МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А. Г. Зыков, В. И. Поляков

АЛГОРИТМЫ КОНСТРУКТОРСКОГО ПРОЕКТИРОВАНИЯ ЭВМ

Учебное пособие по дисциплине «Конструкторско-технологическое обеспечение производства ЭВМ»

Санкт-Петербург

2014

Зыков А.Г., Поляков В.И. Алгоритмы конструкторского проектирования ЭВМ. – СПб: Университет ИТМО, 2014. – 136 с.

Рассмотрены вопросы проектирования технических объектов. Представлены базовые понятия и определения в области систем автоматизированного проектирования, приведена информация по их классификации и видах обеспечения. Даны общие сведения о математическом обеспечении САПР, рассмотрены основные понятия теории множеств и теории графов. Особое внимание в пособии уделяется задачам автоматизированного проектирования электронных средств, подробно рассмотрены методы и алгоритмы решения задач конструкторского проектирования. Приведены элементы теории сложности алгоритмов. Рассмотрен графо-теоретический подход к синтезу топологии. Описаны эволюционные алгоритмы оптимизации.

Учебное пособие предназначено для бакалавров, обучающихся по направлению "Информатика и вычислительная техника", а также для студентов других технических направлений и специальностей.

Рекомендовано к печати Ученым советом факультета Компьютерных технологи и управления, протокол №10 от 11 ноября 2014 г.

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

СОДЕРЖАНИЕ

Введение

Проектирование современных ЭВМ любого назначения представляет собой сложный процесс создания широкого спектра конструкторской документации (КД) на элементы (БИС, СБИС), функциональные узлы, блоки, устройства и ЭВМ в целом, предназначенной для последующего изготовления и эксплуатации. Характер решаемых задач в процессе разработки КД самый разнообразный: от задач по обеспечению механической прочности, надежности и тепловых режимов в конструкциях до задач по обеспечению заданного быстродействия, помехоустойчивости и электромагнитной совместимости.

Проектирование электронной части ЭВМ имеет свои характерные особенности. Они заключаются в том, что электронная часть предназначена для выполнения главной функции ЭВМ, а именно, для обработки информации, обмена и получения результатов вычислений по заданным алгоритмам и программам. Поэтому проектирование электронной части связано с решением широкого спектра сложных специфических задач по выбору оптимальных параметров логических элементов, по компоновке и оптимальному выбору параметров конструкций ЭВМ, по обеспечению быстродействия и помехоустойчивости линий связи в общей системе межсоединений и многие другие. Все задачи взаимосвязаны и требуют для своего решения разработки соответствующих методов, правил, принципов и критериев конструирования.

Если при механическом и тепловом конструировании ЭВМ любого поколения характерны главным образом традиционные методы решения задач, то при конструировании электронной части имеет место постоянное изменение и усложнение решаемых задач. Сначала определяющей задачей было конструирование транзисторов и транзисторных схем (II-е поколение), затем, с появлением интегральных схем (ИС) (III-е поколение), определяющей задачей стала компоновка узлов и блоков на ИС и конструирование линий связи. Переход к широкому применению в ЭВМ БИС и СБИС (IV-е, V-е поколение) привел к существенному изменению принципов компоновки и появлению новых методов обработки информации, что не могло не отразится на изменении методов конструирования и компоновки электронной части ЭВМ. При этом задача обеспечения заданного быстродействия, помехоустойчивости и помехозащищенности устройств сохраняла свою определяющую роль и значимость. Таким образом, в рамках проектирования электронной части ЭВМ шел процесс постепенного формирования самостоятельного направления, именуемого сейчас как “электронное конструирование”.

Актуальность задач нового направления обусловлена рядом причин: значительным повышением быстродействия элементной базы и необходимостью повышения быстродействия устройств и производительности ЭВМ в целом; существенным возрастанием влияния конструкции на быстродействие и помехозащищенность ЭВМ; резким снижением амплитуды и мощности рабочих сигналов логических элементов ИС и БИС, с одной стороны, и повышением уровня внешних помех, с другой.

Прогресс в вычислительной технике немыслим без существенных достижений в области электронного конструирования элементов и устройств, являющихся технической основой построения ЭВМ любого класса. Обеспечение быстродействия и помехозащищенности элементов, узлов и устройств ЭВМ становится сегодня первоочередной и наиболее важной частью конструирования ЭВМ в целом. Особенно остро эта проблема стоит при переходе к сверхскоростным логическим элементам БИС и СБИС.

Проектирование электронного оборудования сегодня невозможно без использования систем автоматизированного проектирования (САПР). Основными причинами создания автоматизированных систем проектирования являются следующие:

1. Разрозненность отдельных методов автоматизации (подготовка документации, чертежей, моделирование технических систем, оптимизация параметров, организация экспертиз, обработка результатов, принятие решений при многокритериальной постановке задач), отсутствие методологического единства, не позволяющие создать эффективную систему проектирования.

2. Выполнение большого объема работ в сжатые сроки.

3. Повышение требований к качеству проектов (изделий, машин и механизмов).

Удовлетворить эти противоречивые требования с помощью простого увеличения численности проектировщиков нельзя, так как возможность параллельного проведения проектных работ ограничена.

Цель САПР – это повышение качества проектов, снижение материальных затрат, сокращение сроков проектирования и ликвидация тенденции к росту числа проектировщиков, а также повышение производительности их труда.

Для САПР характерно системное использование ЭВМ при рациональном распределении функций между человеком и ЭВМ. С помощью ЭВМ решаются задачи, поддающиеся формализации.

Предметом САПР являются формализация проектных процедур, структурирование и типизация процессов проектирования, постановка, модели, методы и алгоритмы решения проектных задач, способы построения технических средств, создания языков, описания программ, банков данных, а также вопросы их объединения в единую проектирующую систему.

Для создания САПР необходимы:

1. совершенствование проектирования на основе применения математических методов и средств вычислительной техники;

2. автоматизация процесса поиска, обработки и выдачи информации;

3. использование методов оптимизации и многовариантного проектирования;

4. применение математических моделей проектируемых объектов;

5. создание банков данных, содержащих сведения справочного характера;

6. повышение качества оформления проектной документации;

7. увеличение творческой доли труда проектировщиков за счет автоматизации нетворческих работ;

8. унификация и стандартизация методов проектирования;

9. подготовка и переподготовка специалистов в области САПР;

10. взаимодействие САПР с автоматизированными системами различного уровня.

Основными достоинствами САПР являются следующие:

1. *Более быстрое выполнение чертежей.* Конструктор, использующий САПР, выполняет чертежи в 3 раза быстрее, чем традиционно.

2. *Повышение точности выполнения чертежей.* Точность чертежа, выполненного вручную, определяется остротой зрения конструктора и толщиной грифеля карандаша. На чертеже, выполненном с помощью САПР, любое место точки определено точно, а для более детального просмотра его элементов любая часть чертежа может быть увеличена.

3. *Повышение качества выполнения чертежей.* Качество изображения на обычном чертеже полностью зависит от мастерства конструктора, тогда как плоттер САПР рисует линии и тексты независимо от индивидуальных способностей человека.

4. *Возможность многократного использования чертежа.* Память ЭВМ хранит библиотеку символов, стандартов, геометрических форм. В состав чертежа входит ряд компонентов, имеющих одинаковую форму. Запоминание этих компонентов и многократное их использование позволяют повысить эффективность проектирования.

5. *Специальные чертежные средства.* Исследование объекта в трехмерном пространстве, в динамике.

6. *Ускорение расчетов и анализа при проектировании.* Разнообразное и математическое обеспечение позволяют произвести расчет заранее.

7. *Высокий уровень проектирования.* Средства ЭВМ позволяют оптимизировать объект проектирования, перебрать варианты.

8. *Сокращение затрат на усовершенствование.* Средства имитации и анализа в САПР позволяют усовершенствовать прототип.

9. *Интеграция проектирования* с другими видами деятельности: автоматизированная система научных исследований (АСНИ); автоматизированная система технологической подготовки производства (АСТПП); гибкое автоматизированное производство (ГАП); изготовление и сборка ЭВМ с помощью роботов; гибкие производственные системы (ГПС); средства автоматизированного тестирования.

1. Автоматизированное проектирование ЭВМ

1.1. Этапы жизненного цикла промышленных изделий

Жизненный цикл промышленных изделий включает ряд этапов, начиная от зарождения идеи нового продукта до утилизации по окончании срока его использования. К основные этапам жизненного цикла промышленной продукции относятся этапы проектирования, технологической подготовки производства (ТПП), собственно производства, реализации продукции, эксплуатации и, наконец, утилизации.

На всех этапах жизненного цикла изделий имеются свои целевые установки. При этом участники жизненного цикла стремятся достичь поставленных целей с максимальной эффективностью. На этапах проектирования, ТПП и производства нужно обеспечить выполнение технического задания (ТЗ) при заданной степени надежности изделия и *минимизации* материальных и временных затрат, что необходимо для достижения успеха в конкурентной борьбе в условиях рыночной экономики. Понятие эффективности включает в себя не только снижение себестоимости продукции и сокращение сроков проектирования и производства, но и обеспечение удобства освоения и снижения затрат на будущую эксплуатацию изделий. Особую важность требования *удобства эксплуатации* имеют для сложной техники, например, в таких отраслях, как авиа-, ракето- или автомобилестроение.

Достижение поставленных целей на современных предприятиях, выпускающих сложные промышленные изделия, оказывается невозможным без широкого использования *автоматизированных систем* (АС), основанных на применении компьютеров и предназначенных для создания, переработки и использования всей необходимой информации о свойствах изделий и сопровождающих процессов. Специфика задач, решаемых на различных этапах жизненного цикла изделий, обусловливает разнообразие применяемых АС.

1.2. САПР ЭВМ и их место среди других автоматизированных систем

Основные типы автоматизированных систем с их привязкой к тем или иным этапам жизненного цикла изделий указаны на рис. 1.1.

Автоматизация проектирования осуществляется САПР. Принято выделять в САПР радиоэлектронной отрасли промышленности системы функционального, конструкторского и технологического проектирования. Первые из них называют системами расчетов и инженерного анализа, или системами *CAE* (*Computer Aided Engineering*). Системы конструкторского проектирования называют системами *CAD* (*Computer Aided Design*). Проектирование технологических процессов составляет часть технологической подготовки производства и выполняется в системах *CAM* (*Computer Aided Manufacturing*). Функции координации работы систем *CAE*/*CAD*/*CAM*, управления проектными данными и проектированием возложены на систему управления проектными данными *PDM* (*Product Data Management*).

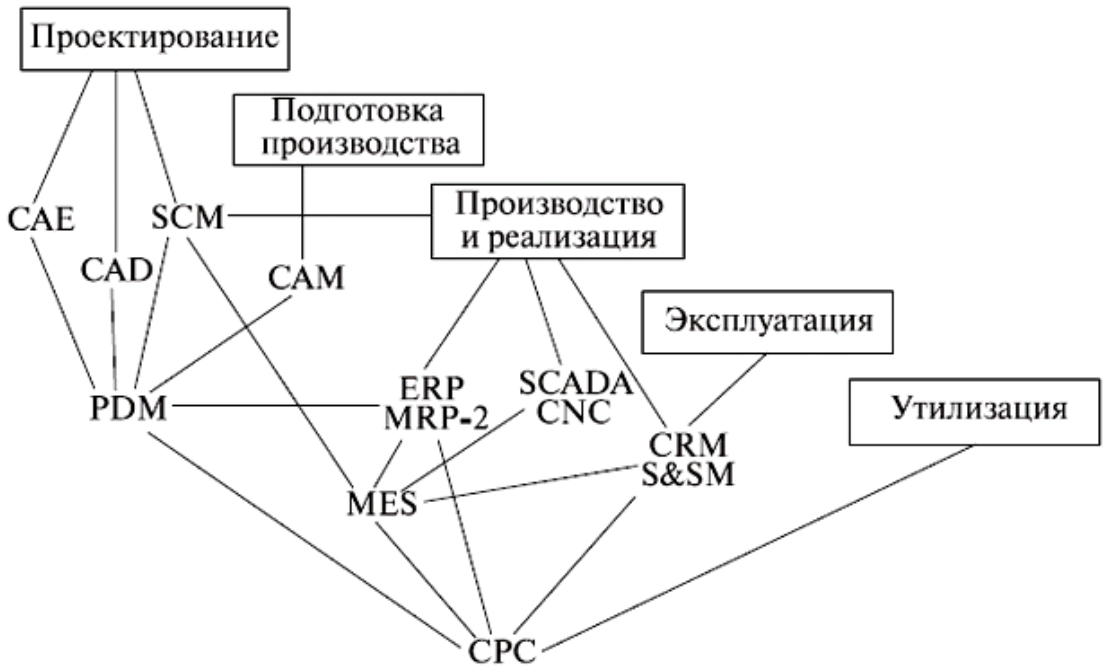


Рисунок 1.1. Этапы жизненного цикла изделий и используемые АС

Уже на стадии проектирования требуются услуги системы управления цепочками поставок *SCM* (*Supply Chain Management*), иногда называемой *CSM* (*Component Supplier Management*). На этапе производства эта система управляет поставками необходимых материалов и комплектующих.

Информационная поддержка этапа производства продукции осуществляется автоматизированными системами управления предприятием (АСУП) и автоматизированными системами управления технологическими процессами (АСУТП). К АСУП относятся системы планирования и управления предприятием *ERP* (*Enterprise Resource Planning*), планирования производства и требований к материалам *MRP-2* (*Manufacturing Requirement Planning*), производственная исполнительная система *MES* (*Manufacturing Execution Systems*), а также *SCM* и система управления взаимоотношениями с заказчиками *CRM* (*Customer Requirement Management*).

Наиболее развитые системы *ERP* выполняют различные бизнес-функции. Они связаны с планированием производства, закупками, сбытом продукции, анализом перспектив маркетинга, управлением финансами, персоналом, складским хозяйством, учетом основных фондов и т. п. Системы *MRP*-2 ориентированы, главным образом, на бизнес-функции, непосредственно связанные с производством, а системы *MES* - на решение оперативных задач управления проектированием, производством и маркетингом.

На этапе реализации продукции выполняются функции управления отношениями с заказчиками и покупателями, проводится анализ рыночной ситуации, определяются перспективы спроса на планируемые изделия. Эти функции осуществляет система *CRM*. Маркетинговые задачи иногда возлагаются на систему *S&SM* (*Sales and Service Management)*, которая, кроме того, используется для решения проблем обслуживания изделий. На этапе эксплуатации применяют

также специализированные компьютерные системы, занятые вопросами ремонта, контроля и диагностики эксплуатируемых систем.

Автоматизированные системы управления технологическими процессами контролируют и используют данные, характеризующие состояние технологического оборудования и протекание технологических процессов. Именно их чаще всего называют системами промышленной автоматизации.

Для выполнения диспетчерских функций (сбор и обработка данных о состоянии оборудования и технологических процессов) и разработки программного обеспечения (ПО) для встроенного оборудования в состав АСУТП вводят систему *SCADA* (*Supervisory Control and Data Acquisition*). Непосредственное программное управление технологическим оборудованием осуществляют с помощью системы *CNC* (*Computer Numerical Control*) на базе контроллеров (специализированных компьютеров, называемых промышленными), которые встроены в технологическое оборудование.

В последнее время усилия многих компаний, производящих программно-аппаратные средства АС, направлены на создание систем электронного бизнеса (*E-Commerce)*. Задачи, решаемые системами *E-Commerce*, сводятся не только к организации на сайтах Internet витрин товаров и услуг. Они объединяют в едином информационном пространстве запросы заказчиков и данные о возможностях множества организаций, специализирующихся на предоставлении различных услуг и выполнении тех или иных процедур и операций по проектированию, изготовлению, поставкам заказанных изделий. Такие системы *E-Commerce* называют системами управления данными в интегрированном информационном пространстве *CPC* (*Collaborative Product Commerce*) или *PLM* (*Product Lifecycle Management*). Проектирование непосредственно под заказ позволяет добиться наилучших параметров создаваемой продукции, а оптимальный выбор исполнителей и цепочек поставок ведет к минимизации времени и стоимости выполнения заказа. Характерная особенность *CPC* - обеспечение взаимодействия многих предприятий, т.е. технология *CPC* является основой, интегрирующей информационное пространство, в котором функционируют *САПР, ERP, PDM, SCM, CRM* и другие АС разных предприятий.

1.3. Проектирование технического объекта

Сущность процесса *проектирования* ЭВМ заключается в разработке конструкций и технологических процессов производства новых электронных средств, которые должны с минимальными затратами и максимальной эффективностью выполнять предписанные им функции в требуемых условиях.

Проектирование любого технического объекта - создание, преобразование и представление в принятой форме образа этого еще не существующего объекта. Образ объекта или его составных частей может создаваться в воображении человека в результате творческого процесса или генерироваться в соответствии с некоторыми алгоритмами в процессе взаимодействия человека и ЭВМ. В любом случае инженерное проектирование начинается при наличии выраженной потребности общества в некоторых технических объектах, которыми могут быть

объекты производства, промышленные изделия или процессы. Проектирование включает в себя разработку технического предложения и (или) ТЗ, отражающих эти потребности, и реализацию ТЗ в виде проектной документации.

Обычно ТЗ представляют в виде некоторых документов, и оно является исходным описанием объекта. Результатом проектирования, как правило, служит полный комплект документации, содержащий достаточные сведения для изготовления объекта в заданных условиях. Эта документация и есть проект, точнее, окончательное описание объекта. Следовательно, проектирование - процесс, заключающийся в получении и преобразовании исходного описания объекта в окончательное описание на основе выполнения комплекса работ исследовательского, расчетного и конструкторского характеров.

Проектирование сложных объектов основано на применении идей и принципов, изложенных в ряде теорий и подходов. Наиболее общим подходом является системный подход, идеями которого пронизаны различные методики проектирования сложных систем. В результате проектирования создаются новые, более совершенные электронные средства, отличающиеся от своих аналогов и прототипов более высокой эффективностью за счет использования новых физических явлений и принципов функционирования, более совершенной элементной базы и структуры, улучшенных конструкций и прогрессивных технологических процессов.

По степени новизны проектируемых изделий различают следующие задачи проектирования:

- частичная модернизация существующей элементной базы (изменение его параметров, структуры и конструкции), обеспечивающая сравнительно небольшое (несколько десятков процентов) улучшение одного или нескольких показателей качества для оптимального решения тех же или новых задач;
- существенная модернизация, которая предполагает значительное улучшение (в несколько раз) показателей качества;
- создание новых ЭВМ, основанных на новых принципах действия, конструирования и производства для резкого увеличения (на несколько порядков) показателей качества при решении тех же или существенно новых задач.

1.4. Этапы проектирования сложных систем

Проектирование является сложным многоэтапным процессом, в котором могут принимать участие большие коллективы специалистов, целые институты и научно-производственные объединения, а также организации заказчиков, которым предстоит эксплуатировать разработанную аппаратуру.

Рассмотрим основные этапы проектирования с позиций технологии обработки информации.

Традиционно проектирование сложных технических систем подразделяют на следующие этапы или стадии разработки (рис. 1.2):

Рисунок 1.2. Этапы проектирования сложных систем

Техническое задание. На этапе разработки *технического задания* решаются следующие задачи:

- поиск и выбор необходимой научно-технической информации (о прототипах, патентных данных и т. д.) из соответствующей базы данных. Новая схема (устройство) может либо иметь, либо не иметь аналогов. В случае если аналоги имеются, можно приступить к этапу проектирования устройства (системы). Но, как правило, аналога нет или разрабатываемая система должна превосходить известный аналог, поэтому необходимо проведение этапа НИР;
- анализ выбранной информации и формулировка на его основе технических требований (ТТ) к проектируемому объекту. Оформление ТТ в соответствии с установленными правилами.

На данном этапе проектирования могут быть автоматизированы операции *поиска информации* и оформления документов. Может быть также автоматизирована некоторая часть вспомогательных действий по анализу выбранной информации, например, группировка ее по заданным признакам, выбор наименее или наиболее сопоставимых друг с другом вариантов и т. д.

Кроме того, на этапе разработки ТЗ решаются и оформляются в соответствующих документах, например, следующие вопросы:

- перечисление функций, выполняемых устройством;
- разработка структурной схемы устройства;
- оформление условий работоспособности устройства;
- оформление требований к выходным параметрам;
- определение характеристик отдельных узлов;
- разработка алгоритмов выполняемых операций.

Этап НИР. Этот этап является предварительным проектированием. Это один из самых ответственных этапов. Для решения его задач необходимо использование ЭВМ - так называемых автоматизированных систем научных исследований (АСНИ).

На этапе НИР необходимо решение следующих задач:

1. Формулирование критериев качества и управления;
2. Управление научным экспериментом;
3. Проведение пассивного или/и активного эксперимента с обработкой результатов;
4. Разработка математических моделей и их верификация по экспериментальным данным;
5. Отработка технологических процессов изготовления объектов ЭВМ с целью поиска норм для параметров, обеспечивающих оптимальные выходные показатели качества;
6. Формирование обобщенного критерия качества, включающего в себя все частные показатели качества. Обобщенный критерий принимается далее за целевую функцию при решении задачи оптимизации;
7. Оптимизация. Производится варьирование входных и управляющих параметров технологического процесса в рамках установленных норм (допусков) с целью получения оптимального критерия качества;
8. Поиск принципиальной возможности построения системы;
9. Разработка новых технических средств, в том числе средств контроля и измерений.

В результате проведения НИР выдается техническое предложение (ТП). Хотя этап НИР является самостоятельным этапом, здесь могут использоваться методы, алгоритмы и программы из САПР.

Этап ОКР. Это этап эскизного проектирования. На данном этапе решаются следующие задачи.

1. Разрабатывается эскиз проектируемой системы (устройства) с детальной разработкой ее возможностей, осуществляется поиск и выбор более детальной информации.
2. На основе анализа полученной информации принимают предварительные проектные решения и оформляют первые проектные документы.
3. Для выработки проектных документов производят различные расчеты, содержание, объем и трудоемкость которых зависят от характеристик проектируемого объекта.

Работы этого этапа в наибольшей степени поддаются автоматизации, и их автоматизация дает наибольший технико-экономический эффект за счет оптимизации проектных решений.

Автоматизация указанных работ достигается за счет применения оптимизационных математических методов.

Этап разработки технического проекта объекта. На этом этапе детализируют и уточняют решения, принятые при эскизном проектировании, и создают

новые, более точные проектные документы. Снова производят поиск, выбор и анализ исходной информации (в основном нормативно-технической и технико-экономической). Снова выполняют многочисленные расчеты, но уже по другим, более точным методикам. Эти работы в значительной степени могут быть автоматизированы.

Большинство документов, сформированных на этапах эскизного и технического проектирования, используются только для выполнения рабочего проектирования и не входят в состав рабочей и эксплуатационной документации. Информация, наработанная на рассмотренных стадиях, служит исходной для рабочего проектирования. Это значит, что в условиях автоматизированного проектирования целесообразно создание банков временной информации по проектируемому объекту.

Рабочее проектирование. На стадии рабочего проектирования основным видом выполняемых работ является оформление проектных решений в виде чертежей, спецификаций к ним и эксплуатационной документации на объект.

Современные средства вычислительной техники позволяют полностью автоматизировать оформление чертежей и спецификаций и в определенной степени - формирование эксплуатационной документации.

Если система автоматизации проектирования выполняет выпуск не только рабочего проекта, но и проектирование технологии, тогда целесообразно не изготавливать чертежи и спецификации в традиционном виде, а передавать проектировщикам-технологам информацию на машинных носителях в виде базы данных о проектируемом объекте.

С точки зрения содержания решаемых задач процесс проектирования разбивают на следующие этапы:

- *системотехническое проектирование,* при котором выбираются и формулируются цели проектирования, обосновываются исходные данные и определяются принципы построения системы. При этом формируется структура проектируемого объекта, его составных частей, которыми обычно являются функционально завершенные блоки, определяются энергетические и информационные связи между составными частями. В результате формируются и формулируются частные технические задания на проектирование отдельных составных частей объекта;
- *функциональное проектирование*, применительно к ЭВМ называемое также схемотехническим, имеет целью аппаратурную реализацию составных частей системы (комплексов, устройств, узлов). При этом выбирают элементную базу, принципиальные схемы и оптимизируют параметры (осуществляют структурный и параметрический синтез схем) с точки зрения обеспечения наилучшего функционирования и эффективного производства. При выборе элементной базы и синтезе схем стремятся учитывать конструкторско-технологические требования;
- *конструирование,* называемое также техническим проектированием, решает задачи компоновки схем и размещения элементов и узлов, осуществления печатных и проводных соединений для электронных изделий всех уровней (мо-

дулей, ячеек, блоков, шкафов), а также задачи теплоотвода, электрической прочности, защиты от внешних воздействий и т.п. При этом стремятся оптимизировать принимаемые решения по конструктивно-технологическим, экономическим и эксплуатационным показателям. На этом этапе проектирования разрабатывают техническую документацию, необходимую для изготовления и эксплуатации ЭВМ;

- *технологическая подготовка* производства обеспечивает разработку технологических процессов изготовления отдельных блоков и всей системы в целом. На этом этапе проектирования создается технологическая документация на основе предшествующих результатов.

Каждый этап проектирования сводится к формированию описаний проектируемого изделия, относящихся к различным иерархическим уровням и аспектам его создания и работы. Этапы проектирования состоят из отдельных проектных процедур, которые заканчиваются частным проектным решением.

Типичными для проектирования ЭВМ процедурами являются *анализ* и *синтез* описаний различных уровней и аспектов.

Процедура *анализа* состоит в определении свойств заданного (или выбранного) описания. Примерами такой процедуры могут служить расчет частотных или переходных характеристик электронных схем, определение реакции схемы на заданное воздействие. Анализ позволяет оценить степень удовлетворения проектного решения заданным требованиям и его пригодность (валидация).

Процедура *синтеза* заключается в создании проектного решения (описания) по заданным требованиям, свойствам и ограничениям. Например, широко используются при проектировании ЭВМ процедуры синтеза электронных схем по их заданным характеристикам в частотной или временной области. При этом в процессе синтеза может создаваться структура схемы (структурный синтез) либо могут определяться параметры элементов заданной схемы, обеспечивающие требуемые характеристики (параметрический синтез).

Процедуры анализа и синтеза в процессе проектирования тесно связаны между собой, поскольку обе они направлены на создание приемлемого или оптимального проектного решения.

Типичной проектной процедурой является *оптимизация*, которая приводит к оптимальному (по определенному критерию) проектному решению. Например, широко используется оптимизация параметров электронных схем с целью наилучшего приближения частотных характеристик к заданным. Процедура оптимизации состоит в многократном анализе при целевом изменении параметров схемы до удовлетворительного приближения к заданным характеристикам. Оптимизация обеспечивает создание (синтез) проектного решения, но включает поэтапную оценку характеристик (анализ).

Проектные процедуры состоят из отдельных *проектных операций*. Например, в процессе анализа математических моделей ЭВМ приходится решать дифференциальные и алгебраические уравнения, осуществлять операции с матрицами. Такие операции могут иметь обособленный характер, но в целом они образуют единую проектную процедуру.

Проектные процедуры и операции выполняются в определенной последовательности, называемой *маршрутом проектирования*.

Маршруты проектирования могут начинаться либо с нижних иерархических уровней описаний (восходящее проектирование), либо с верхних (нисходящее проектирование).

Между всеми этапами проектирования существует глубокая взаимосвязь. Так, определение окончательной конструкции и разработка всей технической документации часто не могут быть выполнены до окончания разработки технологии. В процессе конструирования и разработки технологии может потребоваться коррекция принципиальных схем, структуры системы и даже исходных данных. Поэтому процесс проектирования является не только многоэтапным, но и многократно корректируемым по мере его выполнения, т. е. проектирование носит итерационный характер.

В процессе проектирования необходимо не просто создать аппаратуру, которая будет обеспечивать заданное функционирование, но и оптимизировать ее по широкому спектру функциональных, конструкторско-технологических, эксплуатационных и экономических показателей. На отдельных этапах для отдельных частных задач оптимизацию можно осуществить на основе разработанных формальных математических методов.

2. Общие сведения о САПР

2.1. Блочно-иерархический подход к проектированию

Процесс проектирования, осуществляемый полностью человеком, называют *неавтоматизированным*. В настоящее время наибольшее распространение при проектировании сложных объектов получило проектирование, при котором происходит взаимодействие человека и ЭВМ. Такое проектирование называют *автоматизированным*. Система автоматизированного проектирования - это организационно-техническая система, состоящая из комплекса средств автоматизации проектирования, взаимодействующего с подразделениями проектной организации и выполняющая автоматизированное проектирование.

Представления о сложных технических объектах в процессе их проектирования разделяются на аспекты и иерархические уровни. Аспекты характеризуют ту или иную группу родственных свойств объекта. Типичными аспектами в описаниях технических объектов являются: функциональный, конструкторский и технологический. Функциональный аспект отражает физические и информационные процессы, протекающие в объекте при его функционировании. Конструкторский аспект характеризует структуру, расположение в пространстве и форму составных частей объекта. Технологический аспект определяет технологичность, возможности и способы изготовления объекта в заданных условиях.

Разделение описаний проектируемых объектов на иерархические уровни по степени подробности отражения свойств объектов составляет сущность блочно-иерархического подхода к проектированию.

На верхнем уровне используют наименее детализированное представление, отражающее только самые общие черты и особенности проектируемой системы.

На следующих уровнях степень подробности описания возрастает, при этом рассматривают уже отдельные блоки системы, но с учетом воздействий на каждый из них его соседей. Такой подход позволяет на каждом иерархическом уровне формулировать задачи приемлемой сложности, поддающиеся решению с помощью имеющихся средств проектирования. Разбиение на уровни должно быть таким, чтобы документация на блок любого уровня была обозрима и воспринимаема одним человеком.

Другими словами, блочно-иерархический подход есть декомпозиционный подход, который основан на разбиении сложной задачи большой размерности на последовательно и (или) параллельно решаемые группы задач малой размерности, что существенно сокращает требования к используемым вычислительным ресурсам или время решения задач.

В каждом приложении число выделяемых уровней и их наименования могут быть различными. Так, в радиоэлектронике микроуровень часто называют компонентным, макроуровень - схемотехническим. Между схемотехническим и системным уровнями вводят уровень, называемый функционально - логическим. В вычислительной технике системный уровень подразделяют на уровни проектирования ЭВМ и вычислительных сетей (ВС).

Уровни проектирования можно выделять по степени подробности, с какой отражаются свойства проектируемого объекта. Тогда их называют *горизонтальными* уровнями проектирования.

Уровни проектирования можно выделять также по характеру учитываемых свойств объекта. В этом случае их называют *вертикальными* уровнями проектирования. При проектировании устройств вычислительной техники основными вертикальными уровнями являются функциональное (схемное), конструкторское и технологическое проектирования. При проектировании ЭВМ к этим уровням добавляется алгоритмическое (программное) проектирование.

Пример структурирования описания ЭВМ приведен в Табл. 2.1.

Таблица 2.1. Структурированное описание ЭВМ

<table>
<tr><td rowspan="6">Горизонтальные уровни</td><td colspan="4">Вертикальные уровни</td></tr>
<tr><td>Функциональный</td><td>Алгоритмический</td><td>Конструкторский</td><td>Технологический</td></tr>
<tr><td rowspan="2">Системный</td><td rowspan="2">Программирование системы</td><td>Шкаф, стойка</td><td rowspan="2">Принципиальная схема технологического процесса</td></tr>
<tr><td>Панель</td></tr>
<tr><td rowspan="2">Логический</td><td rowspan="2">Программирование модулей</td><td>ТЭЗ</td><td rowspan="2">Маршрутная технология</td></tr>
<tr><td>Модуль</td></tr>
<tr><td>Схемотехнический</td><td rowspan="2">Проектирование микропрограмм</td><td>Кристалл</td><td rowspan="2">Технологические операции</td></tr>
<tr><td>Компонентный</td><td>Ячейка</td></tr>
</table>

Функциональное проектирование связано с разработкой структурных, функциональных и принципиальных схем. При функциональном проектировании определяются основные особенности структуры, принципы функциониро-

вания, важнейшие параметры и характеристики создаваемых объектов. Функциональное проектирование в САПР может быть как восходящим, так и нисходящим.

Восходящее проектирование характеризуется использованием типовых конфигураций компонентов.

Нисходящее проектирование характеризуется стремлением использовать схемотехнические решения, являющиеся наилучшими для конкретного устройства, и связано с разработкой оригинальных принципиальных схем и структур компонентов.

Алгоритмическое проектирование связано с разработкой алгоритмов функционирования ЭВМ и ВС, с созданием их общего системного и прикладного программного обеспечения. Здесь используется нисходящее проектирование.

Конструкторское проектирование включает в себя вопросы конструкторской реализации результатов функционального проектирования, т.е. вопросы выбора форм и материалов оригинальных деталей, выбора типоразмеров унифицированных деталей, пространственного расположения составных частей, обеспечивающего заданные взаимодействия между элементами конструкции. Для решения конструкторских задач характерно восходящее проектирование.

Технологическое проектирование охватывает вопросы реализации результатов конструкторского проектирования, т.е. рассматриваются вопросы создания технологических процессов изготовления изделий.

В настоящее время проектирование сложного оборудования и его элементов и узлов осуществляется на разных предприятиях с помощью различных САПР, в том числе типовых, например САПР проектирования электронной и вычислительной аппаратуры, САПР проектирования электрических машин и т.д.

2.2. Классификация САПР

Классификацию САПР осуществляют по ряду признаков: по приложению, целевому назначению; масштабам (комплексности решаемых задач); характеру базовой подсистемы – ядра САПР.

По приложениям наиболее представительными и широко используемыми являются следующие группы САПР.

1. САПР для радиоэлектроники: системы *ECAD* (*Electronic CAD*) или *EDA* (*Electronic Design Automation*).

2. САПР для применения в отраслях общего машиностроения. Их часто называют машиностроительными САПР или системами *MCAD* (*Mechanical CAD*).

3. САПР в области архитектуры и строительства.

Кроме того, известно большое число специализированных САПР, выделяемых в указанных группах или представляющих самостоятельную ветвь в классификации. Примерами таких систем являются САПР больших интегральных схем; САПР летательных аппаратов; САПР электрических машин и т. п.

По целевому назначению различают САПР или подсистемы САПР, обеспечивающие разные аспекты проектирования. Так, в составе *MCAD* появляются рассмотренные выше *CAE*/*CAD*/*CAM*-системы.

По масштабам различают отдельные программно-методические комплексы САПР, например: комплекс анализа прочности механических изделий в соответствии с методом конечных элементов (МКЭ) или комплекс анализа электронных схем; системы с уникальными архитектурами не только программного (*software*), но и технического (*hardware*) обеспечений.

По характеру базовой подсистемы различают следующие разновидности САПР.

1. САПР на базе подсистемы машинной графики и геометрического моделирования. Эти САПР ориентированы на приложения, где основной процедурой проектирования является конструирование, т. е. определение пространственных форм и взаимного расположения объектов. К этой группе систем относится большинство САПР в области машиностроения, построенных на базе графических ядер.

В настоящее время широко используют унифицированные графические ядра, применяемые более чем в одной САПР (ядра *Parasolid* фирмы EDS Unigraphics и *ACIS* фирмы Intergraph).

2. САПР на базе СУБД. Они ориентированы на приложения, в которых при сравнительно несложных математических расчетах перерабатывается большой объем данных. Такие САПР преимущественно встречаются в технико-экономических приложениях, например, при проектировании бизнес-планов, но они распространены также при проектировании объектов, подобных щитам управления в системах автоматики.

3. САПР на базе конкретного прикладного пакета. Фактически это автономно используемые ПМК, например, имитационного моделирования производственных процессов, расчета прочности по МКЭ, синтеза и анализа систем автоматического управления и т. п. Часто такие САПР относятся к системам *CAE*. Примерами могут служить программы логического проектирования на базе языка *VHDL*, математические пакеты типа *MathCAD*.

4. Комплексные (интегрированные) САПР, состоящие из совокупности подсистем предыдущих видов. Характерными примерами комплексных *САПР* являются *CAE*/*CAD*/*CAM*-системы в машиностроении или САПР БИС. Так, САПР БИС включает в себя СУБД и подсистемы проектирования компонентов, принципиальных, логических и функциональных схем, топологии кристаллов, тестов для проверки годности изделий. Для управления столь сложными системами применяют специализированные системные среды.

2.3. Структура САПР

Как и любая сложная система, САПР состоит из подсистем. Различают *проектирующие* и *обслуживающие* подсистемы.

Проектирующие подсистемы непосредственно выполняют проектные процедуры. Примерами проектирующих подсистем могут служить подсистемы геометрического трехмерного моделирования механических объектов, изготовления конструкторской документации, схемотехнического анализа, трассировки соединений в печатных платах.

Обслуживающие подсистемы обеспечивают функционирование проектирующих подсистем, их совокупность часто называют системной средой (или оболочкой) САПР. Типичными обслуживающими подсистемами являются подсистемы управления проектными данными, подсистемы разработки и сопровождения программного обеспечения *CASE* (*Computer Aided Software Engineering*), обучающие подсистемы для освоения пользователями технологий, реализованных в САПР.

2.4. Виды обеспечения САПР

Структурирование САПР по различным аспектам обусловливает появление *видов обеспечения САПР*. Принято выделять семь видов обеспечения САПР:

- *техническое* (ТО), включающее различные аппаратные средства (ЭВМ, периферийные устройства, сетевое коммутационное оборудование, линии связи, измерительные средства);
- *математическое* (МО), объединяющее математические методы, модели и алгоритмы для выполнения проектирования;
- *программное* (ПО), представляемое компьютерными программами САПР;
- *информационное* (ИО), состоящее из базы данных, СУБД, а также включающее другие данные, которые применяются при проектировании; отметим, что вся совокупность используемых при проектировании данных называется информационным фондом САПР, а база данных вместе с СУБД носит название банка данных;
- *лингвистическое* (ЛО), выражаемое языками общения между проектировщиками и ЭВМ, языками программирования и языками обмена данными между техническими средствами САПР;
- *методическое* (МтО), включающее различные методики проектирования;
- *организационное* (ОО), представляемое штатными расписаниями, должностными инструкциями и другими документами, которые регламентируют работу проектного предприятия.

3. Математическое обеспечение САПР

Математическое обеспечение САПР состоит из математических моделей объектов проектирования, методов и алгоритмов выполнения проектных операций и процедур (рис. 3.1).

В математическом обеспечении САПР можно выделить специальную часть, в значительной мере отражающую специфику объекта проектирования, физические и информационные особенности его функционирования и тесно привязанную к конкретным иерархическим уровням (эта часть охватывает математические модели, методы и алгоритмы их получения, методы и алгоритмы одновариантного анализа, а также большую часть используемых алгоритмов синтеза), и инвариантную часть, включающую в себя методы и алгоритмы, слабо связанные с особенностями математических моделей и используемые на многих иерархи-

ческих уровнях (это методы и алгоритмы многовариантного анализа и параметрической оптимизации).

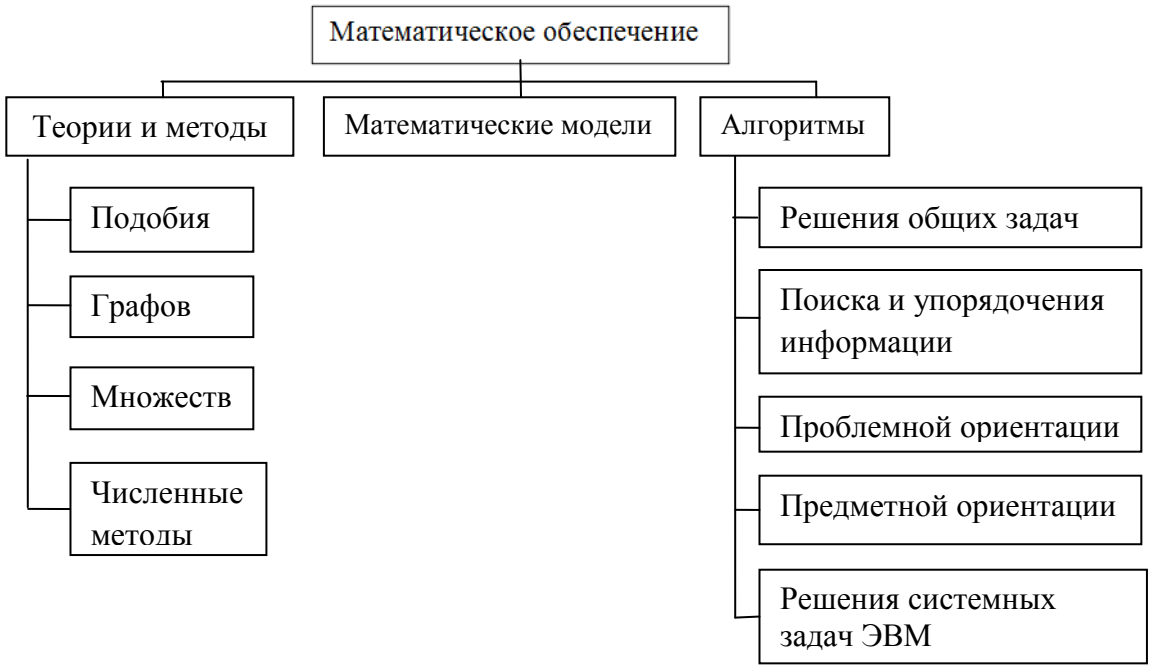


Рисунок 3.1. Структура математического обеспечения САПР

3.1. Требования к математическому обеспечению

Свойства математического обеспечения оказывают существенное, а иногда и определяющее влияние на возможности и показатели САПР.

При выборе и разработке моделей, методов и алгоритмов необходимо учитывать требования, предъявляемые к МО в САПР.

Универсальность. Под универсальностью МО понимается его применимость к широкому классу проектируемых объектов. Одно из отличий расчетных методов в САПР от ручных расчетных методов - высокая степень универсальности. Например, в подсистеме схемотехнического проектирования используются математические модели транзистора, справедливые для любой области работы (активной, насыщения, отсечки, инверсной активной), а методы получения и анализа моделей применимы к любой аналоговой или переключательной схеме на элементах из разрешенного списка; в подсистеме структурного проектирования ЭВМ используются модели и алгоритмы, позволяющие исследовать стационарные и нестационарные процессы переработки информации при произвольных законах обслуживания в устройствах ВС и при произвольных входных потоках.

Высокая степень универсальности МО нужна для того, чтобы САПР была применима к любым или большинству объектов, проектируемых на предприятии.

Алгоритмическая надежность. Методы и алгоритмы, не имеющие строгого обоснования, называют эвристическими. Отсутствие четко сформулированных условий применимости приводит к тому, что эвристические методы могут использоваться некорректно. В результате либо вообще не будет получено ре-

шение (например, из-за отсутствия сходимости), либо оно будет далеким от истинного. Главная неприятность заключается в том, что в распоряжении инженера может не оказаться данных, позволяющих определить, корректны или нет полученные результаты. Следовательно, возможна ситуация, когда неверное решение будет использоваться в дальнейшем как правильное.

Свойство компонента МО давать при его применении в этих условиях правильные результаты называется алгоритмической надежностью. Степень универсальности характеризуется заранее оговоренными ограничениями, а алгоритмическая надежность - ограничениями, заранее не выявленными и, следовательно, не оговоренными.

Количественной оценкой алгоритмической надежности служит вероятность получения правильных результатов при соблюдении оговоренных ограничений на применение метода. Если эта вероятность равна единице или близка к ней, то говорят, что метод алгоритмически надежен.

Применение алгоритмически ненадежных методов в САПР нежелательно, хотя и допустимо в случаях, когда неправильные результаты легко распознаются.

С проблемой алгоритмической надежности тесно связана проблема обусловленности математических моделей и задач. О плохой обусловленности говорят в тех случаях, когда малые погрешности исходных данных приводят к большим погрешностям результатов. На каждом этапе вычислений имеются свои промежуточные исходные данные и результаты, свои источники погрешностей. При плохой обусловленности погрешности могут резко возрасти, что может привести как к снижению точности, так и к росту затрат машинного времени.

Точность. Для большинства компонентов МО важным свойством является точность, определяемая по степени совпадения расчетных и истинных результатов. Алгоритмически надежные методы могут давать различную точность. И лишь в тех случаях, когда точность оказывается хуже предельно допустимых значений или решение вообще невозможно получить, говорят не о точности, а об алгоритмической надежности.

В большинстве случаев решение проектных задач характеризуется:

- совместным использованием многих компонентов МО, что затрудняет определение вклада в общую погрешность каждого из компонентов;
- векторным характером результатов (например, при анализе находят вектор выходных параметров, при оптимизации - координаты экстремальной точки), т.е. результатом решения является значение не отдельного параметра, а многих параметров.

