ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Брянский государственный технический университет

П.В. Казаков, В.А. Шкаберин

ОСНОВЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Утверждено редакционно-издательским советом в качестве учебного пособия

БРЯНСК
ИЗДАТЕЛЬСТВО БГТУ
2007

УДК 004.896

Казаков, П.В. Основы искусственного интеллекта: учеб. пособие / П.В. Казаков, В.А. Шкаберин. - Брянск: БГТУ, 2007. – 196 с. – (Сер. Информационные системы и технологии).

ISBN 5-89838-302-6

Рассматриваются основные методы научного направления «Искусственный интеллект». Приводятся их теоретические основы, а также особенности реализации. Описываются задачи, для которых применение методов искусственного интеллекта наиболее эффективно.

Учебное пособие предназначено для студентов всех форм обучения по направлениям «Информатика и вычислительная техника», «Информационные системы», а также специалистов, интересующихся новыми информационными технологиями.

Ил. - 83. Табл. – 21. Библиогр. – 14 назв.

Научный редактор
Заслуженный деятель науки РФ,
доктор технических наук, профессор В.И. Аверченков

Рецензенты: кафедра «Информационные системы» Орловского государственного технического университета; доктор технических наук, профессор В. Т. Еременко

ISBN 5-89838-302-6

ПРЕДИСЛОВИЕ

Искусственный интеллект является научным направлением, задачи которого связаны с разработкой методов моделирования отдельных функций интеллекта человека с помощью программно-аппаратных средств ЭВМ. Слово «интеллект» употребляется в различных смыслах и, хотя каждый из его обладателей имеет различное субъектное мнение, что следует понимать под человеческим интеллектом, можно утверждать, что одним из главных его проявлений является способность адекватно реагировать на любую, особенно новую, ситуацию путем корректировки поведения на основе имеющегося опыта. Однако реализация этого с помощью имеющихся стандартных вычислительных методов проблематична или невозможна в принципе. Причина этого - свойственный ЭВМ алгоритмический, строго детерминированный подход к обработке информации, часто ограничивающий ее использование на уровне мощного калькулятора с богатыми возможностями визуализации информации при решении задач количественного характера.

В то же время в таких областях, как автоматизация проектирования, автоматическое управление, социально-экономические системы и др. преобладает информация, представленная наборами фактов, гипотез, правил и закономерностей, сформулированных на качественном уровне. При этом задачи, описанные количественными и качественными признаками с преобладанием последних, относятся к слабоструктурированным проблемам и их решение основывается именно на методах искусственного интеллекта. Среди таких подходов, применяемых различными исследователями, можно выделить эвристическое программирование, искусственные нейронные сети, эволюционное моделирование, нечеткое моделирование, представление и обработка знаний и др. В настоящем учебном пособии рассматриваются эти методы искусственного интеллекта, а также особенности их применения при решении ряда практических задач. Знакомство с ними, с одной стороны, необходимо для становления специалиста в области новых информационных технологий, с другой – позволит расширить круг пользователей интеллектуального программного обеспечения.

Учебное пособие состоит из шести глав.

Первая глава знакомит с основными понятиями искусственного интеллекта, а также исторически сложившимися подходами к проведению исследований в этом научном направлении. Рассматриваются возможности, а также области применения искусственного интеллекта.

Вторая глава посвящена методам эвристического программирования при решении задач искусственного интеллекта. Рассматривается ключевое понятие эвристики, а также его роль при решении подобных задач. Приводятся постановка и принципы решения двух главных задач эвристического программирования – поиска в пространстве состояний и пространстве задач. Также глава знакомит с особенностями применения принципов эвристического поиска в игровых моделях, используемых во многих прикладных областях.

Третья глава содержит описание моделей представления знаний в информационных системах. Особое внимание уделяется классическим моделям представления знаний: логической, продукционной, фреймовой, на основе семантической сети. Рассматриваются различные ситуации, где каждая из моделей может быть использована. Здесь также приводятся основы формализации нечеткости в знаниях с применением теории нечетких множеств. В заключение главы отмечаются особенности систематизации знаний на основе онтологий.

Четвертая глава содержит сведения о принципах функционировании и разработки искусственных нейронных сетей, позволяющих моделировать мыслительные процессы человека. Отмечаются задачи, где применение этой технологии наиболее эффективно, отличия в организации и функционировании нейрокомпьютеров от традиционных ЭВМ. Приводится формализация структуры искусственного нейрона, а также особенности объединения их в слои и нейронные сети. Особое внимание уделяется методам обучения нейронных сетей, аспектам, связанным с повышением эффективности их использования. Также рассматриваются модификации моделей нейронных сетей и методов их обучения.

Пятая глава посвящена одному из перспективных направлений в области искусственного интеллекта, связанному с моделированием эволюционных процессов в различных системах. В частности, благодаря своей эффективности методы эволюционного моделирования

получили распространение при решении оптимизационных задач, характеризующихся большой размерностью и сложным для анализа и применения традиционных методов пространством поиска. Рассматриваются такие эволюционные алгоритмы, как генетический алгоритм, генетическое программирование, эволюционное программирование, эволюционные стратегии. Приводится описание используемых в них структур данных и операторов.

Шестая глава знакомит с принципами разработки и применения экспертных систем. Рассматриваются основные понятия, типовая структура экспертных систем, а также их классификация. Особое внимание уделяется проектированию экспертных систем на основе вероятностной модели вывода. Также приводятся особенности создания экспертных систем на основе байесовских сетей доверия. Их применение позволяет учитывать вероятностные причинно-следственные связи между понятиями предметной области и как следствие порождать более сложные умозаключения.

Излагаемый материал иллюстрируется примерами. В конце глав помещены вопросы для самоконтроля.

1. ВВЕДЕНИЕ В ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Искусственный интеллект (ИИ) является одним из приоритетных направлений в современной информатике, связанным с созданием следующей ступени ее развития – новых информационных технологий. Их цель – свести к минимуму участие человека как программиста при создании информационных систем, но привлекать его в качестве учителя, партнера человеко-машинной системы. Однако нельзя понимать термин «искусственный интеллект» буквально. Правильнее его воспринимать как некоторое метафорическое наименование совокупности методов, реализация которых на компьютере позволяет получать результаты близкие к порождаемым человеческим мышлением.

1.1. Некоторые понятия искусственного интеллекта

Идея создания искусственного интеллекта связана с постоянным стремлением человека переложить решение сложных задач на механического, затем электронного помощника. Единственный способ реализовать это заключается в моделировании с помощью различных средств интеллектуальных способностей человека.

Здесь под интеллектом следует понимать способность мозга решать задачи путем приобретения, запоминания и целенаправленного преобразования знаний в процессе обучения на опыте и адаптации к разнообразным условиям [6].

Искусственный интеллект (Artificial Intelligence, AI) как научное направление существует с 1956 года, когда британский математик Алан Тьюринг опубликовал свою статью «Can the Machine Think ?» («Может ли машина мыслить ?»). Также он предложил тест проверки программы на интеллектуальность. Он состоял в следующем: организовывалось «общение» между человеком и компьютерной программой, которые размещались в разных комнатах, и до тех пор, пока

исследователь не определял, кто за стеной – человек или программа, поведение последней считалось интеллектуальным [1]. Исходя из этого, Тьюринг предложил следующий критерий интеллектуальности программы: «Если поведение вычислительной машины, отвечающей на вопросы, невозможно отличить от поведения человека, отвечающего на аналогичные вопросы, то она обладает интеллектом».

В настоящее время существует три основные точки зрения на цели и задачи исследований в области искусственного интеллекта [1,6]. Согласно первой, исследования в этой области относятся к фундаментальным, в процессе которых разрабатываются новые модели и методы решения задач, традиционно считавшихся интеллектуальными и не поддававшихся ранее формализации с помощью классических алгоритмических методов, а также автоматизации. Интеллект и мышление непосредственно связаны с решением таких задач, как доказательство теорем, логический анализ, распознавание ситуаций, планирование поведения, управление в условиях неопределенности и т.п. Характерными чертами интеллекта, проявляющимися в процессе решения подобных задач, являются способность к обучению, обобщению, накоплению опыта и адаптации к изменяющимся условиям в процессе решения задач. Из-за этих качеств интеллекта мозг может решать разнообразные задачи, а также легко перестраиваться с решения одной задачи на другую. Таким образом, мозг, наделенный интеллектом, является универсальным средством решения широкого круга задач (в том числе плохо формализованных), для которых нет стандартных, заранее известных методов решения. Согласно второй точке зрения, это направление связано с новыми идеями решения задач на ЭВМ, с разработкой новых технологий программирования и с переходом к компьютерам с отличной от фон-неймановской архитектурой. Так, в качестве основы для таких систем предлагаются различные подходы на основе искусственный нейронных сетей, моделирующих наиболее общие принципы работы головного мозга [5]. Для таких моделей характерны легкое распараллеливание алгоритмов и связанная с эти высокая производительность, а также возможность работать даже при условии неполной информации об окружающей

среде. Третья точка зрения основана на том, что в результате исследований, проводимых в области ИИ, появляется множество прикладных систем, способных решать задачи, для которых ранее создаваемые системы были непригодны.

Обобщая изложенное, будем определять искусственный интеллект как научное направление, задачи которого связаны с разработкой методов моделирования отдельных функций интеллекта человека с помощью программно-аппаратных средств ЭВМ.

Исторически сложились три основных подхода к проведению исследований в области искусственного интеллекта.

Первый подход (машинный интеллект) в качестве объекта исследования рассматривает именно искусственный интеллект и состоит в моделировании внешних проявлений интеллектуальной деятельности человека с помощью средств ЭВМ. В основе его лежит тезис о том, что машина Тьюринга является теоретической моделью мозга, поэтому главное направление работ связано с созданием алгоритмического и программного обеспечения ЭВМ, позволяющих решать интеллектуальные задачи не хуже человека. Примером может служить шахматная программа, проявление интеллектуальности которой состоит в поиске игровой тактики, близкой к человеческой. Однако достигается это исключительно путем высокой скорости вычислений, в то время как у человека - благодаря высокоэффективному мышлению.

Как известно, человеческий мозг оперирует непрерывной информацией, где каждая мысль существует только внутри своего контекста. Знания хранятся в форме образов, которые часто трудно выразить словами. При этом сами образы характеризуются нечеткостью и размытостью, а обработка информации - небольшой глубиной и высоким параллелизмом. Все это свидетельствует о существенном различии с принципами машины Тьюринга и как следствие требует другого некомпьютерного подхода к моделированию интеллектуальных процессов.

Второй подход (искусственный разум) рассматривает данные о нейрофизиологических и психологических механизмах интеллектуальной деятельности и разумного поведения человека. Он стремится

воспроизвести эти механизмы с помощью программно-аппаратных средств. Развитие этого направления тесно связано с успехами наук о человеке, в первую очередь нейронаук (нейробиологии, генетики и т.п.).

Третий подход ориентирован на создание смешанных человеко-машинных интеллектуальных систем как симбиоз возможностей естественного и искусственного интеллекта. Важнейшими проблемами в этих исследованиях являются оптимальное распределение функций между естественным и искусственным интеллектом, организация диалога между человека и машиной.

Каждое из отмеченных направлений включает целый ряд разделов, к основным из которых можно отнести разработку систем, основанных на знаниях, анализ естественного языка и общения с ЭВМ, распознавание образом, анализ речи, создание адаптивных систем, игры и машинное творчество и др. [1]. В свою очередь, реализация подобных систем может быть выполнена на основе таких технологий ИИ, как представление и обработка знаний, эвристическое программирование, искусственные нейронные сети, эволюционные алгоритмы, нечеткие множества и др.

К настоящему времени разработано целое множество программный систем, в которых реализованы те или иные технологии ИИ. Такие системы принято называть интеллектуальными системами. К первой из подобных систем относят программу «Логик-Теоретик» (А. Ньюэлл, А. Тьюринг и др.), предназначенную для доказательства теорем исчисления высказываний.

Под интеллектуальной системой будем понимать адаптивную систему, позволяющую строить программы целенаправленной деятельности по решению поставленных перед ними задач на основании конкретной ситуации, складывающейся на данный момент в окружающей их среде [9]. В свою очередь, адаптивная система может быть охарактеризована как система, которая сохраняет работоспособность при непредвиденных изменениях свойств управляемого объекта, целей управления или окружающей среды путем смены алгоритма функционирования, программы поведения или поиска оптимальных, в некоторых случаях просто эффективных, решений и

состояний. Традиционно, по способу адаптации различают самонастраивающиеся, самообучающиеся и самоорганизующиеся системы.

К сфере решаемых интеллектуальными системами задач относятся задачи, обладающие, как правило, следующими особенностями:

- неизвестен или не может быть реализован алгоритм решения;
- если существует алгоритмическое решение, но его нельзя использовать из-за ограниченности ресурсов (время, память);
- задача не может быть сформулирована в числовой форме;
- цель нельзя выразить в терминах точно определенной целевой функции.

Разработка интеллектуальных систем, как правило, ведется в рамках одного или нескольких направлений ИИ, которых в настоящее время существует целое множество. Ниже кратко рассматриваются основные из них [1].

1.2. Основные направления исследований в области искусственного интеллекта

Разработка систем, основанных на знаниях. Является одним из главных направлений в искусственном интеллекте. Основной целью создания таких систем является выявление, исследование и применение знаний специалистов для решения различных практических задач. Обычно такие знания формализуются в виде некоторой системы правил. В этой области исследований осуществляется разработка моделей извлечения, представления и структуризации знаний с учетом их компьютеризации в виде базы знаний. Примеры практических разработок подобных систем обычно ассоциируются с экспертными системами.

Разработка систем общения на естественном языке и машинного перевода. Является наиболее важной с точки зрения перехода на новый качественный уровень взаимодействия с компьютером. Попытки создания подобных систем предпринимались с 1950-х годов 20 в. Основу систем машинного перевода составляет

классификация грамматических правил и приемов использования словаря. Однако для обработки сложного разговорного текста необходимы алгоритмы анализа его смысла, создание которых очень трудоемкая и пока нерешенная задача. Поэтому в настоящее время доступны системы, обеспечивающие диалог между человеком и компьютером на упрощенном, урезанном естественном языке, программы электронного перевода эффективные преимущественно при работе с односложным текстом, а также функции ассоциативного контекстного поиска в электронных словарях.

Разработка интеллектуальных систем на основе принципов обучения, самоорганизации и эволюции. Моделирование этих принципов ориентировано на исследование возможностей решения задач с помощью законов функционирования наиболее свойственных биологическим системам. Процесс обучения связан со способностью системы накапливать информацию и рационально корректировать в соответствии с ней свое поведение. Самоорганизация подразумевает способность системы обобщать накопленную информацию, например для поиска в ней закономерностей. Использование принципов эволюции позволяет системе приобретать новые качества и свойства для наиболее оптимального функционирования.

Распознавание образов. Является одним из ранних направлений искусственного интеллекта. Оно связано с моделированием особенностей восприятия внешнего мира, узнавания объектов. В основе этого лежит тот факт, что все объекты могут быть проклассифицированы по определенным признакам и, следовательно, умение различать их проявление и позволяет идентифицировать соответствующий объект.

Игры и машинное творчество. Машинное творчество охватывает сочинение компьютерной музыки, стихов, автоматизацию изобретения новых объектов. Компьютерные игры являются той сферой искусственного интеллекта, которая наиболее знакома большинству пользователей. Уровень реализации ИИ в игре во многом определяет ее интересность, поэтому разработчики компьютерных игр постоянно совершенствуют их интеллектуальную составляющую.

Программное обеспечение систем искусственного интеллекта. Инструментальные средства для разработки интеллектуальных систем включают специальные языки программирования, представления знаний, среды создания систем ИИ, а также оболочки экспертных систем.

Интеллектуальные роботы. Их создание связано с объединением технологий искусственного интеллекта и методов кибернетики, робототехники. В настоящее время их производство ограничивается манипуляторами с жесткой схемой управления, а также роботами развлекательного и бытового назначения с узкой областью применения и ограниченными функциями. Сдерживающим фактором при разработке более совершенных кибернетических систем являются нерешенные проблемы в области машинного зрения, адаптивного поведения, накопления и обработки трехмерной визуальной информации.

Уровень теоретических исследований по искусственному интеллекту в России не уступает мировому. Началом становления этого научного направления в нашей стране следует считать 1954 г., когда в МГУ начал свою работу семинар «Автоматы и мышление» под руководством академика Ляпунова А.А. Впоследствии стали активно развиваться направления, связанные с представлением и обработкой знаний, ситуационным управлением, моделированием рассуждений, распознаванием образов, обработкой естественного языка [3].

Развитие искусственного интеллекта в современной России связано с образованием в 1988 г. Ассоциации искусственного интеллекта, объединившей научные школы, исследователей по различным направлениям ИИ. Под ее эгидой проводятся различные исследования, организуются семинары для специалистов, устраиваются конференции, издается научный журнал.

В то же время проведение прикладных исследований, внедрение их результатов в коммерческие разработки происходит гораздо медленнее, чем за рубежом. Во многом это объясняется консервативностью потенциальных потребителей новых информационных технологий, а также настороженным отношением к возможностям искусственного интеллекта.

Далее рассматриваются основные методы научного направления «Искусственный интеллект», особенности их реализации, а также задачи, для которых их применение наиболее эффективно.

Контрольные вопросы

1. Какой смысл вкладывается в понятие «искусственный интеллект»?

2. Решением каких задач занимается научное направление искусственный интеллект?

3. Какие существуют подходы к исследованиям в области искусственного интеллекта?

4. Что такое интеллектуальная система?

5. В каких областях применение интеллектуальных систем наиболее эффективно?

6. Какие существуют основные направления исследований в области искусственного интеллекта?

7. Подготовьте реферат, посвященный истории развития искусственного интеллекта в России и за рубежом.

2. МЕТОДЫ ЭВРИСТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Эвристическое программирование – разработка стратегий действий на основе заранее заданных эвристик (теоретически необоснованных правил). Эвристический метод - метод решения задачи, основанный на интуиции решающего лица. Этот метод часто не имеет формального доказательства того, что он приводит к цели.

В основе методов эвристического программирования лежит понятие эвристики, которое может рассматриваться, с одной стороны, как информация, отражающая специфику задачи, с другой – как некоторое предположение о способе разрешения проблемы на основе имеющегося опыта решения подобных задач. Сам процесс эвристического программирования можно рассматривать как построение некоторого плана действий (планирование решения) с применением эвристик. Часто эвристическое программирование используется в задачах большой размерности, где решение в принципе может быть найдено перебором.

2.1. Направления эвристического программирования

Эвристическое программирование имеет двоякую цель:

- выяснение сущности естественного интеллекта;
- использование машинного интеллекта для решения сложных задач.

Эвристическое программирование охватывает несколько направлений, связанных с планированием решения задач ИИ (рис. 2.1)

Все задачи, связанные с построением плана действий, можно разбить на два типа [11]:

- планирование в пространстве состояний;
- планирование в пространстве задач.

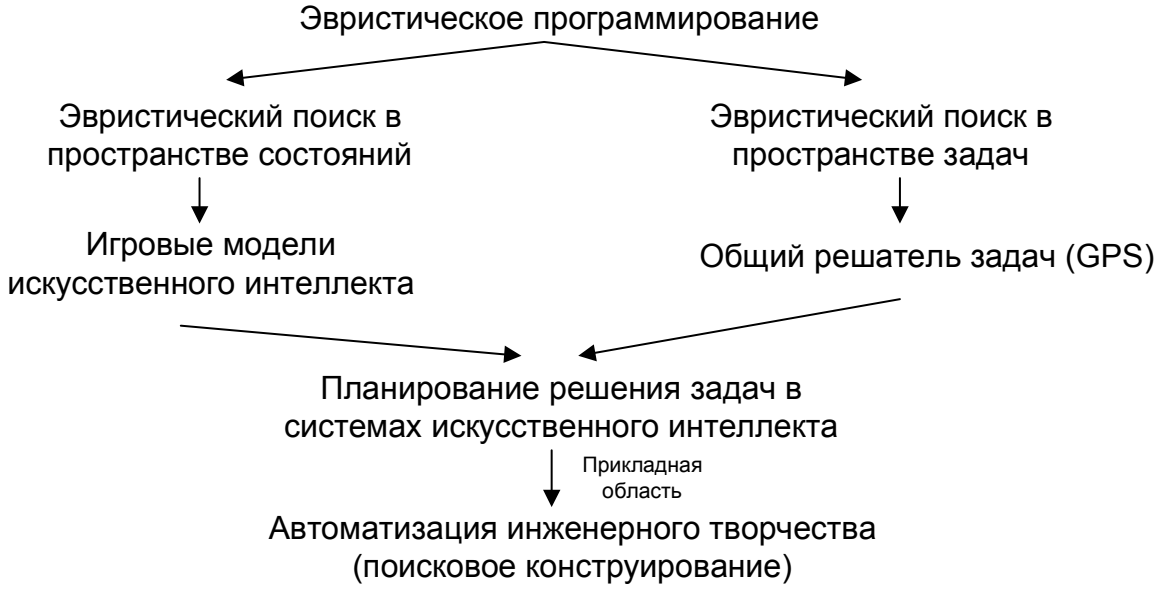


Рис. 2.1. Направления эвристического программирования

В первом случае считается заданным некоторое пространство ситуаций (состояний), требуется найти путь, ведущий из начального состояний в одно из конечных.

Второй тип задач состоит в поиске декомпозиции исходной задачи на подзадачи посредством введения между ними отношений «часть-целое», «задача-подзадача», «общее-частное» и т.п., что в результате приводит к задачам, решение которых известно.

В основе решения перечисленных типов задач лежит тот или иной метод эвристического поиска.

2.2. Эвристический поиск в пространстве состояний

Типичным представителем класса задач, для которых подходит представление (формализация) в пространстве состояний, является головоломка, известная как игра в пятнадцать. В ней используется пятнадцать пронумерованных (начиная с 1) подвижных фишек, расположенных в клетках квадрата 4x4. Одна клетка этого квадрата остается всегда пустой. Передвигая соседние с ней фишки необходимо добиться упорядоченного расположения их номеров. На рис. 2.2 изображены две конфигурации фишек для игры в пятнадцать (а) и для ее

упрощенного варианта - квадрата 3x3 и восьми фишек (б). Рассмотрим задачу перевода начальной (первой) конфигурации в целевую (вторую) конфигурацию. Решением этой задачи будет подходящая последовательность сдвигов фишек, например: «передвинуть фишку 8 вверх, фишку 6 влево и т.д.».

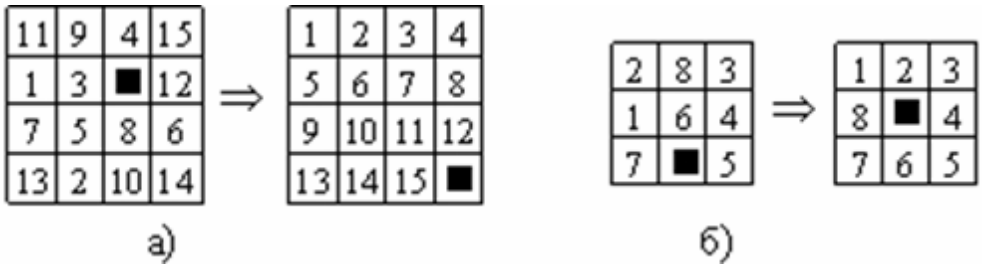


Рис. 2.2. Пример перехода между состояниями игры

Основными особенностями класса задач, к которому принадлежит рассмотренная головоломка, является наличие в каждой задаче точно определенной начальной ситуации и точно определенной цели. Имеется также некоторое множество операций или ходов, переводящих одну ситуацию в другую. Именно из таких ходов состоит искомое решение задачи, которое можно получить (теоретически) методом проб и ошибок. Действительно, отправляясь от начальной ситуации, можно построить все промежуточные конфигурации, возникающие в результате выполнения каждого из возможных ходов, затем построить множество конфигураций после применения следующего хода и так далее – пока не будет достигнута целевая конфигурация.

2.2.1. Представление задачи в виде пространства состояний

Ключевым понятием при формализации задачи в пространстве состояний является понятие состояния, характеризующего некоторый момент решения задачи. Так, для игры в пятнадцать состояние – это просто некоторая конкретная конфигурация фишек. Среди всех состояний выделяются начальное состояние и целевое состояние (целевая конфигурация), в совокупности определяющие задачу, которую надо решить.

Другим важным понятием для рассматриваемого представления является понятие оператора, или допустимого хода в задаче. Оператор преобразует одно состояние в другое, являясь, по сути, функцией,

определенной на множестве состояний и принимающей значения из этого множества. Для рассматриваемой игры удобнее выделить четыре оператора, соответствующие перемещениям пустой клетки влево, вправо, вверх, вниз. В некоторых случаях оператор может оказаться неприменимым к какому-то состоянию: например, операторы вправо и вниз неприменимы, если пустая клетка расположена в правом нижнем углу (соответствующая функция является частично определенной).

В терминах состояний и операторов решение задачи есть определенная последовательность операторов, преобразующая начальное состояние в целевое. Решение задачи ищется в пространстве состояний – множестве состояний, достижимых из начального состояния при помощи операторов. В игре в пятнадцать пространство состояний состоит из всех конфигураций фишек, которые могут быть образованы в результате допустимых перемещений фишек.

Таким образом задача эвристического поиска содержательно может быть представлена в следующем виде [11]. Заданы начальная ситуация (объект, состояние), конечная ситуация и множество операторов, преобразующих одну ситуацию в другую. Требуется найти такую последовательность операторов, которая преобразует начальную ситуацию в конечную наилучшим образом. Математически данная задача описывается с помощью четырех множеств $< S_0, S, F, T >$, где

S – множество состояний;

$S_0 \subseteq S$ - множество начальных состояний;

$T \subseteq S$ - множество конечных состояний;

F – множество операторов.

Метод решения задачи $< S_0, S, F, T >$ называется методом эвристического поиска, если он на каждом шаге находит все возможные применения операторов к данному текущему состоянию, а порядок рассмотрения состояний и порядок применения операторов управляется свойствами уже рассмотренных до этого шага состояний.

2.2.2. Представление пространства состояний

Пространство состояний можно представить в виде графа, вершины которого соответствуют состояниям, а дуги – применяемым операторам. Тогда решение задачи – это путь, ведущий от начального

состояния к целевому. На рис. 2.3 показана часть пространства состояний для игры в пятнадцать, в каждой вершине помещена та конфигурация фишек, которую она представляет. Пространства состояний могут быть большими и даже бесконечными, но в любом случае предполагаются конечность множества допустимых операторов и конечность множества возможных состояний.

Итак, формализация задачи с использованием пространства состояний включает выявление и определение следующих составляющих:

- формы описания состояний и описание исходной задачи;
- множество операторов и их воздействий на описания состояний;
- указание свойств целевых состояний.

Эти составляющие задают (неявно) пространство, в котором требуется провести поиск решения задачи.

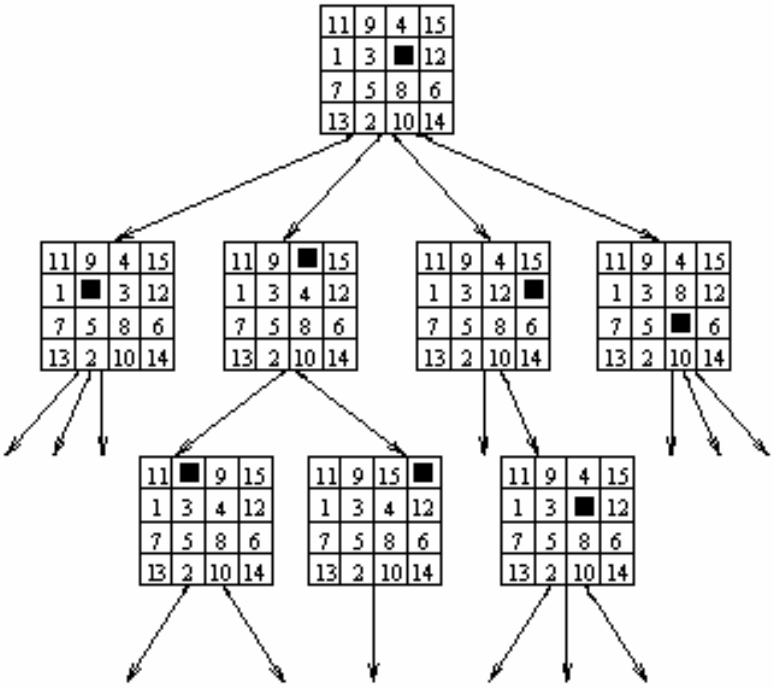


Рис. 2.3. Фрагмент пространства состояний для игры в пятнадцать

Ясно, что решение задачи, представленной описанным способом, можно в принципе обнаружить, осуществляя последовательный поиск или перебор вершин в пространстве состояний. В начале этого процесса к начальному состоянию применяется тот или иной оператор. Затем на каждом шаге поиска к одному из уже полученных

(просмотренных) состояний применяется допустимый оператор и строится новая вершина. Поиск заканчивается, когда построено целевое состояние.

Математически пространство состояний может быть представлено как граф *G(X, E)*, где $X = \{x_0, x_1, ..., x_{n-1}\}$ - множество (в общем случае бесконечное) вершин графа, каждая из которых отождествляется с одним из состояний $s \in S$;

$E = \{(x_i, x_j) \mid x_i, x_j \in X, x_j \in F(x_i)\}$ - множество дуг или ребер графа бесконечное, если бесконечно множество *X*. Все ребра в графе являются взвешенными, т.е. им присвоено некоторое значение (вес), позволяющее оценить с помощью оценочной функции $\varphi(x)$ переход от одного состояния к другому;

F(x) – преобразование, порождающее все дочерние вершины от x.

| *F(x)* | - количество дочерних вершин от x, т.е. вершин, соединенных с x дугой.

Структура графа пространства состояний представляется в виде матрицы весов (значений оценочной функции) графа: $W[w_{i,j}]$, $i = 0,1,2,...,n-1$, в которой $w_{i,j}$ соответствует значение веса ребра (x_i, x_j) или 0, если вершины x_i, x_j не связаны ребром (x_i, x_j).

Если граф пространства состояний не является простым, то он должен быть преобразован в него посредством исключения всех петель и заменой каждого множества параллельных ребер кратчайшим ребром (ребром с наименьшим весом) из этого множества, каждое неориентированное ребро заменяется парой ориентированных.

В множестве вершин X выделяют подмножество вершин $X_0 \subseteq X$, соответствующее множеству начальных состояний $S_0 \subseteq S$ и $X_t \subseteq X$, соответствующих множеству конечных состояний $T \subseteq S$.

Определим путь в графе *G* как $r = (x_1, x_2 ... x_{k-1})$, где $(x_i, x_{i+1}) \in E$, $i = 1,2,...,k-1$.

Если $x_1 \in X_0$, $x_k \in X_t$, то решение задачи эвристического поиска в пространстве состояний сводится к задаче поиска пути r на графе *G*. Сумма всех ребер в пути называется весом или длиной пути.

Рассмотрим еще один пример представления задачи в пространстве состояний, а именно задачу о коммивояжере. В ней коммивояжер, располагая картой дорог между несколькими городами, должен выстроить кратчайший маршрут своей поездки так, чтобы побывать в каждом городе, но не более одного раза. На рис. 2.4 показана карта дорог между 7 городами.

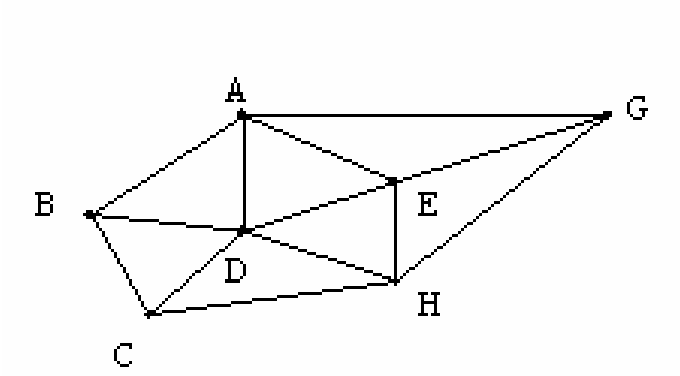


Рис. 2.4. Пример задачи о коммивояжере

Для этой задачи фрагмент пространства состояний, представленного деревом, приведен на рис. 2.5.

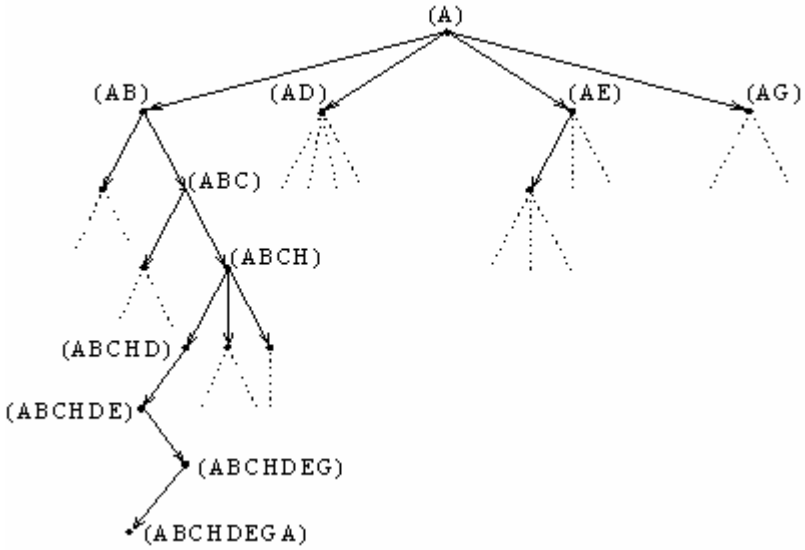


Рис. 2.5. Фрагмент пространства состояний для задачи о коммивояжере

Таким образом, алгоритм поиска решения этой задачи заключается в построении дерева пространства состояний посредством перебора его вершин до тех пор, пока не будет обнаружена целевая вершина. При этом эффективность решение будет зависеть от такого важного параметра, как максимальная глубина дерева.

Вершины и указатели, построенные в процессе перебора, образуют поддерево всего неявно определенного пространства состояний.

Будем называть такое поддерево деревом перебора.

Известные алгоритмы поиска в пространстве состояний различаются несколькими характеристиками [1]:

- использованием или нет эвристической информации;
- порядком раскрытия (обхода) вершин;
- полнотой просмотра пространства;
- направлением поиска.

В соответствии с первой характеристикой алгоритмы делятся на два класса – слепые и эвристические. В слепых алгоритмах поиска в пространстве состояний местонахождение целевой вершины никак не влияет на порядок, в котором рассматриваются (раскрываются) вершины. В противоположность им эвристические алгоритмы (методы) используют (для уменьшения возникающего перебора) априорную (эвристическую) информацию о том, где в пространстве состояний расположена цель, поэтому для раскрытия обычно выбирается более перспективная вершина.

Два основных вида слепых алгоритмов поиска, различающихся порядком раскрытия вершин, – это алгоритмы поиска (перебора) вширь и поиска (перебора) вглубь. Как слепые, так и эвристические алгоритмы могут отличаться полнотой просмотра пространства состояний.

Полные алгоритмы перебора при необходимости осуществляют полный просмотр пространства. В отличие от них, неполные алгоритмы реализуют просмотр лишь некоторой части пространства, и если искомая целевая вершина не находится в этой части, то решение этим алгоритмом не будет найдено.

В соответствии с направлением поиска алгоритмы можно разделить на прямые (поиск ведется от начальной вершины к целевой), обратные (поиск от целевой вершины в направлении к начальной) и двунаправленные (чередование прямого и обратного поиска, или же одновременное их проведение). Наиболее употребительными (отчасти, из-за их простоты) являются прямые алгоритмы.

2.2.3. Алгоритмы слепого перебора

Двумя основными разновидностями слепого перебора являются алгоритмы перебора в ширину и в глубину.

В алгоритме перебора в ширину вершины раскрываются (перебираются) в том порядке, в котором они строятся. На рис. 2.6 приведено дерево, полученное в результате применения алгоритма поиска в ширину для некоторой начальной конфигурации игры в восемь, причем алгоритм работает только до глубины 4. В вершинах дерева помещены соответствующие описания состояний, они занумерованы в том порядке, в котором были раскрыты.

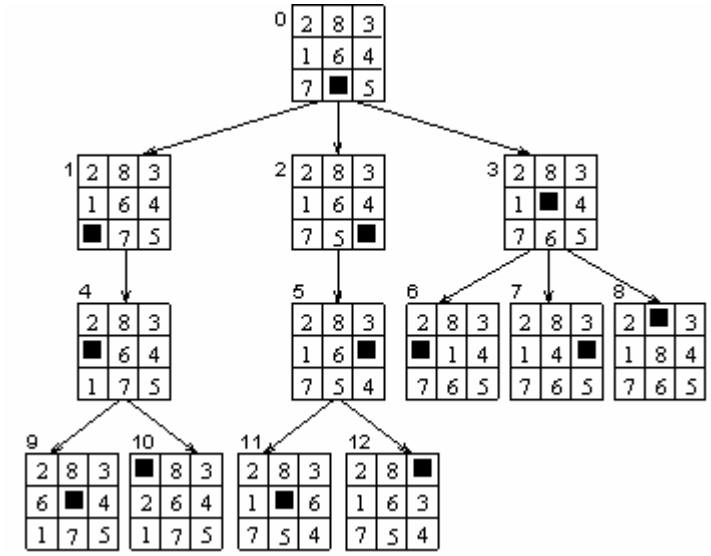


Рис. 2.6. Пример дерева, построенного алгоритмом поиска в ширину

Алгоритм перебора в ширину состоит из следующей последовательности шагов.

1. Поместить начальную вершину в список нераскрытых вершин Open.

2. Если список Open пуст, то окончание алгоритма и выдача сообщения о неудаче поиска в противном случае - переход к следующему шагу.

3. Выбрать первую вершину из списка Open (назовем ее Current) и перенести ее в список Closed.

4. Раскрыть вершину Current, образовав все ее дочерние вершины. Если дочерних вершин нет, то перейти к шагу 2, иначе поместить все дочерние вершины (в любом порядке) в конец списка Open и построить указатели, ведущие от этих вершин к родительской вершине Current.

5. Проверить, нет ли среди дочерних вершин целевых. Если есть хотя бы одна целевая вершина, то закончить алгоритм и выдать

решение задачи в виде последовательности указателей от найденной целевой вершины к начальной. В противном случае перейти к шагу 2.

Можно показать, что при переборе вширь непременно будет найден самый короткий путь к целевой вершине, при условии, что этот путь вообще существует. Если же такого пути нет, то будет сообщено о неуспехе поиска в случае конечных графов, а в случае бесконечных графов алгоритм никогда не кончит свою работу.

Глубину вершины в дереве можно определить следующим образом:

- глубина корня дерева равна нулю;

- глубина каждой некорневой вершины равна глубине ее родительской вершины плюс единица.

В алгоритме перебора в глубину прежде всего раскрываются те вершины, которые были построены последними, то есть имеющие наибольшую глубину.

Основные шагами алгоритма перебора в глубину являются следующие.

1. Поместить начальную вершину в список Open.

2. Если список Open пуст, то закончить алгоритм и выдать сообщение о неудаче поиска, в противном случае перейти к следующему шагу.

3. Выбрать первую вершину из списка Open (назовем ее Current) и перенести ее в список Closed.

4. Если глубина вершины Current равна граничной глубине, то перейти к шагу 2, в ином случае перейти к следующему шагу.

5. Раскрыть вершину Current, построив все ее дочерние вершины. Если дочерних вершин нет, то перейти к шагу 2, иначе поместить все дочерние вершины (в произвольном порядке) в начало списка Open и построить указатели, ведущие от этих вершин к родительской вершине Current.

6. Если среди дочерних есть хотя бы одна целевая вершина, то закончить алгоритм и выдать решение задачи в виде последовательности указателей от найденной целевой вершины к начальной. В противном случае перейти к шагу 2.

На рис. 2.7 показано дерево, построенное алгоритмом поиска в глубину, граничная глубина установлена равной 4. Вершины занумерованы в том порядке, в котором они были раскрыты. В качестве начального состояния взята та же самая конфигурация игры в восемь, что и в примере на рис. 2.6. В обоих этих деревьях построено одинаковое количество вершин, но порядок их раскрытия различается. Видно, что в алгоритме поиска в глубину сначала идет поиск вдоль одного пути, пока не будет достигнута максимальная глубина, затем рассматриваются альтернативные пути той же или меньшей глубины, которые отличаются от него лишь последним шагом, после чего рассматриваются пути, отличающиеся последними двумя шагами, и т.д.

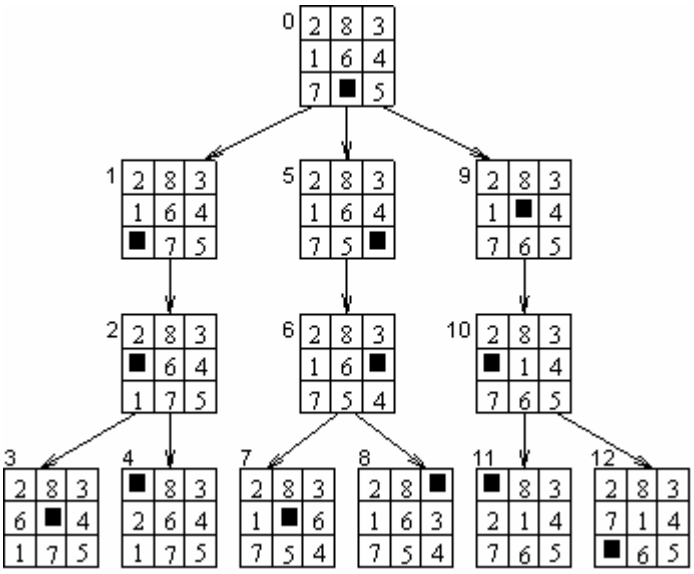


Рис. 2.7. Пример дерева, построенного алгоритмом поиска в глубину

При бесконечном пространстве состояний перебор в глубину может привести к бесконечному процессу поиска, если он пошел по ветви дерева, не содержащей целевого состояния. Поэтому необходимо ограничение этого процесса, например за счет ограничения максимальной глубины просмотра дерева. Это означает, что в ходе перебора раскрываются только вершины с глубиной, не превышающей некоторую заданную граничную глубину, т.е. в первую очередь раскрытию подлежит вершина наибольшей глубины, но не превышающей эту границу. Соответствующий алгоритм поиска называется ограниченным перебором в глубину.

Если сравнивать алгоритмы поиска в ширину и в глубину, то последний, несмотря на свою неполноту, может оказаться предпочтительнее: если он начат с удачной стороны, то целевая вершина будет обнаружена раньше, чем в алгоритме поиска в ширину.

Если перебор осуществляется на графах, а не на деревьях, необходимо внести некоторые очевидные изменения в указанные алгоритмы. В алгоритме полного перебора следует дополнительно проверять, не находится ли уже вновь построенная вершина в списках Open и Closed по той причине, что она уже строилась раньше в результате раскрытия какой-то вершины. Если это так, то такую вершину не нужно снова помещать в список Open. В алгоритме же поиска в глубину, кроме рассмотренного изменения, может оказаться необходимым пересчет глубины порождающейся вершины, уже имеющейся либо в списке Open, либо в списке Closed.

В целом алгоритмы слепого перебора являются неэффективными методами поиска решения, приводящими в случае нетривиальных задач к проблеме комбинаторной сложности. Действительно, если L – длина решающего пути, а B – количество ветвей (дочерних вершин) у каждой вершины, то для нахождения решения надо исследовать B^L путей, ведущих из начальной вершины. Величина эта растет экспоненциально с ростом длины решающего пути, что приводит к ситуации, называемой комбинаторным взрывом.

Таким образом, для повышения эффективности поиска необходимо использовать информацию, отражающую специфику решаемой задачи и позволяющую более целенаправленно двигаться к цели. Такая информация обычно называется эвристической, а соответствующие алгоритмы – эвристическими.

2.2.4. Эвристический поиск

Идея, лежащая в основе большинства эвристических алгоритмов, состоит в том, чтобы оценивать перспективность нераскрытых вершин пространства состояний и выбирать для продолжения поиска наиболее перспективную вершину.

Для этого используется эвристическая оценочная функция. Эта функция определяется на множестве вершин пространства состояний и принимает числовые значения, которые могут интерпретироваться как перспектива раскрытия вершины или вероятность ее расположения на решающем пути. Использование такой функции позволяет сделать поиск упорядоченным.

Рассмотрим основные шаги алгоритма эвристического поиска.

Пусть имеются: одна начальная вершина $x_0 \in X_0$;

S – множество уже выбранных вершин; $\overline{S}$ - множество вершин – кандидатов в S, при этом $S \bigcap \overline{S} = 0$;

X_t - множество конечных вершин;

x – текущая вершина;

x_i - дочерняя вершина x.

Тогда алгоритм поиска в графе пространства состояний заключается в поиске пути, начиная из вершины x_0, просматривая граф в ширину и представляется следующими шагами.

1. Поместить x_0 в $\overline{S}$ и вычислить оценочную функцию $\varphi(x_0)$.

2. Выбрать такую $x \in \overline{S}$, что $\varphi(x) = \min\limits_{y \in \overline{S}}(\varphi(y))$ и поместить ее в S, изъяв из $\overline{S}$. При равенстве выбрать любую.

3. Если $x \in X_t$ - путь найден, иначе продолжить.

4. Найти все $x_i \in F(x)$, если $F(x) = 0$, то перейти к шагу 2, иначе вычислить все $\varphi(x_i)$.

5. Для каждого x_i:

а) если $x_i \notin S \bigcup \overline{S}$, то поместить x_i в $\overline{S}$;

б) если $x_i \in F(x) \bigcap \overline{S}$, то сопоставить x_i наименьшее из старой и вновь полученной оценки $\varphi(x_i)$;

в) если $x_i \in F(x) \bigcap S$, то сопоставить x_i наименьшее из старой и вновь полученной оценки $\varphi(x_i)$, поместить x_i в $\overline{S}$ (изъяв ее из S);

г) в остальных случаях не изменять S и $\overline{S}$.

6. Перейти к шагу 2.

Пункты 5(б), 5(в) отражают действие алгоритма, когда оператор F порождает уже рассмотренные вершины, которые к этому моменту находятся в S или $\overline{S}$, поэтому этим вершинам приписываются наименьшие из возможных оценочных функций.

На рис. 2.8 показано дерево игры в восемь, построенного алгоритмом эвристического перебора с указанной оценочной функцией. Оценка каждой вершины приведена рядом с ней внутри кружка. Отдельно стоящие цифры показывают порядок, в котором раскрывались вершины.

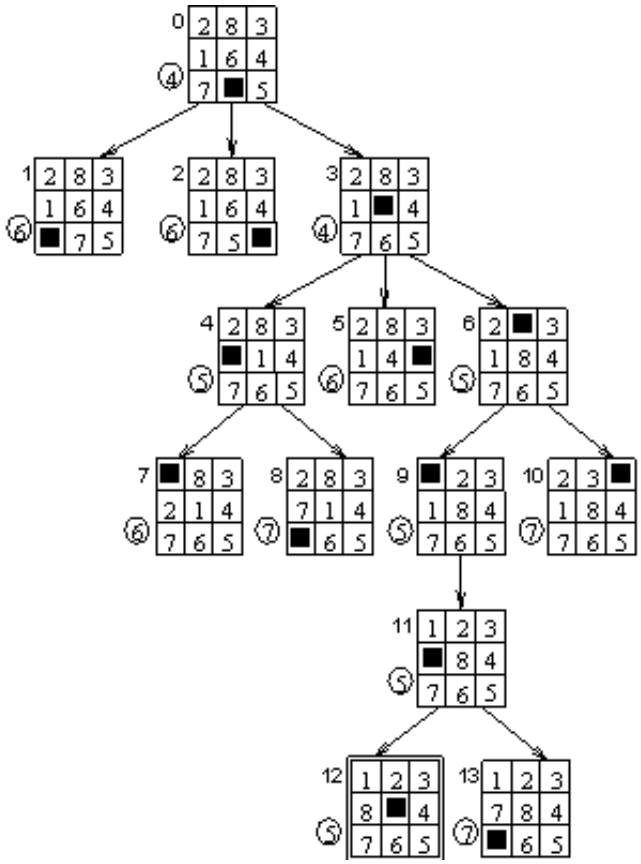


Рис. 2.8. Пример дерева эвристического поиска

Рассмотрим процесс построения дерева поиска пути на основе изложенного ранее алгоритма. Результаты поиска будем представлять в виде двух множеств S и $\overline{S}$, содержащих вершины с указанием в скобках значений весов входящих в них дуг. В качестве оценочной функции будем использовать следующую: $\varphi(x) = D(x) + K(x)$, где

D(x) – глубина вершины x или число ребер дерева на пути от этой вершины к начальной;

K(x) – число фишек позиции – вершины *x*, лежащих не на «своем» месте.

Будем считать, что меньшее значение $\varphi(x)$ соответствует более перспективной вершине, и вершины раскрываются в порядке увеличения (возрастания) значения оценочной функции.

1. $S = \{x_0(4)\}$;
   $\overline{S} = \{x_1(6), x_2(6), x_3(4)\}$.

Вершины x_1, x_2, x_3 являются дочерними для вершины x_0. Вершина x_3 имеет наименьшее значение оценочной функции, поэтому она исключается из $\overline{S}$ и добавляется в S.

2. $S = \{x_0(4), x_3(4)\};$
$\overline{S} = \{x_1(6), x_2(6), x_4(5), x_5(6), x_6(5)\}.$

Новые вершины в $\overline{S}$ есть дочерние для последней добавленной вершины. Вершины x_4, x_6 имеют одинаковую минимальную оценку, выберем из них любую и добавим в *S*.

3. $S = \{x_0(4), x_3(4), x_6(5)\};$
$\overline{S} = \{x_1(6), x_2(6), x_4(5), x_5(6), x_9(5), x_{10}(7)\}.$

4. $S = \{x_0(4), x_3(4), x_6(5), x_9(5)\};$
$\overline{S} = \{x_1(6), x_2(6), x_4(5), x_5(6), x_{10}(7), x_{11}(5)\}.$

5. $S = \{x_0(4), x_3(4), x_6(5), x_9(5), x_{11}(5)\};$
$\overline{S} = \{x_1(6), x_2(6), x_4(5), x_5(6), x_{10}(7), x_{13}(7), x_{12}(5)\}.$

6. $S = \{x_0(4), x_3(4), x_6(5), x_9(5), x_{11}(5), x_{12}(5)\}.$

Последняя вершина, добавленная в S $x_{12} \in T$, следовательно, путь найден: $x_0 \to x_3 \to x_6 \to x_9 \to x_{11} \to x_{12}$.

В ходе решения задачи может возникнуть ситуация, когда при выборе вершин для множества *S* совпадают их оценочные функции, в этом случае необходимо проанализировать всевозможные альтернативные пути, начиная с данного этапа.

Найденный путь решения задачи длиною в пять ходов может быть получен и другими методами перебора, но использование оценочной функции приводит к существенно меньшему числу раскрытий вершин. Действительно, сравнение трех алгоритмов перебора показывает, что в среднем алгоритм эвристического поиска обнаруживает решение быстрее алгоритмов слепого перебора. Важно однако, что использование эвристической функции не может гарантировать сокращение поиска во всех случаях и иногда (хотя и редко) решение задачи может искаться дольше, чем с использованием слепого метода.

