Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных систем и сетей
Кафедра Информатики

К защите допустить:

Заведующий кафедрой ПИКС

_____ И. Н. Цырельчук

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

АЛГОРИТМЫ ПОСТРОЕНИЯ ВЕРОЯТНОСТНЫХ СЕТЕЙ

БГУИР ДП 1-31 03 04 07 093 ПЗ

Студент	Ю. А. Ярошевич
Руководитель	А. А. Волосевич
Консультанты:	
от кафедры информатики	А. А. Волосевич
по экономической части	А. В. Рябов
по охране труда	Е. А. Колосова
Нормоконтролёр	С. И. Сиротко

Рецензент

Минск 2013

РЕФЕРАТ

Ключевые слова: вероятностные модели; байесовы сети; вывод структуры сети по данным; принцип минимальной длинны описания; оценка апостериорной вероятности.

Дипломный проект выполнен на 6 листах формата А1 с пояснительной запиской на 80 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 5 глав, 7 рисунков, 17 таблиц, 65 формул и 31 литературный источник.

Целью дипломного проекта является разработка удобного в использовании инструмента, пригодного для решения практических задач, возникающих в реальных проектах, связанных с вероятностным моделированием.

Для достижения цели дипломного проекта была разработана библиотека кода для Microsoft .NET, предназначенная для представления и обучения структуры вероятностной сети по экспериментальным данным. Библиотека может быть использована в реальных проектах, использующих вероятностный подход к решению проблемы. В библиотеке реализовано несколько алгоритмов, имеющих различные качественные характеристики.

В разделе технико-экономического обоснования был произведён расчёт затрат на создание ПО, а также прибыли от разработки, получаемой разработчиком. Проведённые расчёты показали экономическую целесообразность проекта.

Пояснительная записка включает раздел по охране труда, в котором была произведена оценка пожарной безопасности на предприятии, где частично разрабатывался данный дипломный проект.

АННОТАЦИЯ

на дипломный проект «Алгоритмы построения вероятностных сетей» студента УО «Белорусский государственный университет информатики и радиоэлектроники» Ярошевича Ю. А.

Ключевые слова: вероятностные модели; байесовы сети; вывод структуры сети по данным; принцип минимальной длинны описания; оценка апостериорной вероятности.

Дипломный проект выполнен на 6 листах формата А1 с пояснительной запиской на 80 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 5 глав, 7 рисунков, 17 таблиц, 65 формулы, 31 литературный источник.

Целью дипломного проекта является разработка удобного в использовании инструмента, пригодного для решения практических задач, возникающих в реальных проектах, связанных с вероятностным моделированием.

Для достижения цели дипломного проекта была разработана библиотека кода для Microsoft .NET, предназначенная для представления и обучения структуры вероятностной сети по экспериментальным данным. Библиотека может быть использована в реальных проектах, использующих вероятностный подход к решению проблемы. В библиотеке реализовано несколько алгоритмов, имеющих различные качественные характеристики.

Во введении производится ознакомление с проблемой, решаемой в дипломном проекте.

В первой главе производится обзор предметной области проблемы решаемой в данном дипломном проекте. Приводятся необходимые теоретические сведения, а также производится обзор существующих разработок.

Во второй главе производится краткий обзор технологий, использованных для реализации ПО в рамках дипломного проекта.

В третьей главе производится обзор реализованного ПО. Описываются его составные части и особенности. Приводятся результаты практических испытаний и производится сравнение с существующим ПО.

В четвертой главе производится оценка пожарной безопасности предприятия, на котором частично разрабатывался данный дипломный проект.

В пятой главе производится технико-экономическое обоснование разработки.

В заключении подводятся итоги и делаются выводы по дипломному проекту, а также описывается дальнейший план развития проекта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В век информационных технологий появляются огромные массивы данных, которые можно и нужно уметь обрабатывать с помощью вычислительной техники с целью извлечения знаний. Статистическое моделирование и интеллектуальный анализ данных представляют необходимые инструменты и способы обработки и анализа больших объемов данных. В данном дипломном проекте рассматривается один из способов информационно-статистического моделирования — вероятностные сети, в частности, байесовы сети доверия.

Байесовы сети применяются для решения различных практических задач. Вероятностная природа сетей способствует их успешному применению для создания различных экспертных систем. Одним из первых практических проектов, использующих байесовы сети, стала система медицинской диагностики PathFinder-4 [1]. В прикладном программном обеспечении байесовы сети используются в различных пошаговых мастерах по диагностике неисправностей, исправлению ошибок, консультированию пользователей, например, диагностика неисправности оборудования в Windows [1]. Вероятностные сети применимы также для создания рекомендательных систем, основанных на предпочтениях пользователя и истории его активности.

Важным этапом в применении вероятностных сетей для решения какой-либо задачи является, собственно, её построение, а именно задание структуры сети. Под структурой понимается задание отношений независимости между парами вершин, которые соответствуют случайным величинам из исходной задачи.

Исторически одним из первых способов построения структуры байесовых сетей было привлечение экспертов в конкретной предметной области и разработка архитектуры сети в соответствии с представлением экспертов о решаемой задаче и предметной области. Данный способ имеет ряд очевидных недостатков: необходимость привлечения экспертов; большая трудоемкость процесса построения сети для сложных задач с большим количеством случайных величин и, соответственно, большим количеством узлов; ограниченность модели представлением эксперта о задаче и предметной области.

Появляются новые предметные области и классы задач, в которых довольно сложно найти признанного эксперта. В такой ситуации становится понятным, что привлечение эксперта для разработки байесовой сети не

всегда возможно и расточительно по времени. С другой стороны, сбор экспериментальных данных для решения какой-либо задачи обычно легко доступен. В связи с этим возникает задача обработки этих данных для решения задачи.

В данном дипломном проекте рассматривается задача вывода структуры вероятностной сети по набору экспериментальных данных. С учетом приведённых выше утверждений про потенциальную невозможность привлечения экспертов и доступность экспериментальных данных, умение строить сеть лишь по набору экспериментальных данных становится очень привлекательным. Помимо ускорения процесса построения сети, автоматический вывод структуры имеет ряд дополнительных преимуществ перед «ручным»: становится возможным выявление ранее неизвестных зависимостей между переменными в известных и новых предметных областях; появляется возможность довольно легко обновлять структуру при получении более достоверных экспериментальных данных; создание и применение вероятностных сетей становится более доступным для не-экспертов, и появляется возможность использования вероятностного подхода для решения бóльшего множества прикладных задач.

Не смотря на привлекательность автоматического построения структуры сети, вычислительно эта задача является $\mathcal{NP}$-полной [2]. Многие существующие алгоритмы состоят из двух компонентов: функции для оценки качества сети для имеющихся экспериментальных данных и процедуры поиска структуры сети, оптимизирующую выбранную оценочную функцию. Во многих алгоритмах точное вычисление оценочной функции имеет экспоненциальную сложность по времени, но на практике с помощью различных допущений и оптимизаций её можно аппроксимировать за приемлемое время. Пространство же возможных направленных ациклических графов имеет супер-экспоненциальный порядок роста [3], и полный перебор в таком пространстве возможных решений на практике не возможен для задач с более чем семью наблюдаемыми случайными величинами.

На практике применяют несколько различных способов уменьшения пространства возможных решений, некоторые из них предполагают проведение предварительных вычислений для извлечения первичной информации о взаимоотношениях переменных, другие — требуют априорных знаний об исходном распределении и частичных знаний о зависимостях переменных. Естественно подобные ухищрения в различной степени влияют на качество получаемого результата не в лучшую сторону, но зато позволяют существенно сократить пространство поиска, что в свою очередь позволяет

в разумное время найти структуру сети, которая будет аппроксимировать «истинное» распределение из которого были получены экспериментальные данные.

В данном дипломном проекте реализуются некоторые из известных алгоритмов автоматического вывода структуры сети по данным, дополнительно производятся некоторые оптимизации и улучшения в части их реализации. Также были произведены экспериментальные модификации этих алгоритмов для улучшения качества выводимой сети. В результате получилась библиотека классов для платформы Microsoft .NET, написанная на языках программирования C# и F#, пригодная для решения практических задач в реальных проектах. На данный момент в библиотеке реализована лишь возможность построить структуру сети по данным, но не затронуты очень важные и интересные вопросы, такие как обучение параметров сети и задача статистического вывода суждений в обученной сети.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

В данном разделе будет произведён обзор предметной области задачи, решаемой в рамках дипломного проекта; рассмотрены вопросы о сущности байесовых сетей и принципе их работы; приведена оценка сложности различных проблем, возникающих при применении вероятностных сетей для решения прикладных задач. Также будут рассмотрены принципы работы алгоритмов вывода структуры по данным, реализованных в программном обеспечении разработанном в рамках дипломного проекта, и произведено сравнение с существующим ПО для решения схожих задач.

1.1 Байесовы сети

Байесовы сети являются разновидностью вероятностных графовых моделей (Probabilistic Graphical Models, PGM), представляющих вероятностные и причинно-следственные отношения между переменными в статистическом информационном моделировании [1]. Вероятностные сети являются одним из возможных способов представления совместного распределения множества случайных величин. Данный способ представления распределения является более компактным, чем хранение вероятностей для всех возможных назначений. Здесь и далее под назначением случайных величин $X_1, X_2, \ldots, X_n$ понимаются определенные значения, которые принимают случайные величины, т. е. значения $x_1, x_2, \ldots, x_n$. Табличное представление совместного распределения растет экспоненциально количеству переменных и состояний, которые эти переменные могут принимать. Например, чтобы задать совместное распределение 100 бинарных случайных величин необходимо запомнить $2^{100} - 1$ параметр распределения, что не представляется возможным. Помимо компактного представления функции распределения такие сети кодируют отношения безусловной и условной независимости, что является важным для понимания причинно-следственных отношений между переменными в решаемой задаче. Благодаря информации о независимости, распределение $P(X_1, X_2, \ldots, X_n)$ может быть факторизовано более просто, чем с использованием правила разложения условных вероятностей $P(X_1, X_2, \ldots, X_n) = P(X_1)P(X_2|X_1) \cdots P(X_n|X_1, \ldots, X_{n-1})$. Если имеется некая классическая байесова сеть, значит она кодирует информацию о независимости между переменными. При наличии данной информации совместное распределение случайных величин может быть факторизовано

по формуле:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=0}^{n} P(X_i | X_{\pi_i}), \tag{1.1}$$

где π_i — множество индексов переменных-родителей для переменной X_i.

Данное представление совместного распределения все так же имеет экспоненциальный рост количества стохастических параметров от количества переменных и их состояний. Но на практике, обычно сети имеют небольшую связанность, далекую от полного графа, что позволяет представлять совместное распределение с помощью достижимого количества параметров.

Необходимо отметить, что существует пример успешного коммерческого применения одной из модификаций классической байесовой сети, которая имеет полиномиальный порядок роста количества параметров. К сожалению, работы, в которых данные сети были бы формализованы и математически доказана их корректность, пока не публиковались в открытых источниках. Отличие данной модификации от классических байесовых сетей заключается в том, что таблицы условных распределений $P(X_k|X_{\pi_k})$, где k — индекс случайной величины, ассоциированные с вершинами графа и имеющими размерность $\alpha_k \cdot \prod_{j \in \pi_k} \alpha_j$, заменяются на n таблиц меньшего размера, представляющих условные распределения $P(X_k|X_j)$, где $j \in \pi_k$, $k = 1, \ldots, n$, ассоциированных с дугами графа и имеющими общую размерность $\alpha_k \cdot \sum_{j \in \pi_k} \alpha_j$, где α_j — количество значений, которые может принимать случайная величина X_j. Также в данной модификации вершины графа содержат таблицы безусловного распределения $P(X_k)$. В связи с указанными изменениями в данной модификации используются несколько другие алгоритмы вывода статистических суждений. Разработанная в рамках дипломного проекта библиотека кода предназначена для работы именно с такой модификацией байесовых сетей, т. к. различных инструментов для работы с классическими байесовыми сетями создано достаточно. Несмотря на это, было сделано предположение, что структура классической байесовой сети и описанной выше модификации будет совпадать, и алгоритмы построения, применимые для нахождения структуры по данным для классической байесовой сети, подойдут для указанной модификации. Следовательно, сама библиотека может быть использована и для вывода структуры классических сетей, что будет неоднократно использовано далее для оценки качества структуры сети, полученной по экспериментальным

данным.

Таким образом можно выделить следующие понятия и компоненты, из которых состоят вероятностные сети [1]:

– множество случайных переменных и направленных связей между переменными;

– каждая переменная может принимать одно из конечного множества взаимоисключающих значений;

– переменные вместе со связями образуют ориентированный граф без циклов;

– каждой переменной-потомку X_k с переменными-предками π_k приписывается таблица условных вероятностей $P(X_k|X_{\pi_k})$, либо, для вышеупомянутой модификации — каждой дуге между переменными X и Y сопоставляется таблица условного распределения $P(X|Y)$, каждой вершине — таблица безусловного распределения $P(X)$.

1.2 Построение вероятностной сети

Одним из первых этапов применения байесовых сетей для решения практической задачи, после формулирования проблемы в терминах вероятностей и выделения целевых переменных, является этап описания отношений «причина-следствие» между переменными в виде ориентированных ребер графа [1]. Задание подобных отношений определяет структуру графа вероятностной сети.

От правильности выбора структуры сети зависят многие важные качественные показатели сети, такие как количество необходимых параметров, сложность вывода статистических суждений, возможность обоснования поведения сети при изменении назначений некоторых из переменных.

Исторически одним из первых способов создания байесовых сетей было привлечение экспертов в предметной области решаемой проблемы. Построенная сеть отражала субъективное представление экспертов о проблеме и потенциально могла отличаться от истины. Параметры условных распределений в вершинах графа отражали байесовы вероятности по мнению экспертов. Здесь и далее под байесовой вероятностью понимается степень уверенности в истинности суждения определенного индивидуума, в отличие от частотной вероятности, которая определяется как относительная частота возникновения события в большом числе испытаний.

Далее будет рассмотрен пример ручного построения классической

байесовой сети по имеющимся априорным данным о проблеме.

1.3 «Ручное» построение структуры на основе экспертных знаний

Рассматриваемый здесь пример был взят из домашнего задания, предлагаемого в онлайн курсе Стэндфордского университета по вероятностным графовым моделям [4]. Ниже приводится один из возможных вариантов его решения.

Необходимо разработать модель для прогнозирования своевременности выплат клиентом задолженности банку по кредиту и другим займовым операциям, т. е. модель для оценки кредитоспособности клиента банка. Банк имеет доступ к некоторой информации о клиенте, такой как его *доходы* (Income), *история платежей* (PaymentHistory), накопленные *богатства* (Assets), *возраст* клиента (Age), а также к *соотношению долгов к доходам* (DebtIncomeRatio). Банковский эксперт полагает, что целевая переменная — *кредитоспособность* (CreditWorthiness) — в конечном итоге зависит от *надежности* (Reliability) клиента, его прогнозируемых *будущих доходов* (FutureIncome) и *соотношения долгов к доходам*. По известным на данный момент данным можно построить скелет будущей вероятностной сети — её структуру. Для полноты картины создания сети вручную пример продолжается и добавляется дополнительная информация, имеющаяся у банка, которая поможет задать вероятности в таблицах условного распределения. Необходимо отметить, что указанные вероятности будут байесовыми, т. к. отражают субъективное представление о поведении модели:

– чем лучше история платежей клиента, тем с большей вероятностью он надежен;

– чем старше клиент, тем с большей вероятностью он надежен;

– у старших клиентов с большей вероятностью будет отличная история платежей;

– клиенты с высоким соотношением долгов к доходам с большей вероятностью имеют финансовые трудности, следовательно с меньшей вероятностью имеют хорошую историю платежей;

– чем выше доход человека, тем больше вероятность что он имеет много накопленных богатств;

– чем больше накопленных богатств и выше доходы клиента, тем лучше прогнозируемые будущие доходы;

– при прочих равных, надежные люди с большей вероятностью кредитоспособны, чем ненадежные. Также люди с более высокими прогнозируемыми доходами или с низким соотношением долгов к доходам более кредитоспособны и наоборот.

С учетом всех приведенных выше дополнительных наблюдений, имеющихся у банка, можно сконструировать сеть и задать байесовы вероятности в таблицах распределения. Одна из возможных структур сети приведена на рисунке 1.1. Перевод условных обозначений, используемых на рисунке, на русский язык приведен ранее в тексте данного подраздела. Таблицы с условными и безусловными вероятностями здесь не приводятся для экономии места.

Таким образом, чтобы построить классическую байесову сеть понадобилось ознакомиться с предметной областью и проконсультироваться с экспертом. Также необходимо было учесть все наблюдения, сделанные экспертом, и учесть его опыт и представление о предметной области. В итоге была построена модель, которая соответствует представлению о проблеме эксперта, но на самом деле может отличаться от истинной модели проблемы.

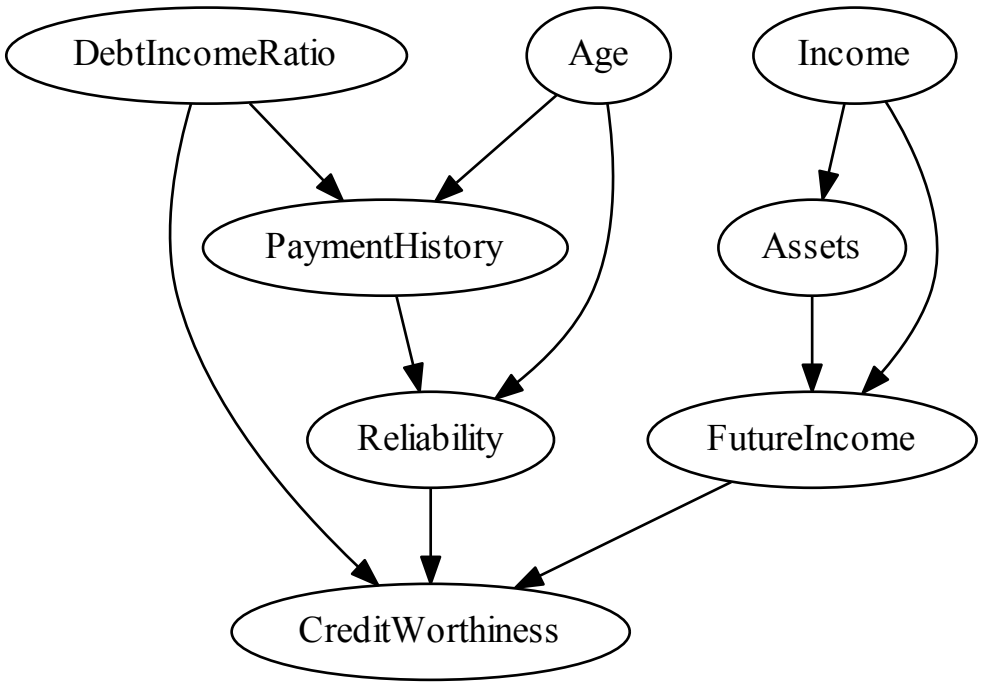


Рисунок 1.1 – Байесова сеть для оценки кредитоспособности.

