УДК 519.6

ПРЕДСКАЗАНИЯ НА ОСНОВЕ БАЙЕСОВСКИХ СЕТЕЙ ДОВЕРИЯ: АЛГОРИТМ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Е. Д. Масленников[1], **В. Б. Сулимов**[1]

Рассматривается алгоритм кластеризации, предназначенный для опроса байесовских сетей доверия. Алгоритм использует представление сети в виде древовидного графа — дерева сочленений. Подробно разбираются этапы построения данной структуры и работы с ней. Обсуждаются особенности программной реализации алгоритма. Работа выполнена при частичной поддержке государственного контракта с Федеральным агентством по науке и инновациям № 02.740.11.0300 от 7 июля 2009 г. и РФФИ (код проекта 09–07–12135 офи-м), а также в рамках научно-исследовательской работы МГУ имени М. В. Ломоносова “Постгеномные исследования и технологии” по теме “Информационные и вычислительные технологии в постгеномных исследованиях”.

Ключевые слова: байесовская сеть, сеть доверия, распространение свидетельств, экспертная система, дерево объединений, дерево сочленений, вероятностное распределение, распространение вероятности.

1. Введение. В настоящее время байесовские сети доверия (БСД) являются инструментом, крайне привлекательным для использования при разработке различных экспертных систем и систем искусственного интеллекта, успешно применяемым, например, при распознавании образов [1, с. 128] и в биоинформатике [1, с. 356]. Подобный интерес напрямую связан с накоплением огромного объема информации в естественно-научных областях и с необходимостью ее анализа при учете постоянного поступления новых данных. Модели на основе БСД способны к самообучению и самосовершенствованию по мере накопления экспериментальной информации. С этим свойством связана относительная нечувствительность таких моделей к возможным ошибочным или неполным данным. Преимуществом моделей БСД является также возможность интеграции разнородных данных — это происходит потому, что БСД моделируют самые общие причинно-следственные зависимости между интересующими исследователя параметрами. Эти зависимости выражены в виде некоторой вероятностной модели, достаточно гибкой для возможного описания причинно-следственных отношений самого общего вида. Алгоритмы обучения БСД допускают распараллеливание вычислений и могут быть реализованы в виде программ, выполняющихся на суперкомпьютерах. Модели БСД легко обходят избыточность данных и не подвержены проблемам оверфиттинга. Такие модели могут включать в себя предварительное априорное суждение или знание об изучаемой системе, полученное на основе предварительных экспериментов, экспертных заключений или интуиции. На сегодняшний день имеется опыт успешного создания экспертных систем на основе БСД [2, с. 15–384; 3].

БСД представляют собой структуры, обычно представленные графически в виде направленного ациклического графа и таблиц условных вероятностей для узлов графа, соответствующих определенным переменным [4, с. 23–42]. Процесс работы с ними заключается в выполнении двух основных операций: обучения (формирования таблиц условных вероятностей) БСД на основе имеющихся данных о переменных сети [5, 6] и непосредственного использования БСД для вычисления различных вероятностей, связанных с переменными сети.

Основной и наиболее часто выполняемой при использовании БСД операцией является опрос сети, т.е. вычисление вероятностей исходов (принятие какого-либо определенного значения из своей области значений) одной или нескольких переменных сети на основе таблиц условных вероятностей сети и, возможно, известных данных (называемых свидетельствами) о значениях, принимаемых другими переменными сети. Процедура опроса выполняется как на этапе обучения сети, так и при непосредственном использовании уже обученной сети, поэтому эффективность работы БСД во многом зависит от используемого алгоритма опроса и его реализации. Все известные методы опроса БСД можно условно разделить на две категории: алгоритмы, использующие представление БСД в более удобной для опроса форме (называемые также алгоритмами кластеризации), и приближенные алгоритмы [7–9], оперирующие стохастическими методами

[1] Научно-исследовательский вычислительный центр, Московский государственный университет им. М. В. Ломоносова, 119991, Ленинские горы, Москва; ООО “Димонта”, ул. Нагорная, д. 15, корп. 8, 117186, Москва; Е. Д. Масленников, студент, e-mail: ixidiciti@gmail.com; В. Б. Сулимов, зав. лаб., e-mail: vladimir.sulimov@gmail.com

вычислений. Имеются и другие методы, не относящиеся ни к одной из упомянутых категорий [10, с. 150–184; 11, с. 57–60], но они предназначены для работы со специфическими БСД некоторого определенного вида, что серьезно ограничивает возможности их применения.

В настоящей статье описывается алгоритм опроса БСД, использующий представление (кластеризацию) исходной сети в виде так называемого дерева сочленений (junction tree). Такой подход позволяет перейти от опроса сети общего вида к работе с древовидным графом, что существенно сокращает время вычислений, избавляя от необходимости во многих промежуточных расчетах. Данный алгоритм имеет ряд преимуществ, определяющих его предпочтение прочим алгоритмам опроса. Во-первых, использование дерева сочленений возможно для сетей любой топологической сложности, что делает этот алгоритм универсальным и применимым к очень широкому кругу задач. Во-вторых, в отличие от стохастических алгоритмов опроса дерево сочленений позволяет получить точные, а не приближенные значения требуемых вероятностей, при этом алгоритм обладает достаточно высокой скоростью работы (что будет показано ниже). Во всех популярных программах работы с БСД именно алгоритм дерева сочленений (в той или иной реализации) является основным алгоритмом опроса. Кроме того, для ряда задач, предполагающих работу с БСД и требующих точных результатов при сложной топологии сети, не существует приемлемого пути опроса, не использующего представления сети в виде дерева сочленений. Несмотря на рекордную распространенность и очевидную перспективность (если не необходимость) использования данного алгоритма, нам не удалось найти его подробного описания в соответствующих публикациях. В связи с этим в данной работе детально излагается наше понимание алгоритма опроса с помощью дерева сочленений и обсуждаются особенности его программной реализации.

2. Построение дерева сочленений. Деревом (древовидным графом) называется множество вершин, соединенных так, что из произвольной вершины A в произвольную вершину B можно попасть единственным образом. Излагаемый алгоритм опроса оперирует деревом сочленений — древовидным графом, каждая вершина которого содержит некоторый набор переменных и таблиц условных вероятностей рассматриваемой БСД. Блок-схема его построения представлена на рис. 1. Дерево сочленений формируется из доменного графа (domain graph) рассматриваемой сети — так называется граф, вершинами которого являются узлы рассматриваемой БСД, а ребрами соединяются те вершины, которые в сети были зависимы друг от друга, т.е. вероятности исходов одной вершины зависят от исходов других(ой) вершин(ы). Доменный граф не содержит в себе таблиц условных вероятностей исходной БСД, поэтому несет в себе информацию лишь о качественных, а не количественных характеристиках зависимостей переменных в сети. Пример БСД и ее доменного графа представлен на рис. 2. На этапах морализации и триангуляции в доменный граф добавляются дополнительные ребра, необходимые для дальнейшего преобразования в древовидный граф. После этого строится заготовка для дерева сочленений — дерево объединений. Оно представляет собой древовидный граф, узлами которого являются объединения вершин доменного графа. Далее в дерево объединений включаются таблицы условных вероятностей из рассматриваемой БСД — этим завершается процесс построения дерева сочленений. Ниже подробно рассмотрены все этапы построения. При этом не будет затрагиваться математическая сторона вопроса, интерпретирующая алгоритм в рамках аппарата теории вероятностей [12], излагается лишь последовательность действий, необходимых для получения дерева сочленений из БСД.

