Laboratory, SPIIRAS. Research interests: algebraic Bayesian networks, algorithms of probabilistic-logic inference under uncertainty. The number of publications – 60. avs@iias.spb.su, www.tulupyev.spb.ru; SPIIRAS, 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone +7(812)328-3337, fax +7(812)328-4450.

Тулупьев Александр Львович — д.ф.-м.н., профессор; заведующий лабораторией теоретических и междисциплинарных проблем информатики СПИИРАН, доцент кафедры информатики математико-механического факультета С.-Петербургского государственного университета (СПбГУ). Область научных интересов: представление и обработка данных и знаний с неопределенностью, применение методов математики и информатики в социокультурных исследованиях, применение методов биостатистики и математического моделирования в эпидемиологии, технология разработки программных комплексов с СУБД. Число научных публикаций — 250. ALT@iias.spb.su, www.tulupyev.spb.ru; СПИИРАН, 14-я линия В.О., д. 39, г. Санкт-Петербург, 199178, РФ; р.т. +7(812)328-3337, факс +7(812)328-4450.

Tulupyev Alexander Lvovich — PhD in Computer Science, Dr. of Sc. Professor; Head of Theoretical and Interdisciplinary Computer Science Laboratory, SPIIRAS, Associate Professor of Computer Science Department, SPbSU. Research area: uncertain data and knowledge representation and processing, mathematics and computer science applications in socio-cultural studies, biostatistics, simulation, and mathematical modeling applications in epidemiology, data intensive software systems development technology. Number of publications — 250. ALT@iias.spb.su, www.tulupyev.spb.ru; SPIIRAS, 14-th line V.O., 39, St. Petersburg, 199178, Russia; office phone +7(812)328-3337, fax +7(812)328-4450.

Рекомендовано ТиМПИ СПИИРАН, зав. лаб. А.Л. Тулупьев, д.ф.-м.н., проф.

Работа поступила в редакцию 01.03.2013.

Поддержка исследования. Работа выполнена при финансовой поддержке РФФИ, гранты №№ 12-01-00945-а и 12-01-31202-мол_а.

РЕФЕРАТ

Фильченков А.А., Фроленков К.В., Сироткин А.В., Тулупьев А.Л. **Система алгоритмов синтеза подмножеств минимальных графов смежности.**

Граф смежности — граф со строго определенными свойствами, который выступает в качестве вторичной структуры алгебраической байесовской сети (АБС) Графы смежности строятся над первичной структурой АБС — набором взаимно непоглощающих подалфавитов некоторого алфавита. В работе предлагается систематизация алгоритмов синтеза минимальных графов смежности (МГС) на основе теоремы о множестве минимальных графов смежности.

Предлагается двухэтапная схема построения подмножеств МГС. Первый этап состоит в построении четырех множеств: множества нагрузок, составляющих первичную структуру, множества стереосепараторов, множества обязательных ребер и множества владений стереосепараторов. Второй этап состоит в построении подмножества МГС на основе указанной четвертки за счет построения для каждого стереосепаратора жил определенного типа, соответствующего типу синтезируемого подмножества МГС.

Предложено три алгоритма построения различных подмножеств МГС: самого множества МГС, локально звездчатых графов смежности, облегчающих построение МГС с минимальным диаметром, и одноэлементного подмножества, содержащего реализацию случайного МГС. В первом случае строятся все жилы, во втором случае строятся жилы, являющиеся звездами, в последнем случае для каждого стереосепаратора строится случайным образом ровно одна жила. Показано, что сложность первых двух алгоритмов линейно зависит от размера синтезируемого множества.

Предложено шесть алгоритмов построения четвертки множеств, основанных на известных алгоритмах построения множества минимальных графов смежности.

Все приведенные в работе подмножества отличались тем, что жилы для них строятся независимо. Схема может быть также реализована для подмножеств МГС, у которых жилы стереосепаратора строятся зависимо от некоторых других жил (например, жил его сыновей). Примером такого подмножества являются звездчатые графы смежности.

Для всех алгоритмов даны оценки их сложности.

SUMMARY

Filchenkov A.A., Frolenkov K.V., Sirotkin A.V., Tulupyev A.L. **Minimal join graph subsets synthesis system.**

Join graph is a graph with strictly defined properties, which can represent algebraic Bayesian network (ABN) secondary. Join graphs are built over the ABN primary structure which is a set of mutually non-absorbent subalphabets of an alphabet. We suggest a system of minimal join graphs (MJG) synthesis algorithms based on the theorem on MJG set.

A two-step scheme for MJG subsets synthesis is proposed. The first step is to construct four sets: a workload set corresponding to the ABN primary structure, stereoseparator set, necessary edges set and set of stereoseparators holdings sets. The second step is to synthesize a MJG subset on given four sets by building sinews of a certain type for each stereoseparator. The type of sinews corresponds to the type of MJG being synthesized.

We proposed three algorithms for synthesizing three different MJG subsets: MJG set, the locally-stellate join graph set (locally-stellate join graphs helps to synthesize MJG with minimum diameter) and one-element subset consisting of a realization of the random MJG. In the first case, all sinews are built. In the second case, sinews which are stars are built. In the latter case, for each stereoseparator exactly one sinew is build, randomly chosen. It is shown that the complexity of the first two algorithms linearly dependent on the size of the set of synthesized.

We proposed six algorithms for constructing the four sets. The algorithms are based on the known algorithms for MJG set synthesis.

All the MJG subsets discussed in the paper are such, that all the sinews can be built independently. The scheme can also be implemented for MJG subsets, such that stereoseparators sinew should be build non-independently, built regardless of some of the other veins (e.g. sinews of its sons). An example for such a subset is stellate join graph set.

For all algorithms estimates of their complexity are given.