1.4 Сложность нахождения структуры по данным

В предыдущем подразделе был приведен пример построения классической байесовой сети «вручную» с использованием экспертных знаний. Для большинства возникающих задач можно собрать большой объём экспериментальных данных без каких-либо усилий. В данной ситуации возникает интересная задача — как построить вероятностную сеть используя лишь эти данные, прибегнув к минимуму экспертных знаний? Данный вопрос затрагивался во многих научных работах [5–8] и имеет различные подходы к решению, но сама задача является $\mathcal{NP}$-полной [2] и требует применения различных ухищрений и эвристик для автоматического построения структуры сети на практике.

Сложность задачи обоснована тем, что количество всевозможных назначений растет экспоненциально количеству переменных и их состояний. Различные алгоритмы построения сетей используют различные оценочные метрики качества сети. Эти метрики, обычно, включают себя подсчёт маргинализованных распределений некоторого подмножества случайных величин. Вычисление данных распределений подразумевает подсчет суммы вероятностей экспоненциального числа назначений. На практике во избежание большого количества вычислений используются различные приближенные методы, зависящие от метрики. Даже при наличии возможности вычислить функцию для оценки качества сети быстро, все равно остаётся проблема супер-экспоненциального роста количества моделей в зависимости от числа переменных. Робинсон в работе [3] предложил следующее рекуррентное соотношение для подсчета числа ацикличных моделей:

$$r(n) = \begin{cases} 1 & , n = 0 \\ \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} 2^{i(n-i)} r(n-i) & , n > 0 \end{cases} \tag{1.2}$$

Для убедительности приведем в таблице 1.1 некоторые числа, касающиеся количества возможных моделей для сети из n переменных [8]. Как видно из таблицы, для более семи вершин полный перебор выполнить проблематично.

Таблица 1.1 – Зависимость числа моделей без циклов от количества вершин, которые нужно проанализировать при полном переборе моделей

Число вершин	Моделей без циклов	Число вершин	Моделей без циклов
1	1	6	3 781 503
2	3	7	1 138 779 265
3	25	8	783 702 329 343
4	543	9	1 213 442 454 842 881
5	29 281	10	4 175 098 976 430 598 100

1.5 Принцип минимальной длинны описания

В данном подразделе рассматривается принцип минимальной длинны описания[1)] (МДО) и его применимость для задания функции оценки качества обучаемой сети. Данный принцип позволяет среди множества моделей выбрать модель с оптимальным соотношением сложности и соответствием модели наблюдаемым данным. Т. е. данный принцип позволяет выбрать несложную и «полезную» модель, устойчивую к проблеме переобучения[2)]. Принцип МДО в своей нестрогой и наиболее общей формулировке гласит: среди множества моделей следует выбрать ту, которая позволяет описать данные наиболее коротко, без потери информации [9]. В контексте поиска модели байесовой сети, соответствующей экспериментальным данным, принцип МДО гласит, что нужно выбрать модель, которая минимизирует сумму длин кодирования самой модели и кодирования экспериментальных данных с помощью этой модели [5], что выражается формулой:

$$l(x^R[n]) = \min_{g \in G} \left[l_G(g) + l_g(x^R[n]) \right] , \tag{1.3}$$

[1)] В англоязычной литературе используется термин minimum description length или сокращенно MDL.

[2)] В англоязычной литературе данная проблема называется overfitting и подразумевает, что модель слишком хорошо объясняет данные на которых она обучалась, но из-за этого непригодна для прогнозирования — работе на данных ранее не известных.

где x^R — вектор размерностью R, содержащий значения переменных (аттрибутов). Представлен как $x^R = (x^{(1)}, x^{(2)}, \ldots, x^{(R)})$, где атрибут $x^{(j)}$ может принимать α_j значений, $j = 1, \ldots, R$.

n — количество случаев в экспериментальных данных;

$x^R[n]$ — набор экспериментальны данных;

G — множество моделей;

$l_G(g)$ — длина описания модели;

$l_g(x^R[n])$ — длина представления данных $x^R[n]$ моделью $g \in G$.

Для вычисления длинны кодирования модели и длинны кодирования данных с использованием модели в реализации дипломного проекта использовались результаты, приведенные в работах [6, 8]. Собственно модель вероятностной сети состоит из таблиц условных и безусловных распределений и отношений «родитель-потомок» между вершинами. Для вычисления длины кодирования модели можно воспользоваться формулой (1.4):

$$l_G(g) = \frac{\log n}{2} \cdot \sum_{k=1}^{R} S_k(g)(\alpha_k - 1)\,, \tag{1.4}$$

где $S_k(g)$ — количество возможных назначений переменных-родителей переменной X_k, способ вычисления данного значения отличается у классических сетей и модификации упомянутой в разделе 1.1 на странице 10.

Значение функции $S_k(g)$ для классических байесовых сетей вычисляется по формуле (1.5), для модификации упомянутой в разделе 1.1 на странице 10 — по формуле (1.6):

$$S_k(g) = \prod_{j \in \pi_k} \alpha_j\,, \tag{1.5}$$

$$S_k(g) = \sum_{j \in \pi_k} \alpha_j\,. \tag{1.6}$$

Длинна представления данных $l_g(x^R[n])$ вычисляется как эмпирическая энтропия n экспериментальных наблюдений $x^R[n]$ при заданной модели g. Эмпирическая энтропия может быть вычислена по

формуле (1.7):

$$\mathcal{H}(x^R[n]|g) = \sum_{k=1}^{R} \sum_{s=1}^{S_k(g)} \sum_{q=0}^{\alpha_{k+1}-1} -n[q,s,k,g] \log \frac{n[q,s,k,g]}{n[s,k,g]}, \qquad (1.7)$$

где $n[s,k,g]$ — количество случаев в экспериментальных данных $x^R[n]$ в которых переменные-родители переменной X_k принимают назначение s;

$n[q,s,k,g]$ — количество случаев в экспериментальных данных $x^R[n]$ в которых переменные-родители переменной X_k принимают назначение s, а переменная X_k принимает назначение q.

Таким образом, приведенные выше формулы, могут быть использованы для оценки качества структуры сети и её сложности. Имея данную оценку можно использовать различные стратегии поиска в пространстве возможных моделей, минимизируя целевую функцию (1.3). Более подробно об используемых стратегиях поиска говорится в разделе 3. За доказательствами и более подробными сведениями о применении принципа МДО в построении байесовых сетей можно обратиться к работам [5, 6, 8, 9].

1.6 Оценка структуры сети на основе апостериорной вероятности

Существуют подходы, использующие байесов метод для оценки качества полученной структуры и алгоритмы на их базе пытаются максимизировать апостериорную вероятность структуры для данного набора экспериментальны данных. Один из возможных подходов к оценке качества двух структур приведет в работе [7]. В программном обеспечении, разработанном в данном дипломном проекте, использовался критерий оценки качества структуры, приведенный в упомянутой выше работе.

Введем некоторые обозначения, в дополнение к тем, которые были введены в подразделе 1.5. Пусть структура вероятностной сети обозначается символом B_S, таблицы условных распределений, ассоциированные с сетью, — B_P. Две вероятностные сети для данного набора экспериментальных данных можно оценить по отношению (1.8)

апостериорных вероятностей:

$$\frac{P(B_{S_i}|x^R[n])}{P(B_{S_j}|x^R[n])} = \frac{\frac{P(B_{S_i},x^R[n])}{P(x^R[n])}}{\frac{P(B_{S_j},x^R[n])}{P(x^R[n])}} = \frac{P(B_{S_i},x^R[n])}{P(B_{S_j},x^R[n])}\,. \tag{1.8}$$

Как видно из приведенной формулы (1.8), научившись вычислять отношение совместных распределений, можно сравнивать апостериорные вероятности структур. Т. к. в разработанном ПО использовались результаты, приведенные в работе [7], то считаем целесообразным привести в данном подразделе базовые формулы и предположения из вышеупомянутой работы.

Для вычисления $P(B_S, D)$ важно сделать несколько важных предположений:

– Экспериментальные данные содержат только дискретные случайные величины и все эти случайные величины присутствуют в истинной структуре B_S модели из которой были получены эти экспериментальные данные. Из данного предположения следует формула (1.9):

$$\int_{B_P} P(x^R[n]|B_S, B_P) f(B_P|B_S) P(B_S) dB_p\,, \tag{1.9}$$

где B_P — вектор, содержащий значения условных вероятностей для назначений переменных из структуры B_S;

f — условная плотность распределения B_P при условии структуры B_S.

– Случаи, зафиксированные в экспериментальных данных, независимы друг от друга, при условии зафиксированной модели, т. е. данное предположение подразумевает, что модель, генерирующая экспериментальные данные не меняется. Это предположение позволяет упростить формулу (1.9) и привести её к виду:

$$P(B_S, x^R[n]) = P(B_S) \int_{B_P} \left[\prod_{j=1}^{n} P(x_j^R|B_S, B_P)\right] f(B_P|B_S) dB_P\,. \tag{1.10}$$

– Экспериментальные данные не должны содержать пропущенных значений для переменных из структуры B_S. Введем дополнительные обозначения. Пусть $x_j^{(i)}$ представляет значение i-й переменной в j-м случае. Пусть ϕ_i представляет из себя вектор уникальных назначений переменных-родителей для i-й переменной, т. е. вектор уникальных назначений для $\forall X_k,\ k \in \pi_i$. Пусть $\sigma(i,j)$ индексная функция, которая возвращает

индекс назначения π_i в j-ом случае из вектора ϕ_i. Введем обозначение для длинны вектора $q_i = |\phi_i|$. Теперь с учетом предположения об отсутствии пропущенных значений можно вычислить вероятность конкретного случая из экспериментальных данных по формуле:

$$P(x_j^R|B_S, B_P) = \prod_{i=1}^{R} = P(X_i = x_j^{(i)}|\phi_i[\sigma(i,j)], B_P)\,. \tag{1.11}$$

Подставляя выражение (1.11) в формулу (1.10) получим:

$$P(B_S, x^R[n]) = P(B_S) \int_{B_P} \left[\prod_{j=1}^{n} \prod_{i=1}^{R} P(X_i = x_j^{(i)}|\phi_i[\sigma(i,j)], B_P) \right] \times \\ \times f(B_P|B_S) dB_P\,. \tag{1.12}$$

Пусть для выбранных i и j $f(P(x_i|\phi_i[j], B_P))$ обозначает плотность распределения возможных значений $P(x_i|\phi_i[j], B_P)$. Необходимо сделать еще одно предположение.

– Для $1 \leq i, i' \leq n$, $1 \leq j \leq q_i$, $1 \leq j' \leq q_i$, если $ij \neq i'j'$, то распределение $f(P(x_i|\phi_i[j]))$ не зависимо от распределения $f(P(x_{i'}|\phi_{i'}[j']))$. Данное предположение по свой сути полагает, что до того, как были получены экспериментальные данные, все возможные назначения равновероятны.

С учетом приведенных выше предположений и теоремы, приведённой и доказанной в работе [7], можно привести формулу для вычисления $P(B_S, D)$:

$$P(B_S, x^R[n]) = P(B_S) \prod_{i=1}^{R} \prod_{j=1}^{q_i} \frac{(\alpha_i - 1)!}{(n[\phi_i[j], i, B_S] + \alpha_i - 1)!} \prod_{k=1}^{\alpha_i} n[v_{ik}, \phi_i[j], i, B_S]!\,, \tag{1.13}$$

где v_{ik} — k-е возможное назначение переменной X_i.

Формула (1.13) позволяет вычислить значение $P(B_S, x^R[n])$, если известна вероятность $P(B_S)$ и подсчитана оставшаяся часть формулы на основе экспериментальны данных $x^R[n]$. Но из-за того, что $P(B_S)$ является чаще неизвестной величиной, чем известной, предполагают, что все возможные структуры сети равновероятны и $P(B_S)$ является некой малой константой. Апостериорную вероятность структуры, при условии данных

можно вычислить по формуле:

$$P(B_S|x^R[n]) = \frac{P(B_S, x^R[n])}{\sum_{B_S} P(B_S, x^R[n])}\,. \tag{1.14}$$

Но, как уже упоминалось, множество возможных структур слишком велико, и на практике значение апостериорной вероятности можно вычислить лишь для малых сетей либо приблизительно. В разделе 3 будет более подробно рассмотрена модификация оценки апостериорной вероятности (1.13), применимая на практике.

1.7 Общая структура алгоритмов

В предыдущих двух подразделах были рассмотрены два подхода к оценке качества структуры сети при известных экспериментальных данных. Как уже упоминалось ранее, выбор функции оценки сети является лишь одним из компонентов большинства алгоритмов построения структуры сети по данным. Важной частью алгоритма является также выбор стратегии поиска в пространстве возможных структур. От выбора стратегии поиска зависит трудоемкость алгоритма, качество найденной сети и, собственно, её структура. В ПО разработанном в рамках дипломного проекта реализовано несколько различных алгоритмов нахождения структуры на основе оценки апостериорной вероятности из подраздела 1.6 и оценки на основе принципа МДО из подраздела 1.5. Стоит упомянуть, что в разработанном ПО реализованы разные стратегии поиска для разных оценок. Так алгоритмы использующие оценку МДО могут находить сети произвольной структуры без каких-либо априорных знаний о предметной области. С другой стороны, для уменьшения пространства возможных решений, алгоритм, использующий оценку апостериорной вероятности, требует априорных знаний об упорядоченности переменных. В данном конкретном случае, переменные должны быть упорядочены так, что возможные переменные-родители данной переменной находятся в упорядоченном списке раньше самой переменной. Во многих других работах накладываются дополнительные ограничения на структуру выводимой сети. Например, в работе [6] используется оценка МДО, но класс находимых структур ограничен деревьями. В работе [10] приводится алгоритм, ограниченный классом полидеревьев.

1.8 Обзор существующих программ

Существует множество решений для работы с классическими байесовыми сетями. Ниже приведены некоторые из задач, которые может выполнять типичный редактор вероятностных сетей:

– ручное, автоматическое, полу-автоматическое построение структуры сети;

– оценка параметров условных распределений по данным;

– различные алгоритмы статистического вывода суждений в сетях;

– генерация экспериментальных данных по готовой сети;

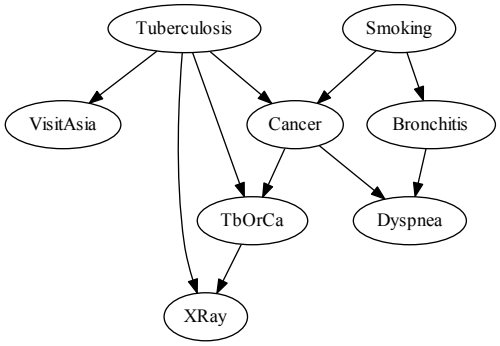
Приняв во внимание тему дипломного проекта, наибольший интерес в существуем ПО будет представлять функциональность автоматического вывода структуры сети по данным. Ниже рассматриваются некоторые из программ для работы с байесовыми сетями.

1.8.1 SAMIAM (`http://reasoning.cs.ucla.edu/samiam`) Бесплатный кросс-платформенный редактор байесовых сетей, написанный на Java. Имеет богатую функциональность по ручному созданию сетей. Поддерживает множество различных алгоритмов вывода статистических суждений и позволяет делать различные типы запросов к сети. Позволяет генерировать экспериментальные данные, соответствующие готовой сети. Но функциональность, обеспечивающая автоматический вывод структуры сети по данным, отсутствует.

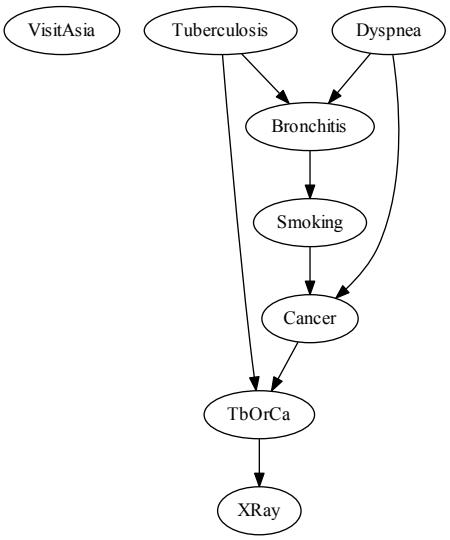
1.8.2 Netica (`http://www.norsys.com/netica.html`) Коммерческое программное обеспечение. Версия программы с ограниченной функциональностью свободно доступна на сайте фирмы Norsys. Netica — мощная, удобная в работе программа для работы с графовыми вероятностными моделями. Она имеет интуитивный и приятный интерфейс пользователя для просмотра и редактирования топологии сети. Соотношения между переменными могут быть заданы, как индивидуальные вероятности, в форме уравнений, или путем автоматического обучения из файлов данных (которые могут содержать пропуски), но, к сожалению, в бесплатном варианте программы данная функциональность недоступна и возможности сравнить с разработанной в дипломном проекте реализацией нет. Созданные сети могут быть использованы независимо, и как фрагменты более крупных моделей, формируя тем самым библиотеку модулей [1].