В связи с этим оценка точности производится с помощью специальных вычислительных экспериментов. В этих экспериментах используются специальные задачи, называемые тестовыми. Количественная оценка погрешности результата решения тестовой задачи есть одна из норм вектора относительных погрешностей: m-норма или l-норма, где l - относительная погрешность определения j-го элемента вектора результатов; m - размерность этого вектора.

Затраты машинного времени. Универсальные модели и методы характеризуются сравнительно большим объемом вычислений, растущим с увеличением размерности задач. Поэтому при решении большинства задач в САПР затраты машинного времени $T_м$ значительны. Обычно именно $T_м$ являются главным ограничивающим фактором при попытках повысить сложность проектируемых на ЭВМ объектов и тщательность их исследования. Поэтому требование экономичности по $T_м$ - одно из основных требований к МО САПР.

При использовании в САПР многопроцессорных ВС уменьшить время счета можно с помощью параллельных вычислений. В связи с этим один из показателей экономичности МО - его приспособленность к распараллеливанию вычислительного процесса.

Используемая память. Затраты памяти являются вторым после затрат машинного времени показателем экономичности МО. Они определяются длиной программы и объемом используемых массивов данных. Несмотря на значительное увеличение емкости оперативной памяти в современных ЭВМ, требование экономичности по затратам памяти остается актуальным. Это связано с тем, что в мультипрограммном режиме функционирования ЭВМ задача с запросом большого объема памяти получает более низкий приоритет и в результате время ее пребывания в системе увеличивается.

Улучшить экономичность по затратам оперативной памяти можно путем использования внешней памяти. Однако частые обмены данными между оперативной памятью и внешней могут привести к недопустимому росту $T_м$. Поэтому при больших объемах программ и массивов обрабатываемой информации целесообразно использовать МО, допускающее построение оверлейных программных структур и реализующее принципы диакоптической обработки информации (диакоптика - один из методов расчленения при исследовании сложных систем, которые могут быть представлены в виде блок-схемы или графа с использованием граф-топологического портрета системы как нового источника информации).

3.2. Требования к математическим моделям

Математической моделью (ММ) технического объекта называется совокупность математических объектов (чисел, скалярных переменных, векторов, матриц, графов и т. п.) и связывающих их отношений, отражающие свойства моделируемого технического объекта, интересующие инженера - проектировщика.

К математическим моделям предъявляются требования универсальности, адекватности, точности и экономичности.

Степень универсальности ММ характеризует полноту отображения в модели свойств реального объекта. Математическая модель отражает лишь некоторые свойства объекта.

Адекватность ММ - способность отражать заданные свойства объекта с погрешностью не выше заданной.

Точность ММ оценивается степенью совпадения значений параметров реального объекта и значений тех же параметров, рассчитанных с помощью используемой ММ.

3.3. Классификация математических моделей

Рассмотрим основные признаки, классификации и типы ММ, применяемые в САПР.

По характеру отображаемых свойств объекта ММ делятся на структурные и функциональные.

Структурные ММ предназначены для отображения структурных свойств объекта. Различают структурные ММ топологические и геометрические.

В топологических ММ отображаются состав и взаимосвязи элементов. Их чаще всего применяют для описания объектов, состоящих из большого числа элементов, при решении задач привязки конструктивных элементов к определенным пространственным позициям (например, задачи компоновки оборудования, размещения деталей, трассировки соединений) или к относительным моментам времени (например, при разработке расписаний, технологических процессов). Топологические модели могут иметь форму графов, таблиц (матриц), списков и т.п.

В геометрических ММ отображаются пространственные свойства объектов, в них дополнительно к сведениям о взаимном расположении элементов содержатся сведения о форме деталей. Геометрические ММ могут выражаться совокупностью уравнений линий и поверхностей; совокупностью алгебраических соотношений, описывающих области, составляющие тело объекта; графами и списками, отображающими конструкции из типовых конструктивных элементов, и т.п. Геометрические ММ применяют при решении задач конструирования в машиностроении, приборостроении, радиоэлектронике, для оформления конструкторской документации, при задании исходных данных на разработку технологических процессов изготовления деталей. Используют несколько типов геометрических ММ.

Функциональные ММ предназначены для отображения физических или информационных процессов, протекающих в объекте при его функционировании или изготовлении. Обычно функциональные ММ представляют собой системы уравнений, связывающих фазовые переменные, внутренние, внешние и выходные параметры.

По степени детализации описания в пределах каждого иерархического уровня выделяют полные ММ и макромодели.

Полная модель - эта модель, в которой фигурируют фазовые переменные, характеризующие состояния всех имеющихся межэлементных связей (т.е. состояние всех элементов проектируемого объекта).

Макромодель - ММ, в которой отображаются состояния значительно меньшего числа межэлементных связей, что соответствует описанию объекта при укрупненном выделении элементов.

По способу представления свойств объекта функциональные ММ делятся на аналитические и алгоритмические.

Аналитические ММ представляют собой явные выражения выходных параметров как функций входных и внутренних параметров.

Алгоритмические ММ выражают связи выходных параметров с параметрами внутренними и внешними в форме алгоритма.

Имитационная ММ - это алгоритмическая модель, отражающая поведение исследуемого объекта во времени, при задании внешних воздействий на объект.

3.4. Формализация проектных задач

Формализация проектной задачи является необходимым условием для ее решения на ЭВМ. К формализуемым задачам относятся прежде всего задачи, всегда считавшиеся рутинными, не требующими существенных затрат творческих усилий инженеров. Это процедуры изготовления конструкторской документации (КД) в условиях, когда содержание КД уже полностью определено, но еще не имеет принятой для хранения и дальнейшего использования формы (например, формы чертежей, графиков, схем, алгоритмов, таблиц соединений); процедуры проведения электрических соединений в печатных платах или выполнения фотоформ в полиграфии. Кроме рутинных к формализуемым задачам относится большинство задач анализа проектируемых объектов. Их формализация достигается благодаря развитию теории и методов автоматизированного проектирования, прежде всего моделирования. В то же время есть много проектных задач творческого характера, для которых способы формализации неизвестны. Это задачи, связанные с выбором принципов построения и организации объекта, синтеза схем и конструкций в условиях, когда выбор варианта производится среди неограниченного множества вариантов и не исключается возможность получения новых, ранее неизвестных решений.

Подход к решению задач указанных групп в САПР неодинаков. Полностью формализуемые задачи, составляющие первую группу задач, чаще всего решаются на ЭВМ без вмешательства человека в процесс решения. Частично формализуемые задачи, составляющие вторую группу задач, решаются на ЭВМ при активном участии человека, т.е. имеет место работа с ЭВМ в интерактивном режиме. Наконец, не формализуемые задачи, составляющие третью группу задач, решаются инженером без помощи ЭВМ.

В настоящее время одним из направлений развития математического обеспечения автоматизированного проектирования является разработка методов и алгоритмов синтеза на различных уровнях иерархического проектирования.

3.5. Математические методы описания моделей конструкций ЭВМ

В САПР для каждого иерархического уровня сформулированы основные положения математического моделирования, выбран и развит соответствующий математический аппарат, получены типовые ММ элементов проектируемых объектов, формализованы методы получения и анализа математических моделей систем. Сложность задач проектирования и противоречивость требований высокой точности, полноты и малой трудоемкости анализа обусловливают целесообразность компромиссного удовлетворения этих требований с помощью соответствующего выбора моделей. Это обстоятельство приводит к расширению множества используемых моделей и развитию алгоритмов адаптивного моделирования.

3.3.1. Понятия теории множеств

Математические методы, положенные в основу алгоритмических процессов конструирования ЭВМ, а также процессы организации входной и выходной информации о проектируемом объекте широко используют понятия и символы теории множеств.

Под *множеством* понимают совокупность объектов любой природы, называемых *элементами* данного множества, обладающих каким-либо общим для множества свойством.

В множестве, по определению, все элементы различны, а порядок перечисления элементов множества произвольный.

Существует два основных способа задания множеств: перечисление и описание.

Первый способ состоит в том, что задается и перечисляется полный список элементов, входящих в множество. Например, множество элементов схемы ЭВМ определяется их списком. Данный способ удобно применять только к ограниченному числу конечных множеств.

Второй способ применяется, когда множество нельзя или затруднительно задать с помощью списка (это может относиться к конечным и бесконечным множествам). В таком случае множества определяются свойствами их элементов.

Множество A считается заданным, если указано *свойство* α, которым обладают все элементы, принадлежащие множеству A, и которым элементы, не принадлежащие множеству A, не обладают.

Если A - произвольное множество, а - α некоторое свойство, то запись $A=\{b \in B/\alpha(b)\}$ означает множество тех и только тех элементов $b \in B$, которые обладают свойством α.

При задании множества вторым способом необходимо так задавать свойство, характеризующее элементы множества, чтобы оно было общим, непротиворечивым для всех его элементов.

Как основное понятие теории, понятие множества не подлежит логическому определению.

Элементы множества могут иметь различную природу. Например, можно говорить о множестве микросхем, входящих в определенную конструкцию ЭВМ, или о множестве чертежей, входящих в полный комплект конструкторской документации для производства какого-либо изделия.

Множество можно задавать не только перечислением его элементов, но и с помощью описательного способа, указывающего характерное свойство, которым обладают все элементы этого множества.

Например, если во всем множестве X микросхем электронного блока сложной радиоаппаратуры есть некоторое множество A гибридных интегральных схем, то это можно записать так: $A=\{x \in X: x$ – гибридная ИС$\}$.

Это читается так: множество A состоит из элементов множества X, обладающих тем свойством, что x является гибридной интегральной схемой.

Здесь введено обозначение $\in$ означающее, что объект x является элементом множества X. Если же некоторый объект y не принадлежит множеству X, то это условие записывают в виде $y \notin X$.

В том случае, когда не вызывает сомнения, из какого множества берутся элементы x, принадлежность их к множеству X можно не указывать.

Число элементов множества $\boldsymbol{X} = (x_1, x_2, ..., x_n)$ называют *мощностью* этого множества и обозначают прямыми скобками, например, $|X| = n$.

Если число элементов множества X конечно, то такое множество называют *конечным*. В противном случае множество будет *бесконечным*.

В теории множеств вводится понятие *пустого множества*, множества, в котором не содержится ни одного элемента.

Пустое множество обозначают специальным символом $\varnothing$. Например, если множество X пусто, то пишут $X = \varnothing$.

Последовательность из n элементов множества называют n - строкой. В отличие от обычного множества, где порядок элементов безразличен, в n - строке обязательно задается их определенная последовательность.

Множество X равно множеству Y, если оба эти множества состоят из одних и тех же элементов.

Если множество X полностью содержится во множестве Y и при этом $|X|<|Y|$, то говорят, что множество X является подмножеством множества Y: $X \subset Y$.

Следовательно, мы рассмотрели два соотношения:

- принадлежность $x \in X$;
- включение $X \subset Y$.

Первое определяет связь между множеством и его элементами, а второе - между двумя множествами.

В случае, когда $X \subset Y$ и, одновременно, $Y \subset X$, имеет место равенство $X=Y$, т.е. множества X и Y совпадают.

Символическая запись $X \neq Y$ означает, что множество X не совпадает с множеством Y.

Операции над множествами.

1. *Объединение множеств*.

Объединение множеств X и Y — это множество, состоящее из всех тех и только тех элементов, которые принадлежат хотя бы одному из множеств X или Y, т.е. принадлежат X или принадлежат Y.

Объединение X и Y обозначается через $X \cup Y$.

Формально $z \in X \cup Y \Leftrightarrow z \in X$ или $z \in Y$.

2. *Пересечение множеств*.

Пересечение множеств X и Y — это множество, состоящее из всех тех и только тех элементов, которые принадлежат как множеству X, так и множеству Y.

Пересечение множеств обозначается $X \cap Y$.

Формально $z \in X \cap Y \Leftrightarrow z \in X$ и $z \in Y$.

3. *Разность множеств*.

Разность множеств определена только для двух множеств. Разностью множеств X и Y называется множество, состоящее из всех тех и только тех элементов, которые принадлежат X и не принадлежат Y.

Обозначается: $X\backslash Y$.

Формально: $z \in X\backslash Y \Leftrightarrow z \in X$ и $x \notin Y$.

3.5.2. Элементы теории графов

Теорию графов применяют для решения таких задач, как анализ электронных схем, разбиение и размещение электронных элементов, проектирование проводного и печатного монтажа, сетевое планирование и многих других. Это объясняется тем, что использование графов сокращает объем вычислений по сравнению с другими методами и, сохраняя наглядность описания объектов (конструкций), позволяет строить компактные и удобные для реализации на ЭВМ алгоритмы преобразований и оптимизации.

Понятие графа $G(X,U)$ опирается на понятие множества.

Под *абстрактным графом* или просто графом $G(X,U)$ понимают совокупность непустого множества X и изолированного от него подмножества U (возможно, пустого), представляющего собой множество всех упорядоченных пар (x_i, x_j), где $x_i, x_j \in X$. Элементы множеств X и U называют, соответственно, *вершинами* и *дугами* (*ребрами*) графа. Следовательно, *граф* - это множество X всех вершин x_i, связи между которыми определены множеством ребер U.

Геометрически граф можно представить в виде множества точек $X=\{x_j\}$ (j=1, 2,..., n) в n-мерном евклидовом пространстве E^n и множества простых, направленных самонепересекающихся кривых $U=\{u_k\}$(k=1, 2, ..., r), соединяющих x_i и x_j, которые находятся в некотором отношении друг к другу.

То, что элемент $x_i \in X$ находится в отношении T_{ij} к элементу $x_j \in X$, отображается на графе соединением элементов x_i и x_j линией со стрелкой в направлении от x_i к x_j. Такие соединения вершин графа с указанием направления называют *ориентированными ребрами* или *дугами* и записывают так: $u_k=(x_i, x_j) \sim x_i T_{ij} x_j$.

Граф, в котором все вершины соединены дугами, называют *ориентированным*, *орграфом*, *направленным* или *несимметрическим графом*.

Аналитически любой ориентированный граф описывается системой алгебраических уравнений, связывающих параметры $x_i \in X$, и наоборот, любая система алгебраических уравнений может быть представлена в виде направленного графа.

Рассмотрим ориентированный граф, показанный на рис. 3.2.

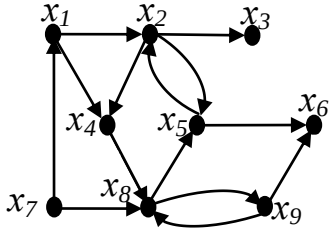


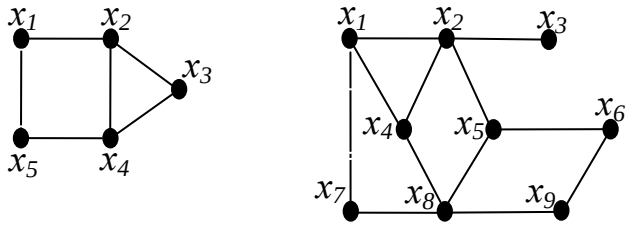
Рисунок 3.2. Ориентированный граф

Этот граф определяет следующую систему уравнений:

$$\begin{cases} x_1 = T_{71}x_7; \\ x_2 = T_{12}x_1 + T_{52}x_5; \\ x_3 = T_{23}x_2; \\ x_4 = T_{14}x_1 + T_{24}x_2; \\ x_5 = T_{25}x_2 + T_{85}x_8; \\ x_6 = T_{56}x_5 + T_{96}x_9\ ; \\ x_8 = T_{78}x_7 + T_{48}x_4 + T_{98}x_9; \\ x_9 = T_{89}x_8. \end{cases}$$

Граф, в котором для любых двух вершин x_i, $x_j \in X$ справедливо выражение $T_{ij} = T_{ji}$, называют *неориентированным, неографом, ненаправленным* или *симметрическим графом*.

В таком графе вершины x_i, и x_j соединены ненаправленной кривой, называемой *ребром графа* (рис. 3.3).



а *б*

Рисунок 3.3. Неориентированные графы

Две вершины x_i, $x_j \in X$ считаются *смежными*, если они определяют ребро (дугу) и, соответственно, два различных ребра (дуги) *смежны*, если они имеют общую вершину

Иными словами, вершина x_j смежна x_i, если $x_j \in Гx_i$,, где $Гx_i$ – *отображение* x_i на множество X.

В связи с тем, что отображение $Гx_i$ представляет собой совокупность всех вершин графа $x_j \in X$, смежных x_i получаем еще один способ задания графа.

Граф задан, если задано непустое множество X и отображение $Г$ множества X в X. Обозначим его $G(X, Г)$. При геометрической реализации такого графа каждую вершину $x_i \in X$ соединяют со всеми вершинами $x_j \in Гx_i$.

Например, для графа $G(X, Г)$ (рис. 3.3 (*а*)) можно записать:

$X=\{x_j\}$ (j=1, 2, ..., 5); $Гx_1=x_2,x_5$; $Гx_2=x_1,x_3,x_4$; $Гx_3=x_2,x_4$; $Гx_4=x_2,x_3,x_5$; $Гx_5=x_1,x_4$.

Вершина x_i *инцидента* ребру (дуге) u_j если она является началом или концом ребра (дуги). Аналогично утверждение, что ребро (дуга) u_j *инцидентно* вершине x_i, если оно входит или выходит из этой вершины.

Число ребер (дуг), инцидентных некоторой вершине x_i, называют *локальной степенью вершины* и обозначают $\rho(x_i)$.

Для графа (рис. 3.3, б) можно записать:

$\rho(x_1) = \rho(x_4) = \rho(x_5) = 3$; $\rho(x_2) = \rho(x_8) = 4$; $\rho(x_3) = 1$; $\rho(x_6) = \rho(x_7) = \rho(x_9) = 2$.

Учитывая, что каждое ребро неориентированного графа инцидентно двум вершинам, получим выражение, связывающее число ребер графа со степенями вершин

$$\sum_{i=1}^{n}\rho(x_i)=2k,\ \text{где}\ n=|X|-\text{число вершин графа},\ k=|U|-\text{число ребер графа}.$$

Из этого выражения следует, что число вершин с нечетной степенью в графе четное, так как при опускании всех вершин x_i с четными степенями $\rho(x_i)$ сумма слева остается четной.

Вершину, не инцидентную никакому ребру (дуге), называют *изолированной*.

Граф, состоящий только из изолированных вершин, называют *нуль – графом* и обозначают G_0.

При использовании графов для анализа электронных устройств в отдельных случаях целесообразно введение связи вершины самой с собой, т.е.

$u_k=(x_i, x_i) \sim x_i T_{ii} x_i$.

Такую связь называют *петлей*.

Граф называют *конечным*, если число его ребер конечно и *бесконечным*, если число его ребер бесконечно.

Граф называют *однородным степени t*, если степени всех его вершин t.

Примеры бесконечных однородных графов приведены на рис. 3.4.

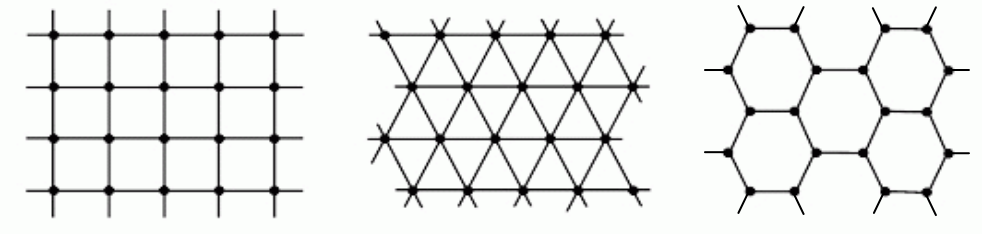


Рисунок 3.4. Бесконечные однородные графы

Бесконечные однородные графы находят широкое применение в задачах трассировки печатных соединений, т.к. их использование позволяет разбивать коммутационное поле печатных плат на элементарные ячейки одинаковой формы.

Существенной характеристикой графа является его *связность*.

Граф, любая пара вершин которого связана, называют *связным графом*. В связном графе, перемещаясь по ребрам из вершины в вершину, можно попасть в каждую вершину. Граф, состоящий из отдельных фрагментов, называют *несвязным*, состоящим из отдельных *компонент связности*.

Если граф несвязный, то множество его вершин можно единственным образом разделить на непересекающиеся подмножества, каждое из которых содержит все связанные между собой вершины и вместе с инцидентными им ребрами образует связный *подграф*.

Таким образом, несвязный граф представляет собой совокупность отдельных частей (подграфов), называемых *компонентами связности*.

Граф, все вершины которого попарно смежны, называют *сильносвязным* или *полным графом* и обозначают K_n, где n – число вершин графа (рис. 3.5)

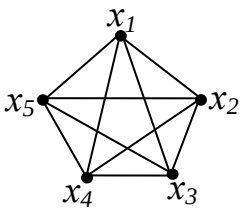


Рисунок 3.5. Полный граф K_5

Последовательность ребер $u_k \in U$ заданных парами вершин вида (x_0, x_1) $(x_1, x_2)...(x_{l-1}, x_l)$, в которой любые два соседних ребра смежные, называется *маршрутом*.

Число ребер в маршруте определяет его длину. Если все ребра в маршруте различны, то такой маршрут является *цепью*.

Если в цепи нет повторяющихся вершин кроме соседних, то такая цепь называется *простой*.

Циклом называют последовательность ребер $u_1=(x_1, x_i)$, ..., $u_k=(x_j, x_1)$, при которой в результате обхода вершин графа по этим ребрам возвращаются в исходную вершину x_1.

Каждое ребро графа встречается в *цикле* не более одного раза, в то время как вершины могут повторяться и несколько раз.

Цикл считают *простым*, если в нем нет повторяющихся вершин, и *сложным*, если такие имеются.

Цикл называют *элементарным*, если он не содержит в себе никаких других циклов.

Цикл считают *минимальным*, если он включает минимальное число ребер, и *максимальным*, если он содержит максимальное число ребер графа.

Таким образом, цикл - это замкнутая цепь.

Большое значение в задачах конструкторского проектирования ЭВМ имеют *эйлеровы* и *гамильтоновы циклы*.

Эйлеров цикл – это цикл, в котором содержатся все ребра графа.

Граф, имеющий такой цикл, называют *эйлеровым* графом. На рисунке 3.6. изображен граф «Сабли Магомета». Это эйлеров граф, эйлеров цикл $x_1\ x_6\ x_9\ x_{10}\ x_{11}\ x_7\ x_4\ x_2\ x_6\ x_8\ x_9\ x_3\ x_{10}\ x_5\ x_4\ x_3\ x_2\ x_1$.

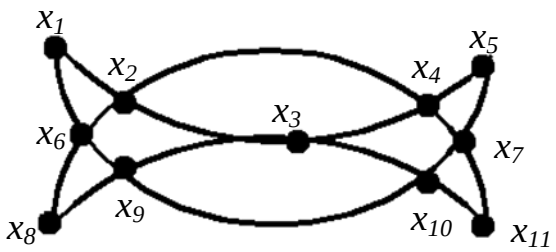


Рисунок 3.6. Пример эйлерова графа

Необходимым и достаточным условием наличия в конечном связном графе эйлерова цикла является четность степеней всех его вершин (*теорема Эйлера*).

Цикл называют *гамильтоновым*, если он проходит через каждую вершину один раз.

Известно (*теорема Оре – Дирака*), что граф имеет гамильтонов цикл, если сумма локальных степеней двух любых вершин графа больше или равна числу вершин графа, т.е. $\forall x_i\, x_j \in X\ [\rho(x_i) + \rho(x_j) \geq n]$.

Из теоремы следует *результат Дирака*: граф имеет гамильтонов цикл, если локальные степени любых его вершин $\forall x_i, x_j \in X\ [\rho(x_i) \geq n/2]$.

Граф с гамильтоновым циклом изображен на рисунке 3.7. Гамильтонов цикл обозначен штрих-пунктирной линией.

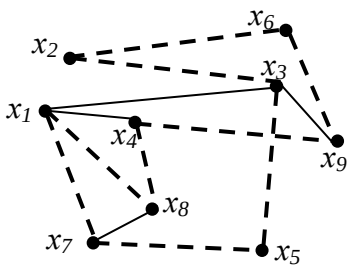
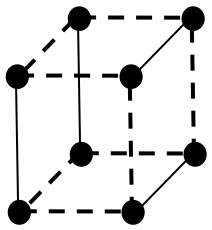


Рисунок 3.7. Пример графа с гамильтоновым циклом

Несмотря на некоторое сходство в определениях эйлеровых и гамильтоновых *циклов*, используемые для их отыскания методы имеют мало что общего. *Критерий существования* эйлерова цикла был установлен просто, для гамильтонова цикла такого общего правила не известно. Более того, иногда для конкретного графа бывает затруднительно сказать, имеет ли он такой цикл или нет. Существуют лишь частные критерии наличия в графе гамильтонова цикла. То, что критерии существования гамильтонова цикла в графе являются достаточными, но не необходимыми можно проиллюстрировать с помощью графа-куба (рис. 3.8).



$\forall x_i, x_j \in X\ [\rho(x_i) + \rho(x_j) = 6 < 8]$
$\forall x_i, x_j \in X\ [\rho(x_i) = 3 < 4]$.

Рисунок 3.8. Трехсвязный кубический граф

При размещении графа в виде *геометрической фигуры с*уществует большая свобода в размещении вершин графа в пространстве и выборе формы соединяющих их ребер (дуг). Следовательно, один и тот же граф может иметь различную геометрическую реализацию.

Два графа G и G' *изоморфны*, если они имеют одинаковое число вершин и если каждой паре вершин, соединенных ребром (дугой), в одном графе, соответствует такая же пара вершин, соединенных ребром (дугой), в другом графе (рис. 3.9).

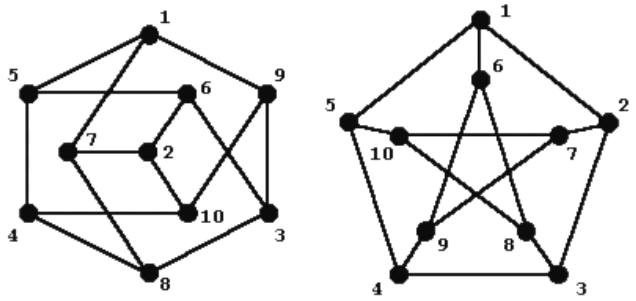


Рисунок 3.9. Пример изоморфных графов

Решение ряда задач конструкторского проектирования ЭВМ связано с изоморфными преобразованиями графа, т.е. с построением графа, изоморфного заданному. Например, с целью сокращения числа пересечений ребер графа, изображенного на плоскости, уменьшения суммарной длины ребер графа.

Граф, у которого существует хотя бы одна пара вершин, соединенная кратными ребрами (дугами в одном направлении), называют *мультиграфом* (рис. 3.10). Максимальное число кратных ребер (*дуг*) в графе называют *мультичислом графа*.

Мультичисло мультиграфа, представленного на рис. 3.10, равно трем.

Если в графе $G(X, U)$ опущены некоторые ребра, а число вершин осталось прежним, то полученный граф $G(X, U')$ называют *частичным графом* $G(X, U)$ или *сугрaфом*. На рис. 3.11 показан частичный граф мультиграфа рис. 3.10.

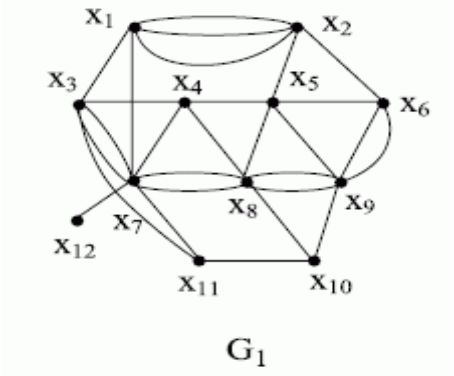


Рисунок. 3.10. Пример мультиграфа G_1

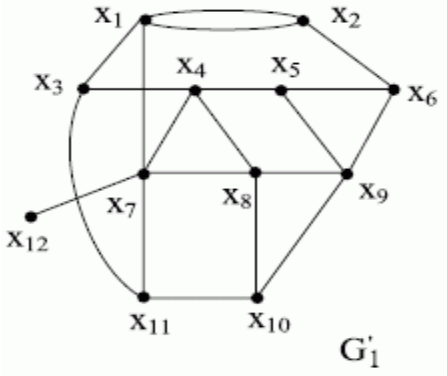


Рисунок 3.11. Частичный граф мультиграфа G_1

3.5.3. Деревья

Связный неориентированный граф, не содержащий циклов, называют *деревом* и обозначается $T(X, W)$. Число ребер дерева $|W| = n - 1$.

Несвязный граф без циклов, отдельные компоненты связности которого являются деревьями, называют *лесом*. Число ребер леса с p компонентами связности $|W| = n - p$.

Все приведенные на рис. 3.12 деревья изоморфны друг другу, т.е. на трех вершинах можно построить только одно неизоморфное дерево.

На n вершинах, пронумерованных числами от 1 до n, можно построить n^{n-2} различных деревьев (*теорема Кэли*). Таким образом, для n=3 можно построить

ровно три дерева, для n=4 – 16 деревьев (рис. 3.13), для n=5 – 125 деревьев и так далее.

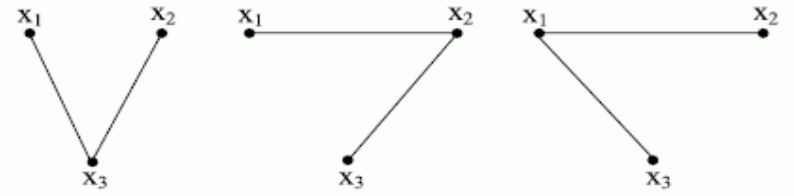


Рисунок 3.12. Три различных дерева, которые можно построить на трех вершинах

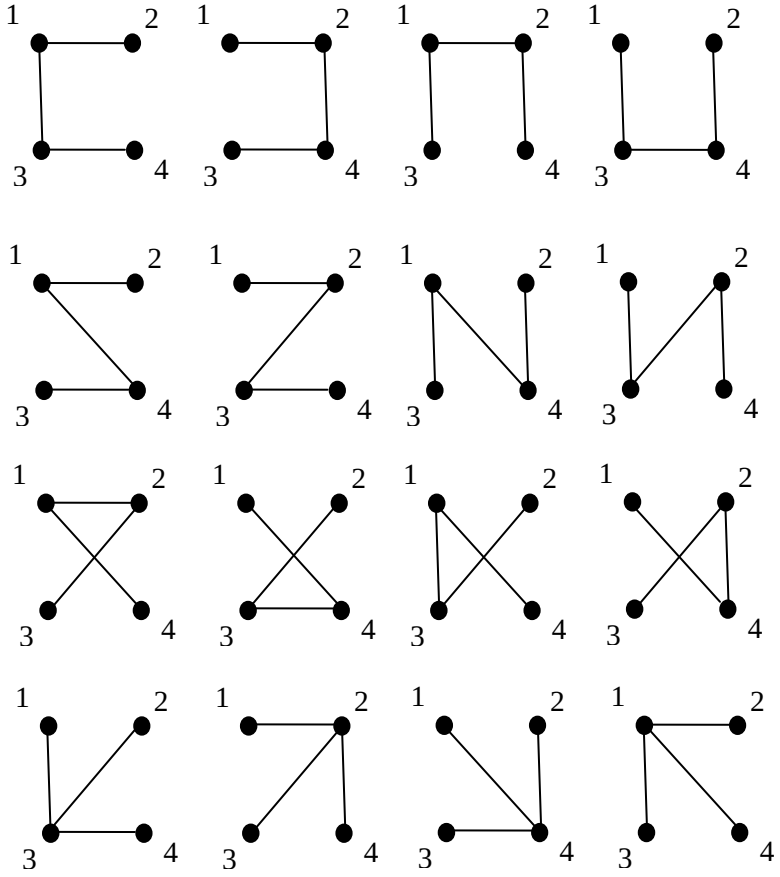


Рисунок 3.13. Деревья, построенные на четырех вершинах

3.5.4. Способы задания графов

Уже отмечалось, что произвольный граф можно задать совокупностью двух множеств: X – множества вершин и U – множества ребер (дуг) графа или множества X и отображения Γ множества X в X.

Другой удобной формой описания графов является представление их с помощью матриц, методика формального получения которых хорошо разработана.

Матрица смежности. Из ранее сказанного известно, что две вершины x_i и $x_j \in X$ графа $G(X, U)$ называются смежными, если они являются граничными вершинами ребра $u_r \in U$

Отношение смежности на множестве вершин графа можно определить, представив каждое ребро как пару смежных вершин, т.е. $u_r=(x_i, x_j)$, $r=1, 2, ..., k$.

Для неориентированных графов такие пары неупорядочены, так что

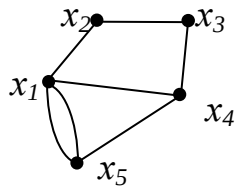
$u_k=(x_i, x_j)=(x_j, x_i)$, а для орграфов - упорядочены, причем, x_i, и x_j означают, соответственно, начальную и конечную вершины дуги u_r. Петля при вершине x_j в обоих случаях представляется неупорядоченной парой (x_j, x_j).

Множество вершин X вместе с определенным на нем отношением смежности полностью определяет граф.

Граф можно представить матрицей смежности $A=\|a_{ij}\|_{n\times n}$, где n число вершин графа.

Строки и столбцы матрицы соответствуют вершинам графа, а ее a_{ij} элемент равен числу кратных ребер, связывающих вершины x_i, и x_j (или направленных от вершины x_i, к вершине x_j для орграфов).

На рис. 3.14 представлены граф и его *матрица смежности*.



$$A=\begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ x_1 & 0 & 1 & 0 & 1 & 2 \\ x_2 & 1 & 0 & 1 & 0 & 0 \\ x_3 & 0 & 1 & 0 & 1 & 0 \\ x_4 & 1 & 0 & 1 & 0 & 1 \\ x_5 & 2 & 0 & 0 & 1 & 0 \end{array}$$

Рисунок 3.14. граф $G(X, U)$ и его матрица смежности

Матрица смежности неориентированного графа всегда симметрична, а *орграфа* - в общем случае несимметрична. Неориентированным ребрам соответствуют пары ненулевых элементов, симметричных относительно главной диагонали матрицы, а петлям - ненулевые элементы главной диагонали.

В столбцах и строках, соответствующих изолированным вершинам, все элементы равны нулю. Элементы матрицы простого графа равны 0 или 1, причем, элементы главной диагонали равны 0.

Правильность составления матрицы легко проверить: для неориентированного графа сумма элементов в каждом i -том столбце или строке соответствует степени вершины $\rho(x_i)$.

При решении целого класса задач проектирования электронных устройств, приходится оперировать матрицами, которые строятся аналогично матрицам смежности, но значения их элементов определяются весом, связанным с ребром (дугой) графа.

Матрица весов. Это квадратная матрица $C=\|c_{ij}\|_{n\times n}$, элемент которой

$$c_{ij}=\begin{cases} t_{ij}, \text{ если } x_i \text{ смежна } x_j, \\ 0, \text{ в противном случае,} \end{cases}$$ где t_{ij} - вес связи (x_j, x_j).

Применение матриц весов позволяет учитывать различные требования к сокращению длины тех или иных электрических соединений в конструкциях ЭВМ, условия тепловой и электромагнитной совместимости отдельных элементов схемы и другие.

Матрица инцидентности. В то время как смежность представляет собой отношение между однородными объектами (вершинами), инцидентность - это отношение между разнородными объектами (вершинами и ребрами).

Рассматривая инцидентность вершин и ребер графа, можно представить его *матрицей инцидентности* $B=\|b_{ij}\|_{n\times k}$, строки которой соответствуют вершинам, а столбцы – ребрам,

где $n=|X|$ – число вершин графа, $k=|U|$ – число ребер графа.

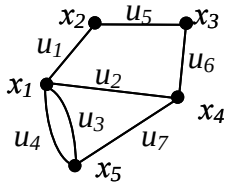
Для неографа:

$$b_{ij}=\begin{cases}1, \text{если вершина } x_i \text{ инцидентна ребру } u_j,\\ 0, \text{в противном случае.}\end{cases}$$

Для орграфа:

$$b_{ij}=\begin{cases}1, \text{если вершина } x_i \text{ начало ребра } u_j,\\ 0, \text{если вершина } x_i \text{ не инцидентна ребру } u_j,\\ -1, \text{если вершина } x_i \text{ конец ребра } u_j.\end{cases}$$

На рис. 3.15 представлены граф $G(X, U)$ и его матрица инцидентности.



$$B=\begin{array}{c|ccccccc} & u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7\\ \hline x_1 & 1 & 1 & 1 & 1 & 0 & 0 & 0\\ x_2 & 1 & 0 & 0 & 0 & 1 & 0 & 0\\ x_3 & 0 & 0 & 0 & 0 & 1 & 1 & 0\\ x_4 & 0 & 1 & 0 & 0 & 0 & 1 & 1\\ x_5 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}$$

Рисунок 3.15. Граф $G(X, U)$ и его матрица инцидентности

Количество единиц в строке равно степени соответствующей вершины (для орграфа количество положительных единиц определяет положительную степень, а количество отрицательных единиц - отрицательную степень).

Граф однозначно задается матрицами смежности и инцидентности. В свою очередь, каждая из этих матриц полностью определяет граф.

Существуют простые приемы перехода от одной матрицы к другой.

Матрица длин. Это квадратная матрица $D=\|d_{ij}\|_{n\times n}$, элемент которой

$$d_{ij}=\begin{cases}l_{ij}, \text{если } x_i \text{ смежна } x_j,\\ 0, \text{в противном случае,}\end{cases}$$ где l_{ij} – длина ребра $(x_j,, x_j)$.

В САПР используют различные метрики.

Если проводники разрешается проводить под любым углом (например, проводной монтаж), то используют евклидову метрику

$l_{ij}=\sqrt{(s_i-s_j)^2+(t_i-t_j)^2}$, где s_i, t_i и s_j, t_j – координаты вершин x_i и x_j, соответственно.

Если проводники разрешается проводить только параллельно осям (ортогонально), то пользуются ортогональной метрикой (так называемое манхэттенское расстояние)

$l_{ij}=|s_i-s_j|+|t_i-t_j|$.

При проектировании печатного монтажа для уменьшения числа межслойных переходов стараются вертикальные и горизонтальные проводники распределять по разным слоям. В этом случае расстояние между двумя точками представляется в виде степенной функции:

$l_{ij} = (s_i - s_j)^k + (t_i - t_j)^r$, где k и r, как правило, равны $1, 2, 3$.

Матрицу длин используют при решении задач оптимизации размещения конструктивных элементов на плате, когда одним из критериев качества является суммарная длина соединений.

3.5.5. Характеристические числа графов

Рассмотрим некоторые характеристические числа графа, не зависящие от изоморфных преобразований. Такие числа называют инвариантами графа, т.е. у изоморфных графов они равны.

Цикломатическое число. *Цикломатическое число* графа указывает то число ребер, которое нужно удалить из данного графа, чтобы получить дерево (для связного графа) или лес (для несвязного графа), т.е. добиться отсутствия у графа циклов.

Пусть связный граф $G(X, U)$ имеет $n=|X|$ вершин и $r=|U|$ ребер. Тогда, дерево, построенное на этом графе, имеет $|W| = n$-1 ребро. По определению цикломатическое число

$\upsilon(G) = r - (n - 1) = r - n + 1$.

Для несвязного графа с p компонентами связности, цикломатическое число

$\upsilon(G) = r - n + p$.

Цикломатическое число всегда неотрицательно.

Основное свойство цикломатического числа формулируется в виде теоремы:

Цикломатическое число графа равно максимальному числу независимых циклов.

Знание цикломатического числа оказывается полезным при анализе топологии электронных схем, а также для решения целого класса задач конструкторского проектирования ЭВМ.

Хроматическое число. Пусть, задан граф $G(X,U)$ без петель. Разобьем множество его вершин на k непересекающихся подмножеств $X_1, X_2, ..., X_k$,

$X = \bigcup_{i=1}^{k} X_i, \forall X_i, X_j \subset X\,[X_i \cap X_j = \varnothing]$ так, чтобы любые две смежные вершины x_i и $x_j \in X$ принадлежали разным подмножествам, т.е. чтобы ребра графа $G(X,U)$ соединяли вершины из разных подмножеств ($\forall X_s \subset X\ [\Gamma X_s \cap X_s = \varnothing]$, где ΓX_s множество вершин, смежных вершинам множества X_s).

Задача раскраски вершин графа формулируется следующим образом. Необходимо раскрасить вершины графа таким образом, чтобы смежные вершины были окрашены в разные цвета. Минимальное число красок, в которые можно

раскрасить граф называется *хроматическим числом графа* и обозначается $\chi(G)$, а граф $G(X, U)$ называют χ-*хроматическим*.

Особое значение имеет частный вид χ-хроматического графа – *бихроматический граф*, для которого множество вершин X можно разбить на два непересекающихся подмножества X_1 и X_2 так, чтобы ребра соединяли вершины разных подмножеств ($X=X_1 \cup X_2$, $X_1 \cap X_2=\varnothing$, $\forall x_i \in X_1$ $[\Gamma x_i \cap X_1=\varnothing]$, $\forall x_j \in X_2$ $[\Gamma x_j \cap X_2=\varnothing]$.

Такие графы называют бихроматическими, двудольными графами или графами Кенига и обозначают $G(X_1, X_2, U)$.

Критерий бихроматичности произвольного графа формулируется теоремой Кенига, согласно которой граф $G(X, U)$ является бихроматическим тогда и только тогда, когда он не содержит циклов нечетной длины.

Если граф $G(X, U)$ - дерево, т.е. в нем отсутствуют циклы, то он является бихроматическим графом и может быть представлен в виде двудольного графа (рис. 3.16).

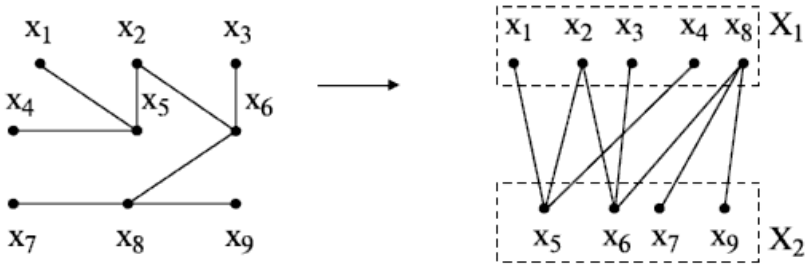


Рисунок 3.16. Пример представления дерева в виде двудольного графа

Граф Кенига $G(X_1, X_2, U)$ называют *полным*, если каждая вершина $x_i \in X_1$ смежна с каждой вершиной $x_j \in X_2$ и наоборот. Полный двудольный граф обозначают K_{nm}, где $n=|X_1|$ и $m=|X_2|$. Полный *бихроматический граф* K_{33} приведен на рис. 3.17.

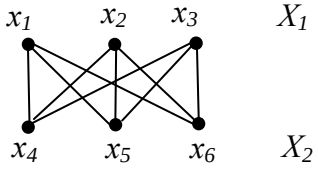


Рис. 3.17. Полный бихроматический граф K_{33}

В отличие от цикломатического числа определение хроматического числа осуществляется с помощью сравнительно сложных алгоритмов, в основу большинства которых положены методы целочисленного линейного программирования.

Число планарности. Граф $G(X, U)$ называют *плоским* тогда и только тогда, когда он ***геометрически реализован*** на плоскости так, что все его ребра пересекаются только в вершинах X графа. Граф $G(X,U)$ называют *планарным*, если он ***может быть геометрически реализован*** на плоскости без пересечения ребер. Т.е., плоскостность это свойство его геометрической реализации на плоскости, а планарность – свойство графа быть реализованным на плоскости. На рис. 3.18 показаны полный планарный граф K_4 и его плоская реализация.

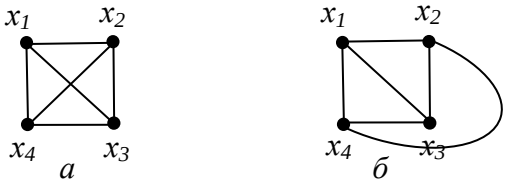


Рисунок 3.18. Полный планарный граф K_4 (*а*) и его плоская реализация (*б*)

На рис. 3.19. приведены непланарные графы Понтрягина-Куратовского (полный граф K_5 (*а*) и полный двудольный граф K_{33} (*б*)).