Ясно, что подбор «хорошей» эвристической функции (существенно сокращающей поиск) наиболее трудный момент при формализации задачи, особенно это важно в больших пространствах состояний. Можно сравнивать различные оценочные функции для одной

задачи по их эвристической силе, т.е. по тому, насколько они сокращают (ускоряют) поиск.

2.2.5. Алгоритм поиска пути на графе

В ряде случаев при наличии графа, описывающего полное пространство состояний и динамически изменяющейся игровой ситуации, возникает задача поиска за минимальное время короткого пути, ведущего из начальной ситуации x_0 в одно из конечных x_t. Для ее решения существует несколько алгоритмов, один из которых состоит в следующем.

Рассматриваемый алгоритм определяет расстояние между вершинами в простом орграфе с неотрицательными весами. Алгоритм, начиная из вершины x_0, просматривает граф в ширину, помечая вершины x_j значениями – метками их расстояний от x_0. Временная метка вершины x_j - это минимальное расстояние от x_0 до x_j, когда в определении пути на графе учитываются не все маршруты из x_0 в x_j. Окончательная метка x_j содержит минимальное расстояние на графе от x_0 до x_j. Таким образом, в каждый момент времени работы алгоритма некоторые вершины будут иметь окончательные метки, а остальная их часть – временные. Алгоритм заканчивается, когда вершина x_t получает окончательную метку, то есть расстояние от x_0 до x_t.

Вначале вершине x_0 присваивается окончательная метка 0 (нулевое расстояние до самой себя), а каждой из остальных $|X|$ - 1 вершин присваивается временная метка ∞ (бесконечность). На каждом шаге алгоритма одной вершине с временной меткой присваивается окончательная и поиск продолжается дальше. На каждом шаге вершины меняются следующим образом.

1. Если имеется дуга из x_i в x_j, то каждой вершине x_j, не имеющей окончательной метки, присваивается новая временная метка – наименьшая из ее временной и числа ($w_{i,j}$+окончательная метка x_i), где x_i - вершина, которой присвоена окончательная метка на предыдущем шаге.

2. Определяется наименьшая из всех временных меток, которая становится окончательной меткой своей вершины, в случае равенства меток выбирается любая из них.

Циклический процесс п.1, п.2 продолжается до тех пор, пока вершина x_t не получит окончательной метки, которая представляет собой короткий путь от этой вершины до начала x_0.

Рассмотрим работу представленного выше алгоритма для графа, заданного в виде матрицы весов графа (табл. 2.1).

Таблица 2.1

Матрица весов графа

W	*X0*	*X1*	*X2*	*X3*	*X4*	*X5*	*X6*
X0	0	7	2	0	0	0	0
X1	0	0	0	1	5	0	10
X2	0	3	0	5	0	0	0
X3	0	0	0	0	3	7	0
X4	0	0	8	0	0	5	2
X5	0	0	0	0	5	0	6
X6	0	0	0	0	0	0	0

Граф со структурой, соответствующей данной матрице представлен на рис. 2.9.

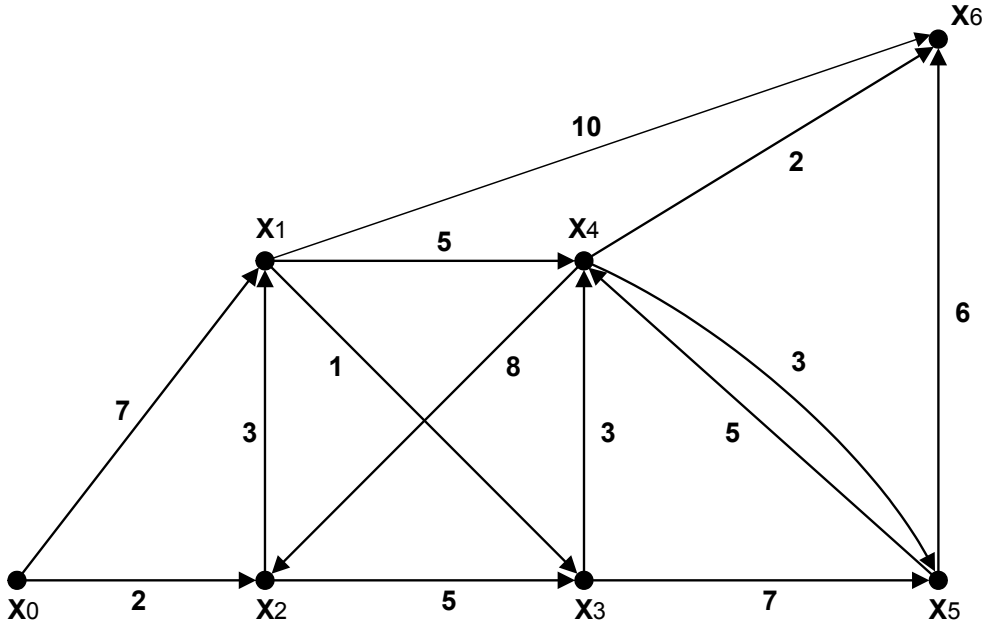


Рис. 2.9. Граф для демонстрации работы алгоритма

Процесс назначения меток вершинам графа на каждом шаге представлен в табл. 2.2.

Таблица 2.2

Процесс работы алгоритма

Шаг	X0	X1	X2	X3	X4	X5	Xt
0	0	∞	∞	∞	∞	∞	∞
1		7	2	∞	∞	∞	∞
2		5		7	∞	∞	∞
3				6	10	∞	15
4					9	13	15
5						12	11

Квадратами выделены окончательные метки. Путь перемещения от x_t к x_0 отмечен в таблице ломаной кривой через вершины $x_0 \to x_2 \to x_1 \to x_3 \to x_4 \to x_6$ и его длина равна 11 (рис. 2.10).

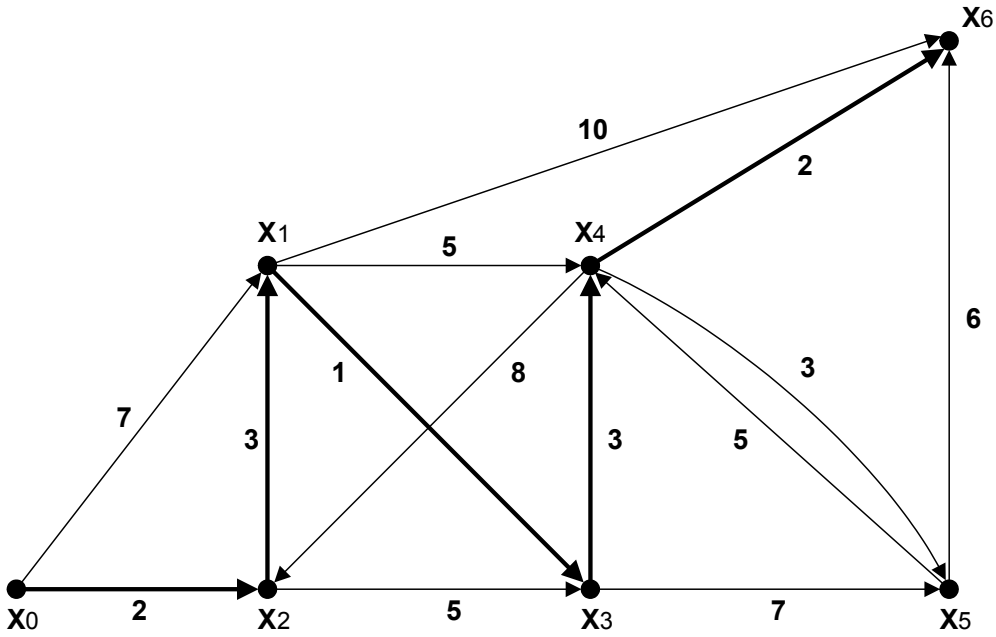


Рис. 2.10. Представление найденного пути на графе примера

Далее приводится алгоритм для создания компьютерной программы поиска пути на графе.

```
For x∈X do
Begin
  Mark[x] = FALSE;
  Dist[x] = ∞;
End;
y = x0;
Mark[x0] = TRUE;
Dist[x0] = 0;
```

```
While not Mark[xt] do
Begin
  For x∈X do
    If not Mark[x] and Dist[x]>Dist[y]+W[y,xt] then
    Begin
      Dist[x]=Dist[y]+w[y,x];
      Prev[x]=y;
    End;
{Поиск новой вершины y∈X с минимальной временной меткой}
    Dist[y] = min(Dist[x]); {x∈X and Mark[x]=FALSE}
    Mark[y] = TRUE;
End.
```

В приведенном алгоритме граф представляется матрицей весов W. Вектор Mark[x] меток вершин устанавливает принадлежность вершины $x \in X$ постоянной (TRUE) или временной (FALSE) метке. Вектор Dist в алгоритме фиксирует текущие значения вершин. Вектор Prev позволяет восстановить в обратной последовательности вершины найденного пути. Prev[x] указывает на вершину с окончательной меткой, ближайшую к вершине x. Последовательность вершин окончательного пути будет иметь следующий вид: xt, Prev[xt], Prev[Prev[xt]], Prev[Prev[Prev[xt]]], … , x0, а значение Dist[xt] составит длину пути из x0 в xt. Очередная новая вершина, претендующая на постоянную метку, обозначается через y.

2.3. Эвристический поиск в пространстве задач

Эвристический поиск в пространстве задач сводится к нахождению доказательства того, что решение данной задачи выводится из решения совокупности ее подзадач. Решение будет получено тогда, когда множество подзадач будут полностью состоять из заведомо разрешимых задач (аксиом) или будет доказано, что исходная задача не имеет решения.

На рис. 2.11 представлен пример задачи T в виде И/ИЛИ дерева.

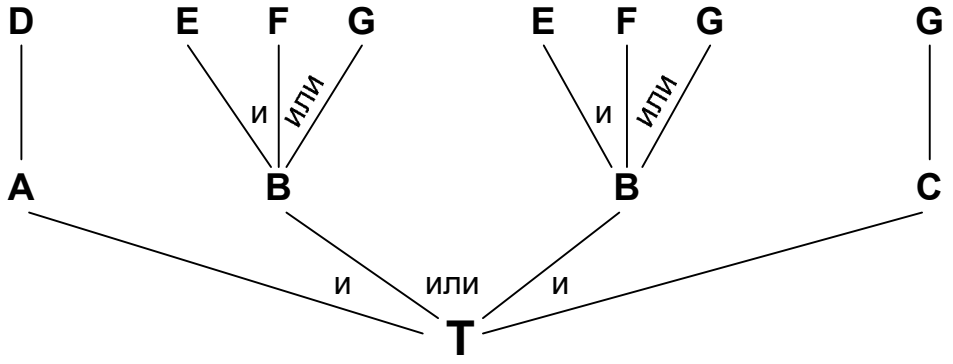


Рис. 2.11. Представление пространства подзадач

Задача T может быть решена, если решены подзадачи A, B или C, D. В свою очередь задача A решаема, если достижима вершина D. Задача B – если E, F или G, а задача C, если решаема подзадача G. Подзадачи D, E, F, G называются разрешимыми.

Существует несколько различных методов эвристического поиска в пространстве задач, например универсальный решатель задач GPS, предложенный Ньюэллом и др. в 1960 г.

В основе работы GPS лежит метод «анализа целей и средств», согласно которому для достижения каждой подцели выбирается оператор, уменьшающий различие между имеющимся и желаемым состоянием объекта. При этом изначально должны быть заданы операторы, различия, таблица связей, а также цель (супер цель) и объекты, участвующие в задаче [11].

В GPS используется четыре вида целей:

- преобразование A в объект B;
- уменьшение различия D между объектами A и B;
- применение оператора Q к объекту A;
- выбор из множества S элемента, наилучшим образом удовлетворяющего критерию C.

Работа метода GPS заключается в циклическом поиске способа уменьшить различие между текущим состоянием объекта и заданной целью, посредством рекурсивной генерации из существующих новых подцелей (рис. 2.12).

Каждой из подцели ставится в соответствие свой метод или оператор ее достижения. Если же на некотором шаге работы GPS невозможно применить такой оператор, то вырабатывается новая подцель – уменьшить различие (между исходным состоянием объекта и

желаемым), препятствующее применению оператора. Для определения операторов, наилучшим образом подходящих для уменьшения каждого различия, используется таблица связей. Информация, необходимая для сравнения объектов, как правило, включается в программу, реализующую GPS.

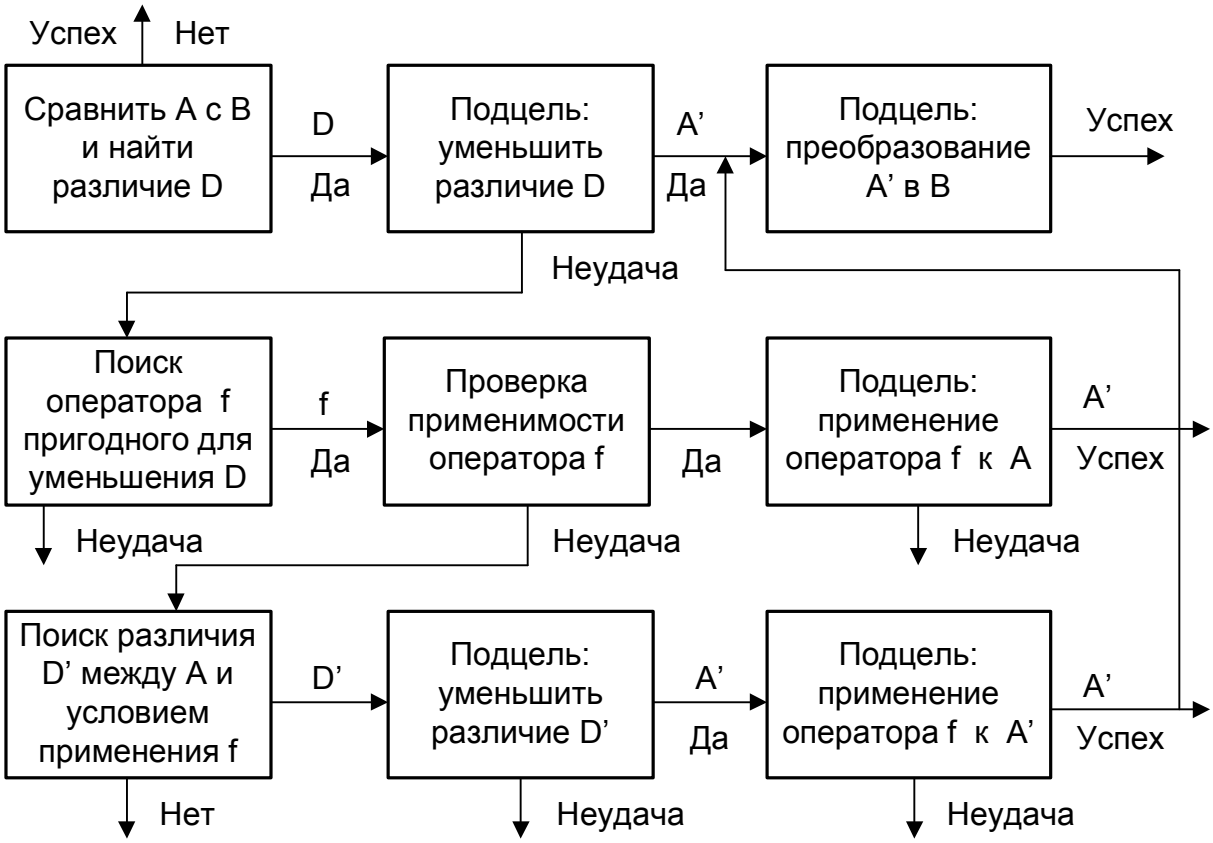


Рис. 2.12. Обобщенная схема работы GPS

Возможности GPS были исследованы на примере решения целого ряда задач, среди которых наиболее известны, такие как задача интегрирования, задача «Ханойская башня», задача «о семи Кенигсбергских мостах» и др.

2.4. Игровая модель эвристического поиска

Игры имеют очень большое значение при исследовании искусственного интеллекта. Игры представляют собой простые проблемные среды для изучения и испытания процедур эвристического поиска. В доказательство можно привести следующий тезис. Ведь никто

не спорит, что человек думает, когда играет. И если человек играет с компьютером, то можно говорить о такой способности применительно к машине.

Игры удобны тем, что известны правила и всегда можно определить, хорошо человек или машины играет или плохо. Кроме того, игры имеют сходство с реальными проблемами, и методы, разработанные для решения простых игровых задач, могут быть распространены и на более трудные, например поисковое конструирование и автоматизацию решения изобретательских задач.

В теории игр для каждой игры характерны следующие особенности [1].

1. Должно быть как минимум два игрока.

2. Игроки поочередно делают ходы, пытаясь максимизировать свой выигрыш.

3. Варианты ходов, делаемых игроками, могут быть как известны, так и не известны другим игрокам.

4. Существуют критерии окончания игры и определения победителей.

5. Существует мера выигрыша – некая выплата в условных очках, баллах или деньгах.

В зависимости от п.3 игры бывают с полной информацией (шахматы, шашки, го, крестики – нолики, реверси и т.д.) и с неполной информацией (большинство карточных игр). Критерий полноты информации – знание всех ходов противника, которые произошли ранее или могут быть сделаны в настоящий момент (в карточных играх игрок не видит карты противника, в то время как в шахматах игрок видит всю доску и все возможные ходы).

2.4.1. Структура игровой модели

Игровая модель представляет собой комбинацию следующих элементов:

- дерево игры: граф $G = (X, E)$;
- порождающие процедуры $F(x)$;
- критерий окончания игры ($x \in X_t$);
- оценочная функция $\varphi(x)$;
- процедуры оценки и выбора хода.

Дерево игры

Весь процесс игры можно представить в виде дерева, узлы которого служат состояниями, а ребра ходами. На рис. 2.13 приведено дерево игры для двух игроков MAX и MIN.

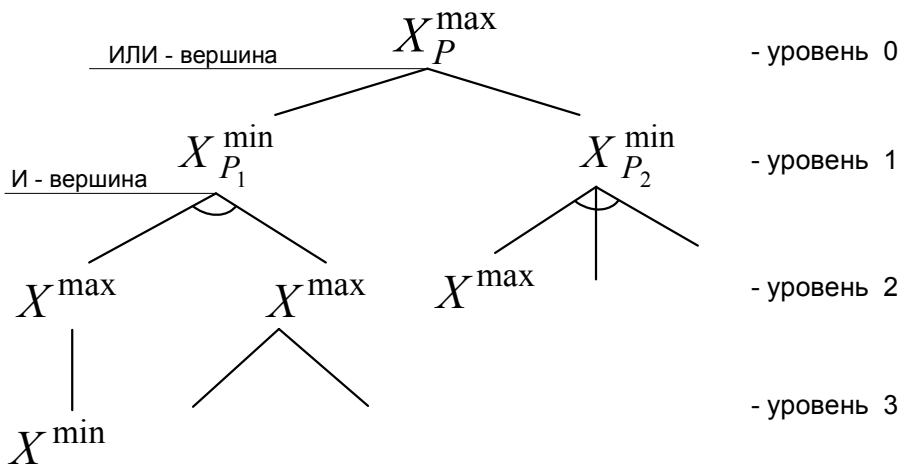


Рис. 2.13. Пример дерева игры для двух игроков

Надпись в узле обозначает отдельного игрока, имеющего право хода в данной ситуации. Позиции P_1, P_2, - приемники позиции P. Также различают уровни позиции. Цель построения дерева игры заключается в определении выигрышной стратегии для одного из игроков (в нашем случае это игрок $X^{\max}$), отправляясь от некоторой фиксированной конфигурации (позиции) игры (не обязательно начальной) независимо от ответов противника.

Для этого могут быть применены разные алгоритмы поиска на И/ИЛИ – графах. Решающее дерево заканчивается на позициях, выигрышных для конкретного игрока, и содержит полную стратегию достижения им выигрыша: для каждого возможного продолжения игры, выбранного противником, в дереве есть ответный ход, приводящий к победе. Лист дерева соответствует состояниям игры, в которых она заканчивается ничьей, проигрышем или выигрышем одного из игроков.

Заметим, что для конфигураций, где ход принадлежит игроку MAX, в игровом дереве получается ИЛИ - вершина, а для позиций, в которых ходит игрок, MIN – И - вершина.

Порождающие процедуры

Задача порождающих процедур состоит в формировании дерева игры, то есть генерация приемников для каждой позиции. В общем

случае процесс формирования дерева - это упорядоченный перебор возможных конфигураций игры. Используется два типа перебора: в ширину и в глубину.

На рис. 2.14 приведено дерево игры двух игроков, полученное в результате применения алгоритма перебора в ширину. Уровень глубины равен четырем. Вершины занумерованы в порядке их генерации.

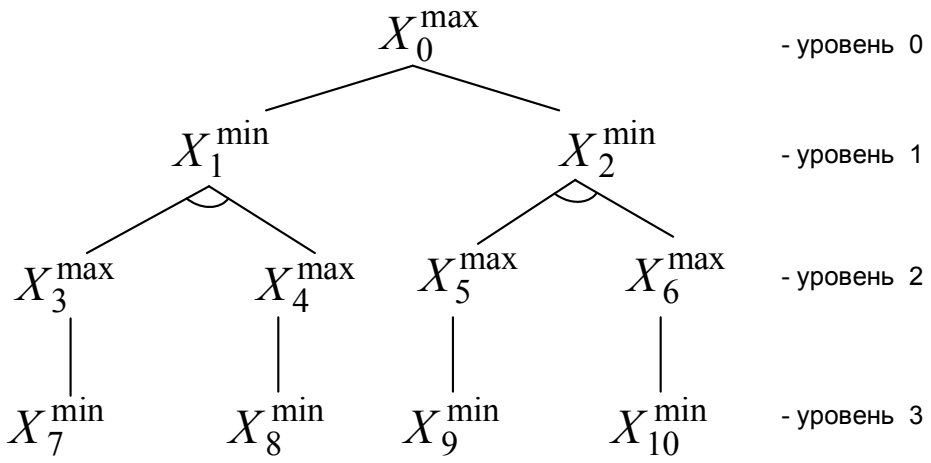


Рис. 2.14. Пример дерева игры, построенного перебором в ширину

На рис. 2.15 показано дерево игры, построенное алгоритмом перебора в глубину. Граничная глубина равна четырем. Вершины занумерованы в том порядке, в котором они были раскрыты.

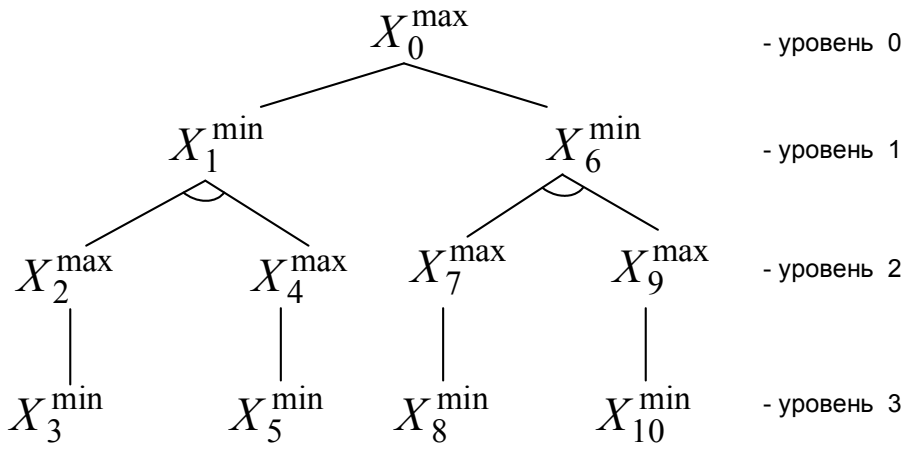


Рис. 2.15 Пример дерева игры, построенного перебором в глубину

Перечисленные алгоритмы перебора имеют как достоинства, так и недостатки по сравнению друг с другом. Главным требованием их применения является ограничение глубины перебора, в противном случае алгоритм никогда не закончит свою работу. Действительно, в

большинстве игр (шахматы, шашки) невозможно построить полное дерево. Так, в шашках общее число вершин оценивается как 10^{40}.

Критерии окончания игры

Предназначены для указания прекращения перебора возможных конфигураций игры. Среди них можно выделить следующие:

- задание значения максимальной глубины;

- мертвая позиция, например в шахматах это позиция, в которой невозможны немедленный ход, взятие фигуры, шах.

Значение максимальной глубины может быть как статическим, так и динамическим. Его выбор может выполняться, например, путем ограничения времени хода игрока.

Оценочная функция

Назначение оценочной функции заключается в определении достоинства игровой позиции. Вычисление оценочной функции позволяет дать эвристическую оценку шансов на выигрыш одного из игроков.

Оценочная функция, как правило, является статической и чаще всего принимает линейную форму $\varphi(x) = c_1 y_1 + c_2 y_2 + \ldots + c_n y_n$,

где y_i – признак позиции, то есть параметр, характеризующий развитие игры;

c_i – вес соответствующего признака y_i.

Например, в шашках $\varphi(x) = 6k + 4m + u$,

где k – перевес в дамках;

m – перевес в шашках;

u – перевес в подвижности.

Значения 6, 4, 1 соответствуют весам признаков.

Выбор оценочной функции, а также глубина перебора являются одними из главных факторов, определяющих алгоритм игры.

Выбор или создание хорошей оценочной функции является необходимым, но не достаточным условием создания эффективной игровой модели с точки зрения интеллектуализации игрового процесса. Важным моментом здесь является подбор алгоритма или процедуры оценки позиций для выбора следующего хода компьютерным противником. При этом большинство алгоритмов игр двух игроков с полной информацией и нулевой суммой выигрышей базируется на

общих идеях минимаксного поиска, а также совершенствующих его различных модификаций.

2.4.2. Алгоритмы оценки игровых ситуаций и выбора хода

Рассмотрим, к примеру, игру «крестики-нолики» на квадрате 3×3. Игрок MAX ходит первым и ставит крестики, а MIN − нолики. Игра заканчивается, когда составлена либо строка, либо столбец, либо диагональ из крестиков (выигрывает MAX) или ноликов (выигрывает MIN). Оценим размер полного дерева игры: начальная вершина имеет 9 дочерних вершин, каждая из которых в свою очередь − 8 дочерних, каждая вершина глубины 2 имеет 7 дочерних и т.д. Таким образом, число концевых вершин в дереве игры равно 9!=362880, но многие пути в этом дереве обрываются раньше на заключительных вершинах. Значит, в этой игре возможен полный просмотр дерева и нахождение выигрышной стратегии. Однако ситуация изменится при существенном увеличении размеров квадрата или в случае неограниченного поля игры.

В таких случаях, как и во всех сложных играх, вместо нереальной задачи поиска полной игровой стратегии решается, как правило, более простая задача − поиск для заданной позиции игры достаточно хорошего первого хода.

С целью поиска достаточно хорошего первого хода просматривается обычно часть игрового дерева, построенного от заданной конфигурации. Для этого применяется один из один из рассмотренных алгоритмов.

После построения таким образом частичного дерева игры вершины в нем оцениваются, и по этим оценкам определяется наилучший ход от заданной игровой конфигурации. При этом для получения оценок концевых вершин (листьев) полученного дерева используется статическая оценочная функция, а для оценивания остальных вершин – корневой (начальной) и промежуточных (между корневой и концевыми вершинами) – используется так называемый минимаксный принцип.

Будем придерживаться общепринятого соглашения, по которому значение статической оценочной функции тем больше, чем

больше преимуществ имеет игрок MAX (над игроком MIN) в оцениваемой позиции. Очень часто оценочная функция выбирается следующим образом:

- статическая оценочная функция положительна в игровых конфигурациях, где игрок MAX имеет преимущества;

- статическая оценочная функция отрицательна в конфигурациях, где MIN имеет преимущества;

- статическая оценочная функция близка к нулю в позициях, не дающих преимущества ни одному из игроков.

Подчеркнем, что с помощью оценочной функции оцениваются только концевые вершины дерева игры, для оценок же промежуточных вершин (и начальной вершины) используется минимаксный принцип, основанный на следующей идее. Если бы игроку MAX пришлось бы выбирать один из нескольких возможных ходов, то он выбрал бы наиболее сильный ход, т.е. ход, приводящий к позиции с наибольшей оценкой. Аналогично, если бы игроку MIN пришлось бы выбирать ход, то он выбрал бы ход, приводящий к позиции с наименьшей оценкой.

Сформулируем теперь сам минимаксный принцип:

- ИЛИ-вершине дерева игры приписывается оценка, равная максимуму оценок ее дочерних вершин;

- И-вершине игрового дерева приписывается оценка, равная минимуму оценок ее дочерних вершин.

Минимаксный принцип положен в основу минимаксной процедуры, предназначенной для определения наилучшего (достаточно хорошего) хода игрока исходя из заданной конфигурации игры S при фиксированной глубине поиска N в игровом дереве [6]. Предполагается, что игрок MAX ходит первым (начальная вершина есть ИЛИ-вершина). Основные этапы этой процедуры таковы:

1. Дерево игры строится (просматривается) одним из известных алгоритмов перебора (как правило, алгоритмом поиска в глубину) от исходной позиции S до глубины N.

2. Все концевые вершины полученного дерева, то есть вершины, находящиеся на глубине N, оцениваются с помощью статической оценочной функции.

3. В соответствии с минимаксным принципом вычисляются оценки всех остальных вершин: сначала вычисляются оценки вершин, родительских для концевых, затем родительских для этих родительских вершин и так далее; таким образом оценивание вершин происходит при движении снизу вверх по дереву поиска – до тех пор, пока не будут оценены вершины, дочерние для начальной вершины, т.е. для исходной конфигурации S.

4. Среди вершин, дочерних к начальной, выбирается вершина с наибольшей оценкой: ход, который к ней ведет, и есть искомый наилучший ход в игровой конфигурации S.

Отмеченные этапы могут быть представлены в виде следующего алгоритма.

```
Function MiniMax(n): double;
{Возвращает g}
begin
  if  n = leaf  then     {n – концевая вершина}
    return  eval(n)     {возвращает исход игры}
  else
    if  n = max  then {n – max вершина}
    begin
     g: = - ∞;{- ∞ – очень малое отрицательное число}
     c:= firstchild (n);
     while c ≠⊥ do  {до тех пор, пока есть потомки }
     begin
       g:= max(g, MiniMax(c));
       c:= nextbrother(c);
     end;
    end
    else {n – min вершина}
    begin
      {+ ∞ – очень большое положительное число}
      g: = + ∞;
      c:= firstchild (n);
      while c ≠⊥ do
      begin
       g:= min(g, MiniMax(c));
       c:= nextbrother(c);
      end;
   end;
  return g
end;
```

Данный алгоритм возвращает минимаксное значение корня – исход игры, если оба игрока будут играть наилучшим образом. Если вершина n – концевая (лист), то функция eval() возвращает исход игры. В противном случае значение узла будет либо максимумом среди значений его потомков, либо минимумом.

На рис. 2.16 показано применение минимаксной процедуры для дерева игры, построенного до глубины N = 3. Концевые вершины не имеют имен, они обозначены своими оценками − значениями статической оценочной функции. Числовые индексы имен остальных вершин показывают порядок, в котором эти вершины строились алгоритмом перебора в глубину. Рядом с этими вершинами находятся их минимаксные оценки, полученные при движении в обратном (по отношению к построению дерева) направлении. Таким образом, наилучший ход – первый из двух возможных.

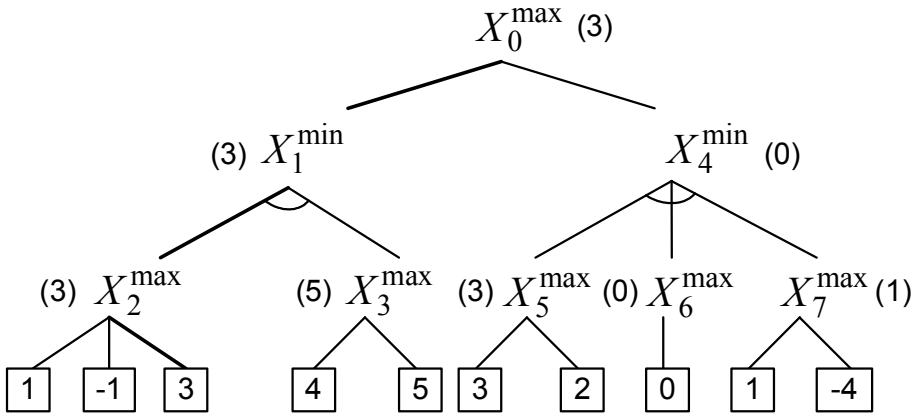


Рис. 2.16 Применение минимаксной процедуры для дерева игры

На рассматриваемом игровом дереве выделена ветвь (последовательность ходов игроков), представляющая так называемую минимаксно-оптимальную игру (или основной вариант игры), при которой каждый из игроков всегда выбирает наилучший для себя ход. Заметим, что оценки всех вершин этой ветви дерева совпадают, и оценка начальной вершины равна оценке концевой вершины этой ветви.

В принципе оценочную функцию можно было бы применить и к промежуточным вершинам, и на основе этих оценок выбрать наилучший первый ход, например сразу выбрать ход, максимизирующий значение оценочной функции среди вершин, дочерних к исходной. Однако считается, что оценки, полученные с помощью минимаксной

процедуры, есть более надежные меры относительного достоинства промежуточных вершин, чем оценки, полученные прямым применением оценочной функции. Действительно, минимаксные оценки основаны на просмотре игры вперед и учитывают разные особенности, которые могут возникнуть в последующем, в то время как простое применение оценочной функции учитывает лишь статические свойства позиции как таковой. Это отличие статических и минимаксных оценок существенно для «активных», динамичных позиций игры (например, в шашках и шахматах к ним относятся конфигурации, в которых возникает угроза взятия одной или нескольких фигур). При так называемых «пассивных», спокойных позициях статическая оценка обычно мало отличается от оценки по минимаксному принципу.

Рассмотрим еще один пример игры, для которой используется минимаксная процедура.

Два игрока (Max и Min) играют в следующую игру. Имеется три кучки камней, содержащих соответственно 2, 3 и 4 камня. За один ход разрешается или удвоить количество камней в какой-нибудь кучке, или добавить по два камня в каждую из трех кучек. Предполагается, что у каждого игрока имеется неограниченный запас камней.

Выигрывает тот игрок, после чьего хода в какой-нибудь кучке становится ≥ 15 камней или во всех трех кучках суммарно становится ≥ 25 камней. Игроки ходят по очереди.

Определим выигрышную стратегию игры для первого игрока Max. Для этого построим дерево игры (рис. 2.17).

Представленное на рисунке дерево игры построено методом перебора в глубину до уровня 3. В качестве оценочной функции использовалось максимальное значение камней среди всех кучек, значение «$+\infty$» соответствует выигрышу игрока Max, «$-\infty$» - игрока Min. Каждая вершина дерева имеет значение оценочной функции (вверху) и порядок раскрытия (внизу).

Следуя принципу минимакса можно определить лучший ход для игрока Max. Этот ход [2, 3, 4] -> [4, 5 ,6], поскольку при любом варианте хода игрока Min игрок Max выигрывает.

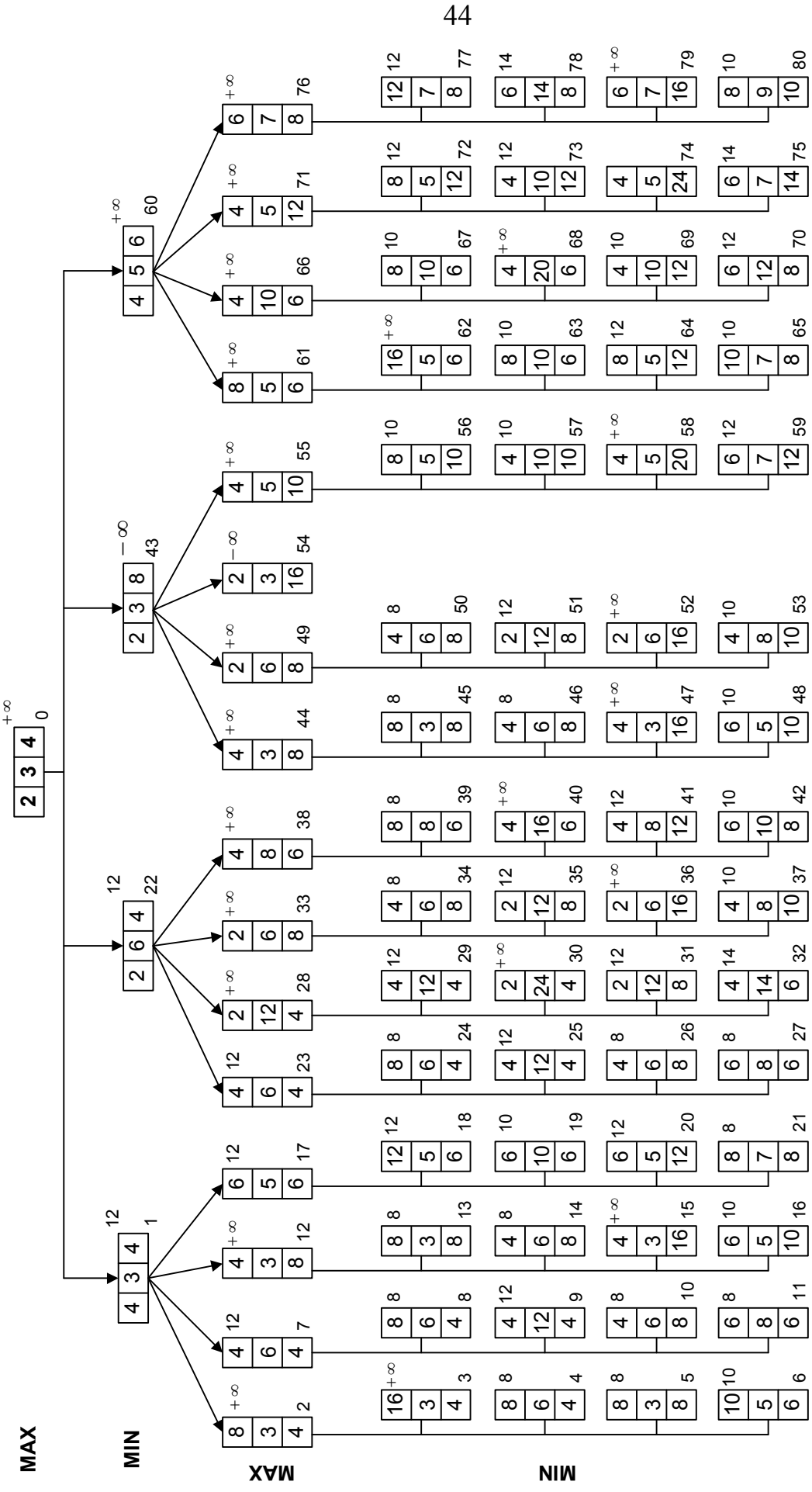


Рис. 2.17. Дерево игры

Таким образом, для определения выигрышного хода первого игрока пришлось перебрать все вершины до установленной глубины. Такой подход является наиболее простым и гарантирующим определение лучшего хода. Однако его вычислительная эффективность сильно зависит от значения глубины дерева и если оно большое, то использование минимаксной процедуры может стать затруднительным с точки зрения времени.

В качестве примера рассмотрим фрагмент дерева игры «крестики - нолики» с правом выбора хода игроком MAX (рис. 2.18).

Статическая оценочная функция для этой игры имеет вид:

$+\infty$, если P – позиция выигрыша игрока MAX;

$-\infty$, если P – позиция выигрыша игрока MIN;

$(N_L^{\max} + N_c^{\max} + N_D^{\max}) - (N_L^{\min} + N_c^{\min} + N_D^{\min})$ - в остальных случаях;

$N_L^{\max}$, $N_c^{\max}$, $N_D^{\max}$ - соответственно число строк, столбцов и диагоналей открытых для игрока MAX (то есть где он еще может поставить выигрышные три крестика подряд).

$N_L^{\min}$, $N_c^{\min}$, $N_D^{\min}$ - аналогичные числа для игрока MIN.

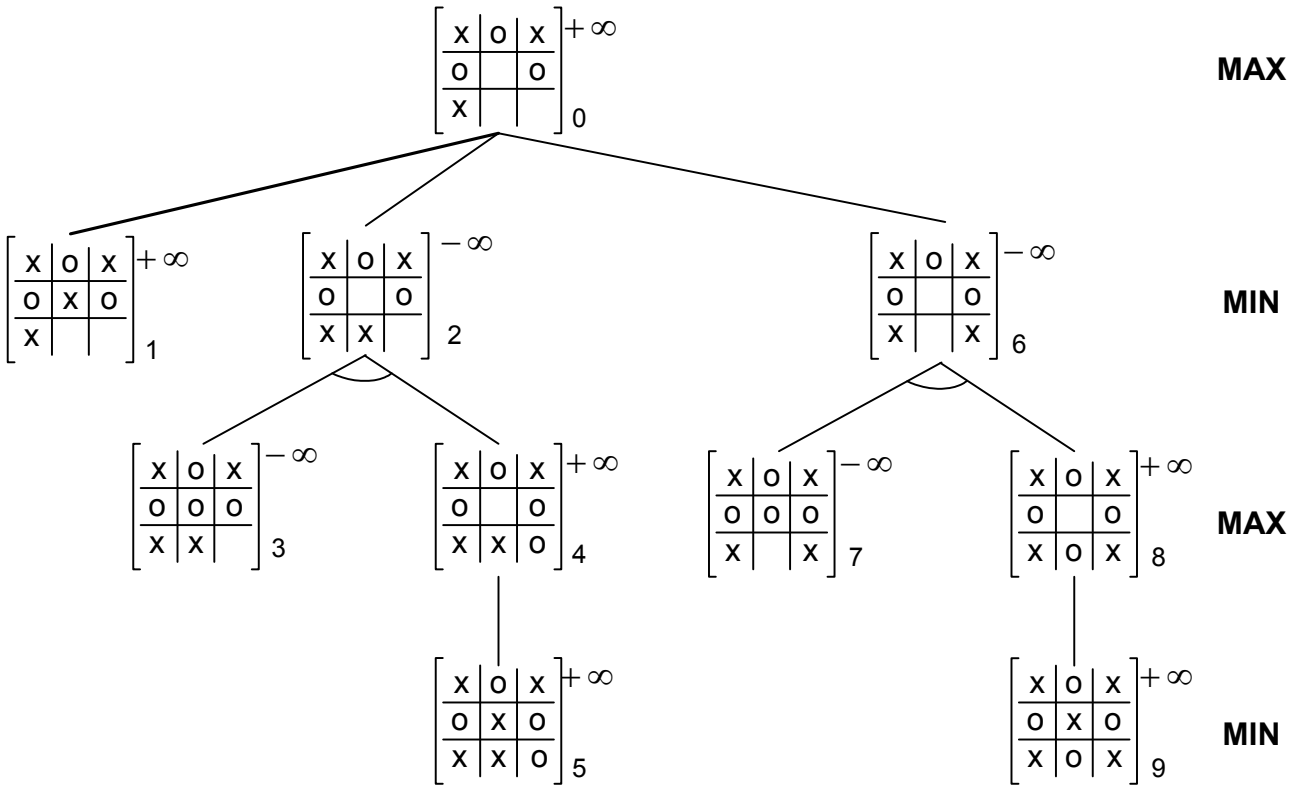


Рис. 2.18. Фрагмент дерева игры «крестики - нолики»

На рис 2.18 представлен фрагмент дерева игры, каждая вершина которого имеет оценку и порядок раскрытия. Оценка листьев дерева вычисляется оценочной функцией, остальные вершины – на основе принципа минимакса. Таким образом возможны три варианта развития игры для игрока MAX. Только один из них является выигрышным.

Допустим, дерево имеет коэффициент ветвления b и глубину d. Минимаксный алгоритм делает обход всего дерева и его сложность равна $O(b^d)$, что не является эффективным. Как видно из рис.2.18, следующий выигрышный ход для игрока MAX мог быть получен без просмотра всего дерева. Однако минимаксная процедура организована таким образом, что процесс построения частичного дерева игры отделен от процесса оценивания вершин.

Такое разделение приводит к тому, что в целом минимаксная процедура − неэффективная стратегия поиска хорошего первого хода. Чтобы сделать процедуру более экономной, необходимо вычислять статические оценки концевых вершин и минимаксные оценки промежуточных вершин одновременно с построением игрового дерева. Этот путь приводит к так называемой альфа-бета процедуре поиска наилучшего первого хода от заданной позиции, на нем можно добиться существенного сокращения вычислительных затрат, прежде всего времени вычисления оценок. В основе такого сокращения поиска лежит достаточно очевидное соображение: если есть два варианта хода одного игрока, то худший в ряде случаев можем сразу отбросить, не выясняя, насколько в точности он хуже.

Рассмотрим сначала идею работы альфа-бета процедуры на примере игрового дерева, приведенного на рис. 2.16. Дерево игры строится до глубины N=3 алгоритмом перебора в глубину. Причем сразу, как это становится возможным, вычисляются не только статические оценки концевых вершин, но и предварительные минимаксные оценки промежуточных вершин. Предварительная оценка определяется соответственно как минимум или максимум уже известных (к настоящему моменту) оценок дочерних вершин, и она может быть получена при наличии оценки хотя бы одной дочерней вершины. Предварительные оценки затем постепенно уточняются по

минимаксному принципу – по мере получения новых (предварительных и точных) оценок в ходе дальнейшего построения дерева.

Пусть таким образом построены вершины $X_1^{\min}$, $X_2^{\max}$ и первые три конечные вершины (рис. 2.19). Эти листья оценены статической функцией, и вершина $X_2^{\max}$ получила точную минимаксную оценку 3, а вершина $X_1^{\min}$ − предварительную оценку 3. Далее при построении и раскрытии вершины $X_3^{\max}$ статическая оценка первой ее дочерней вершины дает величину 4, которая становится предварительной оценкой самой вершин $X_3^{\max}$. Эта предварительная оценка будет потом (после построения второй дочерней вершины) пересчитана, причем согласно минимаксному принципу она может только увеличиться (так как равна максимуму оценок дочерних вершин), но даже если она увеличится, это не повлияет на оценку вершины $X_1^{\min}$, поскольку последняя при уточнении по минимаксному принципу может только уменьшаться (она равна минимуму оценок дочерних вершин). Следовательно, можно пренебречь второй дочерней вершиной (равна 5) для $X_3^{\max}$, не строить и не оценивать ее (так как уточнение оценки вершины $X_3^{\max}$ не повлияет на оценку вершины $X_1^{\min}$). Такое сокращение процедуры поиска в дереве называется отсечением ветвей дерева.

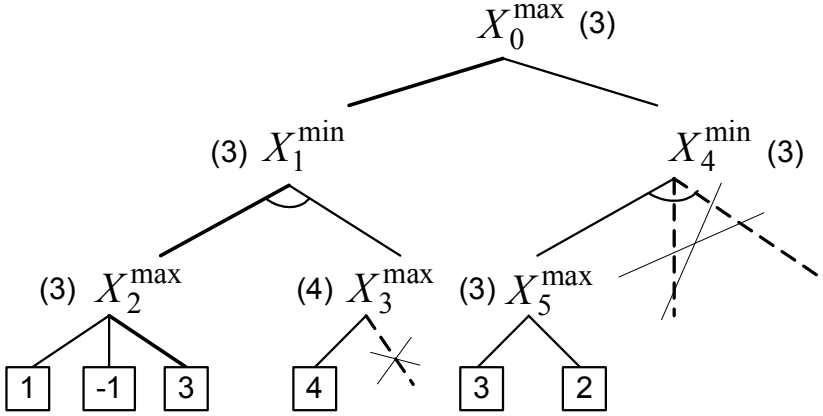


Рис.2.19. Применение альфа-бета процедуры для дерева игры

Продолжим для нашего примера процесс поиска в глубину с одновременным вычислением предварительных (и точных, где это возможно) оценок вершин вплоть до момента, когда построены уже вершины $X_4^{\min}$, $X_5^{\max}$ и две дочерних последней, которые оцениваются статической функцией. Исходя из оценки первой дочерней

вершины начальная вершина $X_0^{\max}$, соответствующая исходной позиции игры, к этому моменту уже предварительно оценена величиной 3. Вершина $X_5^{\max}$ получила точную минимаксную оценку 3, а ее родительская $X_4^{\min}$ получила пока только предварительную оценку 3. Эта предварительная оценка вершины $X_4^{\min}$ может быть уточнена в дальнейшем, но в соответствии с минимаксным принципом возможно только ее уменьшение, а это уменьшение не повлияет на оценку вершины $X_0^{\max}$, поскольку последняя, опять же согласно минимаксному принципу, может только увеличиваться. Таким образом, построение дерева можно прервать ниже вершины $X_4^{\min}$, отсекая целиком выходящие из нее вторую и третью ветви и оставляя ее оценку предварительной.

На этом построение рассматриваемого игрового дерева заканчивается, полученный результат − лучший первый ход − тот же самый, что и при минимаксной процедуре. У некоторых вершин дерева осталась не уточненная, предварительная оценка, однако этих приближенных оценок оказалось достаточно для того, чтобы определить точную минимаксную оценку начальной вершины и наилучший первый ход. В то же время произошло существенное сокращение поиска: вместо 17 вершин построено только 11, и вместо 10 обращений к статической оценочной функции понадобилось всего 6.

Обобщим рассмотренную идею сокращения перебора. Для этого сформулируем правила вычисления оценок вершин дерева игры, в том числе предварительных оценок промежуточных вершин, которые назовем альфа- и бета - величинами.

1. Концевая вершина дерева оценивается статической оценочной функцией сразу, как только она построена.

2 Промежуточная вершина предварительно оценивается по минимаксному принципу, как только стала известна оценка хотя бы одной из ее дочерних вершин, при этом каждая предварительная оценка пересчитывается (уточняется) всякий раз, когда получена оценка еще одной дочерней вершины.

3. Предварительная оценка ИЛИ-вершины (альфа-величина) полагается равной наибольшей из вычисленных к текущему моменту оценок ее дочерних вершин.

4. Предварительная оценка И-вершины (бета-величина) полагается равной наименьшей из вычисленных к текущему моменту оценок ее дочерних вершин.

Следствием из этих правил является то, что альфа-величины не могут уменьшаться, а бета-величины не могут увеличиваться.

Сформулируем теперь правила прерывания перебора, или отсечения ветвей игрового дерева.

1. Перебор можно прервать ниже любой И-вершины, бета-величина которой не больше, чем альфа-величина одной из предшествующих ей ИЛИ-вершин (включая корневую вершину дерева).

2. Перебор можно прервать ниже любой ИЛИ-вершины, альфа-величина которой не меньше, чем бета-величина одной из предшествующих ей И-вершин.

При этом в случае 1 говорят, что имеет место альфа-отсечение (поскольку отсекаются ветви дерева, начиная с ИЛИ-вершин, которым приписана альфа-величина), а в случае 2 − бета-отсечение (поскольку отсекаются ветви, начинающиеся с бета-величин).

Важно, что рассмотренные правила работают в ходе построения игрового дерева вглубь. Это означает, что предварительные оценки промежуточных вершин появляются лишь по мере продвижения от концевых вершин дерева вверх к корню и реально отсечения могут начаться только после того, как получена хотя бы одна точная минимаксная оценка промежуточной вершины.

В рассмотренном примере на рис. 2.19 первое прерывание перебора было альфа -отсечением, а второе − бета-отсечением. Причем в обоих случаях отсечение было неглубоким, поскольку необходимая для соблюдения соответствующего правила отсечения предварительная оценка (альфа- или бета-величина) находилась в непосредственно предшествующей (к точке отсечения) вершине. В общем же случае она может находиться существенно выше отсекаемой ветви – на пути от вершины, ниже которой производится отсечение, к начальной вершине, включая последнюю.