1.8.3 GeNIe (http://genie.sis.pitt.edu/) Бесплатное кросс-платформенное приложение и библиотека для работы с вероятностными графовыми моделями, написанная на C++. Предоставляется функциональность ручного и автоматического создания сетей, а также реализуются алгоритмы вывода статистических суждений. Программа реализует несколько алгоритмов вывода структуры сети по данным, целесообразно привести здесь одно небольшое сравнение с разработанным ПО, т. к. функциональности приложений пересекаются. Более подробное сравнение разработанного ПО с существующим будет произведено в разделе 3. Уместно рассмотреть хорошо изученную сеть Asia[1)]. Экспериментальные данные, содержащие 1000 случаев и имеющие распределение задаваемое сетью, были сгенерированы с помощью одной из вышеперечисленных программ. К этим экспериментальным данным были применены три различных алгоритма вывода структуры, реализованные в GeNIe. Выведенные структуры приведены на рисунке 1.2. Для сравнения, структуры, выведенные одним из алгоритмов из разработанной библиотеки на том же наборе данных, и истинная сеть Asia приведены на рисунке 1.3. Как видно, алгоритм из разработанной библиотеки не нашел одну связь, в отличие от алгоритмов реализованных в GeNIe, которые допустили больше ошибок в структуре.

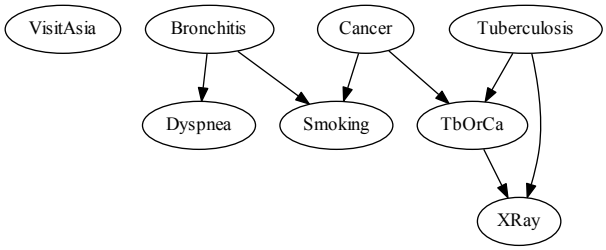
[1)] http://www.bnlearn.com/bnrepository/#asia



(а)

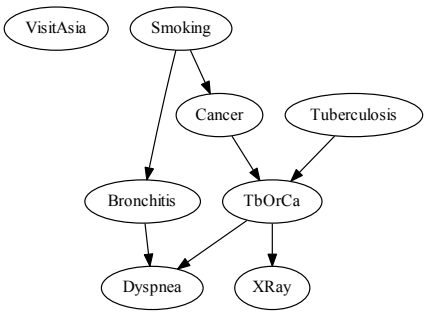


(б)

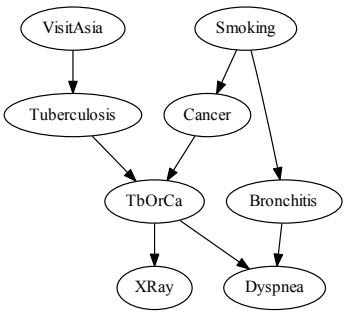


(в)

Рисунок 1.2 – Структуры, выведенные по данным с помощью GeNIe: а — алгоритм Bayesian Search; б — алгоритм K2; в — алгоритм PC;



(а)



(б)

Рисунок 1.3 – Выведенная из данных и оригинальная сеть Asia: а — алгоритм из разработанной библиотеки, использующий оценку МДО; б — истинная сеть Asia

2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

Выбор технологий является важным предварительным этапом разработки сложных информационных систем. Платформа и язык программирования, на котором будет реализована система, заслуживает большого внимания, так как исследования показали, что выбор языка программирования влияет на производительность труда программистов и качество создаваемого ими кода [11, с. 59].

Ниже перечислены некоторые факторы, повлиявшие на выбор технологий:

– Разрабатываемое ПО должно работать на операционной системе Windows 7 и более новых версиях системы.

– Среди различных платформ разработки имеющийся программист лучше всего знаком с разработкой на платформе Microsoft .NET.

– Дальнейшей поддержкой проекта, возможно, будут заниматься разработчики, не принимавшие участие в выпуске первой версии.

– Имеющийся разработчик имеет опыт работы с объекто-ориентированными и с функциональными языками программирования.

Основываясь на опыте работы имеющихся программистов разрабатывать ПО целесообразно на платформе Microsoft .NET. Приняв во внимание необходимость обеспечения доступности дальнейшей поддержки ПО, возможно, другой командой программистов, целесообразно не использовать малоизвестные и сложные языки программирования. С учетом этого фактора выбор языков программирования сужается до четырех официально поддерживаемых Microsoft и имеющих изначальную поддержку в Visual Studio 2012: Visual C++/CLI, C#, Visual Basic .NET и F#. Необходимость использования низкоуровневых возможностей Visual C++/CLI в разрабатываемом ПО отсутствует, следовательно данный язык можно исключить из списка кандидатов. Visual Basic .NET уступает по удобству использования двум другим кандидатам из нашего списка. Оставшиеся два языка программирования C# и F# являются первостепенным, элегантными, мультипарадигменными языками программирования для платформы Microsoft .NET. Таким образом, с учетом вышеперечисленных факторов, целесообразно остановить выбор на следующих технологиях:

– операционная система Windows 7;
– платформа разработки Microsoft .NET;
– языки программирования C# и F#.

Для реализации поставленной задачи нет необходимости в использовании

каких-либо прикладных библиотек для создания настольных или веб-приложений, достаточно использовать стандартные библиотеки указанных выше языков. Поддержка платформой Microsoft .NET различных языков программирования позволяет использовать язык, который наиболее просто и «красиво» позволяет решить возникающую задачу. Разрабатываемое программное обеспечение в некоторой степени использует данное преимущество платформы. Язык C# больше подходит для создания высокоуровнего дизайна проложения (иерархия классов и интерфейсов, организация пространств имен и публичного программного интерфейса), язык F# — для реализации логики приложения, функций и методов [12], прототипирования различных идей. В разрабатываемом программном продукте C# используется для предоставления удобного программного интерфейса, F# — для прототипирования и реализации вычислительной логики. Далее проводится характеристика используемых технологий и языков программирования более подробно.

2.1 Программная платформа Microsoft .NET

Программная платформа Microsoft .NET является одной из реализаций стандарта ECMA-335 [13] и является современным инструментом создания клиентских и серверных приложений для операционной системы Windows. Первая общедоступная версия .NET Framework вышла в феврале 2002 года. С тех пор платформа активно эволюционировала и на данный момент было выпущено шесть версии данного продукта. На данный момент номер последней версии .NET Framework — 4.5. Платформа Microsoft .NET была призвана решить некоторые наболевшие проблемы, скопившиеся на момент её выхода, в средствах разработки приложений под Windows. Ниже перечислены некоторые из них [14, с. XIV – XVII]:

– сложность создания надежных приложений;

– сложность развертывания и управления версиями приложений и библиотек;

– сложность создания переносимого ПО;

– отсутствие единой целевой платформы для создателей компиляторов;

– проблемы с безопасным исполнением непроверенного кода;

– великое множество различных технологий и языков программирования, которые не совместимы между собой.

Многие из этих проблем были решены. Далее более подробно

рассматривается внутреннее устройство Microsoft .NET.

Основными составляющими компонентами Microsoft .NET являются общая языковая исполняющая среда (Common Language Runtime) и стандартная библиотека классов (Framework Class Library). CLR представляет из себя виртуальную машину и набор сервисов обслуживающих исполнение программ написанных для Microsoft .NET. Ниже приводится перечень задач, возлагаемых на CLR [15]:

– загрузка и исполнение управляемого кода;
– управление памятью при размещении объектов;
– изоляция памяти приложений;
– проверка безопасности кода;
– преобразование промежуточного языка в машинный код;
– доступ к расширенной информации от типах — метаданным;
– обработка исключений, включая межъязыковые исключения;
– взаимодействие между управляемым и неуправляемым кодом (в том числе и COM-объектами);
– поддержка сервисов для разработки (профилирование, отладка и т. д.).

Программы написанные для Microsoft .NET представляют из себя набор типов взаимодействующих между собой. Microsoft .NET имеет общую систему типов (Common Type System, CTS). Данная спецификация описывает определения и поведение типов создаваемых для Microsoft .NET [16]. В частности в данной спецификации описаны возможные члены типов, механизмы сокрытия реализации, правила наследования, типы-значения и ссылочные типы, особенности параметрического полиморфизма и другие возможности предоставляемые CLI. Общая языковая спецификация (Common Language Specification, CLS) — подмножество общей системы типов. Это набор конструкций и ограничений, которые являются руководством для создателей библиотек и компиляторов в среде .NET Framework. Библиотеки, построенные в соответствии с CLS, могут быть использованы из любого языка программирования, поддерживающего CLS. Языки, соответствующие CLS (к их числу относятся языки C#, Visual Basic .NET, Visual C++/CLI), могут интегрироваться друг с другом. CLS — это основа межъязыкового взаимодействия в рамках платформы Microsoft .NET [15].

Некоторые из возможностей, предоставляемых Microsoft .NET: верификация кода, расширенная информация о типах во время исполнения, сборка мусора, безопасность типов, — невозможны без наличия подробных

метаданных о типах из которых состоит исполняемая программа. Подробные метаданные о типах генерируются компиляторами и сохраняются в результирующих сборках. Сборка — это логическая группировка одного или нескольких управляемых модулей или файлов ресурсов, является минимальной единицей с точки зрения повторного использования, безопасности и управлениями версиями [16, с. 6].

Одной из особенностей Microsoft .NET, обеспечивающей переносимость программ без необходимости повторной компиляции, является представление исполняемого кода приложений на общем промежуточном языке (Common Intermediate Language, CIL). Промежуточный язык является бестиповым, стековым, объекто-ориентированным ассемблером [16, с. 16 – 17]. Данный язык очень удобен в качестве целевого языка для создателей компиляторов и средств автоматической проверки кода для платформы Microsoft .NET, также язык довольно удобен для чтения людьми. Наличие промежуточного языка и необходимость создания производительных программ подразумевают наличие преобразования промежуточного кода в машинный код во время исполнения программы. Одним из компонентов общей языковой исполняющей среды, выполняющим данное преобразование, является компилятор времени исполнения (Just-in-time compiler) транслирующий промежуточный язык в машинные инструкции, специфические для архитектуры компьютера на котором исполняется программа.

Ручное управление памятью всегда являлось очень кропотливой и подверженной ошибкам работой. Ошибки в управлении памятью являются одними из наиболее сложных в устранении типами программных ошибок, также эти ошибки обычно приводят к непредсказуемому поведению программы, поэтому в Microsoft .NET управление памятью происходит автоматически [16, с. 505 – 506]. Автоматическое управление памятью является механизмом поддержания иллюзии бесконечности памяти. Когда объект данных перестает быть нужным, занятая под него память автоматически освобождается и используется для построения новых объектов данных. Имеются различные методы реализации такого автоматического распределения памяти [17, с. 489]. В Microsoft .NET для автоматического управления памятью используется механизм сборки мусора (garbage collection). Существуют различные алгоритмы сборки мусора со своими достоинствами и недостатками. В Microsoft .NET используется алгоритм пометок (mark and sweep) в сочетании с различными оптимизациями, такими как, например, разбиение всех объектов по

поколениям и использование различных куч для больших и малых объектов.

Ниже перечислены, без приведения подробностей, некоторые важные функции исполняемые общей языковой исполняющей средой:

– обеспечение многопоточного исполнения программы;
– поддержание модели памяти, принятой в CLR;
– поддержка двоичной сериализации;
– управление вводом и выводом;
– структурная обработка исключений;
– возможность размещения исполняющей среды внутри других процессов.

Как уже упоминалось выше, большую ценностью для Microsoft .NET представляет библиотека стандартных классов — соответствующая CLS-спецификации объектно-ориентированная библиотека классов, интерфейсов и системы типов (типов-значений), которые включаются в состав платформы Microsoft .NET. Эта библиотека обеспечивает доступ к функциональным возможностям системы и предназначена служить основой при разработке .NET-приложений, компонент, элементов управления [15].

2.2 Язык программирования C#

C# — объектно-ориентированный, типо-безопасный язык программирования общего назначения. Язык создавался с целью повысить продуктивность программистов. Для достижения этой цели в языке гармонично сочетаются простота, выразительность и производительность промежуточного кода, получаемого после компиляции. Главным архитектором и идеологом языка с первой версии является Андрес Хейлсберг (создатель Turbo Pascal и архитектор Delphi). Язык C# является платформенно нейтральным, но создавался для хорошей работы с Microsoft .NET [18]. Этот язык сочетает простой синтаксис, похожий на синтаксис языков C++ и Java, и полную поддержку всех современных объектно-ориентированных концепций и подходов. В качестве ориентира при разработке языка было выбрано безопасное программирование, нацеленное на создание надежного и простого в сопровождении кода [19].

Язык имеет богатую поддержку парадигмы объекто-ориентированного программирования, включающую поддержку инкапсуляции, наследования и полиморфизма. Отличительными чертами C# с точки зрения ОО парадигмы являются:

– Унифицированная система типов. В C# сущность, содержащая

данные и методы их обработки, называется типом. В C# все типы, являются ли они пользовательскими типами, или примитивами, такими как число, производны от одного базового класса.

– Классы и интерфейсы. В классической объекто-ориентированной парадигме существуют только классы. В C# дополнительно существуют и другие типы, например, интерфейсы. Интерфейс — это сущность напоминающая классы, но содержащая только определения членов. Конкретная реализация указанных членов интерфейса происходит в типах, реализующих данный интерфейс. В частности интерфейсы могут быть использованы при необходимости проведения множественного наследования (в отличие от языков C++ и Eiffel, C# не поддерживает множественное наследование классов).

– Свойства, методы и события. В чистой объекто-ориентированной парадигме все функции являются методами. В C# методы являются лишь одной из возможных разновидностей членов типа, в C# типы также могут содержать свойства, события и другие члены. Свойство — это такая разновидность функций, которая инкапсулирует часть состояния объекта. Событие — это разновидность функций, которые реагируют на изменение состояния объекта [18].

В большинстве случаев C# обеспечивает безопасность типов в том смысле, что компилятор контролирует чтобы взаимодействие с экземпляром типа происходило согласно контракту, который он определяют. Например, компилятор C# не скомпилирует код который обращается со строками, как если бы они были целыми числами. Говоря более точно, C# поддерживает статическую типизацию, в том смысле что большинство ошибок типов обнаруживаются на стадии компиляции. За соблюдение более строгих правил безопасности типов следит исполняющая среда. Статическая типизация позволяет избавиться от широкого круга ошибок, возникающих из-за ошибок типов. Она делает написание и изменение программ более предсказуемыми и надежными, кроме того, статическая типизация позволяет существовать таким средствам как автоматическое дополнение кода и его предсказуемый статический анализ. Еще одним аспектом типизации в C# является её строгость. Строгая типизация означает, что правила типизации в языке очень «сильные». Например, язык не позволяет совершать вызов метода, принимающего целые числа, передавая в него вещественное число [18]. Такие требования спасают от некоторых ошибок.

C# полагается на автоматическое управление памятью со стороны исполняющей среды, предоставляя совсем немного средств для управления

жизненным циклом объектов. Не смотря на это, в языке все же присутствует поддержка работы с указателями. Данная возможность предусмотрена для случаев, когда критически важна производительность приложения или необходимо обеспечить взаимодействие с неуправляемым кодом [18].

Как уже упоминалось C# не является платформенно зависимым языком. Благодаря усилиям компании Xamarin возможно писать программы на языке C# не только для операционных систем Microsoft, но и ряда других ОС. Существуют инструменты создания приложений на C# для серверных и мобильных платформ, например: iOS, Android, Linux и других.

Создатели языка C# не являются противниками привнесения в язык новых идей и возможностей, в отличии от создателей одного из конкурирующих языков. Каждая новая версия компилятора языка привносит различные полезные возможности, которые отчаются требованиям индустрии. Далее приводится краткий обзор развития языка.

Первая версия C# была похожа по своим возможностям на Java 1.4, несколько их расширяя: так, в C# имелись свойства (выглядящие в коде как поля объекта, но на деле вызывающие при обращении к ним методы класса), индексаторы (подобные свойствам, но принимающие параметр как индекс массива), события, делегаты, циклы `foreach`, структуры, передаваемые по значению, автоматическое преобразование встроенных типов в объекты при необходимости (boxing), атрибуты, встроенные средства взаимодействия с неуправляемым кодом (DLL, COM) и прочее [20].

Версия Microsoft .NET 2.0 привнесла много новых возможностей в сравнении с предыдущей версией, что отразилось и на языках под эту платформу. Проект спецификации C# 2.0 впервые был опубликован Microsoft в октябре 2003 года; в 2004 году выходили бета-версии (проект с кодовым названием Whidbey), C# 2.0 окончательно вышел 7 ноября 2005 года вместе с Visual Studio 2005 и Microsoft .NET 2.0. Ниже перечислены новые возможности в версии 2.0

– Частичные типы (разделение реализации класса более чем на один файл).

– Обобщённые, или параметризованные типы (generics). В отличие от шаблонов C++, они поддерживают некоторые дополнительные возможности и работают на уровне виртуальной машины. Вместе с тем, параметрами обобщённого типа не могут быть выражения, они не могут быть полностью или частично специализированы, не поддерживают шаблонных параметров по умолчанию, от шаблонного параметра нельзя наследоваться.

– Новая форма итератора, позволяющая создавать сопрограммы с

помощью ключевого слова `yield`, подобно Python и Ruby.

– Анонимные методы, обеспечивающие функциональность замыканий.

– Оператор ??: `return obj1 ?? obj2;` означает (в нотации C# 1.0) `return obj1 !=null ? obj1 : obj2;`.

– Обнуляемые (nullable) типы-значения (обозначаемые вопросительным знаком, например, `int? i = null;`), представляющие собой те же самые типы-значения, способные принимать также значение null. Такие типы позволяют улучшить взаимодействие с базами данных через язык SQL.

– Поддержка 64-разрядных вычислений позволяет увеличить адресное пространство и использовать 64-разрядные примитивные типы данных [20].

Третья версия языка имела одно большое нововведение — Language Integrated Query (LINQ), для реализации которого в языке дополнительно появилось множество дополнительных возможностей. Ниже приведены некоторые из них:

– Ключевые слова `select`, `from`, `where`, позволяющие делать запросы из SQL, XML, коллекций и т. п.

– Инициализацию объекта вместе с его свойствами:

```
Customer c = new Customer(); c.Name = "James"; c.Age=30;
```

можно записать как

```
Customer c = new Customer { Name = "James", Age = 30 };
```

– Лямбда-выражения:

```
listOfFoo.Where(delegate(Foo x) { return x.size > 10; });
```

теперь можно записать как

```
listOfFoo.Where(x => x.size > 10);
```

– Деревья выражений — лямбда-выражения теперь могут быть представлены в виде структуры данных, доступной для обхода во время выполнения, тем самым позволяя транслировать строго типизированные C#-выражения в другие домены (например, выражения SQL).