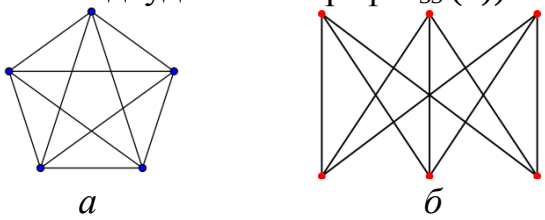


Рисунок 3.19. Непланарные графы K_5 (*а*) и K_{33} (*б*)

Граф K_5 – непланарный граф с минимальным числом вершин, а граф K_{33} – непланарный граф с минимальным числом ребер.

Минимальное число ребер, которое нужно удалить из графа, чтобы он стал планарным, называется *числом планарности* и обозначается $\Theta(G)$. Для полного графа K_n с $n \geq 4$ $\Theta(G) = (n - 3)(n - 4)/2$.

Число планарности графа K_4 равно 0, графа K_5 $\Theta(G) = (5 - 3)(5 - 4)/2=1$ (рис. 3.20).

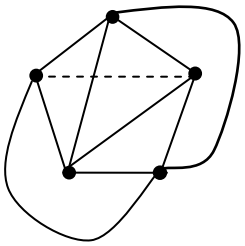


Рисунок 3.20. Ребро, удаление которого делает граф K_5 планарным

Минимальное число плоских суграфов, на которые разбивается граф $G(X,U)$ называется *толщиной графа* t(G).

Число внутренней устойчивости. Множество вершин X_s графа $G(X,U)$ называется *внутренне устойчивым* (*независимым*), если никакие две вершины из этого множества не смежны, $X_s \subset X$ $[\Gamma X_s \cap X_s = \varnothing]$. Внутренне устойчивое множество называется *максимальным*, если оно не является собственным подмножеством некоторого другого независимого множества. Максимальное по мощности независимое множество называется *наибольшим*. Число вершин в наибольшем независимом множестве графа $G(X,U)$ называется *числом внутренней устойчивости* этого графа и обозначается $\alpha(G)$, $\alpha(G)=\max|X_s|$.

Для графа $G(X,U)$, изображенного на рис. 3.21, множества вершин $\{x_3, x_6\}$, $\{x_4, x_6\}$, $\{x_1, x_3, x_7\}$, $\{x_1, x_4, x_5\}$ являются максимальными, но не наибольшими. Множество $\{x_2, x_5, x_7, x_8\}$ является наибольшим, $\alpha(G)=4$.

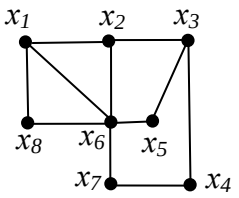


Рисунок 3.21. Граф $G(X,U)$

Заметим, что при раскраске графа, множество вершин, окрашенных в один цвет – внутренне устойчивое.

Число внешней устойчивости. Множество вершин X_s графа G (X,U) называется *внешне устойчивым* (*доминирующим*), если каждая вершина из $X\backslash X_s$ смежна с некоторой вершиной из X_s. Иначе говоря, каждая вершина графа $x_j \in X_s$ или $x_j \in \Gamma X_s$, т.е. находится на расстоянии не более 1 от внешне устойчивого множества, $\Gamma X_s \cup X_s = X$.

Внешне устойчивое множество X_s называется *минимальным*, если при удалении из него любой вершины получается множество, не являющееся внешне устойчивым.

Внешне устойчивое множество, состоящее из минимального числа вершин, называется *наименьшим*.

Число вершин в наименьшем независимом множестве графа G (X,U) называется *числом внешней устойчивости* этого графа и обозначается $\beta(G)$, $\beta(G) = \min|X_s|$.

Для графа G (X,U), изображенного на рис. 3.21, множества вершин $\{x_1, x_3, x_7\}$, $\{x_1, x_4, x_5, x_7\}$ являются минимальными, но не наименьшими. Множества $\{x_3, x_6\}$ и $\{x_4, x_6\}$ является наименьшими, $\beta(G)=2$.

Подмножество вершин графа, являющееся как внутренне устойчивым, так и внешне устойчивым, называется *ядром*.

Плотность графа. Максимальный полный подграф графа G (X,U) называется *кликой* графа G; другими словами, клика графа G есть подмножество его вершин, такое, что между каждой парой вершин этого подмножества существует ребро и, кроме того, это подмножество не принадлежит никакому большому подмножеству с тем же свойством.

Число вершин клики графа называется *плотностью графа* и обозначается $f(G)$.

Для графа $G(X,U)$, изображенного на рис. 3.22 клику образуют вершины (2, 3, 6, 7). Плотность этого графа $f(G)=4$.

1

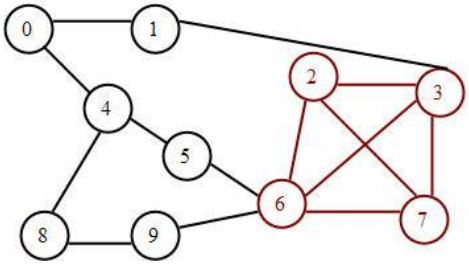


Рисунок 3.22. Граф $G(X,U)$ с кликой (2, 3, 6, 7)

Число Хадвигера. Операцией сжатия графа (стягивание графа) называется замена двух смежных вершин x_i и x_j одной x_s с удалением ребра u_{ij}. *Числом Хадвигера* $\eta(G)$ графа $G(X,U)$ называется такое наибольшее число η, что граф G стягиваем к полному графу K_η. На рис. 3.23,*а* изображен граф Петерсена. В результате стягивания графа (замена вершин x_1 и x_6 на y_1; x_2 и x_7 на y_2 и т.д.) получим полный граф K_5 (рис.3.23,*б*). Число Хадвигера графа Петерсена $\eta(G)$=5.

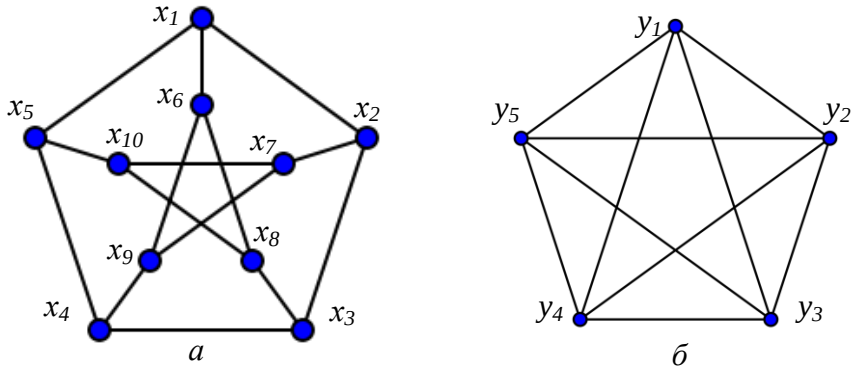


Рисунок 3.23. Граф Петерсена (*а*) и полный граф K_5 (*б*)

3.6. Математические модели электрических схем

Любая функциональная или принципиальная схема состоит из набора элементов, заданным образом соединенных между собой. Поэтому схему можно рассматривать как некоторое множество элементов $E=\{e_1, e_2, ..., e_m\}$, соединенных между собой цепями из множества $U=\{u_1, u_2, ..., u_n\}$. Пусть дана электрическая схема (рис. 3.24).

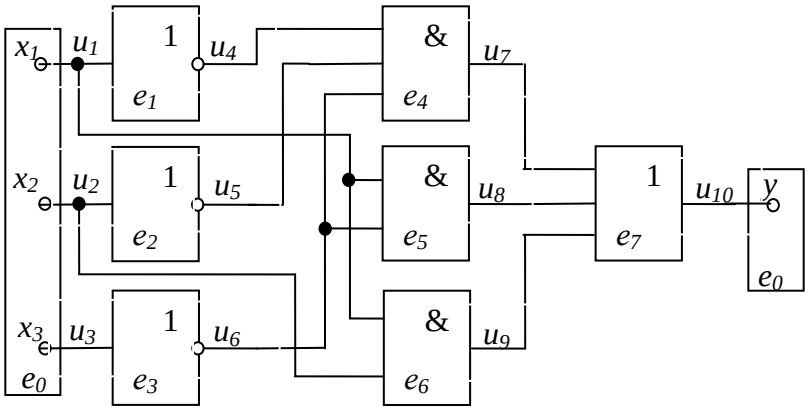


Рисунок 3.24. Электрическая схема

Схему можно представить в виде различных графов.

1. Модель схемы в виде двудольного графа *G*(*E, U, P*)

Модель представляет два множества E и U, вершины которых соединены между собой. Вершина e_i соединяется с цепью u_j, если она принадлежит этой цепи (рис. 3.25).

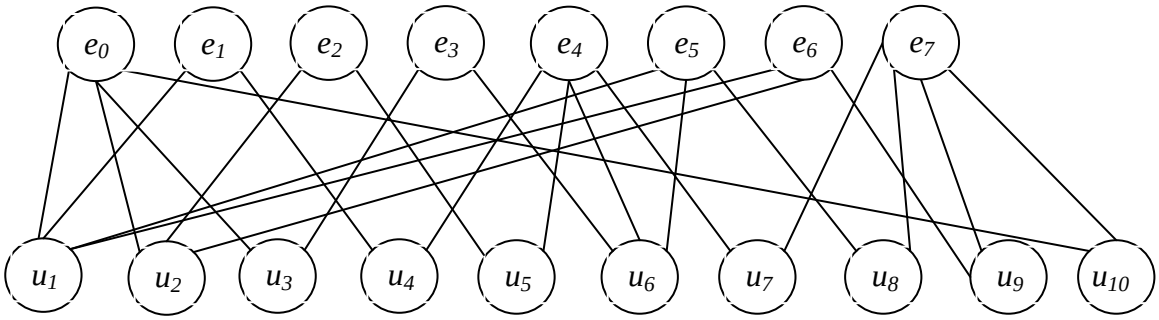


Рисунок 3.25. Двудольный граф, описывающий схему

Достоинство такой модели в том, по ней с точностью до входов/выходов можно восстановить схему, т.к. сохраняется состав цепей.

Недостатком модели является низкая наглядность. Так, по модели сложно определить связность элементов.

2. Модель схемы в виде гиперграфа *H*(*E, U*)

Гиперграфом H(*E, U*) называется граф, ребра которого могут соединять больше двух вершин. Ребра в гиперграфе называются гиперребрами. В модели схемы гиперребро это цепь.

На рис. 3.26 приведена модель схемы в виде гиперграфа.

Достоинства и недостатки у гиперграфа такие же, как и двудольного графа.

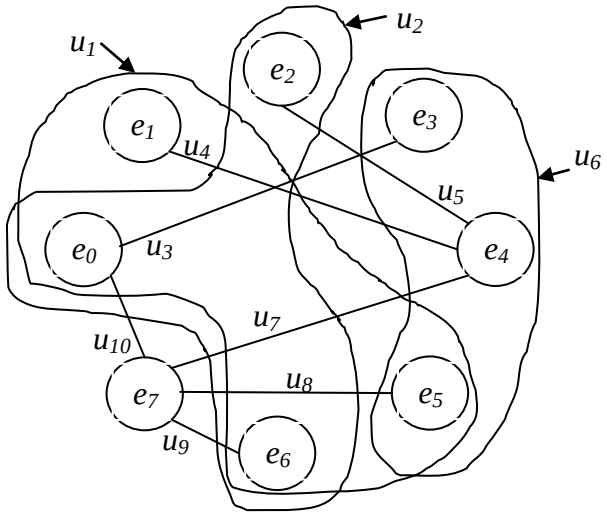


Рисунок 3.26. Гиперграф, описывающий схему

В ЭВМ двудольный граф и гиперграф представляются *матрицей комплексов* $Q=\|q_{ij}\|_{n\times k}$, строки которой соответствуют элементам, а столбцы – цепям, где $n=|E|$ – число элементов схемы, $k=|U|$ – число цепей схемы. Матрица комплексов аналогична матрице инцидентности, отличие состоит в том, что в столбце может быть больше двух единиц. Для рассматриваемой схемы матрица комплексов:

$$Q = \begin{array}{c|cccccccccc} & u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 & u_9 & u_{10} \\ \hline e_0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ e_1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_2 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ e_3 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ e_4 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ e_5 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ e_6 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ e_7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

3. Модель схемы в виде мультиграфа *G*(*E, U*)

Мультиграф легко получить из двудольного графа. Необходимо, последовательно просматривая цепи, соединять между собой элементы, входящие в цепь (рис. 3.27).

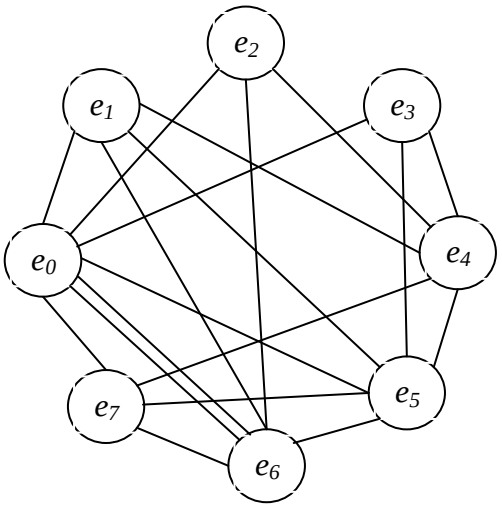


Рисунок 3.27. Модель схемы в виде мультиграфа

Достоинство модели – наглядность. На графе видно, как элементы связаны между собой.

Недостатки модели следующие:

1. Цепи распались на отдельные проводники и невозможно определить, к какой цепи принадлежит конкретный проводник. Схему по модели восстановить нельзя.

2. Главный недостаток модели – неточность, избыточность. Каждая цепь представлена полным подграфом. А на практике, например, при размещении элементов будем пытаться уменьшить длину всех проводников, хотя цепь будет реализована в виде дерева.

4. **Модель схемы в виде взвешенного графа *G*(*E, U*).** Модель строится, как и мультиграф, только проводятся не кратные ребра, а рядом с ребрами ставится вес – кратность ребра. Свойства модели такие же, как и у мультиграфа.

В ЭВМ взвешенный граф и мультиграф представляются *матрицей смежности,* которая в САПР называется *матрицей соединений* $R=\|r_{ij}\|_{n\times n}$. Матрицу соединений легко получить из матрицы комплексов $r_{ij}=\sum_{s=1}^{k} q_{is}q_{js}$.

$R=$

	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7
e_0	0	1	1	1	0	1	2	1
e_1	1	0	0	0	1	1	1	0
e_2	1	0	0	0	1	0	1	0
e_3	1	0	0	0	1	1	0	0
e_4	0	1	1	1	0	1	0	1
e_5	1	1	0	1	1	0	1	1
e_6	2	1	1	0	0	1	0	1
e_7	1	0	0	0	1	1	1	0

Разработан еще целый ряд моделей схемы в виде различных графов, которые с разной степенью детализации отражают свойства объекта.

Для дальнейшего рассмотрения алгоритмов конструкторского проектирования ограничимся описанными моделями.

4. Алгоритмы автоматизированного проектирования ЭВМ

4.1. Основные свойства алгоритмов

Понятие алгоритма принадлежит к числу понятий столь фундаментальных, что не может быть выражено через другие, а должно рассматриваться как неопределяемое. Алгоритм это точное предписание, которое задает вычислительный процесс, начинающийся с произвольных исходных данных и направленный на получение полностью определяемого этими исходными данными результата.

Основные свойства алгоритмов:

1. *Дискретность*. Процесс решения протекает в виде последовательности отдельных действий, следующих друг за другом.

2. *Детерминированность* (определенность). В каждый момент времени, следующий шаг работы однозначно определяется состоянием системы. Таким образом, алгоритм выдаёт один и тот же результат (ответ) для одних и тех же исходных данных.

3. *Определенность*. Каждое действие определено и после выполнения каждого действия однозначно определяется, какое действие будет выполнено следующим.

4. *Конечность*. Алгоритм заканчивает работу после конечного числа шагов.

5. *Результативность*. В момент прекращения работы алгоритма известно, что является результатом.

6. *Массовость*. Алгоритм описывает некоторое множество процессов, применимых при различных входных данных.

7. Алгоритм считается правильным, если при любых допустимых данных он заканчивает работу и выдает результат, удовлетворяющий требованиям задачи.

8. Алгоритм однозначен, если при применении к одним и тем же входным данным он дает один и тот же результат.

4.2. Элементы теории сложности

Первые фундаментальные работы по теории алгоритмов были опубликованы независимо в 1936 году Аланом Тьюрингом, Алоизом Черчем и Эмилем Постом.

Английский математик А.М.Тьюринг описал машину, названную его именем. Он показал, что если проблемы не могут быть решены на его машине, то они не могут быть решены ни на какой другой ЭВМ, т.е. это проблемы для которых алгоритмы не могут быть составлены даже в принципе.

Один из основных результатов Тьюринга – разделение всех представляемых в математике проблем на два класса:

1. проблемы, для которых алгоритмы никогда не могут быть написаны, т.е., в формальном смысле, постоянно не решаемые;
2. проблемы, которые могут быть решены с помощью алгоритмов.

Класс решаемых проблем может быть разделен на два подкласса:

1. подкласс, содержащий только полиномиальные алгоритмы;
2. подкласс, содержащий только экспоненциальные алгоритмы.

Эффективностью алгоритма называют максимальное число элементарных операций, выполняемых при его работе. Ее записывают в виде $O(f(n))$. По определению $O(f(n))$ – это множество всех функций $g(n)$, для которых существует положительная постоянная c и такой номер n_0, что $|g(n)| \le c|f(n)|$ для всех $n > n_0$.

Распространенным критерием оценки алгоритмов является время работы и порядок роста продолжительности работы в зависимости от объема входных данных.

Для каждой конкретной задачи составляют некоторое число, которое называют ее размером. Например, размером задачи вычисления произведения матриц может быть наибольший размер матриц-множителей, для задач на графах размером может быть количество вершин или ребер графа.

В рамках классической теории алгоритмические задачи различаются по классам сложности (P-сложные, NP-сложные, экспоненциально сложные и др.).

Классы сложности - множества вычислительных задач, примерно одинаковых по сложности вычисления. Более узко, классы сложности — это множества предикатов (функций, получающих на вход слово и возвращающих ответ 0 или 1), использующих для вычисления примерно одинаковые количества ресурсов. Каждый класс сложности (в узком смысле) определяется как множество предикатов, обладающих некоторыми свойствами.

Классом сложности X называется множество предикатов $P(x)$, вычислимых на машинах Тьюринга и использующих для вычисления $O(f(n))$ ресурса, где n — длина слова x.

В качестве ресурсов обычно берутся время вычисления (количество рабочих тактов машины Тьюринга) или рабочая зона (количество использованных ячеек на ленте во время работы).

Класс P – задачи, которые могут быть решены за время, полиномиально зависящее от объёма исходных данных, с помощью детерминированной вычислительной машины (например, машины Тьюринга),

Класс NP — задачи, которые могут быть решены за полиномиально выраженное время с помощью недетерминированной вычислительной машины, то есть машины, следующее состояние которой не всегда однозначно определяется предыдущими. К классу NP относятся задачи, решение которых с помощью дополнительной информации полиномиальной длины, мы можем проверить за полиномиальное время. В частности, к классу NP относятся все задачи, решение которых можно проверить за полиномиальное время. Класс P содержится в классе NP.

Работу такой машины можно представить как разветвляющийся на каждой неоднозначности процесс: задача считается решенной, если хотя бы одна ветвь процесса пришла к ответу.

Поскольку класс P содержится в классе NP, принадлежность той или иной задачи к классу NP зачастую отражает наше текущее представление о способах решения данной задачи и носит неокончательный характер. В общем случае нет оснований полагать, что для той или иной NP-задачи не может быть найдено P-решение. Вопрос о возможной эквивалентности классов P и NP (то есть о возможности нахождения P-решения для любой NP-задачи) считается многими одним из основных вопросов современной теории сложности алгоритмов. Ответа на этот вопрос нет. Сама постановка вопроса об эквивалентности классов P и NP возможна благодаря введению понятия NP-полных задач. NP-полные задачи составляют подмножество NP-задач и отличаются тем свойством, что все NP-задачи могут быть тем или иным способом сведены к ним. Из этого следует, что если для NP-полной задачи будет найдено P-решение, то P-решение будет найдено для всех задач класса NP. Примером NP-полной задачи является задача о конъюнктивной форме.

Рассмотрим наиболее часто встречающиеся классы сложности в зависимости от числа входных данных n (в порядке нарастания сложности, т.е. увеличения времени работы алгоритма, при стремлении n к бесконечности):

$O(1)$ - количество шагов алгоритма не зависит от количества входных данных. Обычно это алгоритмы, использующие определённую часть данных входного потока и игнорирующие все остальные данные. Например, чистка 1 квадратного метра ковра вне зависимости от его размеров.

Ряд алгоритмов имеют порядок, включающий $\log_2 n$, и называются логарифмическими (logarithmic). Эта сложность возникает, когда алгоритм неоднократно подразделяет данные на подсписки, длиной 1/2, 1/4, 1/8, и так далее от оригинального размера списка. Логарифмические порядки возникают при работе с бинарными деревьями. Бинарный поиск имеет сложность среднего и наихудшего случаев $O(\log_2 n)$.

Сложность $O(n\log_2 n)$ имеют алгоритмы быстрой сортировки, сортировки слиянием и "кучной" сортировки, алгоритм Краскала - построение минимального связывающего дерева, n - число ребер графа.

Алгоритм со сложностью $O(n)$ - алгоритм линейной сложности. Например, просмотр обложки каждой поступающей книги - то есть для каждого входного объекта выполняется только одно действие;

Алгоритмы, имеющие порядок $O(n^2)$, являются квадратичными (quadratic). К ним относятся наиболее простые алгоритмы сортировки; алгоритм Дейкстры - нахождение кратчайших путей в графе, алгоритм Прима - построение минимального связывающего дерева, n - число вершин графа. Квадратичные алгоритмы используются на практике только для относительно небольших значений n. Всякий раз, когда n удваивается, время выполнения такого алгоритма увеличивается на множитель четыре.

Алгоритм показывает кубическое (cubic) время, если его порядок равен $O(n^3)$, и такие алгоритмы очень медленные. Всякий раз, когда n удваивается, время выполнения алгоритма увеличивается в восемь раз. Алгоритм Флойда–Уоршелла (динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа).

Алгоритм со сложностью $O(2^n)$ имеет экспоненциальную сложность (exponential complexity). Такие алгоритмы выполняются настолько медленно, что они используются только при малых значениях n. Этот тип сложности часто ассоциируется с проблемами, требующими неоднократного поиска дерева решений.

Алгоритмы со сложностью $O(n!)$ - факториальные алгоритмы, в основном, используются в комбинаторике для определения числа сочетаний, перестановок.

В таблице 4.1 приводятся линейный, квадратичный, кубический, экспоненциальный и логарифмический порядки величины для выбранных значений n. Из таблицы очевидно, что следует избегать использования кубических и экспоненциальных алгоритмов, если только значение n не мало.

Таблица 4.1. Сравнение сложности различных алгоритмов

n	$\log_2 n$	$n\log_2 n$	n^2	n^3	2^n
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536
32	5	160	1024	32768	4294967296
128	7	896	16384	2097152	3.4×10^{38}
1024	10	10240	1048576	1073741824	1.8×10^{308}
65536	16	1048576	4294967296	2.8×10^{14}	Избегайте!

Важность проведения резкой границы между полиномиальными и экспоненциальными алгоритмами вытекает из сопоставления числовых примеров роста допустимого размера задачи с увеличением быстродействия B используемых ЭВМ.

В табл. 4.2 указаны размеры задач, решаемых за одно и то же время T на ЭВМ с быстродействием B_1 при различных зависимостях сложности $O(n)$.

Таблица 4.2. Размеры решаемых задач алгоритмов разных класов сложности

$O(n)$	B_1	$B_2 = 100\ B_1$	$B_3 = 1000\ B_1$
n	n_1	$100n_1$	$1000n_1$
n^2	n_2	$10n_2$	$31{,}6n_2$
n^3	n_3	$4{,}64n_3$	$10n_3$
2^n	n_4	$6{,}64 + n_4$	$9{,}97 + n_4$

Эти примеры показывают, что, выбирая ЭВМ в K раз более быстродействующую, получаем увеличение размера решаемых задач при линейных алгоритмах в K раз, при квадратичных - в $K^{1/2}$ раз и т. д.

Иначе обстоит дело с неэффективными алгоритмами. Так, в случае сложности 2^n для одного и того же процессорного времени размер задачи увеличивается только на $\lg K / \lg 2$ единиц. Следовательно, переходя от ЭВМ с $B_1 = 1$ Гфлопс к суперЭВМ с $B_3 = 1$ Тфлопс, можно увеличить размер решаемой задачи только на 10, что совершенно недостаточно для практических задач. Действительно, в таких задачах, как, например, синтез тестов для БИС число входных двоичных переменных может составлять более 100 и поэтому полный перебор всех возможных проверяющих кодов потребует выполнения более 2^{100} вариантов моделирования схемы.

5. Алгоритмы компоновки

Процесс преобразования функционального описания в конструктивное называется *компоновкой*. Иначе, компоновка это распределение элементов низшего уровня иерархии по узлам высшего уровня.

Различают две постановки задачи компоновки:

- *покрытие* – преобразование исходной схемы в схему соединения элемнтов, номенклатура которых задана;
- *разрезание* – разбиение исходной схемы на части.

Известные алгоритмы компоновки можно условно разбить на 5 групп:

1. алгоритмы, использующие методы целочисленного программирования;
2. алгоритмы, основанные на методе ветвей и границ;
3. последовательные алгоритмы;
4. итерационные алгоритмы;
5. смешанные алгоритмы.

Алгоритмы первой и второй групп, хотя и позволяют получить точное решение задачи, однако для устройств реальной сложности фактически не реализуемы на ЭВМ.

В алгоритмах разбиения, опирающихся на идеи *математического программирования*, в основном используются *методы ветвей и границ* и решение задачи о назначении.

Алгоритмы разбиения, использующие методы ветвей и границ, состоят из следующих этапов.

Сначала определяется нижняя оценка разбиения графа на заданное число частей. Затем производится построение *дерева решений* и осуществляется поиск оптимального результата.

Задачу разбиения графа схемы на части можно свести к задаче о назначении следующим образом.

Сначала отыскивают назначение кандидатов (вершин графа) на все части, дающие минимальные суммарные затраты, причем, каждая вершина графа может быть назначена только в одну часть и в каждой части должны содержаться различные вершины графа.

Наибольшее распространение получили приближенные алгоритмы компоновки (последовательные, итерационные, смешанные).

При использовании последовательных алгоритмов сначала по определенному правилу выбирают вершину графа, затем осуществляют последовательный выбор вершин (из числа нераспределенных) и присоединение их к формируемому куску графа. После образования первого куска переходят ко второму и т. д. до получения желаемого разрезания исходного графа.

В итерационных алгоритмах начальное разрезание графа на куски выполняют произвольным образом; оптимизация компоновки достигается парными или групповыми перестановками вершин графа из различных кусков. Процесс перераспределения вершин заканчивают при получении локального экстремума целевой функции, удовлетворяющего требованиям разработчика.

В смешанных алгоритмах компоновки для получения начального варианта “разрезания” используется алгоритм последовательного формирования кусков; дальнейшая оптимизация решения осуществляется перераспределением вершин между отдельными кусками графа.

5.1. Алгоритм покрытия

Важной задачей в общей проблеме конструирования является покрытие, т.е. преобразование функциональных схем в принципиальные.

Под покрытием схемы понимается представление функциональной схемы конструктивными элементами, на которых она будет реализована, и связями между ними.

Конечной целью покрытия является выбор оптимальной элементно-технической базы проектируемого устройства.

В курсовой работе по дисциплине «Дискретная математика» задача покрытия решалась при преобразовании схемы из булева базиса в универсальные базисы с ограничением на число входов.

В основу построения большинства алгоритмов покрытия положена идея выделения из функциональной схемы подсхем, перебора их и проверки на совпадение логических функций элементов подсхем и функции модулей исходного набора. Подсхема закрепляется за модулем, в состав которого входит наибольшее количество ее логических функций.

Предполагается, что элементы заданного набора могут реализовать все функции схемы.

Исходная схема представляет множество $M=\{m_1, m_2, ..., m_k\}$ связанных между собой функциональных элементов, таких как логические элементы И, ИЛИ, НЕ, триггер, усилитель, генератор, формирователь и т. д., каждый из которых реализует некоторую функцию ψ_i, а их множество определяет совокупность всех функций $\Psi=\{\psi_1, \psi_2, ..., \psi_k\}$, выполняемых в схеме. Схема представляется графом $G(M, U)$, множество вершин которого отображает элементы, а множество ребер — связи между ними. Задан ограниченный набор (библиотека) типов модулей $N=\{n_1, n_2, ..., n_p\}$, где p — число типов конструктивных модулей в используемой библиотеке, каждый из которых реализует одну или несколько функций (имеет определенный, заранее известный состав функциональных элементов). Каждый конструктивный модуль n_j можно представить в виде подграфа H_j, в котором вершины отображают функциональные элементы, входящие в модуль.

Требуется разбить исходную схему на подсхемы (подграфы) при выполнении следующих условий:

- каждая подсхема по своему функциональному составу должна соответствовать некоторому модулю n_j из заданного набора N;
- любые две подсхемы разбиения не пересекаются, т. е. не имеют общих элементов.

Для оценки качества покрытия используются следующие показатели:

- число типов модулей, использующихся для покрытия;
- полнота использования функциональных возможностей каждого модуля;
- число всех модулей.

Эвристический алгоритм покрытия включает операции вложения подграфов H_j в граф схемы G. Поочередно для каждого H_j - ищутся в графе G изоморфные подграфы G_j. Найденные подграфы G_j удаляются из графа G. Если после этого часть графа осталась непокрытой, то для каждого H_j - вновь ищутся подграфы G_j в G, такие, что $G_j \subset H_j$. Процесс продолжается, до тех пор, пока вся схема не окажется покрытой модулями заданного набора.

Качество покрытия зависит от порядка, в котором просматриваются подграфы H_j. Поэтому предусматривается получение нескольких вариантов покрытия по рассмотренному последовательному алгоритму с выбором наилучшего, в соответствии с одним из критериев компоновки.

Алгоритмы покрытия упрощаются при использовании набора модулей, состоящих из несвязанных между собой функциональных элементов.

Начальное покрытие может быть улучшено парными перестановками однотипных функциональных элементов различных модулей. Замена элементов производится, если при перестановке увеличивается число внутренних связей модулей и уменьшается число внешних связей.

5.2. Последовательный алгоритм компоновки

Задача компоновки формулируется следующим образом. Дан мультиграф $G(X,U)$. Требуется "разрезать" его на отдельные куски $G_1(X_1, U_1)$, $G_2(X_2, U_2)$,…, $G_k(X_k, U_k)$ с числом вершин в каждой, соответственно, $n_1, n_2, ..., n_k$ $(n_1 + n_2 + ...+ n_k = n)$, так, чтобы число ребер, соединяющих эти куски, было минимальным, т.е. минимизировать $\sum_{i=1}^{k}\sum_{j=1}^{k} U_{ij}$, $i \neq j$ при

$$\forall G_i(X_i,U_i), G_j(X_j,U_j) \subset G(X,U)[G_i(X_i,U_i) \neq G_j(X_j,U_j) \Rightarrow (X_i \cap X_j = \varnothing \;\&\; U_i \cap U_j = U_{ij})]$$

$\bigcup_{i=1}^{k} G_i(X_i,U_i) = G(X,U)$ $i, j = 1, 2, \ldots, k,$ где U_{ij} – множество ребер, соединяющих куски $G_i(X_i, U_i)$ и $G_j(X_j, U_j)$.

Разрезание графа последовательным алгоритмом сводится к следующему.

В графе $G(X,U)$ находят вершину $x_i \in X$ с минимальной локальной степенью $\rho(x_i)$.

Если таких вершин несколько, то предпочтение отдается той вершине, которая имеет большее число кратных ребер. С этой вершины начинается построение первого куска. В первый кусок включаются вершина x_i и все вершины, смежные ей: $X_1 = \{x_i\} \cup \Gamma x_i$. Возможны три случая:

1. Число вершин в первом куске $|X_1| = n_1$. В этом случае считаем, что подграф G_1 образован.

2. Число вершин в первом куске $|X_1| > n_1$. В этом случае ищется вершина $x_j \in \Gamma x_i$ с максимальной характеристикой $\delta(x_j) = \rho(x_j) - 2\sum_{x_s \in X_1} r_{js}$.

Вершина x_j удаляется из X_1 ($X_1 = X_1 \setminus \{x_j\}$).

Характеристика $\delta(x_j)$ показывает, насколько уменьшится число внешних связей куска X_1, если из него удалить вершину x_j (характеристика $\delta(x_j)$ знаковая).

Повторив это конечное число раз, придем к первому случаю.

3. Число вершин в первом куске $|X_1| < n_1$. В этом случае из $X \setminus X_1$ выбирается вершина $x_j \notin X_1$, имеющая минимальное число связей с еще не распределенными вершинами. В первый кусок включаются вершины $X_1 = X_1 \cup \{x_j\} \cup \Gamma x_j$. Здесь опять возможны три рассмотренных варианта.

Образованный подграф G_1 исключается из исходного графа.

Получаем граф $G^*(X^*, U^*)$, где $X^* = X \setminus X_1$, $U^* = U \setminus U_1$.

Далее формируется второй подграф G_2 и т.д.

Процесс повторяется k-1 раз.

Рассмотренный алгоритм прост, легко реализуется на ЭВМ. Также среди достоинств данной группы алгоритмов выступает высокое быстродействие их при решении задач компоновки.

Основным недостатком последовательных алгоритмов является неспособность находить глобальный минимум числа внешних связей. Наибольшая эффективность метода последовательного разбиения графа имеет место, когда чис-

ло вершин графа G значительно больше числа вершин в любой части разбиения $n >> n_1, n_2, ..., n_k$.

В качестве примера рассмотрим компоновку элементов схемы, представленной на рис. 3.24.

Необходимо разрезать граф на три куска, причем, $n_1 = n_2 = 3, n_3 = 2$.

Повторим матрицу соединений схемы:

$R =$

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\rho(x_i)$
x_0	0	1	1	1	0	1	2	1	7
x_1	1	0	0	0	1	1	1	0	4
x_2	1	0	0	0	1	0	1	0	3
x_3	1	0	0	0	1	1	0	0	3
x_4	0	1	1	1	0	1	0	1	5
x_5	1	1	0	1	1	0	1	1	6
x_6	2	1	1	0	0	1	0	1	6
x_7	1	0	0	0	1	1	1	0	4

1. В графе $G(X,U)$ находим вершину $x_i \in X$ с минимальной локальной степенью $\rho(x_i)$, это вершины x_2 и x_3. У этих вершин кратных ребер нет, поэтому в качестве начальной, выбираем любую, например, x_2.
2. В первый кусок включаем $X_1 = \{x_2\} \cup \Gamma x_2 = \{x_0, x_2, x_4, x_6\}$. $|X_1| = 4 > n_1$.
3. Для вершин $\Gamma x_2 = \{x_0, x_4, x_6\}$ определим характеристики $\delta(x_j) = \rho(x_j) - 2\sum_{x_s \in X_1} r_{js}$.

$\delta(x_0) = 7 - 6 = 1$, $\delta(x_4) = 5 - 2 = 3$, $\delta(x_6) = 6 - 6 = 0$. Максимальное уменьшение внешних связей куска X_1 будет, если из куска удалить вершину x_4.

$X_1 = X_1 \setminus \{x_4\}$.

4. $|X_1| = 3 = n_1$. Первый кусок сформирован.
5. Удаляем из графа вершины X_1, $X^* = X \setminus X_1$.
6. Для оставшихся вершин матрица соединений:

$R^* =$

	x_1	x_3	x_4	x_5	x_7	$\rho(x_i)$
x_1	0	0	1	1	0	2
x_3	0	0	1	1	0	2
x_4	1	1	0	1	1	4
x_5	1	1	1	0	1	4
x_7	0	0	1	1	0	2

7. Находим вершину $x_i \in X^*$ с минимальной локальной степенью $\rho(x_i)$, это вершины x_1, x_3 и x_7. Для второго куска выберем вершину x_1. $X_2 = \{x_1\} \cup \Gamma x_1 = \{x_1, x_4, x_5\}$.
8. $|X_2| = 3 = n_2$. . Второй кусок сформирован.
9. Удаляем из графа вершины X_2 из $X^* \setminus X_2$. Нераспределенные вершины составят третий кусок $X_3 = \{x_3, x_7\}$.
10. $|X_3| = 2 = n_3$. Граф разрезан.

Результат работы алгоритма представлен на рис. 5.1.

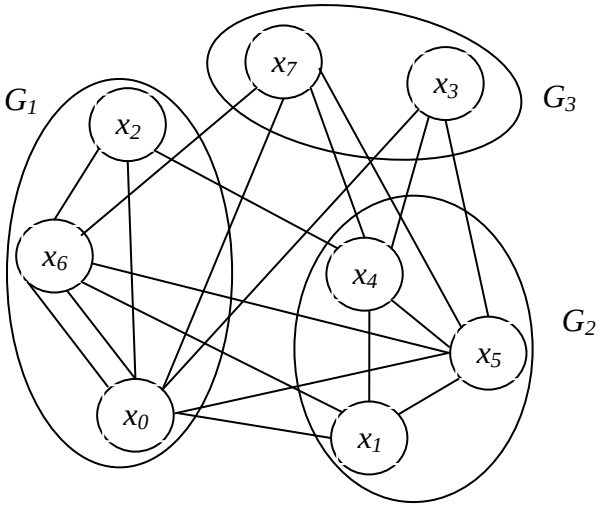


Рисунок 5.1. Результат компоновки последовательным алгоритмом

5.3. Итерационный алгоритм компоновки

Итерационные алгоритмы применяются для улучшения начальной компоновки. В качестве начальной компоновки может служить компоновка, полученная последовательным алгоритмом, компоновка, заданная конструктором, случайная компоновка.

Результат работы итерационного алгоритма зависит от начальной компоновки.

Любой итерационный алгоритм состоит из следующих шагов:

1. Получение очередного варианта решения;
2. Вычисление функции-критерия;
3. Выбор лучшего, в смысле п. 2, варианта;
4. Срабатывание правила останова (число шагов, время, $\Delta F \le \varepsilon$, перебор всех вариантов).

Итерационный алгоритм компоновки состоит в парных перестановках вершин разных кусков с проверкой на каждом шаге изменения числа внешних связей. Цель итерации – минимизация числа связей между кусками.

Алгоритм включает следующие процедуры:

1. Из множества кусков выбираются любые два.
   Например, $G_1(X_1, U_1)$ и $G_2(X_2, U_2)$.
2. Для всех вершин выбранных кусков подсчитывается характеристика

$$\alpha(x_i) = \begin{cases} \sum\limits_{x_j \in X_2} r_{ij} - \sum\limits_{x_j \in X_1} r_{ij}, \text{ если } x_i \in X_1, \\ \sum\limits_{x_j \in X_1} r_{ij} - \sum\limits_{x_j \in X_2} r_{ij}, \text{ если } x_i \in X_2. \end{cases}$$

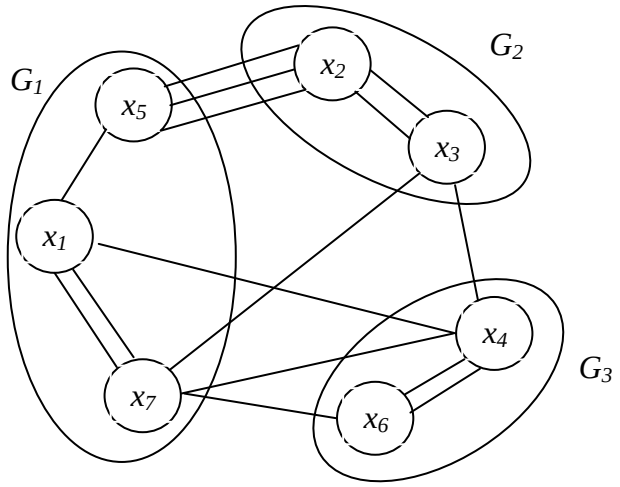
Характеристика $\alpha(x_i)$ показывает, как уменьшится количество внешних связей, если вершину x_i переместить из одного куска в другой.

3. Для каждой пары вершин x_i и x_j из разных кусков подсчитывается значение
   $b_{ij} = \alpha(x_i) + \alpha(x_j) - 2r_{ij}$.

Эта характеристика показывает, как уменьшится количество внешних связей, если вершины x_i и x_j поменять местами.

4. Выбирается пара вершин x_i и x_j, для которых b_{ij} максимально. Эти вершины переставляются.
5. Вычисление b_{ij} повторяется для вновь образованных кусков $G^*_1(X^*_1, U^*_1)$ и $G^*_2(X^*_2, U^*_2)$.
6. П.п. 4, 5 повторяются до тех пор, пока не окажется для всех пар вершин $b_{ij} \le 0$, т.е. никакая перестановка не уменьшит колическтво внешних связей между кусками.
7. Выбирается любая другая пара кусков и п.п. 1-6 повторяются.
8. Процесс повторяется для всех пар кусков, т. е. $k(k-1)/2$ раз.

Пример. Пусть последовательным алгоритмом компоновки получено следующее разбиение (рис. 5.2).



$$R = \begin{array}{c|ccccccc} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \hline x_1 & 0 & 0 & 0 & 1 & 1 & 0 & 2 \\ x_2 & 0 & 0 & 2 & 0 & 3 & 0 & 0 \\ x_3 & 0 & 2 & 0 & 1 & 0 & 0 & 1 \\ x_4 & 1 & 0 & 1 & 0 & 0 & 2 & 1 \\ x_5 & 1 & 3 & 0 & 0 & 0 & 0 & 0 \\ x_6 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \\ x_7 & 2 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}$$

Рисунок 5.2. Начальное разбиение графа и матрица соединений

Выберем два куска $G_1(X_1, U_1)$ и $G_2(X_2, U_2)$. Для вех верши кусков вычислим $\alpha(x_i)$.

$$R' = \begin{array}{c|ccccc|c} & x_1 & x_5 & x_7 & x_2 & x_3 & \alpha(x_i) \\ \hline x_1 & 0 & 1 & 2 & 0 & 0 & -3 \\ x_5 & 1 & 0 & 0 & 3 & 0 & 2 \\ x_7 & 2 & 0 & 0 & 0 & 1 & -1 \\ \hline x_2 & 0 & 3 & 0 & 0 & 2 & 1 \\ x_3 & 0 & 0 & 1 & 2 & 0 & -1 \end{array}$$

Для всех пар вершин разных кусков вычислим b_{ij} и результаты сведем в таблицу.

$b_{12} = \alpha(x_1) + \alpha(x_2) - 2r_{12} = -3 + 1 - 0 = -2$;

$b_{13} = \alpha(x_1) + \alpha(x_3) - 2r_{13} = -3 + -1 - 0 = -4$;

$b_{52} = \alpha(x_5) + \alpha(x_2) - 2r_{52} = 2 + 1 - 6 = -3$ и т.д.

$$B = \begin{array}{c|cc} & x_2 & x_3 \\ \hline x_1 & -2 & -4 \\ x_5 & -3 & 1 \\ x_7 & 0 & -4 \end{array}$$

Положительный эффект дает только перестановка вершин x_5 и x_3. Поменяем их местами и пересчитаем $\alpha(x_i)$.

$R_{12} =$

	x_1	x_3	x_7	x_2	x_5	$\alpha(x_i)$
x_1	0	0	2	0	1	−1
x_3	0	0	1	2	0	1
x_7	2	1	0	0	1	−2
x_2	0	2	0	0	3	−1
x_5	1	0	0	3	0	−2

Пересчитаем b_{ij}

$B =$

	x_2	x_5
x_1	−2	−5
x_3	0	−1
x_7	−3	−4

Все $b_{ij} \leq 0$, поэтому перестановки вершин не приведут к уменьшению числа внешних связей.

Выберем другие два куска $G_1(X_1, U_1)$ и $G_3(X_3, U_3)$ и вычислим $\alpha(x_i)$.

$R_{13} =$

	x_1	x_3	x_7	x_4	x_6	$\alpha(x_i)$
x_1	0	0	2	1	1	0
x_3	0	0	1	1	0	0
x_7	2	1	0	1	1	−1
x_4	1	1	1	0	2	1
x_6	1	0	1	2	0	0

Для всех пар вершин разных кусков вычислим b_{ij} и результаты сведем в таблицу:

$B =$

	x_4	x_6
x_1	−1	−2
x_3	−1	0
x_7	−2	−3

Все $b_{ij} \leq 0$. Выберем нерассмотренные куски $G_2(X_2, U_2)$ и $G_3(X_3, U_3)$ и вычислим $\alpha(x_i)$.