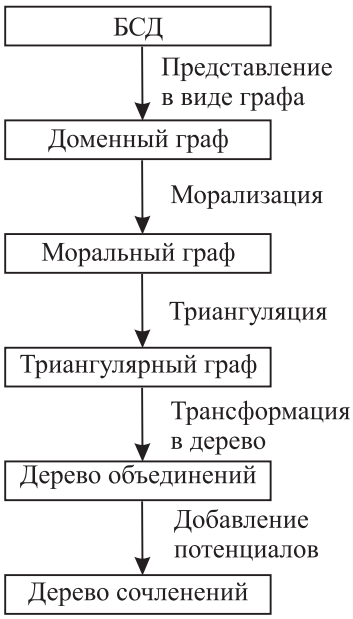

Рис. 1. Блок-схема построения дерева сочленений

Морализация доменного графа. Моральным (moral) называется доменный граф, в котором проведены дополнительные ребра между каждыми вершинами A и B, для которых в исходной БСД найдется вершина C, зависимая и от A, и от B (рис. 3а). Ниже на примере будет показана необходимость морализации доменного графа для дальнейшей работы алгоритма, пока же ограничимся теми соображениями, что дополнительные вершины в моральном графе указывают на косвенную связь между вершинами: на рис. 3а вершины A и B косвенно зависят друг от друга через вершину C; в частности, в случае поступления свидетельства для A изменится распределение вероятностей исходов для вершины C, что, в свою очередь, повлечет за собой изменение значений вероятностей различных исходов вершины B. Сам по себе термин “моральный” употребляется в связи с ассоциативным сходством зависимостей между вершинами и взаимоотношений между родителями и детьми: ситуация, в которой родители, имеющие общих детей, не “соединены” — “аморальна”. Стоит, однако, заметить, что морализация определена для любого числа

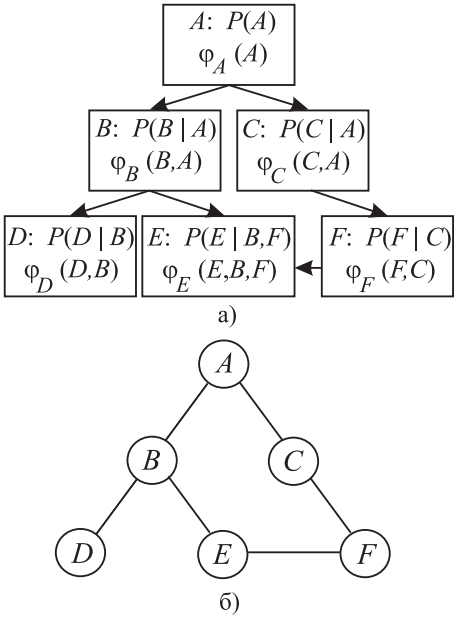

Рис. 2. Пример БСД (а) и его доменный граф (б)

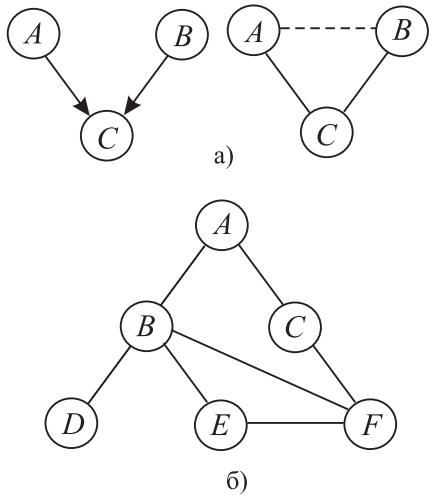

Рис. 3. Принцип морализации (а) и морализованный граф (б) с рис. 2б

(не только для двух) вершин, имеющих общих “потомков”.

Триангуляция. Граф называется триангулярным (triangulated), если в нем отсутствуют циклы из четырех и более вершин. Цикл в данном случае определяется как множество вершин, в котором каждая вершина соединена ровно с двумя другими вершинами этого множества. Применительно к использованию дерева сочленений необходимо добавить, что триангуляция производится для морализованного доменного графа. Отметим также, что термин “триангулярный” может ввести в заблуждение, если ассоциировать его с треугольником (triangle) — на рис. 4а представлен классический пример ошибочной трактовки этого понятия.

Приведем один из возможных алгоритмов триангуляции графа.

1. Пусть дан граф G. Создадим копию этого графа и обозначим ее G'.

2. Пока граф G' не пуст (т.е. в нем присутствуют вершины), выбираем из графа G' вершину V, имеющую наименьшее число соседних вершин. Если вершин, удовлетворяющих этому условию, несколько, то выбираем вершину, имеющую минимальный вес (произведение чисел принимаемых значений переменной узла БСД, соответствующего этой вершине, и переменных узлов, соответствующих вершинам ее соседей). Если несколько вершин имеют минимальный вес — выбираем любую из них. Соединяем все вершины из числа соседей V, т.е. каждый сосед V должен быть соединен со всеми остальными соседями V. Аналогичные соединения также добавляем в граф G. Удаляем V из G' и повторяем шаг 2 алгоритма.

3. Теперь граф G — триангулярный.

Построение дерева объединений (join tree). Заготовкой для дерева сочленений служит дерево объединений — древовидный граф, каждый узел которого представляет собой такое подмножество вершин триангулярного графа, что каждые две вершины этого подмножества соединены ребром графа. Подобное множество вершин также называется кликой (clique) графа. При этом связи между ними организованы так, что для любых двух вершин ψ и ξ любая вершина ζ, расположенная на пути между ними, содержит в себе все переменные, которые принадлежат одновременно и ψ, и ζ: $(\psi \cap \xi) \in \zeta$.

Построение дерева объединений можно осуществить по следующему алгоритму.

1. Пусть дан триангулярный граф G. Создадим некоторую переменную i, которую будем использовать в качестве счетчика, изначально $i = 0$.