– Вывод типов локальной переменной: `var x = "hello";` вместо `string x = "hello";`

– Безымянные типы: `var x = new { Name = "James"};`

– Методы-расширения — добавление метода в существующий класс с помощью ключевого слова `this` при первом параметре статической функции.

– Автоматические свойства: компилятор сгенерирует закрытое поле и соответствующие аксессор и мутатор для кода вида

```
public string Name { get; private set; }
```

C# 3.0 совместим с C# 2.0 по генерируемому MSIL-коду; улучшения в языке — чисто синтаксические и реализуются на этапе компиляции [20].

Visual Basic .NET 10.0 и C# 4.0 были выпущены в апреле 2010 года, одновременно с выпуском Visual Studio 2010. Новые возможности в версии 4.0:

- Возможность использования позднего связывания.
- Именованные и опциональные параметры.
- Новые возможности COM interop.
- Ковариантность и контрвариантность интерфейсов и делегатов.
- Контракты в коде (Code Contracts) [20].

В C# 5.0 было немного нововведений, но они носят большую практическую ценность. В новой версии появилась упрощенная поддержка выполнения асинхронных функций с помощью двух новых слов — `async` и `await` . Ключевым словом `async` помечаются методы и лямбда-выражения, которые внутри содержат ожидание выполнения асинхронных операций с помощью оператора `await`, который отвечает за преобразования кода метода во время компиляции.

2.3 Язык программирования F#

F# — это мультипарадигменный язык программирования, разработанный в подразделении Microsot Research и предназначенный для исполнения на платформе Microsoft .NET. Он сочетает в себе выразительность функциональных языков, таких как OCaml и Haskell с возможностями и объектной моделью Microsoft .NET. История F# началась в 2002 году, когда команда разработчиков из Microsoft Research под руководством Дона Сайма решила, что языки семейства ML вполне подходят для реализации функциональной парадигмы на платформе Microsoft .NET. Идея разработки нового языка появилась во время работы над реализацией обобщённого программирования для Common Language Runtime. Известно, что одно время в качестве прототипа нового языка рассматривался Haskell, но из-за функциональной чистоты и более сложной системы типов потенциальный Haskell.NET не мог бы предоставить разработчикам простого механизма работы с библиотекой классов .NET Framework, а значит, не давал бы каких-то дополнительных преимуществ. Как бы то ни было, за основу был взят OCaml, язык из семейства ML, который не является чисто функциональным и предоставляет возможности для императивного и объектно-ориентированного программирования. Однако Haskell хоть и

не стал непосредственно родителем нового языка, тем не менее, оказал на него некоторое влияние. Например, концепция вычислительных выражений (computation expressions или workflows), играющих важную роль для асинхронного программирования и реализации DSL на F#, позаимствована из монад Haskell [21].

Следует также отметить, что на данный момент F# является, пожалуй, единственным функциональным языком программирования, который продвигается одним из ведущих производителей в области разработки программного обеспечения. Он позволяет использовать множество уже существующих библиотек, писать приложения для самых разных платформ и что не менее важно — делать всё это в современной IDE [21].

Далее рассматриваются некоторые из возможностей предоставляемых F#.

2.3.1 Функциональная парадигма. Будучи наследником славных традиций семейства языков ML, предоставляет полный набор инструментов функционального программирования: здесь есть алгебраические типы данных и функции высшего порядка, средства для композиции функций и неизменяемые структуры данных, а также частичное применение на пару с каррированием. Все функциональные возможности F# реализованы в конечном итоге поверх общей системы типов .NET Framework. Однако этот факт не обеспечивает удобства использования таких конструкций из других языков платформы. При разработке собственных библиотек на F# следует предусмотреть создание объектно-ориентированных обёрток, которые будет проще использовать из C# или Visual Basic .NET [21]. Рекомендации по проектированию таких библиотек приведены в [12].

2.3.2 Императивное программирование. В случаях, когда богатых функциональных возможностей не хватает, F# предоставляет разработчику возможность использовать в коде прелести изменяемого состояния. Это как непосредственно изменяемые переменные, поддержка полей и свойств объектов стандартной библиотеки, так и явные циклы, а также изменяемые коллекции и типы данных.

2.3.3 Объектно-ориентированная парадигма. Объектно-ориентированные возможности F#, как и многое другое в этом языке, обусловлены необходимостью предоставить разработчикам возможность использовать стандартную библиотеку классов .NET Framework. С поставленной задачей язык вполне справляется: можно как использовать

библиотеки классов, реализованные на других .NET языках, так и разрабатывать свои собственные. Следует отметить, однако, что некоторые возможности ОО языков реализованы не самым привычным образом [21].

2.3.4 Система типов. Каждая переменная, выражение или функция в F# имеет тип. Можно считать, что тип — это некий контракт, поддерживаемый всеми объектами данного типа. К примеру, тип переменной однозначно определяет диапазон значений этой переменной; тип функции говорит о том, какие параметры она принимает и значение какого типа она возвращает; тип объекта определяет то, как этот объект может быть использован [21].

F# — статически типизированный язык. Это означает, что тип каждого выражения известен на этапе компиляции, и позволяет отслеживать ошибки, связанные с неправильным использованием объектов определенного типа, до запуска программы. Помимо этого, F# — язык программирования со строгой типизацией, а значит, в выражениях отсутствует неявное приведение типов. Попытка использовать целое число там, где компилятор ожидает увидеть число с плавающей точкой, приведёт к ошибке компиляции [21].

2.3.5 Вывод типов. В отличие от большинства других промышленных языков программирования, F# не требует явно указывать типы всех значений. Механизм вывода типов позволяет определить недостающие типы значений, глядя на их использование. При этом некоторые значения должны иметь заранее известный тип. В роли таких значений, к примеру, могут выступать числовые литералы, так как их тип однозначно определяется суффиксом [21]. В листинге 2.1 приведен простой пример:

Листинг 2.1 – Пример автоматического вывода типа функции

```
> let add a b = a + b;;
val add : int -> int -> int
> add 3 5;;
val it : int = 8
```

Функция `add` возвращает сумму своих параметров и имеет тип `int -> int -> int`. Если не смотреть на сигнатуру функции, то можно подумать, что она складывает значения любых типов, но это не так. Попытка вызвать её для аргументов типа `float` или `decimal` приведёт к ошибке компиляции. Причина такого поведения кроется в механизме вывода типов. Поскольку оператор `+` может работать с разными типами данных, а никакой дополнительной информации о том, как эта функция будет использоваться, нет, компилятор по умолчанию приводит её к типу `int -> int -> int` [21].

В большинстве случаев автоматический вывод типов является очень удобным и способствует написанию полиморфный функций. Алгоритм, используемый компилятором F#, использует глобальный вывод типов и позволяет справляться даже с очень сложными типами. Для демонстрации возможностей вывода типа для полиморфных функций рассмотрим классический пример — комбинаторный базис SKI [22, с. 21]:

$$I = \lambda x.x \tag{2.1}$$

$$K = \lambda x\ y.x \tag{2.2}$$

$$S = \lambda f\ g\ x.(f\ x)(g\ x)\,. \tag{2.3}$$

Пример вывода интерактивного окружения F# приведен в листинге 2.2. Как можно заметить типы полученных функций довольно сложные, но компилятор смог их вывести.

Листинг 2.2 – Пример определения комбинаторного базиса SKI

```
> let I x = x;;
val I : x:'a -> 'a

> let K x y = x;;
val K : x:'a -> y:'b -> 'a

> let S f g x = (f x) (g x);;
val S : f:('a -> 'b -> 'c) -> g:('a -> 'b) -> x:'a -> 'c
```

Обычно при программировании на F# в функциональном стиле нет необходимости указывать типы явно, транслятор сам назначит выражению наиболее общий тип. Однако, иногда бывает полезно ограничить вывод типа. Подобная мера не заставит работать код, который до этого не работал, но может использоваться как документация для понимания его предназначения; также возможно использовать более короткие синонимы для сложных типов. Ограничение типа может быть задано в F# путём добавления аннотации типа после некоторого выражения. Аннотации типов состоят из двоеточия, за которым указан тип. Обычно расположение аннотаций не имеет значения; если они есть, то они заставляют компилятор использовать соответствующие ограничения [22, с. 59].

2.3.6 Стратегии вычислений. Обычно выражения в F# вычисляются «энергично». Это означает, что значение выражения будет вычислено независимо от того, используется оно где-либо или нет. В противоположность жадному подходу существует стратегия ленивых вычислений, которая позволяет вычислять значение выражения только

тогда, когда оно становится необходимо. Преимуществами такого подхода являются:

– производительность, поскольку неиспользуемые значения просто не вычисляются;

– возможность работать с бесконечными или очень большими последовательностями, так как они никогда не загружаются в память полностью;

– декларативность кода. Использование ленивых вычислений избавляет программиста от необходимости следить за порядком вычислений, что делает код проще.

Главный недостаток ленивых вычислений — плохая предсказуемость. В отличие от энергичных вычислений, где очень просто определить пространственную и временную сложность алгоритма, с ленивыми вычислениями всё может быть куда менее очевидно. F# позволяет программисту самостоятельно решать, что именно должно вычисляться лениво, а что нет. Это в значительной степени устраняет проблему плохой предсказуемости, так как ленивые вычисления применяются только там, где это действительно необходимо, позволяя сочетать лучшее из обоих миров [21].

2.3.7 Сопоставление с образцом. Образец — это описание «формы» ожидаемой структуры данных: образец сам по себе похож на литерал структуры данных (т. е. он состоит из конструкторов алгебраических типов и литералов примитивных типов: целых, строковых и т. п.), однако может содержать метапеременные — «дырки», обозначающие: «значение, которое встретится в данном месте, назовем данным именем» [23]. Сопоставление с образцом — основной способ работы со структурами данных в F#. Эта языковая конструкция состоит из ключевого слова `match`, анализируемого выражения, ключевого слова `with` и набора правил. Каждое правило — это пара образец-результат. Всё выражение сопоставления с образцом принимает значение того правила, образец которого соответствует анализируемому выражению. Все правила сопоставления с образцом должны возвращать значения одного и того же типа. В простейшем случае в качестве образцов могут выступать константы:

```
> let xor x y =
      match x, y with
      | true, true -> false
      | false, false -> false
      | true, false -> true
      | false, true -> true ;;
```

```
val xor : bool -> bool -> bool
```

В правилах сопоставления с образцом можно использовать символ подчеркивания, если конкретное значение неважно. Если набор правил сопоставления не покрывает всевозможные значения образца, компилятор выдаёт предупреждение. На этапе исполнения, если ни одно правило не будет соответствовать образцу, будет сгенерировано исключение [21]. Сопоставление с образцом очень мощный механизм, но иногда и его выразительности недостаточно для описание идеи. Язык F# вводит понятие активных шаблонов, когда шаблон может представлять из себя пользовательскую функцию, которая может содержать дополнительную логику обработки.

2.3.8 Вычислительные выражения. Среди нововведений F# можно особо выделить так называемые вычислительные выражения (computation expressions или workflows). Они являются обобщением генераторов последовательности и, в частности, позволяют встраивать в F# такие вычислительные структуры, как монады и моноиды. Также они могут быть применены для асинхронного программирования и создания DSL [21].

Вычислительное выражение имеет форму блока, содержащего некоторый код на F# в фигурных скобках. Этому блоку должен предшествовать специальный объект, который называется еще построителем (builder). Общая форма следующая: `builder { comp-expr }`. Построитель определяет способ интерпретации того кода, который указан в фигурных скобках. Сам код вычисления внешне почти не отличается от обычного кода на F#, кроме того, что в нём нельзя определять новые типы, а также нельзя использовать изменяемые значения. Вместо таких значений можно использовать ссылки, но делать это следует с большой осторожностью, поскольку вычислительные выражения обычно задают некие отложенные вычисления, а последние не очень любят побочные эффекты [21].

2.3.9 Асинхронные потоки операций. Асинхронные потоки операций — это один из самых интересных примеров практического использования вычислительных выражений. Код, выполняющий какие-либо неблокирующие операции ввода-вывода, как правило сложен для понимания, поскольку представляет из себя множество методов обратного вызова, каждый из которых обрабатывает какой-то промежуточный результат и возможно начинает новую асинхронную операцию. Асинхронные потоки операций позволяют писать асинхронный код последовательно, не определяя

методы обратного вызова явно. Для создания асинхронного потока операций используется построитель `async` [21]

3 АРХИТЕКТУРА И МОДУЛИ СИСТЕМЫ

Разработанное программное обеспечение представляет из себя библиотеку кода написанную на языках F# и C#. Библиотека предназначена для представления модификации классических байесовых сетей, упомянутой на странице 10 в подразделе 1.1.

3.1 Типы для работы с графами

Так как в дипломном проекте рассматривается одна из разновидностей графовых моделей, то, очевидно, для представления таких моделей в разрабатываемой библиотеке должна быть часть, отвечающая за представление и работу с графами. При реализации здесь было несколько альтернативных путей: использовать одну из доступных библиотек для платформы Microsoft .NET для работы с графами или реализовать собственную. Среди готовых библиотек можно было бы использовать QuickGraph[1)], Directed Graph for .NET[2)] или GrapheNET[3)]. Но было принято решение остановиться на варианте, подразумевающем разработку собственных типов для работы с графами. Это решение было обосновано тем, что вышеуказанные библиотеки являются сложными и содержат в себе очень много функциональности не нужной для решения поставленных задач, в дополнение не было приобретено лишних мегабайтовых внешних зависимостей для библиотеки.

Одним из важных решений, которое было принято в начале проектирования модуля работы с графами, было использование по возможности неизменяемых структур данных. Это решение выгодно отличает разработанную реализацию от существующих библиотек, от использования которых было принято решение отказаться. Существующие библиотеки ориентированы на работу в императивном стиле и с изменяемым состоянием. Также использование неизменяемых структур данных для реализации типов для представления графов в дальнейшем положительно сказалось на простоте реализации поиска структуры вероятностной сети в алгоритмах вывода структуры по данным. Часть разрабатываемой библиотеки, содержащая типы для работы с графами, реализована на языке программирования F#. Краткое описание основных особенностей данного языка приведено в подразделе 2.3. Решение использовать данный язык было

[1)] http://quickgraph.codeplex.com/
[2)] http://directedgraph4net.codeplex.com/
[3)] http://graphenet.codeplex.com/

продиктовано желанием сократить количество возможных ошибок и размер кодовой базы, необходимой для реализации поставленной задачи, а так же желанием применить в «боевых» условиях язык программирования с хорошей поддержкой функционального программирования.

Внутренним представлением графа является неизменяемый словарь, содержащий вложенный неизменяемый мульти-словарь[1)]. Основные определения структуры данных графа приведены в листинге 3.1:

Листинг 3.1 – Определение структуры данных для представления графа

```
/// Graph arc
type Arc<'T> =
    | Outgoing of 'T
    | Incoming of 'T

/// Immutable Graph class
[<ReferenceEquality; NoComparison>]
type Graph<'Vertex, 'Arc when 'Vertex : comparison and 'Vertex : equality and 'Vertex :>
    IComparable> =
    private Graph of Map<'Vertex, MultiMap<'Vertex, Arc<'Arc>>>
```

Данная структура данных для представления графа подходит для работы как с неориентированными так и с ориентированными мультиграфами. Воспринимать граф как ориентированный или нет задача конкретного алгоритма, работающего с графом. Разработанная библиотека для представления графа предоставляет необходимые операции для манипулирования структурой графа. Библиотека также предоставляет небольшое количество алгоритмов для работы с графами, необходимых в рамках решения задач, возникающих при поиске структуры вероятностной сети. В частности реализованы алгоритмы топологической сортировки, поиска в глубину и ширину, поиска сильно-связанных компонетов и проверки графа на наличие направленных циклов.

Одной из особенностей разработанной библиотеки является ориентированность на использование как из языка F# в «функциональном стиле», так и из языка C# — в «императивном». Для реализации данной возможности были учтены рекомендации приведенные в [12]. Функциональность разработанной библиотеки покрыта большим набором модульных тестов, написанных с использованием библиотек xUnit[2)] и Unquote[3)].

[1)] Словарь позволяющий хранить множество значений с одинаковым ключом.

[2)] `http://xunit.codeplex.com/`

[3)] `http://code.google.com/p/unquote/`

3.2 Представление вероятностной сети

Другой важной частью разработанной библиотеки являются типы для представления и работы с самими вероятностными сетями. Первостепенными требованиями, поставленными перед началом проектирования типов, были следующие пункты:

– Типы предназначены для представления модификации классических байесовых сетей, упомянутой в разделе 1.1 на странице 10.

– Представление сети должно быть «многослойным». Под «многослойностью» понимается возможность расширения представления сети дополнительными «слоями» атрибутов, с целью увеличения количества сценариев, в которых данные типы пригодны к использованию. Например, в случае когда нужно знать лишь структуру сети можно использовать лишь информацию о структуре — граф. Для проведения статистического вывода суждений добавляется дополнительный «слой», содержащий талицы условных и безусловных вероятностей. В случаях, когда нужно отображать сеть пользователю, добавляется еще один «слой», содержащий дополнительную информацию о переменных и их состояниях.

– Сеть должна предоставлять возможность отмены вносимых в нее изменений, т. е. по сути поддерживать версионность.

– Сеть должна предоставлять возможность валидации её структуры.

Приняв во внимание приведенные выше требования были приняты следующие решения:

– Необходимо разработать отдельные типы для представления вершин вероятностной модели и связей между переменными в этой модели. В разработанной библиотеке за это отвечают типы `Node<'T>` и `Link<'T>`, содержащие информацию о переменных, таблицы распределения и дополнительные атрибуты. Использование параметрического полиморфизма в реализации данных типов играет ключевую роль в обеспечении «многослойности» и расширяемости представления вероятностной сети.