$R_{23} =$

	x_2	x_5	x_4	x_6	$\alpha(x_i)$
x_2	0	3	0	0	−3
x_5	3	0	0	0	−3
x_4	0	0	0	2	−2
x_6	0	0	2	0	−2

Все $\alpha(x_i)$ и, соответственно, $b_{ij} \leq 0$, работа алгоритма заканчивается. В результате получили разрезание (рис. 5.3 (*а*)). Начальная компоновка имела 8 внешних связей, после применения итерационного алгоритма число внешних связей уменьшилось до 7. Оптимальная компоновка с 6 внешними связями приведена на рис. 5.3 (*б*).

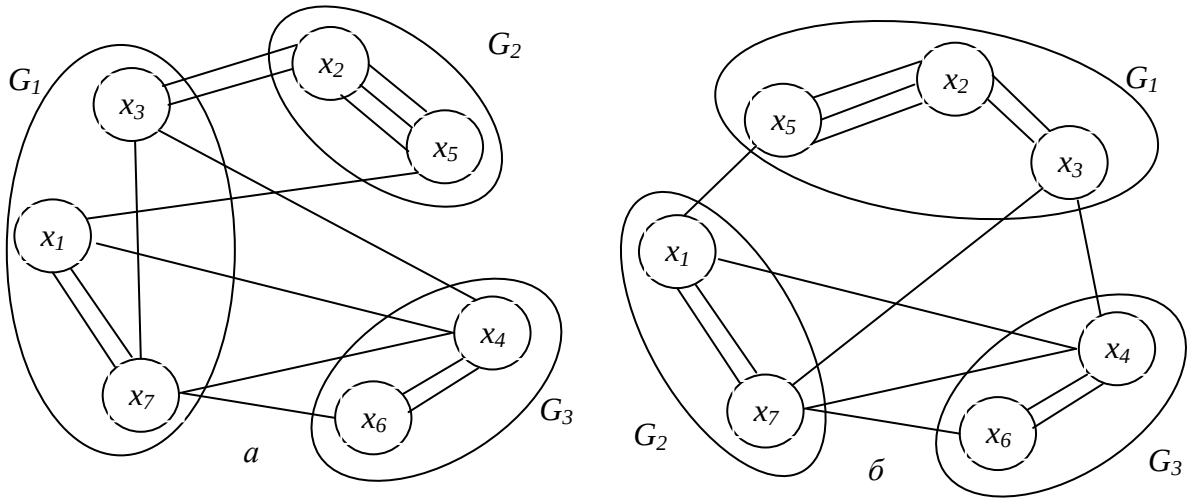


Рисунок 5.3. Улучшенное разбиение графа (*а*) и оптимальная компоновка (*б*)

5.4. Смешанный алгоритм компоновки

Смешанный алгоритм компоновки основан на идеях итерационного алгоритма, но для него не нужно начального разрезания. Он работает со всей матрицей соединений.

Алгоритм включает следующие шаги.

1. В матрице R выделяется подматрица порядка, равного числу вершин n_1 в G_1. Матрица R при этом разбивается на две подматрицы: R_1 и R_2.
2. По матрице R находится характеристики $\alpha(x_i)$.
3. Строится матрица B.
4. Определяется максимальный элемент $b_{ij} \geq 0$. Если в матрице B нет положительных элементов, по переход к п. 6. Переставляются вершины x_i и x_j. Получена матрица R'.
5. Последовательно выполняются п.п. 2-4 до тех пор, пока окажется для всех пар вершин $b_{ij} \leq 0$.
6. Из графа G удаляется часть G_1, соответствующая подматрице R_1.
7. Повторяется работа п.п. 1-6 с матрицами $R_2, R_3, \ldots, R_{k-1}$.
8. Разбиение получено.

Продемонстрируем работу алгоритма на примере компоновки рис. 5.2.

1. В матрице R выделим подматрицу порядка $n_1 = 3$ и вычислим $\alpha(x_i)$.

$R =$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\alpha(x_i)$
x_1	0	0	0	1	1	0	2	4
x_2	0	0	2	0	3	0	0	1
x_3	0	2	0	1	0	0	1	0
x_4	1	0	1	0	0	2	1	−1
x_5	1	3	0	0	0	0	0	4
x_6	0	0	0	2	0	0	1	−3
x_7	2	0	1	1	0	1	0	1

2. Для всех пар вершин разных кусков вычислим b_{ij}

$$B = \begin{array}{c|cccc} & x_4 & x_5 & x_6 & x_7 \\ \hline x_1 & 1 & 6 & 1 & 1 \\ x_2 & 0 & -1 & -2 & 2 \\ x_3 & -3 & 4 & -3 & -1 \end{array}$$

3. Max $b_{15} = 6$. В матрице R переставляем строки и столбцы, соответствующие элементам x_1 и x_5 и пересчитываем $\alpha(x_i)$.

$$R = \begin{array}{c|cccccccc} & x_5 & x_2 & x_3 & x_4 & x_1 & x_6 & x_7 & \alpha(x_i) \\ \hline x_5 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & -2 \\ x_2 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & -5 \\ x_3 & 0 & 2 & 0 & 1 & 0 & 0 & 1 & 0 \\ \hline x_4 & 0 & 0 & 1 & 0 & 1 & 2 & 1 & -3 \\ x_1 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & -2 \\ x_6 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & -3 \\ x_7 & 0 & 0 & 1 & 1 & 2 & 1 & 0 & -2 \end{array}$$

4. Все $\alpha(x_i) \le 0$. Первый кусок сформирован $X_1 = \{x_2, x_3, x_5\}$.
5. Удаляем из графа G подграф G_1.
6. В матрице R' выделим подматрицу порядка $n_2 = 2$ и вычислим $\alpha(x_i)$.

$$R' = \begin{array}{c|cccc} & x_4 & x_1 & x_6 & x_7 & \alpha(x_i) \\ \hline x_4 & 0 & 1 & 2 & 1 & 2 \\ x_1 & 1 & 0 & 0 & 2 & 1 \\ \hline x_6 & 2 & 0 & 0 & 1 & 1 \\ x_7 & 1 & 2 & 1 & 0 & 2 \end{array}$$

7. Для всех пар вершин разных кусков вычислим b_{ij}

$$B = \begin{array}{c|cc} & x_6 & x_7 \\ \hline x_4 & -1 & 2 \\ x_1 & 2 & -1 \end{array}$$

8. Max $b_{16} = b_{47} = 2$. Поменяем местами x_1 и x_6.

$$R' = \begin{array}{c|cccc} & x_4 & x_6 & x_1 & x_7 & \alpha(x_i) \\ \hline x_4 & 0 & 2 & 1 & 1 & 0 \\ x_6 & 2 & 0 & 0 & 1 & -1 \\ \hline x_1 & 1 & 0 & 0 & 2 & -1 \\ x_7 & 1 & 1 & 2 & 0 & 0 \end{array}$$

9. Все $\alpha(x_i) \le 0$. Сформирован второй кусок $X_2 = \{x_4, x_6\}$. Оставшиеся вершины образуют третий кусок $X_3 = \{x_1, x_7\}$.

Получено разрезание графа G, совпадающее с оптимальным (рис. 5.3 (*б*)).

6. Алгоритмы размещения элементов

6.1. Постановка задачи

Задачи размещения элементов и трассировки их соединений тесно связаны и при обычных ("ручных") методах конструирования решаются одновременно. В процессе размещения элементов уточняются трассы соединений, после чего положение некоторых элементов может корректироваться. В зависимости от принятой конструктивно - технологической и схемотехнической базы при решении этих задач используются различные критерии и ограничения. В результате получается точное пространственное расположение отдельных элементов конструктивного узла и геометрически определенный способ соединений выводов этих элементов.

Критерии качества и ограничения, связанные с конкретными задачами размещения и трассировки, опираются на конкретные конструктивные и технологические особенности реализации коммутационной части узла. Всю совокупность критериев и ограничений можно разделить на две группы в соответствии с метрическими и топологическими параметрами конструкции узлов и схем.

К метрическим параметрам относятся размеры элементов и расстояния между ними, размеры коммутационного поля, расстояния между выводами элементов, допустимые длины соединений и т.д.

Топологические параметры в основном определяются принятым в конкретной конструкции способом устранения пересечений соединений и относительным расположением соединений на коммутационном поле. К ним относятся: число пространственных пересечений соединений, число межслойных переходов, близость расположения друг к другу тепловыделяющих элементов или несовместимых в электромагнитном отношении элементов и соединений.

В конкретных задачах указанные параметры в различных сочетаниях могут быть либо главными критериями оптимизации, либо выступать в качестве ограничений.

Совместное решение задач размещения и трассировки представляет значительные трудности ввиду сложного характера взаимосвязи между отдельными параметрами конструкции и схем соединений. В связи с этим при алгоритмическом подходе к их решению они рассматриваются, как правило, раздельно. Сначала осуществляется *размещение* элементов, а затем *трассировка межсоединений*. Если необходимо, этот процесс может быть повторен при другом расположении отдельных элементов.

Основной целью размещения считают создание наилучших условий для последующей трассировки соединений при удовлетворении основных требований, обеспечивающих работоспособность схем.

В общем виде задачу размещения элементов узла описывают следующим образом.

Дано множество конструктивных элементов, связанных между собой в соответствии с принципиальной электрической схемой узла. Требуется разместить элементы на некотором плоском коммутационном поле (КП) таким образом, чтобы некоторый функционал достигал экстремального значения.

Исходными данными для размещения являются: данные о конфигурации и размерах КП (установка и крепление); количество и размеры элементов; схема соединений; ограничения на установку элементов.

Вариантами КП могут быть панель с проводными соединениями, печатная плата, подложка микросборки, кристалл БИС.

Основная сложность в постановке задачи размещения заключается в выборе целевой функции. Это связано с основной целью размещения, которую нельзя оценить без проведения трассировки. Особенностью критериев, используемых в задаче размещения, является их эвристический характер, т. к. все они косвенно учитывают условия трассировки. Классическим критерием является критерий суммарной длины соединений (СДС), который интегральным образом учитывает многочисленные требования, предъявляемые к расположению элементов и трасс их соединений. Это обуславливается рядом факторов:

- уменьшение длин соединений улучшает электрические параметры схемы;
- чем меньше суммарная длина соединений, тем, в среднем, проще их реализация в процессе трассировки;
- уменьшение суммарной длины соединений снижает трудоемкость изготовления монтажных схем, особенно схем проводного монтажа;
- данный критерий относительно прост с математической точки зрения и позволяет косвенным образом учитывать другие параметры схем путем присвоения весовых оценок отдельным соединениям.

Кроме СДС в САПР нашли применение следующие критерии:

- расстояние между элементами, соединенными наибольшим числом связей;
- число пересечений проводников на КП;
- длина наиболее длинных связей;
- число перегибов проводников;
- число межслойных переходов;
- параметры паразитных связей между элементами и проводниками;
- равномерность распределения температуры по поверхности КП и др.

Всю совокупность алгоритмов размещения можно разделить на следующие основные группы:

1. алгоритмы решения математических задач, являющихся моделями задач размещения;
2. конструктивные алгоритмы начального размещения;
3. итерационные алгоритмы улучшения начального размещения;
4. непрерывно - дискретные методы размещения.

К первой группе относится, прежде всего, метод ветвей и границ для задачи квадратичного назначения, к которой при определенных упрощениях сводится задача размещения: набор позиций считается фиксированным, элементы рассматриваются как геометрические точки, схема соединений представляется взвешенным графом соединений.

Вторая и третья группы включают приближенные алгоритмы, в основном предназначенные для оптимизации размещения элементов в фиксированном наборе позиций.

Характерной особенностью конструктивных алгоритмов является то, что они создают размещение. Итерационные алгоритмы предполагают задание начального размещения.

Конструктивные алгоритмы используют последовательный или параллельно - последовательный процесс установки элементов в позиции при локальной оптимизации функции - критерия размещения.

В итерационных алгоритмах производится переразмещение элементов или их групп с целью минимизации выбранного критерия. Эти алгоритмы требуют существенных затрат машинного времени и используются для получения окончательного размещения.

Основной областью применения непрерывно - дискретных методов размещения являются конструкции, в которых позиции для установки элементов заранее не фиксированы. Исходной базой для построения алгоритмов данной группы являются непрерывные модели и механические аналогии задачи размещения.

6.2. Математическая модель задачи размещения

Пусть, даны множество элементов $E=\{e_1, e_2, \ldots, e_m\}$ и множество фиксированных позиций для размещения элементов $P\{ p_1, p_2, \ldots, p_l\}$ ($l \geq m$, будем считать, что $l = m$). Схема задана матрицей соединений $R=\|r_{ij}\|_{m\times m}$, а расстояние между позициями матрицей расстояний $D=\|d_{ij}\|_{l\times l}$. Для вычисления элементов матрицы D будем пользоваться ортогональной метрикой $d_{ij} = |x_i - x_j| + |y_i - y_j|$, где $(x_i, y_i), (x_j, y_j)$ координаты позиций.

Длина соединений между элементами e_i и e_j, помещенными в позиции $p(i)$ и $p(j)$, соответственно, оценивается величиной $r_{ij}\times d_{p(i)p(j)}$. Учитывая симметричность матриц R и D, запишем выражение для суммарной длины соединений

$$F(P) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} r_{ij} \times d_{p(i)\,p(j)}.$$

Таким образом, задача размещения по критерию СДС состоит в минимизации функционала $F(P)$ на множестве перестановок P. Данная задача называется *задачей квадратичного назначения.* Для решения этой задачи предложен ряд алгоритмов, основанных на методе ветвей и границ.

6.3. Метод ветвей и границ

Основная идея метода заключается в разбиении всего множества допустимых решений на подмножества и просмотра каждого подмножества с целью выбора оптимального. Для всех решений вычисляется нижняя граница минимального значения целевой функции. Как только нижняя граница становится больше значения целевой функции для наилучшего из ранее известных, подмножество решений, соответствующее этой границе исключается из области решений. Это обеспечивает сокращение перебора.

Поиск продолжается до тех пор, пока не будут исключены все решения, кроме оптимального.

Различные модификации общего метода отличаются способом расчета нижних границ и способом разбиения поля решений.

Для описания процесса поиска оптимального размещения строится дерево решений. Ребра первого яруса дерева соответствуют назначениям элемента e_1 в позиции, второго – e_2 и т. д. Произвольному размещению элементов соответствует в дереве решений некоторый путь, исходящий из начальной вершины.

Для каждой вершины дерева можно рассчитать нижнюю границу целевой функции для множества путей, связанных с этой вершиной. Если эта граница больше значения целевой функции для известного размещения, то дальнейшее продвижение по дереву в этой вершине прекращается.

Для вычисления нижней границы можно воспользоваться следующим свойством: если r = $\{r_1, r_2, ..., r_m\}$ и d = $\{d_1, d_2, ..., d_m\}$ два вектора, то минимум скалярного произведения r и d, т. е. минимум $\sum_{i=1}^{m} r_i d_{p(i)}$ на множестве всех перестановок P, соответствует расположению состовляющих вектора r в невозрастающем порядке, а вектора d – в неубывающем.

Таким образом, простейшая нижняя граница может быть получена при рассмотрении верхних половин матриц R и D в качестве составляющих векторов r и d длиной $m(m\text{-}1) / 2$ и при выполнении указанного выше упорядочивания.

Пусть имеется некоторое частичное размещение q множества элементов E_k на множестве позиций P_k. Тогда нижняя граница складывается из следующих частей

$F(P) = F(q) + w(P) + v(P)$, где

$F(q) = \sum_{e_i \in E_k} \sum_{e_s \in E_k} r_{is} d_{p(i)p(s)}$ - суммарная длина соединений между размещенными элементами;

$w(q) = \sum_{e_i \in E_k} \sum_{e_s \notin E_k} r_{is} d_{p(i)p(s)}$ - суммарная длина соединений между размещенными и неразмещенными элементами;

$v(q) = \sum_{e_i \notin E_k} \sum_{e_s \notin E_k} r_{is} d_{p(i)p(s)}$ - суммарная длина соединений между неразмещенными элементами.

Пример. Разместить элементы, заданные взвешенным графом G (рис. 6.1. ($а$)) на множестве позиций P (рис. 6.1. ($б$)).

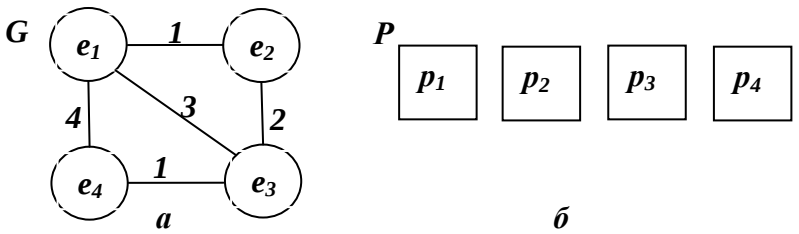


Рисунок 6.1. Взвешенный граф G ($а$) и множество позиций P ($б$)

Составим матрицы соединений R графа и расстояний D множества позиций.

$R=$	e_1	e_2	e_3	e_4
e_1	0	1	3	4
e_2		0	2	0
e_3			0	1
e_4				0

$D=$	p_1	p_2	p_3	p_4
p_1	0	1	2	3
p_2		0	1	2
p_3			0	1
p_4				0

Определим нижнюю границу целевой функции для этих исходных данных.

Для этого упорядочим составляющие вектора r в невозрастающем порядке, а вектора d – в неубывающем.

r = {4 3 2 1 1 0}

d = {1 1 1 2 2 3}

r×d = 4 + 3 + 2 + 2 + 2 + 0 = 13.

Это значит, что для этих исходных данных значение целевой функции $F(P)$ не может быть меньше 13.

1. Помещаем элемент e_1 в позицию p_1.

Т. к. размещен один элемент $F(q) = 0$.

Неразмещенные элементы $\{e_2, e_3, e_4\}$, свободные позиции $\{p_2, p_3, p_4\}$.

Составим вектор, соответствующий первой строке матрицы R r_1 ={4 3 1}, и вектор, соответствующий первой строке матрицы D d_1 ={1 2 3}, суммарная длина соединений между размещенным и неразмещенными элементами

$w(P) = r_1 \times d_1 = 4 + 6 + 3 = 13$.

Для оценки $v(P)$ вычеркнем из матриц R и D первые строки и столбцы и образуем вектора: r ={2 1 0} и d ={1 1 2}, соответствующие верхним половинам усеченных матриц R и D.

Получим $v(P)$ = r×d = 2 + 1 + 0 = 3.

Таким образом, нижняя граница $F(P) = 0 + 13 + 3 = 16$.

2. Помещаем элемент e_1 в позицию p_2. По-прежнему $F(q)=0$.

Неразмещенные элементы $\{e_2, e_3, e_4\}$, свободные позиции $\{p_1, p_3, p_4\}$.

Составим вектор, соответствующий первой строке матрицы R r_1 ={4 3 1}, и вектор, соответствующий второй строке матрицы D d_2 ={1 1 2}, суммарная длина соединений между размещенным и неразмещенными элементами

$w(P) = r_1 \times d_2 = 4 + 3 + 2 = 9$.

Для оценки $v(P)$ вычеркнем из матрицы R первые строку и столбец, а из матрицы D вторые строку и столбец. Образуем вектора: r={2 1 0} и d={1 2 3}, соответствующие верхним половинам усеченных матриц R и D. Получим $v(P)$ = r×d = 2 + 2 + 0 = 4. Таким образом, нижняя граница $F(P) = 0 + 9 + 4 = 13$.

Очевидно, что ввиду симметричности позиций (p_1 и p_4) и (p_2 и p_3) будут получены те же результаты для симметричных позиций. На рис. 6.2 представлены результаты расчета нижних границ для первого яруса дерева решений.

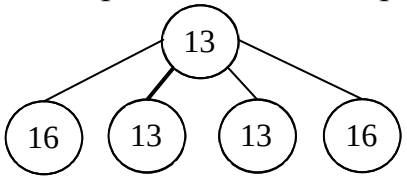


Рисунок 6.2. Нижние границы при размещении e_1

Назначаем элемент e_1 в позицию p_2.

3. Помещаем элемент e_2 в позицию p_1. Размещены два элемента: e_1 в позиции p_2 и e_2 в позиции p_1, $F(q) = r_{12}d_{21} = 1$.

Неразмещенные элементы $\{e_3, e_4\}$, свободные позиции $\{p_3, p_4\}$;

$r_1 = \{4\ \ 3\}$ и $d_2 = \{1\ \ 2\}$, $r_1 \times d_2 = 4 + 6 = 10$;

$r_2 = \{2\ \ 0\}$ и $d_1 = \{2\ \ 3\}$, $r_2 \times d_1 = 4 + 0 = 4$;

$w(P) = 10 + 4 = 14$.

$r = \{1\}$ и $d = \{1\}$, $v(P) = r \times d = 1$. $F(P) = 1 + 14 + 1 = 16$.

4. Помещаем элемент e_2 в позицию p_3. Размещены два элемента: e_1 в позиции p_2 и e_2 в позиции p_3, $F(q) = r_{12}d_{23} = 1$.

Неразмещенные элементы $\{e_3, e_4\}$, свободные позиции $\{p_1, p_4\}$;

$r_1 = \{4\ \ 3\}$ и $d_2 = \{1\ \ 2\}$, $r_1 \times d_2 = 4 + 6 = 10$;

$r_2 = \{2\ \ 0\}$ и $d_3 = \{1\ \ 2\}$, $r_2 \times d_3 = 2 + 0 = 2$;

$w(P) = 10 + 2 = 12$.

$r = \{1\}$ и $d = \{3\}$, $v(P) = r \times d = 3$. $F(P) = 1 + 12 + 3 = 16$.

5. Помещаем элемент e_2 в позицию p_4. Размещены два элемента: e_1 в позиции p_2 и e_2 в позиции p_4, $F(q) = r_{12}d_{24} = 2$.

Неразмещенные элементы $\{e_3, e_4\}$, свободные позиции $\{p_1, p_3\}$;

$r_1 = \{4\ \ 3\}$ и $d_2 = \{1\ \ 1\}$, $r_1 \times d_2 = 4 + 3 = 7$;

$r_2 = \{2\ \ 0\}$ и $d_4 = \{1\ \ 3\}$, $r_2 \times d_4 = 2 + 0 = 2$;

$w(P) = 7 + 2 = 9$.

$r = \{1\}$ и $d = \{2\}$, $v(P) = r \times d = 2$. $F(P) = 2 + 9 + 2 = 13$.

На рис. 6.3 представлены результаты расчета нижних границ для двух ярусов дерева решений.

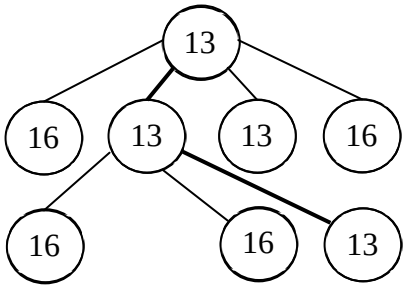


Рисунок 6.3. Нижние границы при размещении e_1 и e_2

Назначаем элемент e_2 в позицию p_4.

6. Помещаем элемент e_3 в позицию p_1. Размещены три элемента: e_1 в позиции p_2, e_2 в позиции p_4, и e_3 в позиции p_1, $F(q) = r_{12}d_{24} + r_{13}d_{21} + r_{23}d_{41} = 2 + 3 + 6 = 11$.

Неразмещенный элемент $\{e_4\}$, свободная позиция $\{p_3\}$;

$r_1 = \{4\}$ и $d_2 = \{1\}$, $r_1 \times d_2 = 4$;

$r_2 = \{0\}$ и $d_4 = \{1\}$, $r_2 \times d_4 = 0$;

$r_3 = \{1\}$ и $d_1 = \{2\}$, $r_2 \times d_1 = 2$;

$w(P) = 4 + 0 + 2 = 6$.

Неразмещенный элемент один, $v(P) = 0$. $F(P) = 11 + 6 + 0 = 17$.

7. Помещаем элемент e_3 в позицию p_3. Размещены три элемента: e_1 в позиции p_2, e_2 в позиции p_4, и e_3 в позиции p_3, $F(q) = r_{12}d_{24} + r_{13}d_{23} + r_{23}d_{43} = 2 + 3 + 2 = 7$.

Неразмещенный элемент $\{e_4\}$, свободная позиция $\{p_1\}$;

$r_1 = \{4\}$ и $d_2 = \{1\}$, $r_1 \times d_2 = 4$;

$r_2 = \{0\}$ и $d_4 = \{1\}$, $r_2 \times d_4 = 0$;

$r_3 = \{1\}$ и $d_3 = \{2\}$, $r_2 \times d_3 = 2$;

$w(P) = 4 + 0 + 2 = 6$.

Неразмещенный элемент один, $v(P) = 0$. $F(P) = 7 + 6 + 0 = 13$.

На рис. 6.4 представлены результаты расчета нижних границ для трех ярусов дерева решений.

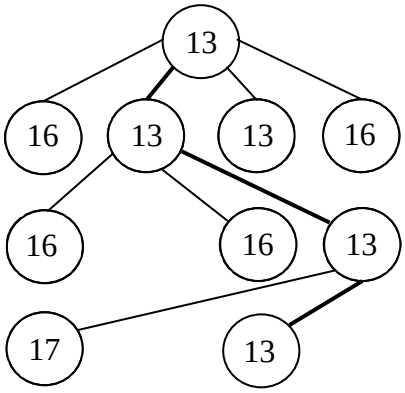


Рисунок 6.4. Нижние границы при размещении e_1, e_2 и e_3

Назначаем элемент e_3 в позицию p_3.

8. Неразмещенный элемент $\{e_4\}$, свободная позиция $\{p_1\}$.

Помещаем $\{e_4\}$ в позицию $\{p_1\}$.

$F(q) = r_{12}d_{24} + r_{13}d_{23} + r_{23}d_{43} + r_{14}d_{21} + r_{24}d_{41} + r_{34}d_{31} = 2 + 3 + 2 + 4 + 0 + 2 = 13$.

$w(P) = v(P) = 0$. $F(p) = 13$. Получено размещение (рис. 6.5):

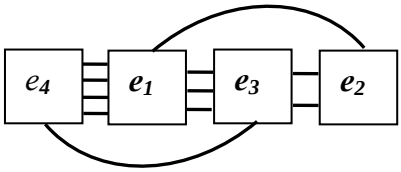


Рисунок 6.5. Размещение элементов

6.4. Конструктивные алгоритмы начального размещения

Характерной особенностью этого класса алгоритмов является то, что они создают размещение.

Среди конструктивных алгоритмов размещения выделяют последовательные и параллельно-последовательные алгоритмы.

Наиболее представительная группа конструктивных алгоритмов – алгоритмы последовательного размещения.

Рассмотрим алгоритм последовательного размещения по связности.

Вводится n - шаговый процесс принятия решений, на каждом шаге которого выбирается один из неразмещенных элементов и помещается в одну из незанятых позиций.

Структура любого последовательного алгоритма размещения определяется правилами выбора очередного элемента и позиции для его установки.

Пусть, E_k - элементы, размещенные за k шагов алгоритма, а P_k - позиции, занятые этими элементами.

Для каждого неразмещенного элемента $e_i \notin E_k$ подсчитывается характеристика

$$C_i = \sum_{e_j \in E_k} r_{ij}.$$

Очередным размещенным элементом является элемент, имеющий максимальную характеристику C_i, т.е. выбор элемента осуществляется по наибольшему числу связей с уже размещенными элементами.

Позиция-кандидат для установки e_i выбирается с учетом связности e_i с уже размещенными элементами.

Для каждой свободной позиции p_t вычисляется характеристика

$$F_t = \sum_{e_j \in E_k} r_{ij} d_{p(i)t}.$$

Выбирается та из позиций, для которой F_t минимальна.

Для экономии вычислений всегда целесообразно рассматривать не все множество позиций $p_t \notin P_k$, а лишь часть, находящихся на периферии множества P_k незанятых позиций.

В любом последовательном алгоритме размещения особо оговаривается правило размещения первого элемента. Первым размещается элемент, входящий в максимальное число цепей. Он помещается в центр коммутационного поля.

Достоинством последовательных алгоритмов является быстродействие.

Недостатком является последовательный характер, оптимизируется только размещение элемента-кандидата.

Для оптимизации размещения используются итерационные алгоритмы.

6.5. Алгоритм обратного размещения

Алгоритм обратного размещения принадлежит группе алгоритмов параллельно-последовательного размещения.

В методе обратного размещения осуществляются предварительные оценки каждого из размещаемых элементов и каждой свободной позиции, после чего все элементы размещаются одновременно.

Даны: матрица соединений R и матрица расстояний D.

Для каждого элемента e_i найдем суммарное число соединений с другими элементами

$$\mathrm{r}_i = \sum_{j=1}^{m} r_{ij}, \text{ где } i = 1, 2, \ldots, m.$$

Для каждой позиции p_i найдем суммарное расстояние до остальных позиций

$$d_i = \sum_{j=1}^{m} d_{ij}, \text{ где } i = 1, 2, \ldots, m.$$

Очевидно, что позиции в центральной части коммутационного поля имеют меньшую характеристику d_i, чем позиции на периферии. Естественно, что центральные позиции наиболее благоприятны для размещения сильно связанных элементов.

Учитывая условия минимальности скалярного произведения r×d, получим следующий алгоритм:

1. Упорядочить элементы e_i в порядке неубывания r_i;

2. Упорядочить позиции p_i в порядке невозрастания d_i;

3. i-ый элемент из упорядоченного списка элементов помещается в i-ую позицию из упорядоченного списка позиций.

Достоинство алгоритма – простота. Он может быть использован даже при ручном проектировании.

Недостаток – неоптимальность размещения.

Пример. Разместить элементы, заданные взвешенным графом G (рис. 6.6 (*а*)) на множестве позиций P (рис. 6.6 (*б*)).

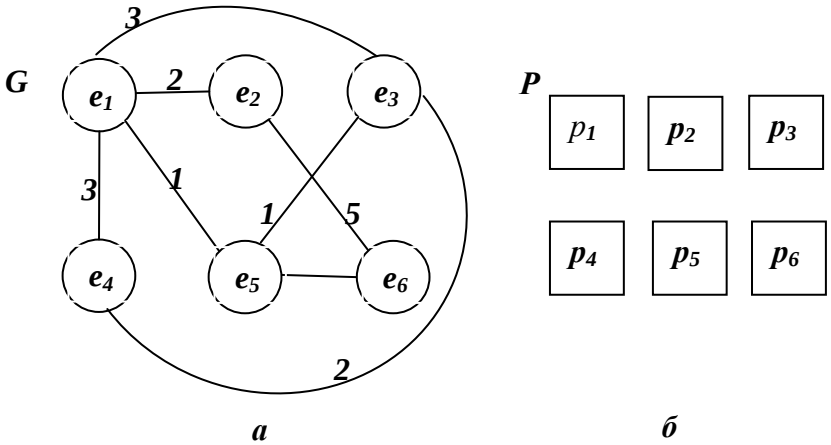


Рисунок 6.6. Взвешенный граф G (*а*) и множество позиций P (*б*)

Составим матрицы соединений R и расстояний D.

$R =$

	e_1	e_2	e_3	e_4	e_5	e_6	r_i
e_1	0	2	3	3	1	0	9
e_2		0	0	0	0	5	7
e_3			0	2	1	0	6
e_4				0	0	0	5
e_5					0	2	4
e_6						0	7

$D =$

	p_1	p_2	p_3	p_4	p_5	p_6	d_i
p_1	0	1	2	1	2	3	9
p_2		0	1	2	1	2	7
p_3			0	3	2	1	9
p_4				0	1	2	9
p_5					0	1	7
p_6						0	9

1. Упорядочим элементы e_i в порядке неубывания r_i $\{e_5, e_4, e_3, e_6, e_2, e_1\}$.
2. Упорядочим позиции p_i в порядке невозрастания d_i $\{p_1, p_3, p_4, p_6, p_2, p_5\}$.
3. Размещаем элементы в соответствии с упорядоченными списками (рис. 6.7).

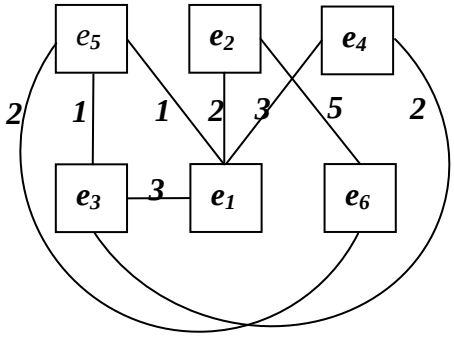


Рисунок 6.7. Результат размещения

Значение целевой функции для полученного размещения $F(p)$=36. Низкое качество размещения объясняется, прежде всего, небольшим количеством позиций, четыре позиции из шести имеют одинаковую характеристику d_i. Но этим можно воспользоваться для улучшения размещения. Можно менять позиции вершин, помещенные в равноценные позиции. Так, можно переставить вершины e_4 и e_6 (рис. 6.8 (*а*)). Целевая функция нового размещения $F(p)$=24. У оптимального размещения (рис. 6.8 (*б*)) значение целевой функции $F(p)$=22.

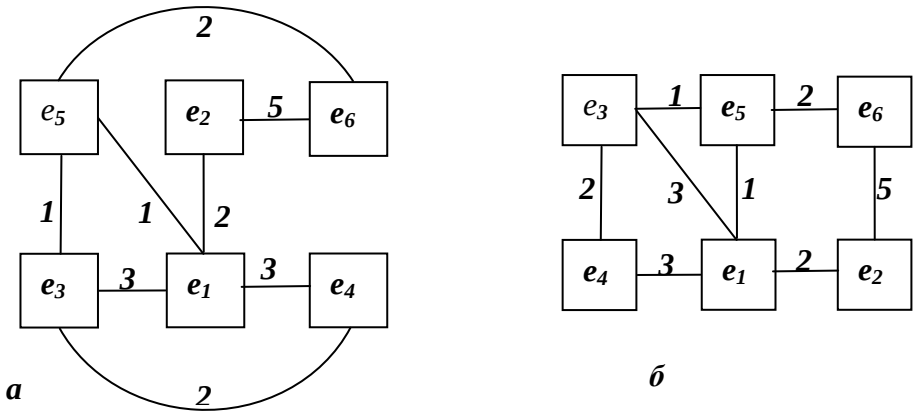


Рисунок 6.8. Улучшенное (*а*) и оптимальное (*б*) размещения

6.6. Итерационные алгоритмы улучшения начального размещения

Для этих алгоритмов необходимо задать начальный вариант размещения.

Итерационные алгоритмы применяются для решения задачи размещения с различными критериями оптимизации: суммарная длина соединений, наиболее длинная связь, суммарное число пересечений соединений и т.д.

В любом итерационном алгоритме исследуется подмножество размещений, в некотором смысле близких к начальному, для выделения в нем размещения с меньшим значением функции - критерия. Найденное размещение вновь принимается за начальное и процесс повторяется. Алгоритм завершается при отыскании некоторого размещения, в окрестности которого отсутствуют варианты с меньшим значением функции - критерия. В большинстве случаев такой процесс приводит к получению локального минимума функции - критерия.

В качестве начального может быть взято размещение, полученное конструктивным алгоритмом размещения, с помощью генератора случайных размещений или заданное конструктором.

Простейшим итерационным алгоритмом является *алгоритм парных перестановок*.

Он заключается в следующем: в начальном размещении меняются местами два элемента и для новой конфигурации вычисляется функция – критерий. Если наблюдается уменьшение функции, то новое размещение заменяет старое, в противном случае этого не происходит. Затем выбирается другая пара элементов и процесс повторяется до тех пор, пока не будет применено правило остановки.

Рассмотрим алгоритм парных перестановок, в котором в качестве функции – критерия используется максимальная длина соединений. Такие задачи называют минимаксными (минимизация максимальной длины соединений). Данный критерий позволяет легко определять пары элементов для перестановки.

Алгоритм состоит из следующих шагов:

1. Для каждой пары соединенных вершин e_i и e_j вычисляется длина проводника

$l_{ij} = |x_i - x_j| + |y_i - y_j|$, где x_i, y_i и x_j, y_j координаты вершин e_i и e_j в размещении.

2. Определяются $maxl_{ij}$ (если их несколько, то подсчитывается их количество k) и вершины e_i, e_j, дающие этот максимум.

3. Вершина e_i переставляется в размещении с соседней вершиной такой, что она находится слева или справа и сверху или снизу от e_i и ***ближе*** к e_j.

4. Для нового размещения повторяется п.1.

5. Определяются $maxl'_{ij}$ (и их количество k').

6. Если $maxl_{ij} > maxl'_{ij}$ или $maxl_{ij} = maxl'_{ij}$ и $k > k'$, то переход к п.3, в противном случае прежнее размещение восстанавливается.

7. Вершина e_j переставляется в размещении с соседней вершиной такой, что она находится слева или справа и сверху или снизу от e_j и ближе к e_i.

8. Пока есть не рассмотренные соседи вершины e_j выполняются п.п. 4-6.

9. Итерационный процесс заканчивается.

Пусть дано размещение рис. 6.9 (*а*).

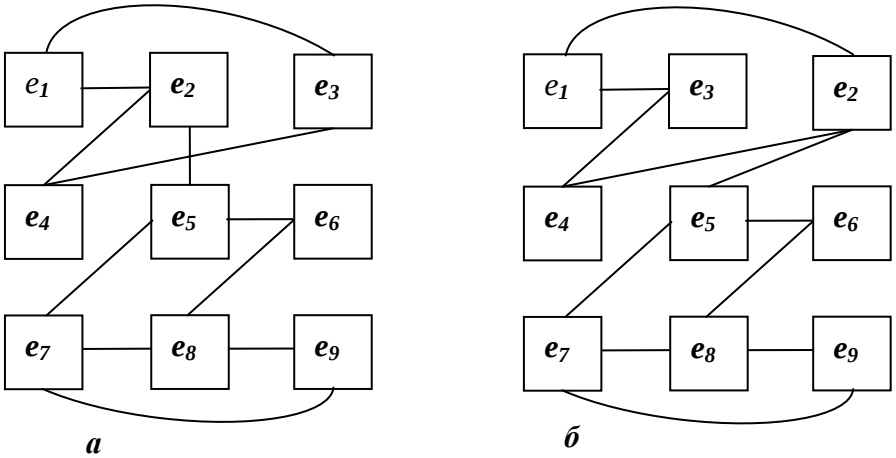


Рисунок 6.9. Положение элементов: до (*а*) и после перестановки (*б*)

Значение функции $F(p)=18$.

1. Для каждой пары соединенных вершин e_i и e_j вычислим l_{ij}:
   $l_{12}=1$; $l_{13}=2$; $l_{24}=2$; $l_{25}=1$; $l_{34}=3$; $l_{56}=1$; $l_{57}=2$; $l_{68}=2$; $l_{78}=1$; $l_{79}=2$; $l_{89}=1$.

$maxl_{ij} = l_{34} = 3$.

2. Меняем местами вершины e_3 и e_2 (рис. 6.9 (*б*)).
3. Для каждой пары соединенных вершин e_i и e_j вычислим новые l'_{ij}:
   $l_{12}=2$; $l_{13}=1$; $l_{24}=3$; $l_{25}=2$; $l_{34}=2$; $l_{56}=1$; $l_{57}=2$; $l_{68}=2$; $l_{78}=1$; $l_{79}=2$; $l_{89}=1$.
   $maxl'_{ij} = l_{24} = 3$. Максимальная длина не уменьшилась, возвращаем размещение (рис. 6.9 (*а*)).
4. Меняем местами вершины e_3 и e_6 (рис. 6.10 (*а*)).

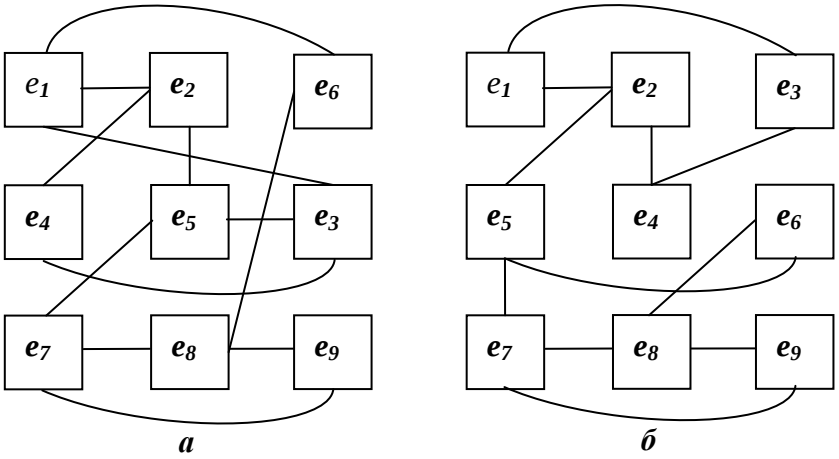


Рисунок 6.10. Перестановка элементов e_3 и e_6 (*а*); e_4 и e_5 (*б*)

5. Для каждой пары соединенных вершин e_i и e_j вычислим новые l'_{ij}:
   $l_{12}=1$; $l_{13}=3$; $l_{24}=2$; $l_{25}=1$; $l_{34}=2$; $l_{56}=2$; $l_{57}=2$; $l_{68}=3$; $l_{78}=1$; $l_{79}=2$; $l_{89}=1$.
   $maxl'_{ij} = l_{13} = l_{68} = 3$. Максимальная длина не уменьшилась, возвращаем размещение (рис. 6.9 (*а*)).
6. Меняем местами вершины e_4 и e_5 (рис. 6.10 (*б*)).
7. Для каждой пары соединенных вершин e_i и e_j вычислим новые l'_{ij}:
   $l_{12}=1$; $l_{13}=2$; $l_{24}=1$; $l_{25}=2$; $l_{34}=2$; $l_{56}=2$; $l_{57}=1$; $l_{68}=2$; $l_{78}=1$; $l_{79}=2$; $l_{89}=1$.
   $maxl'_{ij}=l_{13}=l_{25}=l_{34}= l_{56}= l_{68}= l_{79}=2$. Максимальная длина уменьшилась. Сохраняем полученное размещение. Количество максимумов $k = 6$.
8. Меняем местами вершины e_1 и e_2 (рис. 6.11 (*а*))

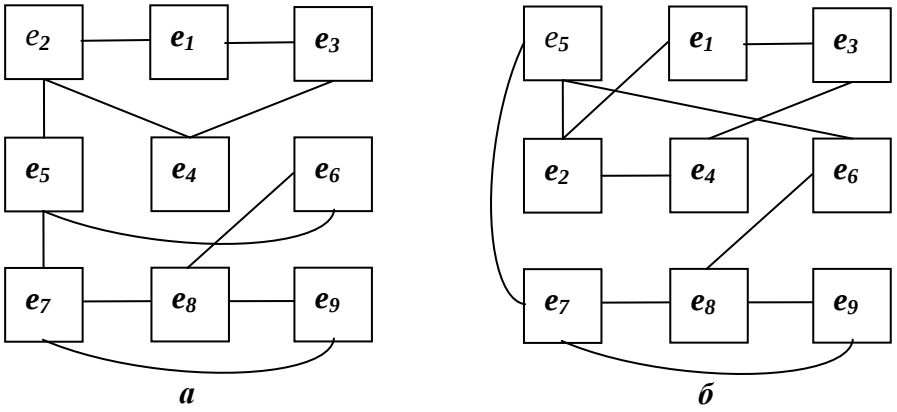


Рисунок 6.11. Перестановка элементов e_1 и e_2 (*а*); e_4 и e_5 (*б*)

9. Для каждой пары соединенных вершин e_i и e_j вычислим новые l'_{ij}:

l_{12}=1; l_{13}=1; l_{24}=2; l_{25}=1; l_{34}=2; l_{56}=2; l_{57}=1; l_{68}=2; l_{78}=1; l_{79}=2; l_{89}=1.

$maxl'_{ij}$ = 2. Максимальная длина не изменилась. Количество максимумов k'=5<k. Сохраняем полученное размещение, k=5.

10. $maxl'_{ij}$ = l_{24}= l_{34}=l_{56}= l_{68}= l_{79}=2. Меняем местами e_2 и e_1 и возвращаемся к размещению рис. 6.10 (*б*).
11. Меняем местами e_2 и e_5 (рис.6.12 (*б*)).
12. Для каждой пары соединенных вершин e_i и e_j вычислим новые l'_{ij}:

l_{12}=2; l_{13}=1; l_{24}=1; l_{25}=1; l_{34}=2; l_{56}=3; l_{57}=2; l_{68}=2; l_{78}=1; l_{79}=2; l_{89}=1.

$maxl'_{ij}$ = l_{56}=3. Максимальная длина увеличилась. Возвращаем размещение.