2. Пока граф G не пуст, выбираем из триангулярного графа G вершину V, имеющую наименьшее число соседних вершин. Если вершин, удовлетворяющих этому условию, несколько, то выбираем вершину, имеющую минимальный вес. Если и таких вершин несколько, выбираем любую из них. Множество, состоящее из вершины V и ее соседей, помечаем как C_i (узел, или вершина дерева); множество вершин из C_i, имеющих соседей, не принадлежащих C_i (если такие вершины имеются), помечаем как S_i (множество-сепаратор (separator)). Удаляем из G тех соседей V, которые не вошли в сепаратор S_i, удаляем V из G,

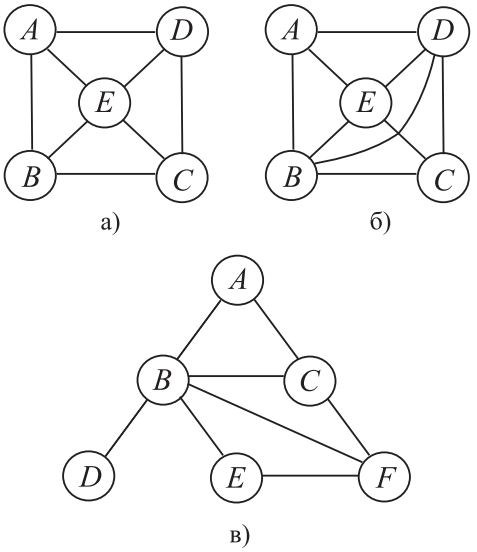

Рис. 4. Примеры нетриангулярного (а) и триангулярного (б) графов; результат триангуляции (в) графа с рис. 3б

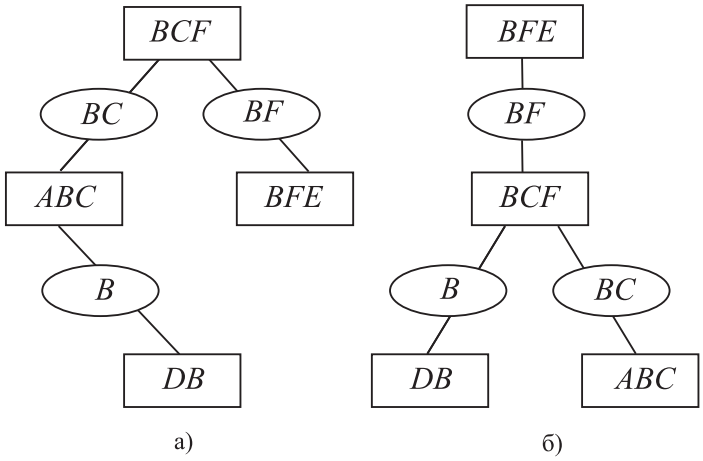

Рис. 5. Возможные варианты дерева объединений для графа на рис. 4в

увеличиваем счетчик i на 1 и повторяем шаг 2 алгоритма.

3. Каждый узел C_i соединяем с сепаратором S_i. При этом узел, который был создан последним, на данном шаге алгоритма остается несоединенным из-за того, что сепараторов на один меньше числа узлов дерева.

4. Каждый сепаратор S_i соединяем с таким узлом C_j, что $j > i$, C_j содержит все элементы S_i ($S_i \in C_j$). Данное соединение не всегда однозначно (рис. 5). Однако любой из полученных древовидных графов можно использовать в качестве заготовки для дерева сочленений.

5. Полученная структура из узлов в виде клик триангулярного графа, соединенных между собой через сепараторы, является деревом объединений.

Потенциалы. Перед тем как перейти к заключительному этапу построения дерева сочленений и дальнейшей работе с ним, коротко остановимся на вопросе об использовании потенциалов в БСД.

Потенциалы являются дискретными функциями нескольких переменных и представляют собой иную форму записи условных вероятностей: $P\big(x|y_1, y_2, \ldots\big) = \varphi(x, y_1, y_2, \ldots)$. Такое представление упрощает как понимание, так и реализацию алгоритма.

Возможно сокращение области определения потенциала — путем маргинализации (marginalization) переменной (здесь и далее суммирование производится по всей области значений, принимаемых переменными, стоящими под знаком суммы):

$$\phi(x_1, x_2, \ldots, x_{k-1}, x_{k+1}, \ldots, x_N) = \sum_{x_k} \varphi(x_1, x_2, \ldots, x_{k-1}, x_k, x_{k+1}, \ldots, x_N).$$

Кроме того, используется понятие проецирования (projection) потенциалов:

$$\psi(x_1, \ldots, x_k) = \phi(x_1, x_2, \ldots, x_N)^{\downarrow(x_1,\ldots,x_k)} \equiv \sum_{x_{k+1},\ldots,x_N} \phi(x_1, x_2, \ldots, x_N).$$

Другими словами, проецирование — это маргинализация всех переменных, не входящих во множество, на которое производится проецирование (в рассмотренном примере — множество $(x_1, \ldots, x_k)$). В ходе дальнейшей работы с деревом сочленений также будет использована операция перемножения потенциалов:

$$\psi(x_1, \ldots, x_n, y_1, \ldots, y_m, z_1, \ldots, z_k) = \sigma(x_1, \ldots, x_n, y_1, \ldots, y_m)\varphi(y_1, \ldots, y_m, z_1, \ldots, z_k).$$

Подробнее рассмотрим эту операцию на следующем простом примере. Пусть имеются переменные A, B и C, принимающие значения “0” или “1”, и потенциалы $\sigma(A, B)$ и $\varphi(B, C)$, принимающие значения в зависимости от A и B (табл. 1 и 2). Тогда потенциал $\psi(A, B, C) = \sigma(A, B)\varphi(B, C)$ будет принимать значения, представленные в табл. 3.

Таблица 1

A	B	$\sigma(A,B)$
0	0	A
0	1	B
1	0	C
1	1	D

Таблица 2

B	C	$\varphi(B,C)$
0	0	E
0	1	F
1	0	G
1	1	H

Таблица 3

A	B	C	$\psi(A,B,C)$
0	0	0	AE
0	0	1	AF
0	1	0	BG
0	1	1	BH
1	0	0	CE
1	0	1	CF
1	1	0	DG
1	1	1	DH

Дерево сочленений. Деревом сочленений рассматриваемой БСД называется ее дерево объединений, в котором каждой вершине сопоставлено некоторое множество потенциалов условных вероятностей сети по правилу, что каждый потенциал сопоставляется любой вершине дерева, полностью содержащей его область определения: $\varphi(x_1, x_2, \ldots, x_N) \to V_i \iff (x_1, x_2, \ldots, x_N)$. В итоге получим структуру из узлов, соединенных между собой через сепараторы, каждый узел содержит множество переменных, представляющих собой клику триангулированного доменного графа, и набор потенциалов сети, области определения которых полностью содержатся во множестве переменных, принадлежащих этой клике. В процессе работы с деревом сепараторы также будут содержать в себе потенциалы, областью определения которых будет являться множество переменных сепаратора — на данном этапе потенциалы содержатся лишь в узлах дерева. Построив дерево сочленений для рассматриваемой БСД, можно перейти непосредственно к процедуре опроса сети с использованием дерева.