– Необходимы типы для представления распределения. В предложенной реализации был разработан тип для представления безусловного распределения случайной величины, эта таблица хранится в сети как один из аттрибутов типа `Node<'T>`, и тип для представления условного распределения пары случайных величин, экземпляр данного типа хранится как аттрибут связи между переменными — `Link<'T>`. Было сочтено целесообразным в качестве внутренней реализации таблиц распределения использовать готовую библиотеку для работы с матрицами и другими

математическими объектами и понятиями — Math.NET Numerics[1]. Соответственно в предложенной реализации использовались типы `Vector <float>` и `Matrix<float>` и сопутствующие операции над ними. Использование данной библиотеки позволило сократить объём сопутствующего кода, необходимого для реализации библиотеки для работы с вероятностными сетями, также уменьшив множество потенциальных ошибок реализации. В данном случае преимущества от использования библиотеки превысили затраты на добавление и поддержку дополнительных зависимостей.

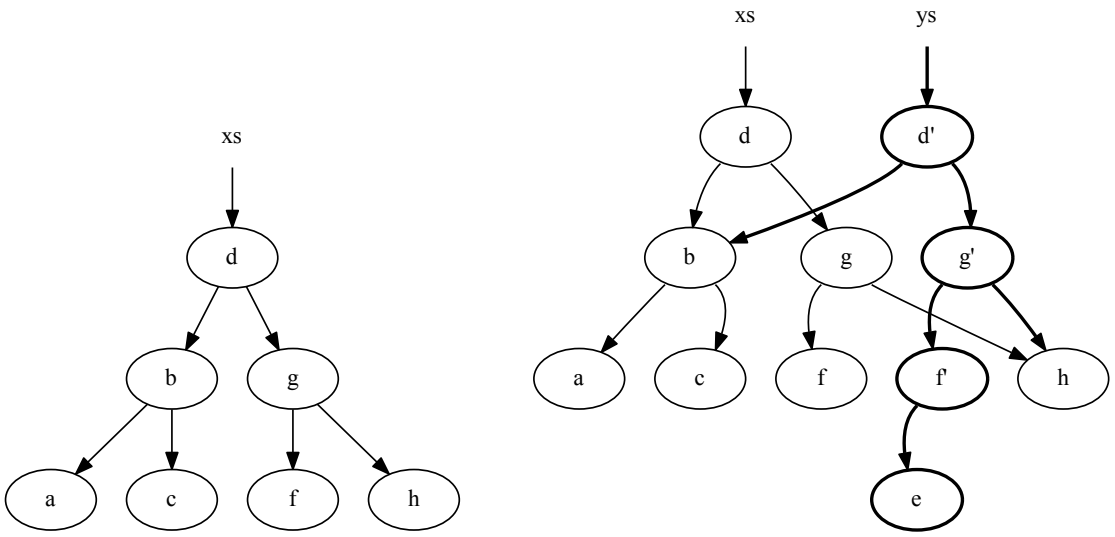
– Требование возможности отмены изменений вносимых в вероятностную сеть привело к реализации сети, как и в случае типов для представления графов, к реализации сети как неизменяемой структуры данных. Все операции, при условии использования специальных функций, возвращают новый экземпляр сети, оставляя старый не изменённым. Подобная реализация типов автоматически дает возможность производить версионирование экземпляров типа, т. к. всегда есть доступ к изменённой копии и исходному экземпляру. С первого взгляда данный подход кажется очень расточительным по памяти, но на самом деле оказывается, что все с точностью до наоборот, т. к. обычно, и в данном конкретном случае, при модификации неизменяемой структуры данных большая часть структуры разделяется между копией и исходной структурой, а физически копирование памяти происходит лишь в тех местах, которые действительно необходимо было поменять. Для убедительности, сказанное проиллюстрировано на рисунке 3.1.

– Валидация сети происходит на этапе её построения и модификации. Дополнительно существуют функции для проверки структуры сети на ацикличность. Ацикличность ориентированного графа проверяется с помощью алгоритма нахождения компонент сильной связности Косарайю[2].

В результате получилось довольно простое и расширяемое представление сети, основное определение которого приведено в листинге 3.2. Параметризация типа параметрами `'NodeAttributes` и `'LinkAttributes` и расширяемое устройство типов `Node<'T>` и `Link<'T>` позволяет достигнуть заявленной расширяемости представления сети и её «многослойности». Библиотека содержит предопредёленные типы, для представления уровней. Параметризация типа `BNet<unit, unit>` представляет структуру сети и таблицы распределения. «Слой» с дополнительными аттрибутами, планируемыми для использования в алгоритмах статистического вывода

[1] `http://numerics.mathdotnet.com/`

[2] `http://en.wikipedia.org/wiki/Kosaraju's_algorithm`



(а) (б)

Рисунок 3.1 – Пример разделения структуры в неизменяемых структурах данных: а — исходное дерево; б — измененное дерево, добавлена вершина *e*;

суждений представлен типами `NodeAttributes<'Annotations>` и `LinkAttributes`, но на момент защиты дипломного проекта, из-за неготовой реализации алгоритмов статистического вывода суждений, данные типы содержат не все необходимые атрибуты для работы таких алгоритмов, а лишь прогнозируемых заготовки. Дополнительный «слой», потенциально необходимый при построении пользовательских приложений, представлен типом `VarAnnotations`. Данный тип содержит дополнительную информацию о переменной, такую как её название и аннотации возможных значений переменной. Таким образом, наиболее полное представление сети, содержащее все «слои», в коде параметризуется следующим образом `BNet<NodeAttributes<VarAnnotations>, LinkAttributes>`.

Листинг 3.2 – Определение структуры данных для вероятностной сети

```
/// Represents immutable BN. Nodes and links between nodes.
type BNet<'NodeAttributes, 'LinkAttributes> =
    private { nodes: Map<int, Node<'NodeAttributes>>;
              links: Map<int * int, Link<'LinkAttributes>>;
              graph: Graph<int, unit>; }
```

Таким образом, использование параметрического полиморфизма и неизменяемых типов данных, позволило добиться поставленных при проектировании целей, а также довольно легкой возможности расширять сеть

в дальнейшем.

3.3 Сохранение сети

Помимо функциональности, связанной с представлением, манипуляцией и выведением структуры сети, разработанная библиотека предоставляет возможность импорта и экспорта вероятностной сети из и в различные форматы. Т. к. библиотека предназначена для работы с модификацией вероятностных сетей, отличающейся в нескольких ключевых моментах от классических байесовых сетей, то нельзя было использовать общепринятые форматы для хранения сетей во внешней памяти и необходимо было разработать свой формат. Разработанный формат хранения сетей основывается на XML и предназначен для полного сохранения состояния представления сети, используемого в программе. Данный формат в большей степени похож на ручную сериализацию, чем на удобный формат для обмена вероятностными сетями. Пример сети, представленной в данном формате, приведен в листинге 3.3.

У разработанного формата есть существенный недостаток, его «понимает» только разработанная библиотека. В связи с тем, что в данном дипломном проекте основной целью является реализация лишь малой части возможных операций над вероятностными сетями — построение структуры по данным, целесообразно было добавить в библиотеку, хотя и весьма ограниченную, возможность загрузки и сохранения вероятностных сетей из и в существующие распространённые форматы. Список поддерживаемых форматов приведён в таблице 3.1. Поддержка нескольких форматов понадобилась потому, что многие существующие программы, которые использовались в различной степени для оценки результатов проделанной работы, поддерживают весьма ограниченный набор форматов. Таким образом, имея возможность экспортировать сеть, построенную одним из алгоритмов реализованных в разработанной библиотеке по данным, в общеиспользуемый формат можно использовать обученную сеть для статистического вывода суждений и других операций в существующих программах, т. к. в данный момент в разработанной библиотеке данная функциональность не реализована. Отдельно стоит отметить возможность сохранения структуры сети в формат представления графов программы GraphViz[1)]. Данное ПО использовалось с целью визуализации выведенных структур и экспорта полученной визуализации в

[1)] `http://www.graphviz.org/`

один из векторных графических форматов. Утилита dot из состава GraphViz умеет автоматически визуализировать сложные графы наилучшим для отображения образом.

Листинг 3.3 – Пример представления простой вероятностной сети в собственном XML-формате

```
<network>
  <variables>
    <variable id="1" dim="2" />
    <variable id="2" dim="3" />
  </variables>
  <node_attributes>
    <node variable_id="1"> <answered>false</answered> </node>
    <node variable_id="2"> <answered>true</answered> </node>
  </node_attributes>
  <link_attributes>
    <link parent_id="1" child_id="2" />
  </link_attributes>
  <variable_annotations>
    <variable_annotation variable_id="1">
      <name>My variable</name>
      <annotations>
        <label>Yes</label> <label>No</label>
      </annotations>
    </variable_annotation>
    <variable_annotation variable_id="2">
      <name>Color</name>
      <annotations>
        <label>Red</label> <label>Green</label> <label>Blue</label>
      </annotations>
    </variable_annotation>
  </variable_annotations>
  <probability_tables>
    <probability_table variable_id="1">
      <vector>0.2 0.8</vector>
    </probability_table>
    <probability_table variable_id="2">
      <vector>0.4 0.3 0.3</vector>
    </probability_table>
  </probability_tables>
  <forward_probability_tables>
    <forward_probability_table variable_id="2" condition_variable_id="1">
      <matrix nrows="3" ncols="2">0.1 0.2 0.7 0.6 0.1 0.3</matrix>
    </forward_probability_table>
  </forward_probability_tables>
</network>
```

3.4 Представление экспериментальных данных

Немаловажной задачей в обучении и построении структуры сети по данным является представление набора экспериментальных данных в оперативной и постоянной памяти. В машинном обучении и других областях, связанных с обработкой массивов данных, для хранения данных на диске в большинстве случаев применяется простой текстовый формат *csv* —

Таблица 3.1 – Поддерживаемые форматы хранения вероятностных сетей

Формат	Поддержка импорта	Поддержка экспорта
Собственный xml-формат	полная	полная
XMLBIF	частичная	частичная
GeNIe	частичная	отсутствует
GraphViz dot	отсутствует	полная

значения, разделённые специальным символом и записанные в текстовый файл построчно. В разработанной библиотеке также используется данный формат для импортирования экспериментальных данных с диска в память программы для дальнейшей обработки. Для чтения *csv* файлов используется легковесная внешняя библиотека LumenWorks.Framework.IO[1)].

Для представления набора экспериментальны данных в библиотеке присутствует специальный тип — DataFrame, который представляет из себя информацию о переменных и, собственно, набор экспериментальных данных в компактном для хранения виде. Из особенностей реализации стоит упомянуть способ достижения компактности хранения. При чтении *csv* файла каждому состоянию переменной назначается некоторое 8-битное число, которое является представлением данного состояния в памяти компьютера. Использование 8-битного числа с одной стороны ограничивает число возможных состояний одной переменной до 256, с другой стороны — данное представление достаточно компактно, чтобы быть пригодным для работы на персональном компьютере разработчика с ограниченным размером ОЗУ и уметь обрабатывать наборы данных из миллионов случаев для десятков переменных. Одной из дополнительных возможностей DataFrame является возможность производить случайные выборки из имеющегося набора данных. Данная возможность была использована при реализации алгоритмов вывода структуры вероятностной сети по данным.

3.5 Байесовы сети Asia и ALARM

Перед тем как перейти к обсуждению разработанных алгоритмов вывода структуры сети по данным целесообразно обсудить известные сети, которые использовались в качестве моделей для вывода по экспериментальным данным. Речь пойдет о ставших уже классикой в таких задачах — сетях Asia и ALARM.

[1)] `http://www.codeproject.com/Articles/9258/A-Fast-CSV-Reader`

3.5.1 Asia Байесова сеть Asia является небольшой синтетической сетью, обычно используемой при изучении вероятностных сетей. Данная вероятностная сеть рассмотрена в работе [24]. Искусственная байеосова сеть Asia предназначена для диагностики у пациентов заболеваний связанных с лёгкими. В перечень диагностируемых болезней входят туберкулёз, рак и бронхит. Данная сеть имеет восемь бинарных случайных величин. Структура данной сети приведена на рисунке 1.3 (б) на странице 23. В таблице 3.2 приведено описание переменных.

Таблица 3.2 – Описание переменных сети Asia

Переменная	Количество состояний	Примечание
VisitAsia	2	посещал ли пациент Азию
Tuberculosis	2	болен туберкулёзом
Smoking	2	курит
Cancer	2	имеет рак легких
TbOrCa	2	имеет рак или туберкулёз
XRay	2	плохая рентгенография
Bronchitis	2	болен бронхитом
Dyspnea	2	испытывает удушье

3.5.2 ALARM Данная байесова сеть также очень часто рассматривается для оценки качества различных алгоритмов, работающих с вероятностными сетями. Данная сеть была рассмотрена в работе [25]. Сеть предназначена для медицинской диагностики, и используется для обработки физиологических наблюдений пациента. Сеть состоит из переменных трех типов: диагнозов, физиологических показателей и скрытых переменных, которые измерить на практике нельзя. Данная сеть содержит 37 переменных и 46 связей между ними. Максимальное количество переменных-родителей равно четырём. Рассматриваемая вероятностная сеть относится к сетям средник размеров. Данная сеть хорошо изучена и представляет интерес, как модель для оценки качества реализованных в библиотеке алгоритмов. Структура сети приведена на рисунке 3.2. Из-за довольно большого количества переменных здесь не приводится их описание и назначение. В этой информации нет необходимости для оценки качества реализованных алгоритмов, важно знать общую структуру сети.

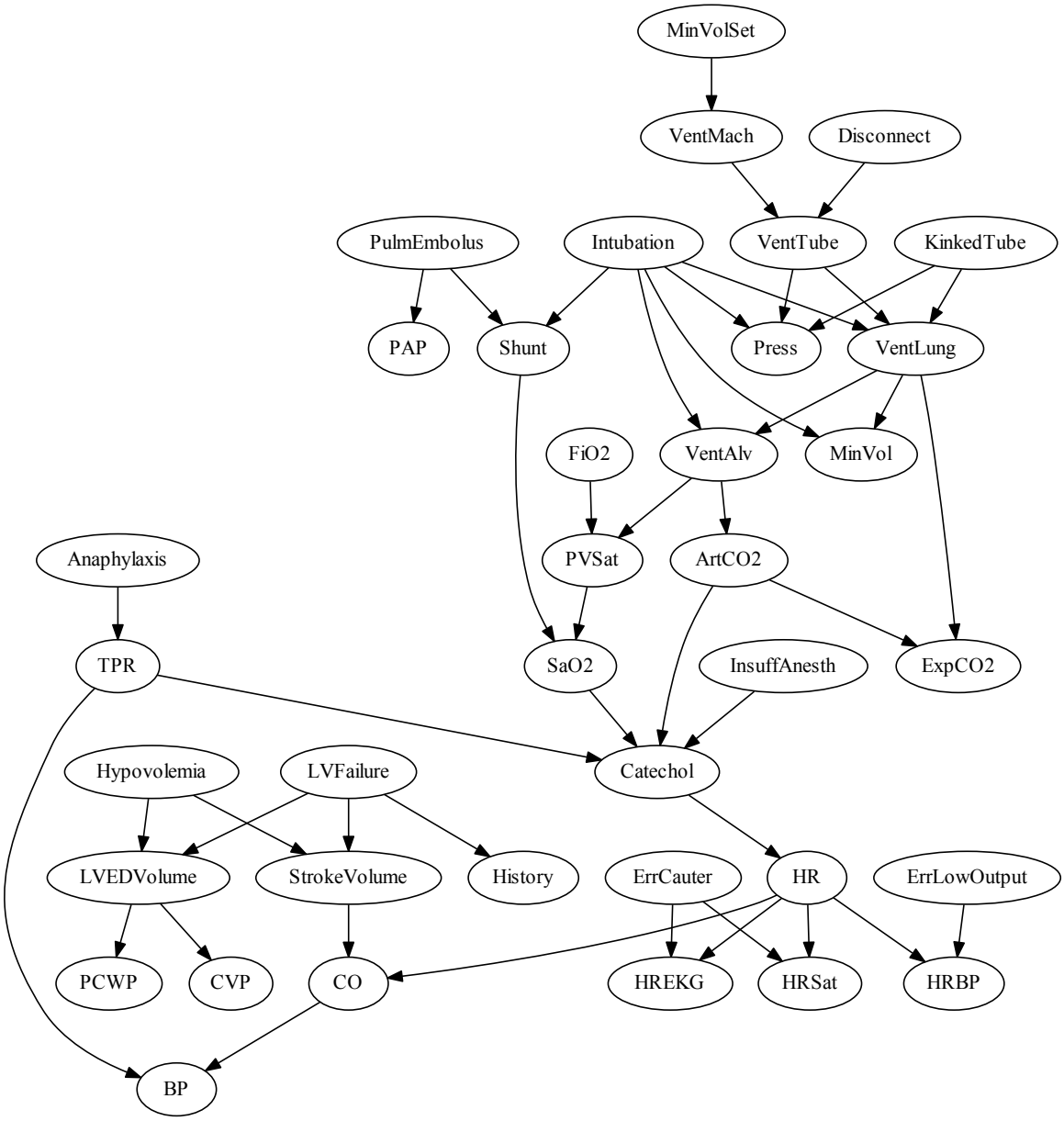


Рисунок 3.2 – Структура байесовой сети ALARM

3.6 Алгоритм на основе оценки апостериорной вероятности структуры

В данном подразделе рассматривается известный алгоритм, использующий оценку апостериорной вероятности в качестве критерия поиска. Подробное описание данной оценки и базового алгоритма поиска приведены в работе [7].