13. Меняем местами e_4 и e_5 (рис.6.12 (*а*)).
14. Для каждой пары соединенных вершин e_i и e_j вычислим новые l'_{ij}:

l_{12}=1; l_{13}=1; l_{24}=1; l_{25}=2; l_{34}=3; l_{56}=1; l_{57}=2; l_{68}=2; l_{78}=1; l_{79}=2; l_{89}=1.

$maxl'_{ij}$ = l_{34}=3. Максимальная длина увеличилась. Возвращаем размещение.

15. Меняем местами e_4 и e_1 (рис.6.12 (*б*)).
16. Для каждой пары соединенных вершин e_i и e_j вычислим новые l'_{ij}:

l_{12}=2; l_{13}=2; l_{24}=1; l_{25}=1; l_{34}=1; l_{56}=2; l_{57}=1; l_{68}=2; l_{78}=1; l_{79}=2; l_{89}=1.

17. $maxl'_{ij}$ = 2. Максимальная длина не изменилась. Количество максимумов k' = 5 = k.

Возвращаем размещение рис. 6.11 (*а*).

Длину соединения (e_2, e_1) уменьшить не удалось. Алгоритм заканчивает работу.

Значение функции $F(p)$=16.

Заметим, что если в размещении рис. 6.11 (*а*) поменять местами элементы e_8 и e_9 (рис. 6.12 (*в*)), то количество максимумов уменьшится (k' =4). Значение функции $F(p)$=15. Но максимальная длина останется такой же, поэтому алгоритм заканчивает работу при первой неудачной попытке уменьшения $maxl'_{ij}$.

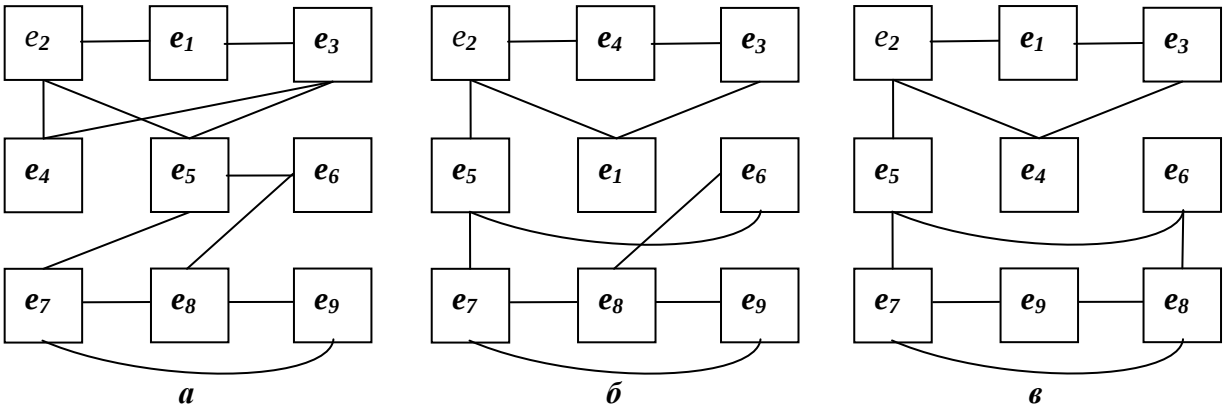


Рисунок 6.12. Перестановка элементов e_4 и e_5 (*а*); e_4 и e_1 (*б*); e_8 и e_9 (*в*)

6.7. Алгоритм групповых перестановок

Классическим алгоритмом такого типа является алгоритм Штейнберга. В алгоритме используется тот факт, что для множества несвязных между собой элементов задача минимизации СДС сводится к задаче о линейном назначении.

Пусть W={e_1, e_2, …, e_k} - некоторое несвязное множество элементов, а L={p_1, p_2, ..., p_k} – множество позиций, занятых этими элементами в исходном

размещении. Можно составить матрицу $A = \|a_{ij}\|_{k\times k}$, элемент a_{ij} которой представляет собой суммарную длину соединений элемента $e_i \in W$, при условии его установки в позицию $p_j \in L$. Поскольку элементы в W не связаны, значение a_{ij} не зависит от положения других элементов W. Решая задачу линейного назначения для матрицы A, получим оптимальное размещение элементов из W в позициях L, для которого СДС оптимальна.

Последовательно исследуются различные несвязные множества элементов, для которых решается задача *линейного назначения*.

В целом алгоритмы размещения с групповыми перестановками характеризуются значительными временными затратами. Эффективность алгоритма $O(m^3)$, где $m = |W|$.

Среди итерационных алгоритмов наиболее эффективны алгоритмы парных перестановок. Они позволяют уменьшить длину межсоединений от 1% до 50% в зависимости от качества начального размещения. Наибольшая скорость уменьшения длины соединений наблюдается на первых итераций. Длина монотонно уменьшается к значениям, близким к 1% при числе итераций $n > 5$.

6.8. Непрерывно-дискретные методы размещения

Для данного класса алгоритмов в отличие от рассмотренных, задание фиксированного набора позиций не обязательно - размещение осуществляется на непрерывной плоскости.

Примером алгоритмов данного класса может служить *метод силовых функций*. Он основан на следующей механической аналогии. Элементы считаются материальными точками, на которые действуют силы притяжения и отталкивания. Сила притяжения между элементами e_i и e_j полагается пропорциональной расстоянию между ними. Силы отталкивания вводятся для предотвращения слияния или перекрытия элементов. Далее осуществляется поиск состояния равновесия системы материальных точек, которое и определяет размещение элементов на плоскости.

Сила притяжения вычисляется следующим образом:

$F_{ij} = k_f r_{ij} d_{p(i)\,p(j)}$, где k_f – коэффициент.

Сила отталкивания вычисляется следующим образом:

$\Phi_{ij} = k_\varphi / d_{p(i)\,p(j)}$, где k_φ – коэффициент.

Составляется система дифференциальных уравнений, описывающая движение материальных точек к положению равновесия.

После решения системы, каждый из элементов e_i, имеющий координаты (x_i, y_i), перемещается в ту из незанятых позиций p_s, для которой изменение суммарной длины соединений минимально.

Метод силовых функций трудоемок и требует подбора коэффициентов опытным путем.

Метод ветвей и границ и непрерывно-дискретные алгоритмы размещения из-за их трудоемкости применяются лишь для задач небольшой размерности ($n \le 20$).

Достаточные результаты достигаются сочетанием конструктивного алгоритма (сложность ~ $O(n)$) с улучшающим итерационным (сложность ~ $O(n^2) \div O(n^4)$).

6.9. Размещение разногабаритных элементов

Рассмотренные ранее алгоритмы не учитывали габариты размещаемых элементов. Задача размещения существенно усложняется при необходимости размещения разногабаритных элементов на плате при нефиксированных позициях для установки элементов.

Различают размещение элементов кратных габаритов и некратных габаритов (существенно разногабаритных).

В первом случае задача сводится к целочисленной задаче размещения с ограничением, при котором один элемент может занимать несколько позиций (рис. 6.13 (*а*)) или в одну позицию может быть помещено несколько элементов (рис. 6.13 (*б*)).

Задача размещения существенно разногабаритных элементов имеет сходство с известной задачей о раскрое материала, т. к. в обоих случаях возникает требование эффективного использования площади. Однако математически обе задачи формулируются по-разному: эффективное использование площади является функционалом для задачи о раскрое и ограничением для задачи размещения. Функционалом задачи размещения может служить СДС.

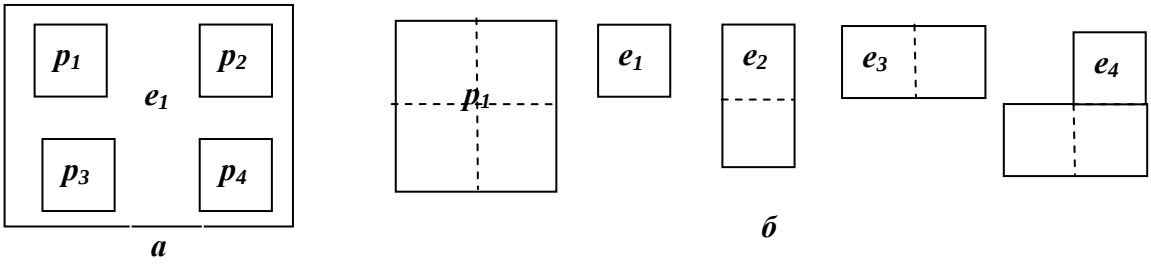


Рисунок 6.13. Разногабаритные элементы с кратными габаритами, занимающими несколько позиций (*а*) или – часть позиции (*б*)

Рассмотрим алгоритм нисходящего проектирования размещения разногабаритных элементов, основанный на последовательном разбиении КП. Алгоритм является вариантом алгоритма дихотомического деления.

Пусть задано множество элементов $E=\{e_1, e_2, ..., e_m\}$ с размерами l^i_x и l^i_y. Вначале рассчитываются размеры поля для размещения элементов

$$L_x = L_y = \sqrt{\sum_{i=1}^{m} l^i_x l^i_y}.$$

Далее множество элементов разбивается на две малосвязанные группы с помощью одного из алгоритмов компоновки. Особенностью является то, что в качестве ограничения размера группы выступает суммарная площадь входящих в нее элементов. Обозначим через $S(E_1)$ и $S(E_2)$ суммарные площади элементов, входящих в группы E_1 и E_2. Тогда

$S(E_1) + S(E_2) = S(E)$ и $|S(E_1) - S(E_2)| \le C$, где C – заданный параметр.

Проведем вертикальный разрез КП таким образом, чтобы площади левой и правой частей были равны соответственно $S(E_1)$ и $S(E_2)$. Координата разреза

$x = S(E_1)/L_y$.

На следующем шаге аналогичная процедура применяется к группам элементов E_1 и E_2, однако теперь уже проводятся горизонтальные разрезы.

Процесс разбиения продолжается до тех пор, пока каждая из полученных групп не будет состоять из одного элемента. Пример размещения разногабаритных элементов приведен на рис. 6.14.

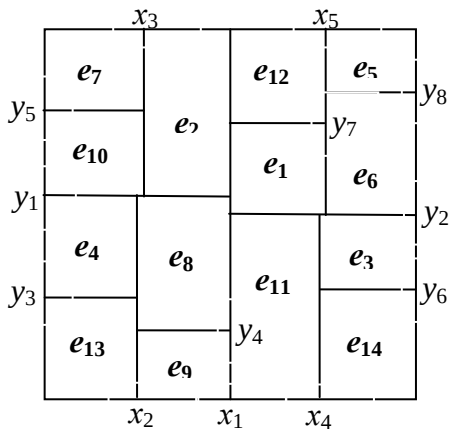


Рисунок 6.14. Размещение, полученное методом дихотомического деления

7. Алгоритмы трассировки межсоединений

Трассировка монтажных соединений - это задача геометрического построения на КП всех цепей данной конструкции, координаты начала и конца которых определены при размещении элементов. Следовательно, задача трассировки состоит в отыскании геометрически определенного способа соединений эквипотенциальных выводов схемы.

При этом необходимо учитывать различные конструктивно-технические ограничения: допускаются пересечения или нет, возможен ли переход с одного слоя на другой, сколько слоев отводится для трассировки, допустимые ширина проводников и расстояния между ними и т. д.

Алгоритмы трассировки существенно зависят от принятой конструкции и технологии изготовления электронной аппаратуры.

В табл. 7.1. приведены основные критерии трассировки в зависимости от технологии изготовления.

Таблица 7.1. Критерии трассировки межсоединений

Технология изготовления	Критерии
Проводной монтах	СДС, максимальная длина связи
ДПП, МПП (МСО)	СДС, число переходных отверстий
МПП (ОКП)	СДС, число слоев
ГИС	число пересечений проводников, площадь кристалла
ИС, БИС, СБИС	число пересечений проводников, площадь кристалла, число межслойных переходов

ДПП – двусторонние печатные платы; МПП – многослойные ПП; МСО - металлизация сквозных отверстий; ОКП – открытые контактные площадки; ГИС – гибридные интегральные схемы.

Этап трассировки является одним из самых трудоемких этапов конструкторского проектирования электронной аппаратуры.

Задачу трассировки можно условно представить в виде четырех последовательно выполняемых этапов, отвечающих на вопросы:

A.Что? (Определение перечня соединений);

B.Где? (Распределение соединений по слоям);

C.Когда? (Определение порядка проведения соединений);

D.Как? (Трассировка проводников).

A. На первом этапе формируется точный список, указывающий, какие соединения (цепи или фрагменты цепей схемы) должны быть проведены. Для решения этой задачи используют алгоритмы построения минимальных связывающих деревьев (МСД).

7.1. Алгоритмы построения минимальных связывающих деревьев

Для построения МСД разработан ряд полиномиальных алгоритмов.

7.1.1. Алгоритм Прима

Алгоритм Прима относится к так называемым "жадным" алгоритмам. Жадные алгоритмы действуют, используя в каждый момент лишь часть исходных данных и принимая лучшее решение на основе этой части. В нашем случае мы будем на каждом шаге рассматривать множество ребер, допускающих присоединение к уже построенной части связывающего дерева, и выбирать из них ребро с наименьшим весом. Повторяя эту процедуру, мы получим остовное дерево с наименьшим весом.

Начнем с произвольной вершины графа x_i и включим ее в остовное дерево. Из всех вершин, соединенных с данной ($x_j \in \Gamma x_i$), ищем вершину, соединенную ребром с наименьшим весом. Это ребро вместе с новой вершиной добавляется в дерево. Из вершин, не вошедших в дерево, ищем вершину, соединенную с уже построенной частью остовного дерева ребром с наименьшим весом. Это ребро вместе с новой вершиной добавляется в дерево и т. д. После того, как в дерево попадут все вершины, работа будет закончена.

Пример. Исходный взвешенный граф изображен на рис.7.1 (*а*).

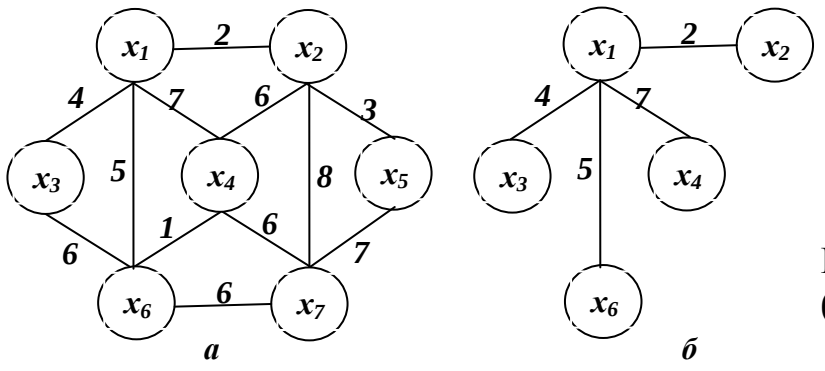


Рисунок 7.1. Исходный граф (*а*) и ребра–кандидаты (*б*)

Ребра, вошедшие в дерево, на рисунках будем отмечать жирной линией.

В начале процесса выбираем произвольную вершину, например, x_1. Выбираем вершины, непосредственно связанные с x_1 (рис.7.1.(*б*)). Ребро наименьшего веса связывает вершины x_1 и x_2, поэтому к уже построенной части МСД добавляется вершина x_2 вместе с ребром (x_1 x_2). При добавлении к дереву вершины x_2 необходимо определить, не следует ли добавить в кандидаты новые ребра. В результате обнаруживаем, что это необходимо проделать с ребрами (x_2 x_5) и (x_2 x_7). Здесь же необходимо проверить, являются ли ребра, ведущие из вершины x_1, в x_3, x_4 и x_7, кратчайшими среди ребер, соединяющих эти вершины с деревом, или есть более удобные ребра, исходящие из x_2. Ребро (x_2 x_4) короче ребра (x_1 x_4), и поэтому необходимо его заменить (рис.7.2.(*а*)).

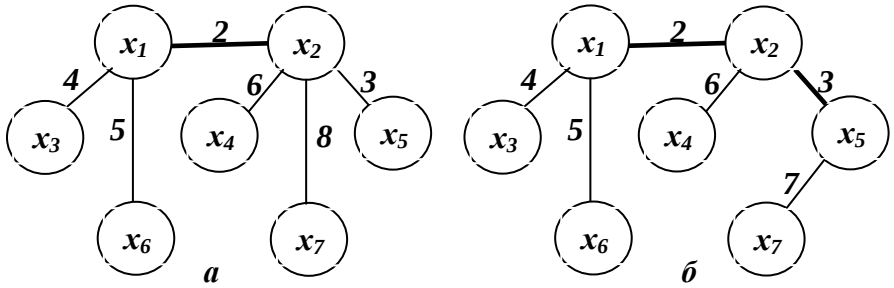


Рисунок 7.2. Добавление ребра (x_1 x_2) (*а*) и ребра (x_2 x_5) (*б*)

Наименьший вес из пяти ребер - кандидатов имеет ребро (x_2 x_5), поэтому к дереву нужно добавить его и вершину x_5 (рис. 7.2 (*б*)). Вес ребра (x_5 x_7) меньше веса ребра (x_2 x_7), поэтому оно замещает последнее. Из четырех ребер, инцидентных построенной части дерева, наименьший вес имеет ребро (x_1 x_3), поэтому следующим к дереву добавляется оно и вершина x_3 (рис. 7.3 (*а*)).

Затем к дереву добавляется ребро (x_1 x_6) вместе с вершиной x_6. Поскольку вес ребра (x_4 x_6) меньше веса ребра (x_2 x_4), а вес ребра (x_6 x_7) меньше веса ребра (x_5 x_7), меняем ребра – кандидаты (рис. 7.3 (*б*)).

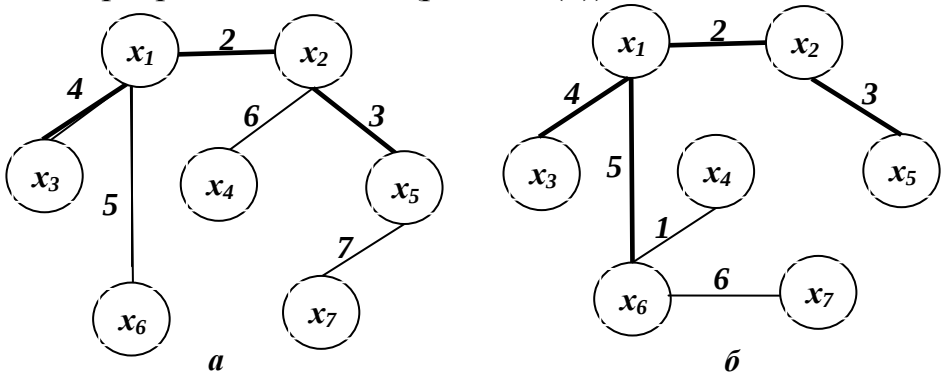


Рисунок 7.3. Добавление в дерево ребра (x_1 x_3) (*а*) и ребра (x_1 x_6) (*б*)

Ребро (x_4 x_6) имеет наименьший вес среди оставшихся, и оно добавляется следующим (рис. 7.4 (*а*)).

Теперь осталось добавить к дереву всего одно ребро (x_6 x_7) (рис. 7.4 (*б*)). После этого процесс завершается, построено МСД с суммарным весом ребер 21.

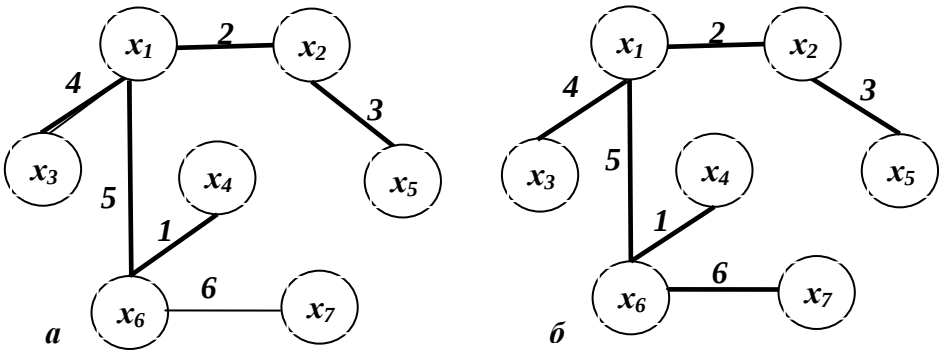


Рисунок 7.4. Добавление ребра $(x_4\, x_6)$ (*а*) и ребра $(x_6\, x_7)$ (*б*)

Сложность алгоритма Прима равна $O(m^3)$, где $m = |X|$.

7.1.2. Алгоритм Краскала

Алгоритм заключается в следующем.

Упорядочим все ребра исходного графа по неубыванию весов.

Просматривая последовательность слева направо, включаем в дерево каждое ребро, не образующее в дереве цикла. О том, что ребро $(x_i\, x_j)$ образует цикл можно судить по тому, что из вершины x_i есть путь в вершину x_j по другим ребрам дерева. Процесс заканчивается, когда в дерево включено $(m - 1)$ ребро.

Построим МСД для графа рис. 7.1 (*а*).

Упорядочим ребра:

$(x_4\, x_6)$, $(x_1\, x_2)$, $(x_2\, x_5)$, $(x_1\, x_3)$, $(x_1\, x_6)$, $(x_2\, x_4)$, $(x_3\, x_6)$, $(x_4\, x_7)$, $(x_6\, x_7)$, $(x_1\, x_4)$, $(x_5\, x_7)$, $(x_2 x_7)$.

Просматривая список, включаем в дерево ребра $(x_4\, x_6)$, $(x_1\, x_2)$, $(x_2\, x_5)$, $(x_1\, x_3)$, $(x_1\, x_6)$ (рис. 7.5 (*а*)). Ребра $(x_2\, x_4)$ (рис. 7.5 (*б*)) и $(x_3\, x_6)$ (рис. 7.5 (*в*)) образуют цикл с уже построенными ребрами, поэтому в дерево не включаются. Следующее включаемое в дерево ребро $(x_4\, x_7)$ (рис. 7.5 (*г*)). Дерево построено.

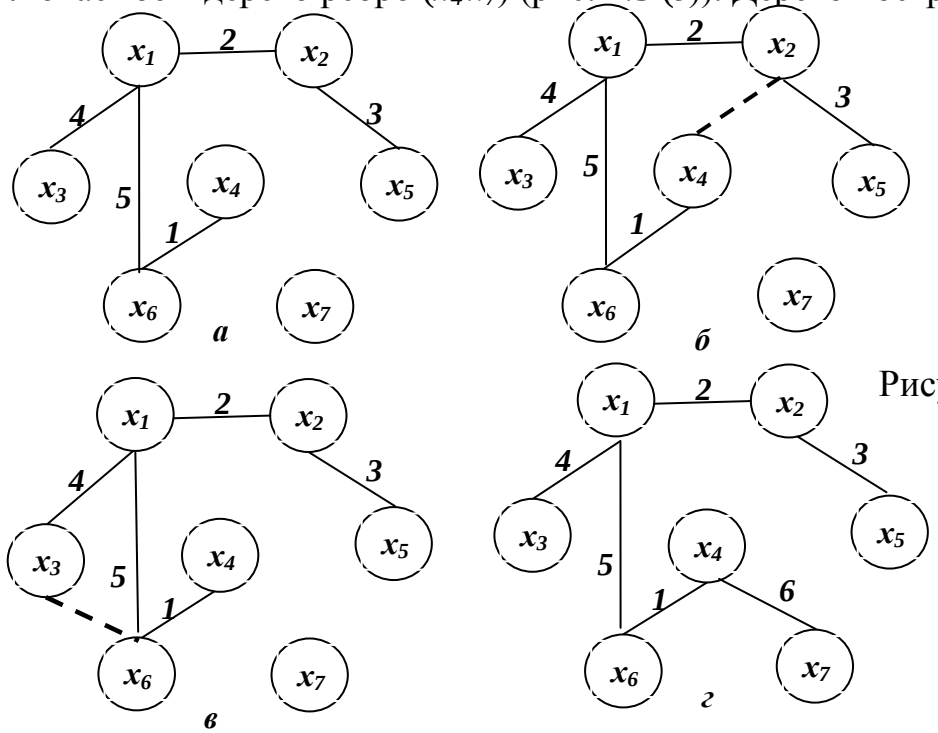


Рисунок 7.5. Построение МСД алгоритмом Краскала

Суммарный вес ребер МСД равен 21.

Сложность алгоритма Краскала равна $O(n \log n)$, где $n = |U|$.

7.1.3. Кратчайшие пути

Пусть дан граф $G(X, \Gamma)$, ребрам которого приписаны веса (стоимости), заданные матрицей $C=\|c_{ij}\|_{m\times m}$. *Задача о кратчайшем пути* состоит в нахождении пути с минимальным суммарным весом от начальной вершины $s\in X$ до конечной $t\in X$ или от начальной вершины $s\in X$ до всех остальных, при условии, что такие пути существуют.

Для нахождения кратчайших путей в графах разработано и реализовано на различных языках программирования много различных алгоритмов. К ним можно отнести алгоритмы: Дейкстры, Дейкстры-Грибова, Левита, Йена, Флойда-Уоршелла, Форда-Беллмана, Джонсона. Кратко рассмотрим эти алгоритмы. Алгоритм Дейкстры находит кратчайшее расстояние от одной из вершин графа до всех остальных. Алгоритм работает только для графов без ребер отрицательного веса. В процессе работы алгоритма поддерживаются три множества: M_0 – вершины, расстояние до которых уже вычислено (но, возможно, не окончательно); M_1 – вершины, расстояние до которых вычисляется; M_2 – вершины, расстояние до которых еще не вычислялось. В простейшем случае, сложность алгоритма составляет $O(m^2 + n)$. Здесь m – количество вершин графа, n – количество ребер графа. Эффективность метода Дейкстры существенно зависит от того, как организован поиск во множестве M_1 вершины с наименьшим текущим расстоянием.

Модификация Грибова заключается в том, что вершины разбиваются на подмножества с близкими, отличающимися меньше чем на минимальную длину ребра, значениями кратчайшего пути в текущую вершину на данном шаге. Затем на каждой итерации берется непустое множество, соответствующее минимальному значению расстояния.

Алгоритм Левита позволяет найти кратчайшие пути от заданной вершины до всех остальных вершин. Алгоритм Левита напоминает метод Дейкстры – в процессе работы поддерживаются три множества (M_0, M_1, M_2). Вершины, входящие во множество M_1, делятся на два упорядоченных множества – основную и приоритетную очередь. Каждой вершине сопоставляется неотрицательное значение длины кратчайшего из известных на данный момент путей в нее из начальной вершины. В сравнении с методом Дейкстры метод Левита проигрывает на том, что некоторые вершины приходится обрабатывать повторно, а выигрывает на более простых алгоритмах включения и исключения вершин из множества M_1. Эксперименты показывают, что для графов с «геометрическим» происхождением, т.е. для графов, построенных на основе транспортных сетей и реальных расстояний, метод Левита оказывается наиболее быстрым. Он выигрывает и по размеру программы. Метод Левита обладает еще и тем преимуществом перед методом Дейкстры, что он применим в случае отрицательных длин ребер. Сложность алгоритма составляет $O(n\cdot m)$.

Алгоритм Йена предназначен для нахождения нескольких путей минимальной суммарной длины во взвешенном графе с неотрицательными весами между двумя вершинами.

Алгоритм Беллмана-Форда решает задачу о нахождении кратчайших путей из одной вершины во все остальные для случая, когда веса ребер могут быть отрицательными. Кроме того алгоритм контролирует отсутствие циклов отрицательного веса, достижимых из исходной вершины. Если в графе нет циклов отрицательного веса, алгоритм находит кратчайшие пути из заданной вершины во все остальные, если же в графе есть циклы отрицательного веса, то, по крайней мере, для некоторых вершин кратчайшего пути не существует. Идея алгоритма, также как и алгоритма Дейкстры, основана на последовательной релаксации ребер, пока не будут найдены кратчайшие расстояния. Сложность алгоритма Форда - Беллмана есть $O(n \cdot m)$.

Алгоритм Флойда-Уоршелла это динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа. Сложность алгоритма $O(m^3)$, т.е. алгоритм имеет кубическую сложность. Тем не менее, алгоритм определяет только кратчайшие расстояния между всеми парами вершин, но не сохраняет информации о кратчайших путях.

Алгоритм Джонсона позволяет найти кратчайшие пути между любыми двумя вершинами графа за $O(m^2 \log(m) + n \cdot m)$. Следовательно, для достаточно разреженных графов, он эффективнее альтернативных алгоритмов возведения в квадрат матрицы весов и алгоритма Флойда-Уоршелла. Этот алгоритм основан на идее изменения весов ребер графа. Если для ребер найти такую функцию, которая изменяет их веса, обеспечивая их неотрицательность и оставляя кратчайшие пути такими же (то есть состоящими из тех же ребер), то задачу поиска кратчайших путей между всеми парами вершин можно решить применив алгоритм Дейкстры для каждой вершины.

Рассмотрим подробно алгоритм Дейкстры. Он основан на приписывании вершинам временных пометок, дающих верхнюю границу длины пути от s к этой вершине. Эти пометки постепенно уточняются, и на каждом шаге итерации точно одна из временных пометок становится постоянной. Это указывает на то, что пометка уже не является верхней границей, а дает точную длину кратчайшего пути от s к рассматриваемой вершине.

Пусть $l(x_i)$ пометка вершины x_i, а $l(x_i)^+$ - постоянная пометка вершины.

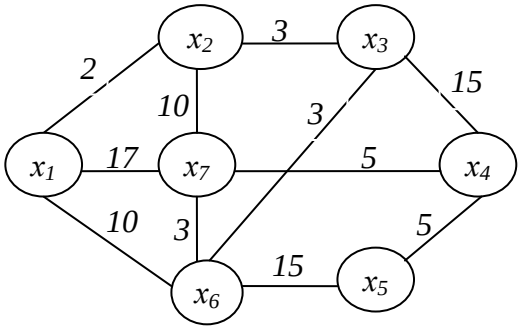
1. Положить $l(s)=0^+$ и считать эту пометку постоянной. Положить $l(x_i)=\infty$ для всех $x_i \neq s$ и считать их временными. Положить $p=s$.
2. Для всех $x_i \in Гp$, пометки которых временные, изменить пометки в соответствии со следующим выражением
$$l(x_i) = min[l(x_i), l(p) + c(p,x_i)].$$
3. Среди всех вершин с временными пометками найти такую, для которой
$$l(x_i^*) = min[l(x_i)].$$
4. Считать пометку вершины x_i^* постоянной $l(x_i^*)^+$ и положить $p= x_i^*$.
5. (Если надо найти лишь путь от s до t).
Если $p=t$, то $l(p)$ – длина кратчайшего пути, конец. Если $p \neq t$, перейти к п.2.
6. (Если надо найти путь от s до всех остальных вершин).
Если все вершины имеют постоянные пометки, то конец, если есть временные пометки, то перейти к п.2.

Сами пути можно получить при помощи рекурсивной процедуры с использованием соотношения:

$$l(x_i') + c(x_i', x_i) = l(x_i),$$

где x_i' – вершина непосредственно предшествующая вершине x_i в кратчайшем пути от s к x_i.

Пример. Дан взвешенный граф $G(X,\Gamma)$ и матрица весов $C=||c_{ij}||_{7\times7}$ (рис. 7.6). Необходимо найти кратчайшие пути от начальной вершины x_1 ко всем остальным вершинам.



$C=$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	0	2				10	17
x_2	2	0	3				10
x_3		3	0	15		3	
x_4			15	0	5		5
x_5				5	0	15	
x_6	10		3		15	0	3
x_7	17	10		5		3	0

Рисунок 7.6. Исходный граф и матрица весов

1. $l(x_1)=0^+$; $l(x_i)=\infty$, для всех $i \neq 1$, $p = x_1$.
   Результаты итерации запишем в таблицу.

$L=$

	1
x_1	0^+
x_2	∞
x_3	∞
x_4	∞
x_5	∞
x_6	∞
x_7	∞

2. $\Gamma p = \{x_2, x_6, x_7\}$ – все пометки временные, уточним их:
   $l(x_2)=min[\infty, 0^+ +2]=2;$
   $l(x_6)=min[\infty, 0^+ +10]=10;$
   $l(x_7)=min[\infty, 0^+ +17]=17.$
3. $l(x_i^*) = min[l(x_i)] = l(x_2) = 2.$
4. x_2 получает постоянную пометку $l(x_2) = 2^+$, $p=x_2$.
5. Не все вершины имеют постоянные пометки, поэтому
   $\Gamma p = \{x_1, x_3, x_7\}$ – временные пометки имеют вершины x_3, x_7, уточняем их:
   $l(x_3)=min[\infty, 2^+ +3]=5;$
   $l(x_7)=min[17, 2^+ +10]=12.$

$L=$

	1	2
x_1	0^+	
x_2	∞	2^+
x_3	∞	∞
x_4	∞	∞
x_5	∞	∞
x_6	∞	10
x_7	∞	17

6. $l(x_i^*) = min[l(x_i)] = l(x_3) = 5.$
7. $l(x_3) = 5^+$, $p=x_3$.
8. Не все вершины имеют постоянные пометки,
   $\Gamma p = \{x_2, x_4, x_6\}$ – временные пометки имеют вершины x_4, x_6, уточняем их:
   $l(x_4)=min[\infty, 5^+ +15]=20;$
   $l(x_6)=min[10, 5^+ +3]=8.$

$L=$

	1	2	3
x_1	0^+		
x_2	∞	2^+	
x_3	∞	∞	5^+
x_4	∞	∞	∞
x_5	∞	∞	∞
x_6	∞	10	10
x_7	∞	17	12

9. $l(x_i^*) = min[l(x_i)] = l(x_6) = 8$.
10. $l(x_6) = 8^+$, $p=x_6$.
11. Не все вершины имеют постоянные пометки
    $Гр = \{x_1, x_5, x_7\}$ – временные пометки имеют вершины x_5, x_7, уточняем их:
    $l(x_5)=min[\infty\ ,\ 8^+ +15]=23;$
    $l(x_7)=min[12,\ 8^+ +3]=11.$
12. $l(x_i^*) = min[l(x_i)] = l(x_7) = 11$.
13. $l(x_7) = 11^+$, $p=x_7$.
14. Не все пометки постоянные
    $Гр = \{x_1, x_2, x_4, x_6\}$ – временную пометку имеет вершина x_4, уточняем ее:
    $l(x_4)=min[20,\ 11^+ +5]=16.$
15. $l(x_i^*) = min[l(x_i)] = l(x_4) = 16$.
16. $l(x_4) = 16^+$, $p=x_4$.
17. Не все пометки постоянные
    $Гр = \{x_3, x_5, x_7\}$ – временную пометку имеет вершина x_5, уточняем ее:
    $l(x_5)=min[23,\ 16^+ +5]=21.$
18. $l(x_i^*) = l(x_5) = 21$.
19. $l(x_5) = 21^+$, $p=x_5$.
20. Все пометки постоянные.

$L=$

	1	2	3	4
x_1	0^+			
x_2	∞	2^+		
x_3	∞	∞	5^+	
x_4	∞	∞	∞	20
x_5	∞	∞	∞	∞
x_6	∞	10	10	8^+
x_7	∞	17	12	12

$L=$

	1	2	3	4	5
x_1	0^+				
x_2	∞	2^+			
x_3	∞	∞	5^+		
x_4	∞	∞	∞	20	20
x_5	∞	∞	∞	∞	23
x_6	∞	10	10	8^+	
x_7	∞	17	12	12	11^+

$L=$

	1	2	3	4	5	6
x_1	0^+					
x_2	∞	2^+				
x_3	∞	∞	5^+			
x_4	∞	∞	∞	20	20	16^+
x_5	∞	∞	∞	∞	23	23
x_6	∞	10	10	8^+		
x_7	∞	17	12	12	11^+	

Кратчайшие расстояния от вершины x_1 до всех вершин найдены. Как найти кратчайший путь до конкретной вершины, покажем на примере вершины x_5.

$l(x_5) = 21,\ Гx_5 = \{x_4, x_6\}$,
$21 = l(x_4) + c(x_4, x_5) = 16+5$,
$21 \neq l(x_6) + c(x_6, x_5) = 8+15$.

$L=$

	1	2	3	4	5	6	7
x_1	0^+						
x_2	∞	2^+					
x_3	∞	∞	5^+				
x_4	∞	∞	∞	20	20	16^+	
x_5	∞	∞	∞	∞	23	23	21^+
x_6	∞	10	10	8^+			
x_7	∞	17	12	12	11^+		

Это означает, что в вершину x_5 мы попали из вершины x_4.
Далее, $l(x_4) = 16,\ Гx_4 = \{x_3, x_5, x_7\}$,
$16 \neq l(x_3) + c(x_3, x_4) = 5+15$,
$16 \neq l(x_5) + c(x_5, x_4) = 21+15$,
$16 = l(x_7) + c(x_7, x_4) = 11+5$.
Это означает, что в вершину x_4 мы попали из вершины x_7.
Далее, $l(x_7) = 11,\ Гx_7 = \{x_1, x_4, x_6\}$,
$11 \neq l(x_1) + c(x_1, x_7) = 0+17$,
$11 \neq l(x_4) + c(x_4, x_7) = 16+5$,
$11 = l(x_6) + c(x_6, x_7) = 8+3$.
Это означает, что в вершину x_7 мы попали из вершины x_6.
Далее, $l(x_6) = 8,\ Гx_6 = \{x_1, x_3, x_5, x_7\}$,

$8 \neq l(x_1) + c(x_1, x_6) = 0+10$,
$8 = l(x_3) + c(x_3, x_6) = 5+3$,
$8 \neq l(x_5) + c(x_5, x_6) = 21+15$,
$8 \neq l(x_7) + c(x_7, x_6) = 11+3$.
Это означает, что в вершину x_6 мы попали из вершины x_3.
Далее, $l(x_3) = 5$, $Гx_3 = \{x_2, x_4, x_6\}$,
$5 = l(x_2) + c(x_2, x_3) = 2+3$,
$5 \neq l(x_4) + c(x_4, x_3) = 16+15$,
$5 \neq l(x_6) + c(x_6, x_3) = 8+3$.
Это означает, что в вершину x_3 мы попали из вершины x_2.
Далее, $l(x_2) = 2$, $Гx_2 = \{x_1, x_3, x_7\}$,
$2 = l(x_1) + c(x_1, x_2) = 0+2$,
$2 \neq l(x_3) + c(x_3, x_2) = 5+3$,
$2 \neq l(x_7) + c(x_7, x_2) = 11+10$.
Это означает, что в вершину x_2 мы попали из вершины x_1.
Кратчайший путь от вершины x_1 до вершины x_5 найден (рис.7.7):

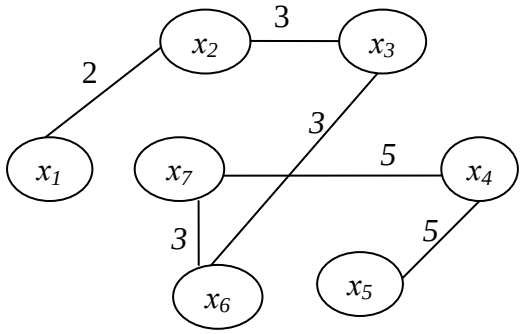


Рисунок 7.7. Кратчайшие пути от вершины x_1 до всех вершин

7.1.4. Задачи, близкие к задаче о кратчайшем пути

1. *Наиболее надежный путь.*

В этом случае вес ребра представляет его надежность. Надежность пути от s к t, составленного из ребер, взятых из множества P, задается формулой

$$\rho(P) = \prod_{(x_i, x_j) \in P} \rho_{ij},$$ где ρ_{ij} – надежность ребра (x_i, x_j).

2. *Самый длинный (критический) путь*.

Задача сетевого планирования, заключающаяся в нахождении самого длинного по временной протяженности пути в сетевом графике, определяющего продолжительность работ по выполнению проекта.

3. *Путь с наибольшей пропускной способностью*.

В этом случае каждое ребро графа имеет пропускную способность q_{ij} и требуется найти путь от s к t с наибольшей пропускной способностью. Пропускная способность пути P определяется ребром из P с наименьшей пропускной способностью, т.е.

$$Q(P) = \min_{(x_i, x_j) \in P} [q_{ij}].$$

Определение. Если множество вершин графа *G(X,U)* разбить на два подмножества X_1 и X_2 (где $X=X_1 \cup X_2$), то множество ребер графа, одни концевые вершины которых лежат в X_1, а другие в X_2, называется *разрезом графа G*.

Теорема Форда – Фалкерсона. Пропускная способность пути с наибольшей пропускной способностью от *s* к *t* равна

$$Q(P) = \min_{K} \{ \max_{(x_i,x_j)\in K} [q_{ij}] \},$$

где *K* – любой (*s*-*t*) разрез.

Для нахождения пути с наибольшей пропускной способностью в неографе предложен следующий алгоритм.

7.1.5. Алгоритм Франка – Фриша

1. Взять (*s*-*t*) разрез $K_1 = (\{s\}, X\backslash\{s\})$ и найти
$$Q_1 = \max_{(x_i,x_j)\in K_1} [q_{ij}].$$
2. Закоротить все ребра графа (x_i, x_j) с $q_{ij} \geq Q_1$, т.е. заменить вершины x_i и x_j на вершину *x*, удалив ребро (x_i, x_j), положить $\Gamma x = \Gamma x_i \cup \Gamma x_j$.
3. Для полученного графа G_1 выбрать другой (*s*-*t*) разрез K_2 и найти
$$Q_2 = \max_{(x_i,x_j)\in K_2} [q_{ij}].$$
4. Закоротить все ребра графа (x_i, x_j) с $q_{ij} \geq Q_2$. Получить граф G_2 … и т.д., пока не будут объединены вершины *s*-*t*.
5. Теперь каждый (*s*-*t*) путь в графе G', образованный вершинами из *G* и теми ребрами, которые оказались закороченными, будет иметь максимальную пропускную способность.

Пример. Найти (*s*-*t*) путь с наибольшей пропускной способностью в графе *G(X,U)* (рис. 7.8).

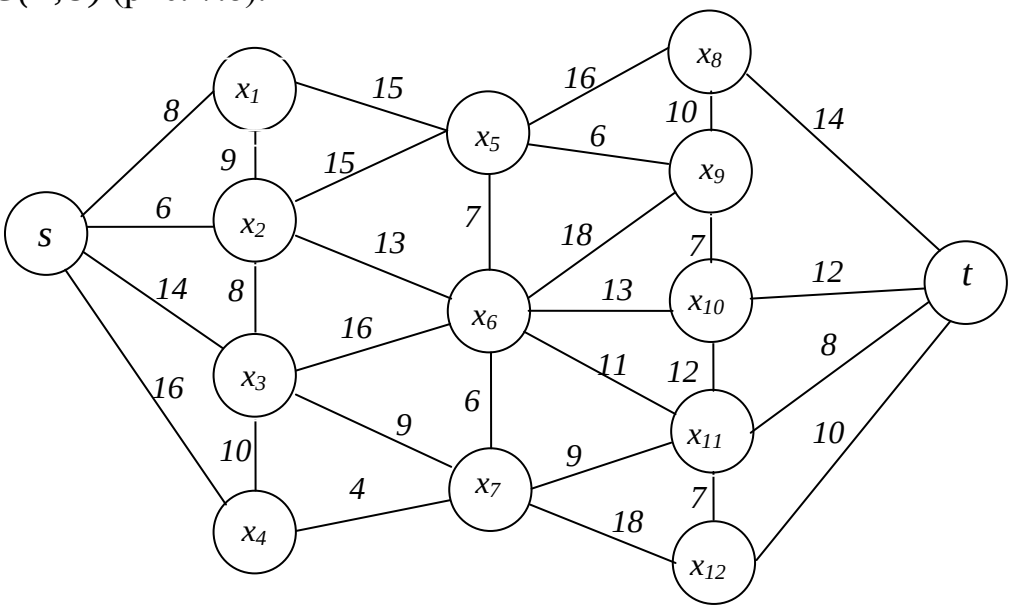


Рисунок 7.8. Исходный граф

1. Проводим разрез $K_1 = (\{s\}, X\backslash\{s\})$ (рис. 7.9).