После прерывания перебора предварительные оценки вершин в точках отсечения остаются неуточненными, но, как уже отмечалось, это не препятствует правильному нахождению предварительных

оценок всех предшествующих вершин, как и точной оценки корневой вершины и ее дочерних вершин, а значит, и искомого наилучшего первого хода.

Формализуем все изложенное в виде компьютерного алгоритма. Для каждого узла введем два параметра: a – нижняя граница его значения, b – верхняя граница.

```
Function Alpha_Beta (n, a, b): double;
{Возвращает g}
begin
  if  n = leaf  then     {n – концевая вершина}
   return  eval(n)     {возвращает исход игры}
  else
   if  n = max  then {n – max вершина}
   begin
     {- ∞ – очень малое отрицательное число}
     g: = - ∞;
     c:= firstchild (n);
     {до тех пор, пока есть потомки }
     while  g < b  and  c ≠⊥  do
     begin
       g:= max(g, Alpha_Beta(c, a, b));
       a:= max(a, g);
       c:= nextbrother(c);
     end;
   end
   else {n – min вершина}
    begin
      {+ ∞ – очень большое положительное число}
      g: = + ∞;
      c:= firstchild (n);
      while g > a  and  c ≠⊥  do
      begin
        g:= min(g, Alpha_Beta(c, a, b));
        b:= min(b, g);
        c:= nextbrother(c);
      end;
    end;
  return g
end;
```

В max – узле переменная g является нижней границей его значения, которое после обновления a:= max(a, g) будет передаваться в

рекурсивные вызовы Alpha_Beta в качестве параметров. Аналогично обрабатываются и min - узлы, где g – текущая верхняя граница значения узла. Отсечение дерева осуществляется условиями while g < b и while g > a соответственно.

Пару параметров a и b называют окном поиска. Предполагается, что каждый вызов Alpha_Beta (n, a, b) должен вернуть значение, лежащее в интервале (a, b). Как только выясняется, что значение выходит за рамки допустимого окна (a, b), поиск прекращается. Для корня дерева делается вызов Alpha_Beta (root, -∞, + ∞), и тем самым образом гарантируется правильность результата. В процессе обработки узлов окно поиска сужается, делая в результате все больше и больше отсечений.

Таким образом альфа-бета-процедура является более эффективной реализацией минимаксного принципа и всегда приводит к тому же результату (наилучшему первому ходу), что и простая минимаксная процедура той же глубины.

Важным является вопрос, насколько в среднем альфа-бета процедура эффективнее обычной минимаксной процедуры. Нетрудно заметить, что число отсечений в альфа-бета процедуре зависит от степени, в которой полученные первыми предварительные оценки (альфа-бета-величины) аппроксимируют окончательные минимаксные оценки: чем ближе эти оценки, тем больше отсечений и меньше перебор. Это положение иллюстрирует пример на рис. 2.18, в котором основной вариант игры обнаруживается практически в самом начале поиска.

Наилучший случай (наибольшее число отсечений) достигается, когда при переборе в глубину первой обнаруживается конечная вершина, статическая оценка которой станет впоследствии минимаксной оценкой начальной вершины. При максимальном числе отсечений требуется строить и оценивать минимальное число концевых вершин. Показано, что в случае, когда самые сильные ходы всегда рассматриваются первыми, число концевых вершин глубины N, которые будут построены и оценены альфа-бета-процедурой, примерно равно числу концевых вершин, которые были бы построены и оценены на глубине N/2 обычной минимаксной процедурой. Таким образом, при фикси-

рованном времени и памяти альфа-бета-процедура сможет пройти при поиске вдвое глубже по сравнению с обычной минимаксной процедурой.

В среднем алгоритм Альфа-Бета обрабатывает не $O(b^d)$, а $O(b^{d/2})$ узлов, то есть его сложность - квадратный корень от сложности минимакса. Но в принципе возможен и неудачный порядок просмотра, при котором придется пройти (без отсечений) через все вершины дерева, и в этом, худшем случае, альфа-бета-процедура не будет иметь никаких преимуществ против минимаксной процедуры. В связи с этим разрабатывались и разрабатываются усовершенствованные алгоритмы альфа-бета-процедуры, например с применением сортировки дочерних узлов (max – узлов по возрастанию, min – узлов по убыванию значений) или на основе таблиц транспозиций, позволяющих отслеживать и сохранять возможные повторяющиеся позиции, тем самым устраняя необходимость в их повторном поиске.

Контрольные вопросы

1. Что такое понятие эвристика и какие существуют направления эвристического программирования?
2. Как представляется пространство состояний и какие существуют типы алгоритмов поиска в нем?
3. В чем состоит отличие слепого перебора от эвристического?
4. В чем состоит алгоритм поиска пути на графе?
5. В чем заключается поиск в пространстве задач?
6. Каковы принципы работы метода GPS?
7. Каковы назначение и особенности игр, где применяется эвристический поиск?
8. Что включает в себя игровая модель поиска?
9. Какие существуют основные критерии окончания игры и в чем назначение оценочной функции?
10. В чем состоит принцип работы минимаксной процедуры оценки позиций?
11. Что такое альфа-бета процедура оценки позиций и каковы ее отличия и преимущества по сравнению с минимаксной?

3. МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Одним из проявлений интеллектуальности компьютерной системы является возможность организации управления в ней знаниями человека (эксперта в некоторой предметной области). Теоретическими и практическими вопросами представления и обработки знаний в компьютерных системах занимается отдельное направление ИИ – инженерия знаний [12]. Программы, использующие достижения в этой области, образуют отдельный класс компьютерных систем, основанный на знаниях. Главными структурными элементами таких систем являются базы знаний и механизм логических выводов.

С позиций ИИ под знаниями подразумевают информацию, которая необходима программе, чтобы она вела себя «интеллектуально». Обычно такая информация представляется в виде фактов и правил, которые ЭВМ может использовать при решении задач по таким алгоритмам, как логические выводы. Знания можно разделить на декларативные и процедурные. Декларативные знания представляют собой описания фактов и явлений, а также связанных с ними закономерностей. Процедурные знания – это описание действий, которые возможны при манипулировании фактами и явлениями для достижения намеченных целей.

К типовым моделям представления знаний принято относить логическую, продукционную, фреймовую и модель семантической сети. Кроме этого, на практике часто находят применение их комбинированные сочетания.

3.1. Логическое представление знаний

Логика как наука была создана Аристотелем (384 – 322 г. до н.э.). Логика – это наука о рассуждениях, которая позволяет определить истинность или ложность того или иного математического утверждения.

В основе логической модели лежит система исчисления предикатов первого порядка, которая, в свою очередь, основана на исчислении высказываний. Под высказыванием будем понимать утверждение, о котором в данной ситуации можно сказать, истинно оно или ложно.

Пример

А: «Сегодня четверг» - ложно или истинно в зависимости от текущего дня недели.

В: «2+2=4» – абсолютно истинное высказывание.

Эти высказывания являются элементарными или простыми, потому что их нельзя разделить на части. Из простых высказываний могут быть построены сложные с помощью известных логических операций: отрицания ($\bar{}$), конъюнкции ($\wedge$), дизъюнкции ($\vee$), импликации ($\rightarrow$), эквиваленции ($\equiv$), исключающего ИЛИ ($\oplus$). При этом любое сложное высказывание может быть упрощено до выражения, содержащего только основные логические операции ($\bar{}$, $\wedge$, $\vee$).

Пример

$$(A \rightarrow B) \vee (\overline{A \wedge B}) = (\overline{A} \vee B) \vee (\overline{A} \vee \overline{B}) = \overline{A} \vee B \vee \overline{A} \vee \overline{B} = 1$$

Для упрощения используются следующие эквивалентные преобразования.

$$A \rightarrow B = \overline{A} \vee B;$$

$$A \equiv B = (A \rightarrow B) \wedge (B \rightarrow A) = (A \wedge B) \vee (\overline{A} \wedge \overline{B});$$

$$A \oplus B = (\overline{A} \wedge B) \vee (A \wedge \overline{B});$$

$$\overline{A \vee B} = \overline{A} \wedge \overline{B};$$

$$\overline{A \wedge B} = \overline{A} \vee \overline{B};$$

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C);$$

$$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C).$$

В процессе упрощения любая логическая формула может быть приведена к эквивалентной ей конъюнктивной нормальной форме (КНФ). КНФ представляет собой конечное число дизъюнктов, объединенных операцией конъюнкции.

Пример

$$(A \to B) \wedge (B \to C) \wedge (\overline{A \wedge C}) = (\overline{A} \vee B) \wedge (\overline{B} \vee C) \wedge (\overline{A} \vee \overline{C})$$

Для проверки истинности высказываний используются таблицы истинности (табл. 3.1).

Таблица 3.1

Таблицы истинности логических операций

A	B	$A \wedge B$	$A \vee B$	$A \to B$	$A \approx B$	$A \oplus B$	$\overline{A}$
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	1
1	0	0	1	0	0	1	0
1	1	1	1	1	1	0	0

Каждая логическая операция имеет множество правил интерпретации (табл. 3.2), на основании которых могут быть эффективно решены задачи описания знаний из различных прикладных областей.

Таблица 3.2

Правила интерпретации логических операций

Операция	**Интерпретация**
$\overline{A}$	«Не», «Неверно»
$A \wedge B$	«И», «Одновременно»
$A \vee B$	«Или», «Хотя бы один»
$A \to B$	«Если А то В», «В необходимо для А», «А только, если В», «А достаточно для В», «В тогда, когда А», «А только тогда, когда В»
$A \equiv B$	«Эквивалентно», «Равносильно»
$A \oplus B$	«Либо А, либо В», «Или А, или В»

Пример

На станцию должны прибыть три пассажирских поезда А, В и С. Ожидающие прибытия поездов пассажиры высказали следующие предположения:

- для того, чтобы вовремя прибыл поезд А, не достаточно, чтобы вовремя прибыл поезд В;

- поезд С опоздает или, если вовремя прибудет поезд А, то вовремя прибудет поезд В;

- прибытие по расписанию поезда А не является необходимым условием прибытия вовремя хотя бы одного из поездов В или С.

Определить, какие из поездов прибыли вовремя.

Выделим из этого предложения элементарные высказывания.

A: «Поезд A прибудет вовремя».

B: «Поезд B прибудет вовремя».

C: «Поезд C прибудет вовремя».

Запишем каждое из предположений на языке логики в соответствии с приведенными правилами интерпретации.

1. $\overline{B \to A}$;

2. $\overline{C} \vee (A \to B)$;

3. $\overline{(B \vee C) \to A}$.

Составим общее логическое выражение и упростим его.

$$(\overline{B \to A}) \cdot (\overline{C} \vee (A \to B)) \cdot \overline{((B \vee C) \to A)} = (\overline{\overline{B} \vee A}) \cdot (\overline{C} \vee \overline{A} \vee B) \cdot (\overline{\overline{B \vee C} \vee A}) =$$
$$(\overline{A}B) \cdot (\overline{C} \vee \overline{A} \vee B) \cdot (\overline{A} \cdot (B \vee C)) = (\overline{A}BC \vee \overline{A}B \vee \overline{A}B) \cdot (\overline{A}B \vee \overline{A}C) =$$
$$\overline{A}B \cdot (\overline{C} \vee 1 \vee 1) \cdot (\overline{A}B \vee \overline{A}C) = \overline{A}B \vee \overline{A}BC = \overline{A}B$$

Из полученных результатов можно сделать вывод, что поезд A – не придет вовремя, B - прибудет вовремя, C – может как опоздать, так и прийти вовремя.

Решение подобных задач связано с доказательством истинности или выполнимости логической формулы, то есть поиском такого набора значений переменных, при котором она равна «истине».

Одной из ключевых сторон представления знаний с помощью логических моделей является возможность моделирования рассуждений. В этом случае применяется так называемый дедуктивный вывод от общего к частному. С его помощью могут быть решены такие задачи, как поиск следствия и поиск доказательства. Вывод представляет собой процедуру, которая из заданной группы выражений (логических утверждений) выводит новое логическое выражение или следствие. Для этого могут быть использованы специальные правила вывода, состоящие из посылок или гипотез и утверждения, называемого заключением. Правильным заключением называется такое, которое истинно всякий раз, когда истинны гипотезы. Для описания правил вывода часто используется нотация, при которой над чертой

записывается группа выражений, принимаемых за посылки, а под чертой записывается заключение.

Рассмотрим наиболее известные из существующих правил вывода [1, 12].

1. Modus Ponendo Ponens: «Если истинна импликация $A \rightarrow B$ и A истинно, то B истинно ».

2. Modus Tollendo Tollens: «Если истинна импликация $A \rightarrow B$ и B ложно, то A ложно».

3. Modus Ponendo Tollens: «Если A истинно и конъюнкция $A \wedge B$ ложна, то B ложно».

4. Modus Tollendo Ponens: «Если A ложно и дизъюнкция $A \vee B$ ложна, то B ложно ».

5. Цепное заключение: «Если истинна импликация $A \rightarrow B$ и истинна импликация $B \rightarrow C$, то импликация $A \rightarrow C$ является истинной ».

Рассмотрим пример вывода с применением этих правил. Пусть заданы следующие посылки.

1. $P \rightarrow Q$: Если растут мировые цены на топливно-энергетические ресурсы, то увеличиваются поступления в бюджет.

2. $(R \vee Q) \rightarrow (R \vee S)$: Если наблюдается рост производства или увеличиваются поступления в бюджет, то следует увеличение производства или укрепление рубля.

3. $(P \rightarrow Q) \rightarrow ((P \vee Q) \rightarrow (R \vee Q))$: Если растут мировые цены на топливно-энергетические ресурсы, то увеличиваются поступления в бюджет, из чего следует, что при росте цен на сырье или при увеличении поступлений в бюджет происходит рост производства или увеличение бюджета.

Используя правило 1, из посылок 1 и 3 можно вывести следующее заключение.

4. $(P \vee Q) \rightarrow (R \vee Q)$: Если растут мировые цены на топливно-энергетические ресурсы или увеличиваются поступления в бюджет, то происходит рост производства или увеличение бюджета.

Из посылок 4 и 2, используя правило 5, можно получить посылку 5.

5. $(P \vee Q) \rightarrow (R \vee S)$: Если растут мировые цены на топливно-энергетические ресурсы или увеличиваются поступления в бюджет, то происходит рост производства или укрепление рубля.

Таким образом, применяя правила логического вывода, можно получить новые логические формулы на основании исходных без использования таблиц истинности. Однако при значительном числе исходных данных возможно получение большого числа цепочек вывода, результаты которых могут противоречить друг другу. Для преодоления таких проблем должно быть организовано специальное управление стратегией логического вывода.

У рассмотренных правил вывода есть одна особенность: они являются тавтологиями, то есть логическими формулами, значение которых будет «истина» при любых значениях, входящих в них переменных. Докажем, что тавтологией является первое правило вывода, для этого построим таблицу истинности (табл. 3.3).

Таблица 3.3

Таблицы истинности для доказательства тавтологии

A	**B**	**A→B**	**A ∧ (A→B)**	**A∧ (A→B) →B**
0	0	1	0	1
0	1	1	0	1
1	0	0	0	1
1	1	1	1	1

Такие тождественно истинные формулы также называются аксиомами исчисления высказываний. Множество аксиом образуют теорию заданной предметной области. Цель построения теории заключается в описании нужных знаний наиболее компактным способом. Если удается построить теорию, которая адекватно описывает заданную область знаний, то все истинные факты будут следствием аксиом этой теории. Таким образом, особенностью логического представления знаний является наличие возможности определения, имеются или отсутствуют противоречия между новыми и уже существующими знаниями.

Для тавтологий может быть применено правило подстановки: если *C(A)* – тавтология и вместо всех вхождений формула *A* в *C* подставить формулу *B*, то *C(B)* – тавтология.

В противовес тавтологиям в логике существуют противоречия – формулы, значением которых всегда будет «ложь» независимо от значений входящих в них переменных. Примером противоречия является выражение $A \wedge \overline{A} = \varnothing$ при любом A.

Другим важным направлением в исчислении высказываний, наряду с поиском следствия, является автоматизация доказательства теорем.

Доказательством будем называть конечный список формул $B_1,...,B_n$, каждая из которых или является некоторой аксиомой исчисления высказываний, или получена по правилам вывода из некоторой пары формул, предшествующих в этом списке.

Доказательство истинности полученной формулы может быть проведено с помощью таблицы истинности (при небольшом числе переменных), а также другими методами, одним из которых является принцип резолюций, предложенный Дж. А. Робинсоном в 1965 г. Этот принцип является правилом вывода и его основное достоинство состоит в возможности полной компьютеризации [12].

Известно, что КНФ не выполнима тогда и только тогда, когда ложен хотя бы один дизъюнкт. Принцип резолюции состоит в том, что невыполнимость формулы можно проверить, порождая логические следствия из нее до тех пор, пока не получится противоречие.

Пусть *S1* и *S2* – дизъюнкты КНФ *S*, *L* – переменная.

Если $L \in S1$ и $\overline{L} \in S2$, то дизъюнкт $R = (S1 \setminus \{L\}) \vee (S2 \setminus \{\overline{L}\})$ - логическое следствие КНФ S (здесь символ «\» означает разность).

Дизъюнкт *R* называется резольвентой дизъюнктов *S1* и *S2*.

При этом *R* является логическим заключением из *S1* и *S2*, следовательно, добавление КНФ S и $S \vee \{R\}$ эквивалентны.

Рассмотрим алгоритм доказательства невыполнимости логических формул, основанный на принципе резолюций.

1. Преобразуем заданные посылки к КНФ S – множеству дизъюнктов.

2. Выбираем *L*, *S1*, *S2*, такие что $L \in S1$ и $\overline{L} \in S2$.

3. Находим *R*.

4. Заменяем S на $S \vee \{R\}$.

5. Продолжаем до тех пор, пока не получим противоречие.

Если процесс не останавливается, то есть множество дизъюнктов пополняется новыми резольвентами, среди которых нет искомого дизъюнкта, то в таком случае нельзя сделать заключение о выводимости формулы.

Пример

Предположим, нужно доказать, что, имея посылки $\overline{P} \to Q$, $\overline{P} \to H$ и $\overline{P}$, нельзя вывести формулу $Q \to \overline{H}$.

1. Сформируем множество дизъюнктов S и пронумеруем их.

$S = \{P \vee Q(1), P \vee H(2), \overline{Q} \vee \overline{H}(3), \overline{P}(4)\}$.

2. Будем находить и добавлять к S резольвенты.

5. $P \vee \overline{H}$ (1, 3); 6. P (2, 5); 7 $\varnothing$ (4, 6).

В скобках указаны *S1*, *S2*. Седьмой дизъюнкт является противоречием, потому что одновременно можем в качестве следствия получить утверждение (P) и его отрицание ($\overline{P}$). Следовательно, нельзя получить из имеющихся посылок заданное следствие.

Отметим, что принцип резолюции послужил основой для создания языка логического программирования Prolog. Работа его интерпретатора заключается в реализации автоматического вывода, подобного приведенному на основе предварительно сформулированных пользователем правил.

Многие утверждения, имеющие форму высказываний, на самом деле ими не являются, так как содержат переменные различных типов, конкретные значения которых не указаны. Поскольку такое утверждение при одних значениях переменных может быть истинным, а при других – ложным, ему не может быть предписано истинное значение. Такое утверждение называется предикатом, а правило описания знаний на основе их исчислением предикатов первого порядка.

Пример

B(x): 3+x = 5;

Предикат с одной переменной называется одноместным, n – n-местным.

Изначально целью системы исчисления предикатов являлось разъяснение логических основ естественного языка.

Считается, что набор значений переменных удовлетворяют предикату, если на этом наборе предикат принимает значение «истина» (*B(2)* – истина, *B(4)* – ложь).

Некоторые предикаты истинны для любого возможного набора значений переменных, а другие только для некоторого. Для обозначения таких ситуаций в исчислении предикатов вводятся специальные символы – кванторы.

$\forall x\, B(x)$ - предикат с квантором всеобщности, имеет интерпретацию «для всех», то есть для любого значения x истинно *B(x)*.

$\exists x\, B(x)$ - предикат с квантором существования, имеет интерпретацию «для некоторого», то есть для некоторого (хотя бы одного значения) x истинно *B(x)*.

С помощью логики предикатов могут быть описаны фразы естественного языка, которые не содержат внутри себя двусмысленностей.

Пример

«Для каждого действия существует равное и противоположно направленное противодействие» - $\forall x \exists y\, R(x, y)$;

$\forall x, y, z\, [Отец(x, y) \wedge Отец(y, z) \rightarrow Дедушка(x, z)]$ - «Если x является отцом y, а y является отцом z, то x является дедушкой z»;

$\forall x\, [Человек(x) \rightarrow (\exists y)\, Отец(x, y)]$ - «Любой человек имеет отца».

Чтобы предикат стал высказыванием, все его переменные должны иметь конкретные значения или быть связанными.

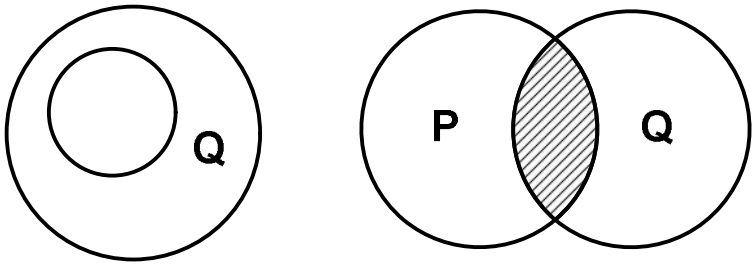
Пример

$P(x, y, z) : x^2 + y^2 \geq z^2$;

$(\exists x)(\forall z)P(x, y, z)$ - не высказывание, так как y - свободная переменная.

Предикаты могут быть использованы для организации дедуктивных рассуждений, первая из известных моделей которых была предложена за 400 лет до нашей эры Аристотелем и называлась силлогистикой. Рассмотрим наиболее простой способ проверки правильности умозаключений на основе диаграмм Эйлера.

Диаграммы Эйлера состоят из кругов, изображающих множества. Так, утверждению «Все P есть Q» будет соответствовать диаграмма, приведенная на рис. 3.1 а.



а) б)

Рис. 3.1. Диаграммы для кванторов всеобщности (а) и существования (б)

Здесь круг, изображающий множество *P*, содержится в круге, изображающем множество *Q*. Другое утверждение, соответствующее квантору существования «Некоторые *P* есть *Q*», приведено на рис. 3.1 б как пересечение множеств *P* и *Q*.

Для проверки умозаключений необходимо попытаться построить диаграмму Эйлера, в которой посылки истинны, а умозаключение ложно. Если такое построение возможно, то умозаключение неправильно.

Пример

Проверить правильности умозаключения.

Все студенты университета – выдающиеся люди;

Все выдающиеся люди – ученые;

Все студенты университета – ученые.

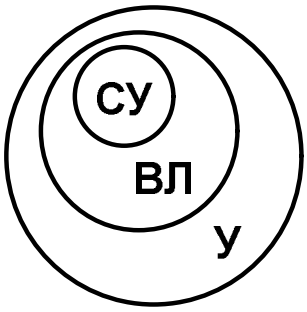


Рис. 3.2. Диаграмма для примера

На рис. 3.2 показана диаграмма, построенная по условию задачи. В соответствии с посылками круг, изображающий студентов университета (СУ), должен быть внутри круга, изображающего выдающихся людей (ВЛ), который, в свою очередь, должен быть внутри круга, изображающего ученых (У). Следовательно, по имеющемуся заключению круг студентов должен находиться внутри круга ученых, то есть умозаключение верно.

Пример

Проверить правильность умозаключения.

Некоторые поэты – неудачники;
Некоторые ученые – неудачники;
Некоторые поэты являются учеными.

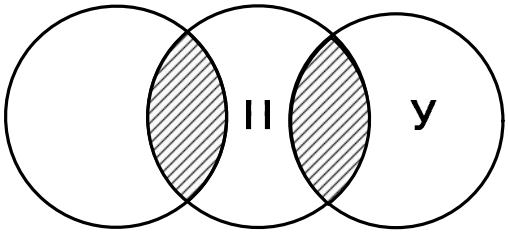


Рис. 3.3. Диаграмма для примера

Как видно из рис. 3.3, можно построить такую диаграмму Эйлера, в которой пересечение кругов поэтов (П) и неудачников (Н) не пусто, поэтому посылки истинны. Однако при этом круги поэтов и ученых не пересекаются, следовательно, заключение не верно.

Пример

Получить умозаключение.

Все поэты счастливы;
Некоторые поэты ленивы;
?

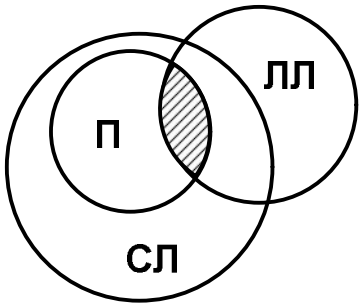


Рис. 3.4. Диаграмма для примера

В соответствии с посылками круг (рис. 3.4), изображающий поэтов (П), должен быть внутри круга, изображающего счастливых людей (СЛ), а пересечение поэтов и ленивых людей (ЛЛ) должно быть не пустым. Данное пересечение содержится в круге, изображающем поэтов и при этом пересечение ленивых и счастливых людей не пусто. Отсюда можно сделать умозаключение : «Некоторые ленивые люди счастливы».

Для проверки более сложных предикатных предложений в исчислении предикатов так же может быть использован принцип резолюций. Однако для его применения необходимо выполнить более сложную унификацию формул для их приведения к системе дизъюнктов. Это связано с наличием дополнительных элементов синтаксиса, в основном кванторов, переменных, предикатов и функций.

Алгоритм унификации включает этапы их преобразования сначала в префиксную, а затем сколемовскую нормальные формы [12]. В то же время предикатная формула может допускать множество нормальных форм и их интерпретаций, вид полученных результатов зависит от порядка применения правил, а также от способа переименования связанных переменных. Поэтому эффективное применение принципа резолюций для автоматизации логического вывода становится возможным лишь в частных случаях.

3.2. Представление знаний семантическими сетями

Под семантической сетью (СС) обычно подразумевают систему знаний некоторой предметной области, представленной в виде сети – ориентированного графа, узлы которого выражают понятия, события, а также свойства предметной области, а дуги являются описаниями их отношений. При этом все узлы и дуги могут быть снабжены метками, которые показывают, что именно они описывают [1, 3].

Формально семантическую сеть можно представить как $SN = <X, R>$, где X – множество объектов сети, а R – множество отношений между ними.

Понятия представляют собой сведения об абстрактных или физических объектах предметной области.

События представляют собой действия, происходящие в реальном мире, определяются указанием типа действия и ролей, которые играют объекты в этом действии.

Свойства используются для уточнения понятий и событий. Применительно к понятиям они описывают их особенности и характеристики (цвет, размер и др.), а применительно к событиям – продолжительность, время, место.

Дуги графа сети отображают многообразие семантических отношений, которые можно условно разделить на четыре класса: лингвистические, логические, теоретико-множественные и квантифицированные.

Лингвистические отношения выражают смысловую взаимосвязь между событиями, между событиями и понятиями или свойствами. При этом лингвистические отношения бывают:

- глагольные (время, вид, род, залог, наклонение);
- атрибутивные (цвет, размер, форма);
- падежными.

Логические отношения представляют собой операции, используемые в исчислении высказываний: дизъюнкция, конъюнкция, инверсия, импликация.

Теоретико-множественные представляют собой отношение подмножества, отношение части и целого, отношение множества и элемента. Типовыми примерами здесь являются отношения включения или совпадения (IS-A) и отношение «целое-часть» (PART-OF).

Квантифицированные отношения представляют собой кванторы общности и существования и используются для описания таких знаний, как «любой автомобиль требует ремонта», «существует водитель А, управляющий автомобилем В».

В общем случае систематизация отношений конкретной семантической сети зависит от специфики знаний предметной области и является сложной задачей.

На рис. 3.5. изображён пример описания структуры понятий с помощью семантической сети. Он иллюстрирует ситуацию взаимоотношения птицы и самолета.

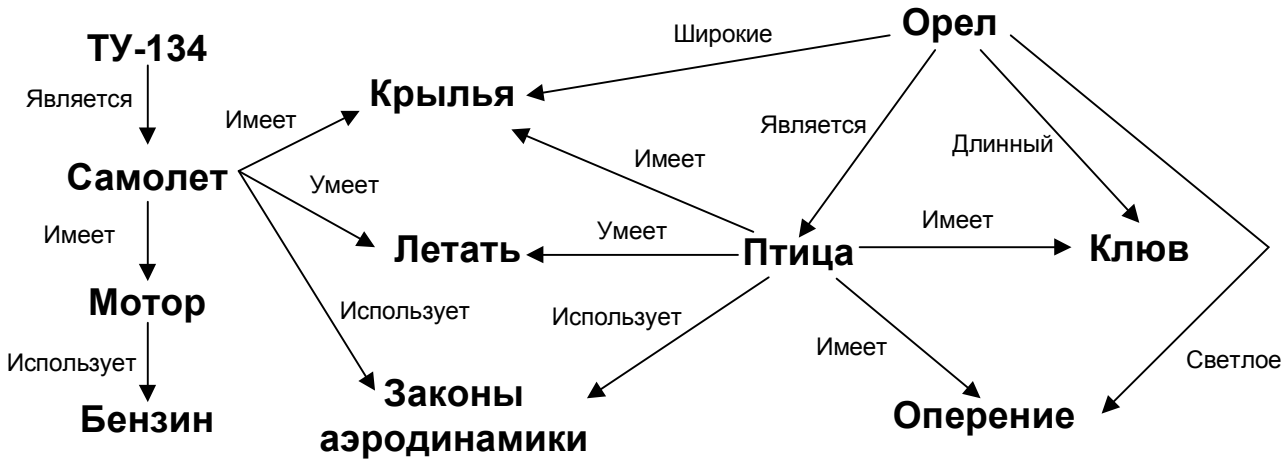


Рис. 3.5. Представление структуры понятий в виде семантической сети

При описании событий и действий с помощью семантической сети должны быть определены объекты, которые действуют, и объекты, над которыми эти действия выполняются. Связываются события и объекты с действиями с помощью падежных отношений (табл. 3.4)

Таблица 3.4

Основные падежные отношения

Квалификатор отношения	**Объекты, с которыми связывается действие**
Агент	Предмет, являющийся инициатором действия
Объект	Предмет, подвергающийся действию
Источник	Размещение предмета, перед действием
Приемник	Размещение предмета после действия
Время	Момент выполнения действия
Место	Место проведения действия
Цель	Действие другого события

На рис. 3.6 показан пример семантической сети, описывающей в виде событий и действий ситуацию, связанную с закупкой и доставкой товаров.

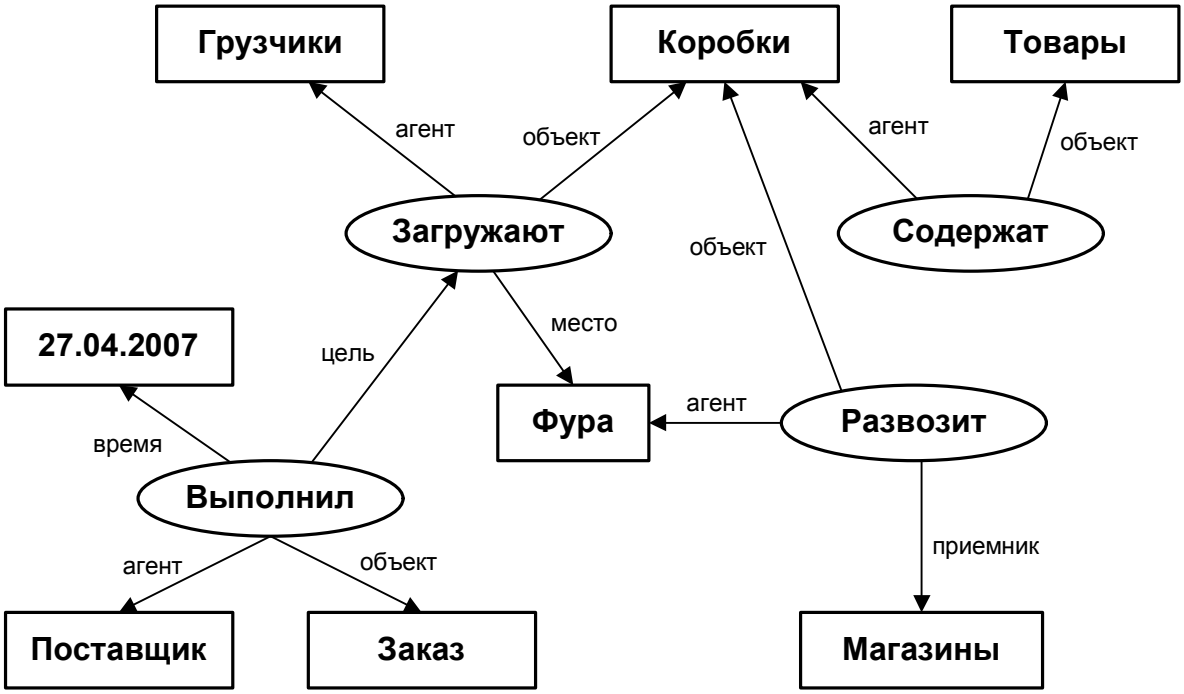


Рис. 3.6. Представление событий и действий в виде семантической сети

Существуют разные типы семантических сетей: дискретные, аналоговые, иерархические, гибридные [14].

Семантическая сеть называется **дискретной**, если в ней имеются только два возможных состояния связей между двумя вершинами – есть связь или нет.

Семантическая сеть называется **аналоговой**, если в ней состояния связей между вершинами *i* и *j* выражается величиной проходимости *Rij* , которая может изменяться в некотором диапазоне значений. Величина *Rij* выражает силу соответствующей связи между вершинами *i* и *j*, что позволяет отображать в сети различные оттенки сигнала.

Семантическая сеть называется **иерархической**, если в ней имеется в основном один тип связей – парадигматический, отражающий, например, родовые отношения.

Гибридные семантические сети сочетают свойства выше описанных типов семантических сетей. Так, нижний уровень такой сети может быть дискретного типа, а верхний – аналогового.

Процедура логического вывода обычно определяется через отношения между множеством дуг, имеющих общие узлы. Можно выделить два подхода к организации логического вывода на семантической сети: дедуктивный и семантический.

Дедуктивный вывод основан на логическом формализме, когда семантическая сеть представляется в виде графа и, по сути, реализует графовую версию исчисления предикатов. Здесь дуги соединяют имена предикатов с их аргументами, которым соответствуют определенные типы узлов сети.

Семантический вывод основан на извлечении из семантической сети новой информации путем поиска в ней, а затем интерпретации интересующих нас участков. Примером такого вывода может служить метод поиска пересечений.

Этот метод выделяет участок сети, связанный с входными понятиями *X1*, *X2*,…,*Xn*. Суть метода состоит в том, чтобы, начиная с вершин *X1*, *X2*,…,*Xn* и двигаясь по дугам, помеченным транзитивными отношениями, искать такие понятия (возможно, ближайшие в некотором смысле), которые находятся на пересечении построенных путей. Тогда множество найденных вершин и соединяющих их дуг образуют контекст, связывающий исходные понятия *X1*, *X2*,…,*Xn*.

Семантические сети обладают набором таких свойств, как хранение сведений об объектах системы и связях между ними, быстрый поиск объекта по заданным характеристикам, возможность обобщения и конкретизации знаний, которые обусловливают их широкое использование среди исследователей в качестве средства умозаключений. Кроме этого, семантические сети находят применение при создании экспертных систем и систем распознавания речи.

3.3. Фреймовая модель представления знаний

Эта модель основана на теории фреймов М. Минского, которая представляет собой систематизированную психологическую модель памяти человека и его сознания. Эта теория является достаточно абстрактной, поэтому описание на ее основе знаний возможно только в сочетании с другими языками, в частности объектно-ориентированного программирования [3, 12].

Во фреймовой системе за единицу представления принят объект, называемый фреймом. С точки зрения памяти фрейм – единица представления знаний, запомненных в прошлом, детали которой при необходимости могут быть изменены согласно текущей ситуации. Алгоритмически фрейм – это структура для описания понятия или ситуации, состоящая из характеристик этой ситуации и их значений. Фрейм имеет имя, служащее для идентификации описываемого им понятия, и содержит ряд слотов, с помощью которых описываются основные элементы этого понятия. Слот может содержать не только конкретное значение, но также имя процедуры, позволяющей вычислить это значение по заданному алгоритму. Так, слот с именем «возраст» может содержать имя процедуры, которая вычисляет возраст человека по дате рождения, записанной в другом слоте, и текущей дате. Процедуры, располагающиеся в слотах, называются присоединенными процедурами, их вызов происходит при обращении к слоту, в котором она помещена.

В общем случае структура данных фрейма может содержать следующие атрибуты [1].

Имя фрейма. Служит для идентификации фрейма в системе и должно быть уникальным.

Имя слота. Должно быть уникальным в пределах фрейма. Может быть или назначено проектировщиком, или выбрано из системных, зарезервированных имен. Системные слоты служат для управления выводом во фреймовой системе. Примером такого слота может быть слот-указатель дочерних фреймов (IS-A).

Указатели наследования. Показывают, какую информацию об атрибутах слотов из фрейма верхнего уровня наследуют слоты с аналогичными именами в данном фрейме.

Указатель типа данных. Показывает тип значения слота. Наиболее типичные типы: frame – указатель на фрейм; real – вещественное число; integer - целое число; list – список и др.

Значение слота. Оно должно соответствовать указанному типу данных и условию наследования.

Демоны. Демоном называется присоединенная процедура, автоматически запускаемая при выполнении некоторого условия, при обращении к некоторому слоту. Типы демонов связаны с условием запуска процедуры. Демон с условием IF-NEEDED (ЕСЛИ-НУЖНО) запускается, если в момент обращения к слоту его значение не установлено. Демон типа IF-ADDED (ЕСЛИ-ДОБАВЛЕНО) запускается при попытке изменения значения слота. Демон IF-REMOVED (ЕСЛИ-УДАЛЕНО)запускается при попытке удаления значения слота. Возможны и другие виды демонов.

Присоединенная процедура. Может быть значением слота и запускается по сообщению, переданному от другого фрейма. Демоны и присоединенные процедуры являются процедурными знаниями, объединенными вместе с декларативными в единую систему. Эти процедурные знания являются средствами управления выводом во фреймовых системах.

Совокупность данных предметной области может быть представлена множеством взаимосвязанных фреймов, образующих единую фреймовую систему, в которой объединяются декларативные и

процедурные знания. Такая система имеет, как правило иерархическую структуру, в которой фреймы соединены друг с другом с помощью родо-видовых связей. На верхнем уровне иерархии находится фрейм, содержащий наиболее общую информацию, истинную для всех остальных фреймов. Для уменьшения информационной избыточности во фреймовых системах реализуется принцип наследования информации, позволяющей общую (глобальную для системы) информацию хранить в отдельном родительском фрейме, а во всех остальных фреймах указывать ссылку на место хранения этой информации. В качестве примера рассмотрим фреймы, приведенные на рис. 3.7.

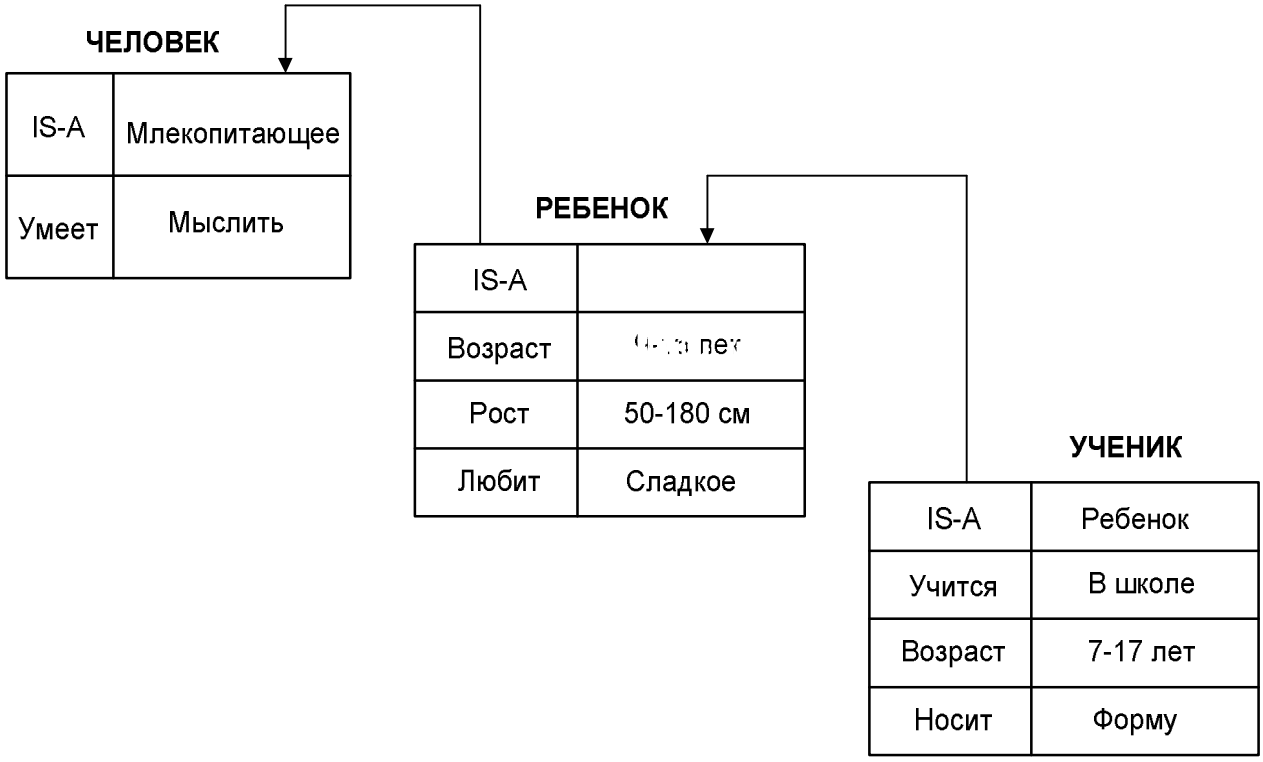


Рис. 3.7. Пример сети фреймов

Как следует из рис. 3.7 понятие «ученик», описываемое соответствующим фреймом, наследует свойства фреймов «ребенок» и «человек», которые находятся на более высоких уровнях иерархии. При возникновении вопроса «Любят ли ученики сладкое?», будет получен ответ «да», потому что этим свойством обладают все дети, что указано во фрейме «ребенок». Наследование может быть частным, например, слот «возраст» для учеников не наследуется из фрейма «ребенок», так как явно указан в собственном фрейме.

Над фреймами можно совершать некоторые теоретико-множественные операции, например объединение или пересечение. В первом случае в результирующем фрейме будут присутствовать все слоты, которые были в исходных. Во втором – результирующий фрейм будет содержать только одинаковые для всех фреймов слоты.

Фреймовые системы подразделяются на статические и динамические, последние допускают изменение фреймов в процессе решения задачи.

Можно выделить три основных способа вывода на фреймовой сети:

- объектно-ориентированное программирование, когда фрейм подменяется понятием объект, сочетающим в себе данные и подпрограммы для их обработки – методы.

- логико-объектные правила вывода объединяют предыдущую технологию с логическим выводом.

- специальные методы вывода, которые определяются непосредственно разработчиком.

3.4. Представление знаний правилами продукций

В продукционной модели описания знаний, предложенной Э. Постом, знания описываются в форме «ЕСЛИ - ТО». Условная часть правила «ЕСЛИ» называется посылкой или антецедентом, а часть «ТО» - заключением (продукцией) или консеквентом. Подобное представление знаний является наиболее простым и легко понимаемым, потому что близко к привычной нам форме рассуждений.

Пример

Если «животное имеет перья», ТО «животное - птица».

Условная часть правила может состоять из множества элементарных высказываний, соединенных логическими связками И, ИЛИ. В свою очередь, заключение выражает либо некоторый факт, либо указание на определенное действие, подлежащее исполнению, при истинности условной части.

Математически продукционная модель может быть описана как $<(i), Q, P, A \to B, N>$, где

i – имя продукции. В качестве имени может выступать понятие, отражающее суть данной продукции или порядковый номер, хранящийся в памяти системы;

Q – сфера применения продукции. Разделение на отдельные сферы позволяет экономить время на поиск нужных знаний.

$A \to B$ – ядро продукции.

P – условие применимости ядра продукции. Р представляет собой логическое выражение, истинность которого приводит к активизации продукции.

N – постусловие продукции, которое необходимо выполнить после реализации консеквента B.

Ядро продукции может быть как однозначным и детерминированным «ЕСЛИ A, ТО B», так и многозначным и недетерминированным «ЕСЛИ A, ТО чаще всего B1, реже B2». Во втором случае указываются альтернативные возможности выбора, которые оцениваются заданными статическими или динамическими приоритетами.

Интеллектуальные системы, где используется такой способ представления знаний, получили название продукционных. В состав таких систем входят база правил, содержащая совокупность знаний, описанных в продукционной форме, база данных или рабочая память и содержащая полученные в ходе работы заключения, а также интерпретатор правил, реализующий механизм вывода (рис. 3.8).

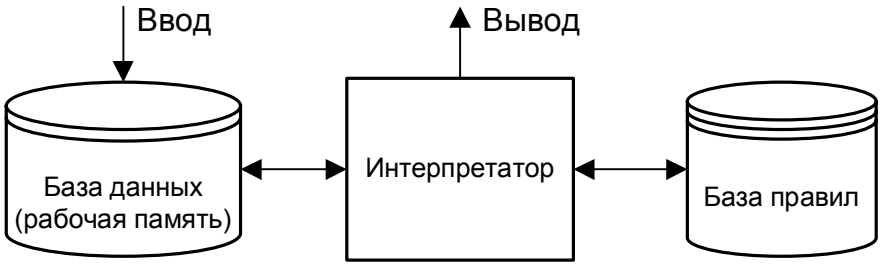


Рис. 3.8. Продукционная система

Механизм вывода состоит в замене одного фрагмента (A) базы правил другим фрагментом (B) и работает циклически до тех пор, пока не будут исчерпаны все правила или выполнено некоторое условие окончания. При этом может возникнуть конфликт выбора правил в

случае многозначных продукций. Для его разрешения используются различные стратегии управления выводом, например через задание приоритетности выбора.

Содержимое рабочей памяти изменяется в процессе решения задачи. Это происходит по мере срабатывания правил. Правило срабатывает, если при сопоставление фактов, содержащихся в рабочей памяти с заключением анализируемого правила, имеет место совпадение. При этом заключение сработавшего правила заносятся в рабочую память. В процессе логического вывода каждое правило из базы правил может сработать только один раз [1].

Пример

Пусть в базе правил имеются следующие правила.

Правило 1. «ЕСЛИ двигатель не заводится И фары не горят, ТО Сел аккумулятор».

Правило 2. «ЕСЛИ указатель бензина находится на нуле, ТО Двигатель не заводится».

Предположим, что в рабочую память от пользователя поступили следующие факты: «Фары не горят» и «Указатель бензина находится на нуле».

Процедура вывода в такой системе будет включать следующие шаги.

1. Сопоставление фактов из рабочей памяти с образцами правил из базы правил. Правило 1 не может сработать, а Правило 2 срабатывает, так как образец, совпадающий с его посылкой, присутствует в рабочей памяти.

2. В результате срабатывания Правила 2 в рабочую память заносится заключение этого правила: «Двигатель не заводится».

3. Второй цикл сопоставления фактов в рабочей памяти с имеющимися правилами. Теперь срабатывает Правило 1, так как конъюнкция условий в его антецеденте становится истинной.

4. Действие Правила 1, которое заключается в выдаче пользователю окончательного заключения – «Сел аккумулятор».

5. Конец работы (база правил исчерпана).

В продукционных системах существует два основных способа получения заключений – прямые и обратные выводы. Прямые

выводы реализуют стратегию «от фатов к заключениям». При обратных выводах выдвигаются гипотезы вероятных заключений, которые могут быть подтверждены или опровергнуты на основании фактов, поступающих в рабочую память.

Рассмотрим пример обратного порядка вывода – от заключений к фактам. Предположим, что в базе правил находятся два правила: Правило1 и Правило 2, а в рабочей памяти – те же факты, что и в предыдущем примере. Процедура вывода в этом случае будет включать следующие шаги.

1. Выдвигается гипотеза об окончательном заключении – сел аккумулятор.

2. Определяется правило, заключение которого соответствует выдвинутой гипотезе. Для нашего случая – Правило 1.

3. Проверяется возможность срабатывания Правила 1. Для этого в рабочей памяти должны присутствовать факты, совпадающие с образцом этого правила. В нашем случае Правило 1 не может сработать из-за отсутствия в рабочей памяти образца «Двигатель не заводится». Этот факт становится новой целью на следующем шаге вывода.

4. Поиск правила, заключение которого соответствует новой цели. Это Правило 2.

5. Проверяется возможность применения Правила 2. Оно срабатывает, так как в рабочей памяти присутствует факт, совпадающий с его образцом.

6. Выполняется Правило 2. В результате заключение «Двигатель не заводится» заносится в рабочую память.

7. Для условной части Правила 1 в рабочей памяти теперь есть все факты, поэтому подтверждается первоначально выдвинутая гипотеза.

8. Конец работы.

Таким образом, результатом работы является подтверждение гипотезы «Сел аккумулятор», при условии «Двигатель не заводится» и «Фары не горят».

Основные достоинства продукционных систем связаны с простотой представления знаний и организацией логического вывода, особенно при использования небольшого (десятки) числа правил.

Наиболее часто продукционные системы используются при создании экспертных систем диагностики для различных предметных областей.

Рассмотренные способы представления знаний предполагают, что в конкретной предметной области знания описываются в детерминированной форме и являются абсолютно достоверными. В то же время задача компьютеризации этих знаний полностью выполняется экспертами, мнения которых являются субъективными и могут не совпадать. В таких случаях при описании знаний могут появляться формулировки вида «примерно», «немного», «почти» и т.п., формализация которых в принципе не может быть выполнена традиционными способами. Поэтому в последнее время большое распространение получили методы описания таких нечетких знаний с помощью математического направления, известного как теория нечетких множеств.

3.5. Представление нечетких знаний

Одним из ключевых проявлений человеческого мышления является способность обрабатывать нечеткую информацию, характеризующуюся неполнотой и неопределенностью, свойственным многим классам реальных проблем. Поэтому важным является создание эффективных методов для отображения таких нечеткостей реального мира и моделирования приближенных рассуждений человека в компьютерных системах. Значительный вклад в эту область был сделан профессором Калифорнийского университета (Беркли) Л. Заде, создавшим математический аппарат теории нечетких множеств. С его помощью могут быть эффективно описаны нечеткие понятия и знания, а также выполняться операции над этими знаниями, нечеткие выводы. Широкое практическое применение теория нечетких множеств получила в таких задачах как автоматическое управления, принятие решений, представление и обработка знаний и др. Далее кратко рассматриваются основы этой теории [1, 14].

Основные понятия нечетких множеств. В основе теории нечетких множеств лежит расширение классического понятия множества за счет возможности принимать функции принадлежности элемента множеству любых значений в интервале [0, 1], а не только фиксированных значений 0 либо 1.