Формулу (1.13) для оценки совместной вероятности на практике напрямую использовать не получится, без введения дополнительных

предположений. Необходимо сделать предположение, что все возможные структуры равновероятны, т. е. $P(B_S)$ равно некоторой малой константе c. Таким образом нахождение оптимальной структуры сводится к максимизации формулы (3.1), т. е. задача сводится к нахождению множества вершин-предков π_i для каждой вершины X_i, оптимизирующих целевую функцию [7].

$$\max \left[P(B_S, x^R[n])\right] = \\ = c \prod_{i=1}^{R} \max_{\pi_i} \left[\prod_{j=1}^{q_i} \frac{(\alpha_i - 1)!}{(n[\phi_i[j], i, B_S] + \alpha_i - 1)!} \prod_{k=1}^{\alpha_i} n[v_{ik}, \phi_i[j], i, B_S]! \right]. \quad (3.1)$$

Таким образом наивный алгоритм поиска состоит в полном переборе всех возможных родителей для каждой вершины и оптимизации при этом функции (3.2):

$$g(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(\alpha_i - 1)!}{(n[\phi_i[j], i, B_S] + \alpha_i - 1)!} \prod_{k=1}^{\alpha_i} n[v_{ik}, \phi_i[j], i, B_S]! \,. \quad (3.2)$$

На практике наивный вариант не годится из-за большого количества возможных вариантов структур сетей. В разработанной реализации использовались те же ограничения и стратегия поиска, что и в работе [7]. Перед началом выполнения алгоритма требуется знание о порядке вершин, таком, что вершины родители всегда находятся раньше вершин потомков. Схематически алгоритм поиска выглядит следующим образом:

Листинг 3.4 – Псевдокод реализации алгоритма K2

```
function k2 =
  (* Input: dataset x^R[n], ordering of variables, u - maximum number of parents per
      variable.
     Output: for each node, a printout of the parents of the node. *)
  for i in 1 .. R do
    π_i := ∅
    P_old := g(i, π_i); // formula(3.2)
    OkToContinue := true;
    while OkToContinue and |π_i| < u do
      let z = node in Pred(X_i) − π_i that maximizes g(i, π_i ∪ z)
      P_new := f(i, π_i ∪ z)
      if P_new > P_old then
        P_old := P_new;
        π_i := π_i ∪ z
      else OkToContinue := false
    end while
    printfn("Node: ", X_i, "Parents of X_i:", π_i)
  end for
```

```
end
```

В разработанной в рамках дипломного проекта реализации за основу был взят алгоритм K2, приведенный в работе [7], псевдокод которого показан в листинге 3.4. В разработанном алгоритме слегка изменен способ подсчета целевой функции. Приняв во внимание ограничение на представление в компьютере вещественных и больших целых чисел, а также то, что операции умножения, деления и возведения в степень более сложные, было принято решение воспользоваться прологарифмированной версией формулы (3.2). Ниже приводится точная формула (3.3), по которой вычисляется оценка в реализованном алгоритме. Данная формула более удобная для вычисления на компьютере:

$$\begin{aligned}
\log(g(i,\pi_i)) &= \sum_{j=1}^{q_i} \log\left(\frac{(\alpha_i-1)!}{(n_{ij}+\alpha_i-1)!}\prod_{k=1}^{\alpha_i} n_{ijk}!\right) = \\
&= \sum_{j=1}^{q_i}\left(\log\frac{(\alpha_i-1)!}{(n_{ij}+\alpha_i-1)!} + \log\prod_{k=1}^{\alpha_i} n_{ijk}!\right) = \\
&= \sum_{j=1}^{q_i}\left(\log(\alpha_i-1)! - \log(n_{ij}+\alpha_i-1)! + \sum_{k=1}^{\alpha_i}\log n_{ijk}!\right) = \\
&= \sum_{j=1}^{q_i}\left(\log\Gamma(\alpha_i) - \log\Gamma(n_{ij}+\alpha_i) + \sum_{k=1}^{\alpha_i}\log\Gamma(n_{ijk}+1)\right) = \\
&= q_i\log\Gamma(\alpha_i) + \sum_{j=1}^{q_i}\left(\sum_{k=1}^{\alpha_i}\log\Gamma(n_{ijk}+1) - \log\Gamma(n_{ij}+\alpha_i)\right),
\end{aligned} \tag{3.3}$$

где Γ — гамма-функция — расширение понятия факториала на поле комплексных чисел;

n_{ijk} — условное, более краткое, обозначение для $n[v_{ik}, \phi_i[j], i, B_S]$;

n_{ij} — условное, более краткое, обозначение для $n[\phi_i[j], i, B_S]$.

Помимо использования формулы (3.3) в реализации были произведены дополнительные оптимизации, продиктованные результатами профилирования реализации алгоритма.

Результаты обучения сетей Asia и ALARM на наборах данных разного объёма реализованным алгоритмом приведены в таблицах 3.3 и 3.4

соответственно[1)]. Как видно из результатов, с увеличением количества данных улучшается качество извлеченной из данных сети. Стоит обратить внимание, что данный алгоритм потребовал априорных знаний о распределении, по которому были сгенерированы данные. Алгоритму на вход необходим определенный порядок вершин и знание максимального количества переменных-родителей для каждой переменной.

Таблица 3.3 – Качество структуры извлеченной из данных для сети Asia алгоритмом K2 из разработанной библиотеки

Размер данных	Соединения			Время построения
	пропущено	добавлено	инвертировано	
1000	1	1	0	00:00:00.01
2000	1	1	0	00:00:00.02
4000	1	0	0	00:00:00.04
8000	1	1	0	00:00:00.09
16 000	0	0	0	00:00:00.17
32 000	0	0	0	00:00:00.36
64 000	0	0	0	00:00:00.80
1 048 576	0	0	0	00:00:11.41

Таблица 3.4 – Качество структуры извлеченной из данных для сети ALARM алгоритмом K2 из разработанной библиотеки

Размер данных	Соединения			Время построения
	пропущено	добавлено	инвертировано	
1000	1	5	0	00:00:00.75
2000	1	1	0	00:00:00.72
4000	1	0	0	00:00:01.29
8000	1	2	0	00:00:02.45
16 000	0	2	0	00:00:04.83
32 000	0	1	0	00:00:09.48
64 000	0	1	0	00:00:18.61
128 000	0	1	0	00:00:36.58
1 048 576	0	1	0	00:04:57.18
8 388 608	0	1	0	00:44:13.00

Произведём сравнение реализованного алгоритма, с аналогичным алгоритмом реализованным в программе GeNIe, описанной в пункте 1.8.3 на странице 21. В таблице 3.5 приводятся полученные результаты.

[1)]Формат времени в колонке «Время построения» — часы:минуты:секунды.милисекунды

Таблица 3.5 – Качество структуры извлеченной из данных для сети Asia программой GeNIe с применением алгоритма Greedy Thick Thinning с оценкой K2

Размер данных	Соединения			Время построения
	пропущено	добавлено	инвертировано	
1000	2	2	2	*не измерялось*
2000	2	3	2	*не измерялось*
4000	1	1	3	*не измерялось*
8000	2	4	2	*не измерялось*
16 000	2	5	2	*не измерялось*
32 000	1	4	2	*не измерялось*
64 000	0	1	3	*не измерялось*
1 048 576	1	4	2	*не измерялось*

Для сравнения реализованных в библиотеке алгоритмов с теми, которые есть в GeNIe, были произведены дополнительные испытания последних. Для наиболее «точного» алгоритма реализованного в GeNIe в таблице 3.6 приводится оценка качества полученной структуры на наборах данных различного размера Для трех других алгоритмов в таблице 3.7 — на самом большом наборе данных. Как видно из полученных экспериментально данных, реализация алгоритма на основе оценки апостериорной вероятности из разработанной библиотеки ведет себя лучше как на малых объёмах данных, так и на больших, не смотря на то, что некоторые алгоритмы реализованные в GeNIe используют тот же метод оценки.

Таблица 3.6 – Качество структуры извлеченной из данных для сети Asia программой GeNIe с использованием алгоритма PC

Размер данных	Соединения			Время построения
	пропущено	добавлено	инвертировано	
1000	2	1	2	*не измерялось*
2000	3	0	0	*не измерялось*
4000	1	0	0	*не измерялось*
8000	2	0	0	*не измерялось*
16 000	3	1	3	*не измерялось*
32 000	2	0	0	*не измерялось*
64 000	1	0	0	*не измерялось*
1 048 576	1	0	0	*не измерялось*

Таблица 3.7 – Качество структуры извлеченной из данных для сети Asia программой GeNIe на наборе данных из 1 048 576 случаев

Алгоритм	Соединения		
	пропущено	добавлено	инвертировано
Bayesian Search	0	2	5
Essential Graph Search	5	0	2
Greedy Thick Thinning с оценкой BDeu	1	4	2

3.7 Алгоритм на основе оценки минимальной длины описания

Помимо алгоритма использующего оценку апостериорной вероятности в разработанной библиотеке был реализован алгоритм использующий оценку на основе принципа МДО. Описание принципа МДО приведено в подразделе 1.5 данной пояснительной записки. Т. к. способ подсчета оценки уже был здесь описан, то необходимо привести описание процедуры поиска. Процедура поиска в разработанной реализации алгоритма поиска вдохновлена процедурой поиска, использованной в работе [8].

Данный алгоритм нахождения структуры не требует предварительных знаний об истинном распределении, в отличие от алгоритма описанного в подразделе 3.6, что является существенным преимуществом на практике. Вместо использования априорных знаний, реализация алгоритма используем предварительные вычисления, извлекающие полезные данные о взаимозависимостях между переменным. Затем эта информация используется в стратегии поиска структуры.

В качестве оценки степени зависимости двух произвольных переменных в работе [26] было предложено использовать значение взаимной информации[1)]. Эта информация задаёт приоритет поиска зависимостей между переменными. По своей сути значение обоюдной информации является аналогом корреляции, но по своему содержанию — это оценка количества информации содержащейся в одной переменной о другой [8]. Значение взаимной информации принимает неотрицательные значения и равно нулю в случае независимости случайных величин. Для вычисления взаимной информации была предложена формула (3.4):

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\, p(y)} \right), \tag{3.4}$$

[1)] В англоязычной литературе используется термин mutual information

где $p(x, y)$ — совместное распределение случайных величин X и Y;
$p(y)$ — безусловное распределение случайной величины X;
$p(x)$ — безусловное распределение случайной величины Y.

На практике при вычислении $I(X;Y)$ следует соблюдать осторожность, т. к. относительные частоты, используемые для оценки $p(x)$, $p(y)$ и $p(x, y)$, могут необоснованно принимать значение 0 из-за отсутствия некоторых состояний переменных в наблюдаемых данных. В таких случаях вместо нуля используется достаточно малое число.

Таким образом алгоритм поиска состоит из следующих шагов:

а) Вычислить значения взаимной информации между всеми парами переменных и отсортировать список уникальных пар по убыванию взаимной информации. Пусть данный список обозначается символом $list$.

б) Алгоритм поиска начинается с извлечения двух пар переменных (X_{i1}, X_{i2}) и (X_{j1}, X_{j2}) из списка $list$ с максимальным значением взаимной информации. Затем среди всех возможных ацикличных моделей, построенных из переменных $X_{i1}, X_{i2}, X_{j1}, X_{j2}$ выбирается модель с наименьшей оценкой, вычисленной по формуле (1.3). Эта модель принимается за стартовую модель g_0

в) Пока список $list$ не пуст из него извлекается пара переменных (X_{k1}, X_{k2}) и строится множество новых моделей $\{g_0; g_0 \cup (X_{k1}, X_{k2}); g_0 \cup (X_{k2}, X_{k1})\}$. И этого множества удаляются циклические модели и выбирается модель с минимальной оценкой по формуле (1.3) и присваивается переменной g_0.

г) Когда список $list$ пуст, то поиск прекращается, модель g_0 считается оптимальной и рассматривается как структура вероятностной сети выведенная из данных. На практике можно завершить алгоритм раньше не дожидаясь пустоты $list$.

При реализации данного алгоритма важным моментом для повышения производительности является мемоизация значений функций $n[s, k, g]$ и $n[q, s, k, g]$, т. к. оценка длины описания считается для всей модели сразу, но между двумя последовательными итерациями разница между моделями составляем максимум одно соединение, т. е. для большинства вершин множество предков не меняется и значения $n[s, k, g]$ и $n[q, s, k, g]$ остаются неизменными и их можно не вычислять каждый раз. Так, для сети ALARM на экспериментальных данных из $1\,048\,576$ случаев, время построения сети сократилось с более чем четырех часов, до двух с половиной минут. Ниже приведены результаты качества обучения данного алгоритма для сети Asia, в

табилце 3.8, и для сети ALARM, в таблице 3.9.

Таблица 3.8 – Качество структуры извлеченной из данных для сети Asia алгоритмом из разработанной библиотеки, использующим оценку МДО

Размер данных	Соединения			Время построения
	пропущено	добавлено	инвертировано	
1000	2	1	0	00:00:00.26
2000	2	2	2	00:00:00.28
4000	1	2	2	00:00:00.47
8000	1	1	0	00:00:00.93
16 000	1	2	0	00:00:01.70
32 000	0	1	1	00:00:03.37
64 000	0	1	0	00:00:07.05
1 048 576	0	1	0	00:01:46.53

Таблица 3.9 – Качество структуры извлеченной из данных для сети ALARM алгоритмом из разработанной библиотеки, использующим оценку МДО

Размер данных	Соединения			Время построения
	пропущено	добавлено	инвертировано	
1000	6	3	11	00:00:01.78
2000	5	5	11	00:00:01.50
4000	3	4	11	00:00:01.84
8000	3	7	9	00:00:02.26
16 000	2	10	17	00:00:03.27
32 000	2	13	19	00:00:05.17
64 000	1	17	15	00:00:09.30
128 000	1	21	21	00:00:18.50
1 048 576	0	22	16	00:02:22.82

Как видно данный алгоритм работает хуже алгоритма из подраздела 3.6. По результатам измерений, приведенных в табилце 3.9 можно видеть, что с увеличением набора данных структура сети усложняется. Количество пропущенных связей уменьшается, но также растет количество лишних и инвертированных связей. На практике возможна доработка структуры сети с участием эксперта. Это займет меньше времени, чем разработка сети «с нуля», т. к. большинство связей и общая структура уже найдены. В защиту данного алгоритма можно сказать, что он работает без каких-либо априорных знаний о структуре сети и не ограничен максимальным количеством вершин-предков, как алгоритм из предыдущего подраздела.

Также можно заметить, что не смотря на то, что алгоритм проигрывает реализации алгоритма K2, качество обучаемой структуры в многих случаях лучше того, что может предоставить программа GeNIe. Сравнение с различными алгоритмами GeNIe приведено в таблице 3.10, необходимо отметить, что алгоритмы из GeNIe субъективно работают в разы медленнее разработанной библиотеки на данных из $1\,048\,576$ случаев для сети ALARM. Например, выполнение алгоритма Bayesian Search заняло более четырёх часов.

Таблица 3.10 – Качество структуры извлеченной из данных для сети ALARM программой GeNIe на наборе данных из $1\,048\,576$ случаев

Алгоритм	Соединения		
	пропущено	добавлено	инвертировано
Bayesian Search	8	66	21
PC	создал циклическую структуру		
Essential Graph Search	32	2	4
Greedy Thick Thinning с оценкой BDeu	0	26	24
Greedy Thick Thinning с оценкой K2	0	30	23

В процессе оценки результатов данного алгоритма было выявлено экспериментальным путём, что оценка на основе МДО довольная чувствительна к данным, т. е. имея два набора данных одинакового размера, сгенерированных одним распределением, можно получить немного разные сети из-за случайных различий в данных. В связи с этим была проделана следующая модификация в существующем алгоритме, которая позволила получать улучшенные сети на малых объёмах данных. Когда на вход алгоритму подается набор данных из n случаев, то алгоритм случайным образом отбрасывает из него небольшой процент данных и строит структуру сети на уменьшенном объёме данных. Далее указанная операция повторяется некоторое количество раз. В итоге получается некоторое множество сетей из которых выбирается лучшая, используя оценку МДО и исходный набор экспериментальных данных. Результаты работы модифицированного алгоритма приведены в таблицах 3.11 и 3.12. Как видно из результатов, в некоторых случаях результаты незначительно улучшились, в некоторых — ухудшились. С учетом времени работы алгоритмов вопрос о целесообразности данной модификации остается открытым.

Также, вероятно, стоит упомянуть что в разработанной библиотеке

Таблица 3.11 – Качество структуры извлеченной из данных для сети Asia модифицированным алгоритмом из разработанной библиотеки, использующим оценку МДО

Размер данных	Соединения			Время построения
	пропущено	добавлено	инвертировано	
1000	1	0	1	00:00:00.92
2000	1	0	0	00:00:00.36
4000	0	0	0	00:00:00.42
8000	0	0	1	00:00:00.67
16 000	1	2	0	00:00:01.44
32 000	0	1	0	00:00:02.25
64 000	0	1	0	00:00:03.90
1 048 576	0	1	0	00:01:06.87

Таблица 3.12 – Качество структуры извлеченной из данных для сети ALARM модифицированным алгоритмом из разработанной библиотеки, использующим оценку МДО

Размер данных	Соединения			Время построения
	пропущено	добавлено	инвертировано	
1000	7	6	10	00:00:11.21
2000	3	8	10	00:00:08.22
4000	3	8	15	00:00:09.89
8000	3	7	11	00:00:13.00
16 000	1	7	15	00:00:18.10
32 000	2	12	9	00:00:31.90
64 000	0	8	17	00:00:51.33
128 000	0	11	16	00:01:33.80
1 048 576	0	15	11	00:13:51.61

реализован еще один алгоритм, использующий оценку МДО, но с использованием другой стратегии поиска и другого способа вычисления длинны описания. Данный алгоритм и оценка были позаимствованы из работы [5]. Реализация данного алгоритма проявила себя хуже, чем предыдущие два рассмотренных алгоритма и поэтому детальная информация по данному алгоритму здесь не приводится.

В качестве промежуточного итога для данного раздела стоит отметить, что два известных алгоритма и одна модификация, реализованные в библиотеке, показывают результаты лучше как по качеству, так и по времени, чем все алгоритмы, представленные в бесплатной программе GeNIe. Из-

за лицензионных ограничений сравнить реализованные алгоритмы с другим коммерческим ПО не удалось.

4 ОХРАНА ТРУДА

4.1 Обеспечение пожарной безопасности на предприятии малого бизнеса «Техартгруп»

Целью дипломного проекта является реализация и анализ алгоритмов построения вероятностных сетей. Вероятностная сеть является компактным и эффективным способом представления знаний. Вероятностные сети используются в программном обеспечении для принятия решения в условиях недостаточной определенности. Данный способ статистического моделирования показал свою пригодность в реальных условиях в сложных предметных областях: медицине, космической промышленности, финансовой сфере и других областях. Первоначальные стадии разработки дипломного проекта выполнялись на предприятии ООО «Техартгруп» во время прохождения преддипломной практики. В настоящем разделе рассматриваются вопросы, связанные с обеспечением пожарной безопасности на предприятии.