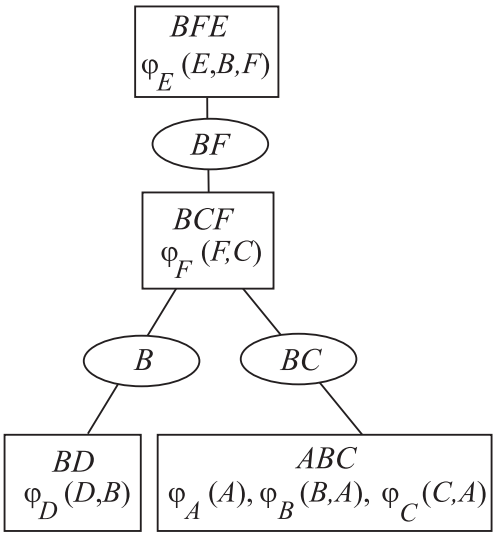

Рис. 6. Дерево сочленений для БСД на рис. 2а

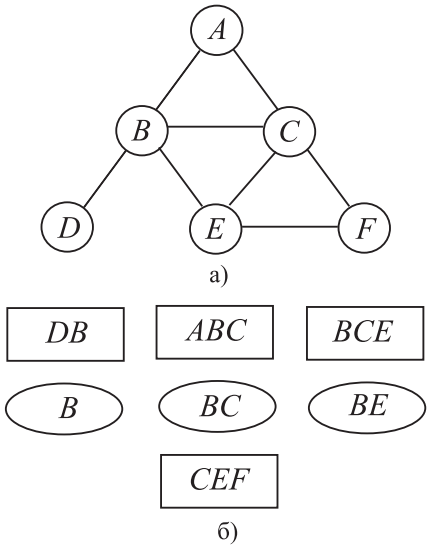

Рис. 7. Результат триангуляции графа на рис. 2б (а) и набор узлов и сепараторов, из которых можно составить дерево смежности для этого графа (б)

Именно на стадии финальной инициализации дерева сочленений можно отчетливо показать необходимость морализации доменного графа. Пусть строится дерево объединений для графа на рис. 2б, при этом триангуляция этого графа была произведена без предшествующей морализации (рис. 7а). Тогда дерево объединений будет состоять из узлов и сепараторов, изображенных на рис. 7б. В таком случае переход к дереву сочленений окажется невозможным, так как в дереве объединений нет узла, содержащего, например, область определения потенциала $\varphi_E(E, B, F)$.

3. Опрос БСД с помощью дерева. Процесс вычисления конкретных вероятностей исходов рассматривается как распространение (propagation) свидетельства в дереве сочленений — считая одну из вершин "корнем" дерева, потенциалы узлов (и сепараторов) обновляются с учетом поступившего свидетельства последовательно от "листьев" к "корню". Следует отметить, что отсутствие свидетельства принципиально не сказывается на работе алгоритма, а результатом опроса в этом случае будут являться значения вероятностей, с какими переменные сети принимали бы различные значения так, как если бы были независимы друг от друга. Иначе этот процесс называется вычислением абсолютных вероятностей переменных БСД.

На рис. 8 представлена блок-схема распространения свидетельства в дереве сочленений. После сбора сведений возможны два варианта дальнейшей работы: либо вычисление вероятностей исходов той переменной сети, что содержалась в "корне" (рис. 8а), либо запуск распределения сведений с последующим вычислением вероятностей исходов всех переменных сети (рис. 8б). На рис. 9 на примере схематично представлены обе эти процедуры, а ниже рассмотрены алгоритмы, их реализующие.

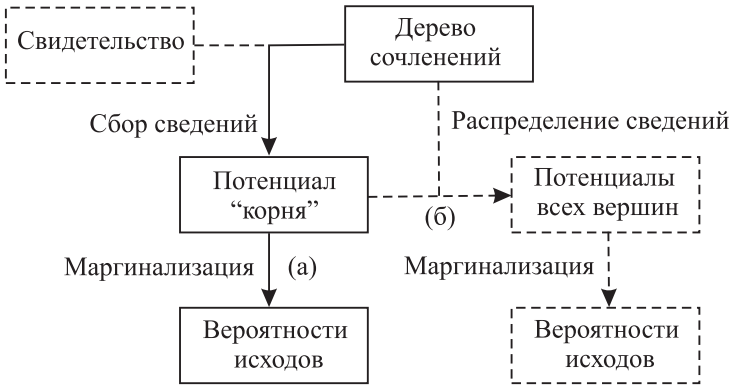

Рис. 8. Блок-схема распространения свидетельства в дереве сочленений: а) для вычисления вероятностей исходов одной переменной, б) для вычисления вероятностей исходов всех переменных сети. Пунктиром показаны необязательные блоки

Сбор сведений (Collect evidence). Приведенный ниже алгоритм решает задачу вычисления вероятностей исходов некоторой переменной исследуемой БСД при заданном свидетельстве.

1. Пусть требуется вычислить вероятности различных исходов переменной V сети. Выбираем "корень" дерева, т.е. любую вершину дерева сочленений, содержащую переменную V (обозначим эту вершину через T). На данном этапе предполагается, что все сепараторы дерева не содержат потенциалов.

2. Распространение начать с вершин, с которыми соединен только один сепаратор. Отметим, что на данном и следующем этапах мы не рассматриваем вершину T. Распространение осуществляется следующим образом: все потенциалы, принадлежащие данной вершине, перемножаются между собой, затем полученный потенциал проецируется на множество переменных сепаратора, с которым соединена вершина, и помещается в сепаратор.

3. Далее распространение можно продолжить через остальные вершины по следующему правилу. Ищем вершину, у которой только один соседний сепаратор не содержит потенциала; перемножаем между собой все потенциалы этой вершины и полученный потенциал перемножаем с потенциалами соседних сепараторов, затем проецируем на множество переменных "пустого" соседнего сепаратора; повторяем шаг 3 алгоритма, если это возможно. Напомним, что по ходу распространения мы не трогаем вершину T.

4. Теперь все соседние сепараторы вершины T содержат потенциалы. Перемножим все потенциалы вершины T и соседних сепараторов. Теперь спроецируем полученный потенциал на множество $\{V\}$. Полученный потенциал, зависящий только от одной переменной V, после нормировки на 1 будет содержать в себе искомые вероятности. Следует отметить, что необходимости в промежуточных нормировках потенциала на каждом шаге распространения свидетельства нет — их отсутствие (или наличие) на итоговый результат никак не влияет.

Поясним, каким образом при распространении учитываются свидетельства. В распространение включаются дополнительные потенциалы вида $\{0,\dots,1,\dots,0,0\}$, зависящие от одной переменной (на которую и поступило свидетельство), где 1 соответствует значению, принимаемому переменной. Такую функцию следует добавить на этапе сбора сведений в месте, где используется потенциал, соответствующий узлу БСД, относящемуся к переменной, на которую поступило свидетельство.

Распределение сведений (Distribute evidence). Алгоритм сбора сведений успешно справляется с задачей вычисления вероятностей исходов одной переменной, однако при анализе всех переменных сети рациональнее воспользоваться следующим ниже алгоритмом, нежели прибегать к процедуре сбора сведений для каждой переменной сети.