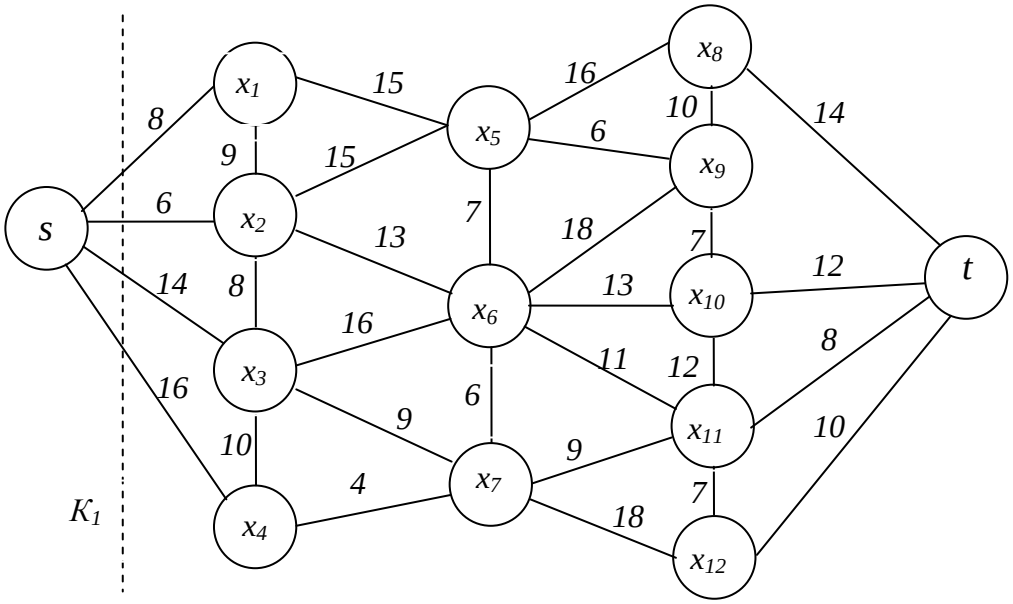


Рисунок 7.9. Разрез K_1

2. Находим $Q_1 = \max\limits_{(x_i, x_j) \in K_1} [q_{ij}] = 16.$

3. Закорачиваем все ребра графа (x_i, x_j) с $q_{ij} \geq Q_1$.

4. Это ребра (s, x_4), (x_3, x_6), (x_5, x_8), (x_6, x_9) и (x_7, x_{12}). Получаем граф G_1 (рис. 7.10).

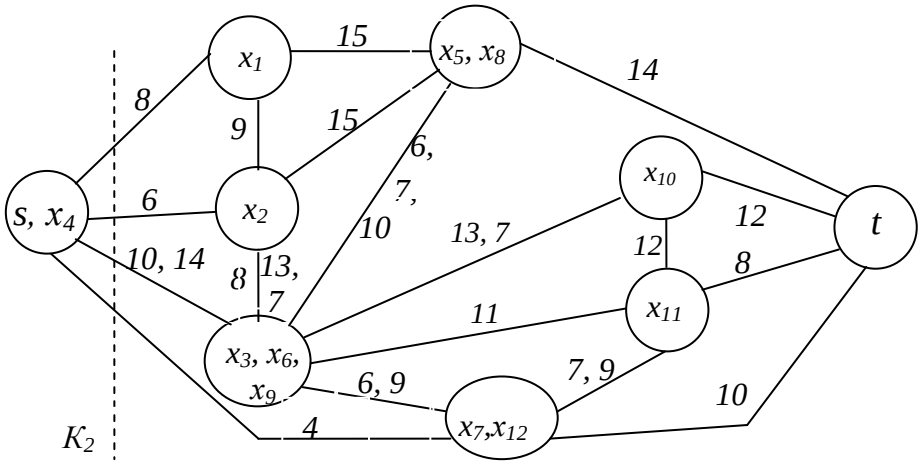


Рисунок 7.10. Разрез K_2

5. Проводим разрез K_2, находим $Q_2 = \max\limits_{(x_i, x_j) \in K_1} [q_{ij}] = 14.$

6. Закорачиваем все ребра графа (x_i, x_j) с $q_{ij} \geq Q_2$. Это ребра (s, x_4, x_4, x_6, x_9), (x_3, x_6), (x_1, x_2, x_5, x_8, t). Получаем граф G_2 (рис. 7.11).

7. Проводим разрез K_3, находим $Q_3 = \max\limits_{(x_i, x_j) \in K_1} [q_{ij}] = 13.$

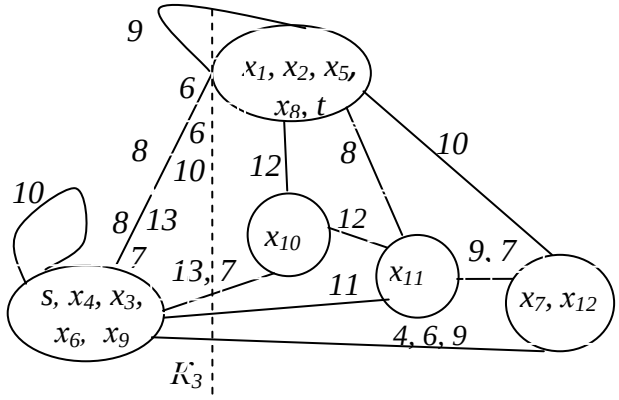


Рисунок 7.11. Разрез K_3

8. Закорачиваем все ребра графа (x_i, x_j) с $q_{ij} \geq Q_3$. Получаем граф G_3 (рис. 7.12).

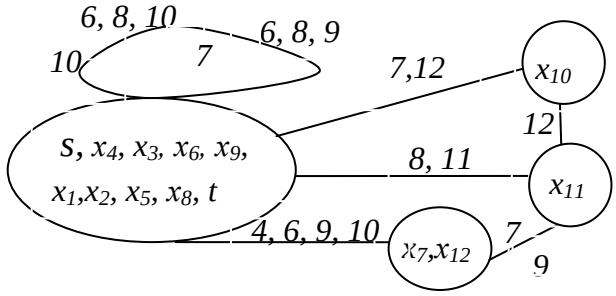


Рисунок 7.12. Окончательный граф

9. Вершины *s*-*t* объединены. Пропускная способность искомого пути $Q(P)=13$.
10. Строим граф, вершины которого – вершины исходного графа G, а ребра – ребра с пропускной способностью $q_{ij} \geq Q(P)=13$.

На рисунке 7.13 показан путь с наибольшей пропускной способностью.

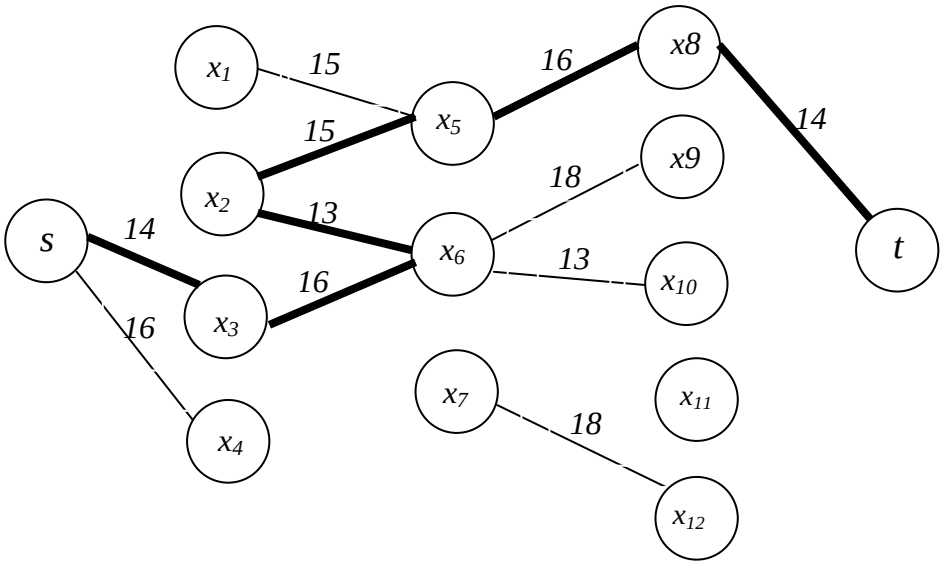


Рисунок 7.13. Путь с наибольшей пропускной способностью.

7.1.6. Задача Штейнера

Виды используемых деревьев определяются технологией выполнения соединений и схемотехническими требованиями. При автоматизированном конструировании схем проводного и печатного монтажа ставятся разные задачи построения минимальных деревьев соединений. В первом случае не допускается вводить дополнительные вершины (соединение «контакт» - «контакт»). Для решения такой задачи используют рассмотренные ранее эффективные алгоритмы Краскала, Прима и другие. Во втором случае разрешается вводить дополнительные вершины и соединяющие их ребра (соединения «контакт» - «проводник», «проводник» - «проводник»). Такая задача называется задачей Штейнера, деревья соединений – деревьями Штейнера (ДШ), а дополнительные вершины – точками Штейнера (ТШ).

Задача построения дерева Штейнера имеет следующий вид. Требуется найти такой связный ациклический граф с использованием дополнительных вершин Штейнера (дерево Штейнера) $T' = (X',U')$, что X' включает все вершины X и дополнительные вершины (вершины Штейнера) и суммарный вес ребер в T' будет минимальным. На рис. 7.14 (*а*) показано МСД, суммарная длина ребер - 10,123, а на рис. 7.14 (*б*) – дерево Штейнера, суммарная длина ребер – 9,196.

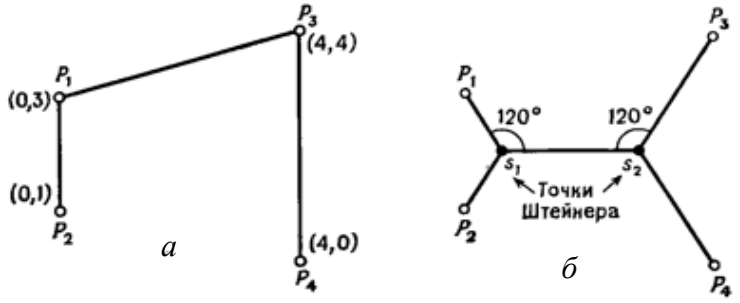


Рисунок 7.14. МСД (*а*) и дерево Штейнера (*б*)

Задача о соединении точек оптимальным образом – одна из старейших оптимизационных задач. Она восходит еще к П. Ферма. Еще в XVII столетии была предложена следующая задача. На плоскости найти такую точку P, которая минимизирует суммарное расстояние от P до трех заданных точек плоскости. Ферма, Торричелли и Кавальери независимо друг от друга нашли решение этой задачи.

Общим решением этой задачи является следующее. Точка P либо лежит внутри треугольника, построенного на заданных точках, либо совпадает с одной из этих точек. В первом случае каждый из углов между отрезками, соединяющими P с каждой их точек, составляет 120^0. Во втором случае угол, образованный отрезками, соединяющими эту точку с другими заданными точками, равен или больше 120^0. Рис. 7.15. демонстрирует способ решения задачи, найденный в середине XVII века.

Для отыскания точки P, ближайшей (в смысле суммарного расстояния) к заданным точкам A, B, C, сначала строится треугольник (ABC) и на одной из его сторон, например на AC, строится равносторонний треугольник. (ABX) так, что точка B лежит вне этого треугольника. Точка пересечения окружности, описан-

ной вокруг равностороннего треугольника (*ABX*), с отрезком (*BX*) есть искомая точка *P*.

Точкой Торричелли треугольника (*ABC*) называется такая точка *P*, из которой стороны данного треугольника видны под углом 120^0, т.е. углы $\angle APB$, $\angle APC$ и $\angle BPC$ равны 120^0.

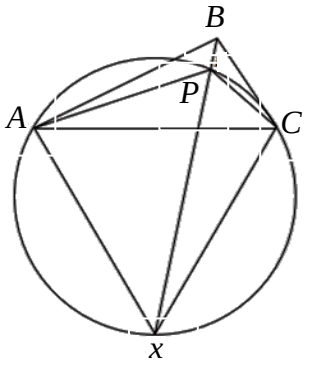


Рисунок 7.15. Нахождение точки Торричелли

Трехточечная задача используется как составная часть алгоритмов решения задачи Штейнера.

Перебираем тройки вершин графа и находим точки Торричелли. Выбираем такие точки, включение которых в дерево даст максимальный эффект.

Задача Штейнера относится к классу так называемых NP-полных задач, поэтому алгоритмы, дающие точные решения, как правило, не могут быть использованы в САПР из-за неприемлемой временной сложности. Это обстоятельство послужило стимулом для разработки многочисленных эвристических алгоритмов. Наибольший практический интерес представляет алгоритм последовательного введения дополнительных вершин в дерево Прима-Краскала.

Возможно, наиболее важным практическим применением задачи Штейнера является конструирование интегральных электронных схем. Более короткая сеть проводящих линий на интегральной схеме требует меньшего времени зарядки-разрядки по сравнению с более длинной сетью и повышает, таким образом, быстродействие схемы. Однако задача отыскания кратчайшей сети на интегральной схеме имеет другую геометрию, так как проводники на ней обычно проходят лишь в двух направлениях — горизонтальном и вертикальном. Такая задача получила название прямоугольной задачи Штейнера.

Для случая ортогональной метрики доказано, что если через каждую точку из исходного множества точек провести горизонтальные и вертикальные линии, то решение задачи Штейнера можно получить, рассматривая в качестве возможных дополнительных вершин только точки пересечения полученной сетки линий (рис.7.16 (*а*)).

Алгоритм последовательно вводит в текущее дерево Прима-Краскала каждую из дополнительных вершин решеточного графа, строит новое дерево Прима и запоминает полученный выигрыш в длине. После оценки всех дополнительных точек в текущее дерево включается точка с максимальным выигрышем (рис.7.16 (*б*)). При этом, чтобы избежать в процессе преобразования появления

избыточных точек, необходимо учитывать, что дополнительные точки в дереве Штейнера могут иметь только степень 3 и 4.

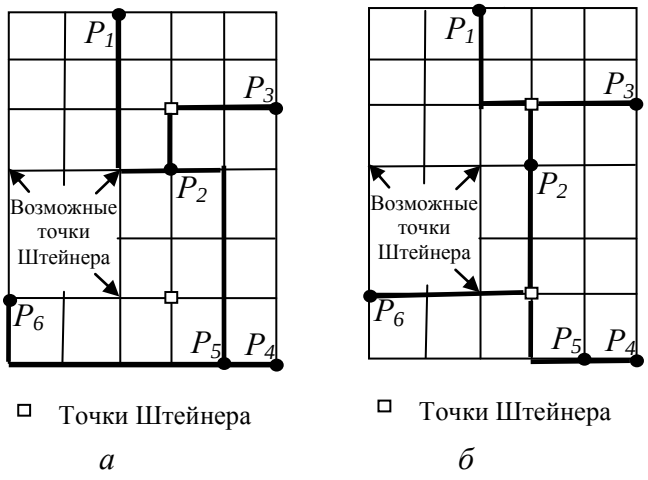


Рисунок 7.16. Кратчайшие остовное дерево (*а*) (L = 18)
и дерево Штейнера (*б*) (L = 15)

Рассмотрим некоторые эвристические методы построения деревьев Штейнера.

Метод Ханана

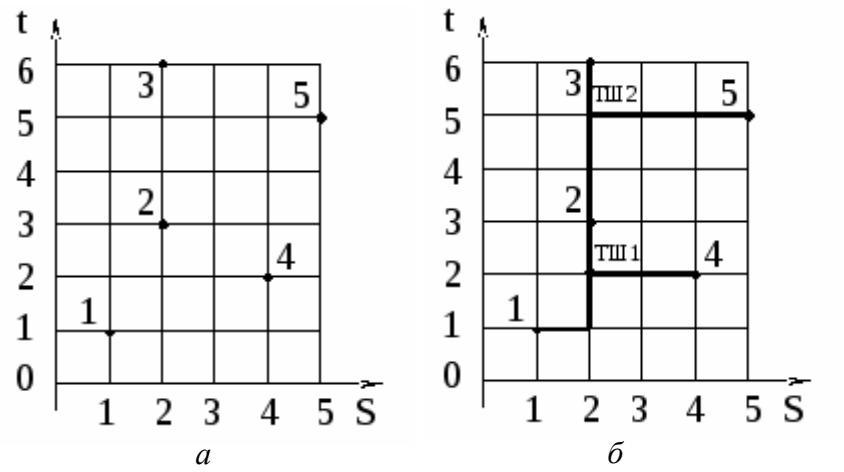
1. Все множество вершин с координатами s разбивается на классы $S_1, S_2, ..., S_m$ в порядке неубывания координаты s_i так, чтобы у всех вершин одного класса была одинаковая координата s.
2. Анализируется множество вершин класса S_1 и соединяется между собой прямой.
3. Образуется множество I, состоящее из вершин, включенных в дерево и точек, принадлежащих построенной прямой.
4. Выбирается следующее множество S_{i+1}, имеющее наименьшую координату s. Для вершины $x_k \in S_{i+1}$ определяется точка x_j, для которой $d_{kj} = \min\{d_{kg}\}$ $x_g \in I$. Вершина x_k соединяется двухзвенной ломаной линией с точкой $x_j \in I$.
5. Для очередной вершины класса S_{i+1} операция подсоединения к дереву выполняется в соответствии с пунктами 3, 4.
6. Пункты 3, 4, 5 повторяются для всех множеств S_i до построения полного дерева.

Рассмотрим пример построения дерева Штейнера по методу Ханана для вершин, представленных на рис. 7.17 (*а*).

1. Разобьем множество вершин $X = \{x_1, x_2, ..., x_5\}$ на классы по неубыванию координаты s_i, получим: $S_1=\{x_1\}$, $S_2=\{x_2, x_3\}$, $S_3=\{x_4\}$, $S_4=\{x_5\}$.
2. В классе S_1 всего одна вершина, поэтому соединение вершин не производим.
3. Вершину $x_2 \in S_2$ соединяем с x_1 двухзвенной ломаной линией, причем сначала в направлении перпендикулярном оси s, а затем оси t.

4. Вершину $x_3 \in S_2$ соединяем с ближайшей точкой уже построенного фрагмента.
5. Переходим в следующий класс и $x_4 \in S_3$ соединяем с ближайшей точкой дерева, получаем ТШ 1.
6. Вершину $x_5 \in S_4$ соединяем по кратчайшему расстоянию с ближайшей точкой построенного дерева, получаем ТШ 2.

Результат построения дерева Штейнера с суммарной длиной ребер L=11 представлен на рис. 7.17 (*б*).



а *б*

Рисунок 7.17. Вершины графа (*а*); дерево Штейнера (метод Ханана) (*б*)

Столб Штейнера

Метод "Столб Штейнера" является одним из наиболее эффективных по времени реализации эвристических алгоритмов построения дерева Штейнера и предусматривает следующий порядок действия.

1. Все вершины проецируются на ось *s*.
2. Расстояние от наименьшей до наибольшей координаты *s* делится пополам, и из этой точки проводится перпендикуляр (столб Штейнера).
3. Из каждой вершины опускается перпендикуляр до пересечения со столбом Штейнера.

Рассмотрим пример построения дерева Штейнера по данному методу для представленного на рис. 7.17 (*а*) множества вершин.

1. Находим среди множества проекций на ось *s* (s_1=1, s_2= s_3=2, s_4=4, s_5=5) минимальное и максимальное значения: $\min s_i = s_1 = 1$, $\max s_i = s_5 = 5$.
2. Вычисляем координату проведения столба Штейнера: $s_i = (s_1 + s_5) / 2 = 3$.
3. Проводим столб Штейнера с координатой $s_i = 3$ и опускаем из всех вершин перпендикуляры до пересечения со столбом.

Результат построения дерева Штейнера с суммарной длиной ребер L=12 представлен на рис. 7.18 (*а*).

Аналогично строится горизонтальный столб (рис. 7.18 (*б*)), L=12.

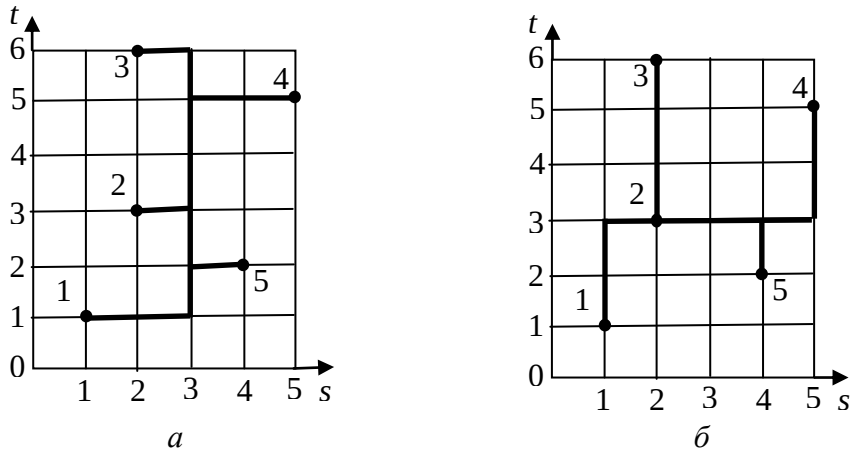


Рисунок 7.18. Деревья Штейнера, построенные методом "Столб Штейнера", (*а*) – вертикальный; (*б*) – горизонтальный столбы

Временная сложность алгоритма $O(n^2)$.

В. На втором этапе предпринимается попытка точно решить, на каком слое каждое соединение может располагаться.

Расслоение можно проводить до, после или во время трассировки отдельных соединений. Расслоение основано на анализе схемы соединений для выявления конфликтующих проводников.

Если расслоение проводится до собственно трассировки, то для контактов каждой цепи строится минимальный охватывающий прямоугольник. Строится граф перекрытий прямоугольников, вершины которого цепи, а ребра между вершинами проводятся, если прямоугольники перекрываются. Граф перекрытий раскрашивается в минимальное число цветов. В результате: цвет – слой.

Если расслоение проводится после трассировки, то получают совмещенную топологию. Строят граф пересечений, в котором вершины – проводники, а ребра их соединяют, если проводники пересекаются. Граф пересечений раскрашивается в минимальное число цветов.

Расслоение в процессе трассировки будет рассмотрено в подразделе 7.4.7.

7.2. Алгоритмы раскраски графа

Задача раскраски вершин графа относится к NP-полным задачам. Различают точные и приближенные алгоритмы раскраски.

Примером точных алгоритмов служит алгоритм Вейсмана. Иногда его называют алгоритмом, использующим метод Магу.

7.2.1. Алгоритм Вейсмана

Алгоритм состоит из двух частей:

1. Построение семейства максимальных внутренне устойчивых множеств (МВУМ) (метод Магу);
2. Выбор минимального числа МВУМ, покрывающих все вершины графа (метод Петрика). Метод Петрика использовался в курсовой работе по дисциплине «Дискретная математика» для нахождения минимальных нормальных форм.

Идея метода Магу заключается в последовательном удалении из графа звездного подграфа с вершиной x_i.

Используется преобразование булевых выражений

$$(x_i \vee x_1)\,(x_i \vee x_2)...(x_i \vee x_j) = x_i \vee x_1\,x_2...x_j.$$

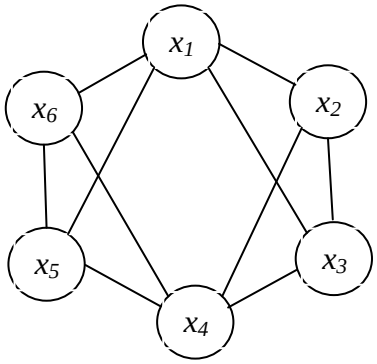
Алгоритм состоит из следующих операций:

1. В матрице соединений R для каждой вершины подсчитывается число ненулевых элементов r_i;
2. Находится вершина x_i с $\max r_i$, если таких вершин несколько, то выбирается любая;
3. Для выбранной вершины x_i записывается выражение $C_i = (x_i \vee x_a\,x_b...x_q)$, где $\Gamma x_i = \{x_a, x_b, ..., x_q\}$;
4. Из матрицы R удаляются строка и столбец, соответствующие вершине x_i;
5. Если $R \neq \varnothing$, то переход к п. 2, иначе к п. 6;
6. Составляется конъюнкция $П = \wedge C_i$. Раскрываются скобки. В полученной дизъюнкции на основе законов булевой алгебры выполняется минимизация. Т. к. в выражении нет отрицаний, используются только два закона: тавтологии $aa=a$ и поглощения $a \vee ab = a$;
7. Результат минимизации записывается в виде $П = \vee K_j$;
8. Для каждого K_j ищутся вершины графа, не вошедшие него. Получено φ_j и семейство МВУМ $\Psi = \{\varphi_1, \varphi_2, ..., \varphi_l\}$;
9. Для каждой вершины $x_i \in X$ определяются подмножества φ_j, в которые входит вершина $x_i \in \varphi_j$. Составляется дизъюнкция $t_i = \vee \varphi_j$;
10. Составляется конъюнкция $П' = \wedge t_i$. Раскрываются скобки. В полученной дизъюнкции на основе законов булевой алгебры выполняется минимизация;
11. Получена дизъюнкция конъюнктивных термов $П' = \vee(\wedge \varphi_j)$. Выбирается конъюнктивный терм $\wedge \varphi_j$ с минимальным числом сомножителей.

Количество сомножителей в этом терме и есть хроматическое число графа. Число минимальных термов – число вариантов раскраски графа. А каждое φ_j – множество вершин, которые можно окрасить в один цвет.

Заметим, что п.п. 1-8 составляют метод Магу, а п.п. 9-11 – метод Петрика.

Пример**.** Раскрасить вершины графа рис.7.19.



$R =$

	x_1	x_2	x_3	x_4	x_5	x_6	r_i
x_1	0	1	1	0	1	1	4
x_2		0	1	1	0	0	3
x_3			0	1	0	0	3
x_4				0	1	1	4
x_5					0	1	3
x_6						0	3

Рисунок 7.19. Исходный граф и его матрица соединений

1. В матрице R подсчитываем число ненулевых элементов r_i;
2. $\max r_i = r_1 = r_4 = 4$, выбираем x_1;

3. $\Gamma x_1 = \{x_2, x_3, x_5, x_6\}$, записываем выражение

$C_1 = (x_1 \vee x_2\, x_3\, x_5\, x_6)$;

4. Из матрицы R удаляем строку и столбец, соответствующие вершине x_1;

$R =$

	x_2	x_3	x_4	x_5	x_6	r_i
x_2	0	1	1	0	0	2
x_3		0	1	0	0	2
x_4			0	1	1	4
x_5				0	1	2
x_6					0	2

5. $R \neq \varnothing$, $\max r_i = r_4 = 4$;

$\Gamma x_4 = \{x_2, x_3, x_5, x_6\}$, $C_4 = (x_4 \vee x_2\, x_3\, x_5\, x_6)$;

6. Из матрицы R удаляем строку и столбец, соответствующие вершине x_4;

$R =$

	x_2	x_3	x_5	x_6	r_i
x_2	0	1	0	0	1
x_3		0	0	0	1
x_5			0	1	1
x_6				0	1

7. $R \neq \varnothing$, $\max r_i = r_2 = r_3 = r_5 = r_6 = 1$, выбираем x_2;

$\Gamma x_2 = \{x_3\}$, $C_2 = (x_2 \vee x_3)$;

8. Из матрицы R удаляем строку и столбец, соответствующие вершине x_2;

$R =$

	x_3	x_5	x_6	r_i
x_3	0	0	0	0
x_5		0	1	1
x_6			0	1

9. $R \neq \varnothing$, $\max r_i = r_5 = r_6 = 1$, выбираем x_5;

$\Gamma x_5 = \{x_9\}$, $C_5 = (x_5 \vee x_6)$;

10. Из матрицы R удаляем строку и столбец, соответствующие вершине x_5;

$R =$

	x_3	x_6	r_i
x_3	0	0	0
x_6		0	0

11. $R = \varnothing$;

12. Составляем конъюнкцию C_i и выполняем минимизацию

$\Pi = \wedge C_i = C_1\, C_2\, C_4\, C_5 = (x_1 \vee x_2\, x_3\, x_5\, x_6)(x_2 \vee x_3)(x_4 \vee x_2\, x_3\, x_5\, x_6)(x_5 \vee x_6) =$

$= x_1\, x_2\, x_4\, x_5 \vee x_1\, x_2\, x_4\, x_6 \vee x_1\, x_3\, x_4\, x_5 \vee x_1\, x_3\, x_4\, x_6 \vee x_2\, x_3\, x_5\, x_6 = \vee K_j =$

$= K_1 \vee K_2 \vee K_3 \vee K_4 \vee K_5$;

13. Для каждого K_j ищем φ_j:

$\varphi_1 = \{x_3, x_6\}$, $\varphi_2 = \{x_3, x_5\}$, $\varphi_3 = \{x_2, x_6\}$, $\varphi_4 = \{x_2, x_5\}$, $\varphi_5 = \{x_1, x_4\}$. Получено семейство МВУМ Ψ;

14. Для каждой вершины определим подмножества φ_j, в которые она входит. Строим дизъюнкцию $t_i = \vee \varphi_j$;

$t_1 = \varphi_5$; $t_2 = \varphi_3 \vee \varphi_4$; $t_3 = \varphi_1 \vee \varphi_2$; $t_4 = \varphi_5$; $t_5 = \varphi_2 \vee \varphi_4$; $t_6 = \varphi_1 \vee \varphi_3$;

15. Составляем конъюнкцию и выполняем минимизацию булевой функции

$П' = \wedge t_i = t_1\, t_2\, t_3\, t_4\, t_5\, t_6 = \varphi_5(\varphi_3 \vee \varphi_4)(\varphi_1 \vee \varphi_2)\, \varphi_5(\varphi_2 \vee \varphi_4)(\varphi_1 \vee \varphi_3) = \varphi_1\varphi_4\varphi_5 \vee \varphi_2\varphi_3\varphi_5$.

Хроматическое число графа $\chi(G) = 3$. Существует два варианта раскраски графа.

Первый: в синий цвет вершины $\varphi_1 = \{x_3, x_6\}$, в зеленый - $\varphi_4 = \{x_2, x_5\}$, в красный - $\varphi_5 = \{x_1, x_4\}$.

Второй: в синий цвет вершины $\varphi_2 = \{x_3, x_5\}$, в зеленый - $\varphi_3 = \{x_2, x_6\}$, в красный - $\varphi_5 = \{x_1, x_4\}$.

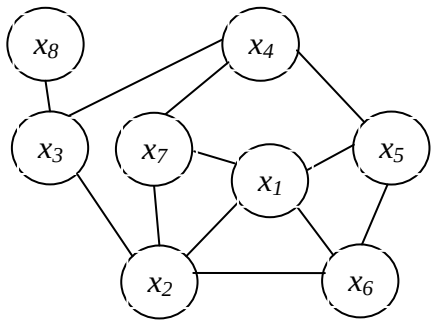
Недостататком точных алгоритмов является низкое быстродействие. Поэтому на практике используют приближенные алгоритмы, примером которых может служить алгоритм, использующий упорядочивание вершин.

7.2.2. Алгоритм, использующий упорядочивание вершин

Алгоритм состоит из следующих операций:

1. Положить $j = 1$;
2. В матрице R подсчитываем число ненулевых элементов r_i;
3. Упорядочим вершины графа в порядке невозрастания r_i.;
4. Просматривая последовательность слева направо, красить в цвет j каждую неокрашенную вершину, не смежную с уже окрашенными в этот цвет;
5. Если остались неокрашенные вершины, то удалить из матрицы R строки и столбцы, соответствующие окрашенным вершинам. Положить $j = j + 1$ и перейти к п. 2, иначе, задача решена.

Пример. Раскрасить вершины графа рис.7.20.



$R =$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	r_i
x_1	0	1	0	0	1	1	1	0	4
x_2		0	1	0	0	1	1	0	4
x_3			0	1	0	0	0	1	3
x_4				0	1	0	1	0	3
x_5					0	1	0	0	3
x_6						0	0	0	3
x_7							0	0	3
x_8								0	1

Рисунок 7.20. Исходный граф и его матрица соединений

1. Положим $j = 1$;
2. Упорядочим вершины графа в порядке невозрастания r_i.
$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$;
3. Красим в первый цвет вершины x_1 и x_3. Вершины x_4, и x_8 смежны вершине x_3, остальные – смежны вершине x_1;
4. Остались неокрашенные вершины, поэтому удалим из матрицы R строки и столбцы, соответствующие вершинам x_1 и x_3. Положим $j = j + 1 = 2$.

$R =$

	x_2	x_4	x_5	x_6	x_7	x_8	r_i
x_2	0	0	0	1	1	0	2
x_4		0	1	0	1	0	2
x_5			0	1	0	0	2
x_6				0	0	0	2
x_7					0	0	2
x_8						0	0

5. Упорядочим вершины графа в порядке невозрастания r_i: $x_2, x_4, x_5, x_6, x_7, x_8$;
6. Красим во второй цвет вершины x_2, x_4 и x_8. Вершины x_5 и x_7, смежны вершине x_4, вершина x_6 смежна вершине x_2;
7. Остались неокрашенные вершины, удалим из матрицы R строки и столбцы, соответствующие вершинам x_2, x_4 и x_8. Положим $j = j + 1 = 3$.

$R =$

	x_5	x_6	x_7	r_i
x_5	0	1	0	1
x_6		0	0	1
x_7			0	0

8. Упорядочим вершины графа в порядке невозрастания r_i: x_5, x_6, x_7.
9. Красим в третий цвет вершины x_5 и x_7. Вершина x_6 смежна вершине x_5;
10. Осталась неокрашенная вершина, удалим из матрицы R строки и столбцы, соответствующие вершинам x_5 и x_7. Положим $j = j + 1 = 4$.
11. В четвертый цвет окрашиваем вершину x_6.

Все вершины окрашены.

Достоинство алгоритма – быстродействие. Недостаток – не оптимальность.

Для раскраски вершин графа приближенным алгоритмом потребовалось четыре цвета. А хроматическое число графа $\chi(G) = 3$. Действительно, если в первый цвет окрасить вершины x_1, x_4 и x_8, во второй – x_2 и x_5, то в третий можно окрасить оставшиеся вершины x_3, x_6 и x_7.

Это связано с тем, что алгоритм рассматривает только количественные характеристики вершин, и не рассматривает качественные. Так, степени вершин x_3 и x_4 одинаковые, но вершина x_3 смежна висячей вершине x_8, а вершина x_4 – сильносвязным вершинам x_5 и x_7.

С. На третьем этапе определяется последовательность проведения соединений, т. е. указывается, когда каждый проводник будет проведен.

7.3. Порядок проведения проводников

Трассировка цепей выполняется последовательно, и каждая проложенная трасса является препятствием для еще не проведенных. В связи с этим большое значение приобретает задача нахождения последовательности проведения соединений в каждом слое. Порядок трассировки оказывает значительное влияние на качество трассировки.

Рассмотрим пример (рис. 7.21 (*а*)). Необходимо соединить одноименные контакты.

Если первыми соединять контакты *a*, то контакты *b* можно соединить с небольшим обходом (рис. 7.21 (*б*)). Если же первыми соединять контакты *b*, то соединение контактов *a* значительно увеличит длину проводника (рис. 7.21 (*в*)).

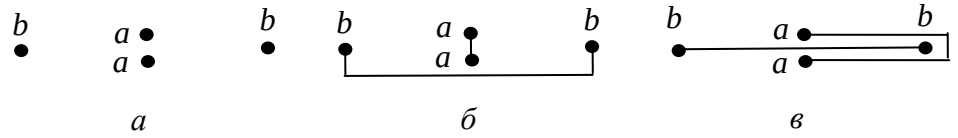


Рисунок 7.21. Зависимость качества трассировки от порядка проведения проводников

Еще один пример (рис. 7.22 (*а*)). Задача та же. Если первыми соединять контакты *b,* то и контакты *a* легко соединяются (рис. 7.22 (*б*)). Если же первыми соединять контакты *a*, то контакты *b* окажутся заблокированными (рис. 7.22 (*в*)).

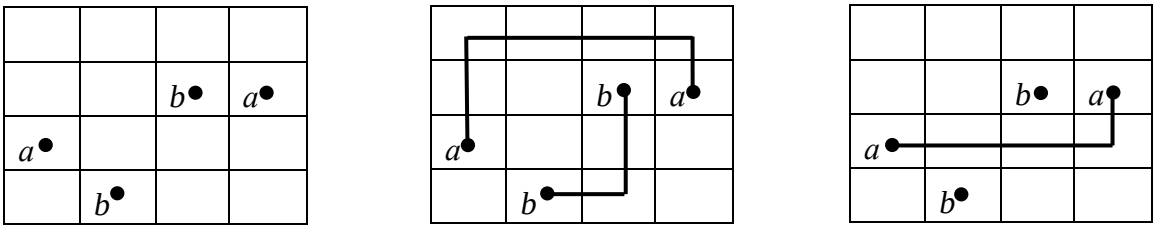


Рисунок 7.22. Влияние порядка проведения проводников

Идея алгоритмов определения порядка проведения соединений заключается в том, что первыми проводятся проводники, меньше «мешающие» другим проводникам.

Метод прямоугольников

1. Для всех пар контактов строятся минимальные охватывающие прямоугольники;
2. Для каждого прямоугольника подсчитывается число контактов других соединений, попавших в данный прямоугольник;
3. Соединения упорядочиваются по неубыванию этих чисел;
4. Получен порядок проведения соединений.

Пример. Определить порядок проведения соединений (рис. 7.23 (*а*)).

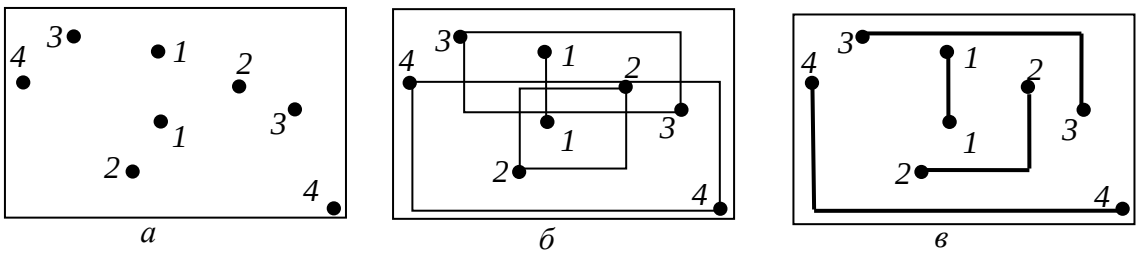


Рисунок 7.23. Определение порядка проведения проводников

1. Строим минимальные охватывающие прямоугольники (рис. 7.23 (*б*));
2. Число контактов других соединений, попавших в данный прямоугольник *1* – 0; *2* – 1; *3* – 2; *4* – 4;

3. Упорядочим соединения *1, 2, 3, 4*;
4. Проведем соединения в найденном порядке (рис. 7.23 (*в*)).

В пакетах автоматизированного проектирования заложена возможность редактирования стратегии трассировки, а именно, выбрать порядок проведения проводников (RouteOrder): Short – Long – сначала короткие, потом длинные, или Long -Short- сначала длинные, потом короткие.

Анализируя результаты работы метода, можно сделать вывод, что первыми следует проводить короткие проводники. Но в этом случае автоматически разводятся простые соединения, заполняя коммутационное пространство (согласно закону Паркинсона «Работа заполняет время, отпущенное на неё»), оставляя длинные проводники на ручную доработку. Поэтому при применении варианта Short-Long число неразведенных связей меньше, но ручная допроводка сопряжена с большими трудностями, чем при использовании варианта Long-Short**.** С другой стороны, если первыми проводить длинные соединения, то на заполненном коммутационном поле короткие проводники станут длинными. Лучший вариант можно выбрать, применив обе стратегии трассировки.

D. На четвертом этапе дается ответ, каким образом (каким алгоритмом) должно быть проведено каждое соединение.

7.4. Трассировка соединений

Это основной и наиболее трудоемкий этап, определяющий эффективность и качество трассировки в целом.

Алгоритмы трассировки можно разбить на следующие группы:

1. Волновые;
2. Лучевые;
3. Канальные.

7.4.1. Волновой алгоритм трассировки

Многие алгоритмы трассировки основаны на волновом алгоритме (алгоритме, предложенном Ли).

Коммутационное поле разбивается на дискреты. Размеры дискретов определяются шириной проводников и расстоянием меду ними (рис. 7.24).

Получаем дискретное рабочее поле (ДРП).

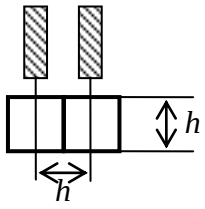


Рисунок 7.24. Дискреты

Волновой алгоритм состоит из двух этапов:

1. Распространение волны;
2. Проведение трассы.

В первой части моделируется процесс распространения волны от источника ячейки A по свободным ячейкам ДРП. Алгоритм последовательно строит $\Phi_1(A)$,

$\Phi_2(A)$, ..., $\Phi_k(A)$ – 1, 2, ..., k фронты. Множество ячеек ДРП, входящих в i-ый фронт A называется i-й окрестностью ячейки A – $O_i(A)$. Если проведение пути между ячейкой-источником A и ячейкой-приемником возможно, то на каком-либо шаге k окажется, что $B \in O_k(A)$. Распространение волны заканчивается.

Если же в следующий фронт не удается включить ни одной ячейки, т. е. $O_{i-1}(A) = O_i(A)$, то пути между A и B не существует.

Ячейкам i-го фронта присваивается значение весовой функции p_i.

Во второй части алгоритма начиная от ячейки-приемника B, по определенным правилам выполняется переход от ячейки k-го фронта к ячейке k-1 фронта и т. д. до ячейки-источника A. Пройденные ячейки составляют искомый путь.

Пусть функция-критерий будет СДС. Тогда вес ячейки i-го фронта будет $p_i = p_{i-1} + 1$, т.е. равен расстоянию от ячейки A до ячеек i-го фронта. В этом случае, при построении пути осуществляется переход к тем ячейкам, в которых значение p_i монотонно убывает.

Пример. Соединить волновым алгоритмом ячейки A и B (рис. 7.25 (*а*)).

На рис. 7.25 показаны: первый шаг распространения волны (*б*); четвертый шаг – (*в*); седьмой шаг – (*г*); заключительный шаг – (*д*).

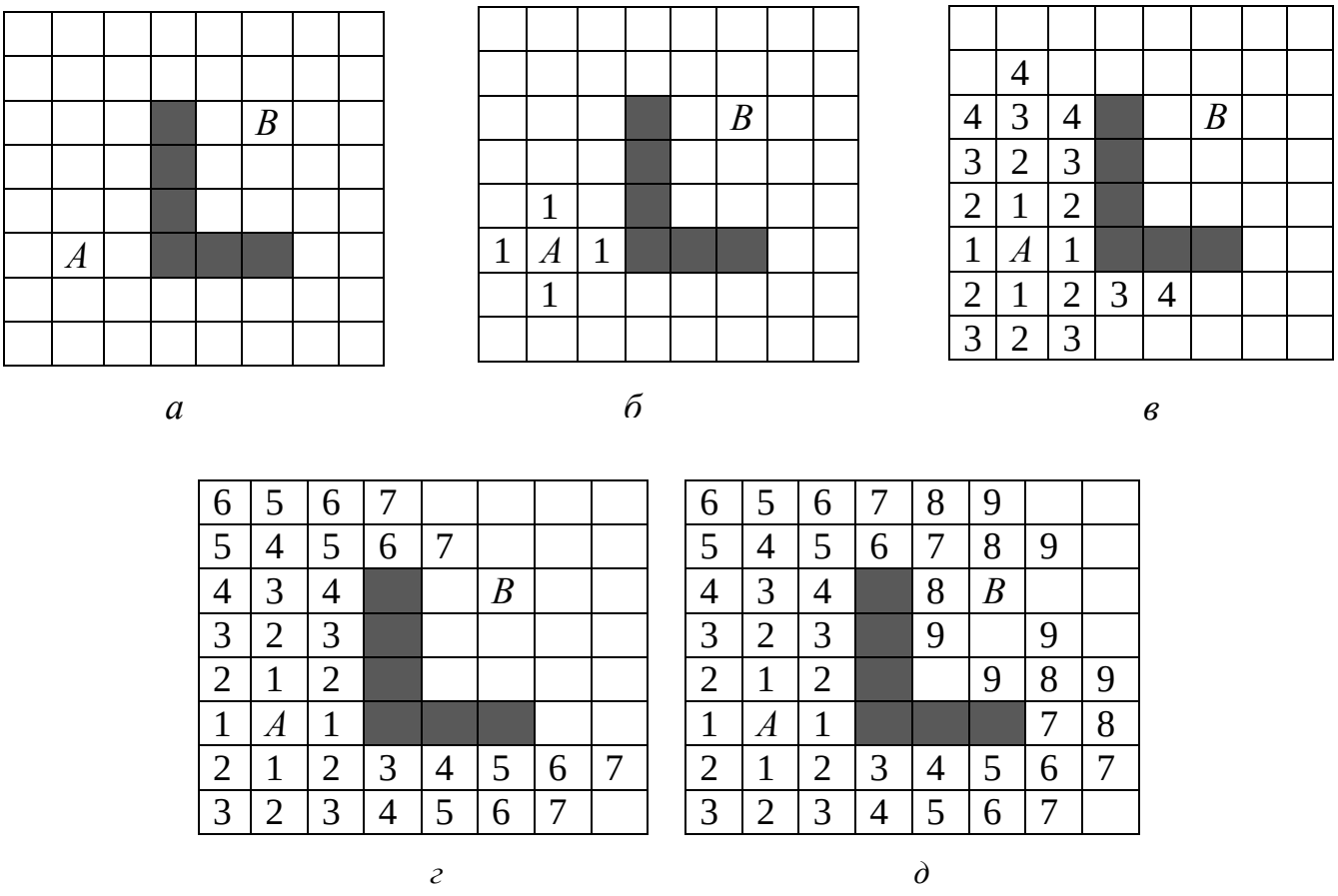


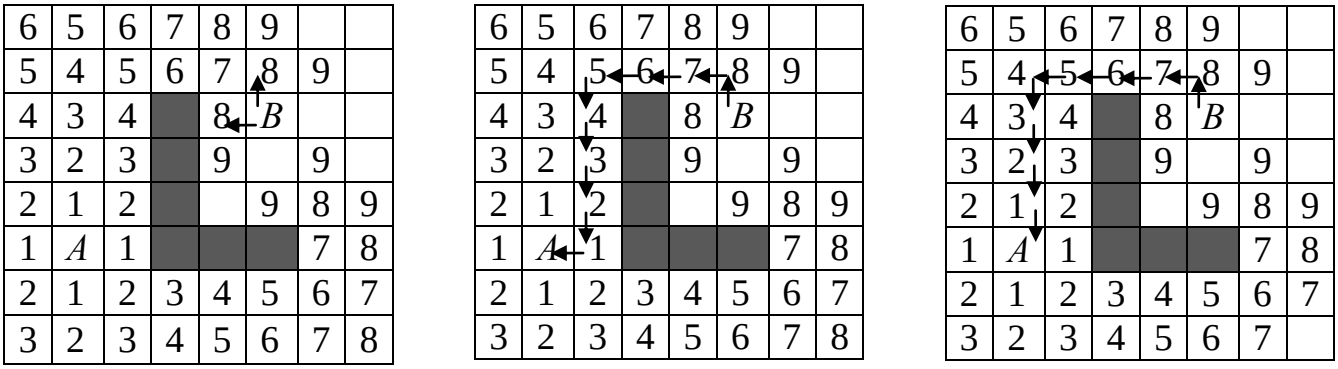
Рисунок 7.25. Распространение волны

На девятом шаге волна достигла ячейку B.