Предприятие «Техартгруп» занимается предоставлением услуг по разработке информационных систем для иностранных предприятий. В минском офисе компании на данный момент работает более 200 человек. Большое количество конкурирующих компаний, разрабатывающих программное обеспечение в Минске, способствует повышению общего уровня условий труда. Это, в частности, сказывается на комфортабельности рабочих мест. Работникам предоставляются светлые, проветриваемые, тихие кабинеты, гибкий график рабочего времени, специальные комнаты отдыха и т. д. Современные компании негласно ориентируются на соответствие лучшим мировым практикам в области охраны труда и, в частности, пожарной безопасности.

На предприятии «Техартгруп» за пожарную безопасность отвечает директор компании. Для каждого нового сотрудника производится инструктаж по пожарной безопасности и технике безопасности, а так же знакомство с планом эвакуации при возникновении черезвычайных ситуаций [27, с. 324]. За проведение инструктажа отвечает специальный человек из отдела материально-технического снабжения предприятия. В компании действует набор правил, обязательных для исполнения сотрудниками. В целях повышения пожарной безопасности курение в здании офиса запрещено. Все сотрудники обязаны в конце рабочего дня выключить свои персональные компьютеры и обесточить их. В конце рабочего дня

специальный сотрудник проверяет соблюдение данного правила в каждом рабочем кабинете, чтобы там были выключены все электрические приборы: компьютеры, электрические чайники, кондиционеры, освещение и т. д. Все рабочие компьютеры подключены к источникам бесперебойного питания, которые подключены к сетевыми фильтрам, защищающим от скачков напряжения в электросети.

Офис компании расположен в центре города. Здание офиса представляет собой монолитную железобетонную конструкцию высотой шесть этажей, офис компании находится на двух верхних этажах. Конструкция здания предусматривает три способа эвакуации с этажа: выход в паркинг, лестничная клетка с выходом на улицу, лестничная клетка с выходом на первый этаж паркинга. В случае недоступности основных эвакуационных выходов из каждого кабинета можно через окно попасть на лоджию [27, с. 314 – 316]. Схемы эвакуации выдаются в виде электронного документа каждому новому сотруднику, а также находятся на специальном стенде в рабочих кабинетах. Все кабинеты офиса расположены вдоль длинного коридора, который оборудован специальными аварийными светильниками и знаками, указывающими направление эвакуации. На случай отключения электроэнергии компания имеет два дизельных-генератора, обеспечивающих нужды предприятия на случай отключения электроэнергии.

Офис компании оборудован необходимыми средствами сигнализации о пожаре [28, с. 215]. Каждый кабинет оборудован пожарным дымовым оптико-электрическим точечным извещателем ИП212-02М1 (рисунок 4.1). На предприятии производиться регулярный контроль и проверка работоспособности пожарных извещателей специальным человеком из отдела материально-технического снабжения предприятия. В коридорах дополнительно установлены ручные пожарные извещатели ИП 5-2Р (рисунок 4.1). Для извещения о пожаре также может быть использована корпоративная электронная почта, а также другие современные способы обмена информацией.

На случай возникновения пожара в каждом рабочем кабинете находиться ручной порошковый огнетушитель ОП-10 (з) МИГ М (рисунок 4.2), пригодный для тушения пожаров различного типа, в том числе для тушения электрических приборов [27, с. 221 – 323]. Каждый этаж здания офиса оборудован двумя пожарными кранами для тушения пожара. Пожарные краны расположены в противоположных частях коридора, недалеко от эвакуационных выходов (рисунок 4.2). На случай воспламенения электрической проводки или другого электрического

(а)

(б)

Рисунок 4.1 – а — автономный пожарный извещатель; б — ручной пожарный извещатель.

оборудования в каждом кабинете установлены электрические щитки, необходимые для отключения подачи электроэнергии в пределах кабинета. Во всех помещениях офиса предприятия установлена оросительная система пожаротушения для ликвидации возгорания до приезда пожарной службы [27, с. 318 – 320]. При расследовании возможных причин возникновения пожара может быть задействована система видео-наблюдения, установленная во всех помещениях предприятия.

(а)

(б)

Рисунок 4.2 – а — порошковый огнетушитель ОП-10 (з) МИГ М; б — пожарный кран.

Основной род деятельности на предприятии — разработка информационных систем — не предусматривает непосредственный контакт

с горючими или легко-воспламеняющимися веществами, что сильно снижает риски возникновения пожара на предприятии. Наиболее вероятными причинами возникновения пожара, с учетом специфики предприятия, могут являться нарушение правил внутреннего распорядка — курение на рабочем месте, и неисправность электрического оборудования, которого в офисе компании достаточно [27, с. 312]. С целью снижения риска возникновения пожара по причине неисправности электрического оборудования в компании запрещено пользоваться неисправным оборудованием, а все исправное оборудование подключается в сеть через специальные сетевые фильтры и источники бесперебойного питания. В целом правила распорядка на предприятии и высокая культура работы с электрическим оборудованием снижают риски возникновения пожара до минимума.

Большой проблемой в достижении максимальной пожарной безопасности предприятия является доступность подъезда пожарной техники к зданию офиса. В будние дни прилегающие улицы, стоянки, пешеходные переходы заняты неправильно припаркованным личным транспортом. Большую часть светлого времени суток движение по прилегающим улицам очень затруднено. Данную проблему предприятие не в силах решить самостоятельно, проблема заключается в низкой культуре владельцев транспорта и игнорировании многочисленных нарушений правил дорожного движения сотрудниками ГАИ. Таким образом, изложенные выше предложения, не смотря на проблемы с подъездом пожарной техники, обеспечат пожарную безопасность на предприятии «Техартгруп».

5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

Целью дипломного проекта является разработка и анализ алгоритмов построения вероятностных сетей. Вероятностные сети используются в ПО для принятия решения в условиях недостаточной определенности. Данный способ статистического моделирования показал свою пригодность в реальных условиях в сложных предметных областях: медицине, космической промышленности, финансовой сфере и других областях. Для достижения указанной цели планируется разработать ПО для представления и синтеза вероятностных сетей. Сеть синтезируется по набору данных и позволяет производить оценку параметров распределения случайных величин, характеризующих неизвестные данные.

5.1 Расчёт затрат, необходимых для создания ПО

Целесообразность создания коммерческого ПО требует проведения предварительной экономической оценки и расчета экономического эффекта. Экономический эффект у разработчика ПО зависит от объёма инвестиций в разработку проекта, цены на готовый программный продукт и количества проданный копий, и проявляется в виде роста чистой прибыли.

Оценка стоимости создания ПО со стороны разработчика предполагает составление сметы затрат, которая включает следующие статьи расходов:

– заработную плату исполнителей, основную ($З_о$) и дополнительную ($З_д$);
– отчисления в фонд социальной защиты населения ($З_{сз}$);
– налоги от фонда оплаты труда ($Н_е$);
– материалы и комплектующие (М);
– спецоборудование ($Р_с$);
– машинное время ($Р_м$);
– расходны на научные командировки ($Р_{нк}$);
– прочие прямые расходы ($П_з$);
– накладные расходы ($Р_н$);
– расходы на сопровождение и адаптацию ($Р_{са}$).

Исходные данные для разрабатываемого проекта указаны в таблице 5.1.

На основании сметы затрат и анализа рынка ПО определяется плановая отпускаемая цена. Для составления сметы затрат на создание ПО необходима предварительная оценка трудоемкости ПО и его объёма. Расчет объёма

Таблица 5.1 – Исходные данные

Наименование	Условное обозначение	Значение
Категория сложности		2
Коэффициент сложности, ед.	$К_с$	1,12
Степень использования при разработке стандартных модулей, ед.	$К_т$	0,7
Коэффициент новизны, ед.	$К_н$	0,7
Годовой эффективный фонд времени, дн.	$Ф_{эф}$	231
Продолжительность рабочего дня, ч.	$Т_ч$	8
Месячная тарифная ставка первого разряда, Br	$Т_{м1}$	600 000
Коэффициент премирования, ед.	К	1,5
Норматив дополнительной заработной платы, ед.	$Н_д$	20
Норматив отчислений в ФСЗН и обязательное страхование, %	$Н_{сз}$	34,5
Норматив командировочных расходов, %	$Н_к$	15
Норматив прочих затрат, %	$Н_{пз}$	20
Норматив накладных расходов, %	$Н_{рн}$	100
Прогнозируемый уровень рентабельности, %	$У_{рп}$	35
Норматив НДС, %	$Н_{дс}$	20
Норматив налога на прибыль, %	$Н_п$	18
Норматив расхода материалов, %	$Н_{мз}$	3
Норматив расхода машинного времени, ч.	$Н_{мв}$	4,5
Цена одного часа машинного времени, Br	$Н_{мв}$	5000
Норматив расходов на сопровождение и адаптацию ПО, %	$Н_{рса}$	30

программного продукта (количества строк исходного кода) предполагает определение типа программного обеспечения, всестороннее техническое обоснование функций ПО и определение объёма каждой функций. Согласно классификации типов программного обеспечения [29, с. 59, приложение 1], разрабатываемое ПО с наименьшей ошибкой можно классифицировать как ПО методо-ориентированных расчетов.

Общий объём программного продукта определяется исходя из

количества и объёма функций, реализованных в программе:

$$V_o = \sum_{i=1}^{n} V_i \,, \tag{5.1}$$

где V_i — объём отдельной функции ПО, LoC;
n — общее число функций.

На стадии технико-экономического обоснования проекта рассчитать точный объём функций невозможно. Вместо вычисления точного объёма функций применяются приблизительные оценки на основе данных по аналогичным проектам или по нормативам [29, с. 61, приложение 2], которые приняты в организации.

Таблица 5.2 – Перечень и объём функций программного модуля

№ функции	Наименование (содержание)	Объём функции, LoC	
		по каталогу (V_i)	уточненный ($V_i^{\text{у}}$)
101	Организация ввода информации	100	60
102	Контроль, предварительная обработка и ввод информации	520	520
111	Управление вводом/выводом	2700	700
304	Обслуживание файлов	520	580
305	Обработка файлов	750	750
309	Формирование файла	1100	1100
506	Обработка ошибочных и сбойных ситуаций	430	430
507	Обеспечение интерфейса между компонентами	730	730
605	Вспомогательные и сервисные программы	460	280
701	Математическая статистика и прогнозирование	8370	3500
Итог		15 680	8650

Каталог аналогов программного обеспечения предназначен для предварительной оценки объёма ПО методом структурной аналогии. В разных организациях в зависимости от технических и организационных условий, в которых разрабатывается ПО, предварительные оценки могут

корректироваться на основе экспертных оценок. Уточненный объём ПО рассчитывается по формуле:

$$V_{\text{у}} = \sum_{i=1}^{n} V_i^{\text{у}}, \tag{5.2}$$

где $V_i^{\text{у}}$ — уточненный объём отдельной функции ПО, LoC;
n — общее число функций.

Перечень и объём функций программного модуля перечислен в таблице 5.2. По приведенным данным уточненный объём некоторых функций изменился, и общий объём ПО составил $V_o = 15\,680$ LoC, общий уточненный общем ПО — $V_{\text{у}} = 8650$ LoC.

По уточненному объёму ПО и нормативам затрат труда в расчете на единицу объёма определяются нормативная и общая трудоемкость разработки ПО. Уточненный объём ПО — 8650 LoC. ПО относится ко второй категории сложности: предполагается его использование для сложных статистических расчетов и решения задач классификации, также необходимо обеспечить переносимость ПО [29, с. 66, приложение 4, таблица П.4.1]. По полученным данным определяется нормативная трудоемкость разработки ПО. Согласно укрупненным нормам времени на разработку ПО в зависимости от уточненного объёма ПО и группы сложности ПО [29, с. 64, приложение 3] нормативная трудоемкость разрабатываемого проекта составляет $\text{Т}_{\text{н}} = 224$ чел./дн.

Нормативная трудоемкость служит основой для оценки общей трудоемкости $\text{Т}_{\text{о}}$. Используем формулу (5.3) для оценки общей трудоемкости для небольших проектов:

$$\text{Т}_{\text{о}} = \text{Т}_{\text{н}} \cdot \text{К}_{\text{с}} \cdot \text{К}_{\text{т}} \cdot \text{К}_{\text{н}}, \tag{5.3}$$

где $\text{К}_{\text{с}}$ — коэффициент, учитывающий сложность ПО;
$\text{К}_{\text{т}}$ — поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;
$\text{К}_{\text{н}}$ — коэффициент, учитывающий степень новизны ПО.

Дополнительные затраты труда на разработку ПО учитываются через

коэффициент сложности, который вычисляется по формуле

$$К_с = 1 + \sum_{i=1}^{n} К_i \,, \tag{5.4}$$

где $К_i$ — коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;
n — количество учитываемых характеристик.

Наличие двух характеристик сложности позволяет [29, с. 66, приложение 4, таблица П.4.2] вычислить коэффициент сложности

$$К_с = 1 + 0{,}12 = 1{,}12 \,. \tag{5.5}$$

Разрабатываемое ПО использует стандартные компоненты. Степень использования стандартных компонентов определяется коэффициентом использования стандартных модулей — $К_т$. Согласно справочным данным [29, с. 68, приложение 4, таблица П.4.5] указанный коэффициент для разрабатываемого приложения $К_т = 0{,}7$. Трудоемкость создания ПО также зависит от его новизны и наличия аналогов. Разрабатываемое ПО не является новым, существуют аналогичные более зрелые разработки у различных компаний и университетов по всему миру. Влияние степени новизны на трудоемкость создания ПО определяется коэффициентом новизны — $К_н$. Согласно справочным данным [29, с. 67, приложение 4, таблица П.4.4] для разрабатываемого ПО $К_н = 0{,}7$. Подставив приведенные выше коэффициенты для разрабатываемого ПО в формулу (5.3) получим общую трудоемкость разработки

$$Т_о = 224 \times 1{,}12 \times 0{,}7 \times 0{,}7 \approx 123 \,\text{чел./дн.} \tag{5.6}$$

На основе общей трудоемкости и требуемых сроков реализации проекта вычисляется плановое количество исполнителей. Численность исполнителей проекта рассчитывается по формуле:

$$Ч_р = \frac{Т_о}{Т_р \cdot Ф_{эф}} \,, \tag{5.7}$$

где $T_о$ — общая трудоемкость разработки проекта, чел./дн.;
$Ф_{эф}$ — эффективный фонд времени работы одного работника в течение года, дн.;
$T_р$ — срок разработки проекта, лет.

Эффективный фонд времени работы одного разработчика вычисляется по формуле

$$Ф_{эф} = Д_г - Д_п - Д_в - Д_о\,, \tag{5.8}$$

где $Д_г$ — количество дней в году, дн.;
$Д_п$ — количество праздничных дней в году, не совпадающих с выходными днями, дн.;
$Д_в$ — количество выходных дней в году, дн.;
$Д_п$ — количество дней отпуска, дн.

Согласно данным, приведенным в производственном календаре для пятидневной рабочей недели в 2013 году для Беларуси [30], фонд рабочего времени составит

$$Ф_{эф} = 365 - 9 - 104 - 21 = 231 \text{ дн.} \tag{5.9}$$

Учитывая срок разработки проекта $T_р = 3 \text{ мес.} = 0{,}25 \text{ года}$, общую трудоемкость и фонд эффективного времени одного работника, вычисленные ранее, можем рассчитать численность исполнителей проекта

$$Ч_р = \frac{123}{0{,}25 \times 231} \approx 2 \text{ рабочих.} \tag{5.10}$$

Вычисленные оценки показывают, что для выполнения запланированного проекта в указанные сроки необходимо два рабочих. Информация о работниках перечислена в таблице 5.3.

Таблица 5.3 – Работники, занятые в проекте

Исполнители	Разряд	Тарифный коэффициент	Чел./дн. занятости
Программист I-категории	13	3,04	61
Ведущий программист	15	3,48	62

Месячная тарифная ставка одного работника вычисляется по формуле

$$T_ч = \frac{T_{м_1} \cdot T_к}{Ф_р}\,, \tag{5.11}$$

где $Т_{м_1}$ — месячная тарифная ставка 1-го разряда, Br;
$Т_к$ — тарифный коэффициент, соответствующий установленному тарифному разряду;
$Ф_р$ — среднемесячная норма рабочего времени, час.

Подставив данные из таблицы 5.3 в формулу (5.11), приняв значение тарифной ставки 1-го разряда $Т_{м_1} = 600\,000$ Br и среднемесячную норму рабочего времени $Ф_р = 160$ часов получаем

$$Т_ч^{\text{прогр. I-разр.}} = \frac{600\,000 \times 3{,}04}{160} = 11\,400\,\text{Br/час}; \tag{5.12}$$

$$Т_ч^{\text{вед. прогр.}} = \frac{600\,000 \times 3{,}48}{160} = 13\,050\,\text{Br/час}. \tag{5.13}$$

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле

$$З_о = \sum_{i=1}^{n} Т_ч^i \cdot Т_ч \cdot Ф_п \cdot К\,, \tag{5.14}$$

где $Т_ч^i$ — часовая тарифная ставка i-го исполнителя, Br/час;
$Т_ч$ — количество часов работы в день, час;
$Ф_п$ — плановый фонд рабочего времени i-го исполнителя, дн.;
К — коэффициент премирования.

Подставив ранее вычисленные значения и данные из таблицы 5.3 в формулу (5.14) и приняв коэффициент премирования $К = 1{,}5$ получим

$$З_о = (11400 \times 61 + 13050 \times 62) \times 8 \times 1{,}5 = 18\,054\,000\,\text{Br}\,. \tag{5.15}$$

Дополнительная заработная плата включает выплаты предусмотренные законодательством от труде и определяется по нормативу в процентах от основной заработной платы

$$З_д = \frac{З_о \cdot Н_д}{100\%}\,, \tag{5.16}$$

где $Н_д$ — норматив дополнительной заработной платы, %.