1. Пусть имеется вершина дерева T, для которой уже проделана процедура сбора сведений. Полученные в результате этого потенциалы сепараторов $\{S_i\}$ обозначим через $\{\psi_i\}$. (Далее все потенциалы,

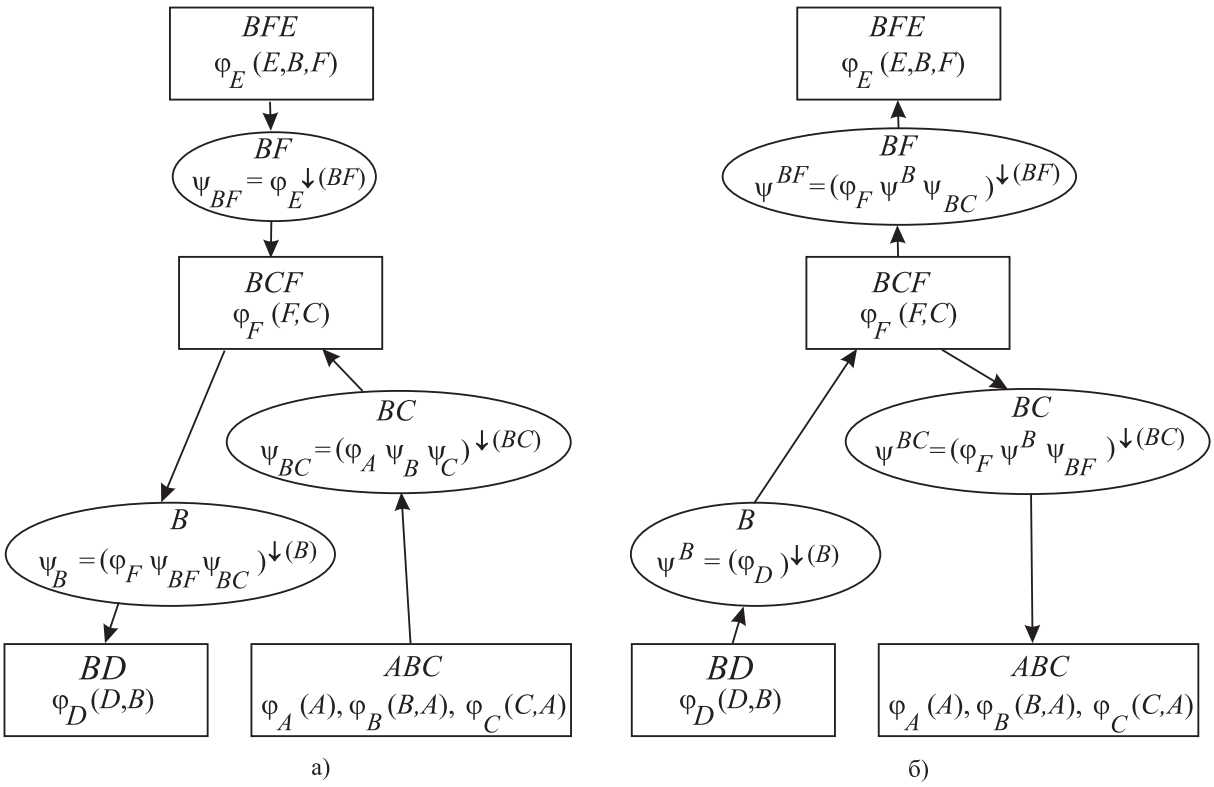

Рис. 9. Сбор сведений (а) и распределение сведений (б) в дереве сочленений на рис. 6. Стрелками показаны направления, по которым осуществляется распространение свидетельства. В данном случае целью сбора сведений является вычисление вероятностей исходов переменной D

получаемые в результате распределения сведений, будем обозначать $\{\psi^i\}$.) Обозначим через ζ множество сепараторов, смежных с вершиной T. В каждый сепаратор S_j из ζ поместим потенциал

$$\psi^j = \left(\varphi_T \cdot \prod_{i\neq j:\, S_i\in\zeta} \psi_i\right)^{\downarrow(S_j)},$$

где φ_T — потенциал вершины T. Все эти сепараторы содержат теперь в себе по два потенциала: ψ^i и ψ_i ($\forall i : S_i \in \zeta$). Кроме того, вероятность любой переменной V, содержащейся в T, можно вычислить по формуле

$$\forall V \in T: \;\; P(V) = \left(\varphi_T \cdot \prod_{i:\, S_i\in\zeta} \psi_i\right)^{\downarrow(V)},$$

2. Ищем вершину дерева T', у которой более одного смежного сепаратора, и лишь один из них содержит два потенциала. Множество сепараторов, смежных с T', обозначим через ζ', а сепаратор с двумя потенциалами — через S_k. В каждый сепаратор S_j (кроме S_k) из ζ' поместим потенциал

$$\psi^j = \left(\varphi_{T'} \cdot \psi^k \prod_{i\neq j\neq k:\, S_i\in\zeta'} \psi_i\right)^{\downarrow(S_j)},$$

где $\varphi_{T'}$ — потенциал вершины T'. Вероятность любой переменной V, содержащейся в T', можно вычислить по формуле

$$\forall V \in T': \;\; P(V) = \left(\varphi_{T'} \cdot \psi^k \prod_{i\neq k:\, S_i\in\zeta'} \psi_i\right)^{\downarrow(V)},$$

Отметим, что эта формула также верна и для вершин с единственным смежным сепаратором.

3. Шаг 2 следует повторять до тех пор, пока все сепараторы дерева не будут содержать по два потенциала. Когда это произойдет, распределение сведений (да и распространение в дереве в целом) можно считать завершенным.

4. Реализация алгоритма. На сегодняшний день дерево сочленений — наиболее распространенный алгоритм опроса, используемый при работе с БСД. Однако в случае достаточно громоздких сетей часто приходится считаться с высокими требованиями к аппаратным ресурсам и длительным временем работы программы, при этом дело вовсе не в нерациональности алгоритма дерева сочленений, а в его программной реализации. В связи с этим в данном разделе будут рассмотрены особенности написания программ, использующих данный алгоритм. Опираться будем на язык C++, хотя все эти соображения в той или иной степени применимы ко всем языкам программирования.

Представление в памяти. В общем случае сеть удобно представить в виде набора динамического массива структур, включающих в себя имя узла, список родительских узлов и его таблицу условных вероятностей в потенциальном виде. Однако, если стоит задача реализации одного лишь алгоритма опроса (а не универсальной программы для работы с БСД), представление сети в памяти можно существенно упростить. Нетрудно заметить, что в ходе работы алгоритма используются не непосредственно потенциалы всех узлов сети, а лишь их произведения (содержащиеся в узлах дерева сочленений). Поэтому имеет смысл сразу при получении на входе программы таблиц условных вероятностей узлов сети ассоциировать их с соответствующими вершинами дерева сочленений и хранить лишь их произведения. Разумеется, для этого необходимо построить дерево смежности до того, как программа получит данные о таблицах условных вероятностей (о том, как это можно реализовать, будет сказано ниже). Структуру, соответствующую вершине БСД, можно описать следующим образом:

```
struct node
{
    string name;        //имя вершины БСД
    vector<string> value;       //имена принимаемых значений
    vector<int> parent; //индексы вершин БСД - предков данной вершины
    int clique;         //индекс ассоциируемого узла дерева сочленений
};
```

Информация об узлах-родителях данной вершины хранится в виде массива, содержащего индексы вершин БСД, соответствующие узлам-родителям. В этой структуре необходимо также хранить информацию о вершине (индекс той вершины) дерева сочленений, в которую на этапе его инициализации был добавлен потенциал этого узла.

В процессе построения дерева смежности предстоит работа с графами. Структуру, отвечающую вершине графа, можно представить в виде

```
struct graphnode
{
    vector<int> neighbour;
    bool removed;
};
```

Первая строка тела структуры содержит массив индексов соседних вершин графа. Для удобства при создании графа индексы его вершин соответствуют индексам вершин БСД. Поясним смысл переменной, описанной во второй строке. Вспомним, что в процессе триангуляции графа предполагается создание копии исходного графа, в котором затем последовательно удаляются вершины и добавляются ребра, причем добавляются они также и в исходный граф. В данном случае вместо двух графов можно вполне обойтись одним, если в каждой вершине поместить логическую переменную, отвечающую за “наличие” вершины в графе. Иными словами, вместо удаления вершины во временном графе-копии просто меняем значение этой самой переменной.