Пусть U полное множество объектов некоторого класса. Нечеткое подмножество F множества U, именуемое далее нечетким множеством определяется через функцию принадлежности $\mu_F(u), u \in U$. Эта функция отображает элементы u множества U на множество вещественных чисел отрезка [0, 1], которые указывают степень принадлежности каждого элемента нечеткому множеству F.

Если полное множество U состоит из конечного числа множеств $u_1, u_2, ..., u_n$, то нечеткое множество F можно представить в следующем виде:

$$F = \mu_F(u_1)/u_1 + \mu_F(u_2)/u_2 + ... + \mu_F(u_n)/u_n = \sum_{i=1}^{n} \mu_F(u_i)/u_i .$$

Важно, что в этом выражении знак «+» означает не сложение, а скорее объединение, символ / показывает, что значение $\mu_F(u_i)/u_i$ относится к элементу u_i, а не означает деление на него.

В случае непрерывного множества U можно F записать как интеграл $F = \int_u \mu_F(u)/u$.

В качестве примера рассмотрим U – множество людей в возрасте от 0 до 100 лет, и пусть понятия «молодой», «среднего возраста» и «старый» представлены нечеткими множествами *F1*, *F2* и *F3* соответственно. Эти нечеткие множества являются подмножествами множества *U*. Функции принадлежности элементов множества *U* к понятиям, представленным нечеткими множествами *F1*, *F2*, *F3* показаны на рис. 3.9.

При этом форма задания значений нечетких множеств имеет следующий вид:

$$F1 = \mu_{F1}(u) = 1/0 + 1/10 + 0{,}8/20 + 0{,}3/30;$$
$$F2 = \mu_{F2}(u) = 0{,}5/30 + 1/40 + 0{,}5/50;$$
$$F3 = \mu_{F3}(u) = 0{,}4/50 + 0{,}8/60 + 1/70 + 1/80 + 1/90.$$

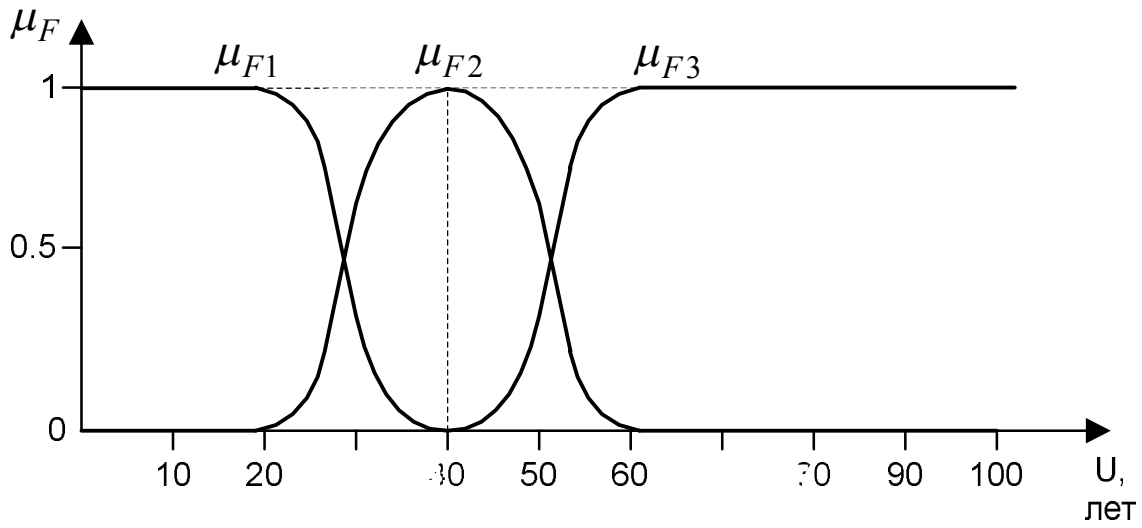


Рис. 3.9. Функции принадлежности нечетких множеств примера

Как следует из рис. 3.9 принадлежность к нечеткому множеству *F1*, соответствующему понятию «молодой», составляет 1 для детей, 0,8 – для двадцатилетнего человека, а степень принадлежности тридцатилетнего равна 0,3. При записи функции принадлежности элементы нечеткого множества со значениями $\mu_F(u_i)/u_i = 0$ не включаются. Аналогично описаны функции принадлежности для двух других нечетких множеств.

Отметим, что обычно функции принадлежности для нечетких множеств строятся субъективно по результатам опросов экспертов.

Операции над нечеткими множествами. Подобно классическим над нечеткими множествами можно выполнять теоретико-множественные операции. Далее рассматриваются основные из них: дополнение, объединение и пересечение.

1. Операция дополнения

$$\overline{F} = \sum_{i=1}^{n} (1 - \mu_F(u_i))/u_i, \;\; \mu_{\overline{F}}(u) = 1 - \mu_F(u).$$

Следуя примеру, дополнением нечеткому множеству «молодой» будет соответствовать нечеткое множество понятия «немолодой», функция принадлежности которого показана на рис. 3.10, а математическая запись имеет вид:

$$\overline{молодой} = \mu_{\overline{F1}} = 0.2/20 + 0.7/30 + 1/40 + 1/50 + 1/60 + 1/70 + 1/80 + 1/90.$$

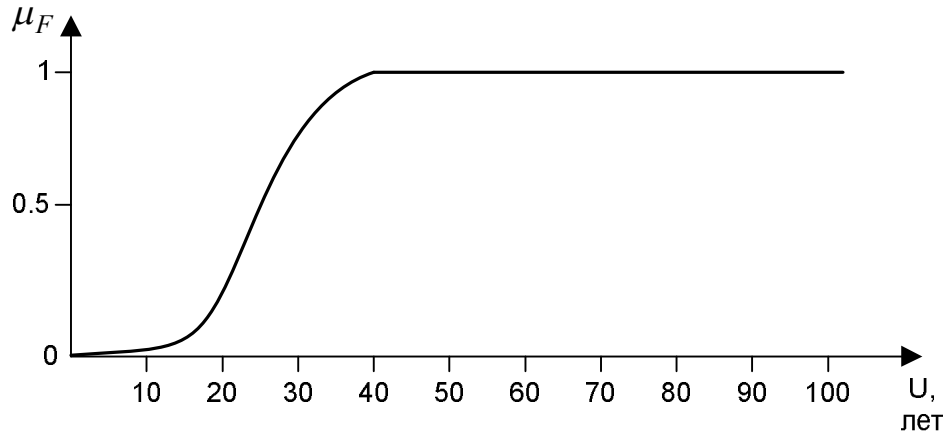


Рис. 3.10. Функция принадлежности дополнения нечеткого множества *F1*

2. Операция объединения

$$F \cup G = \sum_{i=1}^{n} (\mu_F(u_i) \vee \mu_G(u_i))/u_i, \mu_{F \cup G}(u) = \mu_F(u) \vee \mu_G(u).$$

Здесь операция $\vee$ соответствует взятию максимума. Для определения понятия, которому будет соответствовать объединение нечетких множеств *F1* и *F2*, вычислим функцию принадлежности:

$$(\textit{молодой} \cup \textit{средний}) = \mu_{F1 \cup F2}(u) = 1/0 + 1/10 + 0{,}8/20 + 0{,}5/30 + 1/40 + 0{,}5/50.$$

Ниже на рис. 3.11 приводится график этой функции, а ее лингвистической интерпретацией является понятие «человек молодого или среднего возраста».

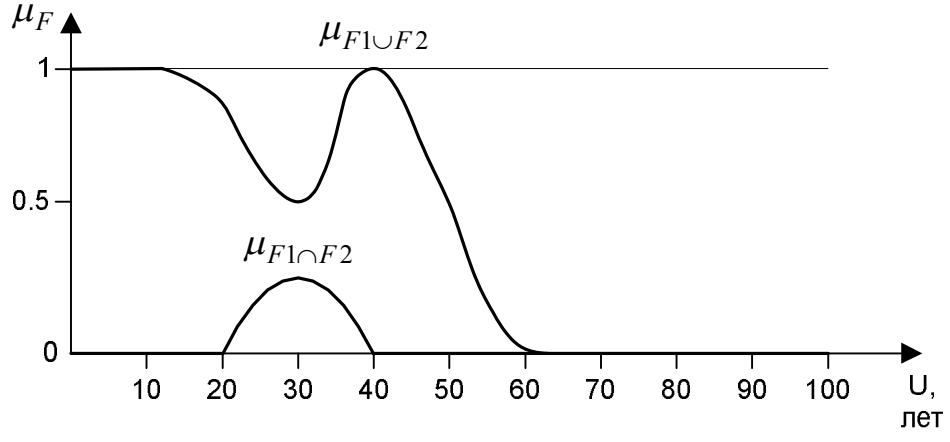


Рис. 3.11. Функция принадлежности объединения нечетких множеств *F1* и *F2*

3. Операция пересечения

$$F \cap G = \sum_{i=1}^{n} (\mu_F(u_i) \wedge \mu_G(u_i))/u_i, \mu_{F \cap G}(u) = \mu_F(u) \wedge \mu_G(u).$$

Здесь операция $\wedge$ соответствует взятию минимума. Для определения понятия, которому будет соответствовать пересечение множеств *F1* и *F2,* вычислим функцию принадлежности:

$$(молодой \cap средний) = \mu_{F1 \cap F2}(u) = 0{,}3/30$$

Остальные члены этой функции, соответствующие значениями аргумента, кратным 10, равны нулю. На рис. 3.10 приводится график этой функции, а ее возможными лингвистическими интерпретациями являются понятия «уже не молодой, но еще не средний возраст», «одновременное молодой и средний возраст».

Нечеткие отношения. Часто понятия предметной области связаны различными отношениями. Для организации нечетких выводов необходимо формализовать понятие нечеткого отношения.

Нечетким отношением R между полными множествами U и V называется нечеткое подмножество прямого декартова произведения $U \times V$, определяемое следующим образом:

$$R = \sum_{i=1}^{l} \sum_{j=1}^{m} \mu_R(u_i, v_j)/(u_i, v_j),$$

где $U = \{u_1, u_2, ..., u_n\}$, $V = \{v_1, v_2, ..., v_m\}$

Предположим, что между понятиями, представленными нечеткими множествами $F \subseteq U$ и $G \subseteq V$, существует отношение, заданное правилом «ЕСЛИ F, ТО G». Тогда один из способ построения такого нечеткого отношения R между F и G состоит в следующем:

$$R = F \times G = \sum_{i=1}^{l} \sum_{j=1}^{m} \mu_F(u_i) \wedge \mu_G(v_j)/(u_i, v_j), \quad \mu_R(u,v) = \mu_F(u) \wedge \mu_G(v). \quad (3.1)$$

Пусть U и V множества натуральных чисел от 1 до 4. Определим понятия «малые числа» и «большие числа» с помощью нечетких множеств F и G соответственно.

$U = V = \{1, 2, 3, 4\}$;

$F = 1/1 + 0.6/2 + 0.1/3$;

$G = 0.1/2 + 0.6/3 + 1/4$.

Пусть задано правило: «ЕСЛИ u – малое число, ТО v - большое» или $F \rightarrow G$. Построим для него соответствующее нечеткое отношение $R = F \times G$.

$$\mathrm{R} = \begin{array}{c} \\ v_j \end{array} \begin{array}{|cccc|} \multicolumn{4}{c}{u_i} \\ \hline 0 & 0.1 & 0.6 & 1 \\ 0 & 0.1 & 0.6 & 0.6 \\ 0 & 0.1 & 0.1 & 0.1 \\ 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Заметим, что наряду с рассмотренным есть и другие способы задания нечетких отношений.

Композиция нечетких отношений. Необходима при наличии цепочки правил, образующих знания в виде нечетких множеств, отношений. Выполнена она может быть с помощью операций свертки, которые включают максиминную, минимаксную, максимультипликативную и др. Рассмотрим применение наиболее распространенной - первой из них.

Пусть R – нечеткое отношение из области U в область V, а S – нечеткое отношение из области V в область множества W. Тогда нечеткое отношение из U в область W определяется как максиминная свертка

$$R \circ S = \sum_{i=1}^{l} \sum_{k=1}^{n} \bigvee_{v_j \in V} (\mu_R(u_i, v_j) \wedge \mu_S(v_j, w_k))/(u_i, w_k), \qquad (3.2)$$

где $\vee$ - знак взятия максимума для всех v_j;

$\wedge$ - знак взятия минимума.

Пусть для $W = \{1, 2, 3, 4\}$ определены следующим образом нечеткие множества:

$\overline{F}(\subset V)$ = «не малые числа» = 0/1+0.4/2+0.9/3+1/4;

$H(\subset W)$ = «очень большие числа» = 0/1+0/2+0.5/3+1/4.

Тогда, если есть правило «ЕСЛИ v не малое число, ТО w очень большое число», нечеткое отношение S из V в W согласно (3.1) определяется как

$$\mathrm{S} = \begin{array}{c} \\ v_j \end{array} \begin{array}{|cccc|} \multicolumn{4}{c}{w_k} \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0.4 \\ 0 & 0 & 0.5 & 0.9 \\ 0 & 0 & 0.5 & 1 \\ \hline \end{array}$$

Если далее вычислить по формуле (3.2) свертку с нечетким отношением R, то из двух значений «ЕСЛИ u – малое число,

ТО *v* - большое», «ЕСЛИ *v* не маленькое число, ТО *w* очень большое» получим следующее нечеткое отношение из *U* в *W*:

$$R \circ S = \underset{v \in V}{\vee} \{\mu_R(u,v) \wedge \mu_S(v,w)\} =$$

$$= \begin{bmatrix} 0 & 0{,}1 & 0{,}6 & 1 \\ 0 & 0{,}1 & 0{,}6 & 0{,}6 \\ 0 & 0{,}1 & 0{,}1 & 0{,}1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}4 & 0{,}4 \\ 0 & 0 & 0{,}5 & 0{,}9 \\ 0 & 0 & 0{,}5 & 1 \end{bmatrix} =$$

u_i \ w_k			
0	0	0,5	1
0	0	0,5	0,6
0	0	0,1	0,1
0	0	0	0

Полученное нечеткое отношение показывает взаимосвязь областей U и W. При парных сравнениях элементов *i*-й строки и *j*-го столбца из них выбирается наименьший. Затем из четырех минимальных элементов выбирается максимум, который является результатом и записывается в ячейку с координатами *i*, *j*.

Нечеткие выводы. Рассмотрим традиционный дедуктивный метод логического вывода Modus Ponendo Ponens для нечетких множеств. Его интерпретация «Если *F* – истина И импликация $F \rightarrow G$ тоже истина» то *G* – истина» в этом случае остается неизменной. Однако в среде нечетких множеств этот вывод записывается с некоторыми изменениями: из факта *F'* и правила $F \rightarrow G$ можно вывести *G'*. Здесь *F'*, *G'* образуют нечеткие множества рассматриваемого правила вывода на полных множествах *F*, *G* соответственно.

Для организации нечеткого отношения из правила $F \rightarrow G$ может быть использована формула (3.1), а при наличии цепочки из нескольких правил вывода – максиминная свертка (3.2). В свою очередь вывод *G'* определяется из максиминной свертки нечеткого множества *F'* и отношения *R*:

$$G' = F' \circ R = \sum_{i=1}^{m} \underset{u_i \in U}{\vee} (\mu_{F'}(u_i) \wedge \mu_R(u_i, v_i))/v_i \,, \qquad (3.3)$$

где $F, F' \subset U;\ G, G' \subset V$.

Пусть, продолжая предыдущий пример, U = V = {1, 2, 3, 4}, $F(\subset U)$ = «малые числа», $G(\subset V)$ = «большие числа». Кроме того, пусть в качестве исходной посылки для вывода задан факт «*u* – число около 2», представленное нечетким множеством *F'* = «около 2» = 0,3/1+1/2+0,3/3+0/4.

Пусть также задано правило $F \rightarrow G$: «ЕСЛИ u - малые числа, то v - большие», формализованное в виде ранее описанного отношения R. Используя комбинационное правило вывода (3.3), определим вывод G', соответствующий ответу на вопрос «Что представляет собой v, если u – число около 2?» и если области U и V связаны отношением R.

$$G' = F' \circ R' = [0{,}3 \quad 1 \quad 0{,}3 \quad 0] \circ \begin{bmatrix} 0 & 0{,}1 & 0{,}6 & 1 \\ 0 & 0{,}1 & 0{,}6 & 0{,}6 \\ 0 & 0{,}1 & 0{,}1 & 0{,}1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0{,}1 & 0{,}6 & 0{,}6 \end{bmatrix}$$

Здесь в результирующем векторе каждый элемент j представляет значение принадлежности v_j множества G'. На рис. 3.12 показан график функции принадлежности результата нечеткого вывода.

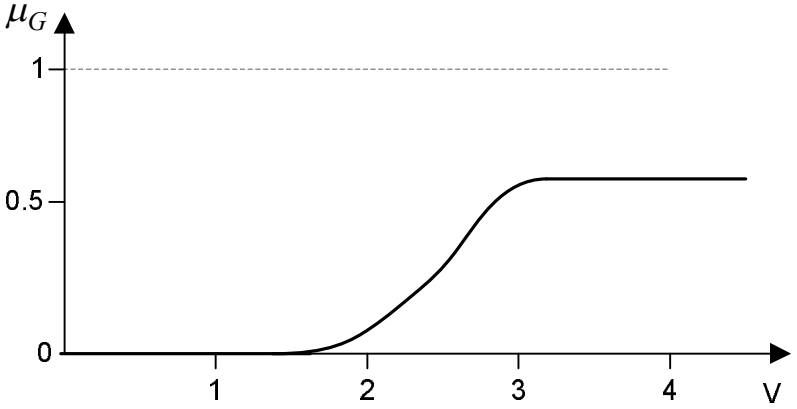


Рис. 3.12. Функция принадлежности результата нечеткого вывода

Исходя из графика можно предложить следующую лингвистическую интерпретацию: «v – не очень большое число» или «v – до некоторой степени большое число».

При описании объектов и явлений с помощью нечетких множеств используется понятие нечеткой переменной, которая может принимать нечеткие значения.

Нечеткая переменная определяется тройкой $<\alpha, U, F>$,

где α - наименование переменной;

U – универсальное множество, область определения α;

F – нечеткое множество на U, описывающее ограничения на возможные нечеткой переменной α.

Обычно нечеткие переменные используются для моделирования различных систем, и, следовательно, над этими переменными приходится выполнять большие объемы разных операций. Для удобства реализации этого, а также организации ввода-вывода и хранения информации рекомендуется использовать функции принадлежности стандартного вида, с которыми проще выполнять расчеты. В частности, такими функциями являются трапециевидные (рис. 3.13), где $\mu_F(u)$ характеризуется $<m, n, l, k>$. Как частный случай при $m = n$ получается треугольная форма функции принадлежности.

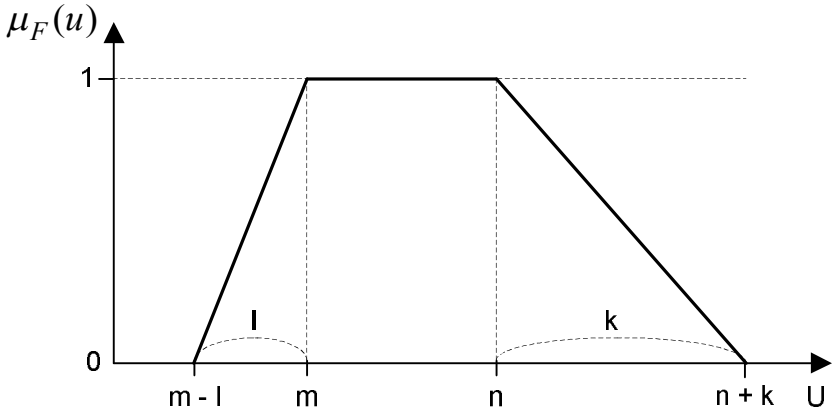


Рис. 3.13. Трапециевидная функция принадлежности

В качестве примера рассмотрим ситуацию, когда при составлении проекта бюджета рассматриваются различные источники финансирования. Причем некоторые из них характеризуются неточностью оценки денежных сумм на день оценивания, а другие малой надежностью. Кроме того, из бюджета необходимо отдать долги, количество которых также неточно, так как зависит от того, потребует ли кредитор все или только часть в следующем финансовом периоде. Эксперты высказали следующие предположения об источниках финансирования и долге.

Источник А: финансирование обеспечивается, его сумма может изменяться от 40 до 100 млн в зависимости от конъюнктуры, но с наибольшей вероятностью можно ожидать поступления в сумме от 50 до 70 млн.

Источник В: источник надежен и разумно полагать, что финансирование будет предоставлено и составит сумму 100 - 110 млн.

Источник C: источник ненадежен, а если и даст, то не более 20 млн.

Долг D: плата за кредиты 50 - 100 млн., но наиболее вероятна выплата 80 млн.

В результате имеется три источника поступлений и один источник расхода. Построим на основе их описаний трапециевидные функции принадлежности для каждой из четырех нечетких переменных (рис. 3.14). После задания всех нечетких переменных возникает задача определения суммы всего бюджета, которая также будет нечеткой величиной. А для этого требуется выполнить арифметические операции над нечеткими переменными.

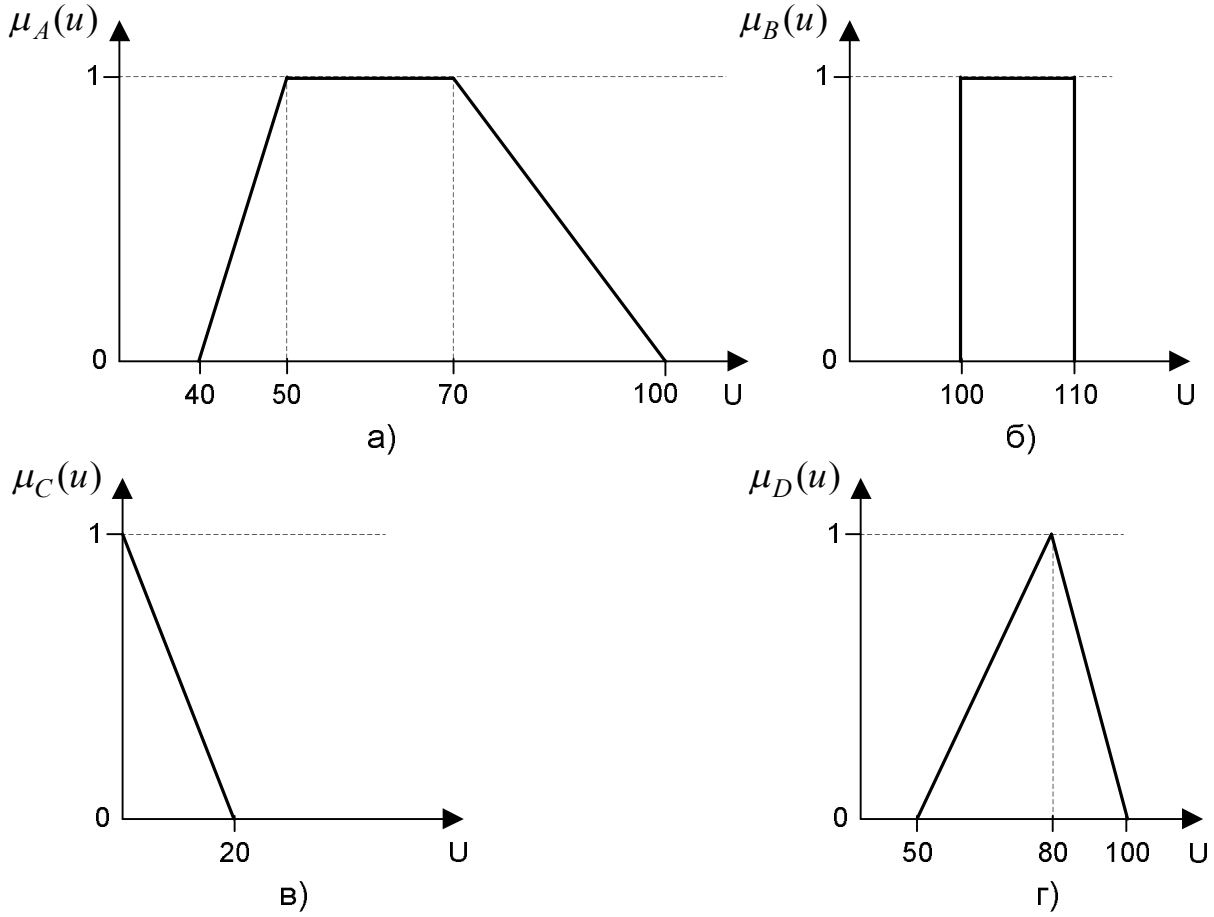


Рис. 3.14. Нечеткое описание проекта бюджета:
а – A = (50, 70 ,10 ,30); **б** – B = (100, 110, 0, 0);
в – C = (0, 0, 0, 20); **г** – D = (80, 80, 30 ,20)

Определение этих операций рассмотрим на примере двух нечетких переменных F1, F2, которые заданы своими трапециевидными функциями принадлежности: $\mu_{F1}(u)$ = <*m1*, *n1*, *l1*, *k1*>,

$\mu_{F2}(u)$ = <*m2*, *n2*, *l2*, *k2*>. Результатом операций будет нечеткая переменная F , параметры функции принадлежности которой определяются в соответствии с табл. 3.4.

Таблица 3.4

Основные арифметические операции над нечеткими переменными с трапециевидной функцией принадлежности

Тип операции	**Зависимости для вычисления параметров функции принадлежности**
$F = F1 + F2$	$m = m1 + m2;\ n = n1 + n2;$ $l = l1 + l2;\ k = k1 + k2$
$F = F1 - F2$	$m = m1 - n2;\ n = n1 - m2;$ $l = l1 + k2;\ k = k1 + l2$
$F = F1 * F2$	$m = m1*m2;\ n = n1*n2;$ $l = m1*m2 - (m1 - l1)*(m2 - l2);$ $k = (n1 + k1)*(n2+k2) - n1*n2$
$F = F1 / F2$	$m = m1/n2;\ n = n1/m2;$ $l = (m1*k2 + n2*l1) / (n2*n2+n2*k2);$ $k = (m2*k1 + n1*l2) / (m2*m2 - m2*l2)$

На основе приведенных описаний арифметических операций можно для рассматриваемого примера определить оценку бюджета с учетом долгов T как сумму трех источников финансирования минус предполагаемые платы по кредиту. Причем результат будет также нечеткой переменной T = A + B + C - D = (50+100+0 - 80, 70+110+0 - 80, 10+0+0 + 20, 30+0+20 + 30) = (70, 100, 30, 80). Ее функция принадлежности приведена рис. 3.15.

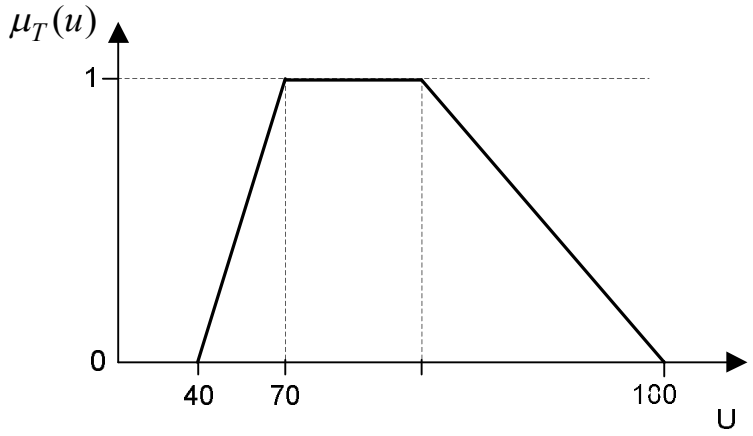


Рис. 3.15. Результирующее нечеткое описание бюджета

Таким образом, в бюджете может быть сумма от 40 до 180млн, но с наибольшей степенью уверенности можно говорить о суммах от 70 до 100млн.

Наряду с нечеткой переменной существует понятие **лингвистической переменной**. С ее помощью можно эффективно представлять вербальные описания некоторой количественной величины. Так, с их помощью можно описать качественную информацию об объекте принятия решений, представленную в словесной форме экспертами.

3.6. Систематизация знаний на основе онтологий

В настоящее время особую актуальность получает идея компьютеризации знаний о целых предметных областях для повышения качества работы с программами за счет наделения их интеллектуальными свойствами, а также создания условия для взаимодействия автономных электронных агентов, выполняющих например, поиск информации в компьютерных сетях.

Центральной компонентой таких систем стали онтологии, создание которых представляет собой методологию кодирования и представления декларативных знаний [3, 9].

Изначально термин «онтология» имел исключительно философскую основу, согласно которой онтология (филос.) – философское учение о бытии, его основах, принципах, структуре и закономерностях. И лишь в последнее время он был переосмыслен с позиций информатики. **Онтология** (информ.) – набор определений (на формальном языке) фрагмента декларативных знаний, ориентированный на совместное многократное использование различными пользователями в своих приложениях. Из этого определения следует, что онтология включает в себя комплекс понятий от самых общих до наиболее конкретных, охватывающий полный спектр объектов и отношений, включая события и процессы, а также значения (атрибутов и отношений), определяемые, если необходимо, во времени и пространстве.

Эта система объектов связывается как универсальными зависимостями типа «общее-частное», «часть-целое», «причина-следствие» и т.п., так и специфическими для соответствующей предметной области. Другими словами, онтология – модель предметной области, использующая все допустимые средства представления знаний, релевантные для этой предметной области.

Можно выделить следующие причины, когда возникает потребность в создании онтологий:

- совместное использование людьми или программными агентами общего понимания структуры информации;
- повторное использование знаний в предметной области;
- анализ знаний в предметной области;
- для того чтобы сделать допущения в предметной области явным.

Совместное использование людьми или программными агентами общего понимания структуры информации. Является одной из наиболее общих целей разработки онтологий. Так, пусть несколько различных Интернет-сайтов содержат информацию по медицине или предоставляют информацию о платных медицинских услугах, оплачиваемых через Интернет. Если эти Интернет-сайты совместно используют и публикуют одну и ту же базовую онтологию терминов, которыми они все пользуются, то компьютерные агенты могут извлекать информацию из этих различных сайтов и накапливать ее. Впоследствии агенты могут использовать накопленную информацию для ответов на запросы пользователей или как входные данные для других приложений.

Обеспечение возможности использования знаний предметной области. Подобное свойство онтологий позволяет формировать единую понятийную базу предметной области, которую в дальнейшем могут использовать как специалисты, так и информационные системы из других областей, а также интегрировать ее в другие прикладные области для их расширения.

Создание явных допущений в предметной области. Дает возможность легко изменить эти допущения при изменении знаний о предметной области. Жесткое кодирование предположений о мире на языке программирования приводит к тому, что эти предположения не только сложно найти и понять, но и также сложно изменить, особенно не программисту. Кроме того, явные спецификации знаний в предметной области полезны для новых пользователей, которые должны узнать значения терминов предметной области.

Анализ знаний в предметной области. Возможен, когда имеется декларативная спецификация терминов. Формальный анализ таких терминов ценен при попытке выявления соответствий между внешней информацией и понятиями предметной области, например, при компьютерном тестировании.

Онтологии можно применять в качестве строительных блоков компонентов баз знаний, схемы объектов в объектно-ориентированных системах, концептуальной схемы баз данных, структурированного глоссария взаимодействующих сообществ, словаря для связи между агентами, определения классов для программных систем.

Важно отметить, что, как правило, создание онтологии предметной области самой по себе не является главной целью. Разработка онтологии напоминает процесс определения набора данных и их структуры для использования другими программами.

Онтологии включают описание как очень общих, так и специфических для конкретной предметной области терминов и могут быть формализовано записаны как $O = <X,\ R,\ F>$,

где X – конечное, непустое множество понятий (терминов) предметной области;

R – конечное множество отношений между понятиями X;

F – конечное множество функций интерпретации, заданных на понятиях X и (или) отношениях R.

Для такой формализации онтологии возможны следующие частные случаи.

1. $R = Ø$, $F = Ø$. В этом случае онтология представляет собой простой словарь и ее практическое использование целесообразно в том случае, если термины принадлежат очень узкой предметной области и их смыслы уже заранее согласованы в пределах некоторого сообщества. Примером такой ситуации могут служить индексы поисковых машин сети Интернет.

2. $R = Ø$, $F \neq Ø$. Здесь каждому элементу множества X может быть поставлена соответствующая функция интерпретации. При этом часть таких интерпретирующих терминов задается процедурно, а не декларативно, то есть смыл терминов предметной области может динамически меняться в зависимости от перехода между прикладными областями.

В общем случае ($X \neq Ø$, $R \neq Ø$, $F \neq Ø$) основной акцент в онтологии делается на задание семантических отношений, имеющихся в предметной области вместе с созданием единой иерархии понятий, унификации терминов и правил их интерпретации.

В настоящее время не существует единственного «правильного» способа разработки онтологии. Чаще всего используется итерационный подход, когда, начиная с некоторого чернового варианта, онтология постепенно уточняется с добавлением к ней новых деталей. В общем случае процесс создания онтологии включает рассмотрение следующих вопросов.

1. Обозначение целей и области применения создаваемой онтологии. Для этого необходимо определить, для чего создается онтология и как она будет в дальнейшем использоваться.

2. Построение онтологии

Фиксирование знаний о предметной области, которое включает:

- определение основных понятий и их взаимоотношений в выбранной предметной области;

- создание точных непротиворечивых определений для каждого основного понятия и отношения;

- определение терминов, которые связаны с этими терминами и отношениями;

- окончательное согласование всех названных этапов.

Кодирование, которое подразумевает:

- разделение совокупности основных терминов, используемых в онтологии, на отдельные классы понятий;

- выбор или разработку специального языка для представления онтологии;

- непосредственно задание фиксированной концептуализации на выбранном языке представления знаний.

Как методология инженерии знаний информация в онтологии, как правило, представляется в виде семантической или фреймовой сети. При таком описании узлами сети являются понятия предметной области, а дуги определяют их отношения. Для конкретизации таких понятий они организовываются в иерархию классов, правила проектирования которой близки к технологии объектно-ориентированного программирования. Отличие состоит в том, что объектно-ориентированное программирование сосредоточивается главным образом на операторных свойствах класса, тогда как разработчик онтологии принимает решения, основываясь на структурных свойствах класса.

Для описания классов онтологии используются фреймы, представляющие собой именованную структуру данных, состоящую из слотов, с помощью которых описываются свойства понятий для их конкретизации.

Процесс создания онтологии как способ моделирования предметной области является очень трудоемким и предполагает в зависимости от субъективного мнения эксперта множество альтернатив ее вариантов. В связи с этим важным является автоматизация проектирования и систематизация уже созданных онтологий в электронном виде с помощью существующего программного обеспечения.

Сейчас можно выделить несколько программных сред разработки онтологий, наиболее популярной из которых является «Protégé» (http://protege.stanford.edu). Кроме этого, для целого ряда областей уже существует опыт разработки онтологий, которые могут использоваться экспертами по предметным областям для совместного использования и аннотирования информации в своей области. Так, онтологии в области медицины доступны в электронном виде и могут быть интегрированы в используемые интеллектуальные системы.

В качестве примера рассмотрим фрагмент процедуры создания онтологии для предметной области «Искусственный интеллект». Процесс проектирования онтологии будет включать:

- определение классов в предметной области;
- расположение классов в таксономическую иерархию;
- определение слотов и описание допустимых значений этих слотов;
- заполнение слотов значениями.

Существует несколько подходов к разработке иерархии классов, наиболее распространенный из которых – нисходящее проектирование, оно начинается с определения самых общих понятий предметной области с последующей их конкретизацией.

В соответствии с этим для заданной предметной области были определены следующие классы для наиболее общих понятий: «Технологии искусственного интеллекта», «Интеллектуальные системы», «Программное обеспечение», «Области применения», «Источники информации» (рис. 3.16).

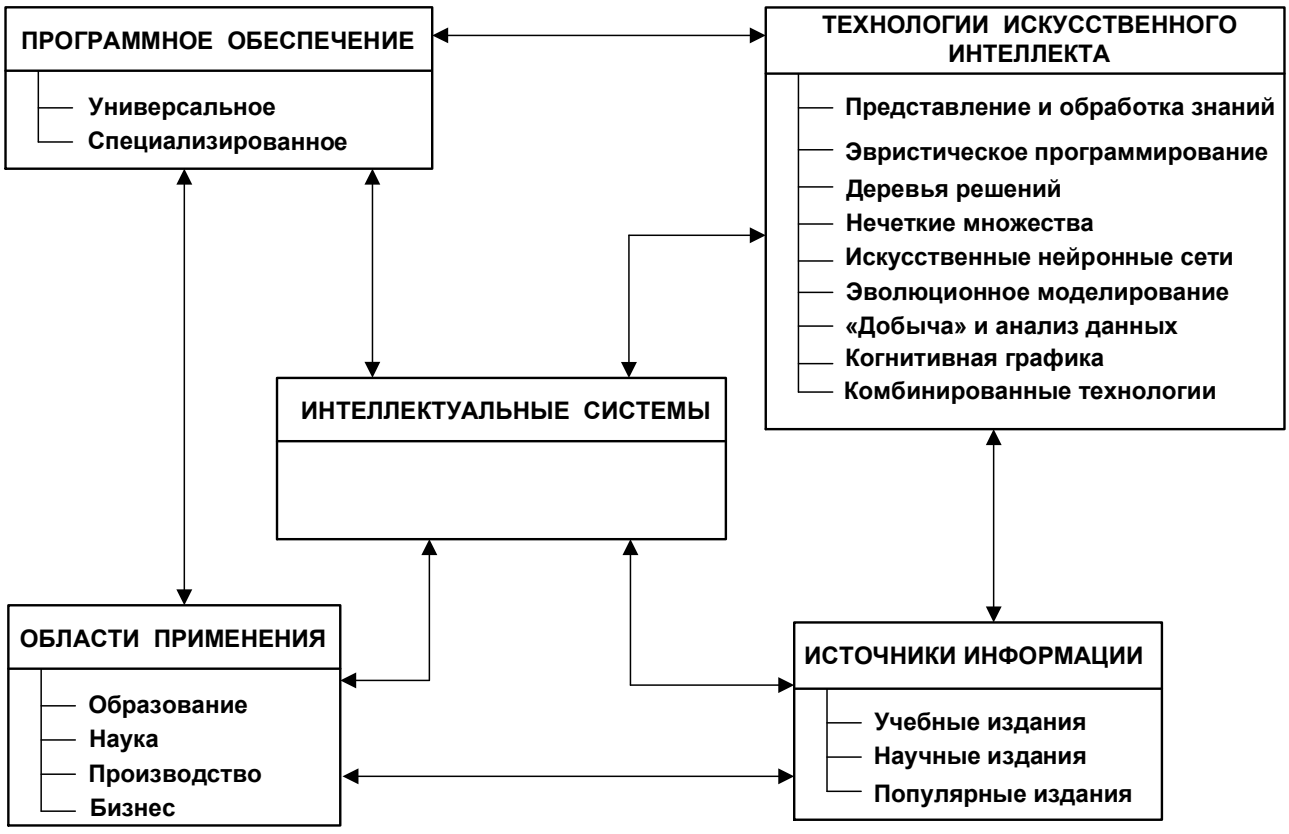


Рис. 3.16. Таксономия классов понятий предметной области

Далее каждый из классов был уточнен множеством подклассов, например «Интеллектуальные системы» до «Экспертные системы», «Системы анализа естественного языка», «Системы обработки естественного языка», «Системы анализа изображений» и др. На рис. 3.17 показаны различные уровни таксономии понятия «Интеллектуальные системы».

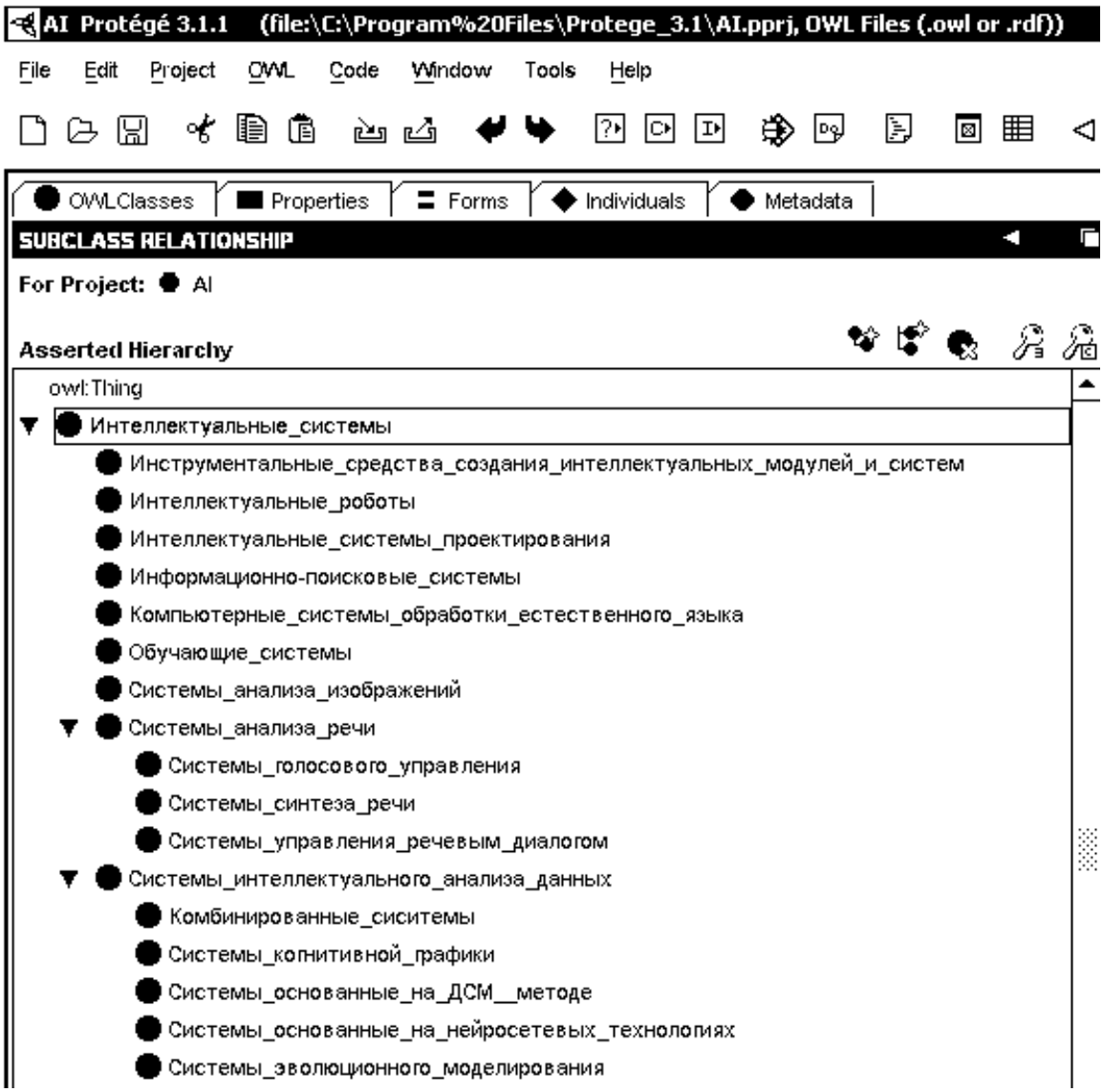


Рис. 3.17. Таксономия одного класса понятий предметной области, построенная в системе «Protégé»

Для уменьшения информационной избыточности онтологии реализуется принцип наследования информации, позволяющей общую (глобальную для онтологии) информацию хранить в отдельном родительском фрейме, а во всех остальных фреймах указывать ссылку на место хранения этой информации.

Для дальнейшего уточнения понятий должна быть описана внутренняя структура классов через заполнение соответствующих слотов. Для этих целей был определен список терминов, связанных с каждым из понятий и образующих тезаурус или словарь предметной области. На рис. 3.18 показаны слоты класса «Система нейросетевого моделирования»

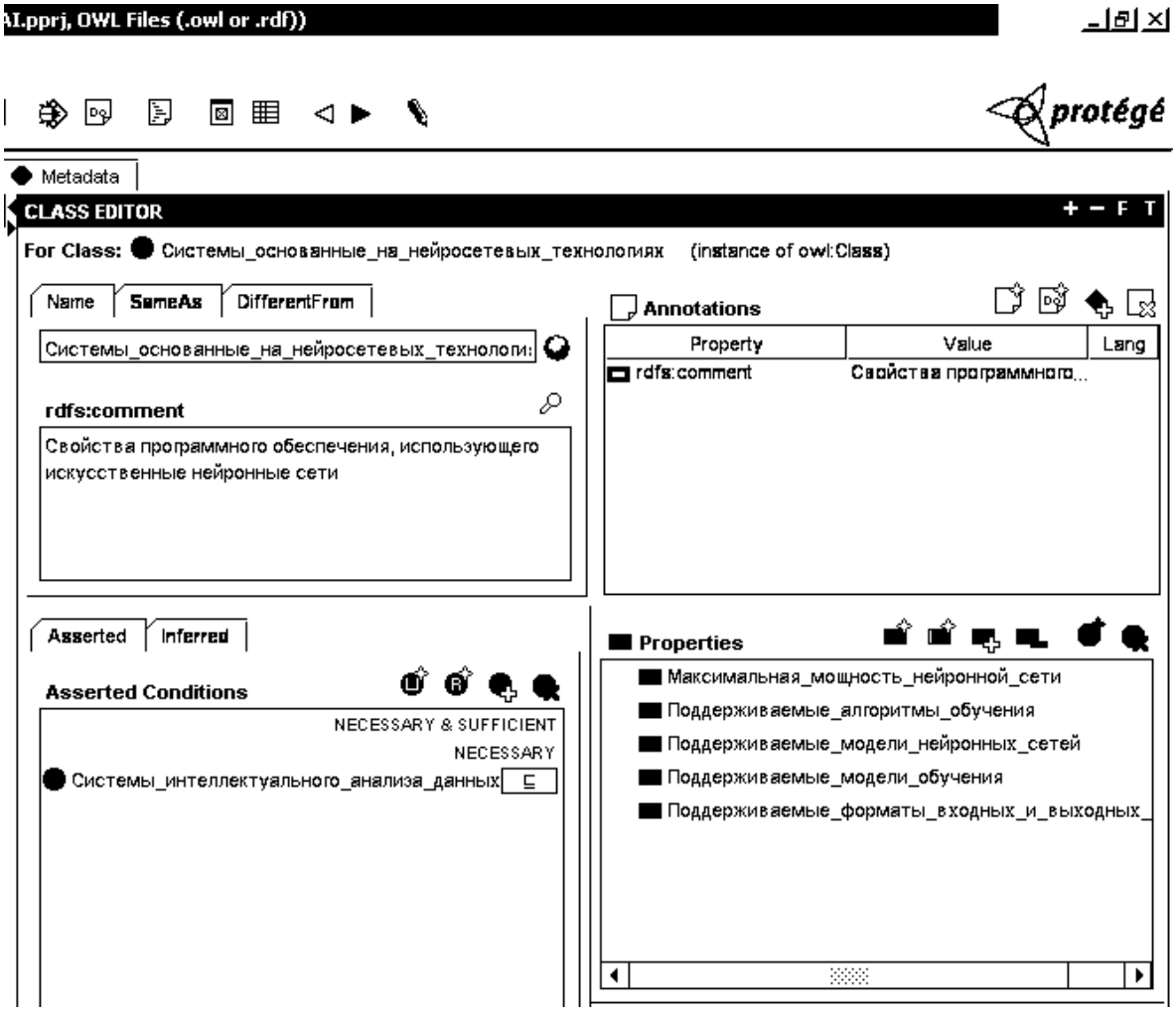


Рис. 3.18. Уточнение понятия предметной области через заполнение слотов класса

Как следует из экранной формы, для каждого слота должны быть заданы тип и возможны диапазон значений, которые он может содержать. Основными типами значений являются строка, число, логическое значение, перечисление.

Последним этапом является создание отдельных экземпляров классов (объектов) в иерархии. Для определения такого экземпляра класса необходимо выбрать класс, создать объект и ввести значения слотов, например создадим объект «Система нейросетевого моделирования» для класса «Интеллектуальные системы» (рис. 3.19).

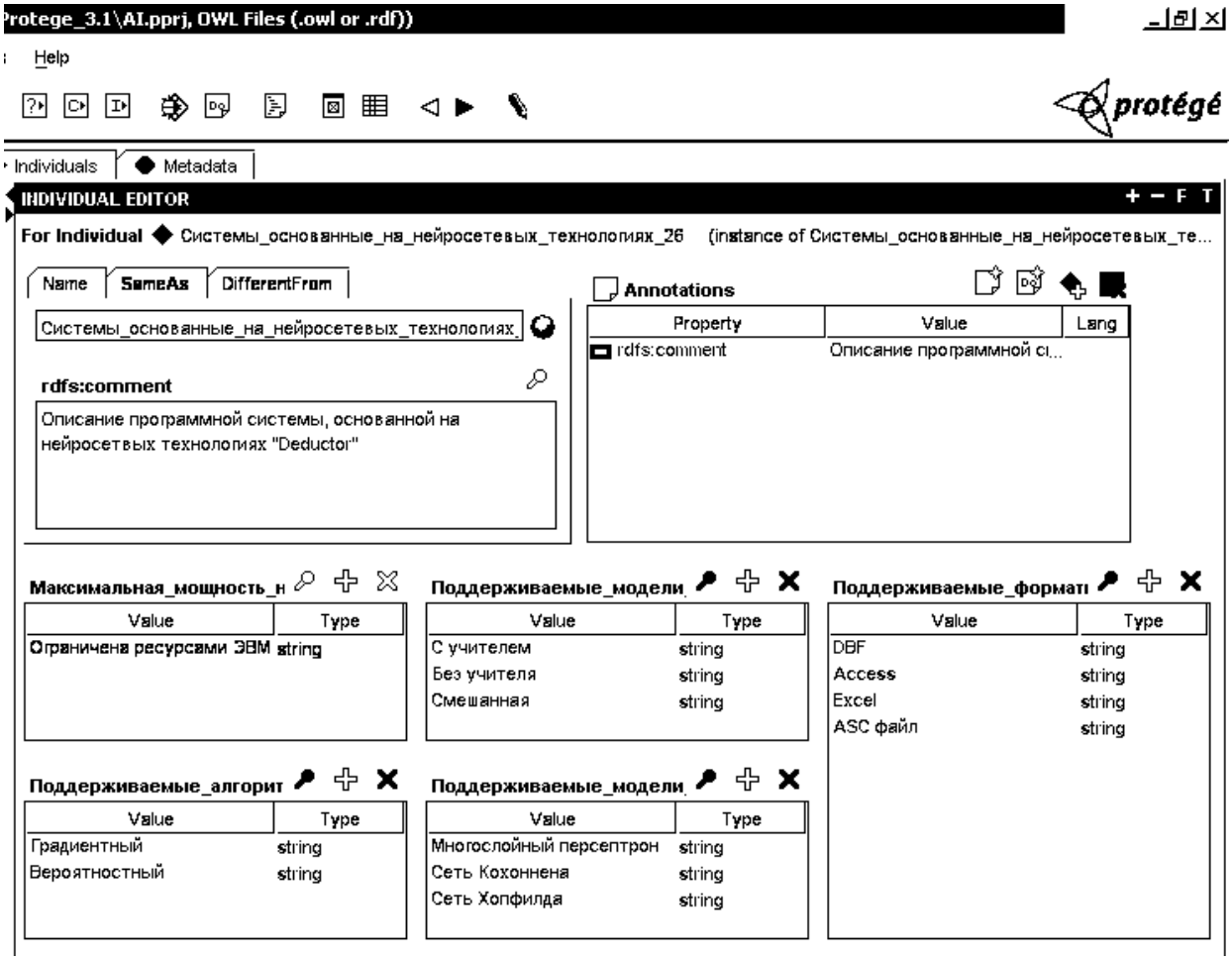


Рис. 3.19. Создание объекта класса

Впоследствии созданная онтология может быть использована как при обработке естественного языка электронных документов, посвященных данной предметной области, так и для интеллектуализации работы поисковых систем, например в сети Интернет.