Приняв норматив дополнительной заработной платы $Н_д = 20\%$ и

подставив известные данные в формулу (5.16) получим

$$З_д = \frac{18\,054\,000 \times 20\%}{100\%} \approx 3\,610\,800\,\text{Br}. \tag{5.17}$$

Согласно действующему законодательству отчисления в фонд социальной защиты населения составляют 34% , в фонд обязательного страхования — $0{,}5\%$, от фонда основной и дополнительной заработной платы исполнителей. Общие отчисления на социальную защиту рассчитываются по формуле

$$З_{сз} = \frac{(З_о + З_д) \cdot Н_{сз}}{100\%}. \tag{5.18}$$

Подставив вычисленные ранее значения в формулу (5.18) получаем

$$З_{сз} = \frac{(18\,054\,000 + 3\,610\,800) \times 34{,}5\%}{100\%} \approx 7\,474\,356\,\text{Br}. \tag{5.19}$$

По статье «материалы» проходят расходы на носители информации, бумагу, краску для принтеров и другие материалы, используемые при разработке ПО. Норма расходов $Н_{мз}$ определяется либо в расчете на 100 строк исходного кода, либо в процентах к основной зарплате исполнителей $3\% — 5\%$. Затраты на материалы вычисляются по формуле

$$М = \frac{З_о \cdot Н_{мз}}{100\%} = \frac{18\,054\,000 \times 3\%}{100\%} \approx 541\,620\,\text{Br}. \tag{5.20}$$

Расходы по статье «машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО, которое определяется по нормативам в машино-часах на 100 строк исходного кода в зависимости от характера решаемых задач и типа ПК, и вычисляются по формуле

$$Р_м = Ц_м \cdot \frac{V_о}{100} \cdot Н_{мв}, \tag{5.21}$$

где $Ц_м$ — цена одного часа машинного времени, Br;
$Н_{мв}$ — норматив расхода машинного времени на отладку 100 строк исходного кода, часов.

Согласно нормативу [29, с. 69, приложение 6] норматив расхода машинного времени на отладку 100 строк исходного кода составляет $Н_{мв} = 15$, применяя понижающий коэффициент $0{,}3$ получаем $Н'_{мв} = 4{,}5$. Цена одного часа машинного времени составляет $Ц_м = 5000\,\text{Br}$. Подставляя

известные данные в формулу (5.21) получаем

$$Р_м = 5000 \times \frac{8650}{100} \times 4{,}5 = 1\,946\,250\,\text{Br}. \qquad (5.22)$$

Расходы по статье «научные командировки» вычисляются как процент от основной заработной платы, либо определяются по нормативу. Вычисления производятся по формуле

$$Р_к = \frac{З_о \cdot Н_к}{100\%}, \qquad (5.23)$$

где $Н_к$ — норматив командировочных расходов по отношению к основной заработной плате, %.

Подставляя ранее вычисленные значения в формулу (5.23) и приняв значение $Н_к = 15\%$ получаем

$$Р_к = \frac{18\,054\,000 \times 15\%}{100\%} = 2\,708\,100\,\text{Br}. \qquad (5.24)$$

Статья расходов «прочие затраты» включает в себя расходы на приобретение и подготовку специальной научно-технической информации и специальной литературы. Затраты определяются по нормативу принятому в организации в процентах от основной заработной платы и вычисляются по формуле

$$П_з = \frac{З_о \cdot Н_{пз}}{100\%}, \qquad (5.25)$$

где $Н_{пз}$ — норматив прочих затрат в целом по организации, %.

Приняв значение норматива прочих затрат $Н_{пз} = 20\%$ и подставив вычисленные ранее значения в формулу (5.25) получаем

$$П_з = \frac{18\,054\,000 \times 20\%}{100\%} = 3\,610\,800\,\text{Br}. \qquad (5.26)$$

Статья «накладные расходы» учитывает расходы, необходимые для содержания аппарата управления, вспомогательных хозяйств и опытных производств, а также расходы на общехозяйственные нужны. Данная статья затрат рассчитывается по нормативу от основной заработной платы и вычисляется по формуле.

$$Р_н = \frac{З_о \cdot Н_{рн}}{100\%}, \tag{5.27}$$

где $Н_{рн}$ — норматив накладных расходов в организации, %.

Приняв норму накладных расходов $Н_{рн} = 100\%$ и подставив известные данные в формулу (5.27) получаем

$$Р_н = \frac{18\,054\,000 \times 100\%}{100\%} = 18\,054\,000\,\text{Br}. \tag{5.28}$$

Общая сумма расходов по смете на ПО рассчитывается по формуле

$$С_р = З_о + З_д + З_{сз} + М + Р_м + Р_{нк} + П_з + Р_н. \tag{5.29}$$

Подставляя ранее вычисленные значения в формулу (5.29) получаем

$$С_р = 55\,999\,926\,\text{Br}. \tag{5.30}$$

Расходы на сопровождение и адаптацию, которые несет производитель ПО, вычисляются по нормативу от суммы расходов по смете и рассчитываются по формуле

$$Р_{са} = \frac{С_р \cdot Н_{рса}}{100\%}, \tag{5.31}$$

где $Н_{рса}$ — норматив расходов на сопровождение и адаптацию ПО, %.

Приняв значение норматива расходов на сопровождение и адаптацию $Н_{рса} = 30\%$ и подставив ранее вычисленные значения в формулу (5.31) получаем

$$Р_{са} = \frac{55\,999\,926 \times 30\%}{100\%} \approx 16\,799\,978\,\text{Br}. \tag{5.32}$$

Полная себестоимость создания ПО включает сумму затрат на разработку, сопровождение и адаптацию и вычисляется по формуле

$$С_п = С_р + Р_{са}. \tag{5.33}$$

Подставляя известные значения в формулу (5.33) получаем

$$С_п = 55\,999\,926 + 16\,799\,978 = 72\,799\,904\,\text{Br}. \tag{5.34}$$

5.2 Расчёт экономической эффективности у разработчика

Важная задача при выборе проекта для финансирования это расчет экономической эффективности проектов и выбор наиболее выгодного проекта. Разрабатываемое ПО является заказным, т. е. разрабатывается для одного заказчика на заказ. На основании анализа рыночных условий и договоренности с заказчиком об отпускной цене прогнозируемая рентабельность проекта составит $У_{рп} = 35\%$. Прибыль рассчитывается по формуле

$$П_с = \frac{С_п \cdot У_{рп}}{100\%}, \tag{5.35}$$

где $П_с$ — прибыль от реализации ПО заказчику, Br;
$У_{рп}$ — уровень рентабельности ПО, %.

Подставив известные данные в формулу (5.35) получаем прогнозируемую прибыль от реализации ПО

$$П_с = \frac{72\,799\,904 \times 35\%}{100\%} \approx 25\,479\,966\,\text{Br}. \tag{5.36}$$

Прогнозируемая цена ПО без учета налогов включаемых в цену вычисляется по формуле

$$Ц_п = С_п + П_с. \tag{5.37}$$

Подставив данные в формулу (5.37) получаем цену ПО без налогов

$$Ц_п = 72\,799\,904 + 25\,479\,966 = 98\,279\,870\,\text{Br}. \tag{5.38}$$

Налог на добавленную стоимость рассчитывается по формуле

$$НДС = \frac{Ц_п \cdot Н_{дс}}{100\%}, \tag{5.39}$$

где $Н_{дс}$ — норматив НДС, %.

Норматив НДС составляет $Н_{дс} = 20\%$, подставляя известные значения в формулу (5.39) получаем

$$НДС = \frac{98\,279\,870 \times 20\%}{100\%} \approx 19\,655\,974\,\text{Br}. \tag{5.40}$$

Расчет прогнозируемой отпускной цены осуществляется по формуле

$$Ц_о = Ц_п + НДС\,. \tag{5.41}$$

Подставляя известные данные в формулу (5.41) получаем прогнозируемую отпускную цену

$$Ц_о = 98\,279\,870 + 19\,655\,974 \approx 117\,935\,844\,\text{Br}\,. \tag{5.42}$$

Чистую прибыль от реализации проекта можно рассчитать по формуле

$$П_ч = П_с \cdot \left(1 - \frac{Н_п}{100\%}\right)\,, \tag{5.43}$$

где $Н_п$ — величина налога на прибыль, %.

Приняв значение налога на прибыль $Н_н = 18\%$ и подставив известные данные в формулу (5.43) получаем чистую прибыль

$$П_ч = 25\,479\,966 \times \left(1 - \frac{18\%}{100\%}\right) = 20\,893\,572\,\text{Br}\,. \tag{5.44}$$

Программное обеспечение разрабатывалось для одного заказчика в связи с этим экономическим эффектом разработчика будет являться чистая прибыль от реализации $П_ч$. Рассчитанные данные приведены в таблице 5.4. Таким образом было произведено технико-экономическое обоснование разрабатываемого проекта, составлена смета затрат и рассчитана прогнозируемая прибыль, и показана экономическая целесообразность разработки.

Таблица 5.4 – Рассчитанные данные

Наименование	Условное обозначение	Значение
Нормативная трудоемкость, чел./дн.	$Т_н$	224
Общая трудоемкость разработки, чел./дн.	$Т_о$	123
Численность исполнителей, чел.	$Ч_р$	2
Часовая тарифная ставка программиста I-разряда, Br/ч.	$Т_ч^{прогр.\ I\text{-}разр.}$	11 400
Часовая тарифная ставка ведущего программиста, Br/ч.	$Т_ч^{вед.\ прогр.}$	13 050
Основная заработная плата, Br	$З_о$	18 054 000
Дополнительная заработная плата, Br	$З_д$	3 610 800
Отчисления в фонд социальной защиты, Br	$З_{сз}$	7 474 356
Затраты на материалы, Br	М	541 620
Расходы на машинное время, Br	$Р_м$	1 946 250
Расходы на командировки, Br	$Р_к$	2 708 100
Прочие затраты, Br	$П_з$	3 610 800
Накладные расходы, Br	$Р_н$	18 054 000
Общая сумма расходов по смете, Br	$С_р$	55 999 926
Расходы на сопровождение и адаптацию, Br	$Р_{са}$	16 799 978
Полная себестоимость, Br	$С_п$	72 799 904
Прогнозируемая прибыль, Br	$П_с$	25 479 966
НДС, Br	НДС	19 655 974
Прогнозируемая отпускная цена ПО, Br	$Ц_о$	117 935 844
Чистая прибыль, Br	$П_ч$	20 893 572

ЗАКЛЮЧЕНИЕ

В данном дипломном проекте был рассмотрен вопрос автоматического построения структуры вероятностной сети на основе экспериментальных данных. В рамках дипломного проекта была разработана библиотека кода для представления и автоматического построения структуры сети. В разработанной библиотеке использовались два различных подхода к оценке качества сети, на основе принципа МДО и оценке апостериорной вероятности структуры для имеющихся экспериментальных данных. Также для разных оценок использовались разные стратегии поиска оптимальной структуры сети в пространстве возможных решений.

В целом были получены удовлетворительные результаты на хорошо изученных и известных сетях Asia и ALARM. Результаты работы реализованных в библиотеке функций поиска в большинстве случаях превосходят по качеству функциональность уже существующего программного обеспечения. Также был предложен способ улучшения качества обучаемой сети на малом объеме данных, основанный на предварительной рандомизации экспериментальных данных. Данный способ удовлетворительно зарекомендовал себя в проведённых тестах. Помимо предложенной модификации были произведены небольшие улучшения в хорошо известных алгоритмах, направленные на повышение скорости их работы. Для повышения производительности применялась мемоизация и использовались прологарифмированные версии некоторых оценок.

В результате цель дипломного проекта была достигнута. Было создано программное обеспечение. Но за рамками рассматриваемой темы осталось еще много других алгоритмов вывода структуры и интересных вопросов, связанных, например, со статистическим выводом суждений в вероятностных сетях, нахождением параметров распределения и других вопросов, возникающих при работе с вероятностными сетями. Эти задачи также являются нетривиальными и требуют детального изучения и проработки — задача статистического вывода, например, является $\mathcal{NP}$-трудной [31] — и не рассматриваются в данном дипломном проекте из-за временных ограничений на его создание. В дальнейшем планируется развивать и довести существующее ПО до полноценной библиотеки, способной решать более широкий класс задач, возникающих в области применения вероятностных сетей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Терехов, С. А. Введение в байесовы сети / С. А. Терехов // Научная сессия МИФИ–2003. V Всероссийская научно-техническая Конференция «Нейроинформатика–2003»: Лекции по нейроинформатике. Часть 1. — М.: МИФИ, 2003. — 188 с.

[2] Chickering, David Maxwell. Learning Bayesian Networks is NP-Complete. — 1996.

[3] Robinson, R. W. Counting unlabeled acyclic digraphs / R. W. Robinson // Combinatorial Mathematics V / Ed. by CharlesH Little. — Springer Berlin Heidelberg, 1977. — Vol. 622 of Lecture Notes in Mathematics. — Pp. 28–43. http://dx.doi.org/10.1007/bfb0069178.

[4] Stanford University. Probabilistic Graphical Models [Электронный ресурс]. — Электронные данные. — Режим доступа: https://www.coursera.org/course/pgm. — Дата доступа: 08.05.2013.

[5] Lam, Wai. Learning Bayesian belief networks: An approach based on the MDL principle / Wai Lam, Fahiem Bacchus // Computational Intelligence. — 1994. — Vol. 10. — Pp. 269–293.

[6] Suzuki, Joe. A Construction of Bayesian Networks from Databases Based on an MDL Principle / Joe Suzuki // Proceedings of the Ninth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-93). — San Francisco, CA: Morgan Kaufmann, 1993. — Pp. 266–273.

[7] Cooper, Gregory F. A Bayesian method for constructing Bayesian belief networks from databases / Gregory F. Cooper, Edward Herskovits // Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence. — UAI'91. — San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991. — Pp. 86–94. http://dl.acm.org/citation.cfm?id=2100662.2100674.

[8] Терентьев, А. Н. Эвристический метод построения байесовых сетей / А. Н. Терентьев, П. И. Бидюк // Математические машины и системы. — 2006. — № 3.

[9] Grünwald, Peter. A Tutorial Introduction to the Minimum Description Length Principle / Peter Grünwald // Advances in Minimum Description Length: Theory and Applications. — MIT Press, 2005.

[10] Rebane, George. The Recovery of Causal Poly-Trees From Statistical Data / George Rebane, Judea Pearl // Uncertainty in Artificial Intelligence 3 Annual Conference on Uncertainty in Artificial Intelligence (UAI-87). — Amsterdam, NL: Elsevier Science, 1987. — Pp. 175–182.

[11] Макконнелл, С. Совершенный код. Мастер-класс / Пер. с англ. /

С. Макконнелл. — СПб. : Издательско-торговый дом «Русская редакция», 2005. — 896 с.

[12] Research, Microsoft. Draft F# Component Design Guidelines. — 2010. — April.

[13] Common Language Infrastructure (CLI). Partitions I to VI. — 2012. — June. http://www.ecma-international.org/publications/standards/Ecma-335.htm.

[14] Рихтер, Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. — 2-е изд. — СПб. : Питер, Русская Редакция, 2007. — 656 с.

[15] Марченко, А. Л. Основы программирования на C# 2.0 / А. Л. Марченко. — БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий — ИНТУИТ.ру, 2007. — 552 с.

[16] Richter, Jeffrey. CLR via C# / Jeffrey Richter. Microsoft, Developer Reference. — 4-th edition. — One Microsoft Way, Redmond, Washington 98052-6399 : Microsoft Press, 2012. — 896 P.

[17] Абельсон, Харольд. Структура и интерпретация компьютерных программ / Харольд Абельсон, Джеральд Джей Сассман, Джули Сассман. — Добросвет, 2006. — 608 с.

[18] Albahari, Joseph. C# 5.0 in a Nutshell / Joseph Albahari, Ben Albahari. — 5-th edition. — O'Reilly Media, Inc, 2012. — June. — 1062 P.

[19] Волосевич, А. А. Язык C# и основы платформы .NET: Учебно-метод. пособие по курсу «Избранные главы информатики» для студ. спец. I-31 03 04 «Информатика» всех форм обуч. / А. А. Волосевич. — Минск , 2006. — 60 с.

[20] C Sharp [Электронный ресурс]. — Электронные данные. — Режим доступа: http://ru.wikipedia.org/wiki/C_Sharp. — Дата доступа: 22.03.2013.

[21] Лазин, Е. Введение в F# / Е. Лазин, М. Моисеев, Д. Сорокин // Практика функционального программирования. — 2010. — № 5. — 149 с.

[22] Harrison, Jonh. Introduction to Functional Programming / Jonh Harrison. — University of Cambridge , 1997. — December. — 168 P.

[23] Кирпичёв, Е. Элементы функциональных языков / Е. Кирпичёв // Практика функционального программирования. — 2009. — Декабрь. — № 3. — 196 с.

[24] Lauritzen, S. L. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems / S. L. Lauritzen, D. J. Spiegelhalter // Journal of the Royal Statistical Society. Series B (Methodological). — 1988. — Vol. 50, no. 2. http://dx.doi.org/10.2307/2345762.

[25] The ALARM Monitoring System: A Case Study with Two Probabilistic

Inference Techniques for Belief Networks / I. A. Beinlich, H. J. Suermondt, R. M. Chavez, G. F. Cooper // Second European Conference on Artificial Intelligence in Medicine / Ed. by J. Hunter, J. Cookson, J. Wyatt. — Vol. 38. — Berlin, Germany: Springer-Verlag, 1989. — Pp. 247–256.

[26] Chow, C. I. Approximating discrete probability distributions with dependence trees / C. I. Chow, Sexior Member, C. N. Liu // IEEE Transactions on Information Theory. — 1968. — Vol. 14. — Pp. 462–467.

[27] Михнюк, Т. Ф. Охрана труда : учебник [утв. МО РБ] / Т. Ф. Михнюк. — Минск : ИВЦ Минфина, 2009. — 345 с.

[28] Синилов, В. Г. Системы охранной, пожарной и охранно-пожарной сигнализации : учебник для нач. проф. образования / В. Г. Синилов. — 5-е изд. — М. : Издательский центр «Академия», 2010. — 512 с.

[29] Палицын, В. А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В. А. Палицын. — Минск : БГУИР, 2006. — 76 с.

[30] Календарь праздников на 2013 год для Беларуси [Электронный ресурс]. — Электронные данные. — Режим доступа: http://calendar.by/2013/#bkm. — Дата доступа: 05.03.2013.

[31] Koller, D. Probabilistic Graphical Models: Principles and Techniques / D. Koller, N. Friedman. — MIT Press, 2009. — 1270 P.