Наиболее часто используемые в программе структуры — потенциалы. Поэтому, для снижения потребляемых программой ресурсов, потенциалы рекомендуется представить в как можно более простой форме, например в виде

```
struct potential
{
    vector<int> node;
    vector<float> prob;
};
```

В первой строке тела структуры описан динамический массив, отвечающий за переменные, входящие в область определения потенциала. Во всей программе переменные удобно идентифицировать не их

именами, а индексами, соответствующими индексам узлов сети, отвечающих данным переменным. Таким образом, имена переменных представлены в программе лишь один раз — внутри описания соответствующих узлов, в дальнейшем используются их индексы. Поэтому в теле структуры потенциала (равно как и всех остальных используемых структур, кроме самих узлов) массив переменных имеет целочисленный тип.

В следующей строке представлены значения функции потенциала. Первый элемент массива соответствует значению функции в случае, когда все переменные из области определения принимают первое из своих возможных значений, второй элемент — когда все переменные, кроме последней, принимают свое первое значение, а последняя переменная — свое второе значение и т.д. Для наглядности рассмотрим пример. Пусть область определения потенциала состоит из трех переменных A, B и C. Пусть A и C принимают значения 1 и 2, а B принимает значения 1, 2, 3. Тогда зависимость между индексами массива со значениями функции потенциала и значениями переменных будет такой, какой она представлена в табл. 4.

Теперь рассмотрим структуры, используемые непосредственно при описании дерева сочленений. Вот один из возможных вариантов представления сепаратора:

Таблица 4

Индекс	Значение переменной		
0	A	B	C
1	1	1	1
2	1	1	2
3	1	2	1
4	1	2	2
5	1	3	1
6	1	3	2
7	2	1	1
8	2	1	2
9	2	2	1
10	2	2	2
11	2	3	1
12	2	3	2

```
struct separator
{
    potential pot;
    bool empty;
};
```

В этой структуре нет ничего особенного: потенциал, область определения которого несет в себе информацию о переменных сепаратора, и логическая переменная, отвечающая за достижение распространением сведений данного сепаратора. Отметим, что в случае реализации алгоритма распределения сведений описание сепаратора должно содержать в себе два потенциала.

В свою очередь, структура, характеризующая вершину дерева, может выглядеть так:

```
struct treenode
{
    vector<int> node;
    potential pot;
    bool evd;
    vector<int> spr;
};
```

Она состоит из двух массивов: массива переменных, содержащихся в вершине, и массива смежных с вершиной сепараторов, представленных индексами этих сепараторов. Кроме того, в ней содержатся потенциал данной вершины дерева (являющийся, по сути, произведением некоторых потенциалов БСД) и логическая переменная, отвечающая за достижение распространением сведений данной вершины.

Отметим также, что в некоторых компиляторах C++ вместо целочисленного типа int целесообразно использовать тип short, занимающий вдвое меньше места в памяти. При выборе используемого типа следует исходить как из диапазона значений, принимаемых переменными того или иного типа в используемой версии компилятора, так и из круга задач, которые предполагается решать с помощью программы.

Входные данные. Выше было упомянуто, что, в связи с упрощенной конструкцией структур узлов БСД, необходимо строить дерево смежности до того, как программа получит данные о таблицах условных вероятностей сети. Поэтому файлы, содержащие данные о БСД и предназначенные для обработки программой, рекомендуется представить следующим образом. Сначала идет описание “топологии” сети: обозначаются имена узлов БСД и указываются их родительские узлы. Затем, после обозначения всех вершин сети, следует описание их таблиц условных вероятностей: имена значений, принимаемых переменными сети, и непосредственно потенциалы узлов. В файле может присутствовать также информация об известных экспериментальных данных (свидетельствах), однако (в зависимости от поставленной задачи, разумеется) рекомендуется свидетельства представлять в отдельном файле или реализовать возможность добавления свидетельств через интерфейс во время работы программы. В любом случае данные о свидетельствах программа должна получать уже после данных о сети и ее потенциалах.

Ниже приведен пример файла БСД (формата ".cpt"), предназначенного для обработки разработанной нами программой junty (Масленников Е.Д., Сулимов В.Б. Программа для ЭВМ JUNTY. Свидетельство о государственной регистрации программ для ЭВМ № 2010611063 от 6 мая 2010 г., Федеральная служба по интеллектуальной собственности, патентам и товарным знакам), реализующей алгоритм дерева сочленений.

файл "net.cpt":

```
//Простой пример файла с описанием сети.
//(Обсуждается зависимость цвета волос детей
//от цвета волос родителей)
//В файле можно использовать ‘‘//’’ в качестве обозначения комментариев,
//Пустые строки НЕ игнорируются программой - в качестве внеформатного
//разделителя используется символ ‘‘//’’
//
//Описание топологии сети в формате
//’’имя_узла имя_1-го_родителя имя_2-го_родителя ‘‘ и т.д.
mother
father
son mother father
dauther mother father

//после описания топологии - пустая строка
//
//Список таблиц условных вероятностей - в том порядке, в каком
//переменные записаны в описании топологии.
//После каждой таблицы - пустая строка.
//Первая строка каждой таблицы - список значений очередной переменной.
//
//mother
blonde  brown
.2      .8

//father
blonde  brown
.15     .85

//son
blonde brown  //father  mother
.75     .25     blonde  blonde
.55     .45     blonde  brown
.4      .6      brown           blonde
.03     .97     brown           brown

//dauther
blonde brown  //father  mother
.85     .15     blonde  blonde
.45     .55     blonde  brown
.65     .35     brown           blonde
.05     .95     brown           brown

//Конец файла.
```

Операции над потенциалами. При работе с потенциалами, если они представлены в виде структур, описанных выше, надо постоянно следить за правильным порядком записи значений вероятностей в соответствующий массив. Приведем код функции исключения (маргинализации) переменной из потенциала — обращение к этой функции происходит каждый раз в процессе проецирования потенциала, поэтому она является одной из наиболее часто вызываемых в программе:

```
potential eliminate(potential p, int n)         //исключаем переменную, соответствующую узлу
                                                //БСД с индексом n
{
    potential c;                //сюда будет записываться результат
    int i=epos(n,p.node);       //позиция исключаемой переменной в массиве, характеризующем
                                //область определения потенциала
    int s=1;            //вспомогательная переменная для определения нужного индекса в
                        //массиве вероятностей
    c.node=p.node;              //результирующий потенциал имеет область определения
                                //исходного потенциала
    c.node.erase(c.node.begin()+i);     //минус исключаемая переменная
    //в следующих строках и пригождается переменная s
    for (int j=i+1;j<p.node.size();j++) s*=bn[p.node[j]].value.size();
   //bn - это массив структур узлов, в нем и хранится БСД
   //bn, очевидно, описан глобально, т.е. вне тела функции
    for (int d=0;d<p.prob.size();d+=(s*bn[n].value.size()))
    {
        for (int j=d;j<(d+s);j++)
        {
            float x=0;          //в эту переменную будет записываться суммарная вероятность,
                                //полученная в результате удаления переменной
          //пересчет вероятности с учетом удаления переменной
          for (int k=0;k<bn[n].value.size();k++) x+=p.prob[j+k*s];
            c.prob.push_back(x);        //запись вероятности в результирующий потенциал
        }
    }
    return c;           //возвращаем результат
}
```

Функция “epos” является вспомогательной, она возвращает позицию элемента массива в массиве (или -1, если элемент отсутствует в массиве). Аналогичным образом происходит работа с массивом вероятностей при написании функции умножения потенциалов.