Таким образом, создание онтологий является перспективной технологией управления информацией о предметной области. Ее использование представляет собой подход к структуризации и систематизации знаний, в частности, для повышения интеллектуальности информационно-поисковых систем. В то же время отсутствие унифицированных правил формирования онтологий приводит к тому, что проектирование онтологий становится исключительно творческой процедурой, зависящей от множества субъективных факторов ее разработчика. Следствиями этого является появление проблемы логической верности построенных онтологий, а также ограниченные возможности применения онтологий в качестве универсального средства описания декларативных знаний предметной области.

Контрольные вопросы

1. В чем смысл понятия «знания» с точки зрения их использования в интеллектуальных системах?

2. Какие существуют основные модели представления знаний?

3. В чем заключаются особенности логической модели представления знаний?

4. Какие основные правила вывода применяются в исчислении высказываний (предикатов)?

5. В чем заключается представление знаний на основе семантической сети?

6. Опишите фрагмент выбранной предметной области (ситуации) в виде семантической сети.

7. Каковы особенности фреймовой модели представления знаний? Приведите пример ее использования для описания знаний предметной области.

8. Каковы особенности продукционной модели представления знаний? Приведите примеры задач, решаемых с помощью продукционных систем.

9. Каковы отличия нечетких множеств от обычных? Назовите компоненты нечеткого множества.

10. Какие операции можно выполнять над нечеткими множествами? Приведите примеры их использования.

11. Дайте определения понятиям «нечеткое отношение», «нечеткая переменная». Приведите примеры решения задач с их использованием.

12. Дайте определение понятию «онтология». Как они применяются для систематизации знаний?

4. ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Одним из наиболее перспективных современных направлений в области искусственного интеллекта является попытка смоделировать мыслительные процессы человека. При этом за основу берутся достижения не в области информатики и математического моделирования, а результаты исследования процессов функционирования нервной системы человека. В то же время знания о работе мозга столь ограничены, что при разработке искусственных нейронных сетей (ИНС) приходится выходить за пределы современных биологических знаний в поисках структур, способных выполнять полезные функции. Несмотря на то, что связь с биологией слаба, искусственные нейронные сети продолжают сравниваться с мозгом. Их функционирование часто напоминает человеческое познание, поэтому трудно избежать этой аналогии.

4.1. Возможности искусственных нейронных сетей

Нервная система человека, построенная из элементов, называемых нейронами, имеет огромную сложность. Около 10^{11} нейронов участвуют в примерно 10^{15} передающих связях, имеющих длину метр и более. Каждый нейрон обладает многими качествами, общими с другими элементами тела, но его уникальной способностью являются прием, обработка и передача электрохимических сигналов по нервным путям, которые образуют коммуникационную систему мозга.

На рис. 4.1 показана структура биологического нейрона. Каждый нейрон имеет отростки нервных волокон двух типов – дендриты, по которым принимаются импульсы, и единственный аксон, по которому нейрон может передавать импульс.

Дендриты идут от тела нервной клетки к другим нейронам, где они принимают сигналы в точках соединения, называемых синапсами. Принятые синапсом входные сигналы подводятся к телу нейрона.

Импульсы, поступившие к нейрону одновременно по нескольким дендритам, суммируются. Если суммарный импульс превышает некоторый порог, нейрон возбуждается, формирует собственный импульс и передает его далее по аксону другим нейронам.

У этой основной функциональной схемы много усложнений и исключений, тем не менее большинство искусственных нейронных сетей моделируют лишь эти простые свойства.

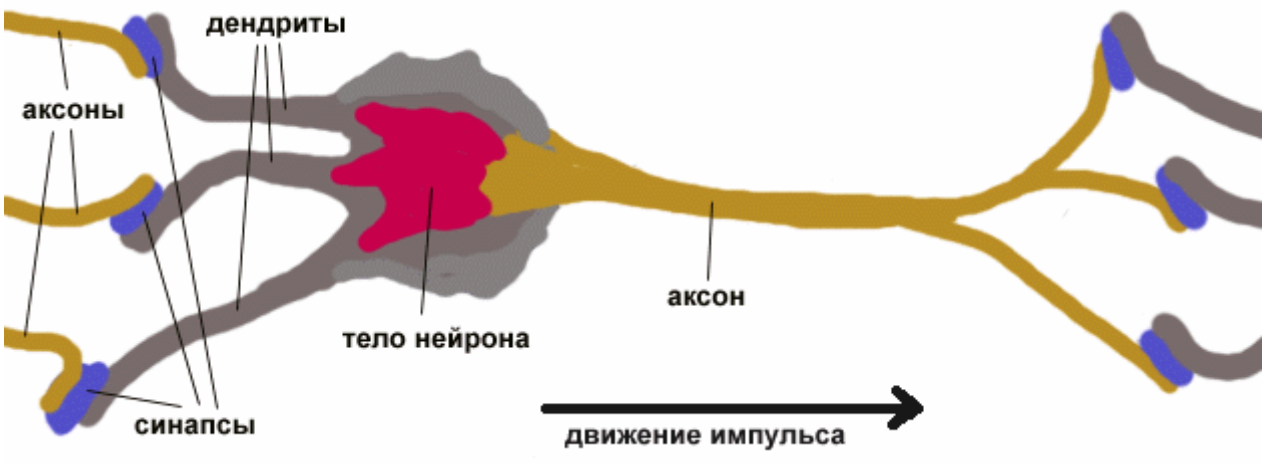


Рис. 4.1. Биологический нейрон

Таким образом, ИНС является приближенной электронной моделью нейронной структуры головного мозга. Известно, что принцип работы мозга отличается от вычислительной работы современных компьютеров с архитектурой фон Неймана, поскольку главным образом связан с процессами обучения на приобретаемом опыте. Поэтому применение нейронных сетей часто дает выигрыш в тех задачах, где применение традиционных подходов к их решению проблематично.

В то же время нейросети нельзя считать универсальным инструментом для решения всех вычислительных проблем. Традиционные компьютеры и вычислительные методы часто являются идеальными для многих применений. Современные цифровые вычислительные машины превосходят человека по способности производить числовые и символьные вычисления. Однако, изменив вычислительный характер задачи на восприятие внешней информации (например, узнавание человека в толпе по его лицу), можно убедиться в

обратном, когда свойственное мозгу естественное мышление в виде сложных образов позволяет решить подобную задачу с высокой скоростью и абсолютной точностью [5].

Такие преимущества ИНС могут быть объяснены наличием у них ряда качеств, которыми не обладают компьютеры с традиционной архитектурой:

- распределенное представление информации и параллельные вычисления;
- способность к обучению и обобщению;
- адаптивность и самоорганизуемость;
- толерантность к ошибкам;
- низкое энергопотребление.

В свою очередь, реализация ИНС на аппаратном уровне, то есть в виде нейрокомпьютера, обладающего отмеченными выше особенностями, приведет к существенным изменениям большинства характеристик, свойственных традиционным компьютерам (табл. 4.1).

Таблица 4.1

Сравнение компьютера на основе архитектуры фон Неймана с нейрокомпьютером

Характеристика	**Традиционный компьютер**	**Нейрокомпьютер**
Процессор	Сложный Высокоскоростной Один или несколько	Простой Низкоскоростной Большое количество
Память	Отдельно от процессора Локальная Адресация по адресу	В процессоре Распределенная Адресация по смыслу
Вычисления	Централизованные Последовательные Алгоритмические	Распределенные Параллельные Самообучение
Надежность	Уязвимость	Живучесть
Специализация	Числовые операции	Проблемы восприятия
Среда функционирования	Строго ограниченная	Без ограничений
Метод обучения	По правилам	По примерам
Применение	Числовая обработка информации	Распознавание, классификация, управление на основе образов

Как видно из табл. 4.1, нейрокомпьютер в сравнении с машиной фон Неймана имеет принципиально другой способ организации вычислительного процесса: он не программируется с использованием явных правил и кодов в соответствии с заданным алгоритмом, а обучается.

Актуальность исследований в этом направлении подтверждается массой различных применений НС. Опыт использования ИНС показал их эффективность в таких задачах, как прогнозирование, распознавание, классификация, аппроксимация функций, адаптивное управление, сжатие данных и др. [1, 5, 13].

4.2. Искусственный нейрон

В качестве научного направления ИНС впервые заявили о себе в 40-е годы 20 в. и связывают это с работами ученых Маккалока и Питтса. Стремясь воспроизвести функции человеческого мозга, исследователи создали простые программные модели биологического нейрона и системы его соединений.

Основу любой ИНС составляют отдельные элементы (ячейки), имитирующие работу нейронов мозга – искусственные нейроны. Каждый нейрон характеризуется своим текущим состоянием по аналогии с нервными клетками головного мозга, которые могут быть возбуждены или заторможены. Он обладает группой синапсов – однонаправленных входных связей, соединенных с выходами других нейронов, а также имеет аксон – выходную связь данного нейрона, с которой сигнал (возбуждения или торможения) поступает на синапсы следующих нейронов. На рис. 4.2 представлен общий вид нейрона. Здесь множество входных сигналов, обозначенных $x_1, x_2, ..., x_n$, поступает на искусственный нейрон. Эти входные сигналы, в совокупности обозначенные вектором X, соответствуют сигналам, приходящим в синапсы биологического нейрона [13].

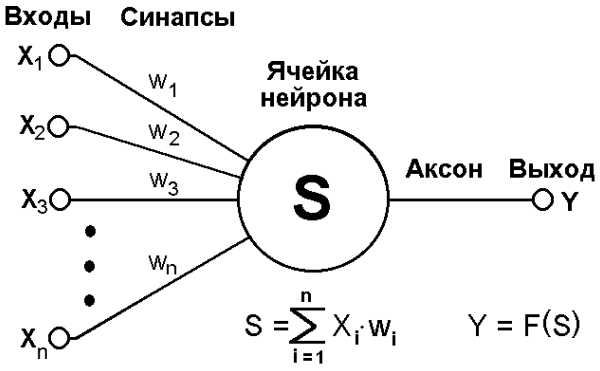


Рис. 4.2. Искусственный нейрон

Каждый сигнал характеризуется величиной синаптической связи или ее весом w_i, который по физическому смыслу эквивалентен электрической проводимости. Множество весов в совокупности обозначается вектором *W*. Суммирующий блок S, соответствующий телу биологического элемента, алгебраически складывает взвешенные входы, определяя текущее состояние нейрона.

$$S = \sum_{i=1}^{n} x_i \cdot w_i . \tag{4.1}$$

В некоторых приложениях ИНС могут быть эффективно использованы другие формулы нахождения взвешенной суммы весов нейрона, например (4.2, 4.3).

$$s = \sum_{i=1}^{n} x_i^2 \cdot w_i ; \tag{4.2}$$

$$s = \sum_{i=1}^{n} x_i \cdot x_{((i+1) \bmod n)} \cdot w_i . \tag{4.3}$$

Введение такого рода нелинейности в целом увеличивает вычислительную мощь сети, то есть позволяет из меньшего числа нейронов с «нелинейными» синапсами сконструировать ИНС, выполняющую работу обычной ИНС с большим числом стандартных нейронов и более сложной конфигурации.

Далее для определения значения выхода нейрона его состояние преобразуется с помощью активационной функции *F*. На рис. 4.3 представлены различные виды такой функции.

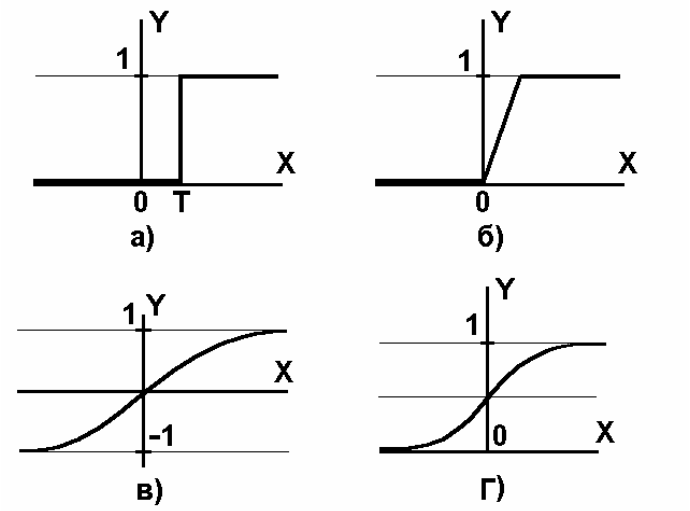


Рис. 4.3. Виды активационной функции:
а - функция единичного скачка; б - линейный порог (гистерезис);
в - сигмоид – гиперболический тангенс; г - сигмоид – формула (4.4)

Одной из наиболее распространенных является нелинейная функция с насыщением, так называемый сигмоид (т.е. функция S-образного вида).

$$f(x) = \frac{1}{1 + e^{-\alpha x}} \quad . \tag{4.4}$$

При уменьшении α сигмоид становится более пологим, в пределе при $\alpha = 0$, вырождаясь в горизонтальную линию на уровне 0,5, при увеличении α сигмоид приближается по внешнему виду к функции единичного скачка с порогом T в точке $x = 0$. Из выражения для сигмоида очевидно, что выходное значение нейрона лежит в диапазоне [0,1]. Одно из ценных свойств сигмоидной функции – простое выражение для ее производной, применение которого будет рассмотрено в дальнейшем.

$$f'(x) = \alpha \cdot f(x)(1 - f(x)).$$

Следует отметить, что сигмоидная функция дифференцируема на всей оси абсцисс, что используется в некоторых алгоритмах обучения. Кроме того, она обладает свойством усиливать слабые сигналы лучше, чем большие и предотвращает насыщение от больших сигналов, так как они соответствуют областям аргументов, где сигмоид имеет пологий наклон.

При создании бинарных ИНС, которые оперируют двоичными сигналами, и выход каждого нейрона может принимать только два значения: логический ноль и логическая единица, часто используется функция единичного скачка. Такая функция имеет два возможных значения: 0, если S меньше порога T и 1 – в противном случае. Если же выходные значение нейрона могут принимать непрерывные значения (аналоговая ИНС), то в качестве активационной функции используется уже сигмоид [13].

В целом же выбор функции определяется характером решаемых задач, а также видом нейронной сети. В настоящее время существует множество разновидностей ИНС, которые можно классифицировать по следующим признакам:

- тип входной информации (аналоговые и двоичные).
- характер обучения (с учителем и без).
- характер настройки синапсов (фиксированные и динамические связи).
- метод обучения (обратное распространение, конкурентное обучение, правило Хебба, гибридные методы).
- характер связей (прямое и прямо-обратное распространение информации).
- архитектура: персептроны, самоорганизующиеся ИНС (Кохонена), ИНС с обратными связями (Хэмминга, Хопфилда, двунаправленная ассоциативная память), гибридные ИНС (встречного распространения).

Рассмотренная ранее простая модель искусственного нейрона игнорирует многие свойства своего биологического двойника. Так, она не принимает во внимание задержки во времени, которые воздействуют на динамику системы. Поэтому наиболее полно свойства биологической системы удается смоделировать при соединении нейронов в слои, а те в свою очередь – в нейронные сети.

Простейшая сеть состоит из группы нейронов, образующих слой (рис. 4.4). Такая система называется персептрон, или однослойный персептрон [5].

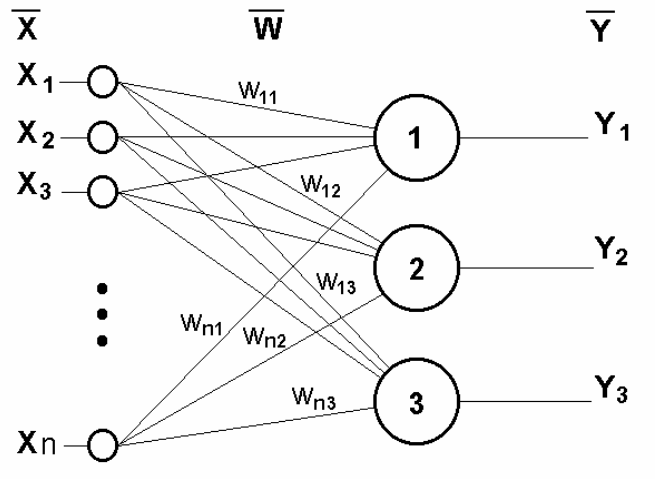


Рис. 4.4. Однослойный персептрон

Как следует из рис. 4.4, каждый элемент из множества входов X отдельным весом соединен с каждым искусственным нейроном. На n входов поступают некие сигналы, проходящие по синапсам на три нейрона, образующие единственный слой этой ИНС и выдающие три выходных сигнала

$$y_j = f\left[\sum_{i=1}^{n} x_i \cdot w_{ij}\right],$$

где $j = 1..3$.

Очевидно, что все весовые коэффициенты синапсов одного слоя нейронов можно свести в матрицу W, в которой каждый элемент w_{ij} задает величину i-й синаптической связи j-го нейрона. Таким образом, процесс, происходящий в ИНС, может быть записан в матричной форме:

$$Y = F(XW),$$

где X, Y – соответственно входной и выходной сигнальные векторы.

Подобная сеть с тремя нейронами может быть использована для решения задачи распознавания восьми объектов. Для этого используется активационная функция в виде единичного скачка, принимающая значения 0 или 1. Каждая комбинация нулей и единиц будет соответствовать определенному объекту. Для примера рассмотрим задачу распознавания буквы алфавита. Цель состоит в том, чтобы ИНС смогла определять букву по входному образу даже если информация

в нем содержит помехи. На рис. 4.5 показана структура нейронной сети для этой задачи [13].

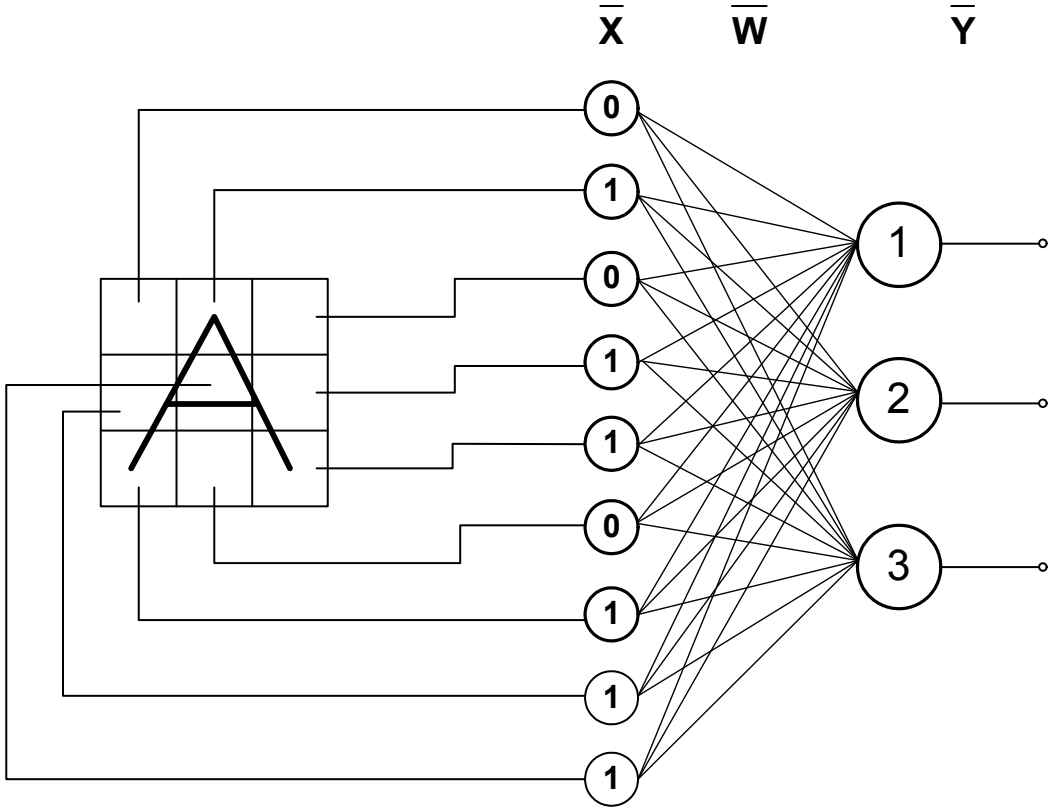


Рис. 4.5. Персептронная система распознавания изображений

Допустим , что входные образы нанесены на карты. Каждая карта разбита на квадраты, и от каждого квадрата на персептрон подается вход. Если через квадрат проходит линия, то соответствующий нейронный вход равен единице, в противном случае он равен нулю. Выход может быть двоичным кодом или числом, представляющим определенную букву.

В качестве другого примера приведем задачу прогнозирования, которая также может быть успешно решена с помощью ИНС. Постановка задачи заключалась в предсказании победы правящей или оппозиционной партии на основе ответов на 12 вопросов. Ниже приводится список таких вопросов.

1. Правящая партия была у власти более одного срока?

2. Правящая партия получила больше 50% голосов на прошлых выборах?

3. В год выборов была активна третья партия?

4. Была серьезная конкуренция при выдвижении от правящей партии?

5. Кандидат от правящей партии был президентом в год выборов?

6. Был ли год выборов временем спада или депрессии?

7. Был ли рост среднего национального валового продукта на душу населения больше 2.1%?

8. Произвел ли правящий президент существенные изменения в политике?

9. Во время правления были существенные социальные волнения?

10. Администрация правящей партии виновна в серьезной ошибке или скандале?

11. Кандидат от правящей партии – национальный герой?

12. Кандидат от оппозиционной партии – национальный герой?

Ответы на вопросы описывают ситуацию на момент, предшествующий выборам. Ответы кодировались следующим образом: «да» - единица, «нет» - минус единица. Отрицательный сигнал на выходе сети интерпретируется как предсказание победы правящей партии. В противном случае ответом считается победа оппозиционной партии. Все нейроны реализуют пороговую функцию, равную единице, если алгебраическая сумма входных сигналов нейрона больше либо равна нулю, и минус единица при сумме меньшей нуля.

Отметим, что представление входной информации в виде комбинаций нулей и единиц является достаточно удобным для обучения нейросети. однако это не всегда возможно, особенно в случае, если элементы входных сигналов принадлежат разным типам данных или интервалы значений входных данных существенно различаются.

Для того чтобы сеть в процессе своей работы выдавала правильный результат, нужно определить значения весов ее синаптических связей. Эта задача является одной из самых сложных наряду с определением топологии сети.

Для настройки ИНС, то есть определения весов связей используются специализированные алгоритмы обучения. Способность ИНС обучаться является, пожалуй, самым ярким их свойством. Впервые такая возможность была доказана Розенблатом. Цель обучения сети заключается в том, чтобы для некоторого множества входов она

могла выдавать желаемое (или, по крайней мере, близкое к нему) множество выходов. Каждое такое входное (или выходное) множество рассматривается как вектор. Обучение ведется путем последовательного предъявления входных векторов с одновременной подстройкой весов в соответствии с определенной процедурой. В процессе обучения веса сети постепенно становятся такими, чтобы каждый входной вектор вырабатывал выходной вектор.

Различают две основные группы алгоритмов обучения: с учителем и без учителя.

Обучение с учителем предполагает, что для каждого входного вектора существует целевой вектор, представляющий собой требуемый выход. Вместе они называются обучающей парой. Обычно сеть обучается на некотором числе таких обучающих пар. Предъявляется выходной вектор, вычисляется выход сети и сравнивается с соответствующим целевым вектором, разность (ошибка) с помощью обратной связи подается в сеть и веса изменяются в соответствии с алгоритмом, стремящимся минимизировать ошибку. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня.

Несмотря на многочисленные прикладные достижения, обучение с учителем критиковалось за свою биологическую неправдоподобность. Трудно вообразить обучающий механизм в мозге, который бы сравнивал желаемые и действительные значения выходов, выполняя коррекцию с помощью обратной связи. Если допустить подобный механизм в мозге, то откуда тогда возникают желаемые выходы? Обучение без учителя является намного более правдоподобной моделью обучения в биологической системе. Развитая Кохоненом и многими другими, она не нуждается в целевом векторе для выходов и, следовательно, не требует сравнения с предопределенными идеальными ответами. Обучающее множество состоит лишь из входных векторов. Обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные векторы, то есть чтобы предъявление достаточно близких входных векторов давало одинаковые

выходы. Процесс обучения выделяет статистические свойства обучающего множества и группирует сходные векторы в классы. Предъявление на вход вектора из данного класса даст определенный выходной вектор, но до обучения невозможно предсказать, какой выход будет производиться данным классом входных векторов. Выходы такой сети должны трансформироваться в форму, по которой можно идентифицировать связь между входом и выходом, установленную сетью [5].

Кроме этого, алгоритмы обучения могут разделяться на детерминированные и стохастические. В первом случае подстройка весов представляет собой жесткую последовательность действий, во втором – она производится на основе действий, подчиняющихся некоторому случайному процессу.

Для задач прогнозирования и распознавания, как правило, используется алгоритм обучения с учителем, который состоит из следующих шагов.

1. Проинициализировать элементы весовой матрицы (обычно небольшими случайными значениями).

2. Подать на входы один из входных векторов, которые сеть должна научиться различать, и вычислить ее выход.

3. Если выход правильный, перейти на шаг 4.

Иначе вычислить разницу между эталонным и полученным значениями выхода

$$\delta = Y_I - Y$$

Модифицировать веса в соответствии с формулой

$$w_{ij}(t+1) = w_{ij}(t) + \nu \cdot \delta \cdot x_i ,$$

где t, t+1 – номера соответственно текущей и следующей итераций; ν – коэффициент скорости обучения, $0 < \nu < 1$ ($\nu_{\text{opt}} = 0{,}6 \ldots 0{,}8$); i – номер входа; j – номер нейрона в слое.

Очевидно, что, если $Y_I > Y$ весовые коэффициенты будут увеличены, и они уменьшат ошибку. В противном случае они будут уменьшены, и Y тоже уменьшится, приближаясь к Y_I.

Коррекция весов с использованием коэффициента различия (δ) между требуемым и реальным выходом называется дельта-правилом.

4. Цикл с шага 2, пока сеть не перестанет ошибаться.

На втором шаге на разных итерациях поочередно в случайном порядке предъявляются все возможные входные вектора. К сожалению, нельзя заранее определить число итераций, которые потребуется выполнить, а в некоторых случаях и гарантировать полный успех.

Итак, работа всех ИНС сводится к классификации (обобщению) входных сигналов, принадлежащих n - мерному гиперпространству, по некоторому числу классов. С математической точки зрения это происходит путем разбиения гиперпространства гиперплоскостями. Для однослойного перцептрона такая плоскость может быть выражена как

$$\sum_{i=1}^{n} x_i \cdot w_{ik} = T_k \ , \ k = 1 \ldots m$$

Каждая полученная область является областью определения отдельного класса. Число таких классов для одной НС перцептронного типа не превышает 2^m, где m – число выходов сети.

Однако не все задачи ИНС могут быть ею решены. Так, однослойный перцептрон, состоящий из одного нейрона с двумя входами (рис. 4.6) не способен воспроизвести такую функцию как «исключающее или» (эта задача известна как проблема «исключающее ИЛИ») [13]. Рассмотрим ее подробнее.

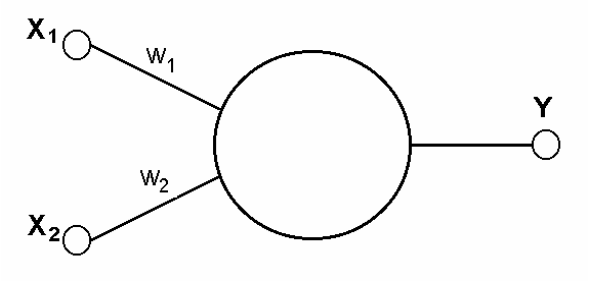


Рис. 4.6. Перцептрон для задачи «исключающее ИЛИ»

Все возможные комбинации сигналов на входах x_1 , x_2 будут состоять из четырех точек на плоскости $x_1 - x_2$ (рис. 4.7). Так, точка $x_1 = 0$ и $y2 = 0$ обозначена на рисунке как точка A_0.

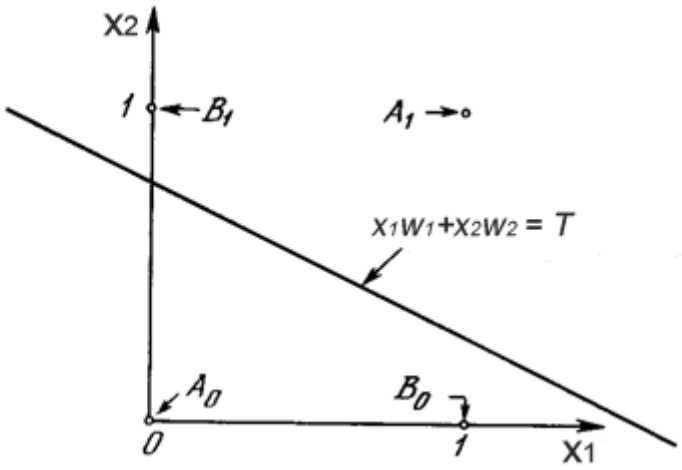


Рис. 4.7. Задача «исключающее ИЛИ»

В табл. 4.2 приведены требуемые связи между входами и выходом, где входные комбинации, которые должны давать нулевой выход, помечены A_0 и A_1, единичный выход – B_0 и B_1.

Таблица 4.2

Таблица истинности для функции исключающее ИЛИ

Точки	**Значения x_1**	**Значения x_2**	**Требуемый выход**
A_0	0	0	0
B_0	1	0	1
B_1	0	1	1
A_1	1	1	0

В качестве активационной функции используется сети пороговая функция, где Y принимает значение ноль, когда S меньше 0,5, и единица в случае, когда S больше или равно 0,5. Нейрон выполняет следующее вычисление:

$$S = x_1 w_1 + x_2 w_2 .$$

Никакая комбинация значений двух весов не может дать соотношения между входом и выходом, задаваемого табл. 4.2. Чтобы понять это ограничение, зафиксируем S на величине порога 0,5. Сеть в этом случае описывается уравнением (4.5). Это уравнение линейно по x_1 и x_2, то есть все значения по x_1 и x_2, удовлетворяющие этому уравнению, будут лежать на некоторой прямой в плоскости $x_1 - x_2$.

$$x_1 w_1 + x_2 w_2 = 0{,}5 \qquad (4.5)$$

Любые входные значения для x_1 и x_2 на этой линии будут давать пороговое значение 0,5 для S. Входные значения с одной стороны прямой обеспечат значения S больше порога, следовательно, $Y = 1$. Входные значения по другую сторону прямой обеспечат значения S меньше порогового значения, делая Y равным 0. Изменения значений w_1, w_2 и порога будут менять наклон и положение прямой. Для того чтобы сеть реализовала функцию исключающее ИЛИ, заданную табл. 4.2, нужно расположить прямую так, чтобы точки А были с одной стороны прямой, а точки В – с другой. Попытавшись нарисовать такую прямую на рис. 4.7, можно убедится, что это невозможно. Это означает, что какие бы значения ни приписывались весам и порогу, сеть неспособна воспроизвести соотношение между входом и выходом, требуемое для представления функции исключающее ИЛИ.

Функции, которые не реализуются однослойной сетью, называются линейно неразделимыми. Решение задач, подпадающих под это ограничение, заключается в применении 2-х и более слойных сетей или сетей с нелинейными синапсами, однако и тогда существует вероятность, что корректное разделение некоторых входных сигналов на классы невозможно.

4.3. Многослойные нейронные сети

Одной из главных особенностей ИНС является параллельный характер ее работы. Это достигается путем объединения большого числа нейронов в слои и соединения определенным образом нейронов различных слоев. При этом обработка взаимодействия всех нейронов ведется послойно.

Теоретически число слоев и число нейронов в каждом слое может быть произвольным, однако фактически оно ограничено ресурсами компьютера или специализированной микросхемы, на которых реализуется ИНС. Чем сложнее ИНС, тем более сложные задачи она может решать. Так, с помощью многослойной ИНС с двумя перерабатывающими слоями можно с любой точностью аппроксимировать

любую многомерную функцию на отрезке [0;1], что следует из теоремы Колмогорова, утверждающей, что любую непрерывную многомерную функцию на единичном отрезке [0;1] можно представить в виде конечного числа одномерных.

Замечание. Отмеченная теорема имеет свой вариант, предложенный Хехт-Нильсоном в терминах нейросетей: функция многих переменных достаточно общего вида может быть представлена с помощью двухслойной нейронной сети с прямыми полными связями с n компонентами входного сигнала, $2m+1$ компонентами первого («скрытого» слоя) с заранее известными ограниченными функциями активации (например, сигмоидальными) и m компонентами второго слоя с неизвестными функциями активации. Данная теорема позволяет получить доказательство решаемости задачи представления функции достаточно произвольного вида на ИНС, а также указание для задач этого типа минимальных значений числа нейронов сети, необходимых их решения.

Ранее был приведен алгоритм обучения однослойной ИНС, когда подстройка синаптических связей идет в направлении, минимизирующем ошибку на выходе сети. В многослойных же ИНС идеальные выходные значения нейронов всех слоев, кроме последнего, как правило, не известны и такую сеть уже невозможно обучить, руководствуясь только величинами ошибки на выходах ИНС. Один из вариантов решения этой проблемы – разработка выходных сигналов, соответствующих входным, для каждого слоя ИНС, что является очень трудоемкой операцией и не всегда возможно.

Поэтому для обучения многослойной ИНС используется процедура, известная как метод обратного распространения ошибки. Математическое обоснование этого метода строится на оптимизационном методе градиентного спуска, задача которого в этом случае заключается в минимизации ошибки сети.

Данный метод обучения применим к ИНС с любым числом слоев. Далее для простоты будет рассматриваться двухслойная нейронная сеть (рис. 4.8), которая включает входной, скрытый и выходной слои, также будем считать, что число нейронов в слоях одинаковое.

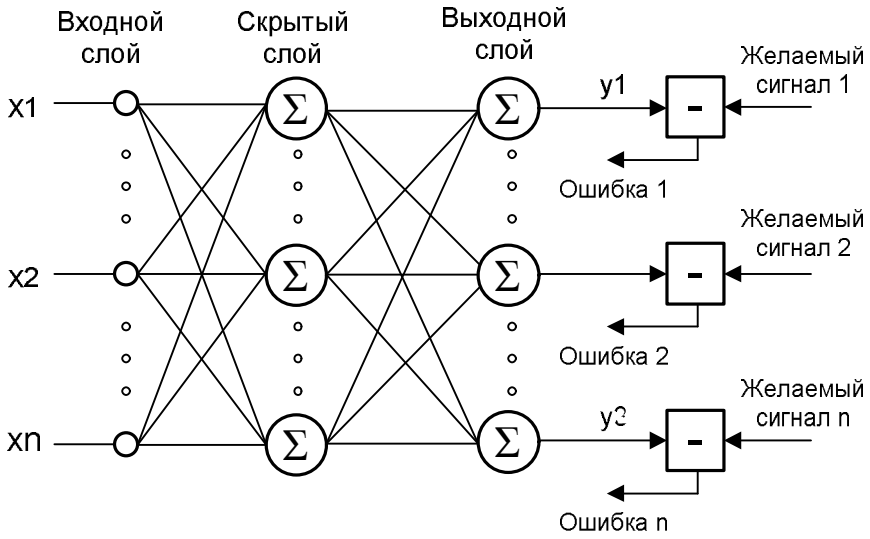


Рис. 4.8. Нейронная сеть для задачи обучения многослойных ИНС

В работе процедуры обратного распространения ошибки можно выделить два основной этапа: проход вперед и обратный проход.

Проход вперед. Сети последовательно подаются на вход вектора X из обучающего множества, для каждого из которых определяется выходной вектор Y. Обработка происходит послойно, то есть множество выходов предыдущего слоя является входами для следующего. В векторном виде этот процесс можно записать как

$$Y^l = F(X^l W^l),\ l = 1\ ..\ M,$$

где M – число слоев;

W – матрица весов между нейронами слоя l;

$X^l W^l$ - текущее состояние нейронов слоя (S^l) в векторной форме;

Y^l - вектор значений выходов слоя l, является входом ($Y^l = X^{l+1}$) для слоя l+1.

Важно отметить, что активационная функция F должна выбираться с учетом требования ее дифференцируемости на всей области определения. Поэтому в качестве такой функции часто используется сигмоид.

Обратный проход

1. С помощью модифицированного дельта – правила (4.6) корректируются веса выходного слоя (4.7, 4.8)

$$\delta^m = (Y^m - d^m)F'(S^m), \qquad (4.6)$$

где δ^m - величина ошибки (коэффициент коррекции веса) в векторной форме для слоя m;

d^m - вектор эталонных выходных значений нейронов слоя m.

$$\Delta w_{i,j}^m = -\nu \cdot \delta_j^m \cdot y_i^{m-1},\ i = 1..n, j = 1..\ n, \qquad (4.7)$$

где n – число нейронов в слое;

$\Delta w_{i,j}^m$ - изменение веса между i – м и j – м нейроном слоев m-1 и m соответственно;

δ_j^m - коэффициент коррекции веса слоя m нейрона j;

y_i^{m-1} - значение выхода нейрона i, слоя m-1.

$$w_{i,j}^l = w_{i,j}^l + \Delta w_{i,j}^l \qquad (4.8)$$

2. По формулам (4.9, 4.10) рассчитываются соответственно δ^l и Δw^l для всех остальных слоев, $l = 1..m$-1.

$$\delta_i^{(l)} = \left[\sum_{j=1}^{n} \delta_j^{(l+1)} \cdot w_{i,j}^{(l+1)} \right] \cdot f'(y_i). \qquad (4.9)$$

Согласно этой формуле, ошибка предыдущего слоя определяется рекурсивно через ошибку следующего.

3. Корректируются все веса в ИНС на основе (4.8).

4. Определяется среднеквадратичная ошибка как разница между эталонным и выходным вектором

$$E = \tfrac{1}{2} \sum_{p=1}^{P} (y_{i,p}^m - d_{i,p})^2\ ,$$

где P – число образов для обучения ИНС;

$y_{i,p}^m$ - реальное выходное значение нейрона i при подаче на входы ИНС p – го образа;

$d_{i,p}$ - эталонное выходное значение этого нейрона.

Если ошибка существенна, то перейти на шаг 1. Иначе закончить.

В целом метод обратного распространения ошибки позволяет достаточно эффективно проводить обучение многослойных ИНС, однако в сложных задачах его использование может быть сопряжено с

целым рядом особенностей, приводящих к недопустимо долгому времени обучения. Рассмотрим их подробнее.

Снижение эффективности обучения. Из выражения (4.7) следует, что, когда выходное значение $y_i^{(m-1)}$ стремится к нулю, эффективность обучения заметно снижается. При двоичных входных векторах в среднем половина весовых коэффициентов не будет корректироваться, поэтому область возможных значений выходов нейронов [0,1] желательно сдвинуть в пределы [-0,5, 0,5], что достигается простыми модификациями логистических функций, например сигмоид с экспонентой преобразуется к виду

$$f(x) = -0.5 + \frac{1}{1 + e^{-\alpha \cdot x}} \ .$$

С помощью таких средств время сходимости алгоритма обучения сокращается в среднем от 30 до 50%.

Нормализация входных значений. Значения вектора, предъявляемого на вход сети, могут сильно отличаться друг от друга (то есть включать как очень большие, так и очень малые). В таком случае их рекомендуется нормализовать в единичный вектор с помощью деления каждой компоненты входного вектора на его длину.

$$x_i^* = \frac{x_i}{\sqrt{\sum_{j=1}^{n} x_j^2}} . \qquad (4.10)$$

Влияние диапазона начальных весов решение. Всем весам сети перед началом обучения следует придать начальные значения. При этом рекомендуется присвоить им небольшие значения, которые могут быть определены по формуле.

$$w_{i,j}^l = \frac{1}{\sqrt{n^l}} , \qquad (4.11)$$

где n^l - число нейронов в слое l.

Выбор шага обучения. В процессе обучения сети значения весов могут в результате коррекции стать очень большими, тогда в соответствии с (4.6, 4.9) производная $(F'(S^l) = \frac{\partial Y^l}{\partial S^l})$ будет очень мала и

обучение остановится. Кроме этого, применение градиентного метода оптимизации в большинстве случаев позволяет найти лишь субоптимальный, а не глобальный минимум целевой функции. Эти проблемы связаны с выбором величины скорости обучения. Доказательство сходимости обучения в процессе обратного распространения основано на производных, то есть приращения весов, и, следовательно, скорости обучения должны быть бесконечно малыми, однако в этом случае обучение будет происходить неприемлемо медленно. С другой стороны, слишком большие коррекции весов могут привести к постоянной неустойчивости процесса обучения. Поэтому в качестве v обычно выбирается число меньше 1, но не очень маленькое, например 0,1, и оно может постепенно уменьшаться в процессе обучения. Кроме того, для исключения случайных попаданий в локальные минимумы иногда, после того как значения весовых коэффициентов стабилизируются, v кратковременно сильно увеличивают, чтобы начать градиентный спуск из новой точки. Если повторение этой процедуры несколько раз приведет алгоритм в одно и то же состояние ИНС, можно более или менее уверенно утверждать, что найден именно глобальный минимум.

Выбор структуры ИНС. Определение оптимальной структуры (топологии, конфигурации) ИНС, а именно: числа слоев и нейронов в них, является трудной задачей и представляет собой целое направление нейрокомпьютерной науки. Тем не менее можно выделить следующие рекомендации по определению топологии ИНС [5, 13].

1. Для некоторых типовых задач (распознавания, классификации) уже существуют оптимальные конфигурации ИНС, поэтому целесообразно попытаться использовать именно их.

2. Необходимо соизмерять объем сети с числом обучающих примеров, число которых должно быть пропорционально сложности сети.

3 Число нейронов в скрытых слоях однородных многослойных ИНС с сигмоидальными активационными функциями может быть рассчитана на основе экспериментальной формулы для оценки необходимого числа синаптических весов N_w

$$\frac{N_y N_p}{1+\log_2(N_p)} \le N_w \le N_y \cdot \left(\frac{N_p}{N_x}+1\right) \cdot (N_x+N_y+1) + N_y,$$

где N_y - размерность выходного сигнала;

N_x - размерность входного сигнала;

N_p - число элементов обучающего множества.

Кроме этого, для повышения эффективности процесса обучения (не менее, чем в 10-100 раз при прочих равных условиях) можно рекомендовать использование метода многостраничного обучения, который предполагает особым образом подходить к формированию обучающей выборки. Под страницей подразумевается серия примеров, предъявляемая сети для обучения, при этом модификации параметров сети осуществляются, исходя из всех примеров страницы. При использовании данного метода необходимо учитывать следующее:

- первую страницу рекомендуется формировать из опорных примеров, характеризующих особенности функции, которую должна будет реализовывать обученная нейронная сеть;

- в ходе обучения объем страницы и разнообразие примеров на ней можно увеличить (совсем необученная сеть слишком медленно учится на больших страницах, а после предварительного обучения появляются возможности для быстрого освоения все больших страниц);

- каждая страница должна быть достаточно разнообразной и в ней должны присутствовать представители разных групп примеров.

В большинстве же случаев оптимальный вариант конфигурации ИНС может быть получен на основе интуитивного выбора или экспериментально.

Число нейронов в выходном слое. Последний слой нейронов играет важную роль при работе ИНС, поскольку именно он отвечает за окончательную классификацию образов. Так, для разделения множества входных образов по двум классам достаточно всего одного нейрона. При этом каждый логический уровень – «1» и «0» – будут

обозначать отдельный класс. На двух выходах можно закодировать уже 4 класса и так далее. Однако результаты работы сети, имеющей максимально плотную структуру, не очень надежны. Для повышения достоверности классификации желательно ввести избыточность путем выделения каждому классу одного нейрона в выходном слое или, что еще лучше, нескольких, каждый из которых обучается определять принадлежность образа к классу со своей степенью достоверности, например высокой, средней и низкой. Такие ИНС позволяют проводить классификацию входных образов, объединенных в нечеткие (размытые или пересекающиеся) множества. Это свойство приближает подобные НС к условиям реальной жизни.

Рассмотрим возможности применения ИНС для оценки кредитоспособности физического лица. В качестве факторов, определяющих кредитоспособность, будем использовать возраст, образование, площадь квартиры, наличие автомобиля, длительность проживания в данном регионе. В табл. 4.3 приведена часть накопленной статистики возвратов или невозвратов взятых кредитов, в некоторой финансовой организации.

Таблица 4.3

Фрагмент статистики по кредитам

Сумма кредита, руб.	**Возраст**	**Образование**	**Площадь квартиры, м^2**	**Автомобиль**	**Срок проживания**	**Давать кредит**
13500	37	Специальное	46	Отечеств.	28	Нет
7500	38	Среднее	29	Импорт.	12	Да
14500	60	Высшее	34	Нет	30	Нет
15000	28	Специальное	14	Отечеств.	21	Да
32000	59	Специальное	53	Отечеств.	29	Да
11500	25	Специальное	38	Отечеств.	9	Да
61500	29	Высшее	26	Нет	18	Нет
25500	68	Высшее	45	Отечеств.	30	Нет
…	…	…	…	…	…	…

Данные из табл. 4.3 являются обучающим множеством, на котором будет тренироваться ИНС. Учитывая тот факт, что поля имеют как числовые, так и символьные значения, требуется их нормализация, которая в данном случае может быть проведена следующими способами.

1. Линейная нормализация. Используется только для непрерывных числовых полей. Позволяет привести числа к диапазону [min..max], где минимальному числу из исходного диапазона будет соответствовать min, а максимальному – max. Остальные значения распределятся между min и max. Пример такой нормализации к диапазону [0..1] приведен в табл. 4.4.

Таблица 4.4

Пример линейной нормализации

Поле до нормализации	**Поле после нормализации**
-5 (min)	0
2,3	0,8111
1.1	0,6777
4 (max)	1
3,5	0,9444

Графическая интерпретация такого преобразования представлена на рис. 4.9.

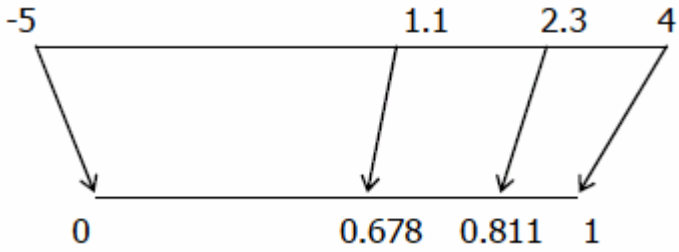


Рис. 4.9. Линейная нормализация

2. Уникальные значения. Используется для дискретных значений. Такими являются строки, числа или даты, заданные дискретно. Если величины можно сравнить друг с другом, то их нормализация заключается в их упорядочивании по возрастанию и нумерации с последующей заменой их значений порядковым номером.

Пример нормализации этого вида приведен в табл. 4.5.

Таблица 4.5

Пример нормализации уникальных значений

Поле до нормализации	**Поле после нормализации**
Маленький	1
Средний	2
Большой	3
Огромный	4

3. Битовая маска. Используется для дискретных значений. Этот вид нормализации следует использовать для величин, которые можно только сравнивать на равенство или неравенство, но нельзя определить, какое больше, а какое меньше. Все значения заменяются порядковыми номерами, а номер рассматривается в двоичном виде или в виде маски из нулей и единиц. Тогда каждая позиция маски рассматривается как отдельное поле, содержащее ноль или единицу. После такой нормализации на вход нейросети будет подаваться не одно это поле, а столько полей, сколько разрядов в маске. Пример нормализации этого вида приведен в табл. 4.6, где каждому символьному значению поля ставится в соответствие двоичный код.

Таблица 4.6

Пример нормализации уникальных значений

Поле до нормализации	**Поля после нормализации (битовая маска)**
Москва	00
Воронеж	01
Рязань	10
Тула	11

Таким образом, поля «Сумма кредита», «Возраст», «Площадь квартиры» и «Длительность проживания», имеющие непрерывные значения, с помощью линейной нормализации будут преобразованы к интервалу [-1..1]. Образование представлено тремя уникальными значениями, которые, сравнив, можно упорядочить по возрастанию как среднее, специальное и высшее. Следовательно, здесь может быть

использован второй вид нормализации. Значения поля «Автомобиль» явно упорядочить нельзя, поэтому с помощью третьего типа нормализации их необходимо закодировать в виде битовой маски. Для этого потребуется два бита (импортный автомобиль – 00, отечественный автомобиль - 01, нет автомобиля - 10).

Дальнейшее решение этой задачи будет выполняться в системе интеллектуального анализа данных Deductor фирмы BaseGroup Labs.

Вначале необходимо импортировать исходные данные и назначить категории полям таблицы (входное, выходное, информационное). После чего в обучающем множестве определяется число записей, используемых для тестирования ИНС. На следующем шаге должна быть выбрана конфигурация ИНС и параметры обучения. Для рассматриваемой задачи использовалась ИНС со структурой 7x2x1, где во входном слое семь нейронов, скрытом – два нейрона и в выходном – один нейрон, значение выхода которого является решением о предоставлении кредита.

После обучения нейросеть можно использовать для принятия решения о выдаче кредита физическому лицу. Это можно сделать, применяя анализ «что-если» (рис. 4.10).

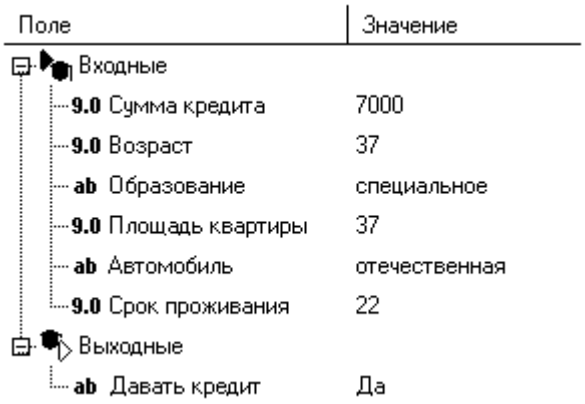


Рис. 4.10. Определение с помощью ИНС решения о предоставлении кредита

Кроме этого, может быть определена зависимость выходного поля от одного из входных факторов при фиксированных значениях остальных. Так, требуется узнать, на какую сумму кредита может рассчитывать человек, обладающий определенными характеристиками (рис. 4.11).