При построении пути возможны два выхода из ячейки B (рис. 7.26 (*а*)). Направление движения определяется правилом приоритетов – вес какой соседней

ячейки проверяется. Пусть ячейки проверяются в следующем порядке: ↑→↓←. Тогда искомый путь будет выглядеть так (рис.7.26 (*б*)).

Если приоритеты будут ↑←↓→, искомый путь будет выглядеть так (рис.7.26 (*в*))



а *б* *в*

Рисунок 7.26. Построение пути

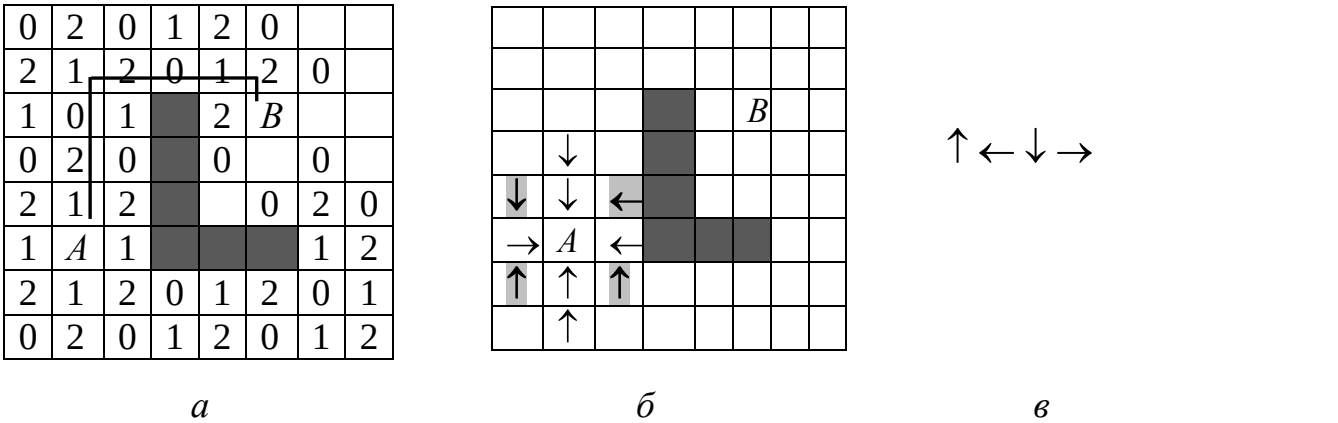
Вычислительная сложность волнового алгоритма близка к $O(n^2)$.

Каждая ячейка ДРП в процессе работы может быть в одном из следующих состояний: свободная, занятая или содержать весовую метку 1, 2, ..., *L,* где *L* – максимальная длина пути на ДРП. Таким образом, требуемое число разрядов памяти на одну ячейку ДРП составляет $N = \lceil \log_2(L + 2) \rceil$, где $\lceil x \rceil$ - ближайшее не меньшее *x* целое.

Поэтому при реализации волнового алгоритма важная проблема – сокращение объема памяти, необходимой для хранения ДРП.

7.4.2. Волновой алгоритм с кодированием по *mod* 3

С целью уменьшения объема памяти для хранения ДРП используется то обстоятельство, что при построении пути при выборе очередной ячейки, соседней с ячейкой с весом *k* находятся только ячейки с весами *k* – 1 и *k* + 1. Поэтому достаточно хранить не сами веса, а только отметки 0, 1, 2, сравнимые с *k* по модулю 3 (рис. 7.27 (*а*)).



а *б* *в*

Рисунок 7.27. Распространение волны по *mod* 3 (*а*), два шага метода путевых координат (*б*), правило приоритетов (*в*)

Для проведения пути необходимо выбирать последовательность 0, 1, 2 в обратном порядке.

При кодировании по *mod* 3 каждая ячейка ДРП в процессе работы может быть в одном из следующих состояний: свободная, занятая или содержать весовую метку 0, 1, 2. Таким образом, требуемое число разрядов памяти на одну ячейку ДРП составляет

$N = \lceil \log_2(5) \rceil = 3$. Важно, что N не зависит от длины пути.

7.4.3. Метод путевых координат

Путевой координатой ячейки c_i фронта Φ_k будем называть ту, соседнюю с ней ячейку фронта Φ_{k-1}, от которой она получает свой вес.

Для рассматриваемой окрестности соседства имеем четыре возможные путевые координаты ↑, ←, ↓, →. Если имеется несколько соседних с c_i ячеек фронта Φ_{k-1} (рис. 7.27 (*б*)), то назначение путевой координаты производится согласно выбранному правилу приоритетов, т.е. порядку просмотра соседних ячеек (рис. 7.27(*в*)). Результат этапа распространения волны показан на рис. 7.28 (*а*).

Этап проведения пути состоит в отслеживании путевых координат в размеченном ДРП, начиная от ячейки - приемника B (рис. 7.28 (*б*)).

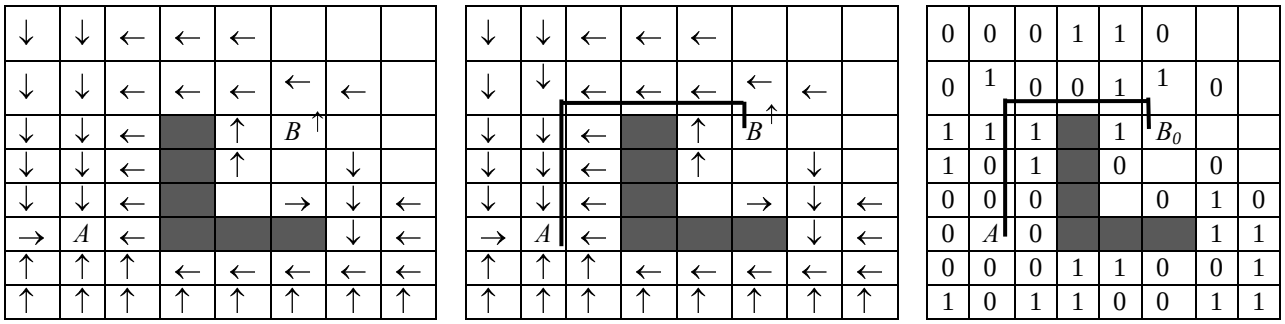


Рисунок 7.28. Распространение волны методом путевых координат (*а*), построение пути (*б*), Метод Акерса (*в*)

При кодировании методом путевых координат каждая ячейка ДРП в процессе работы может быть в одном из следующих шести состояний: свободная, занятая или содержать путевую координату ↑, ←, ↓, →. Таким образом, требуемое число разрядов памяти на одну ячейку ДРП составляет

$N = \lceil \log_2(6) \rceil = 3$.

7.4.4. Метод Акерса

Для определения последовательности ячеек, составляющих путь, достаточно, чтобы при распространении волны ячейкам присваивались значения отметок из последовательности, в которой каждый член имеет разных соседей слева и справа, т.е. $p_{k-1} \neq p_k \neq p_{k+1}$.

Акерс предложил последовательность 00110011..., которая обладает этим свойством.

Для построения пути необходимо выбрать последовательность в обратном порядке (рис. 7.28 (*в*)).

При кодировании методом Акерса каждая ячейка ДРП в процессе работы может быть в одном из следующих четырех состояний: свободная, занятая или содержать отметки 0 или 1. Таким образом, требуемое число разрядов памяти на одну ячейку ДРП составляет $N = \lceil \log_2(4) \rceil = 2$.

7.4.5. Оптимизация пути по нескольким параметрам

Волновой алгоритм может одновременно учитывать несколько критериев: длина пути, число перегибов, число переходов со слоя на слой и т.д. Тогда вес ячейки *k*-го фронта будет вычисляться по формуле

$p_k = p_{k-1} + \sum_{i=1}^{n} a_i f_i(k)$, где a_i - весовой коэффициент, учитывающий важность *i*-го критерия; $f_i(k)$ – значение *i*-го параметра для рассматриваемой ячейки; n – количество учитываемых критериев.

Пример. Пусть необходимо оптимизировать трассу по длине и числу перегибов, причем число перегибов в три раза важнее длины пути:

$$\begin{cases} a_1 = 1, \\ a_2 = 3, \end{cases} \quad \begin{cases} p_1 = 1, \\ p_2 = \begin{cases} 1, \text{есть перегиб}, \\ 0, \text{нет перегиба}. \end{cases} \end{cases}$$

Тогда, $p_k = p_{k-1} + 1 + 3p_2$.

На втором шаге алгоритма в ячейку после изгиба ставится метка 5 (рис. 7.29 (*а*)), и волна в этой ячейке «засыпает», пока есть возможность ставить метки 3, 4 и 5. После этого она становится активной и идет дальше.

Проведение пути заключается в выборе ячейки с меньшим весом, при этом, пока возможно, сохраняется направление движения (рис. 7.29 (*б*)).

Построен путь с двумя перегибами и длиной – 11. Минимальная возможная длина - 9 при пяти перегибах (пунктирный путь на рис. 7.29 (*б*)).

Волновой алгоритм характеризуется высокой эффективностью нахождения пути, но требует значительных временных затрат. Причем 90% времени тратится на распространение волны и 10% - на проведение пути.

9	5	9	10	11	12	13	14
8	4	8	9		16	B_{17}	
7	3				19		
6	2		10	14	15		
5	1	5	6	7			17
1	*A*					15	16
5	1	5	6	7		14	15
6	2	6	7	8	9	10	11

а

9	5	9	10	11	12	13	14
8	4	8	9		16	B_{17}	
7	3				19		
6	2		10	14	15		
5	1	5	6	7			17
1	*A*					15	16
5	1	5	6	7		14	15
6	2	6	7	8	9	10	11

б

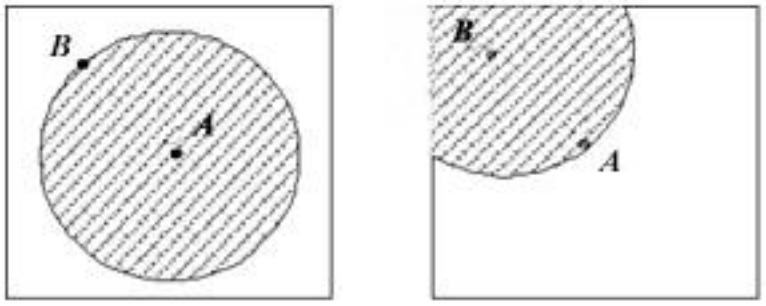
Рисунок 7.29. Распространение волны (*а*), построение пути (*б*)

7.4.6. Методы повышения быстродействия волнового алгоритма

Методы сокращают время распространения волны. Это методы:

1. Выбор источника волны.

Время распространения волны пропорционально площади заштрихованных фигур (рис. 7.30). В качестве источника волны выбирается ячейка, ближе расположенная к краю КП.



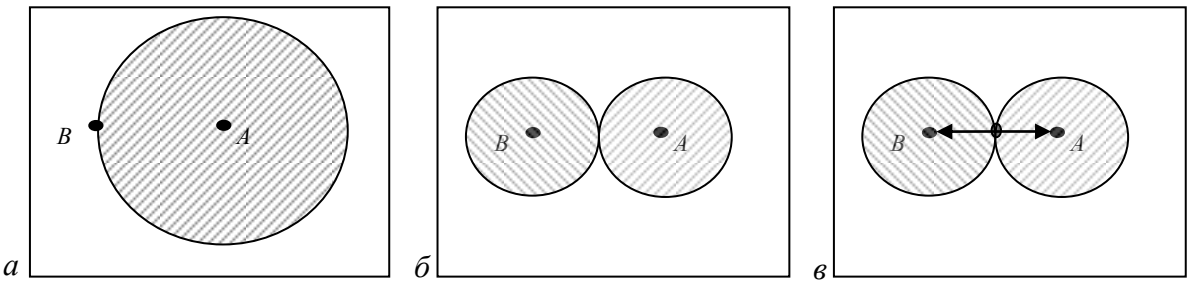
а *б*

Рисунок 7.30. Источники волны ячейка *A* (*а*), ячейка *B* (*б*)

2.

3. Метод встречной волны.

Волна попеременно распространяется из двух источников (*A* и *B*) до тех пор, пока волны не встретятся (рис. 7.31 (*б*)). Путь строится, начиная с ячейки встречи, до источников (рис. 7.31 (*в*)).



а *б* *в*

Рисунок 7.31. Метод встречной волны

Площадь заштрихованной фигуры рис. 7.31 (*а*) $S_1 = \pi L^2_{AB}$, а фигуры рис. 7.31 (*б*) – $S_2 = 2\pi (L_{AB}/2)^2 = \pi L^2_{AB}/2$, т.е. в два раза меньше. Время распространения волны во втором случае, с учетом потери времени на организацию попеременной волны, почти в два раза меньше.

4. Обрамление соединяемых точек.

Путь ищется не во всем коммутационном поле а в окрестности ячеек *A* и *B* (рис.7.32)

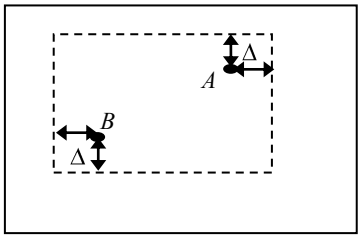


Рисунок 7.32. Обрамление области поиска пути

Если не удается найти путь, то рамка раздвигается на Δ дискрет с каждой стороны. В пакетах автоматизированного проектирования в стратегии трассировки задается параметр Δ и количество неудачных попыток нахождения пути (обычно три) после чего обрамление снимается и путь ищется на всем КП.

7.4.7. Многослойная трассировка

Разработаны алгоритмы, являющиеся обобщениями волнового алгоритма и позволяющие осуществлять трассировку в пространстве. В качестве рабочего пространства используется набор слоев, связанных между собой межслойными переходами (рис. 7.33).

Одним из таких алгоритмов является алгоритм многослойной трассировки Хейса. При распространении волны свободным ячейкам ДРП$_i$ присваивается индекс длины пути p_i и индекс количества межслойных переходов v_i.

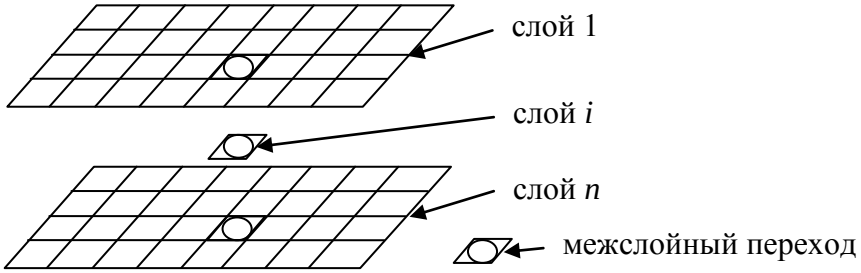


Рисунок 7.33. Дискретное рабочее поле для многослойной трассировки

При расчете длины p_i межслойные переходы учитываются путем добавления k единиц длины на каждый переход.

Рассмотрим трассировку соединений двухслойной печатной платы. Примем цену перехода $k = 1$ (рис. 7.34).

В алгоритме Хейса волна распространяется независимо в каждом слое, что приводит к значительным затратам времени и памяти при реализации.

Джейер распространил однослойный вариант трассировки на n слоев. В алгоритме Джейера используется единственное плоское ДРП, которое отображает состояние ячеек для всех слоев. В этом ДРП реализуется процедура распространения волны. Оно происходит одновременно во всех слоях схемы.

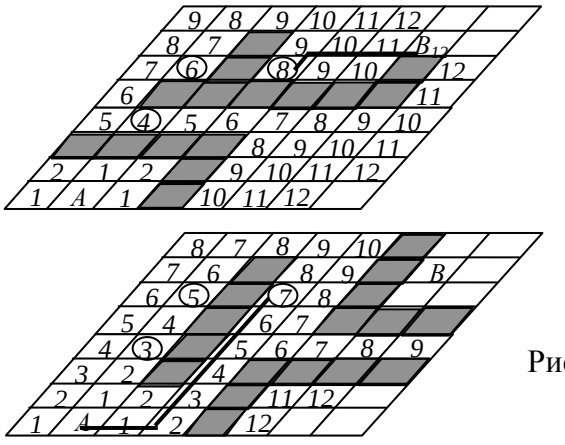


Рисунок 7.34. Двухслойная трассировка

Для каждой ячейки ДРП вводятся следующие поля описания (рис. 7.35):

p – индекс длины (вес);

$A = ||a_i||_n$ – n признаков занятости ячейки по каждому слою;

$B = ||b_i||_n$ – n признаков допустимости для включения ячейки в путь по каждому слою;

v – возможные переходы.

v	b_n	...	b_2	b_1	a_n	...	a_2	a_1	p
$2n+3$	$2n+2$		$n+4$	$n+3$	$n+2$		4	3	2 1

Рисунок 7.35. Формат ячейки ДРП

Для кодирования каждой ячейки ДРП требуется $2n + 3$ разрядов.

Достоинство волнового алгоритма то, что он всегда строит путь, если он существует и этот путь минимальной длины.

Недостатками алгоритма являются трудоемкость, большой объем требуемой оперативной памяти и последовательный характер построения трасс. Сложность волнового алгоритма составляет $O(N \cdot M)$, где N – число клеток ДРП; M – число контактов.

7.5. Лучевые алгоритмы трассировки

В большинстве схем при правильной их компоновке, значительная часть соединений достаточно простой формы и для проведения пути нет необходимости рассматривать все клетки решетки. В этих случаях целесообразно использовать лучевые алгоритмы трассировки. Основная идея лучевых алгоритмов состоит в исследовании ДРП для определения пути между вершинами по некоторым заранее заданным направлениям (лучам).

Рассмотрим следующие лучевые алгоритмы:

1. Алгоритм Л.Б. Абрайтиса;
2. Алгоритм Миками – Табучи (малоповоротный алгоритм).

7.5.1. Алгоритм Абрайтиса

Работа алгоритма заключается в следующем. Задается число лучей, распространяемых из точек A и B, а также порядок присвоения путевых координат (обычно число лучей для каждой ячейки - источника принимается одинаковым). Лучи $A(1)$, $A(2)$,…, $A(n)$ и $B(1)$, $B(2)$,…, $B(n)$ считают одноименными, если они распространяются из одноименных источников A или B. Лучи $A(i)$ и $B(i)$ являются разноименными по отношению друг к другу.

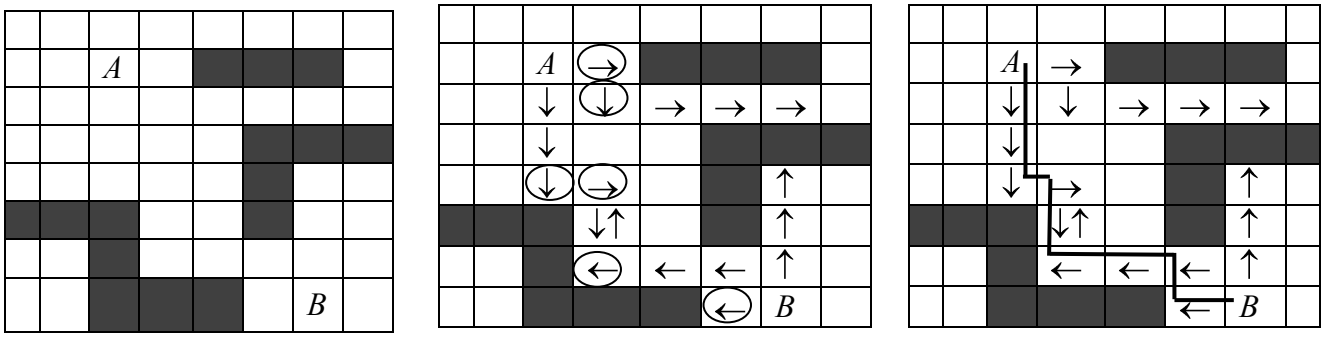
Распространение лучей производят одновременно из обоих источников до встречи двух разноименных лучей в некоторой ячейке C. Путь проводится из ячейки C и проходит через ячейки, по которым распространялись лучи.

Возьмем для источников A и B по два луча с противоположными направлениями. Если A выше и левее B, то A^1 - $\downarrow$, $\rightarrow$; A^2- $\rightarrow$, $\downarrow$; B^1 - $\uparrow$, $\leftarrow$; B^2- $\leftarrow$, $\uparrow$.

Если на пути луча встречается препятствие, то с помощью альтернативного луча алгоритм пытается обойти препятствие, например, если препятствие встретилось у луча A^2 $\rightarrow$, то препятствие обходится с помощью луча $\downarrow$. При первой возможности основной луч выходит на свое направление.

При распространении луча может возникнуть ситуация, когда препятствие не удается обойти ни в одном направлении. В этом случае луч считается заблокированным и его распространение прекращается.

Пример. Соединить алгоритмом Абрайтиса ячейки *A* и *B* (рис. 7.36 (*а*))



а *б* *в*

Рисунок 7.36. Построение пути алгоритмом Абрайтиса

На рис. 7.36 (*б*) показан процесс распространения лучей. Овалами показаны ячейки, в которых луч меняет направление распространения на альтернативное. Вертикальный луч ячейки *B* на четвертом шаге «затух». Ячейка встречи разноименных лучей *C* (↓↑). Начиная с нее, строится путь к ячейка источникам (рис. 7.36 (*в*)).

Из рисунков видно, что площадь, занятая метками, значительно меньше, чем у волнового алгоритма.

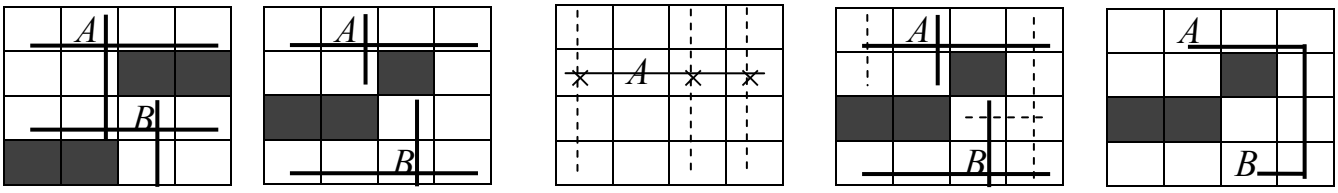
Обычно с помощью лучевого алгоритма удается построить до 70-80% трасс, остальные проводят, используя волновой алгоритм или вручную. Кроме того, волновой алгоритм на заполненном ДРП работает значительно быстрее.

Применение алгоритма Абрайтиса особенно выгодно при проектировании плат с невысокой плотностью монтажа.

7.5.2. Алгоритм Миками – Табучи

Из ячеек *A* и *B* до встречи с препятствием проводятся по два перпендикулярных луча. Если разноименные лучи пересеклись, то путь найден (рис. 7.37 (*а*).

Если нет (рис. 7.37 (*б*), то ячейки ранее построенных лучей становятся источниками для проведения перпендикулярных лучей (рис. 7.37 (*в*) и т.д.



а *б* *в* *г* *д*

Рисунок 7.37. Порядок построения пути алгоритмом Миками - Табучи

На рис.7.37 (*г*) показаны вторичные лучи ячеек *A* и *B*. Разноименные лучи пересеклись. Путь строится по лучам, построенным раньше (рис. 3.37.(*д*)).

Формально алгоритм Миками – Табучи можно записать следующим образом:

1. Положить $i = 0$.
2. Через ячейки A и B провести перпендикулярные лучи. Ячейки, по которым прошли лучи поместить в списки A_i и B_i.
3. Если $A_i \cap B_i \neq \varnothing$, то путь найден. Конец.
4. Если нет, то для всех A_i и B_i найти точки ветвления.
5. Положить $i = i + 1$. Провести через них перпендикулярные лучи. Ячейки, по которым прошли лучи поместить в списки A_i и B_i
6. Если $A_i \cap B_i \neq \varnothing$, то путь найден. Конец.
7. Пока $A_i \cap B_i \neq \varnothing$ или $A_i \neq A_{i-1}$ или $B_i \neq B_{i-1}$ повторить п.п. 3-5.
8. Пути между ячейками не существует.

Если ограничить число повторов п.п. 3-5 одним, то будут построены малоповоротные пути (рис.7.38).

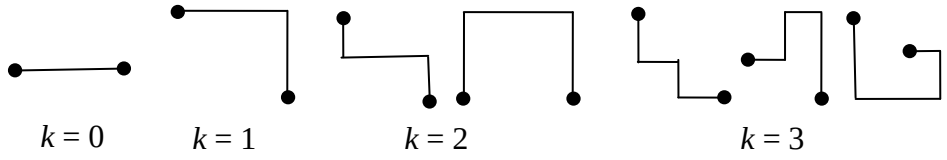


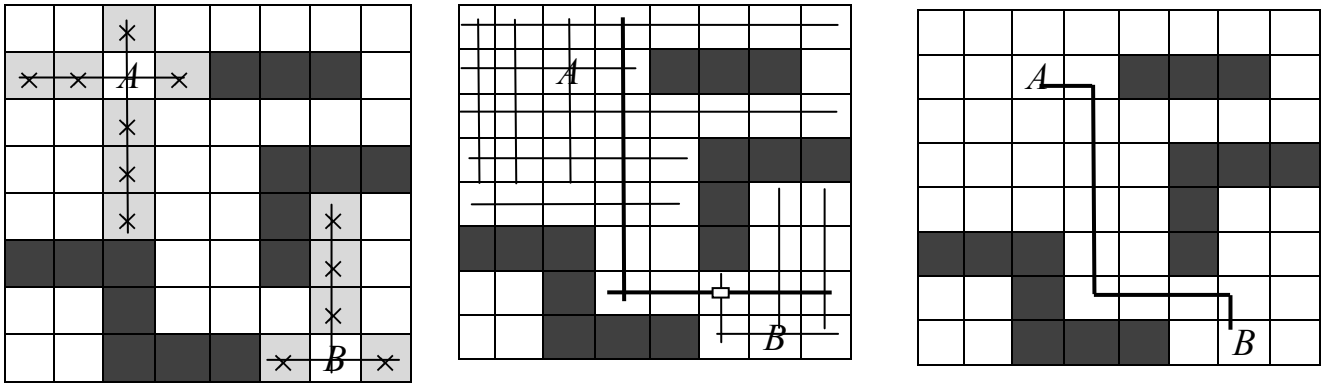
Рисунок 7.38. Конфигурации трасс с разным числом поворотов k

Рассмотрим работу алгоритма для примера рис. 7.36 (*а*).

Проведем через ячейки A и B перпендикулярные лучи. Ячейки, по которым прошли лучи поместим в списки A_0 и B_0. Найдем точки ветвления (рис 7.39 (*а*)).

$A_0 \cap B_0 = \varnothing$. Положим $i = i + 1$ и проведем вторичные лучи (рис 7.39 (*б*)).

$A_1 \cap B_1 \neq \varnothing$. Строим путь (рис. 7.19 (*в*)). Заметим, что к ячейке B трасса поворачивает в луче B_0, а не в B_1 (ячейка, помеченная ▫), так как пересеклись вторичные лучи вертикальный A и горизонтальный B.



а *б* *в*

Рисунок 7.39. Построение пути алгоритмом Миками - Табучи

Алгоритм строит малоповоротные, но не оптимальные пути. Скорость работы на два порядка выше, чем у волнового алгоритма.

Недостатком лучевых алгоритмов является их последовательный характер. Оптимизируется каждая отдельная трасса без прогноза последующих соединений.

В меньшей степени этот недостаток свойственен канальным алгоритмам.

7.6. Канальные алгоритмы трассировки

В настоящее время распространение получили канальные алгоритмы трассировки, основанные на проложении трасс по укрупненным дискретам КП, в качестве которых служит система горизонтальных и вертикальных каналов. Ширина каждого канала зависит от числа прокладываемых в них соединений. Канал разбивается на линейки, называемые магистралями. Любое соединение при канальной трассировке будет представлять совокупность объединенных в одну цепь участков магистралей.

Реализация канального алгоритма предполагает выполнение двух процедур:

1. Распределение соединений по каналам с учетом их оптимальной загрузки;
2. Оптимизация расположения соединений на магистралях каналов.

Для заданной конструктивно-технологической базы каждому каналу монтажной плоскости можно поставить в соответствие число, называемое пропускной способностью канала и обозначающее максимальное допустимое число проводников, проходящих через сечение канала с выполнением технологических ограничений. При этом процедура оптимального распределения соединений по каналам в простом случае сводится к их равномерной загрузке, а в более сложных случаях, кроме того, осуществляется учет электромагнитной и тепловой совместимости соседних проводников.

Целью выполнения второй процедуры является минимизация числа переходных отверстий с одного слоя монтажной плоскости на другой.

Задачи первой процедуры решаются с помощью алгоритмов построения минимальных связывающих деревьев (МСД). В процессе построения МСД для каждого соединения учитывается загрузка каждого из каналов, через которые оно проходит.

Часто для распределения соединений по каналам вместо процедур построения МСД используются волновые процедуры, при этом дискретами для распространения волны на коммутационном поле являются каналы.

Задача распределения соединений по магистралям обычно формулируется следующим образом. Дан горизонтальный канал, ограниченный верхним и нижним рядами. Между верхним и нижним рядами заданы множества свободных линий, магистралей. Необходимо в общем случае ломаными линиями, проходящими по участкам магистралей, соединить все абсциссы одноименных групп контактов (каждая группа соответствует одной и той же цепи), затем вертикальными отрезками соединить контакты. Трассировку следует производить по выбранному критерию качества (суммарная длина, число межслойных переходов, процент реализованных соединений, количество занятых магистралей и т. д.).

Первый алгоритм канальной трассировки (так называемая «классическая» постановка задачи, не допускающая излома или перехода горизонтального сегмента соединения с магистрали на магистраль) был предложен Хашимото и Стивенсоном в 1971 г. Алгоритм получил также название «алгоритм левого конца».

Пусть задано некоторое множество отрезков, распределенных в одном канале $S=\{M_1, M_2, \ldots, M_n\}$. Два отрезка считаются пересекающимися, если $M_i \cap M_j \neq \varnothing$. Графом интервалов $G(S)$ множества S называется граф, вершинами которого являются отрезки M_i, а ребра соответствуют пересечению M_i и M_j. Теперь задача распределения соединений по магистралям сводится к минимальной раскраске графа интервалов $G(S)$. При этом окрашенные одним цветом отрезки помещаются на одной магистрали, а хроматическое число графа интервалов – необходимое для проведения соединений число магистралей.

Алгоритм «левого конца» является достаточно простым алгоритмом раскраски графа.

Алгоритм заключается в следующем:

1. Упорядочить отрезки по координатам левого конца;

2. Просматривая последовательность слева направо, красить вершины первой по порядку краской, которой не окрашены смежные с ней вершины.

Пример. Даны отрезки, координаты, которых получены на первом этапе. Строим граф интервалов (рис. 7.40).

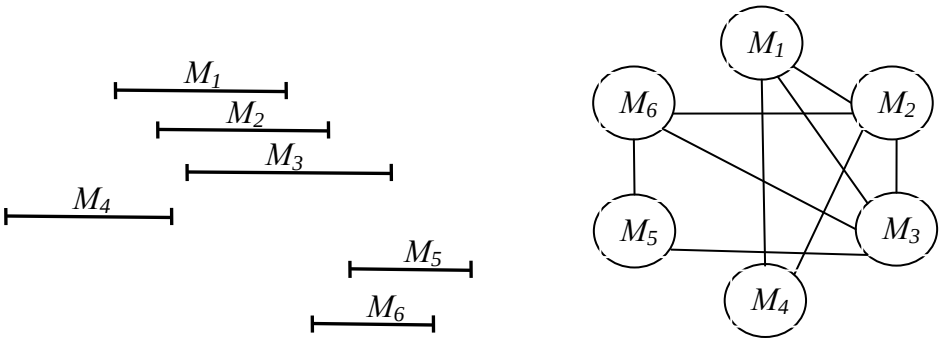


Рисунок 7.40. Отрезки соединений и граф интервалов $G(S)$

Упорядочим отрезки по координатам левого конца: $M_4, M_1, M_2, M_3, M_6, M_5$

Красим вершину M_4 в первый цвет (помещаем на первую магистраль). Вершина M_1 смежна вершине M_4, красим ее во второй цвет (помещаем на вторую магистраль). Вершина M_2 смежна вершинам M_4 и M_1, красим ее в третий цвет (помещаем на третью магистраль). Вершина M_3 не смежна вершине M_4, красим ее в первый цвет (помещаем на первую магистраль). Вершина M_6 смежна вершине M_4 и не смежна вершине M_1, красим ее во второй цвет (помещаем на вторую магистраль). Вершина M_5 смежна вершинам M_3 и M_6, и не смежна вершине M_2, красим ее в третий цвет (помещаем на третью магистраль). Результаты распределения отрезков по магистралям приведены на рис. 7.41 (*а*).

Однако на практике недостаточно разместить проводники на горизонтальных магистралях, необходимо подсоединить их вертикальными проводниками к соответствующим выводам канала. Для этого, вводится граф вертикальных ограничений, показывающий какое соединение должно стоять выше, а какое ниже. На рис. 7.41(*б*) показан горизонтальный канал. Граф вертикальных ограничений для соединений этого канала приведен на рис. 7.41(*в*).

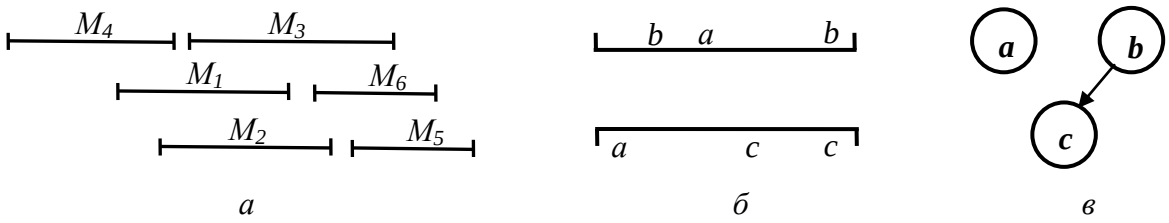


Рисунок 7.41. Распределение проводников в канале (*а*); горизонтальный канал (*б*); граф вертикальных ограничений (*в*)

Распределение проводников по магистралям с учетом графа вертикальных ограничений показано на рис. 7.42 (*а*). Распределение не оптимально. Оптимальное распределение, полученное с помощью «алгоритма правого конца», показано на рис. 7.42 (б).

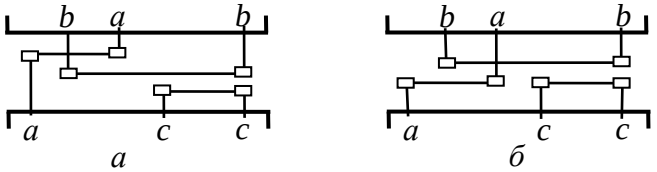


Рисунок 7.42. Распределение соединений по «алгоритму левого конца» (*а*) и оптимальное распределение по «алгоритму правого конца» (*б*)

Д.Н. Дойч предложил алгоритм «Dogleg» (дословно – собачья лапа, резкое искривление), разрешающий строить ломаную линию. Алгоритм состоит из двух частей:

1. Соединения упорядочиваются по левому концу с учетом графа вертикальных ограничений. Соединения проводятся на возможно верхних магистралях с разрешением изломов.

2. Начиная с правого соединения, они спрямляются с целью уменьшения числа межслойных переходов.

На рис. 7.43 (*а*) показаны контакты, которые необходимо соединить. На рис. 7.43 (*б*) приведен граф вертикальных ограничений. На рис. 7.43.(*в*) показан результат первого этапа алгоритма, а на рис. 7.43 (*г*) – окончательный результат. Удалось спрямить только соединение *d – d*.

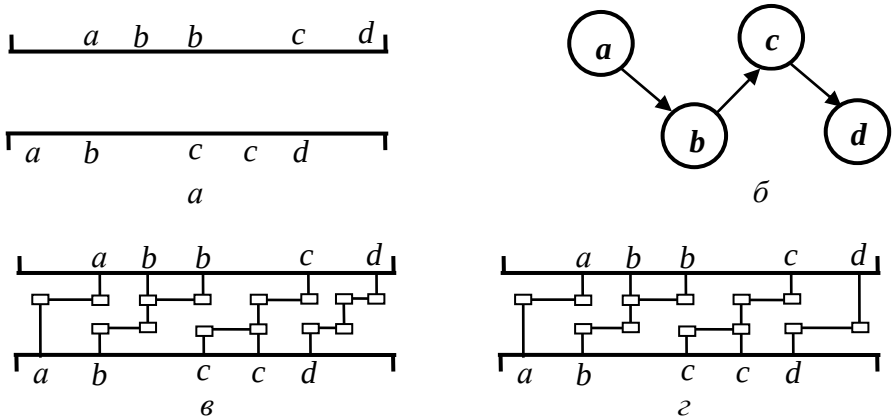


Рисунок 7.43. Распределение соединений по магистралям алгоритмом «Dogleg»

Хороший результат дает модификация алгоритма «Dogleg», заключающаяся в попеременном использовании «алгоритмов левого (для верхних магистралей) и правого (для нижних) концов».

Следует отметить, что при классической постановке задачи возможно возникновение тупиковых ситуаций, обусловленных наличием циклов в графе вертикальных ограничений.

Рассмотрим пример на рис. 7.44 (*а*). В графе вертикальных ограничений есть цикл. Задача не может быть решена в классической постановке, поэтому необходимо применить излом горизонтальных сегментов с магистрали на магистраль. Обычно в таких случаях вводятся дополнительные псевдоконтакты, например *a*' и *a*" (рис. 7.44 (*б*) или 7.44 (*в*)), около которых соединение изламывается, и далее обычным образом оперируют двумя отрезками *a*-*a*' и *a*"-*a*.

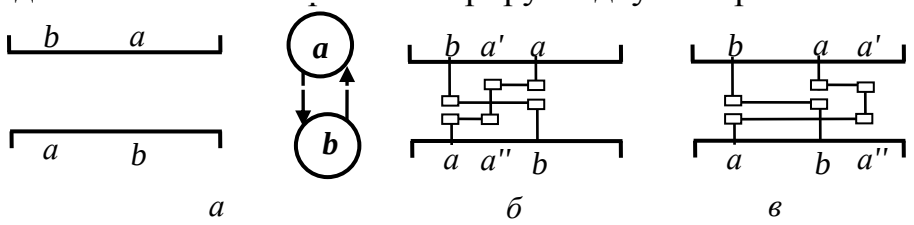


Рисунок 7.44. Обработка тупиковой ситуации алгоритмом «Dogleg»

Низкая временная и пространственная сложность алгоритмов канальной трассировки делает их наиболее приемлемыми в САПР электронных систем, где решаются задачи огромной размерности (несколько миллионов транзисторов).

Рассмотренный метод канальной трассировки успешно применим к регулярным структурам печатных плат или БИС**.**

Достоинствами канальных алгоритмов являются:

1. Быстродействие (на один – два порядка выше, чем у волновых алгоритмов);

2. Меньший расход оперативной памяти ЭВМ (хранится не все ДРП, а только дискреты одного канала);

3. Параллельный характер работы. При распределении соединений по магистралям учитываются конфликты с другими проводниками.

Недостатком канальных алгоритмов является то, что они менее универсальны, используются только для регулярных конструкций.

7.7. Программа автоматической трассировки SPECCTRA

Рассмотренные алгоритмы имеют общей чертой наличие координатной сетки. Принципиально отличными от них являются бессеточные алгоритмы программы SPECCTRA.

SPECCTRA — коммерческая программа автоматической трассировки печатных плат от компании Cadence Design Systems. Современное название Allegro PCB Router. SPECCTRA успешно трассирует платы большой сложности благодаря применению нового принципа представления графических данных — так называемой ShapeBased-технологии. В отличие от известных ранее пакетов, в которых графические объекты представлены в виде набора точек-координат, в этой программе используются более компактные способы их математического

описания. За счёт этого повышается эффективность трассировки печатных плат с высокой плотностью расположения компонентов, обеспечивается автоматическая трассировка одной и той же цепи трассами разной ширины и др.

SPECCTRA использует адаптивные алгоритмы, реализуемые за несколько проходов трассировки. На первом проходе выполняется соединение абсолютно всех проводников без обращения внимания на возможные конфликты, заключающиеся в пересечении проводников на одном слое и нарушении зазоров. На каждом последующем проходе автотрассировщик пытается уменьшить количество конфликтов, разрывая и вновь прокладывая связи (метод *ripup-and-retry*) и проталкивая проводники, раздвигая соседние (метод *push-and-shove*).

Программа совместима с большинством современных систем проектирования печатных плат, благодаря использованию стандартного промышленного формата DSN для описания проектов и Do-файлов для задания стратегии трассировки.

В заключении необходимо отметить следующее:

1. Ни один алгоритм трассировки не гарантирует 100% трассировку, поэтому необходимо предусмотреть доработку топологии. В связи с этим, алгоритмы трассировки не обязательно должны обеспечивать максимальное число разведенных соединений, но, главное, достаточно простую конфигурацию трасс.

2. При проектировании ЭВМ необходимо удовлетворять противоречивым требованиям. Например, при размещении модулей на КП в центр помещается элемент, имеющий максимальное число связей. При работе именно этот модуль выделяет наибольшее количество тепла. Но центральная позиция платы или кристалла хуже всего вентилируется.

8. Графо-теоретический подход к синтезу топологии

Для схем с однослойной коммутацией синтез топологии по традиционной схеме размещение-трассировка не оправдан, т.к. при этом учитываются метрические, а не топологические критерии и ограничения. При неравномерном использовании коммутационных слоев (например, когда слои изготавливаются на основе алюминия и поликремния) проектирование двухслойных схем оказывается близким к проектированию однослойных схем.

Графо-теоретический подход к синтезу топологии состоит из следующих этапов.

1. Построение графовой модели схемы.
2. Анализ планарности графа.
3. Планаризация графа.
4. Реализация непланарных соединений.
5. Построение плоского чертежа схемы.
6. Синтез геометрии схемы.

Одной из основных топологических задач проектирования коммутации является возможность разложения соединений на плоскости без пересечений. Эта задача связана с определением планарности графа. Среди критериев планарности графа наиболее известен ***критерий Понтрягина-Куратовского***:

Граф $G(X, U)$ – планарен тогда и только тогда, когда он не содержит подграфов гомеоморфных полному графу K_5 или полному двудольному графу $K_{3,3}$.