Распространение в дереве. Теперь перейдем непосредственно к процедуре опроса, рассмотрев функцию, реализующую сбор сведений с целью вычисления вероятностей исходов некоторой переменной данной сети (возможно, с некоторыми заданными свидетельствами).

```
potential collect_evidence(int n)         //n - индекс переменной в БСД
{
    int i;
    cevd=clq.size();    //cevd - переменная, определенная вне тела функции,
        //предназначенная для контроля количества
        //вершин дерева, до которых дошло распространение
    for (i=0;i<spr.size();i++)  //инициализация
    {
        spr[i].empty=true;
        clq[i].evd=false;
    }
    clq[i].evd=false;   //вершин на одну больше сепараторов

//пока количество неосвидетельствованных вершин более одной,
//производится распространение во все вершины (кроме той, с которой был
//ассоциирован потенциал переменной сети с индексом n) через все
//незаполненные сепараторы
    while (cevd>1)
        for (i=0;i<clq.size();i++) evdpass(i,bn[n].clq);

//получение итогового потенциала с областью определения из одной переменной
//(с индексом n в БСД)
```

```
    potential c=clq[bn[n].clq].pot;
    for (i=0;i<clq[bn[n].clq].spr.size();i++)  c=merge(c,spr[clq[bn[n].clq].spr[i]].pot);
    return norm(c,n);   //итоговое проецирование и нормировка
}
```

Функция “evdpass” осуществляет распространение через вершину дерева, если до этой вершины еще не дошло распространение и если вершина имеет ровно один незаполненный сепаратор, смежный с ней. Кроме того, данная функция не производит распространение через вершину, с которой был ассоциирован потенциал целевой переменной сети. Функция “merge” предназначена для перемножения двух потенциалов. Отдельно следует упомянуть функцию “norm”, которая проецирует потенциал на множество из некоторой одной переменной и нормирует его. Использование этой функции (в данном месте программы) рациональнее последовательного вызова функции “eliminate” (см. выше) до тех пор, пока область определения потенциала не будет состоять из одной переменной, и последующей нормировки. Приведем ее код:

```
potential norm(potential p, int n) //проекция на {n} и нормировка
{
    potential c;
        //вспомогательные переменные
    int e=epos(n,p.nd), q=1,w=1,ss=bn[n].val.size();
    vector<float> qq(ss, 0.0);
    c.nd.push_back(n);
    c.prob=qq;  //инициализация потенциала с нулевыми значениями
    for (int i=e+1;i<p.nd.size();i++) q*=bn[p.nd[i]].val.size();
    for (int i=0;i<e;i++) w*=bn[p.nd[i]].val.size();
    int z=0;    //переменная, отвечающая интересующим индексам для суммирования
        //суммирование
    for (int j=0;j<w;j++)
    {
        for (int k=0;k<ss;k++)
        {
            for (int i=0;i<q;i++)
            {
                c.prob[k]+=p.prob[z];
                z++;
            }
        }
    }
        //нормировка
    float x=0;
    for (int i=0;i<c.prob.size();i++) x+=c.prob[i];
    for (int i=0;i<c.prob.size();i++) c.prob[i]/=x;
    return c;
}
```

Добавление свидетельств. Напоследок отметим некоторые особенности обработки программой свидетельств. При рассмотрении данного вопроса нужно принимать во внимание предполагаемую область применения программы. В случае если на вход программы за время ее работы поступает не более одного набора свидетельств, то для его включения в процесс распространения достаточно домножить потенциал(ы) вершин(ы) дерева сочленений, с которой(ыми) был(и) ассоциирован(ы) на этапе инициализации потенциал(ы) узла(ов) БСД переменной(ых), на которую(ые) поступило свидетельство, на потенциал(ы) одной переменной вида $\{0, 0, \ldots, 1, 0\}$, где 1 соответствует наблюдавшемуся(имся) значению(ям).

Если в процессе работы программы с одной БСД предполагается поочередная обработка нескольких независимых друг от друга свидетельств (например, в случае наличия возможности интерактивного задания свидетельств через интерфейс программы), встает вопрос сохранения изначальных потенциалов в узлах дерева сочленений (тех, что получены на этапе инициализации, до добавления свидетельств). Поэтому в данном случае рекомендуется немного видоизменить структуру, отвечающую вершине дерева сочленений.

```
struct treenode
{
    vector<int> node;
    potential pot;
    bool evd;
    vector<int> anode;
    vector<int> spr;
};
```

Добавленная строка отвечает массиву индексов тех переменных БСД, потенциал которых были ассоциирован с данной вершиной дерева на этапе инициализации. В процессе распространения через эту вершину проверяется, не имеется ли свидетельств на какую-либо из этих переменных. В случае присутствия таких свидетельств, в распространении участвует не потенциал вершины, а произведение потенциала одной переменной вида $\{0, 0, \ldots, 1, 0\}$ (где 1 соответствует наблюдавшемуся значению) и потенциала вершины. Последний при этом сохраняется, так как в процессе распространения изменения претерпевают только потенциалы сепараторов.

5. Валидация программы. В этом разделе представлены результаты валидации разработанной нами программы "junty", реализующей алгоритм дерева сочленений. "Junty" представляет собой консольное кросс-платформенное приложение, позволяющее для данных БСД и свидетельств получить вероятности исходов одной или нескольких переменных сети (предварительно обученной или с искусственно заданными таблицами условных вероятностей).

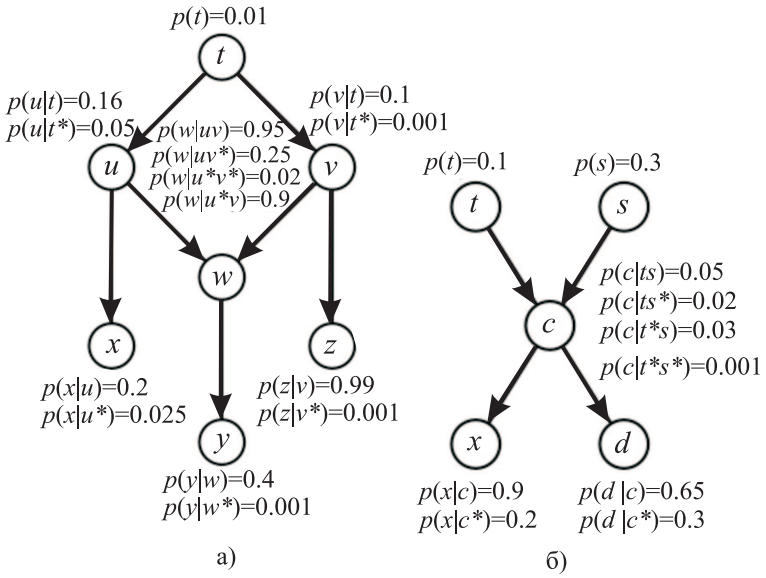

Рис. 10. Примеры простых БСД, использованных при валидации программы: а) [13, с. 346], б) вариация примера из [11, с. 31]