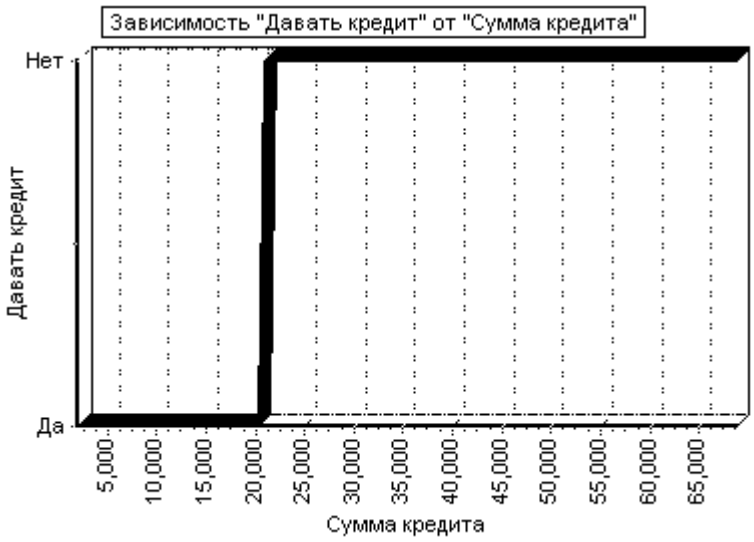


Рис. 4.11. Зависимость результата от заданного фактора

Из приведенной диаграммы можно сделать вывод, что человек в возрасте 37 лет со специальным образованием, имеющий квартиру площадью 37 кв. м, отечественный автомобиль и проживающий в данной местности 22 года может рассчитывать на сумму кредита не больше 20 000.

В целом целесообразность применения ИНС для решения задачи прогнозирования не ограничивается только возможностью работы с произвольными типами данных, но и связана с независимостью от равномерности распределения точек аппроксимации, наличием гибких средств нормализации исходных данных, а также эффективностью в задачах большой размерности.

Как отмечалось ранее, существует две основные концепции обучения ИНС: с учителем и без учителя. Если в первом случае процедура обучения подразумевает предоставление сети как входные, так и выходные образы, то во втором случае для обучения сети достаточно только входной информации. Подобные возможности ИНС открывают целый ряд областей, где они могут быть успешно применены, особенно в задачах классификации и кластеризации информации.

4.4. Самоорганизующиеся нейронные сети

Нейронные сети, использующие принципы обучения без учителя, часто относят к самоорганизующимся, а именно способным особым образом группировать и обобщать информацию (следует отметить, что не все специалисты в области ИНС поддерживают такую терминологию, так как под самоорганизацией обычно подразумевают изменение структуры системы, а не связанной с ней информации, поэтому более логичным будет термин самообучающиеся нейронные сети).

Процесс обучения без учителя, как и в случае обучения с учителем, заключается в подстраивании весов синапсов. При этом их подстройка может проводиться только на основании информации о состоянии нейрона и уже имеющихся значениях весовых коэффициентов.

Среди алгоритмов, предназначенных для данного типа обучения, можно выделить два - предложенных Хеббом и Кохоненном [5, 13]. Каждый из них имеет строгую математическую теорию, а также ряд модификаций. Однако именно модель обучения без учителя, предложенная Кохоненном в 1984 году и являющаяся классической, получила наибольшее распространение при решении практических задач. Причинами этого являются более точная работа, а также близость заложенных в ней идей к реальному функционированию мозга.

В мозге нейроны располагаются в определенном порядке так, что некоторые внешние физические воздействия вызывают ответную реакцию нейронов из определенной области мозга. Так, в той части мозга, которая отвечает за восприятие звуковых сигналов, нейроны группируются в соответствии с частотами входного сигнала, на которых они резонируют. Хотя строение мозга в значительной степени предопределяется генетически, отдельные структуры мозга формируются в процессе самоорганизации. Алгоритм Кохоненна в некоторой степени напоминает процессы, происходящие в мозге.

Алгоритм Кохоненна дает возможность строить нейронную сеть для разделения векторов входных сигналов на подгруппы. Сеть состоит из *N* нейронов, образующих прямоугольную решетку на плоскости (рис. 4.12). Элементы входных сигналов подаются на входы всех нейронов сети. В процессе работы алгоритма настраиваются синаптические веса нейронов.

Входные сигналы - вектора действительных чисел - последовательно предъявляются сети. Желаемые выходные сигналы не определяются. После того, как было предъявлено достаточное число входных векторов, синаптические веса сети определяют кластеры.

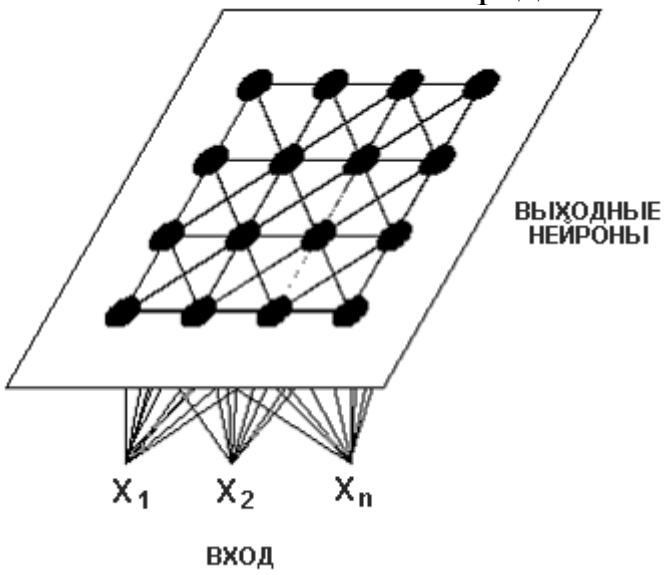


Рис. 4.12. Сеть Кохоненна

Таким образом, с помощью обучения без учителя ИНС позволяют эффективно решать еще один класс задач – кластеризации или классификации информации. Несмотря на то, что в этом контексте эти задачи схожи по характеру, однако в других случаях они различаются. Поэтому следует дать определения терминам класс и кластер.

Класс – множество объектов, имеющих схожие признаки, характеристики или свойства.

Кластер – множество объектов, находящихся на расстоянии друг от друга, не превышающем некоторое значение меры близости.

Для отнесения объекта к кластеру необходимо найти кластер с минимальным расстоянием от этого объекта до центроида кластера.

Как правило, для этого используется расчет Евклидова расстояния или расстояния Хеминга.

В результате решения задачи кластеризации получается карта кластеров, где близко расположенным кластерам соответствуют более близкие друг к другу входные вектора нейросети. Все это позволяет выполнять наглядное упорядочивание многопараметрической информации. Важно, что при этом могут быть обнаружены неожиданные скопления «близких» данных, последующая интерпретация которых пользователем может привести к получению нового знания об исследуемом объекте.

Рассмотрим шаги процесса обучения нейросети по алгоритму, предложенному Кохоненном. Отметим, что в данном случае под нейросетью понимается однослойный персептрон – слой Кохоненна [13].

1. Инициализация сети.

Весовым коэффициентам сети присваиваются малые случайные значения, которые рекомендуется определять по формуле (4.11).

2. Сети предъявляется новый входной вектор.

При этом рекомендуется выполнить нормализацию значений вектора по формуле (4.10).

3. Вычисляется Евклидово расстояние до всех нейронов слоя Кохоненна

$$D_j = \sqrt{\sum_{i=1}^{n} (x_i - w_{i,j})^2} ,$$

где x_i - компонента i входного вектора X;

$w_{i,j}$ - вес входа i нейрона j.

Нейрон, который имеет весовой вектор, самый близкий к X объявляется «победителем». Этот весовой вектор, обозначаемый Wc, становится основным в группе весовых векторов, которые лежат в пределах расстояния D от Wc.

Следует отметить, что иногда слишком часто «побеждающие» нейроны принудительно исключаются из рассмотрения, чтобы «уравнять права» всех нейронов слоя. Простейший способ реализовать это заключается в торможении (уменьшении значения синаптической связи) только что выигравшего нейрона.

4. Настраиваются «нейрон – победитель» и его соседи.

Для всех весовых векторов в пределах расстояния D от Wc изменяются значения синаптических связей

$$w_{i,j}(t+1) = w_{i,j}(t) + \nu(t) \cdot (x_i - w_{i,j}(t)),$$

где t – номер цикла обучения;

$\nu(t)$ – шаг обучения, уменьшающийся с течением времени.

5. Повторяются шаги со 2 по 4 для каждого входного вектора.

После обучения кластеризация выполняется посредством подачи на вход сети испытуемого вектора. Нейрон со значением, равным единице, является индикатором кластера.

Для определения выходных значений слоя Кохоненна используется правило «победитель получает все». Согласно ему для данного входного вектора только один нейрон слоя Кохоненна выдает единицу, все остальные – ноль. Расчет значения выхода каждого нейрона выполняется в соответствии с формулой (4.1). Нейрон с максимальным значением S объявляется «победителем», и его выход устанавливается равным единице.

В процессе обучения ИНС значения D и ν постепенно уменьшаются. В начале обучения коэффициент ν рекомендуется устанавливать приблизительно равным 1 и уменьшать в процессе обучения до 0, в то время как D может в начале обучения равняться максимальному расстоянию между X и весовыми векторами, а в конце обучения стать настолько маленьким, что будет обучаться только один нейрон.

Обучающий алгоритм настраивает синаптические связи в окрестности «победившего» нейрона таким образом, чтобы они были похожими на входной вектор. Сначала в эту окрестность могут входить все нейроны слоя Кохоненна, однако в процессе обучения сети такие зоны соседства будут сужаться, образуя топологические области (карты), которые соответствуют определенным входным векторам. Так, на рис. 4.13 показаны зоны соседства (C_j) в различные моменты времени для нейрона j, которые уменьшаются в процессе обучения сети. В результате фактически возникает группировка входных векторов в кластеры, каждый из которых ассоциируется с

определенным нейроном. Подобный принцип функционирования ИНС известен как самоорганизующиеся карты Кохоненна [5].

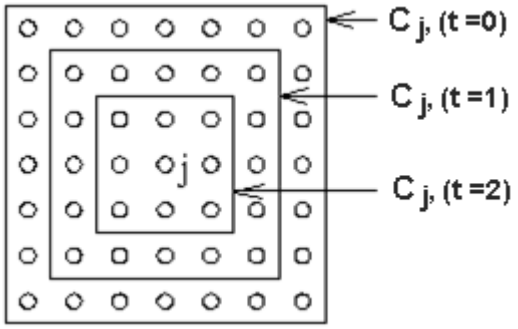


Рис. 4.13. Зоны топологического соседства на различных циклах обучения ИНС

Самоорганизующиеся карты представляют собой отображение многомерного распределения точек на двумерную решетку с регулярным соседством меду узлами. При этом близким узлам на карте отвечают близкие вектора в исходном многомерном пространстве, то есть сохраняется не только структура разбиения точек на кластеры, но и отношение топологической близости между ними. Отметим, что карты Кохоненна дают высокую степень наглядности при изучении распределения экспериментальных данных. Неоднородности в «раскраске» карты могут отвечать различным режимам поведения установки или прибора, например служить указанием на нежелательные соотношения параметров при эксплуатации.

Приведем некоторые из простых правил, позволяющих повысить эффективность работы описанного алгоритма.

1. Инициализация весовых коэффициентов случайными значениями может привести к тому, что различные классы, которым соответствуют плотно распределенные входные вектора, сольются или, наоборот, раздробятся на дополнительные подклассы в случае близких векторов одного и того же класса. Для избежания такой ситуации используется метод выпуклой комбинации. Суть его сводится к тому, что входные нормализованные вектора подвергаются преобразованию

$$x_i = \nu(t) \cdot x_i + (1 - \nu(t)) \frac{1}{\sqrt{n}}.$$

2. Согласно рекомендации Кохоненна для получения хорошей статистической точности число обучающих циклов должно быть по крайней мере в 500 раз больше числа выходных нейронов.

В общей постановке задачи кластеризации принято выделять два основных случая:

- с изначально известным числом кластеров (классов);

- с заранее неопределенным числом кластеров.

Применение ИНС при решении задач кластеризации для первого случая было рассмотрено ранее, поэтому для полноты изложения кратко отметим особенности использования нейронных сетей для второй ситуации.

При решении задач кластеризации с неизвестным числом кластеров критерием принадлежности конкретному классу служит заданное заранее расстояние (мера сходства) между сигналами. Мерой сходства/различия для бинарных сигналов может служить расстояние Хемминга (число отличающихся битов в двух бинарных векторах). Для других типов сигналов - Евклидово расстояние. В свою очередь, процесс определения кластеров заключается в следующем.

Путем попарного сравнения выбираются два наиболее различающихся входных сигнала. Далее считают, что первый сигнал принадлежит первому кластеру, другой - второму кластеру. Строится обучающая выборка традиционного вида, содержащая два примера - пары (вход, известный выход). По данной выборке проводится обучение сети алгоритмом обратного распространения.

Случайным образом выбирается один из оставшихся входных сигналов и вводится в нейронную сеть. Если среднеквадратичное отклонение текущего выходного сигнала от одного из известных выходных сигналов обучающей выборки меньше заданного порогового значения, то входной сигнал считается принадлежащим кластеру

соответствующего примера. В противном случае считается, что входной сигнал принадлежит новому кластеру. Строится обучающая выборка из трех примеров и по ней выполняется обучение.

Аналогичные действия повторяются до тех пор, пока не закончатся входные сигналы.

Затем для каждого сформированного на первом этапе кластера строится «центральный» или «эталонный» образ, каждый элемент которого есть среднее арифметическое соответствующих элементов всех примеров данного кластера.

Далее с помощью меры Хемминга или Евклида определяется расстояние между парами «центральных» образов. Если расстояние меньше некоторого порогового значения, то два кластера объединяются в один. Соответствующим образом формируется новая обучающая выборка, и по ней проводится обучение нейронной сети. Эту операцию повторяют до тех пор, пока не останется ни одной пары кластеров, расстояние между «центральными» образами которых меньше порогового значения.

В качестве примера применения самоорганизующихся карт рассмотрим решение задачи кластеризации типа цветков ириса на основе их характеристик. Эта задача, известная как «ирисы Фишера», относится к тестовым и входит в группу задач для исследования возможностей нейросетевого моделирования (проект ELENA - Enhanced Learning for Evolutive Neural Architectures).

Ниже представлен фрагмент таблицы исходных данных (табл. 4.7) для кластеризации, в качестве средства программной реализации которой будет использоваться система Deductor.

Характеристики цветков соответствуют названиям столбцов таблицы. Задача состоит в том, чтобы определить по различным параметрам цветка его класс. Предполагается, что цветы одного класса имеют схожие значения параметров, следовательно, они должны располагаться в одном кластере.

Процесс формирования карты этих кластеров заключается в выполнении следующей последовательности действий.

На первом шаге настраивается назначение столбцов таблицы, при этом входными данными считаются «Длина чашелистика», «Ширина чашелистика», «Длина лепестка», «Ширина лепестка», выходными – «Класс цветка».

Таблица 4.7

Фрагмент исходных данных кластеризации

Длина чашелистика	**Ширина чашелистика**	**Длина лепестка**	**Ширина лепестка**	**Класс цветка**
50	33	14	2	Setosa
64	28	56	22	Virginica
65	28	46	15	Versicolo
67	31	56	24	Virginica
63	28	51	15	Virginica
46	34	14	3	Setosa
62	22	45	15	Versicolo
59	32	48	18	Versicolo
…	…	…	…	…

На втором шаге необходимо настроить способ разделения исходного множества данных на тестовое и обучающее, а также число примеров в том и другом множестве.

Следующий шаг предлагает настроить параметры карты (количество ячеек по X и по Y, их форму) и параметры обучения (способ начальной инициализации, тип функции соседства, перемешивать ли строки обучающего множества и количество эпох, через которые необходимо перемешивание) (рис. 4.14).

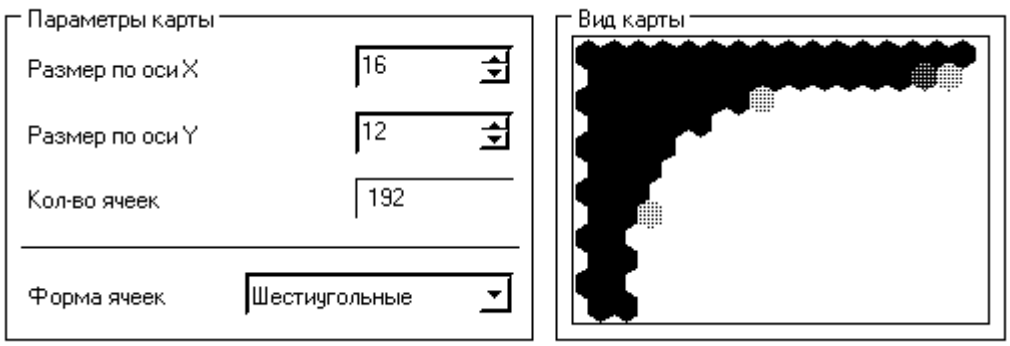


Рис. 4.14. Настройка карты отображение кластеров

На четвертом шаге настраиваются параметры остановки обучения (рис. 4.15).

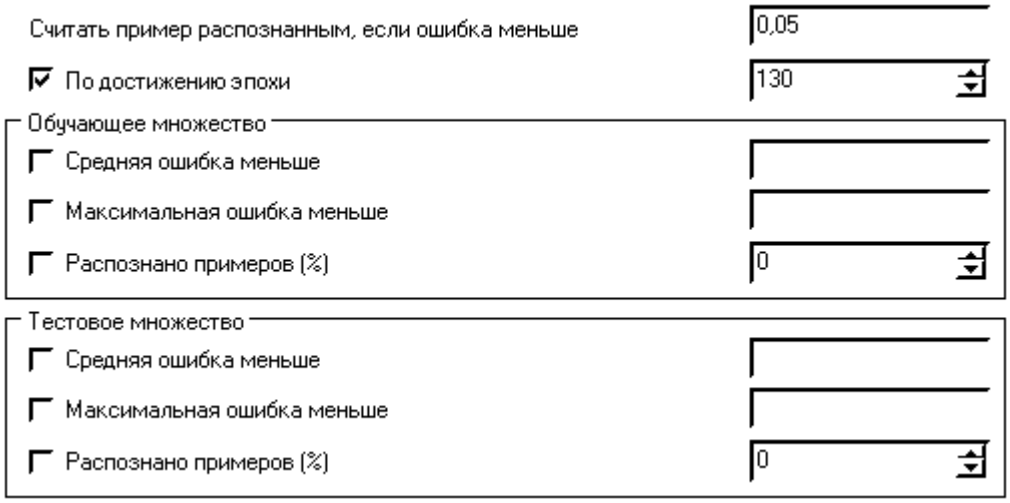


Рис. 4.15. Настройка условий прекращения обучения

На пятом шаге настраиваются остальные параметры обучения (рис. 4.16) – способ начальной инициализации, тип функции соседства и также параметры кластеризации – автоматическое определение числа кластеров с соответствующим уровнем значимости либо фиксированное число кластеров. Предоставляется возможность настроить интервалы обучения. Каждый интервал задается числом эпох, радиусом обучения и скоростью обучения. Число кластеров устанавливается равным трем.

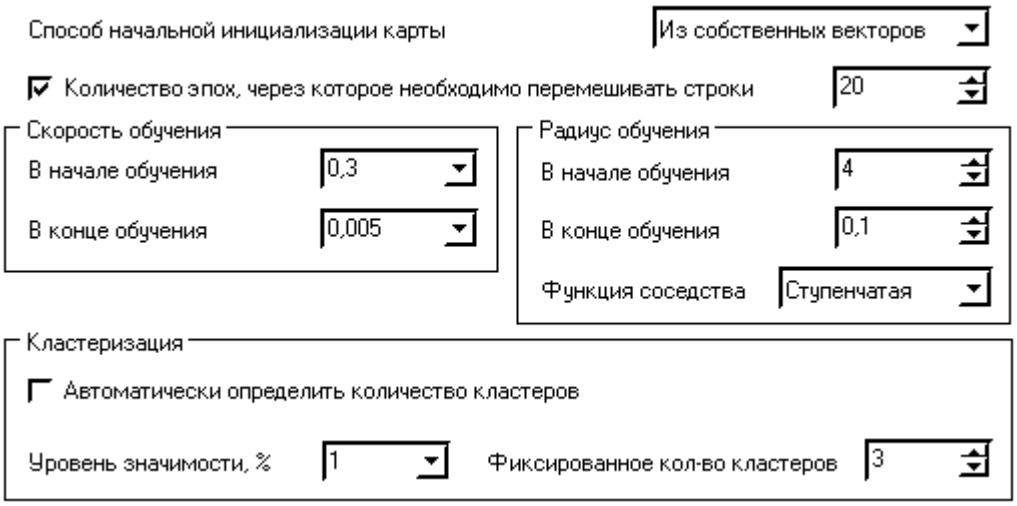


Рис. 4.16. Настройка параметров обучения

На последнем шаге запускается сам процесс обучения (рис. 4.17). При этом во время обучения могут быть просмотрены число распознанных примеров и текущие значения ошибок.

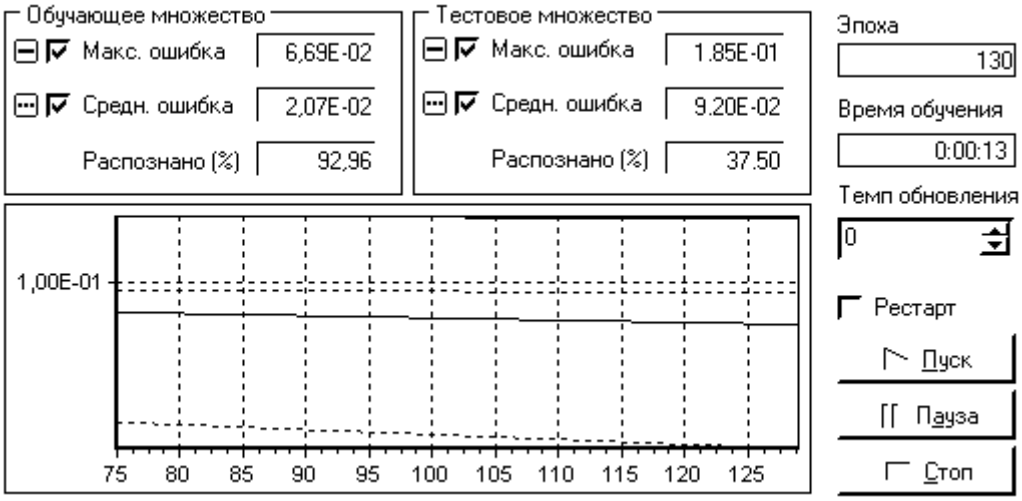


Рис. 4.17. Визуализация процесса обучения нейросети

Далее после настройки параметров отображения могут быть просмотрены результаты кластеризации данных (рис. 4.18).

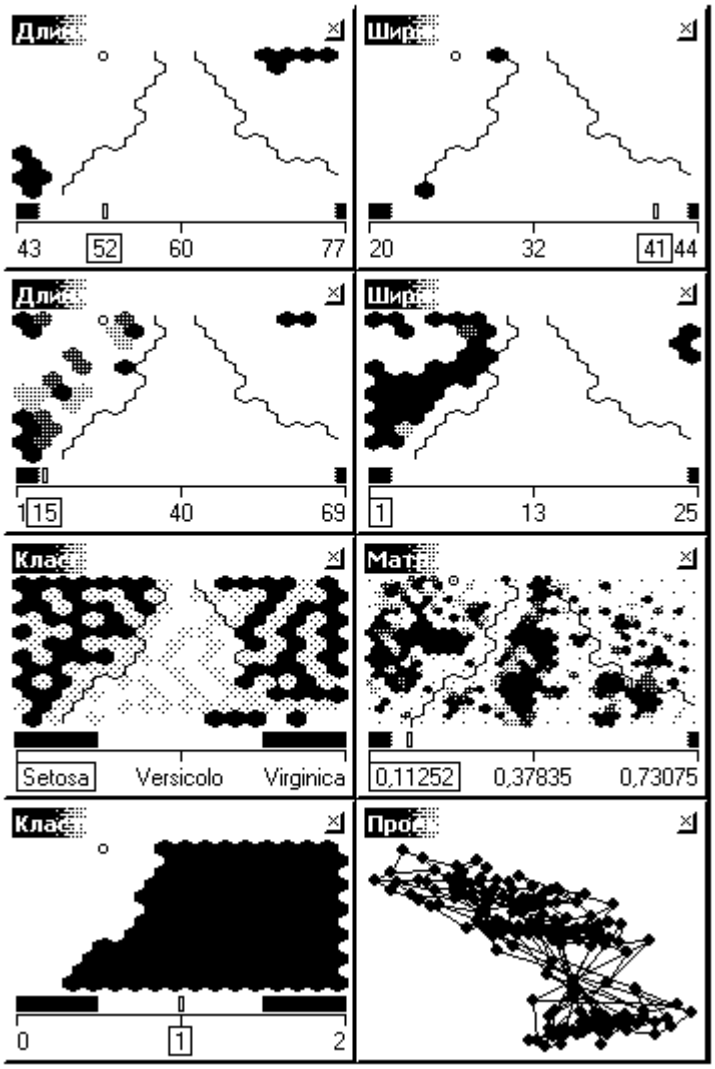


Рис. 4.18. Результаты кластеризации

Как следует из рис. 4.18, анализ результатов может быть выполнен не только в целом по конкретному цветку, но и по каждой из его характеристик. Качество кластеризации можно оценить, просмотрев карту «КЛАСС ЦВЕТКА». На ней видно, что большинство цветов были классифицированы правильно. Заметим, что все цветы класса Setosa попали в один кластер. Это говорит о значительном отличии параметров цветов этого класса от других. Явное различие наблюдается по длине и ширине лепестка. То, что часть примеров Virginica попала в класс Versicolo и наоборот, говорит о меньшем различии этих классов. На картах, в отличие от Setosa, не видны резкие отличия параметров цветов этих двух классов. Этим как раз и объясняется «проникновение» некоторой части примеров в другой кластер.

Таким образом, применение самоорганизующихся карт позволило визуально представить начальное многомерное (четырехмерное) пространство исходных данных в двумерной форме, удобной для восприятия и анализа. Информация, описывающая цветки, была сгруппирована в три кластера, соответствующие типам цветков «Setosa», «Versicolo» и «Virginia». Вследствие этого в дальнейшем может быть достаточно точно определен тип цветка на основе его характеристик, отсутствующих в данных кластеризации.

4.5. Развитие моделей нейросетей и методов их обучения

Среди моделей нейронных сетей существуют такие, которые по способу обучения нельзя отнести к традиционным (обучающихся с учителем или без него). В таких сетях весовые коэффициенты синапсов рассчитываются только однажды перед началом функционирования сети на основе информации об обрабатываемых данных. Из сетей с подобной логикой работы наиболее известна сеть Хопфилда, которая обычно используется для организации ассоциативной памяти.

Отметим, что ассоциативность – важная особенность человеческой памяти, то есть для порождения некоторого воспоминания человеку необходимо провести ассоциацию связанного с ним события или

объекта. Так, несколько музыкальных тактов могут вызвать целую гамму чувственных воспоминаний, включая пейзажи, звуки и запахи. Напротив, обычная компьютерная память является адресуемой, то есть информация извлекается по предъявляемому адресу.

Структурная схема сети Хопфилда приведена на рис. 4.19. Она состоит из единственного слоя нейронов, число которых является одновременно числом входов и выходов сети. Каждый нейрон связан синапсами со всеми остальными нейронами, а также имеет один входной синапс, через который осуществляется ввод сигнала. Выходные сигналы, как обычно, образуются на аксонах.

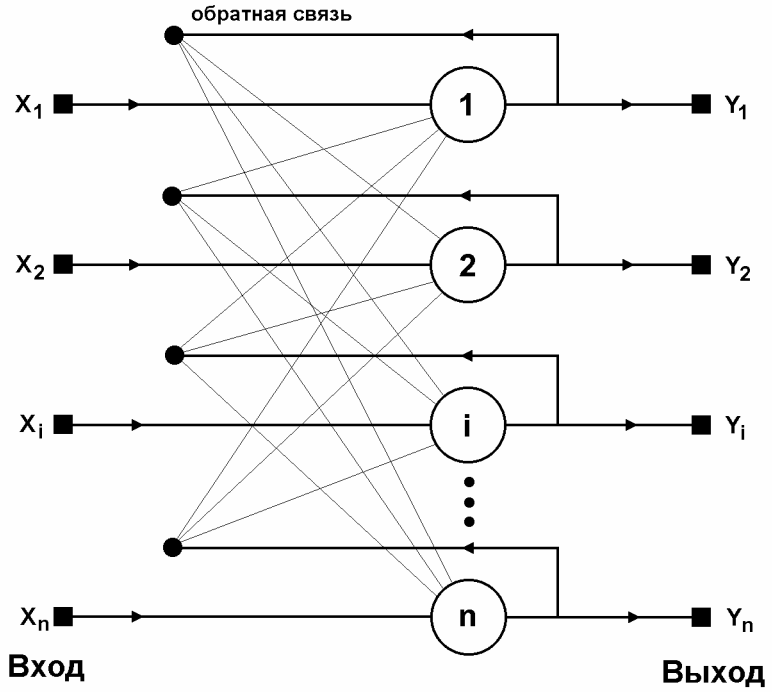


Рис. 4.19. Структура нейронной сети Хопфилда

Как видно из рис. 4.19 сеть Хопфилда в отличие от ранее рассмотренных моделей ИНС имеет обратные связи. В результате отклик таких сетей является динамическим, то есть после предъявления нового входа вычисляется выход и, передаваясь по обратной связи, модифицирует вход. Этот процесс повторяется многократно. Все это является причиной возможной неустойчивости сети , когда выход вместо стабилизации в постоянное значение непредсказуемо изменяется. Проблема устойчивости явилась главным препятствием, не дающим полноценно использовать этот тип ИНС, однако в

последствии была получена теорема, описавшая подмножество сетей с обратными связями, выходы которых в конце концов достигают устойчивого состояния.

Задача, решаемая данной сетью в качестве ассоциативной памяти, как правило, формулируется следующим образом [5, 13]. Известен некоторый набор двоичных сигналов (изображений, звуковых оцифровок, прочих данных, описывающих некие объекты или характеристики процессов), которые считаются образцовыми. Сеть должна уметь из произвольного неидеального сигнала, поданного на ее вход, выделить («вспомнить» по частичной информации) соответствующий образец (если такой есть) или «дать заключение» о том, что входные данные не соответствуют ни одному из образцов. В общем случае любой сигнал может быть описан вектором $X = \{x_i: i = 1...n\}$, n – число нейронов в сети и размерность входных и выходных векторов. Каждый элемент x_i равен либо +1, либо -1. Обозначим вектор, описывающий k-й образец, через X^k, а его компоненты соответственно – x_i^k, $k = 1...p$, p – число образцов. Когда сеть распознáет (или «вспомнит») какой-либо образец на основе предъявленных ей данных, ее выходы будут содержать именно его, то есть $Y = X^k$, где Y – вектор выходных значений сети. В противном случае выходной вектор не совпадет ни с одним из образцов.

Если, например, сигналы представляют собой некие изображения, то, отобразив в графическом виде данные с выхода сети, можно будет увидеть картинку, полностью совпадающую с одной из образцовых (в случае успеха), или же «вольную импровизацию» сети (в случае неудачи).

Пример постановки задачи распознавания с применением сети Хопфилда приведен на рис. 4.20. Здесь имеется пять образов, которые представляют собой изображения букв размером 5 на 6 точек (рис. 4.20 а). Требуется определить образ на основе подаваемых на вход сети зашумленных изображений (рис. 4.20 б).

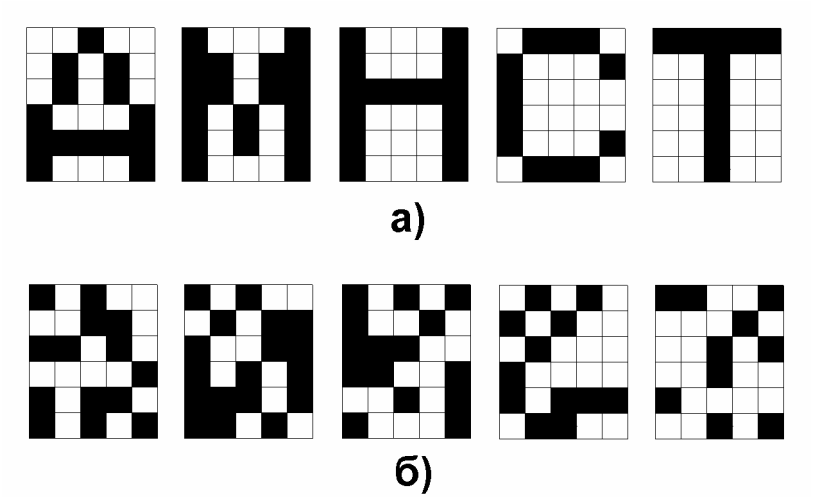


Рис. 4.20. Пример задачи распознавания:
а – эталонные образы; б – зашумленные варианты эталонных образов

Подобно другим моделям ИНС, процесс настройки сети Хопфилда включает два этапа: инициализацию и обучение.

На стадии инициализации сети весовые коэффициенты синапсов устанавливаются следующим образом:

$$w_{i,j} = \begin{cases} \sum_{k=1}^{p} x_i^k x_j^k, i \neq j; \\ 0, i = j. \end{cases}$$

Здесь i, j – индексы соответственно предсинаптического и постсинаптического нейронов; x_i^k, x_j^k – i-й и j-й элементы вектора k-го образца.

После задания весов сеть может быть использована для получения запомненного выходного вектора поданному входному вектору, который может быть неправильным или неполным. Но перед этим должна быть настроена по следующему алгоритму.

1. На входы сети подается неизвестный сигнал. Фактически его ввод осуществляется непосредственной установкой значений аксонов: $y_i(0) = x_i$, $i = 1...n$.

2. Рассчитывается новое состояние нейронов (4.12) и новые значения аксонов (4.13)

$$S_j(t+1) = \sum_{i=1}^{n} w_{ij} y_i(t). \quad (4.12)$$

$$y_j(t+1) = f(S_j(t+1)), \quad (4.13)$$

где f – активационная функция в виде скачка, представленная на рис. 4.21.

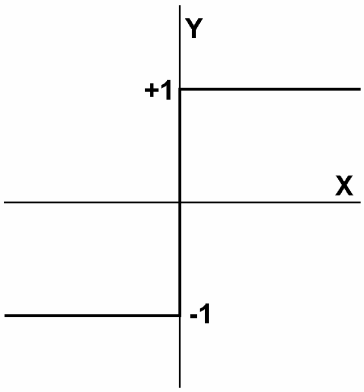


Рис. 4.21. Активационная функция для сети Хопфилда

3. Проверяется, изменились ли выходные значения аксонов за последнюю итерацию. Если да – переход к пункту 2, иначе (если выходы застабилизировались) – окончание настройки. При этом выходной вектор представляет собой образец, наилучшим образом сочетающийся с входными данными.

Важным вопросом, связанным с нейросетью Хопфилда, является количество запоминаемой информации. Несмотря на то, что ИНС с бинарным выходным слоем может иметь 2^n состояний для сети Хопфилда, число запоминаемых образов (емкость ИНС) ограничивается значением 0,15*n. Поэтому, если при решении задачи нет потребности в получении конкретного образа, а достаточно определить его номер, для реализации ассоциативной памяти эффективно может быть использована модификация сети Хопфилда, предложенная Хеммингом.

Совершенствование возможностей нейросетей связано не только с разработкой их новых моделей, но и с поиском и исследованием других методов обучения. Рассмотренные ранее методы обучения принято относить к детерминированным, когда коррекция весов выполняется на основе известных их текущих значений, а также величин входов и разницы между фактическими и желаемыми значениями выходов сети. Однако при обучении ИНС величины синаптических связей могут корректироваться и псевдослучайно, сохраняя те изменения, которые ведут к улучшению функционирования сети.

Методы, в которых заложен такой механизм обучения, известны как стохастические. Часто подобные методы используют идею так называемого Больцмановского обучения, в основе которого лежит имитация физической аналогии отжига металла.

В металле, нагретом до температуры, превышающей его точку плавления, атомы находятся в сильном беспорядочном движении. Как и во всех физических системах, атомы стремятся к состоянию минимума энергии (единому кристаллу в данном случае), но при высоких температурах энергия атомных движений препятствует этому. В процессе постепенного охлаждения металла возникают все более низкоэнергетические состояния, пока в конце концов не будет достигнуто самое низкое из возможных состояний, соответствующее глобальному минимуму энергии. В процессе отжига распределение энергетических уровней описывается следующим соотношением:

$$P(c) = e^{-\frac{c}{kT}}, \tag{4.14}$$

где $P(c)$ – вероятность того, что система находится в состоянии с энергией c; k – постоянная Больцмана; T – температура по шкале Кельвина.

При высоких температурах $P(c)$ приближается к единице для всех энергетических состояний. Таким образом, высокоэнергетическое состояние почти столь же вероятно, как и низкоэнергетическое. По мере уменьшения температуры вероятность высокоэнергетических состояний уменьшается по сравнению с низкоэнергетическими. При приближении температуры к нулю становится весьма маловероятным, чтобы система находилась в высокоэнергетическом состоянии.

В свою очередь, алгоритм Больцмановского обучения ИНС заключается в следующем.

1. Определяется переменная T, представляющая искусственную температуру, ей присваивается большое начальное значение.

2. Сети предъявляются множество входов, на основе которых вычисляются выходы и целевая функция. Последняя представляет собой среднюю квадратичную ошибку между полученным и

желаемым выходным вектором из обучающего множества, переменными здесь являются веса сети.

3. Текущее значение веса случайно изменяется, а выходы сети и целевая функция пересчитываются.

4. Если целевая функция уменьшилась (улучшилась), то изменение веса сохраняется. Если же изменение веса приводит к увеличению целевой функции, то вероятность сохранения этого изменения вычисляется с помощью распределения Больцмана (4.14). Выбирается случайное число r из равномерного распределения от нуля до единицы. Если $P(c)$ больше, чем r, то изменение сохраняется, в противном случае величина веса возвращается к предыдущему значению.

Это позволяет системе делать случайный шаг в направлении выхода из локальных минимумов.

Для завершения больцмановского обучения повторяют шаги 3, 4 для каждого из весов сети, постепенно уменьшая температуру T, пока не будет достигнуто допустимо низкое значение целевой функции. В этот момент предъявляется другой входной вектор и процесс обучения повторяется. Сеть обучается на всех векторах обучающего множества, с возможным повторением, пока целевая функция не станет допустимой для всех них.

Однако экспериментальное исследование возможностей Больцмановского обучения показало, что этот процесс имеет большую временную трудоемкость. Поэтому для определения случайного изменения синаптических весов могут быть использованы другие способы, например обучение Коши, метод монте-Карло, а также комбинированные алгоритмы, сочетающие детерминированное и стохастическое обучение.

В настоящее время искусственные нейронные сети как направление искусственного интеллекта не ограничивается рассмотренными типами ИНС, а развивается в направлении усложнения конфигураций нейросетей, решаемых ими задач, а также совершенствования методов обучения. Так, большой интерес у исследователей вызывает изучение свойств, которые появляются при объединении различных типов ИНС в единые нейросетевые ансамбли. Примером здесь может служить нейросеть встречного распространения, включающая слои Гроссберга и Кохоненна и позволяющая выполнять сжатие

информации. Кроме этого, особое место занимают исследования, направленные на моделирование нейрофизиологической деятельности человека (двунаправленная ассоциативная память, когнитрон), виртуальной реальности (оптические нейронные сети) в сочетании с новыми подходами к обучению, такими как генетические алгоритмы.

Контрольные вопросы

1. Что общего и каковы отличия в функционировании нейрокомпьютера и компьютера с архитектурой фон Неймана?
2. При решении каких задач применение ИНС наиболее эффективно?
3. Какова модель искусственного нейрона и какие существуют основные активационные функции?
4. Что такое персептрон?
5. Каковы особенности алгоритмов обучения нейронной сети?
6. В чем заключаются принципы работы алгоритма обучения без учителя?
7. Какие функции не могут быть реализованы ИНС?
8. Каковы особенности функционирования многослойной ИНС?
9. В чем состоит алгоритм обучения многослойной ИНС?
10. Какие существуют проблемы и способы их преодоления при обучении многослойной ИНС?
11. Что такое самоорганизующиеся ИНС?
12. В чем состоит алгоритм обучения без учителя, предложенный Кохоненном?
13. Каковы особенности и принципы функционирования ИНС Хопфилда?
14. Что такое стохастические методы обучения?

5. МЕТОДЫ ЭВОЛЮЦИОННОГО МОДЕЛИРОВАНИЯ

Эволюционное моделирование (ЭМ) представляет собой направление в искусственном интеллекте, предназначенное для воспроизведения естественных эволюционных процессов с помощью компьютерных программ. При этом основу эволюционного моделирования образуют такие основополагающие принципы развития живой природы, как наследование и улучшение в поколениях полезных свойств, изменчивость, а также естественный отбор.

Алгоритмическую основу ЭМ составляют эволюционные алгоритмы (ЭА), включающие генетические алгоритмы, генетическое программирование, эволюционные стратегии и эволюционное программирование [2].

5.1. Генетические алгоритмы

Генетический алгоритм (ГА) является наиболее известным среди других эволюционных алгоритмов. Концепции ГА были разработаны Дж. Холандом и теоретически обоснованы в его работе «Адаптация в природных и искусственных системах».

С практической точки зрения генетические алгоритмы представляют собой группу методов адаптивного поиска и многопараметрической оптимизации, копирующие механизмы естественной биологической эволюции и основанные на использовании в процессе поиска сразу нескольких, закодированных соответствующим образом точек, которые образуют развивающуюся по заданным случайным законам популяцию.

Как известно, вся информация о признаках и свойствах биологической особи и человека в частности (цвет глаз, наследственные болезни, тип волос и т.д.) кодируется в виде особого генетического

кода внутри хромосомы. Аналогично и в ГА вся информация о решаемой задаче хранится закодированной и представляется в виде хромосомы.

Для уточнения ряда биологических понятий применительно к генетическому алгоритму приведем их определения.

Хромосома (особь, индивид, строка, решение) – структура данных, содержащая закодированные в виде генов параметры задачи и описывающая ее решение в виде точки пространства поиска.

Ген (признак) – элемент, из которых состоит хромосома, как правило, соответствует закодированному значению одного параметра задачи.

Аллель – элементарный участок хромосомы, образующий ген.

Математически ГА можно рассматривать как метод стохастической оптимизации для задач дискретной оптимизации вида [9]: минимизировать *f(x)* при условии, что $x \in \Omega = \{0,1\}^n$,

где $f : \Omega \to R$ представляет целевую функцию (ЦФ);

$x : \Omega \to R$ - n – мерный двоичный вектор из дискретного множества Ω, называемый хромосомой длины *n;*

множество $\Omega = \{0,1\}^n$ представляет собой множество вершин n – мерного гиперкуба с ребром равным 1;

$R = (-\infty; +\infty)$ - множество действительных чисел.

С позиций ЭА целевая функция, моделирующая собой условия для адаптации биологических особей во внешней среде, называется fitness-функцией. Считается, что чем выше значение fitness-функции имеет хромосома, тем лучше решение, которое она представляет, удовлетворяет цели.

Генетические алгоритмы в процессе своей работы используют два разделенных пространства: пространство поиска и пространство решений.

Пространство поиска представляет собой область двоичных наборов закодированных решений задачи, область решений – множество конкретных решений.

На рис. 5.1 представлено отображение параметров задачи в их хромосомное представление в ГА.

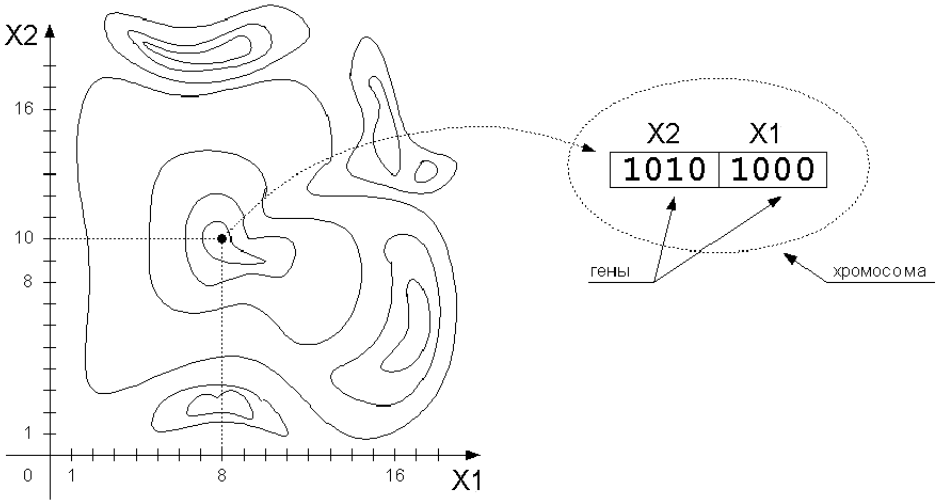


Рис. 5.1. Бинарное кодирование параметров задачи в ГА

Для повышения эффективности работы ГА, кроме обычной двоичной кодировки, для кодового представления параметров может использоваться код Грея (первые 16 слов этого кода для $n = 4$ приведены в табл. 5.1).

Таблица 5.1

Сравнение двоичного кода и кода Грея

Целое	**Двоичный код**	**Код Грея**
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Отличительной характеристикой кода Грея является его высокая помехозащищенность, благодаря непрерывности бинарной

комбинации, изменение кодируемого числа на единицу соответствует изменению кодовой комбинации только в одном разряде.

На рис. 5.2 приведен пример кодирования параметров задачи для случая, когда требуется оптимизировать функцию трех переменных $f = f(x_1, x_2, x_3)$. Каждый из трех параметров x_1, x_2, x_3 кодируется в виде пятибитового гена $x^{(i)}$, а вектор $X = (x_1, x_2, x_3)$ соответственно в виде составной хромосомы *x* общей длиной 15 битов.

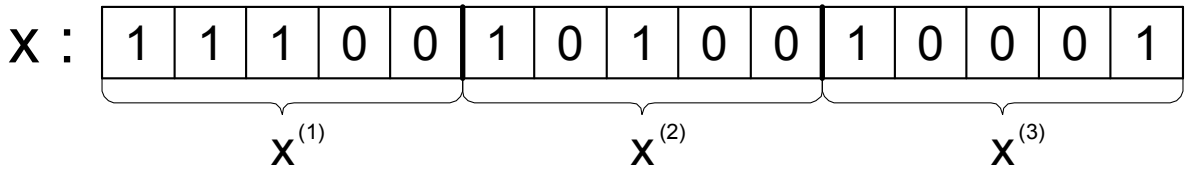


Рис. 5.2. Кодирование вектора параметров

Также наряду с бинарным кодированием решений в ГА может быть использовано вещественное, а также целочисленное десятичное представление хромосом, что бывает эффективно при решении комбинаторных задач оптимизации.

Процесс работы ГА состоит в постепенном изменении (эволюции) объектов популяции в соответствии с принципом, что каждая новая популяция есть комбинация из лучших элементов предыдущей, а также новых решений, необходимых для продолжения поиска. В результате эффективно используется информация, накопленная в процессе эволюционного моделирования.

Для вычисления последующих популяций к предыдущим применяются специальные генетические операторы: отбор или селекция (selection), кроссинговер или скрещивание (crossingover) и мутация (mutation) [1, 2, 9].

Оператор отбора. Используется для определения на основе значений fitness-функции хромосом-кандидатов в следующее поколение. Для реализации этого в ГА применяются различные схемы селекции, наиболее простая из которых – пропорциональный отбор, при котором число копий хромосомы в следующем поколении определяется как $N \cdot \overline{f}(x_i)$,

где N – общее число хромосом;

$\overline{f}(x_i)$ - относительная оптимальность хромосомы x_i в популяции.

$$\bar{f}(x_i) = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_i)}. \qquad (5.1)$$

Следствием такой схемы отбора является то, что в следующее поколение не попадут хромосомы с низкой оптимальностью (табл. 5.2, хромосома x_3), что неоправданно, так как они могут быть полезны при выполнении последующих генетических операторов. В связи с этим вместо пропорционального отбора обычно используется его стохастическая модификация, известная как метод «колесо рулетки», при котором хромосомы – кандидаты из *G(t)* - го поколения - выбираются для выживания в следующем *G(t+1)* – м поколении путем использования колеса рулетки. Каждая хромосома $x_i(t)$ в популяции представлена на колесе в виде сектора, ширина которого пропорциональна соответствующему значению fitness-функции. Таким образом, те хромосомы, которые имеют большую оптимальность, соответствуют большему сектору на колесе, а хромосомы с меньшей оптимальностью – наименьшему сектору колеса рулетки. Процедура отбора сводится к вращению колеса рулетки *N* раз и принятием в качестве кандидатов в следующем поколении тех хромосом $x_1, x_2, x_3, ..., x_N$, которые будут выделены по завершении вращения.

В результате добавления доли случайности при вращении рулетки все хромосомы получают шанс иметь свои копии в следующем поколении.

В качестве примера функционирования процедуры пропорционального отбора рассмотрим оператор селекции применительно к популяции из пяти хромосом $P_5 = \{x_1, x_2, ..., x_5\}$ с числом бит в кодировании *n=5*.

В качестве начальной популяции хромосом принимается *G(0) = {(10110), (11000), (11110), (01001), (00110)}*. Для каждой хромосомы x_i в популяции ее оптимальность оценивается с помощью $f(x_i)$. Соответствующая доля на колесе рулетки $\bar{f}(x_i)$, выделенная для *i*- й хромосомы x_i, рассчитывается в соответствии с (5.1). Список членов начальной популяции с соответствующими им значениями fitness-функции, а также число копий хромосомы x_i в следующем поколении

при использовании пропорционального отбора в табл. 5.2. Колесо рулетки для данного примера показано на рис. 5.3 (цифры на колесе указывают номера хромосом).

Таблица 5.2

Значения функции пригодности для членов начальной популяции

G(0)	**Хромосома** x_i	**Оптимальность** $f(x_i)$	**Относительная оптимальность** $\bar{f}(x_i)$	**Число копий при пропорциональном отборе**
x_1	10110	2,23	0,14	1
x_2	11000	7,27	0,47	2
x_3	11110	1,05	0,07	0
x_4	01001	3,35	0,21	1
x_5	00110	1,69	0,11	1

Таким образом, чтобы вычислить следующую популяцию хромосом, колесо рулетки крутят пять раз.

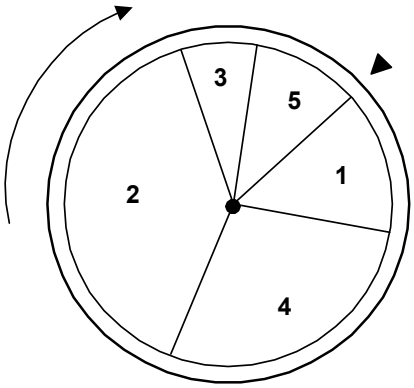


Рис. 5.3. Колесо рулетки при выполнении оператора отбора

Кроме этого, существуют и другие схемы селекции, наиболее известны из которых турнирный отбор, линейное ранжирование, равномерное ранжирование.

Перечисленные схемы отбора обладают одним общим недостатком - наилучшая хромосома в популяции может быть потеряна в любом поколении и нет гарантии, что результаты эволюции, достигнутые в ряде поколений, не будут утрачены.

Одним из способов преодоления данной проблемы является использование элитного отбора, который гарантирует сохранение лучшей хромосомы популяции в следующем поколении. Данный вид

отбора гарантирует асимптотическую сходимость решения к глобальному оптимуму, но скорость сходимости при этом может существенно различаться в зависимости от вида конкретной решаемой задачи.

Как отмечалось ранее, выбранные таким образом хромосомы являются только кандидатами для следующей популяции, в которую попадают либо их копии, либо модификации, вследствие применения операторов кроссинговера и мутации.