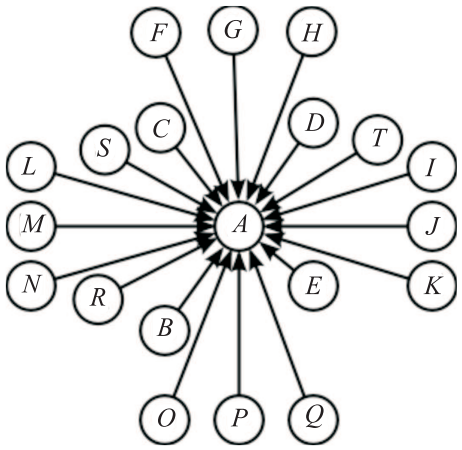

Рис. 11. Топология, требующая при заданном количестве переменных (с заданным количеством принимаемых значений) наибольших затрат вычислительных ресурсов

В начале валидации была проконтролирована работа программы с некоторыми простыми сетями, для которых результаты работы программы можно было проверить, вычислив необходимые вероятности "вручную". Примеры таких сетей, взятых нами из литературы [11, с. 31; 13, с. 346], представлены на рис. 10. Кроме того, результаты также контролировались программой "Netica" (www.norsys.com), предназначенной для работы с БСД и использующей для опроса некоторую вариацию алгоритма дерева сочленений. В ходе валидации подобными сетями результаты программы "junty" в 100% случаев совпадали (с точностью до округления итоговых вероятностей в четвертом знаке после запятой) с результатами программы "Netica" и прямыми расчетами (на калькуляторе).

Дальнейшая валидация осуществлялась на сложных сетях различной топологии, и для примера одна из них приведена на рисунке (http://www.dimonta.com/files/fig11.png), при этом результаты контролировались с помощью программы "Netica". При работе с подобными сетями результаты программы "junty"

опять же совпадали (с точностью до округления итоговых вероятностей в четвертом знаке после запятой) с результатами, полученными программой “Netica”, в 100% случаев.

Следует упомянуть скорость проведения опроса. Для оценки характерных времен работы программы была выбрана сеть такой топологии (рис. 11): 19 независимых между собой переменных и еще одна переменная, непосредственно зависящая от всех остальных. При заданном количестве переменных эта топология требует максимальных затрат вычислительных ресурсов, так как программе приходится работать с потенциалами максимально возможного (в рамках данного набора переменных) размера. Программа, откомпилированная в ОС Windows с помощью компилятора GCC, затратила на опрос такой БСД 6.7 секунд (включая время считывания и записи в файл) на процессоре Intel Centrino с тактовой частотой 1.8 ГГц, частотой шины 768 МГц с объемом оперативной памяти 1024 Мб. Заметим, что затраченное время в разы больше времен, необходимых для обсчета большинства сетей, имеющих практическое значение. Например, эта же программа на том же компьютере обрабатывает сеть, приведенную на рисунке http://www.dimonta.com/files/fig11.png, за 0.3 секунды (включая время считывания и записи в файл).

6. Заключение. Рациональное использование БСД, ее быстрая и надежная работа в первую очередь зависят от метода опроса сети и его программной реализации. В настоящей работе детально описан и программно реализован алгоритм опроса БСД, основанный на построении дерева сочленений. Этот алгоритм сочетает в себе три преимущества: точность результатов, малое время работы и универсальность (независимость от вида БСД). Он не труден в понимании, не требует глубокого знания математического аппарата теории вероятностей (как и других областей), ограничиваясь использованием лишь основных тождеств. При соблюдении некоторых несложных правил, учитывающих особенности рационального использования машинных ресурсов, алгоритм также весьма прост для программной реализации.

Разработана программа “junty”, осуществляющая расчет вероятности наступления интересующего события в условиях некоторой модели в виде БСД с учетом наличия некоторого числа экспериментальных фактов (при этом их отсутствие не препятствует выполнению программы — просто данный случай обычно не представляет интереса с практической точки зрения). Валидация этой программы показала ее эффективную и надежную работу. Разработанная программа нацелена на применение в экспертных системах персонифицированной медицины.

Авторы благодарны Д. Втюриной, А. Сулимову, А. Романову, профессору И. Курочкину, И. Упорову, а также всем участникам семинара НИВЦ МГУ “Молекулярное моделирование и разработка биологически активных соединений” за тестирование программы, полезное обсуждение и ценные замечания.

Авторы признательны чл.-корр. РАН С. Д. Варфоломееву за постановку проблемы персонифицированной медицины и поддержку соответствующих работ.

СПИСОК ЛИТЕРАТУРЫ

1. *Mittal A., Kassim A.* Bayesian network technologies: application and graphical models. New York: IGI Publishing, 2007.
2. *Pourret O., Naïm P., Marcot B.* Bayesian networks: a practical guide to applications. New York: Wiley, 2008.
3. *Sebastiani P., Ramoni M.F., Nolan V., Baldwin C.T., Steinberg M.H.* Genetic dissection and prognostic modeling of overt stroke in sickle cell anemia // Nature Genet. 2005. **37**, N 4. 435–440.
4. *Jensen F.V., Nielsen T.D.* Bayesian networks and decision graphs. New York: Springer, 2007.
5. *Dempster A., Laird N., Rubin D.* Maximum likelihood from incomplete data via the EM algorithm // J. of the Royal Statistical Society. 1997. **39**, N 1. 1–38.
6. *Bender J., Koller D., Russel R., Kanazava K.* Adaptive probabilistic networks with hidden variables // Machine Learning. 1997. **29**, N 2–3. 213–244.
7. *Henrion M.* Propagating uncertainty in Bayesian networks by logic sampling // Uncertainty in Artificial Intelligence. Vol 2. J. Lemmer and L. Kanal, Eds. Amsterdam: North-Holland, 1988. 149–163.
8. *Fung R., Chang K.-C.* Weighting and integrating evidence for stochastic simulation in Bayesian networks // Proc. of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89). Amsterdam: North-Holland, 1989. 475–482.
9. *Shachter R., Peot M.* Simulation approaches to general probabilistic inference on belief networks // Proc. of the Fifth Workshop on Uncertainty in Artificial Intelligence (UAI-89). Amsterdam: North-Holland, 1989. 311–318.
10. *Pearl J.* Probabilistic reasoning in intelligent systems. San Mateo: Kaufmann, 1988.
11. *Korb K., Nicholson A.* Bayesian Artificial Intelligence. London: Chapman & Hall/CRC, 2004.
12. *Huang C., Darwiche A.* Inference in belief networks: a procedural guide // Approximate Reasoning. 1996. **15**, N 3. 225–263.
13. *Тулупьев А., Николенко С., Сироткин А.* Байесовские сети: логико-вероятностный подход. СПб.: Наука, 2006.

Поступила в редакцию
25.09.2010