Оператор кроссинговера. Применяется с некоторой предварительно заданной вероятностью P_c к случайно выбранной паре хромосом из *G(t)*, прошедших отбор. При этом для этой пары определяется случайное число $k \in \{1,2,...,n-1\}$, называемое местом (сайтом) кроссинговера, и затем биты из двух выбранных хромосом меняются местами после *k* – го бита с вероятностью P_c. Этот процесс повторяется для остальных хромосом до тех пор, пока популяция *G(t+1)* не окажется заполненной. На рис.5.4 графически представлены действия оператора кроссинговера для двух 5 – ти битовых хромосом.

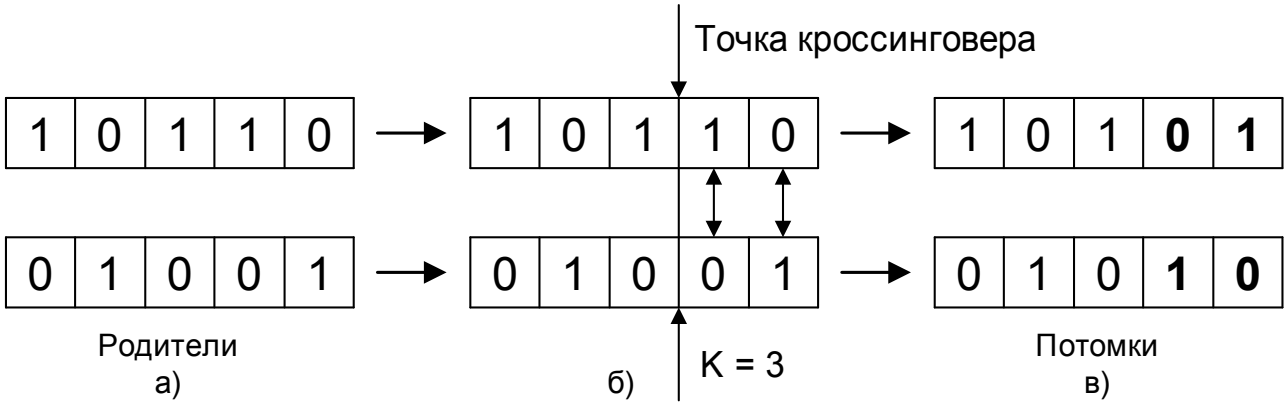


Рис. 5.4. Графическая интерпретация оператора кроссинговера

Как видно из рис. 5.4, две родительские хромосомы (а) вступают в кроссинговер для выбранного сайта *k=3* (б) так, что хромосомы потомков (в) отличаются от родительских хромосом значениями 4 – го и 5 – го битов.

Развитием одноточечного кроссинговера для бинарного кодирования являются многоточечный, на основе метода «золотого сечения», чисел Фибоначчи, дихотомии и др., отличающиеся друг от друга эффективностью синтеза «хороших» решений. Соответственно

существуют версии оператора кроссинговера, предназначенные и для вещественного представления хромосом (арифметический, геометрический, смешанный, нечеткий и др.)

Различные типы кроссинговера обладают общим положительным свойством – контроль баланса между дальнейшим использованием уже найденных хороших подобластей пространства поиска и исследованием новых областей. Это достигается путем сохранения общих блоков внутри хромосом - родителей и одновременного исследования новый областей в результате обмена фрагментами хромосом.

Совместное использование операторов отбора и кроссинговера приводит к тому, что области пространства, обладающие лучшей в среднем оптимальностью, содержат больше членов популяции, чем другие. Как следствие эволюция популяции направляется к областям, содержащим оптимум с большей вероятностью, чем другие.

Оператор мутации. После кроссинговера к хромосомам поколения *G(t+1)*– кандидатам применяется мутация, которая в общем случае может рассматриваться как процесс перехода между различными состояниями пространства поиска. Оператор мутации состоит в случайном изменении (на противоположное) значения каждого бита с вероятностью P_m, т.е. для каждого бита (гена) хромосомы его значение меняется с 0 на 1 или с 1 на 0 с вероятностью P_m.

Допустим в качестве примера, что $P_m = 0,1$ и хромосома $x = (11100)$ должна подвергнуться мутации. Для определения битов в гене, которые необходимо изменить выбирается независимое случайное число $\widetilde{P}_j$ для каждого бита хромосомы. Если окажется, что j – й бит следует изменить, иначе значение j – го бита сохраняется. Пусть для рассматриваемого вектора x были получены следующие случайные числа: $P_1 = 0,91$, $P_2 = 0,43$, $P_3 = 0,03$, $P_4 = 0,67$, $P_5 = 0,29$. Тогда результатом мутации является изменение *3* – го бита хромосомы (рис.5.5).

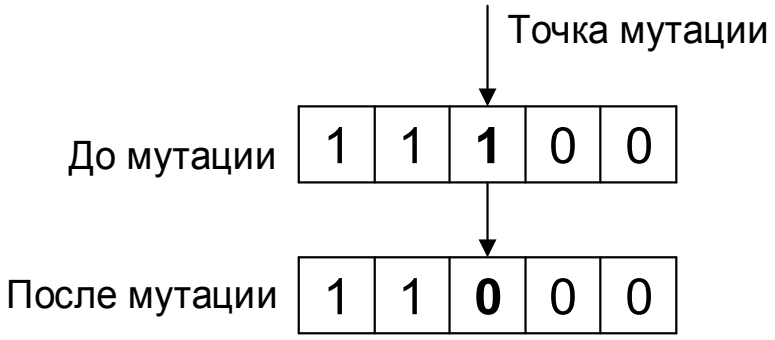


Рис. 5.5. Графическая интерпретация оператора мутации

Разновидностью мутации является инверсия, при которой участок хромосомы покидает свое место в цепи хромосомы и, развернувшись на 180 градусов, вновь занимает прежнее положение (рис.5.6).

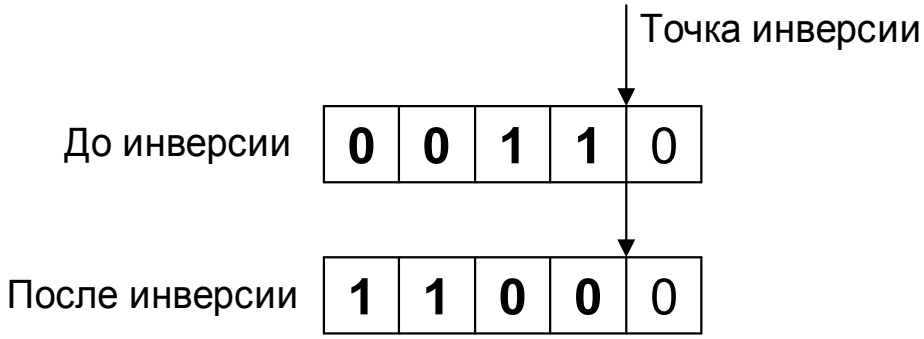


Рис. 5.6. Инверсия хромосомы

Эффективность ГА во многом зависит не только от внутренней структуры и модификаций его базовых операторов, но и от значений вероятностей их применения (P_c, P_m). Обычно в основе определения значений для операторов ГА лежат результаты исследований их применения в различных сочетаниях к ряду тестовых оптимизационных задач. При этом влияние операторов на качество результата работы ГА определяется как независимо друг от друга (*Pm* = 0, *Pc* = 0,1..1,0 и наоборот), так и совместно. Представленные на рис. 5.7 усредненные показатели для всех этапов тестирования иллюстрируют доминирование компромиссного варианта на основе комбинации направленного и случайного поиска.

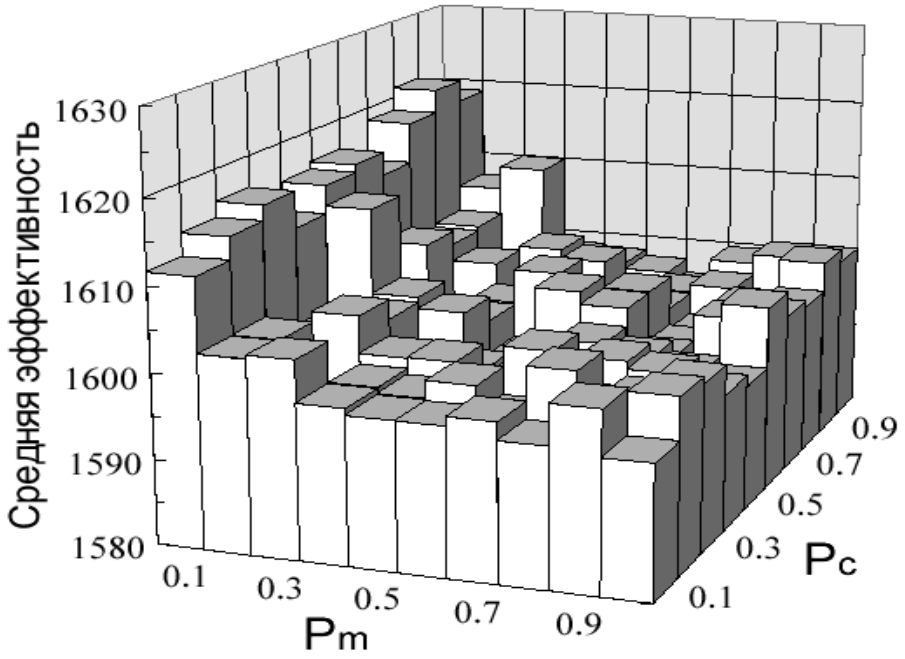


Рис. 5.7. Средние значения коэффициента эффективности ГА при варьировании вероятностями операторов

Также, исходя из эмпирических соображений было установлено, что наиболее приемлемыми значениями вероятности кроссинговера являются: $0{,}6 \le P_c \le 0{,}99$. Мутация инициирует разнообразие в популяции, позволяя просматривать больше точек в пространстве поиска и преодолевать таким образом локальные оптимумы в ходе поиска. Однако слишком частое применение мутации приводит к разрушению хромосом с высокой приспособленностью, что влияет на сходимость решения. Именно поэтому применение мутации обычно осуществляется с малой вероятностью, порядка $0{,}01 \le P_m \le 0{,}1$. Относительно оператора инверсии считается, что вероятность его применения должна быть очень низкой, как правило она составляет $P_{inv} = 0{,}001$.

В общем случае выбор значений для соотношений операторов ГА зависит от многих факторов: тип задачи, размер популяции, особенности генетических операторов и т.д., поэтому в большинстве работ при отсутствии предопределенных значений лучшая их комбинация определяется на основе имитационного моделирования (прогонов ГА) с различными вариациями и последующим анализом результатов.

Математическое обоснование работы ГА как метода адаптивного поиска связано с понятием шаблона или схемы, введенного Дж. Холландом. Результаты исследований и анализа функционирования ГА были сформулированы им в виде «фундаментальной теоремы о шаблонах» [2].

В соответствии с приведенным описанием ГА процесс его функционирования графически можно представить с помощью схемы, приведенной на рис. 5.8.

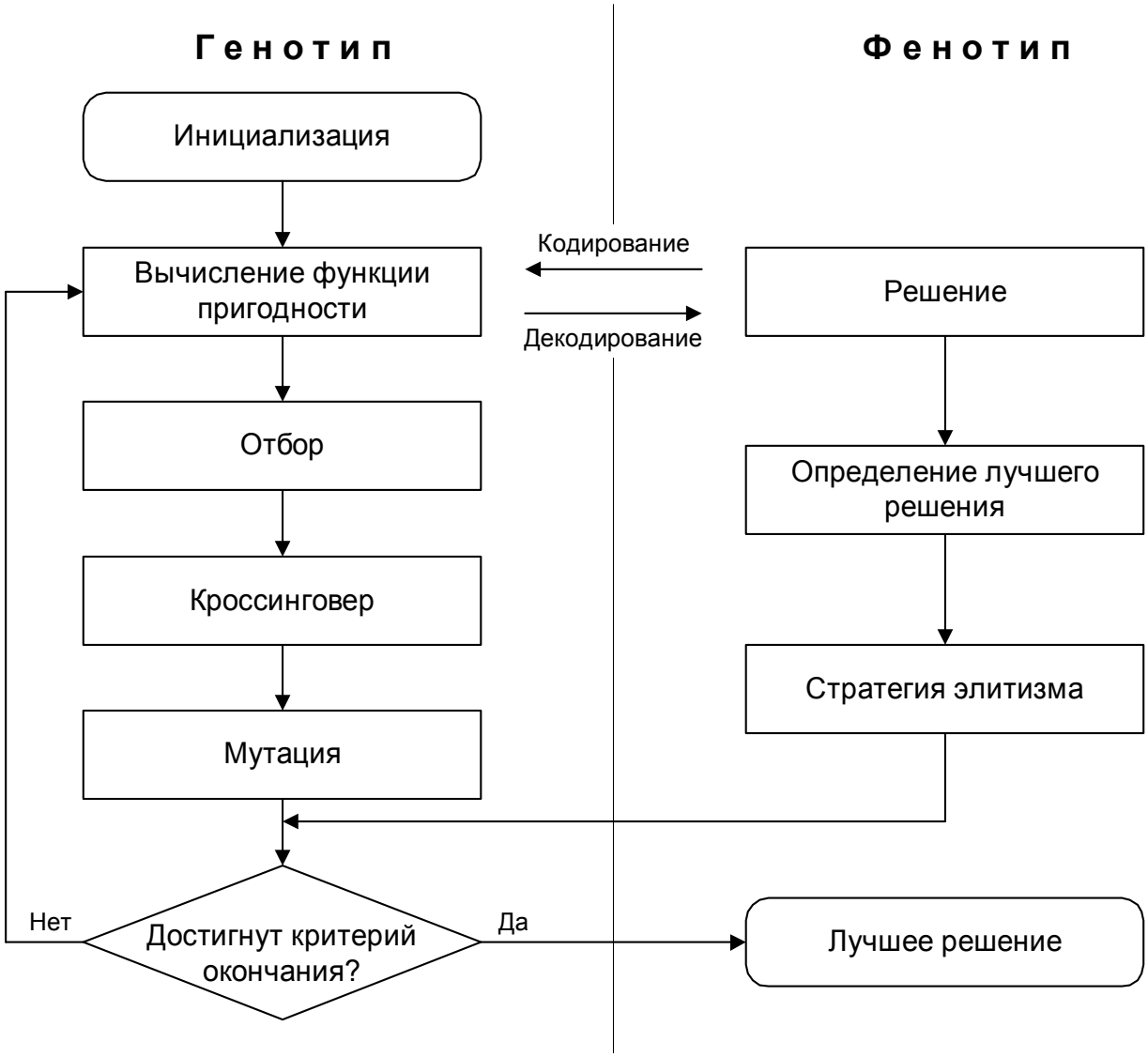


Рис. 5.8 Схема функционирования генетического алгоритма

Приведенная схема ГА описывает этапы его выполнения следующим образом:

- в соответствии с определенными ограничениями инициализируется исходная популяция потенциальных решений;

- каждая хромосома в популяции декодируется в необходимую форму фенотипа для последующей оценки, вычисляется значение целевой функции каждого решения с последующей трансформацией в значение fitness-функции каждой хромосомы генотипа;

- к членам популяции применяется оператор отбора, на основе которого создается новая популяция, причем с большей вероятностью воспроизводятся наиболее эффективные элементы;

- применяется оператор кроссинговера с заданной вероятностью для получения потомков;

- с определенной вероятностью выполняется оператор мутации к каждой из полученных на предыдущем шаге хромосом;

- при необходимости применяется стратегия элитизма – в новую популяцию добавляется лучшая хромосома предыдущего поколения;

- процесс останавливается, если удовлетворены условия останова эволюции, в противном случае повторяются этапы оценки и воспроизведения новой популяции.

Критерий остановки эволюции может быть определен произвольным образом (например, получение удовлетворительного решения, достижение определенного числа поколений, количество затраченного времени, низкая скорость сходимости и т.п.).

Рассмотрим применение генетического алгоритма для оптимизации функции вида $f = \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|})$, где -500 <= *Xi* <= 500, n = 10. Эта функция (рис. 5.9) является достаточно сложной для поиска экстремума с помощью традиционных методов оптимизации, так как имеет более 10 миллионов локальных оптимумов и один глобальный минимум со значением - 4189.83 в точке *Xi* = 420,967.

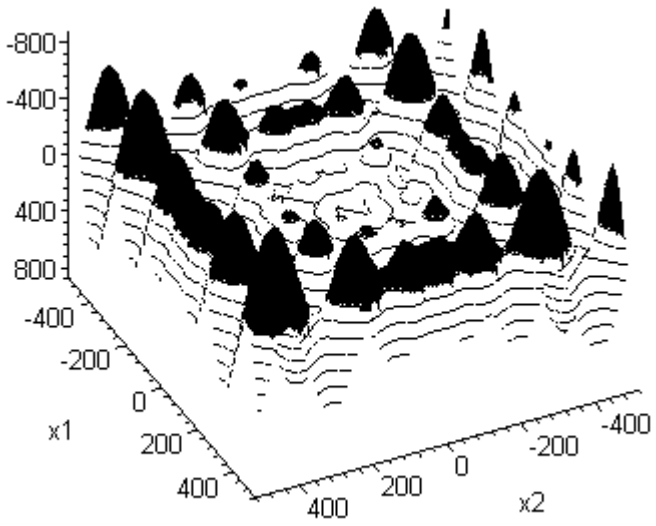


Рис. 5.9. График оптимизируемой функции (n = 2)

Процесс решения этой задачи будет заключаться в последовательном запуске ГА с различными значениями вероятностей применения генетических операторов. При наличии априорной информации о координатах оптимального решения может быть оценено качество работы ГА. Для каждого случая ГА запускался по пять раз. В качестве результата принималось минимальное значение функции, которое преобладало в большинстве случаев (табл. 5.3). Размер популяции устанавливался равным 100, число поколений – равным 300.

Таблица 5.3

Результаты эксперимента

Вероятность кросс-синговера	**Вероятность мутации**	**1**	**2**	**3**	**4**	**5**
0,9	0,1	-4185,76	-4189,35	-4188,81	**-4189,83**	-4177,84
0,8	0,2	**-4189,83**	-4189,27	**-4189,83**	**-4189,83**	**-4189,83**
0,7	0,3	-4189,02	-4189,39	-4136,50	-4185,14	-4188,67

Как следует из результатов эксперимента, минимальное значение целевой функции, найденное ГА, совпадает с глобальным оптимумом. На рис. 5.10 показана динамика исследования пространства решений на протяжении заданного числа поколений.

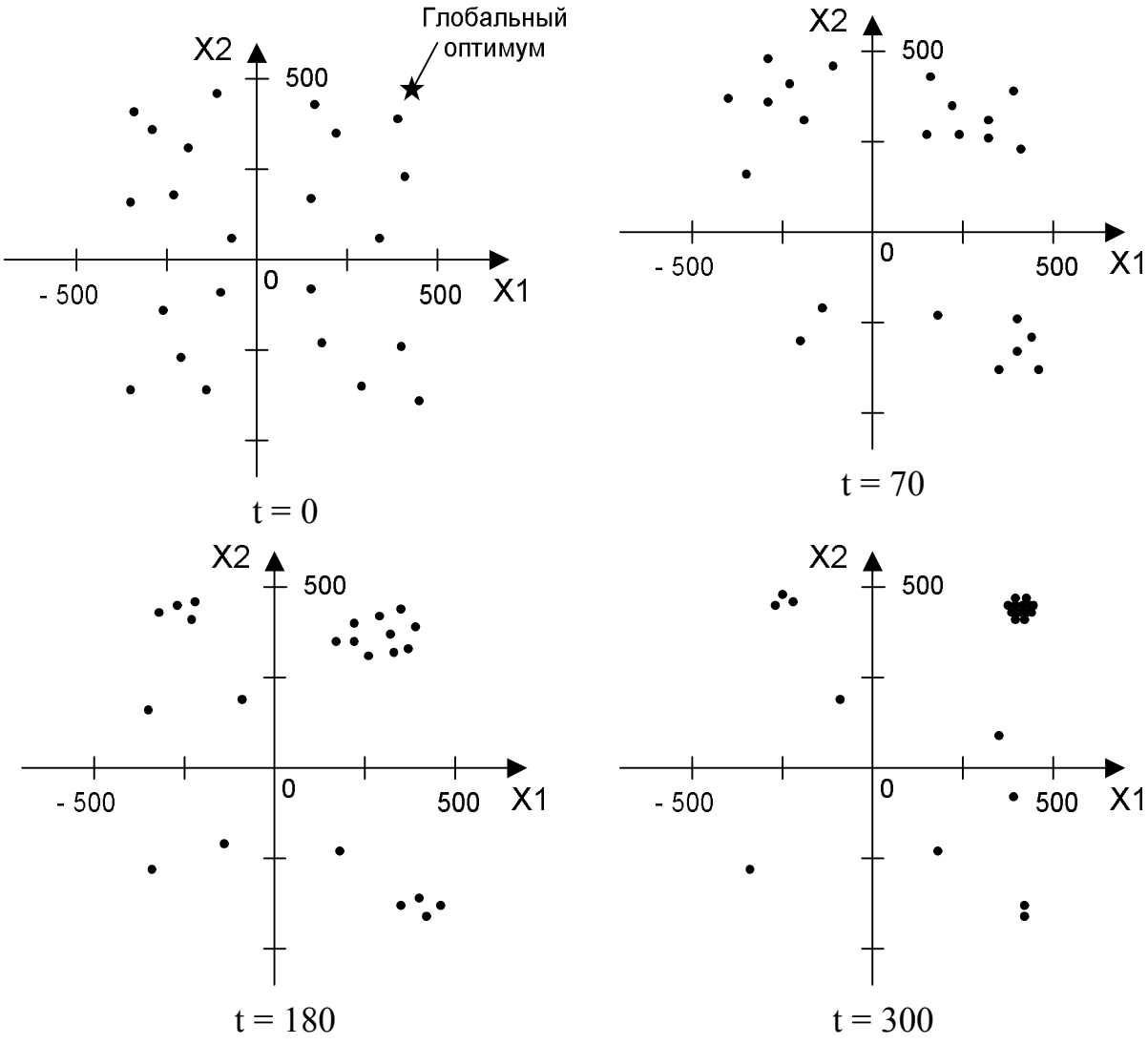


Рис. 5.10. Динамика исследования ГА пространства решений

В начальный момент времени (t = 0) популяция заполнена случайными решениями, распределенными почти по всему полю поиска. Далее под действием законом эволюционного моделирования происходит постепенное перемещение популяции в области с наилучшими (наименьшими) значениями целевой функции. При этом случайные изменения решений в популяции, вызываемые оператором мутации приводят к перемещению между областями пространства решений, а применение кроссинговера – к обследованию этих областей. К поколению t = 180 в популяции теряется разнообразие и начинает проявляться эффект насыщения, то есть среди хромосом можно выделить группу, которая из-за высокой по сравнению с другими решениями оптимальностью, тиражирует свои копии в следующие поколения.

На последних поколениях популяция полностью вырождается в «облачко» и в основном представляет собой копии самой лучшей на протяжении всей эволюции хромосомы, которая и определяет оптимальное решения задачи.

Эффективность применения ГА при оптимизации в многомерных пространствах объясняется прежде всего параллельным характером его работы, когда на каждом шаге синтезируется целая популяция возможных решений, причем на основе накопленной информации предыдущих поколений об оптимизируемой функции. Следствием этого является широкое исследование пространства решений и как результат - высокая вероятность нахождения глобального оптимума.

Дальнейшее совершенствование генетических алгоритмов связано с проведением исследований, направленных на развитие теории Холланда с целью создания новых генетических операторов, разработкой различных модификации ГА.

5.2. Генетическое программирование

Развитием генетического алгоритма для решения оптимизационных задач в пространстве компьютерных программ является генетическое программирование (ГП). В данном случае в качестве объектов, составляющих популяцию, выступают не двоичные хромосомы, а компьютерные программы, которые, будучи исполненными, представляют собой кандидатов на решение поставленной задачи.

Концепция ГП была предложена Дж. Коза в попытке создания средств автоматического программирования на основе эволюционных технологий. Предпосылкой этому послужила одна из главных проблем ИИ – обучение компьютера решению задачи без непосредственного программирования.

Особенностью ГП является то, что в нем не требуется кодировать параметры решения и все манипуляции с ними выполняются на уровне фенотипа. Для формирования начальной популяции, представляющей собой совокупность случайно сформированных

программ, используются заранее определенные множества функций и терминальных символов из предметной области задачи.

Множество функций может включать как арифметические (+, - , * и т.д.), логические операции (AND, OR, NOT), элементарные математические функции (sin, cos, exp, log и т.д.), так и условные, а также циклические операторы программирования.

Множество терминальных символов обычно включает переменные, а также числовые или логические константы.

Для примера, предположим, что необходимо реализовать функцию вида *F=(X* AND *Y)* OR *(*NOT *X* AND NOT *Y)*, тогда указанное множество операций можно определить в виде {*AND, OR, NOT*}, а множество терминальных символов - как {*X, Y*}. На рис. 5.11 приведено графическое изображение данного выражения в виде дерева – иерархического представления ГП. Здесь внутренние вершины имеют смысл названных функций, а внешние (оконечные) вершины представляют собой терминальные символы [9].

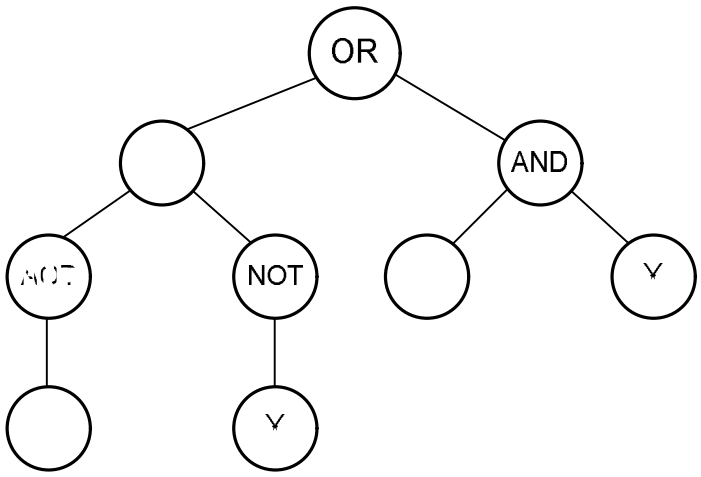


Рис. 5.11. Представление компьютерной программы в ГП

Таким образом, пространство решений в ГП можно рассматривать как множество деревьев с упорядоченными ветвями, внутренние вершины которых соответствуют функциям , а внешние - терминальным символам. Потребность в использовании подобного иерархического представления была связана с особенностями применения кроссинговера к хромосомам переменной длины, представление информации в которых подобно синтаксису языка LISP.

Для оценки качества работы сгенерированных в ходе эволюции популяции программ используется fitness-функция, обычно представляющая собой среднеквадратичную ошибку результата? возвращаемого синтезированной программой, и эталонного значения.

Для формирования новой популяции в ГП используются следующие операции: отбор, кроссинговер и мутация (факультативно).

В результате отбора определяются кандидаты в следующее поколение, обладающие лучшими значениями fitness-функции.

Реализация кроссинговера начинается со случайного выбора по одному из узлов у каждого родителя, которые представляют собой их точки кроссинговера. Первый из потоков образуется посредством удаления фрагмента дерева первого родителя ниже его точки кроссинговера и включения в это место фрагмента дерева, также расположенного ниже сайта кроссинговера второго родителя.

Предположим, что для выполнения операции кроссинговера выбраны две родительские программы, эквивалентные следующим логическим выражениям:

$$F1 = (\text{NOT } X) \text{ OR } (X \text{ AND } Y);$$

$$F2 = (X \text{ OR } (\text{NOT } Y)) \text{ OR } ((\text{NOT } X) \text{ AND } (\text{NOT } Y)).$$

Будем считать, что в качестве точки кроссинговера выбрана вторая вершина первого дерева (операция NOT), а второго родителя соответственно – шестая вершина (операция AND). Иерархическое представление выражений *F1*, *F2*, а также их фрагменты участвующие в кроссинговере приведены на рис. 5.12.

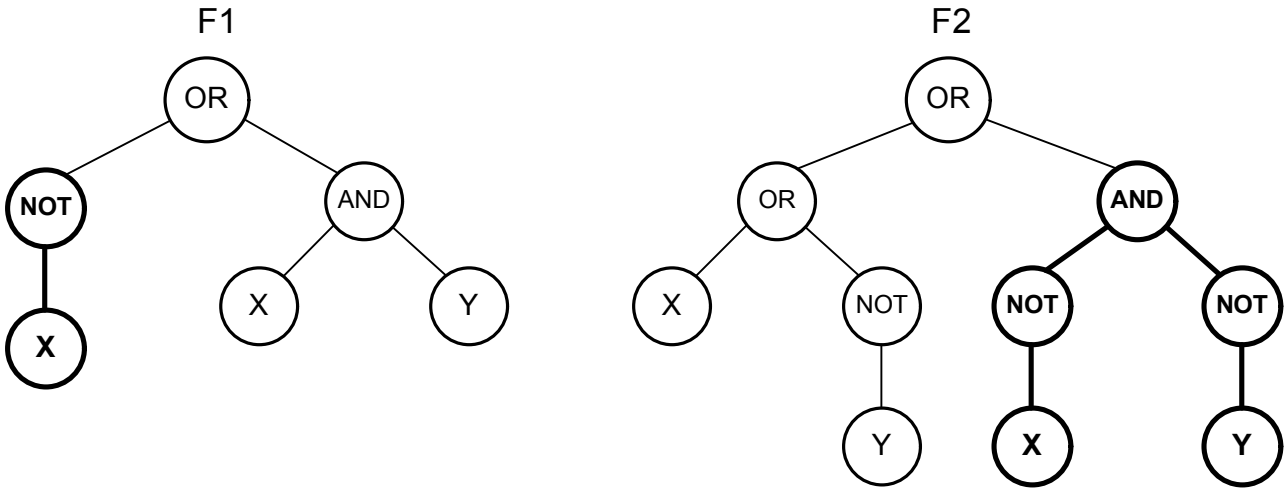


Рис. 5.12. Фрагменты программ, участвующие в кроссинговере

В результате две программы-потомка, полученные после кроссинговера, будут иметь вид, показанный на рис. 5.13, им будут соответствовать следующие логические выражения:

$$F1 = ((\text{NOT } X) \text{ AND } (\text{NOT } Y)) \text{ OR } (X \text{ AND } Y);$$

$$F2 = (X \text{ OR } (\text{NOT } Y)) \text{ OR } (\text{NOT } X).$$

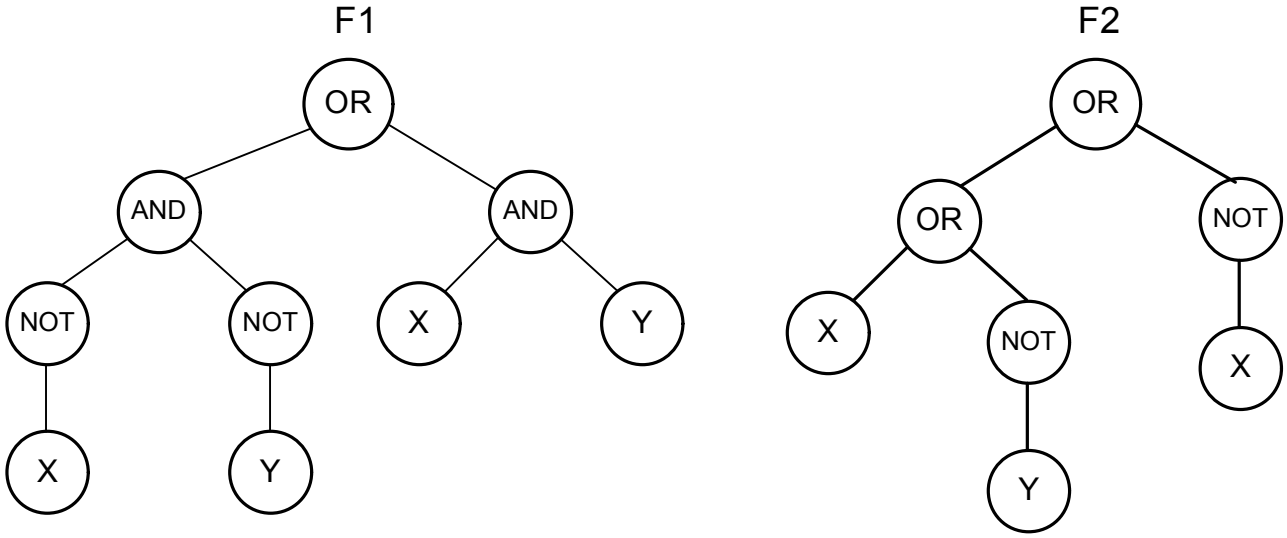


Рис. 5.13. Программы, полученные в результате кроссинговера

При этом выражение *F1* соответствует исходной эталонной программе.

В ряде случаев в ГП также может применяться модифицированный оператор мутации. В ходе своего выполнения он выбирает случайный узел дерева программы и удаляет все ветви, расположенные ниже него, заменяя при этом их случайно сгенерированным поддеревом. Однако по причине схожести выполняемых функций операторов кроссинговера и мутации применение последнего часто является необязательным.

В общем случае процедура ГП состоит в реализации следующих этапов.

1. Генерируется начальная популяция программ, составленных случайным образом из некоторого числа функций и терминальных символов, принадлежащих соответствующим множествам.

2. Реализуется определенная последовательность действий до тех пор, пока не будет выполнен критерий останова ГП:

- выполняется тестирование сгенерированных программ, после чего им присваивается соответствующее значение fitness-функции;

- создается новая популяция путем использования генетических операций, которые применяются к выбранным на основе их качества работы членам популяции.

3. Наилучшая программа, появившаяся в последнем поколении к моменту выполнения критерия останова, принимается в качестве результата ГП.

Критерием останова может выступать достижение заданного числа поколений или выполнение некоторого специального условия, определяемого спецификой решаемой задачи.

Примером применения технологии генетического программирования может служить решение задачи определения аналитического представления кривой, заданной в двумерной системе координат (рис. 5.14 (t = 0)).

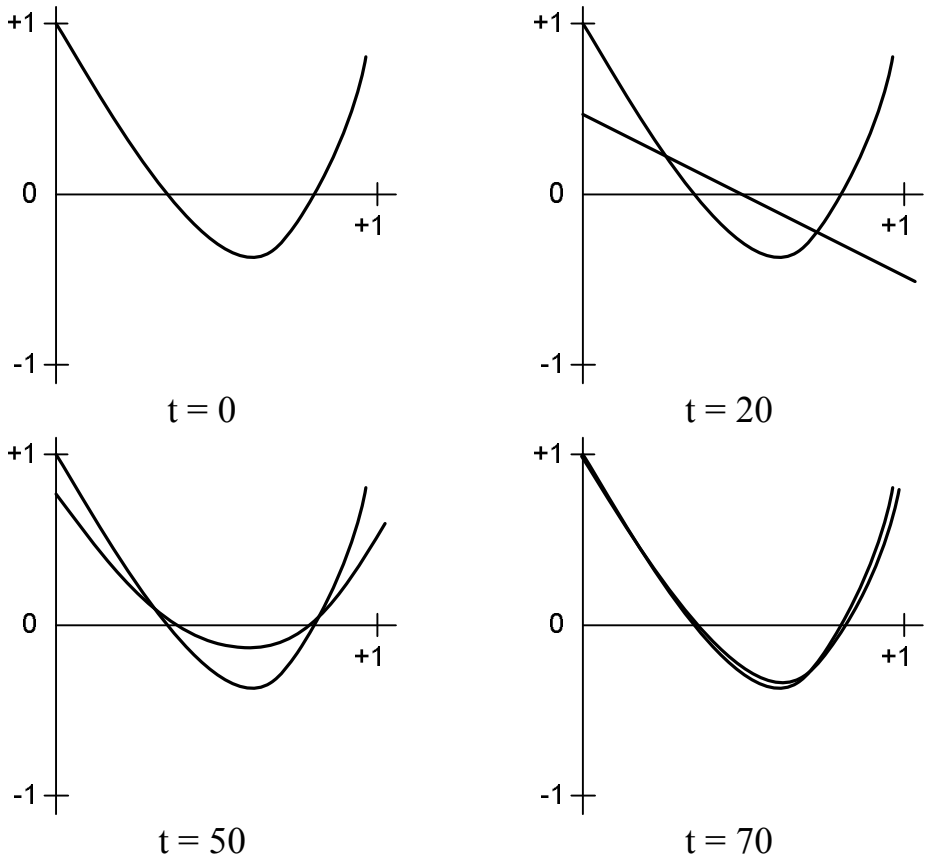


Рис. 5.14. Изменение графика аналитического выражения в процессе генетического программирования

На рис. 5.14 показана последовательность поиска такого выражения, причем в качестве функций использовались только арифметические операции {+, -, *, \}. Размер популяции, вероятности

применения генетических операторов кроссинговера и мутации были равны 100, 0,8, и 0,2 соответственно. Максимальная глубина древовидного представления выражений задавалась равной шести. Качество генерируемых выражений определялось как мера близости их значений и фактических значений в точках кривой. При этом от поколения к поколению наблюдалась тенденция постоянного уменьшения их различия.

Процесс генетического программирования продолжался на протяжении t = 70 поколений, в течение которых графики генерируемых выражений и, следовательно, само выражение постепенно эволюционировали от совершенно не похожих на заданную кривую до относительно близких к ней.

На последнем поколении была получена формула вида (*(-1(* 4 x)) (+ (- (* (* x x) 0.4375) (* (* (* x x) x) 2)) 1)) или $(1-4*x)*(0.4375*x^2-2*x^3+1)$, достаточно точно описывающая заданную кривую. Более того, при расширении выбранного функционального множества дополнительными функциями, например тригонометрическими, могут быть синтезированы, наряду с полученной, и другие формы аналитических выражений этой кривой.

Дальнейшее совершенствование генетического программирования связано с разработкой способов уменьшения структурной избыточности генерируемых программ, а также необоснованным разрушением генетическими операциями их синтаксически правильных фрагментов.

5.3. Эволюционные стратегии

Эволюционные стратегии (ЭС) были разработаны в 1960-х годах для структурного синтеза технических систем. При этом первоначально подобные исследования проводились без применения компьютеров, то есть создавалась реальная физическая модель, которая затем исследовалась, модифицировалась путем изменения позиций, добавления или удаления сегментов конструкции.

Первый компьютерный алгоритм ЭС был предложен в 1965 г. и затем усовершенствован в 1973 г. И. Риченбергом и имело достаточно простую форму, известную как бинарная или (1+1) - ЭС. В ней использовалось всего два объекта – родитель и потомок. Так же, как и в ГП отсутствовало различие между фенотипом и генотипом, каждый объект популяции представлялся как вектор вещественных чисел.

Процесс генерации новых решений в таких ЭС состоял в применении одного оператора мутации к случайно выбранному родителю. Мутация применялась независимо к каждому члену популяции по нормальному закону распределения случайной величины с нулевым математическим ожиданием и с предопределенным значением среднеквадратичного отклонения. Далее определялся уровень приспособленности потомка, и если он был выше, чем у соответствующего родителя, то потомок попадал в следующее поколение в качестве родителя, в противном случае он исключался и операция мутации повторялась к его родителю с целью получения нового потомка. Этот вариант ЭС имел ряд недостатков, вызванных двухточечной схемой работы, наиболее существенными из которых является медленная скорость нахождения решения.

Дальнейшее совершенствование этой технологии эволюционного моделирования связано с разработкой в начале 1980-х годов ЭС, использующих идею популяции решений. Они получили название $(\mu+\lambda)$ – ЭС и (μ,λ) – ЭС, где μ определяет число родителей; λ - число потомков.

В $(\mu+\lambda)$ – ЭС μ родителей порождает λ потомков, среди общего числа которых происходит отбор μ лучших особей в следующее поколение. В (μ,λ) – ЭС в отборе участвуют только сгенерированные потоки, поэтому для сохранения численности популяции и выполнения условия $\lambda > \mu$ рекомендуется использовать соотношение родитель : потоки как 1:7. Наряду с оператором мутации, в этих схемах ЭС применяется оператор рекомбинации, аналогичный кроссинговеру в ГА.

Ниже приводятся основные этапы эволюционных стратегий, основанных на использовании популяций.

1. Создается и инициализируется начальная популяция, которая может быть сгенерирована случайно или создана посредством мутации единственного решения введенного пользователем.

2. Для генерации потомков случайно выбираются родители из популяции родителей.

3. До достижения заданного числа популяции потомков реализуется следующая последовательность действий:

- на основе операторов рекомбинации определяется популяция потомков, при этом может быть использована информация об одном или о двух родителях подобно оператору кроссинговера в ГА для каждого параметра решения;

- генерируются новые решения на основе мутации, используя комбинации параметров ЭС.

4. Определяется целевая функция найденных решений, лучшие потомки отбираются для популяции родителей, после обновления популяции родителей происходит случайный выбор последних для генерации новых потомков.

5. Процесс останавливается, если найдены допустимые решения или выполнены условия останова.

5.4. Эволюционное программирование

Эволюционное программирование (ЭП) было разработано Л. Фогелем в 1960-х и во многом напоминает эволюционные стратегии. Однако изначальная область применения ЭП была связана с синтезом оптимальной структуры конечных автоматов, а именно: с исследованием динамики изменения таблиц переходов в пространстве конечных автоматов. Считается, что цель ЭП как эволюционной технологии заключалась в организации процедуры создания машинного интеллекта. Под интеллектуальным поведением подразумевалась возможность прогнозирования диаграммы конечных состояний, переводящих автомат к поставленной цели наиболее эффективным способом.

В ЭП нет разделения между генотипом и фенотипом, а единственным эволюционным оператором является мутация, который применяется к единственной популяции решений. Позже в конце 1980-х годов были разработаны модификации ЭП, включающие стандартное ЭП, мета – ЭП (или просто ЭП) и Рмета – ЭП. Их отличие друг от друга состоит в использовании различных механизмов организации направленной мутации с целью повышения уровня самоадаптации и имитации эффекта от применения оператора кроссинговера.

В качестве fitness-функции обычно используется некоторая мера близости между результирующим вектором значений конечного автомата и эталонной последовательностью символов.

Процесс функционирования ЭП состоит из следующих этапов.

1.Создается и инициализируется популяция на основе случайно сгенерированных особей или их случайно мутированных версий.

2. Определяется целевая функция всех элементов популяции.

3. До тех пор пока популяция не увеличится в два раза, реализуется следующая последовательность действий:

- выбирается родитель методом турнирной селекции из случайно сгенерированных групп;

- генерируется потомок посредством мутации копии родителя.

4. Определяется значение fitness-функции потомков для их оценки.

5. Удаляется половина популяции с наименьшими значениями fitness-функции.

6. Процесс останавливается, если найдены допустимые решения или выполнены условия останова.

В настоящее время попытки применения ЭА при решении прикладных задач охватывают не только класс традиционных задач оптимизации, но и распространяются на другие направления искусственного интеллекта, например управление сложными динамическими объектами в условиях неопределенности, принятие решений на основе нечеткого многокритериального выбора и т.п. Подобные исследования подразумевают под собой создание теоретических и практических основ для модификаций ЭА, подавляющее большинство среди которых концентрируется вокруг следующих направлений:

- интеграция с другими областями «мягкой» математики: нейронные сети, нечеткое моделирование;

- применение в самоорганизующихся, адаптивных системах;

- использование в качестве базового для моделирования систем искусственной жизни;

- реализация на параллельных архитектурах.

Контрольные вопросы

1. В чем характерные особенности эволюционного моделирования как направление искусственного интеллекта?
2. Какие существуют методы эволюционного моделирования?
3. Что такое генетический алгоритм? Дайте определения основным терминам, используемым в ГА.
4. Каковы способы представления информации в ГА?
5. В чем заключаются основные операторы ГА?
6. В чем состоит последовательность работы ГА?
7. Каковы особенности выбора значений параметров ГА?
8. Что такое генетическое программирование?
9. Как применяются генетические операторы при синтезе компьютерных программ?
10. В чем состоит последовательность работы процедуры генетического программирования?
11. Что такое эволюционные стратегии и каковы их особенности? Опишите последовательность работы этого эволюционного алгоритма.
12. Что такое эволюционное программирование и каковы его особенности? Опишите последовательность работы этого эволюционного алгоритма.
13. Каковы перспективы развития методов эволюционного моделирования?

ГЛАВА 6. РАЗРАБОТКА ЭКСПЕРТНЫХ СИСТЕМ

Экспертные системы (ЭС) как самостоятельное направление в искусственном интеллекте сформировалось в конце 1970-х годов. Начало истории развития ЭС связано с исследованиями по разработке ЭВМ пятого поколения, в котором основное внимание уделялось развитию «интеллектуальных способностей», позволяющих им оперировать не только данными, но и знаниями – подобно специалистам или экспертам при выработке умозаключений.

Экспертные системы предназначены исключительно для решения практических задач, поэтому они стали первыми интеллектуальными системами, которые привлекли внимание потенциальных потребителей продукции искусственного интеллекта. Наибольшее распространение ЭС получили в таких областях, как производство, бизнес, медицина, то есть там, где особенно часто возникает необходимость принятия решений.

6.1. Основные понятия

Экспертные системы можно определить как программу, которая на основе заложенных в ней знаний (опыта) эксперта может дать интеллектуальный совет или принять решение относительно некоторого вопроса в узкой предметной области, а также объяснить ход своих рассуждений понятным для пользователя образом [4, 10].

Обычно ЭС используются как инструмент для автоматизации работы эксперта. Кроме этого, ЭС может выступать в роли:

- консультанта для неопытных или непрофессиональных пользователей;
- ассистента эксперта-человека в процессах анализа вариантов решений;
- партнера эксперта в процессе решения задач, требующих привлечения знаний из разных предметных областей.

Следствием этого являются следующие преимущества от использования ЭС.

1. Постоянство. Человеческая компетенция ослабевает со временем. Перерыв в деятельности человека-эксперта может серьёзно отразиться на его профессиональных качествах.

2. Легкость передачи или воспроизведения знаний. Передача знаний от одного человека к другому, как правило, долгий и дорогой процесс, в отличие от которого передача электронной информации представляет собой простой процесс копирования программы или файла данных.

3. Устойчивость результатов. Эксперт-человек может принимать в тождественных ситуациях разные решения из-за эмоциональных факторов. Результаты ЭС – всегда стабильны.

4. Стоимость. Услуги экспертов, особенно высококвалифицированных, очень дороги. В противоположность этому ЭС характеризуются дорогостоящей разработкой, но дешевой эксплуатацией.

Главное отличие ЭС от традиционных систем обработки данных состоит в том, что в них преобладает символьный, а не числовой способ представления данных, а в качестве методов обработки информации применяются процедуры логического вывода и эвристического поиска решений (табл. 6.1).

Таблица 6.1

Отличия ЭС от обычных систем обработки данных

Характеристика	**Разработка ЭС**	**Традиционное программирование**
Тип обработки	Символьный	Числовой
Метод	Эвристический поиск	Точный алгоритм
Задание шагов решения	Неявное	Явное
Искомое решение	Удовлетворительное	Оптимальное
Управление и данные	Смешаны	Разделены
Знания	Неточные	Точные
Модификации	Частые	Редкие

Кроме этого, в отличие от традиционной программы ЭС при решении задачи выполняет не только предписанную алгоритмом последовательность действий, но и сама предварительно формирует их.

Также ЭС имеет возможность самообучаться на решаемых задачах, автоматически дополняя базу знаний результатами полученных выводов и решений.

В общем случае ЭС состоит из следующих основных компонент: базы знаний, рабочей памяти (базы данных), решателя (интерпретатора), системы объяснений, модуля приобретения знаний, интерфейса с пользователем (рис. 6.1) [10].

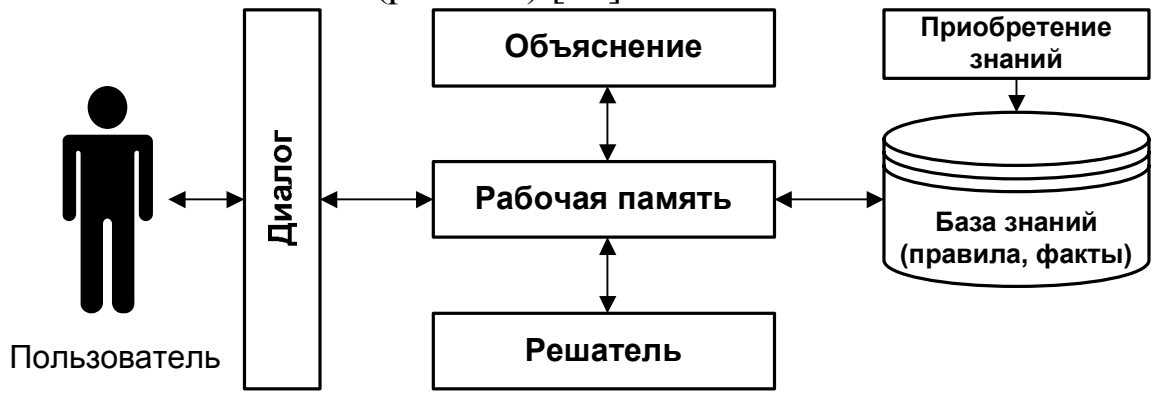


Рис. 6.1. Структура экспертной системы

База знаний ЭС предназначена для хранения информации, описывающей предметную область и обычно представляется в виде набора правил, а также ассоциированных с ними данных и процедур для их преобразования.

База данных (рабочая память) служит для хранения текущих данных решаемой задачи.

Решатель (интерпретатор) формирует последовательность применения правил и обрабатывает их, используя данные из рабочей памяти и знания из базы знаний. Наиболее простая реализация такого логического вывода соответствует продукционной форме представления знаний и рассматривалась ранее. Однако для общего случая знания о предметной области характеризуются неопределенностью, которая может проявляться как в истинности самой посылки, так и в заключении. Поэтому в ЭС при описании знаний часто добавляется вероятностная мера, а решатель соответственно реализует связанные с этим расчеты для выработки заключения и его вероятностной оценки.

Система объяснений показывает, каким образом система получила решения задачи и какие знания при этом использовались. Это облегчает тестирование ЭС и повышает доверие пользователя к

полученному результату. Благодаря объяснительному компоненту эксперт при тестировании ЭС локализует причины ее неудачной работы, что позволяет эксперту целенаправленно модифицировать старые и вводить новые знания. Результатом работы объяснительного компонента обычно является статистика по использованию из базы знаний правил и данных в процессе выполнения ЭС процедуры логического вывода.

Модуль приобретения знаний необходим для заполнения ЭС знаниями в диалоге с пользователем-экспертом, а также для добавления и модификации заложенных в систему знаний.

Любая ЭС должна иметь по крайней мере два режима работы. В режиме **приобретения знаний** эксперт наполняет систему знаниями, обычно в виде совокупности данных и правил. В дальнейшем эти знания будут использоваться для самостоятельного решения ЭС задач из конкретной предметной области. В режиме **консультаций** пользователь ЭС сообщает системе конкретные данные о решаемой задаче, на основе которых решатель формирует ответ пользователю, обычно в виде некоторого предположения.

6.2. Принципы разработки экспертных систем

Разработка ЭС выполняется группой людей, включающей экспертов, инженеров по знаниям и программистов. Инженер по знаниям помогает эксперту выявить и структурировать знания, необходимые для работы ЭС, определяет способ представления знаний в ЭС. При наличии коллектива экспертов возникает проблема оценки непротиворечивости формулируемых ими суждений. В такой ситуации могут применяться известные способы экспертного оценивания (методы ранжирования, парных сравнений, непосредственной оценки) или выделяется среди экспертов один, отвечающий за непротиворечивость знаний [4]. На рис. 6.2 показано взаимодействие основных участников создания и эксплуатации ЭС.

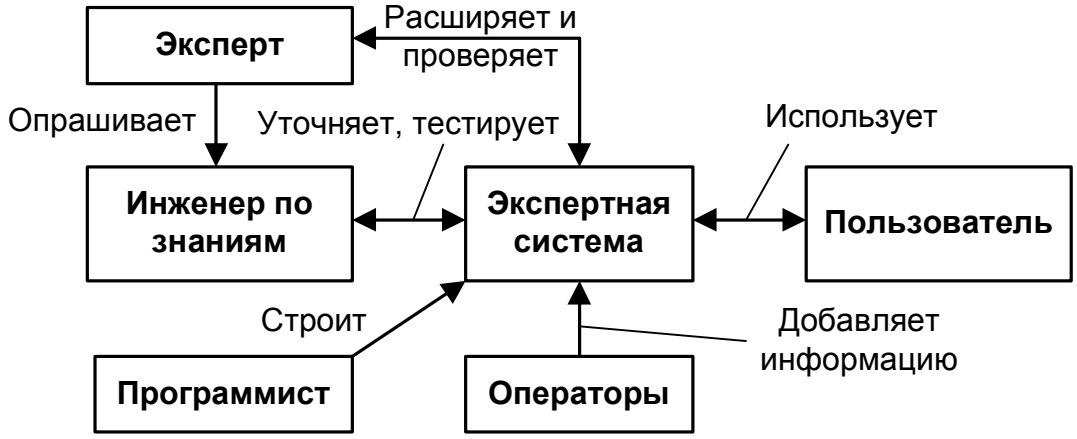


Рис. 6.2. Взаимосвязи участников разработки и пользователей ЭС

В этой схеме наиболее принципиальными являются такие объекты, как эксперт и инженер по знаниям.

Эксперт – человек, способный ясно выражать свои мысли и пользующийся репутацией специалиста, умеющего находить правильные решения проблем в конкретной предметной области. Эксперт обладает особыми приемами, чтобы сделать поиск решения более эффективным.

Инженер по знаниям – человек, как правило, имеющий познания в информатике и искусственном интеллекте и знающий как строить ЭС. Инженер по знаниям опрашивает экспертов, систематизирует знания, решает, каким образом они должны быть представлены в ЭС.

Перед тем как приступить к созданию ЭС, инженер по знаниям должен рассмотреть вопрос о целесообразности разработки ЭС для данной области. Положительное решение принимается, если отсутствуют традиционные вычислительные средства решения задачи, а также имеется возможность извлечь и эффективно формализовать экспертные знания.

Условие возможности реализации ЭС можно сформулировать в виде следующих требований [1]:

- существуют эксперты в данной области, которые решают задачу значительно лучше, чем начинающие специалисты;

- эксперты сходятся в оценке предлагаемого решения, так как в противном случае будет невозможно оценить качество разработанной ЭС;

- эксперты способны вербализовать (выразить на естественном языке) и объяснить используемые ими методы;

- решение задачи требует только рассуждений, а не действий;

- задача не должна быть слишком трудной (то есть ее решение должно занимать у эксперта несколько часов, а не недель или лет);

- задача, хотя и не должна быть выражена в формальном виде, все же должна относиться к достаточно «понятной» и структурированной области, то есть должна существовать возможность выделения основный понятий, отношений и способов получения решения задачи;

- решение задачи не должно в значительной степени опираться на «здравый смысл» (то есть широкий спектр общих сведений о мире и о способе его функционирования, которые знает и умеет использовать любой нормальный человек), так как подобные знания пока не представляется возможным в достаточном количестве заложить в ЭС.

В основе технологии разработки ЭС лежат шесть основных этапов (рис. 6.3), включающих идентификацию, концептуализацию, формализацию, выполнение, тестирование и опытную эксплуатацию [10].

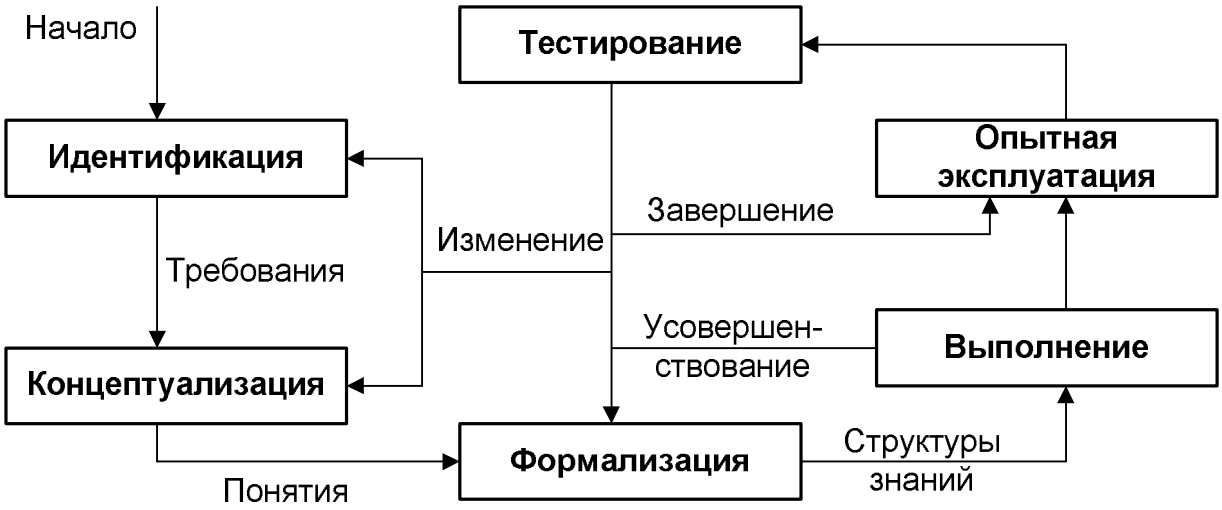


Рис. 6.3. Этапы разработки экспертных систем

На этапе **идентификации** определяются задачи, подлежащие решению, цели разработки, эксперты и типы пользователей.

На этапе **концептуализации** проводится содержательный анализ предметной области, выявляются используемые понятия и их взаимосвязи, определяются методы решения задач. Для определения

основных понятий предметной области могут быть использованы методы, основанные на различных психологических эффектах. Так, в методе «составление оглавления учебника» эксперту предлагается представить ситуацию, в которой его попросили написать учебник. Необходимо составить на бумаге перечень предполагаемых глав, разделов, параграфов, пунктов и подпунктов книги. Другой - «текстологический метод» формирования системы понятий - заключается в том, что эксперту дается задание выписать из руководств (книг по специальности) некоторые элементы, представляющие собой единицы смысловой информации.

Эффективность выполнения этих этапов во многом определяется успешным формированием авторитетной группы экспертов и получением от них качественных знаний.

На этапе **формализации** выбираются инструментальные средства и способы представления всех видов знаний, формализуются основные понятия, определяются способы интерпретации знаний.

На этапе **выполнения** осуществляется заполнение базы знаний. Поскольку основу ЭС составляют именно знания, то данный этап является наиболее важным и трудоемким. На данном этапе происходит занесение в ЭС предварительно формализованных в «понятном» системе знаний экспертов, а также возможно создание одного нескольких опытных прототипов экспертной системы.

На этапе **тестирования** эксперт и инженер по знаниям в интерактивном режиме проверяют компетентность ЭС. Процесс тестирования продолжается до тех пор, пока эксперт не решит, что система достигла требуемого уровня компетентности.

На этапе **опытной эксплуатации** проверяется пригодность ЭС для передачи конечным пользователям. Полученные результаты могут показать необходимость в существенной модификации ЭС.

Как и разработка любого программного обеспечения, процесс создания ЭС не сводится к строгой последовательности перечисленных этапов. В ходе разработки возникает потребность неоднократно возвращаться на более ранние этапы и пересматривать принятые там решения.

6.3. Классификация экспертных систем

В зависимости от своих функциональных свойств экспертные системы принято классифицировать по следующим признакам [1]:

- способу формирования решения;
- способу учета временного признака;
- виду используемых данных и знаний;
- числу используемых источников знаний.

По способу формирования решения ЭС можно разделить на анализирующие и синтезирующие. В системах первого типа выбирается решение из множества известных решений на основе анализа знаний, в системах второго типа решение синтезируется из отдельных фрагментов знаний.

В зависимости от способа учета временного признака ЭС делят на статические и динамические. Большинство ЭС являются статическими, в них не учитываются изменения в окружающем мире за время решения задачи. Однако существует класс задач, где выполнения этого требования является обязательным. Такие ЭС относятся к динамическим, их связь с реальным миром осуществляется через систему контроллеров и датчиков. Следствием этого является существенное изменение в механизмах логического вывода для адекватного отражения временной логики происходящих в реальном мире событий.

По видам используемых данных и знаний различают ЭС с детерминированными и неопределенными знаниями. ЭС могут создаваться с использованием одного или нескольких источников знаний.

В соответствии с перечисленными признаками можно выделить четыре основных класса ЭС (рис. 6.4): классифицирующие, доопределяющие, трансформирующие и мультиагентные [1].

Классифицирующие ЭС предназначены для решения задач распознавания ситуаций. Основным методом формирования решений в таких системах является дедуктивный логический вывод.

Доопределяющие ЭС используются для решения задач с не полностью определенными данными и знаниями.

Рис. 6.4. Основные классы экспертных систем

Трансформирующие ЭС относятся к синтезирующим экспертным системам, в которых предполагается повторяющееся преобразование знаний в процессе решения задач.

Мультиагентные ЭС представляют собой системы, основанные на интеграции нескольких разнородных источников знаний, которые обмениваются между собой получаемыми результатами в процессе решения задачи.

Однако такое многообразие ЭС появилось не сразу, а стало результатом эволюции из ряда поколений, на каждом из которых они предназначались для решения исключительно специфических для данного поколения задач. Процесс развития технологий ЭС условно можно разделить на три поколения.

Экспертные системы первого поколения в основном содержали исследовательские прототипы для исследования и обоснования теоретических основ искусственного интеллекта. Проводившиеся в этот период исследования носили фундаментальный характер, направленный на исследование отдельных фрагментов приобретения, представления и использования знаний, различных механизмов вывода.

Практическое использование ЭС носило исключительно консультирующий характер, то есть они представляли собой системы с

интеллектом пассивного ассистента пользователя. Они располагали только теми знаниями, которые были получены от экспертов, переработаны «инженерами знаний» и введены в базу знаний в «удобном» для машины виде. ЭС была способна манипулировать этими знаниями, имитируя процесс логического вывода, и выдавать ответы на запросы пользователя. Система не имела механизмов, которые позволяли бы ей критически оценивать вводимые в ее память знания, выявлять в них противоречия, автоматически обнаруживать закономерности, использовать их для предсказания и извлекать новые знания из данных. Дальнейшее развитие ЭС первого поколения было связано с использованием в них нечетких выводов.

Основным отличием **экспертных систем второго поколения** от ЭС первого поколения является их интегрированность. На основе объединения с традиционными информационными технологиями они становятся гибридными системами (интеллектуальными комплексами моделирования). Гибридные ЭС определяются как объединение пакетов прикладных программ и сложных расчетно - логических систем с традиционными ЭС в основном продукционного типа. Подобные ЭС должны решать проблему сочетания двух видов информации: количественных моделей точных наук и качественного опыта применения таких моделей в конкретных ситуациях. При этом точные модели избавляются от разного рода неформальных надстроек, а ЭС – от несвойственных им количественных зависимостей.

Таким образом, направление гибридных ЭС предполагает объединение процедурных методов моделирования с недетерминированными методами вывода, используемыми в технологии ЭС. Создание таких систем привело к разработке нового подхода к математическому моделированию, позволяющему проводить качественное моделирование с использованием информации в виде фактов и правил. Такие комплексы включают расчетный (имитационный) компонент, обеспечивающий количественные решения, и эвристический (логический) компонент, позволяющий успешно решать качественные задачи.

При применении гибридных ЭС при автоматизации задач проектирования, кроме отмеченных ранее функций, обеспечиваются:

- генерация новых фактов и знаний путем логического вывода из имеющегося набора правил и фактов;

- реализация различных схем решения задач, отличающихся последовательностью использования типовых для системы методов;

- возможность обоснования решений на основе многокритериального анализа, планирования вычислений, логического вывода, оптимизации.

Главное направление происходящих в настоящее время смен концепций создания ЭС и использования средств искусственного интеллекта – это переход от предположений, справедливых только для изолированных систем искусственного интеллекта, и от индивидуальных, автономных систем к распределенной обработке информации и разработке многоагентных интеллектуальных систем - **перспективных экспертных систем**. Основой для создания подобных ЭС являются результаты, имеющиеся в области методов обнаружения закономерностей, распознавания образов, структурно – логического анализа данных и знаний.

Основной отличительной функцией ЭС является умение давать правильные предсказания, рекомендации на основе обработки поступающих данных и выявления устойчивых (закономерных) связей между характеристиками данных.

Данные системы обладают средствами самостоятельного извлечения знаний из данных, поступающих в систему в ходе ее создания и эксплуатации. Это дает возможность ЭС обнаруживать противоречия между имеющимися и вновь поступающими знаниями и данными (проверка адекватности базы знаний). Подобные возможности позволяют рассматривать данный класс ЭС как «активные» экспертные системы, выполняющие роль активного помощника пользователя (советующие системы).

Эволюция функциональных усовершенствований ЭС различных поколений, а также их характеристики представлены в табл. 6.2.

Таблица 6.2

Характеристики экспертных систем

Основные характеристики	**ЭС первого поколения**	**ЭС второго поколения**	**Перспективные ЭС**
Способы извлечения знаний, структура баз знаний	«Инженер знаний», эмпирические знания эксперта	«Инженер знаний», исследования по автоматическому извлечению знаний из базы знаний	Автоматическое обнаружение из баз знаний (из текстов, руководств, схем и т.д.)
Типы баз знаний	Отдельные формы: продукция, фреймы, семантические сети, решающие деревья	Работа с любыми формами знаний (включая инженерные знания)	Библиотека форм знаний, имитационные модели, сценарии
Источник знаний	Эксперт	Эксперты, базы знаний	Базы знаний, данные, статистические или эмпирические таблицы вида «объект - свойство - время»
Наличие базы данных	Отсутствует	Частично, базы данных из таблиц вида «объект – свойство»	Базы данных из таблиц вида «объект - свойство – время»
Логический вывод	Вывод по дедукции	Дедукция, нечеткие выводы, индукция, немонотонные рассуждения, частично рассуждения по аналогии	Дедукция, индукция, немонотонные рассуждения, методы близости в пространстве знаний, рассуждения по аналогии
Язык общения с пользователем	Фразы и термины жесткой конструкции прикладной области	Проблемно-ориентированный естественный язык	Сценарии диалога, формирование терминологии под прикладную область и форм сообщений
Устный диалог	Отсутствует	Ограниченный словарь	Словари по определяющим терминам прикладных областей в процессе общения и использования

Продолжение табл. 6.2

Основные характеристики	**ЭС первого поколения**	**ЭС второго поколения**	**Перспективные ЭС**
Проверка адекватности баз знаний	Отсутствует	Частичная проверка непротиворечивости и полноты баз знаний	Проверка непротиворечивости, полноты, работа с информацией с НЕ – факторами
Прогнозирование недостающих данных в базе данных	Отсутствует	Исследования по автоматическому прогнозированию величин, отсутствующих в базе данных	Автоматическое прогнозирование величин, отсутствующих в базе данных
Модель пользователя	Отсутствует	Обучение систем по адаптации под конкретного пользователя	Программы адаптации под конкретных пользователей и интерфейс с системами проектирования
Выдаваемые результаты	Числовые данные, стандартные рекомендации	Числовые данные, рекомендации с сопровождающим пояснением, обучение систем обнаружению новых закономерностей	Числовые данные, рекомендации с сопровождающим пояснением, формулирование обнаруженных новых закономерностей, тенденций, диаграммы
Вид обрабатываемой информации	Статическая	Динамическая, статическая	Статическая, динамическая, потоковая
Принцип построения и использования системы	Обособленное использование ЭС для решения задачи	Гибридное построение ЭС (эвристическая и имитационная компоненты)	Гибридные интеллектуальные нечеткие системы (интеллектуальные интегрированные комплексы моделирования), открытая система
Обработка распределенных знаний	Отсутствует	Исследования по построению распределенных ЭС	Распределение ЭС, многоагентные системы искусственного интеллекта

Окончание табл. 6.2

Основные характеристики	ЭС первого поколения	ЭС второго поколения	Перспективные ЭС
Функции системы	Пассивный помощник пользователя	Активный помощник пользователя, моделирование процессов инженерной деятельности	Комплексная компьютеризация деятельности, активный помощник пользователя
Ввод новых знаний, модификация знаний	В режиме ввода информации	В рабочем режиме экспертной системы	В рабочем режиме экспертной системы

6.4. Пример экспертной системы

В качестве примера рассмотрим процедуру создания ЭС в области медицинской диагностики. Подобные ЭС имеют большую практическую ценность, потому что позволяют работать (определять диагноз) в условиях неполной информации, то есть, в отличие, например от медицинской энциклопедии, где по каждому заболеванию можно найти абсолютно полный набор точных симптомов, ЭС позволяет решить обратную задачу, а именно: определение болезни по предъявляемым ей симптомам. В этом случае приходится сталкиваться с неопределенностью представляемой информации, так как в очень редких случаях отдельные симптомы однозначно характеризуют болезнь (например, высокая температура далеко не всегда связана с таким заболеванием как грипп). Для управления такой неопределенностью, свойственным многим прикладным областям, в ЭС часто используют вероятностную модель на основе теоремы Байеса [4, 10].

В основе этой теоремы лежит понятия условной вероятности $P(A|B) = x$, которое означает, что при условии наступления события B (и всего остального, что не имеет отношения к A) вероятность возникновения A равна x. Пусть имеется условная вероятность $P(A|B)$ наступления некоторого события A при условии, что наступило событие B. Теорема Байеса позволяет решить обратную задачу –

определить вероятность наступления более раннего события B, если известно, что более позднее событие A наступило.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}.$$

Здесь $P(B|A)$ называют правдоподобием, а знаменатель $P(B)$ - свидетельством.

Эффект применения данной теоремы в ЭС состоит в том, что, изучая следствие, можно определить вероятность причины. В этом случае знания в ЭС будут представляться в форме: «Если H является истиной, то E будет наблюдаться с вероятностью P»,

где H – событие, заключающееся в том, что данная гипотеза верна;

E – событие, заключающееся в том, что наступило определенное доказательство (свидетельство), которое может подтвердить правильность указанной гипотезы.

Таким образом, разрабатываемая ЭС будет определять вероятность наличия у больного заболевания при подтверждении или опровержении им некоторой группы симптомов. Для простоты будем предполагать, что ЭС должна определять только такие болезни, как грипп и отравление. Основные симптомы этих заболеваний приведены в табл. 6.3.

Таблица 6.3

Основные симптомы

№	Симптом
1	Высокая температура
2	Боль в животе
3	Кашель
4	Тошнота
5	Головная боль
…	…

Далее необходимо связать болезни и их симптомы, а именно: создать базу знаний ЭС. Этот этап должен выполняться непосредственно экспертами, именно от точности формализации их знаний будет зависеть качество работы ЭС. В табл. 6.4 приведен фрагмент базы знаний примера.

Таблица 6.4

База знаний примера

№	(H)	P(H)	[j, P(E|H), P(E|$\overline{H}$)]
1	Грипп	0.01	(1, 0.7, 0.01)
2			(2, 0.01, 0.5)
3			(3, 0.7, 0.3)
4			(4, 0.1, 0.5)
5			(5, 0.8, 0.1)
6	Отравление	0.01	(1, 0.4, 0.5)
7			(2, 0.7, 0.1)
8			(3, 0.01, 0.5)
9			(4, 0.8, 0.1)
10			(5, 0.6, 0.2)

Здесь поле H характеризует название возможного заболевания, например «Грипп». Следующее поле P(H) определяет априорную вероятность этого заболевания при отсутствии дополнительной информации. Для нашего случая P(H) = 0,01, что означает ситуацию, когда любой наугад взятый человек болеет гриппом. Значения последнего столбца состоят из трех элементов. Первый определяет номер симптома. Следующие два соответственно вероятности получения ответа «Да» на этот вопрос, в случае возможности или невозможности заключения. Так, для первого симптома $P(E|H) = 0.7$, $P(E|\overline{H}) = 0{,}01$, это означает, что если у пациента грипп, то он в семи случаях из десяти ответит «Да» на этот симптом, а если у него нет гриппа, то он ответит утвердительно лишь в одном случае из ста. Положительный ответ подтверждает гипотезы о том, что у него грипп, отрицательный ответ опровергает это заболевание. Для второго симптома P(E|H) = 0,01, следовательно, вероятность его появления у больного гриппом очень мала. В целом в зависимости от болезни каждый симптом может проявлять себя по-разному. Кроме этого, в качестве симптомов могут указываться вопросы, например «Наблюдаете ли вы такой симптом на протяжении большей части жизни ?».

Таким образом, при формировании базы знаний некоторой прикладной области экспертами должны быть сформулированы следующие вероятности:

- априорные вероятности всех возможных гипотез ($P(H)$);

- условные вероятности возникновения свидетельств, при условии существования каждой из гипотез ($P(E|H)$).

При этом условные вероятности должны быть получены для всех свидетельств и заключений, предполагая, что все свидетельства независимы в рамках одного заключения.

Все это свидетельствует о необходимости проведения исследований перед назначением соответствующих вероятностей при формировании базы знаний ЭС. Для повышения качества экспертных оценок могут быть использованы специальные методы определения коэффициентов уверенности и доверия.

Перед началом работы с ЭС рекомендуется определить степень важности имеющихся симптомов по следующей формуле:

$$R_j = \sum_{i=1}^{n} | P(E \mid H_i) - P(E \mid \overline{H}_i) | .$$

После ранжирования симптомов по значению *Rj* ЭС может работать в режиме консультаций, при этом вопросы пользователю задаются, начиная с наиболее важного симптома. На вопросы ЭС пользователь отвечает «Да» или «Нет» (значения 1 или 0 соответственно), определяя наличие симптома.

Каждый раз после поступления новой информации от пользователя априорные вероятности гипотез новыми значениями. Для этого используется правило Байеса, являющееся следствием теоремы.

$$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E \mid H)P(H) + P(E \mid \overline{H})P(\overline{H})} .$$

Вероятность подтверждения некоторой гипотезы *H* при наличии определенных подтверждающих свидетельств *E* вычисляется на основе априорной вероятности этой гипотезы без подтверждающих свидетельств и вероятности осуществления свидетельств при условиях, что гипотеза верна или неверна.

Приведенная формула используется для корректировки вероятности гипотезы при положительном ответе пользователя на вопрос. При отрицательном ответе данная зависимость принимает вид

$$P(H \mid E) = \frac{(1 - P(E \mid H))P(H)}{[(1 - P(E \mid H))P(H) + (1 - P(E \mid \overline{H}))P(\overline{H})]} .$$

Имеющиеся заключения упорядочиваются по убыванию значений $P(H)$. Заключение с максимальной вероятностью принимается в качестве результата работы экспертной системы.

Следуя примеру, предположим, что ЭС нужно выполнить диагностику заболевания гриппом. Тогда значения вероятностей обеих болезней в процессе консультации с ЭС будут изменяться следующим образом (табл. 6.5). Симптомы ранжированы в соответствии со степенью их важности.

Таблица 6.5

Результаты работы ЭС

Симптом	**Ответ**	**P(Грипп)**	**P(Отравление)**
Головная боль	Да	0,074	0,029
Тошнота	Нет	0,125	0,006
Высокая температура	Да	0,9	0,005
Боль в животе	Нет	0,94	0,0017
Кашель	Да	0,97	0,0003

Как видно из табл. 6.5, после получения ответов на все вопросы, ЭС полностью подтвердила с высокой вероятностью изначально предполагаемый диагноз гриппа у пациента.

Рассмотренная ЭС является достаточно простой и имеет целый ряд допущений, главное из которых состоит в предположении об абсолютной достоверности получаемых от пользователя ответов (Да, Нет). Однако чаще всего в таких случаях информация носит многозначный, неточный характер («Может быть», «Скорее да, чем нет» и т.п.), что требует применения в ЭС более сложных механизмов логического вывода, например на базе математических основ теории нечетких множеств.

Кроме этого, при организации на основе вероятностной модели приходится сталкиваться с необходимостью получения огромного числа вероятностей, образующих базу знаний. Так, если некоторая медицинская область имеет 100 диагнозов и 700 симптомов, то уже в этом случае должны быть получены 70100 значений вероятностей

(70000 условных и 100 априорных). Также необходимо учитывать требование независимости симптомов, что часто не соответствует действительности.

Решением этой проблемы может служить применение байесовских сетей доверия, с помощью которых описываются информационные потоки в предметных областях, характеризующихся наследованной неопределенностью. Было показано, что с помощью таких сетей можно создать согласованную и непротиворечивую базу знаний без необходимости в предположении условной независимости.

Также следует отметить перспективность построения ЭС на основе искусственных нейронных сетей.

6.5. Разработка экспертных систем на основе байесовских сетей доверия

Байесовские сети доверия применяются для описания знаний тех областей, которые характеризуются наследованной неопределенностью содержащихся в них причинно-следственных связей [7].

Сети Байеса представляют собой направленный граф без циклов, обладающий следующими свойствами:

- каждая вершина представляет собой событие, описываемое случайной переменной, которая может иметь несколько состояний;

- каждая перемененная может принимать одно из конечного множества взаимоисключающих значений;

- каждой переменной-потомку A с переменными предками $B_1,...,B_n$ приписывается таблица условных вероятностей $P(A \mid B_1,...,B_n)$.

- если переменная A не содержит предков на графе, то вместо условных вероятностей используются безусловные (априорные) вероятности $P(A)$.

Таблица условных вероятностей каждой дочерней вершины содержит вероятности состояний этой вершины, при условии возможных состояний ее родительских вершин.

На рис. 6.5 показан пример простой байесовской сети. Здесь вершины A, B, C соответствуют ситуации, когда A, B являются причиной для C. Важно, что при этом должно выполняться условие независимости случайных переменных, соответствующих вершинам графа. Две вершины A и B являются условно независимыми при данной третьей вершине C, если при известном значении C значение B не увеличивает информативность о значениях A: $P(C|\ A, B) = P(C|A)$. Тогда вероятность пребывания вершины C в различных состояниях C_k зависит от состояний A_i и B_j вершин A, B и определяется как

$$P(C_k) = \sum_i \sum_j P(C_k \mid A_i, B_j) P(A_i, B_j) .$$

Принимая во внимание тот факт, что A и B события условно независимые, $P(A_i, B_j) = P(A_i) \cdot P(B_j)$.

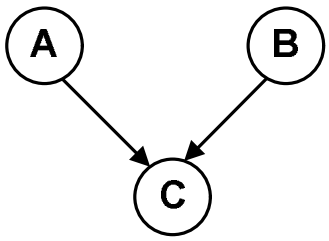


Рис. 6.5. Простейшая байесовская сеть

В байесовских сетях вероятностная оценка может производиться в обоих направления на графе. Для этого используется теорема Байеса для случая влияния свидетельства E на множество несовместимых, взаимоисключающих гипотез $H_i, i = 1..m$.

$$P(H_i \mid E) = \frac{P(E \mid H_i) P(H_i)}{\sum_{k=1}^{m} P(E \mid H_k) P(H_k)} .$$

Полученное значение $P(H_i \mid E)$ называется апостериорной вероятностью гипотез H_i по свидетельству E.

В случае множества условно независимых свидетельств эта формула принимает вид

$$P(H_i \mid E_1 E_2 ... E_n) = \frac{P(E_1 \mid H_i) P(E_2 \mid H_i) \cdot ... \cdot P(E_n \mid H_i) P(H_i)}{\sum_{k=1}^{m} P(E_1 \mid H_k) P(E_2 \mid H_k) \cdot ... \cdot P(E_n \mid H_k) P(H_k)} .$$

Таким образом, введение в сеть новых свидетельств приводит к их распространению по графу, в результате каждому высказыванию, ассоциированному с вершинами графа, назначается вычисленная апостериорная вероятность, определяющая степень доверия к этому высказыванию.

В качестве примера предположим, что есть некоторый набор предположений относительно возможности потери деревьями в саду листвы: «листья часто опадают, если дерево засыхает из-за недостатка влаги или если дерево болеет». Эта ситуация может быть описана с помощью байесовской сети доверия (рис. 6.6), содержащей три вершины «Болеет», «Засохло» и «Облетело». При этом каждая вершина может иметь одно из двух возможных состояний 1/0 (да/нет).

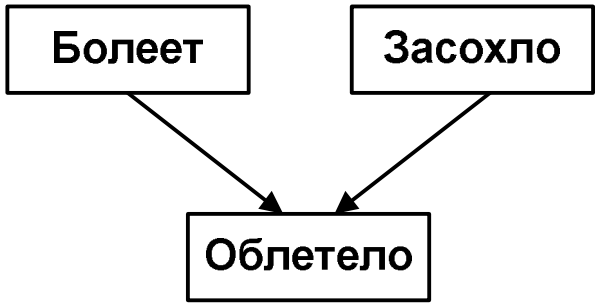


Рис. 6.6. Байесовская сеть примера

Количественно данную сеть можно описать с помощью следующих вероятностных правил.

1. P(Болеет = «да») = 0,1
2. P(Болеет = «нет») = 0,9
3. P(Засохло = «да») = 0,1
4. P(Засохло = «нет») = 0,9
5. ЕСЛИ Болеет = «да» И Засохло = «да», ТО
P(Облетело = «да») = 0,95
6. ЕСЛИ Болеет = «да» И Засохло = «да», ТО
P(Облетело = «нет») = 0,05
7. ЕСЛИ Болеет = «нет» И Засохло = «да», ТО
P(Облетело = «да») = 0,85
8. ЕСЛИ Болеет = «нет» И Засохло = «да», ТО
P(Облетело = «да») = 0,15

9. ЕСЛИ Болеет = «да» И Засохло = «нет», ТО
P(Облетело = «да») = 0,9
10. ЕСЛИ Болеет = «да» И Засохло = «нет», ТО
P(Облетело = «нет») = 0,1
11. ЕСЛИ Болеет = «нет» И Засохло = «нет», ТО
P(Облетело = «да») = 0,02
12. ЕСЛИ Болеет = «нет» И Засохло = «нет», ТО
P(Облетело = «нет») = 0,98

Вершины «Болеет», «Засохло» не имеют родительских вершин, поэтому для них записывают безусловные вероятности нахождения этих вершин в каждом из возможных состояний. Этому соответствуют первые четыре правила. Остальные правила описывают вероятности пребывания дочерней вершины «Облетело» в зависимости от состояний родительских вершин и представляются в виде таблиц условных вероятностей.

Предположим, стало известно о том, что дерево засохло, то есть P(Облетело = «да») = 1. Тогда, используя свойство обратного оценивания, можно определить вероятности влияния на это ситуаций, что дерево – болеет и дерево – засохло: P(Болеет = «да» | Облетело = «да») = 0.49, P(Засохло = «да» | Облетело = «да») = 0.47.

Рассмотрим более сложный случай описания с помощью сети Байеса базы знаний, в которой можно выделить заболевания, симптомы, их проявления, а также факторы риска, влияющие на возникновения заболеваний. На рис. 6.7 приведена структура такой сети, взаимоотношения между узлами которой соответствуют следующему набору медицинских знаний:

- отдышка может быть из-за туберкулеза, рака легких или бронхита, а также вследствие отсутствия перечисленных заболеваний или наличия более, чем одного;

- визит в Азию повышает шансы туберкулеза;

- курение является фактором риска как для рака легких, так и бронхита;

- результаты рентгена, содержащие патологию, не позволяют однозначно диагностировать рак и туберкулез, так же как не подтверждают факт наличия или отсутствия отдышки.

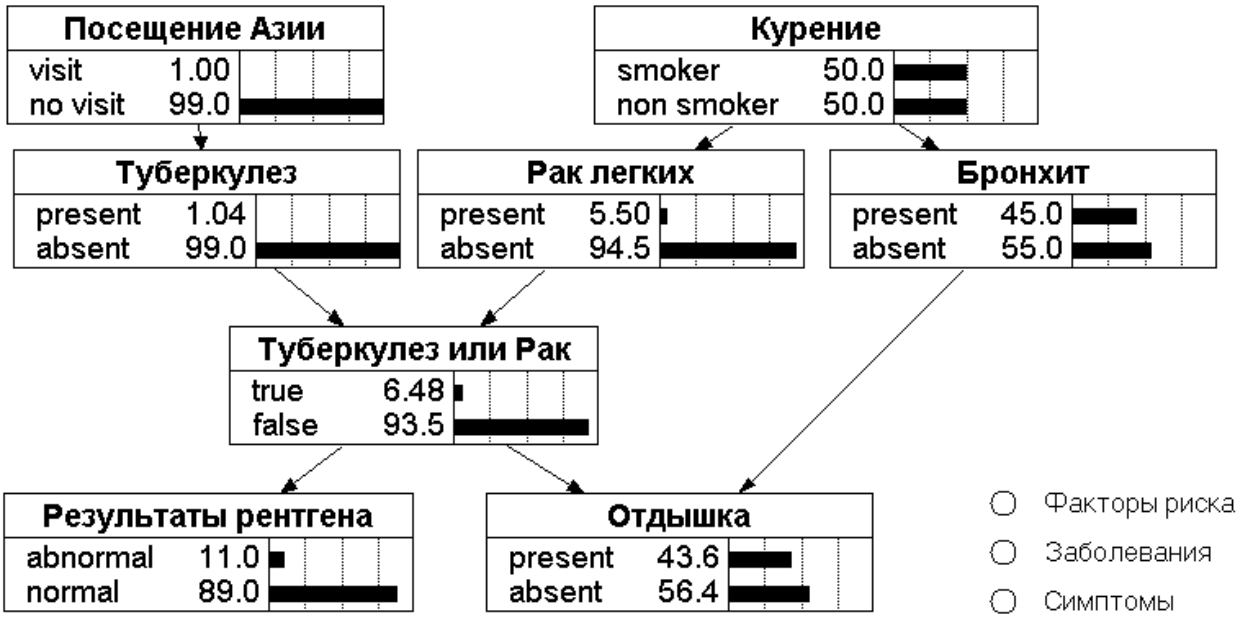


Рис. 6.7. Структура сети Байеса для примера

Вершина «Туберкулез или Рак» играет роль промежуточной переменной, соответствующей дизъюнкции и означает наличие одной, двух болезней или их отсутствие.

Данная сеть Байеса была построена в системе NETICA фирмы Norsys Software Corp. При задании начальных безусловных вероятностей предполагались следующие статистические данные о некоторой достаточно большой группе людей:

- 1% людей посещал Азию;
- 50% людей курит;
- 1% имеет заболевание туберкулезом;
- 5,5% имеют рак легких;
- 45% страдают легкой или хронической формой бронхита;
- 6,48% имеют туберкулез или рак легких;
- у 11% результаты рентгена могут показать наличие патологии;
- 43,6% страдают отдышкой.

Теперь при обращении пациента и введении в сеть информации о его симптомах или свидетельств о факторах риска могут быть получены заключения в виде вероятностных оценок возможных заболеваний, а также наличия сопутствующих им причин. Так, при подтверждении пациентом наличия у него отдышки значения в узлах сети изменятся следующим образом (рис. 6.8).

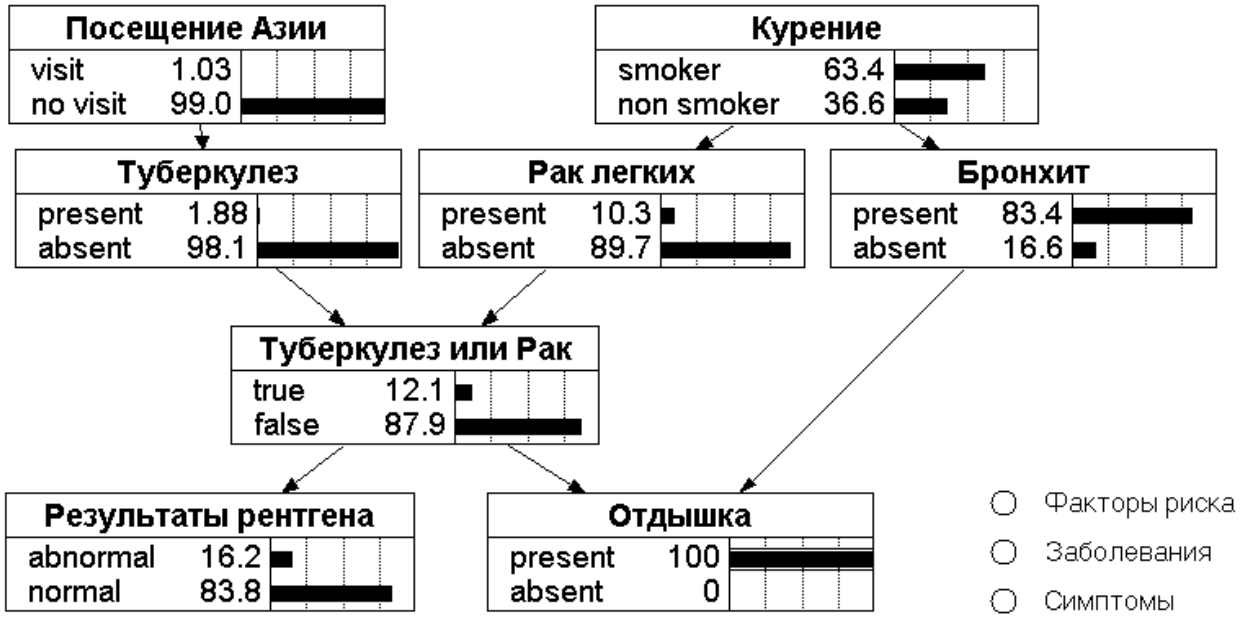


Рис. 6.8. Изменение значений в узлах сети после ввода новых данных

Возникновение таких изменений, как отмечалось ранее, вызвано пересчетом вероятностей при прямом и обратном распространении новых значений по узлам сети, в частности:

- вероятность заболевания бронхитом (доверия к этому диагнозу) увеличилась с 45 до 83,4%;

- шансы, что пациент при этом курит увеличились с 50 до 63,4%;

- вероятность наличия патологий по результатам рентгена увеличилась, но незначительно с 11 до 16%.

Продолжая опрашивать пациента и одновременно корректируя на основе его свидетельств значения в узлах сети будут в разной степени изменяться вероятности диагнозов. Так, при подтверждении фактов посещения Азии, а также курения, но при наличии результатов рентгена без патологий вероятности диагнозов изменятся следующим образом: бронхит – 92%, туберкулез – 0,02%, рак легких – 0,39%. Итогом является возможность с большой долей уверенности констатировать диагноз и назначить пациенту соответствующее лечение.

Благодаря своим свойствам байесовские сети доверия широко применяются не только в таких областях, как медицина, но и при стратегическом планировании, в финансах и экономике.

Развитием сетей Байеса является создание на их основе **диаграмм влияния** для принятия решений. Конструктивно это реализовывается за счет добавления к уже рассмотренным вершинам шансов новых типов вершин: узлов полезности, обозначаемых в виде ромбов и узлов решения, обозначаемых прямоугольниками (рис. 6.9)

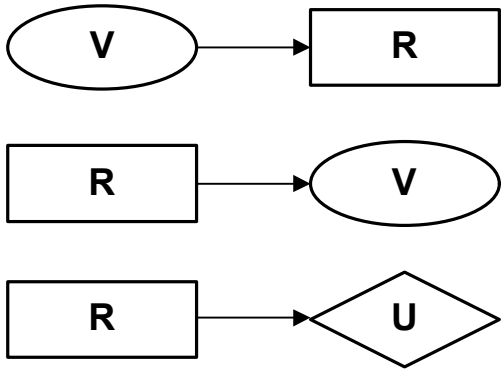


Рис. 6.9 Новые типы узлов сети Байеса

Указания, содержащиеся в узлах решения, указывают на временные причинно-следственные связи:

- стрелка, идущая от случайной переменной к переменной решения, указывает, что значение этой переменной станет известно, когда будет принято решение;

- стрелка, идущая от одной переменной решения к другой, отражает хронологическую последовательность соответствующих решений.

Важно, что сеть должна быть без циклов и содержать путь, проходящий через все узлы решения.

При создании диаграмм ставится цель поиска оптимального решения. Для этого с каждым состоянием сети связывается узел полезности, который характеризуется функцией полезности. Эта функция отражает ценность принятого решения в зависимости от состояний узлов-родителей. Узлы полезности не имеют потомков. Принятое решение оказывает влияние на значение вероятностей состояний узлов, поэтому можно вычислить ожидаемую ценность каждого из альтернативных решений. Затем из предложенных решений выбрать решение с максимальной полезностью. Диаграмма влияния может содержать несколько вершин полезности. В этом случае полная функция полезности вычисляется как сумма отдельных функций.

В качестве примера рассмотрим простую ситуацию принятия решения о необходимости брать с собой зонт, выходя из дому, основываясь на информации о прогнозе погоды. Диаграмма влияния для этой задачи приведена на рис. 6.10.

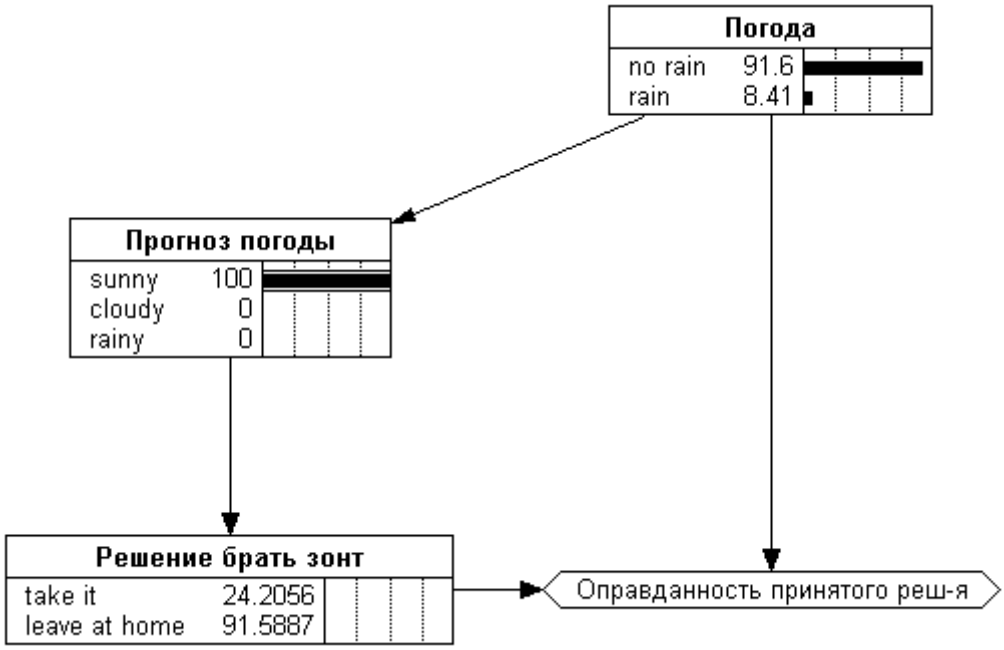


Рис. 6.10. Структура диаграммы влияния для примера

Диаграмма имеет два узла шансов. Первый представляет возможный прогноз погоды утром: sunny (солнечно), cloudy (облачно), raining (дождливо). Второй - вероятность дождя в течение дня. Также имеется узел решения (брать (take it) или не брать (leave at home) зонт) и узел полезности, позволяющий количественно оценить оправданность принятого решения.

Из анализа связей между узлами видно, что вершина «Прогноз погоды» имеет стрелку в вершине «Решение брать зонт», тем самым предполагая, что человек будет знать прогноз погоды перед принятием решения. Однако между узлами «Погода» и «Решение брать зонт» связь отсутствует, что вполне оправдано, так как, зная погоду в течение дня, решение о необходимости брать зонт будет очевидным В свою очередь, стрелки от вершин «Погода» и «Решение брать зонт» к вершине «Оправданность принятого решения» определяют следующие отношения между узлами:

- ЕСЛИ «нет дождя» и «не взял зонт», ТО
  Оправданность решения = 100;
- ЕСЛИ «есть дождь» и «взял зонт», ТО Оправданность решения = 70;
- ЕСЛИ «нет дождя» и «взял зонт», ТО Оправданность решения = 20;
- ЕСЛИ «есть дождь» и «не взял зонт», ТО
  Оправданность решения = 0;

Предположим, что по прогнозу погоды будет солнечно. Тогда, подтвердив в узле «Прогноз погоды» событие «солнечно», значения в узлах сети изменятся следующим образом (рис. 6.10). Согласно узлу

«Погода», вероятность дождя составит 8,41%, и зонт имеет смысл оставить дома, так как в узле полезности значение оправданности такого решения равно 91,58.

В заключение отметим наиболее известные классические экспертные системы, имеющие опыт практического применения, а также современные средства разработки ЭС:

- MICIN – экспертная система для медицинской диагностики, разработанная группой исследователей по инфекционным заболеваниям Стенфордского университета;
- PUFF – экспертная система, являющаяся модификацией MICIN для диагностики легочный заболеваний;
- DENDRAL – экспертная система для распознавания химических структур по данным спектрометрии вещества;
- PROSPECTOR - экспертная система, предназначенная для поддержки поиска месторождений полезных ископаемых.
- NETICA – система разработки ЭС на основе байесовских сетей доверия, позволяющая создавать и редактировать причинно-следственные связи, заполнять таблицы условных вероятностей, а также проводить расчеты для принятия решений по всем событиям, входящим в сеть.

Контрольные вопросы

1. Что такое экспертная система и каково ее назначение?
2. Какие основные понятия связаны с экспертными системами?
3. Какие типы задач решаются с применением ЭС?
4. Какие компоненты образуют структуру ЭС?
5. Каковы условия возможности реализации ЭС?
6. Какие особенности свойственны участникам разработки ЭС?
7. Каковы этапы разработки ЭС?
8. Какие существуют основные классы ЭС? Охарактеризуйте поколения их развития.
9. Каковы принципы создания ЭС на основе вероятностной модели?
10. Расскажите как используется теорема Байеса для управления неопределенностью и логическим выводом в ЭС.
11. Каковы особенности разработки и принципы работы Байесовских сетей доверия? Приведите примеры задач, где сети Байеса могут быть эффективно использованы.
12. Что такое диаграммы влияния и каковы особенности их создания?

ЗАКЛЮЧЕНИЕ

Компьютерные системы являются неотъемлемой частью в работе любой организации, помогая человеку справляться с решением самых разных задач. В то же время эффективность таких систем во многом зависит от используемых в них технологий представления и обработки информации, дальнейшее повышение качественного уровня которых связано не с увеличением вычислительных возможностей ЭВМ, а с наделением информационных систем интеллектуальными способностями на основе новых информационных технологий.

Согласно Норберту Винеру, наиболее перспективные научные исследования находятся в так называемых пограничных областях, которые нельзя точно отнести к той или иной дисциплине. Именно такое научное направление образует искусственный интеллект.

В последнее время на основе методов искусственного интеллекта происходит создание новых технологий обработки информации, включающих синтез и анализ естественного языка, управление знаниями, моделирование естественный процессов обучения и адаптации, координирование процессов принятия решений, планирования действий и т.д. Реализация этого в интеллектуальных компьютерных системах позволяет устранить многие ограничения в решении реальных задач производства, экономики, социальной сферы, от которых зависит национальная конкурентоспособность. Понимание этого, а также необходимость в стимулировании развития этого направления информатики послужило причиной внесения искусственного интеллекта в перечень критических технологий Российской Федерации.

СПИСОК ИСПОЛЬЗОВАННОЙ И РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная

1. Андрейчиков, А.В. Интеллектуальные информационные системы: учеб. для вузов / А.В. Андрейчиков, О.Н. Андрейчикова. – М.: Финансы и статистика, 2004. – 424 с.

2. Гладков, Л.А. Методы генетического поиска / Л.А. Гладков, Л.А. Зинченко, В.В. Курейчик, В.М. Курейчик, Б.К. Лебедев, Е.В. Нужнов, С.Н. Сорокин. - Таганрог: Изд-во ТРТУ, 2002. – 122 с.

3. Гаврилова, Т.А. Базы знаний интеллектуальных систем: Учеб. пособие для вузов / Т.А. Гаврилова, В. Ф. Хорошевский. – СПб.: Питер, 2000. – 382с.

4. Джексон, П. Введение в экспертные системы: [пер. с англ.] / П. Джексон. – М.: Издательский дом «Вильямс», 2001. – 624 с.

5. Круглов, В.В., Искусственные нейронные сети: теория и практика / В.В. Круглов, В.В. Борисов. – М.: Горячая линия – Телеком, 2001. – 382 с.

6. Люггер, Д. Искусственный интеллект: стратегии и методы решения сложных проблем: [пер. с англ.] / Д. Люгер. – М.: Издательский дом «Вильямс», 2003. – 863 с.

7. Терехов, С.А. Введение в байесовы сети / С.А. Терехов // Научная сессия МИФИ – 2003. V Всероссийская научно-техническая конференция «Нейроинформатика - 2003»: лекции по нейроинформатике. -. М.:, 2003. – ч.1. С. 149-186.

Дополнительная

8. Аверченков, В.И. Управление информацией о предметной области на основе онтологий / В.И. Аверченков, П.В. Казаков // Вестник БГТУ. - Брянск; 2006. - №4. - С. 53-57.

9. Васильев, В.И. Интеллектуальные системы управления с использованием генетических алгоритмов / В.И. Васильев, Б.Г. Ильясов // Информационные технологии (Приложение). - 2000. - №12. - 24 с.

10. Искусственный интеллект: В 3 кн. Кн.1. Системы общения и экспертные системы: Справ. / под ред. Э.В. Попова. – М.: Радио и связь, 1990. – 464 с.

11. Слейгл, Д. Искусственный интеллект. Подход на основе эвристического программирования: [пер. с англ.] / Д. Слейгл. – М.: Мир, 1973. – 320 с.

12. Осуга С. Обработка знаний: [пер. с япон.] / С. Осуга. - М.: Мир, 1989. - 293 с.

13. Уссерман, Ф. Нейрокомпьютерная техника: теория и практика / Ф. Уссерман Ф. – М.: Мир, 1992. – 237с.

14. Уэно Х. Представление и использование знаний: [пер. с япон.] / Х. Уэно, М. Исидзука. - М.: Мир, 1990. - 241 с.

ОГЛАВЛЕНИЕ

Учебное издание

ПАВЕЛ ВАЛЕРЬЕВИЧ КАЗАКОВ

ВИТАЛИЙ АЛЕКСАНДРОВИЧ ШКАБЕРИН

ОСНОВЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Редактор Т.И. Королева

Компьютерный набор П.В. Казаков

Темплан 2007 г., п. 54

Подписано в печать 28.11.07 Формат 60x84 1/16 Бумага офсетная Офсетная печать
Усл. печ. л. 11,39 Уч.-изд. л. 11,39 Тираж 140 экз. Заказ

Брянский государственный технический университет,
241035, Брянск, бульвар им. 50 - летия Октября, 7, БГТУ, (4832) 58-82-49.
Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16.