Однако он неконструктивен (требует перебора). Известны другие критерии, но их также трудно использовать. Разработан ряд алгоритмов определения планарности графа удобных для реализации на ЭВМ.

Критерий Бадера. Граф $G(X,U)$ планарен, если его граф пересечений G' является бихроматическим (двудольным) графом. Критерий справедлив для графов, имеющих гамильтонов цикл.

8.1. Разбиение графа на планарные суграфы

При разработке топологии возникает задача выделения из графа, описывающего схему, максимальных планарных суграфов (добавление любого ребра делает его непланарным).

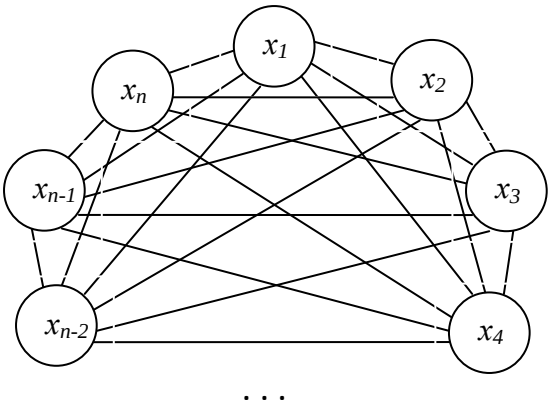
Граф $G(X,U)$ называется m-планарным, если существует m планарных суграфов $L_1(X,Z_1)$, $L_2(X,Z_2)$, …, $L_m(X,Z_m)$, таких, что $U = \bigcup_{i=1}^{m} Z_i$. Наименьшее m называется *толщиной графа*.

8.1.1. Построение графа пересечений *G'*

Приняты следующие допущения:

1. Граф G имеет гамильтонов цикл;
2. Два ребра пересекаются только один раз;
3. Ребра, инцидентные одной вершине, не пересекаются;
4. Ребра графа не пересекают ребер гамильтонова цикла;
5. Ребра вершины x_j могут пересекать ребра вершины x_i при условии, что $j > i$.

Рассмотрим некоторый граф $G(X,U)$ (рис. 8.1).



p_i – число пересечений ребер i-ой вершины.

Рисунок 8.1. Граф $G(X,U)$

Согласно п. 5 принятых допущений $p_1=0$.

Рассмотрим ребро (x_2x_n). Число его пересечений с ребрами вершины x_1

$$p_{2n} = r_{2n}\sum_{i=3}^{n-1} r_{1i}.$$

Ребро (x_2x_{n-1}) пересекает

$$p_{2(n-1)} = r_{2(n-1)} \sum_{i=3}^{n-2} r_{1i} \quad \text{ребер.}$$

Общее число пересечений ребер вершины x_2

$$p_2 = \sum_{j=4}^{n} r_{2j}.$$

Для вершины x_k

$$p_k = \sum_{j=4}^{n} r_{kj}.$$

Общее число пересечений ребер графа

$$P(G) = \sum_{k=2}^{n-2} p_k.$$

Проще определять число пересечений по матрице соединений R. Будем использовать треугольную часть матрицы. Учитывая, что ребра графа не пересекают ребер гамильтонова цикла, в матрице R единицы, соответствующие гамильтонову циклу, заменим на "×".

$R(G)=$

	1	2	3	…	n-1	n
1	0	×	R_{2n}			×
2		0	×			r_{2n}
3			0	×		
...					×	
n-1					0	×
n						0

Определим p_{2n}, для чего в матрице R выделим подматрицу R_{2n}. Сумма единиц подматрицы R_{2n} соответствует p_{2n}. Для $p_{2(n-1)}$ выделим $R_{2(n-1)}$ и т.д. от R_{2n} до $R_{(n-2)n}$. Одновременно строим граф пересечений G'. Ребра графа G, не имеющие пересечений в G' не учитываются. Для определения двудольности G' используем максимальные внутренне устойчивые множества.

8.1.2. Нахождение максимальных внутренне устойчивых множеств

Для нахождения МВУМ при раскраске графа применялся метод Магу. Рассмотрим другой алгоритм, а именно - модифицированный алгоритм *Г.А. Петухова.*

1. В матрице R заменим все диагональные элементы на "1", r_{jj} =1. Положим i=1, α=1;
2. В i-той строке находим элементы r_{ij} = 0 ($j>i$). Номера нулевых элементов помещаем в список $J(j)$. Если все r_{ij} =1, то $\psi_\alpha=\{x_i\}$, $\alpha=\alpha+1$, перейти к п.7;
3. Записываем дизъюнкцию $M_{ij}= r_i \vee r_j$;
4. В строке M_{ij} находим m_k=0 ($k>j$), если все m_k=1, то перейти к п. 6;
5. Записываем дизъюнкцию $M_{ijk}= M_{ij} \vee r_k$; Переходим к п. 4.
6. Если в дизъюнкции нет ни одного нулевого элемента, $\psi_\alpha=\{x_i, x_j, x_k...\}$, $\alpha = \alpha+1$.
   Пока в списке $J(j)$ есть не рассмотренные элементы, выбрать следующий нулевой элемент и перейти к п. 3;

7. Положить $i = i+1$, пока $i<n$ перейти к п.2;
8. Семейство внутренне устойчивых множеств $\Psi_{G'}$ построено.

8.1.3. Выделение из G' максимального двудольного подграфа H'

Введем следующий критерий

$$\alpha_{\gamma\delta}=|\psi_\gamma| + |\psi_\delta| - |\psi_\gamma \cap \psi_\delta|.$$

Отсюда граф пересечений G' двудольный, а соответствующий ему граф G – планарен, если

$$\alpha_{\gamma\delta}=|\psi_\gamma| + |\psi_\delta| = |X'|, \text{ а } |\psi_\gamma \cap \psi_\delta| = \emptyset.$$

Естественно, что чем ближе $\alpha_{\gamma\delta}$ к $|X'|$, тем большее число вершин содержит выделяемый двудольный подграф H'. Для его выделения необходимо определить $\alpha_{\gamma\delta}$ для всех пар $(\psi_\gamma, \psi_\delta) \in \Psi$ и выбрать ψ_γ и ψ_δ с $\max\alpha_{\gamma\delta}$. Максимум удобно искать по матрице $A=||\alpha_{\gamma\delta}||$.

Максимальному H' в исходном графе G соответствует суграф H, содержащий максимальное число непересекающихся ребер. В графе H ребра, вошедшие в одно внутренне устойчивое множество, проводятся внутри гамильтонова цикла, а во второе – вне его.

Из множеств семейства $\Psi_{G'}$ исключаются ребра, вошедшие в ψ_γ и ψ_δ. Одинаковые множества объединяются в одно.

Описанная процедура повторяется до тех пор, пока $\Psi_{G'}$ не станет пусто.

Напомним, что критерий Бадера справедлив для графов, имеющих гамильтонов цикл.

8.2. Нахождение гамильтонова цикла. Алгоритм Робертса-Флореса

Цикл, включающий все вершины один раз, называется гамильтоновым. Не известно общего критерия существования гамильтонова цикла в произвольном графе. Проблема существования гамильтонова цикла принадлежит к классу *NP*-полных.

Алгоритм Робертса-Флореса состоит в следующем. Некоторая начальная вершина (x_1) выбирается в качестве отправной и образует первый элемент множества S, которое будет хранить уже найденные вершины цепи. К S добавляется первая вершина a, смежная с x_1, $a \in \Gamma x_1$. Затем к множеству S добавляется первая возможная вершина b. Под "возможной" вершиной понимается вершина:

1. $b \in \Gamma a$;
2. $b \notin S$.

Существует две причины, препятствующие включению некоторой вершины на шаге r в множество $S = \{x_1, a, b, \ldots, x_r\}$:

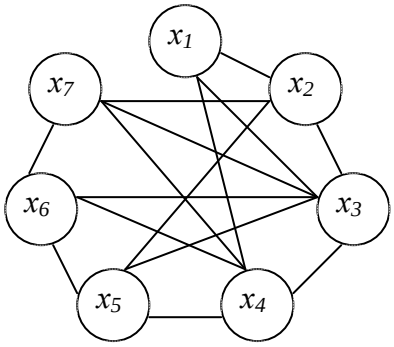
1. у x_r нет "возможной" вершины;
2. цепь S имеет длину $|S|=n$. В этом случае:
   а) есть ребро (x_r, x_1) и гамильтонов цикл найден;
   б) такого ребра нет и найдена гамильтонова цепь.

В случаях 1 и 2 б) следует прибегнуть к возвращению, которое заключается в удалении из S последней включенной вершины x_r и добавлении к S новой первой "возможной", следующей за x_r вершины. Если не существует никакой возможной вершины, делается следующий шаг возвращения и т.д.

Поиск заканчивается в случае когда S состоит из одной x_1 и нет не рассмотренных "возможных" вершин.

Пример. Планаризовать граф $G(X,U)$ (рис.8.2).

8.2.1. Нахождение гамильтонова цикла



$R(G)=$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	0	1	1	1			
x_2	1	0	1		1		1
x_3	1	1	0	1	1	1	1
x_4	1		1	0	1	1	1
x_5		1	1	1	0	1	
x_6			1	1	1	0	1
x_7		1	1	1		1	0

Рисунок 8.2. Граф $G(X,U)$ и его матрица соединений

Включаем в S начальную вершину $S=\{x_1\}$.

Первая "возможная" вершина $x_2 \in \Gamma x_1$, $S=\{x_1, x_2\}$ и т.д. до вершины x_7:

$S=\{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$. Ребра (x_7, x_1) нет. Найдена гамильтонова цепь. Прибегнем к возвращению. Удалим из S вершину x_7.

$S=\{x_1, x_2, x_3, x_4, x_5, x_6\}$.

У x_6 больше нет "возможных" вершин. Удалим и ее. $S=\{x_1, x_2, x_3, x_4, x_5\}$.

У x_5 больше нет "возможных" вершин. Удалим ее. $S=\{x_1, x_2, x_3, x_4\}$. Следующая "возможная" вершина x_6. $S=\{x_1, x_2, x_3, x_4, x_6\}$.

Следующая "возможная" вершина x_5. $S=\{x_1, x_2, x_3, x_4, x_6, x_5\}$.

У x_5 больше нет "возможных" вершин. Удалим ее. $S=\{x_1, x_2, x_3, x_4, x_6\}$. Следующая "возможная" вершина x_7. $S=\{x_1, x_2, x_3, x_4, x_6, x_7\}$.

У x_7 больше нет "возможных" вершин. Удалим из S вершину x_7.

$S=\{x_1, x_2, x_3, x_4, x_6\}$. У x_6 больше нет "возможных" вершин. Удалим ее. $S=\{x_1, x_2, x_3, x_4\}$. Следующая "возможная" вершина x_7. $S=\{x_1, x_2, x_3, x_4, x_7\}$. Следующая "возможная" вершина x_6. $S=\{x_1, x_2, x_3, x_4, x_7, x_6\}$.

Следующая "возможная" вершина x_5. $S=\{x_1, x_2, x_3, x_4, x_7, x_6, x_5\}$.

Ребра (x_5, x_1) нет. Найдена гамильтонова цепь. Удаляем вершины x_4, x_7, x_6, x_5.

$S=\{x_1, x_2, x_3\}$.

И т.д., пока последней не окажется вершина x_4, которая образует цикл с вершиной x_1. Гамильтонов цикл будет

$$S=\{x_1, x_2, x_3, x_5, x_6, x_7, x_4\}.$$

На рис. 8.3 показан полученный гамильтонов цикл.

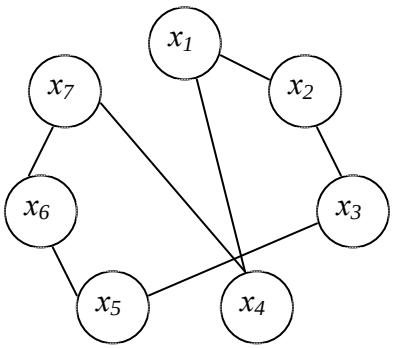


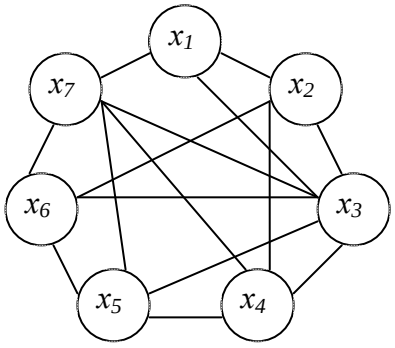
Рисунок 8.3. Гамильтонов цикл графа $G(X,U)$

8.2.2. Построение графа пересечений G'

Перенумеруем вершины графа таким образом, чтобы ребра гамильтонова цикла были внешними.

до перенумерации	x_1	x_2	x_3	x_5	x_6	x_7	x_4
после перенумерации	x_1	x_2	x_3	x_4	x_5	x_6	x_7

Тогда граф $G(X,U)$ будет выглядеть так (рис. 8.4.)



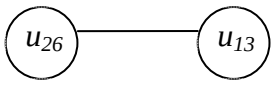
$R(G)=$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	p_i
x_1	0	×	1				×	
x_2		0	×	1		1		2
x_3			0	×	1	1	1	4
x_4				0	×		1	3
x_5					0	×	1	2
x_6						0	×	
x_7							0	

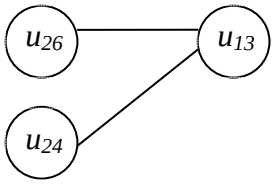
$$\sum p_i = 11$$

Рисунок 8.4. Граф $G(X,U)$
с перенумерованными вершинами и его матрица соединений

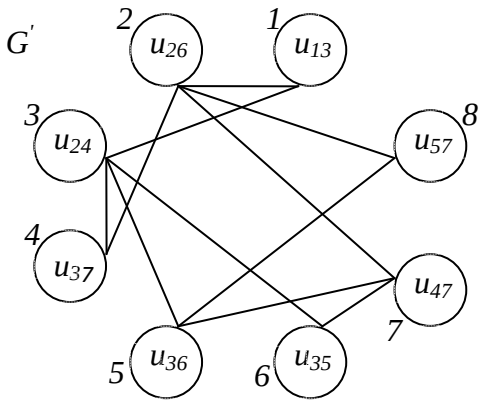
Определим p_{26}, для чего в матрице R выделим подматрицу $R_{26.}$
Ребро (x_2x_6) пересекается с ребром (x_1x_3). Строим граф пересечений G'



Определим p_{24}, для чего в матрице R выделим подматрицу $R_{24.}$
Ребро (x_2x_4) пересекается с ребром (x_1x_3). p_2=2.
Продолжаем строить граф пересечений G'.



После обработки остальных ребер получим граф пересечений G' (рис. 8.5). Число пересечений ребер графа $P(G) = p_2 + p_3 + p_4 + p_5 = 11$.



$R(G')=$

	1	2	3	4	5	6	7	8
1	1	1	1					
2	1	1		1			1	1
3	1		1	1	1	1		
4		1	1	1				
5			1		1		1	1
6			1			1	1	
7		1			1	1	1	
8		1			1			1

Рисунок 8.5. Граф пересечений G' и его матрица соединений

8.2.3. Построение семейства $\Psi_{G'}$

В первой строке матрицы $R(G')$ находим номера нулевых элементов. Составляем список $J(j) = \{4, 5, 6, 7, 8\}$.

Для первого нулевого элемента составляем дизъюнкцию

$$M_{14} = r_1 \vee r_4 = \{11110000\}.$$

В строке M_{14} находим номера нулевых элементов. Составляем список $J'(j') = \{5, 6, 7, 8\}$.

Записываем дизъюнкцию

$$M_{145} = M_{14} \vee r_5 = \{11111011\}.$$

В строке M_{145} находим $m_6=0$. Записываем дизъюнкцию

$$M_{1456} = M_{145} \vee r_6 = \{11111111\}.$$

В строке M_{1456} все «1».

Построено $\boldsymbol{\psi_1 = \{u_{13}, u_{37}, u_{36}, u_{35}\}}$.

Из списка $J'(j')$ выбираем следующий элемент. Записываем дизъюнкцию

$$M_{146} = M_{14} \vee r_6 = \{11110110\}.$$

В строке M_{146} находим $m_8=0$. Записываем дизъюнкцию

$$M_{1468} = M_{146} \vee r_8 = \{11111111\}.$$

В строке M_{1468} все «1».

Построено $\boldsymbol{\psi_2 = \{u_{13}, u_{37}, u_{35}, u_{57}\}}$.

Из списка $J'(j')$ выбираем следующий элемент. Записываем дизъюнкцию

$$M_{147} = M_{14} \vee r_7 = \{11111110\}.$$

И, наконец, $M_{1478} = M_{147} \vee r_8 = \{11111111\}$.

В строке M_{1478} все «1».

Построено $\boldsymbol{\psi_3 = \{u_{13}, u_{37}, u_{47}, u_{57}\}}$.

Из списка $J'(j')$ выбираем следующий элемент. Записываем дизъюнкцию

$$M_{148} = M_{14} \vee r_8 = \{11110001\}.$$

В строке остались незакрытые «0».

Из списка $J(j)$ выбираем следующий нулевой элемент r_5. Составляем дизъюнкцию

$$M_{15}= r_1 \vee r_5=\{11101011\}.$$

В строке M_{15} находим нулевой элемент – m_6=0. Записываем дизъюнкцию

$$M_{156}= M_{15} \vee r_6= \{11101111\}.$$

В строке остался незакрытый «0». Понятно, что «0» в четвертой позиции элементами с номерами $j > 4$ закрыть не удастся.

Во второй строке ищем первый нулевой элемент – r_{23}. Составляем дизъюнкцию

$$M_{23}= r_2 \vee r_3= \{11111111\}.$$

В строке M_{23} все «1».

Построено $\boldsymbol{\psi_4=\{u_{26}, u_{24}\}}$.

Во второй строке ищем следующий нулевой элемент – r_{25}. Составляем дизъюнкцию

$$M_{25}= r_2 \vee r_5= \{11111011\}.$$

И, наконец, $M_{256}= M_{25} \vee r_6= \{11111111\}$.

Построено $\boldsymbol{\psi_5=\{u_{26}, u_{36}, u_{35}\}}$.

Во второй строке ищем следующий нулевой элемент – r_{26}. Составляем дизъюнкцию

$$M_{26}= r_2 \vee r_6= \{11110111\}.$$

В строке остался незакрытый «0».

В третьей строке ищем первый нулевой элемент – r_7. Составляем дизъюнкцию

$$M_{37}= r_3 \vee r_7= \{11111110\}.$$

И, наконец, $M_{378}= M_{37}$ v $r_8= \{11111111\}$.

Построено $\boldsymbol{\psi_6=\{u_{24}, u_{47}, u_{57}\}}$.

В третьей строке ищем следующий нулевой элемент – r_{38}. Составляем дизъюнкцию

$$M_{38}= r_3 \vee r_8= \{1111101\}.$$

В строке остался незакрытый «0».

Из матрицы $R(G')$ видно, что строки с номерами $j > 3$ «0» в первой позиции закрыть не смогут.

Семейство максимальных внутренне устойчивых множеств $\Psi_{G'}$ построено. Это

$\boldsymbol{\psi_1=\{u_{13}, u_{37}, u_{36}, u_{35}\}}$;
$\boldsymbol{\psi_2=\{u_{13}, u_{37}, u_{35}, u_{57}\}}$;
$\boldsymbol{\psi_3=\{u_{13}, u_{37}, u_{47}, u_{57}\}}$;
$\boldsymbol{\psi_4=\{u_{26}, u_{24}\}}$;
$\boldsymbol{\psi_5=\{u_{26}, u_{36}, u_{35}\}}$;
$\boldsymbol{\psi_6=\{u_{24}, u_{47}, u_{57}\}}$.

Для каждой пары множеств вычислим значение критерия

$$\alpha_{\gamma\delta}=|\psi_\gamma| + |\psi_\delta| \quad |\psi_\gamma \cap \psi_\delta|.$$

Результаты вычислений запишем в матрицу $A= ||\alpha_{\gamma\delta}||$.

$$\alpha_{12}=|\psi_1| + |\psi_2| \quad |\psi_1 \cap \psi_2| =4+4-3=5.$$

$\alpha_{13}=|\psi_1| + |\psi_3| \quad |\psi_1 \cap \psi_3| = 4+4-2=6$ и т.д.

$A=$

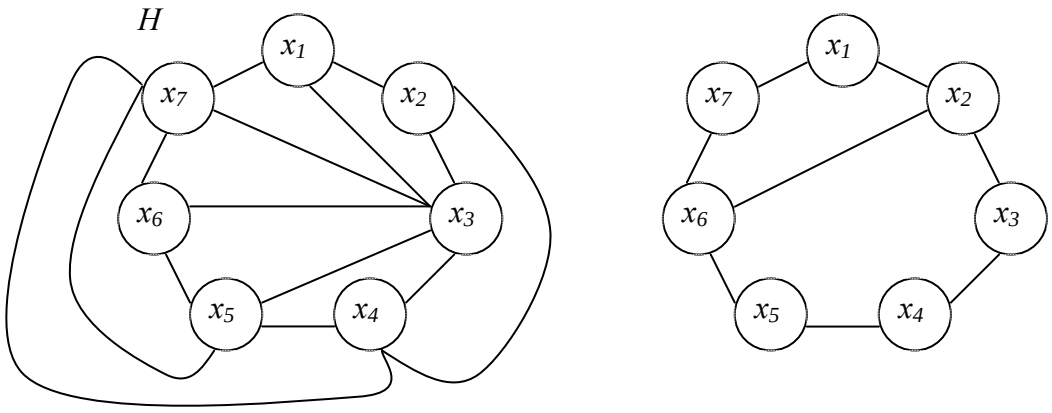
	1	2	3	4	5	6
1	0	5	6	6	5	***7***
2		0	5	6	6	6
3			0	6	***7***	5
4				0	4	4
5					0	6
6						0

$\max\alpha_{\gamma\delta} = \alpha_{16} = \alpha_{35} = 7$, дают две пары множеств ψ_1, ψ_6 и ψ_3, ψ_5.

Возьмем множества

$\boldsymbol{\psi_1 = \{u_{13}, u_{37}, u_{36}, u_{35}\}}$ и $\boldsymbol{\psi_6 = \{u_{24}, u_{47}, u_{57}\}}$.

В суграфе H, содержащем максимальное число непересекающихся ребер, ребра, вошедшие в $\boldsymbol{\psi_1}$, проводим внутри гамильтонова цикла, а в $\boldsymbol{\psi_6}$ – вне его (рис. 8.6 (*а*)).



а *б*

Рисунок 8.6. Планарные суграфы

Удалим из $\Psi_{G'}$ ребра, вошедшие в ψ_1 и ψ_6

$\boldsymbol{\psi_4 = \{u_{26}\}}$; $\boldsymbol{\psi_5 = \{u_{26}\}}$.

Объединим одинаковые множества, не реализованным осталось единственное ребро $\boldsymbol{\psi_4 = \{u_{26}\}}$.

Проведем его (рис. 8.6 (*б*)). Все ребра графа G реализованы. Толщина графа $m = 2$.

Если взять множества $\boldsymbol{\psi_3 = \{u_{13}, u_{37}, u_{47}, u_{57}\}}$ и $\boldsymbol{\psi_5 = \{u_{26}, u_{36}, u_{35}\}}$, то не реализованным окажется ребро $\boldsymbol{\{u_{24}\}}$.

9. Верификация. Основные понятия

9.1. Место верификации при проектировании вычислительных систем

Согласно закону Мэрфи: Если какая-нибудь неприятность может произойти, она случается. Человек может ошибиться, поэтому он обязательно ошибается. Ошибки случаются при проектировании аппаратуры, при написании программ, при изготовлении аппаратуры. Иногда ошибки обходятся очень дорого. Приведем некоторые примеры последствий, к которым приводят такие ошибки.

В 1994 году профессор математики из Вирджинии при вычислении обратных величин простых чисел обнаружил, что микропроцессор Pentium в некоторых случаях неправильно делит числа с плавающей точкой. Через месяц фирма Intel согласилась заменить микропроцессоры с неправильно спроектированным устройством деления, что обошлось ей в 300 миллионов долларов (итоговые потери корпорации составили 500 миллионов долларов). Этот случай нанес большой ущерб репутации фирмы.

В 1991 году во время Войны в заливе (США против Ирака) батарея американских зенитных ракет «Patriot» не смогла перехватить запущенную иракцами ракету «Scud» советского производства. Ракета попала в казарму американских солдат, при этом погибло 28 человек. Причиной этого была погрешность вычисления времени. При вычислении нужно было умножать время, задаваемое тактовым генератором компьютера (оно измерялось в десятых долях секунды) на 1/10, но это десятичное число невозможно точно представить в двоичном виде. Для хранения этой константы использовался 24-разрядный регистр. Разница между точным значением 1/10 и ее неточным представлением составляет около 0,000000095 в десятичном виде. Компьютер был включен около 100 часов, за это время накопилась ошибка в измерении времени в 0,34 секунды. Скорость ракеты «Scud» составляла примерно 1700 м/сек, то есть за это время она прошла более 500 метров. Этого хватило для того, чтобы зенитные ракеты не смогли ее перехватить.

Слово *верификация* произошло от латинского *verus* - истинный и *facere* - делать. Вообще, верификация означает подтверждение того, что описание проекта полностью соответствует спецификации (техническому заданию) проектируемой системы. Спецификация — документ, подробно перечисляющий условия, которым должна соответствовать изготовляемая или проектируемая система.

Верификация определяется как разновидность анализа, имеющая целью установление соответствия двух описаний одного и того же объекта. Различают верификацию структурную и функциональную. При структурной верификации устанавливается соответствие структур, отображаемых двумя описаниями, при функциональной (параметрической) - проверяется соответствие процессов функционирования и выходных параметров, отображаемых сравниваемыми описаниями. Структурная верификация связана с меньшими затратами вычислительных ресурсов, чем функциональная. Поэтому последняя часто выполняется не в полном объеме и после того, как проверено соответствие структурных

свойств. Примером структурной верификации служит установление соответствия системы электрических межсоединений на печатной плате и в принципиальной электрической схеме, заданных своими топологическими моделями в виде графов. Верификация в этом случае сводится к установлению *изоморфизма графов*. Функциональная верификация в данном примере выполняется путем анализа переходных процессов с учетом перекрестных помех и задержек сигналов в длинных линиях, определяемых конструктивным исполнением электронного блока.

9.2. Изоморфизм графов

Вопрос о том, изоморфны ли два данных графа, в общем случае оказывается сложным.

Для изоморфизма двух n-вершинных графов теоретически безукоризненный способ проверки состоит в проверке всех $n!$ взаимно однозначных соответствий между множествами вершин и установлении, совмещаются ли полностью ребра графа хотя бы при одном соответствии. Однако даже весьма грубая оценка показывает, что такое решение «в лоб» практически непригодно: уже при n=20 перебор всех $n!$ вариантов потребовал бы 40 лет машинного времени.

Подобная ситуация, естественно, толкнула многих математиков на попытки найти такой инвариант (число или систему чисел), который бы, с одной стороны, легко вычислялся по заданному графу, а с другой – обладал свойством полноты, т.е. определял граф однозначно с точностью до изоморфизма.

Инвариантом графа $G(X, U)$ называется параметр, имеющий одно и то же значение для всех графов, изоморфных графу G.

Вначале естественно поставить вопрос: какие характеристики графов инвариантны относительно изоморфизма?

Среди самых очевидных инвариантов отметим следующие:

1. Число вершин $|X| = m$.
2. Число ребер $|U| = k$.
3. Число компонент связности $p(G)$.
4. Последовательность степеней вершин, т.е. список степеней всех вершин в невозрастающем порядке значений ρx_i.
5. Плотность $f(G)$ – число вершин клики графа G.
6. Число внутренней устойчивости $\alpha(G)$.
7. Хроматическое число $\chi(G)$.
8. Число Хадвигера $\eta(G)$.

Эвристики для решения задач изоморфизма обычно состоят в попытках показать, что два рассматриваемые графа не изоморфны. Для этого составляется список различных инвариантов в порядке, определяемом сложностью вычисления инварианта. Затем последовательно сравниваются значения параметров графов. Как только обнаруживаются два различных значения одного и того же параметра, приходят к заключению, что данные графы не изоморфны.

Множество инвариантов, которое позволило бы этой процедуре установить изоморфность графов за полиномиальное время, называется *кодом* графа. К сожалению, на сегодняшний день такое множество не найдено.

По существу, эвристический алгоритм рассматриваемого типа сводится к сравнению неполных кодов двух графов. Конечно, рассмотрение большого числа инвариантов увеличивает вероятность правильного заключения об изоморфизме при совпадении всех значений параметров, но в общем случае ничего не гарантирует.

Пример. Для графов на рис. 9.1 определим значения инвариантов.

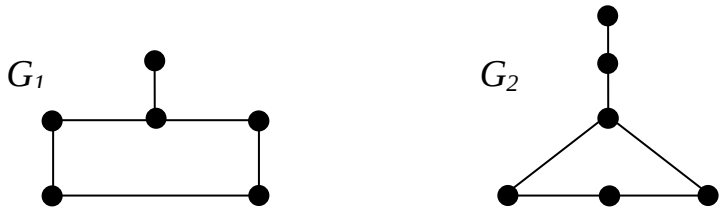


Рисунок 9.1. Графы G_1 и G_2

Инвариант	G_1	G_2	Совпадение
m	6	6	+
k	6	6	+
$p(G)$	1	1	+
Список ρx	3, 2, 2, 2, 2, 1	3, 2, 2, 2, 2, 1	+
$f(G)$	2	2	+
$\alpha(G)$	3	3	+
$\chi(G)$	2	3	-
$\eta(G)$	3	3	+

Хроматические числа рассматриваемых графов разные. Это позволяет сделать вывод, что графы G_1 и G_2 не изоморфны.

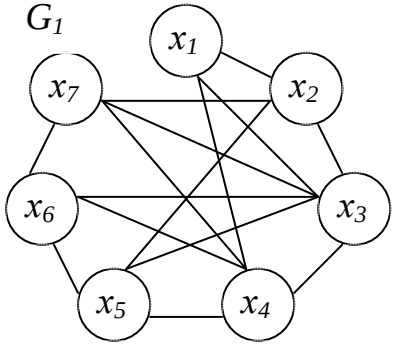
Для изоморфизма графов ***необходимо*** совпадение инвариантов, однако ***достаточным*** это условие не является.

Первые четыре инварианта относятся к группе «легко вычислимых». Наиболее «богатый» из них – упорядоченный список степеней вершин. Не будучи идеальным средством распознавания изоморфизма, список может, тем не менее, во многих случаях оказать существенную помощь.

Во-первых, если списки не совпадают, то отсюда сразу следует не изоморфизм графов G_1 и G_2.

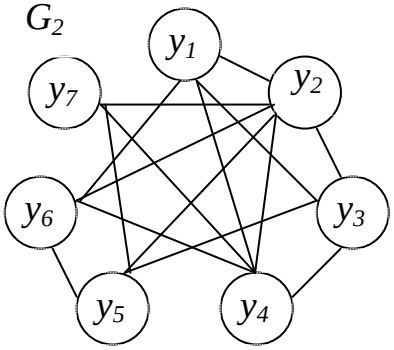
Во-вторых, если списки совпали, то для проверки графов G_1 и G_2 на изоморфизм требуется перебор не всех $n!$ соответствий между вершинами, а лишь тех, при которых сопоставляются вершины с одинаковой степенью. Так, в рассмотренном примере надо перебрать лишь $4! = 24$ соответствия вместо $6! = 720$.

Пример. Проверить на изоморфизм графы G_1 и G_2 (рис. 9.2)



$R(G_1)=$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\rho(x)$
x_1	0	1	1	1				3
x_2	1	0	1		1		1	4
x_3	1	1	0	1	1	1	1	6
x_4	1		1	0	1	1	1	5
x_5		1	1	1	0	1		4
x_6			1	1	1	0	1	4
x_7		1	1	1		1	0	4



$R(G_2)=$

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	$\rho(y)$
y_1	0	1	1	1		1		4
y_2	1	0	1	1	1	1	1	6
y_3	1	1	0	1	1			4
y_4	1	1	1	0		1	1	5
y_5		1	1		0	1	1	4
y_6	1	1		1	1	0		4
y_7		1		1	1		0	3

Рисунок 9.2. Графы G_1 и G_2 и их матрицы соединений

Для графа G_1 $\Sigma\rho(x)=30$. Список $P(x) = \{6, 5, 4, 4, 4, 4, 3\}$.
Для графа G_2 $\Sigma\rho(y)=30$. Список $P(y) = \{6, 5, 4, 4, 4, 4, 3\}$.

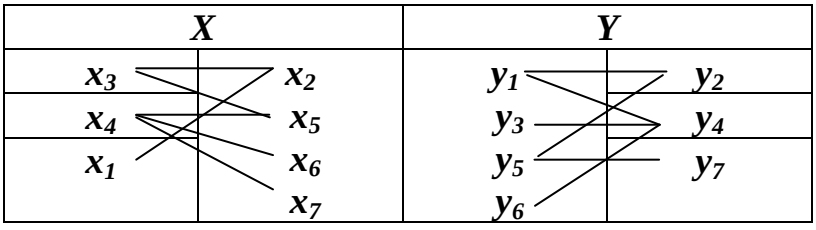
Разобьем вершины обоих графов на классы по их степеням.

	$\rho(x)=\rho(y)=6$	$\rho(x)=\rho(y)=5$	$\rho(x)=\rho(y)=4$	$\rho(x)=\rho(y)=3$
X	x_3	x_4	x_2, x_5, x_6, x_7	x_1
Y	y_2	y_4	y_1, y_3, y_5, y_6	y_7

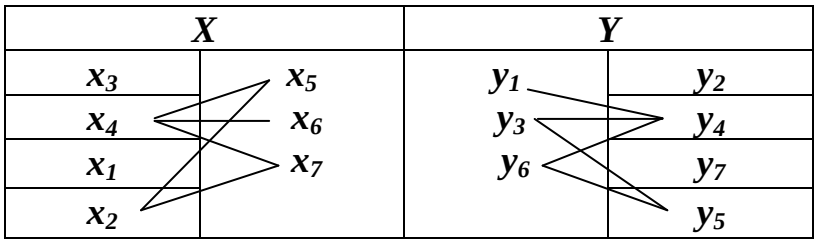
Из таблицы сразу видно соответствие вершин графов:

$\boldsymbol{X}$	$\boldsymbol{Y}$
$\boldsymbol{x_3}$	$\boldsymbol{y_2}$
$\boldsymbol{x_4}$	$\boldsymbol{y_4}$
$\boldsymbol{x_1}$	$\boldsymbol{y_7}$

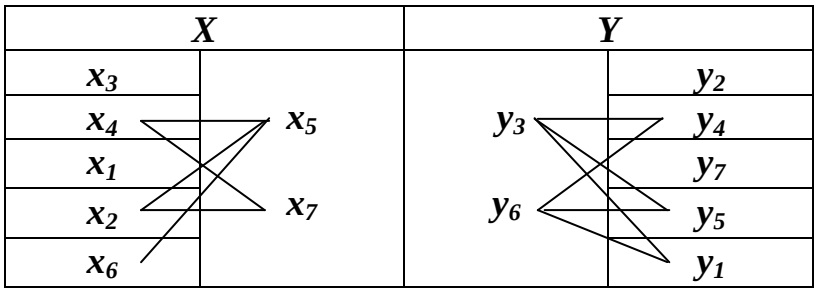
Для определения соответствия вершин с $\rho(x)=\rho(y)=4$ попробуем связать вершины из классов с $\rho(x)=\rho(y)=5$ и $\rho(x)=\rho(y)=3$ с неустановленными вершинами.



Анализ связей вершин показывает соответствие вершин $\boldsymbol{x_2}$ и $\boldsymbol{y_5}$ (соединены с установленными вершинами $x_3 \sim y_2$ и $x_1 \sim y_7$. С учетом этого



Анализ связей вершин показывает соответствие вершин x_6 и y_1 (единственные имеют одну связь с классом вершин $\rho(x)= \rho(y)=5$. С учетом этого



Анализ связей вершин показывает, что существует две пары соответствий оставшихся вершин: вершины x_5 и y_3 и вершины x_7 и y_6, или x_6 и y_3 и вершины x_7 и y_3. Это соответствует действительности, т.к. вершины x_5 и x_7 в графе G_1 и вершины y_3 и y_6 в графе G_2 смежны с одними и теми же вершинами.

Из сказанного можно сделать вывод, что графы G_1 и G_2 изоморфны.

10. Нахождение эйлерова цикла
10.1. Алгоритм Флери

Элегантный алгоритм нахождения эйлерова цикла был предложен Флери в 1883 году.

Алгоритм заключается в следующем:

1. Положить текущий граф равным $G(X, U)$, а текущую вершину — равной произвольной вершине $x_i \in X$.

2. Выбрать произвольное ребро u_{ij} текущего графа, инцидентное текущей вершине x_i с учетом следующего ограничения: если степень текущей вершины в текущем графе больше 1, нельзя выбирать ребро, удаление которого из текущего графа увеличит число компонент связности в нем (т.е. ребро, являющееся мостом).

3. Назначить текущей x_j вершину, инцидентную ребру u_{ij}.

4. Удалить u_{ij} из текущего графа и внести в список.

5. Если в текущем графе еще остались ребра, то положить $i=j$ и вернуться на шаг 2.

Пример. Рассмотрим граф, изображенный на рис. 10.1 (*а*) (он эйлеров в силу теоремы Эйлера о циклах) и найдем в нем эйлеров цикл.

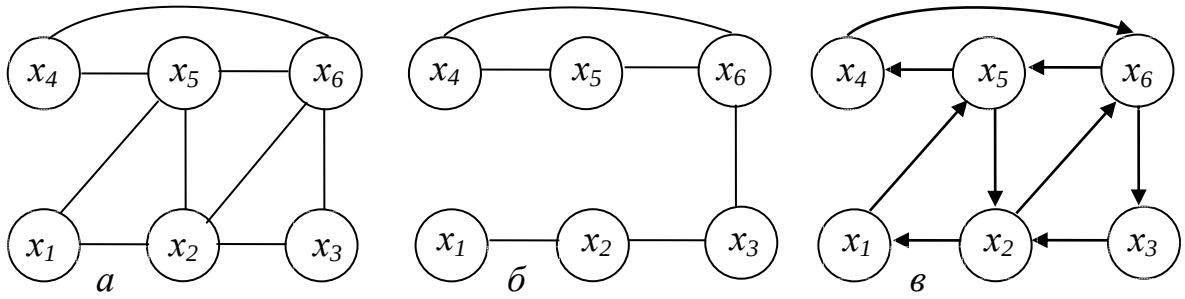


Рисунок 10.1. Исходный граф (*а*), текущий граф (*б*) и эйлеров цикл (*в*)

Пусть на шаге 1 выбрана вершина x_1. При выборе на шаге 2 ограничение никак не сказывается; пусть выбрано ребро (x_1, x_5). На двух следующих итерациях ограничений на выбор по-прежнему не возникает; пусть выбраны ребра (x_5, x_2) и (x_2, x_6). Тогда текущим графом становится граф, изображенный на рис. 10.1 (*б*) (текущая вершина — x_6). На следующей итерации нельзя выбрать ребро (x_6, x_3) из-за ограничения; пусть выбрано ребро (x_6, x_5). Дальнейший выбор ребер определен однозначно (текущая вершина всегда будет иметь степень 1), так что в итоге будет построен следующий эйлеров цикл (рис. 10.1 (*в*)): $x_1 \to x_5 \to x_2 \to x_6 \to x_5 \to x_4 \to x_6 \to x_3 \to x_2 \to x_1$.

Сложность алгоритма $O(k)$, где $k = |U|$ - число ребер.

10.2. Сравнение эйлеровых и гамильтоновых циклов

Несмотря на внешнюю схожесть определений эйлерова и гамильтонова циклов, задачи нахождения циклов этих двух типов в данном графе разительно отличаются по сложности. Задача о нахождении эйлерова цикла — это простая с математической точки зрения задача: существует эффективный критерий существования эйлерова цикла (теорема Эйлера); если критерий выполнен, имеется эффективный алгоритм для нахождения цикла (например, алгоритм Флери). «Эффективный» в данном случае означает «требующий проведения небольшого числа операций относительно размера графа». Ни критерия гамильтоновости графа, ни эффективного алгоритма нахождения гамильтонова цикла в произвольном гамильтоновом графе, неизвестно (и скорее всего, не существует). Задача о нахождении гамильтонова цикла — это NP-трудная задача.

Следующие четыре графа демонстрируют отсутствие тесной взаимосвязи между существованием эйлеровых и гамильтоновых циклов (рис. 10.2).

Однако, двойственность между эйлеровыми и гамильтоновыми циклами (замена вершины на ребро и наоборот) приводит к тесной связи между этими двумя понятиями в применении к графу G и соответствующему ему реберному графу, определяемому ниже.

Реберный граф G_l графа G имеет столько же вершин, сколько ребер у графа G. Ребро между двумя вершинами графа G_l существует тогда и только тогда, когда ребра графа G, соответствующие этим двум вершинам, смежны (т.е. инцидентны одной и той же вершине графа G).

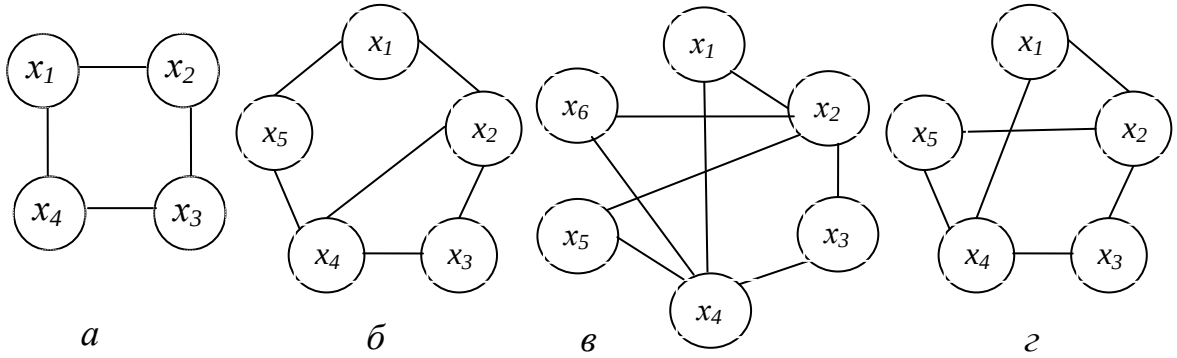


Рисунок 10.2. Графы: эйлеров и гамильтонов (*а*); неэйлеров и гамильтонов (*б*); эйлеров и негамильтонов (*в*); неэйлеров и негамильтонов (*г*)

Верны два следующих утверждения о взаимоотношении между эйлеровыми и гамильтоновыми циклами, принадлежащие Ф. Харари.

1. Если граф G имеет эйлеров цикл, то граф G_l имеет как эйлеров, так и гамильтонов циклы.

2. Если граф G имеет гамильтонов цикл, то граф G_l также имеет гамильтонов цикл.

Обращение этих утверждений неверно!

11. Эволюционные алгоритмы оптимизации

11.1. Генетические алгоритмы

Природа поражает своей сложностью и богатством проявлений. Среди примеров можно назвать сложные социальные системы, иммунные и нейронные системы, сложные взаимосвязи между видами. Они - всего лишь некоторые из чудес, ставшие очевидными при глубоком исследовании природы вокруг нас. Наука - это одна из систем, которая объясняет окружающее и помогает приспособиться к новой информации, получаемой из внешней среды. Многое из того, что мы видим и наблюдаем, можно объяснить теорией эволюции через наследственность, мутацию и отбор.

Поэтому не удивительно, что ученые, занимающиеся компьютерными исследованиями, обратились к теории эволюции. Возможность того, что вычислительная система, наделенная простыми механизмами изменчивости и отбора, могла бы функционировать по аналогии с законами эволюции в естественных системах, была очень привлекательной. Эта надежда является причиной появления ряда вычислительных систем, построенных на принципах естественного отбора.

Ключевую роль в эволюционной теории играет естественный отбор. Его суть состоит в том, что наиболее приспособленные особи лучше выживают и приносят больше потомков, чем менее приспособленные. Заметим, что сам по себе естественный отбор еще не обеспечивает развития биологического вида. Поэтому очень важно понять, каким образом происходит наследование, то есть, как свойства потомка зависят от свойств родителей.

Основной закон наследования интуитивно понятен каждому - он состоит в том, что потомки похожи на родителей. В частности, потомки более приспособ-

ленных родителей будут, скорее всего, одними из наиболее приспособленных в своем поколении. Чтобы понять, на чем основано это сходство, нужно немного углубиться в построение естественной клетки - в мир генов и хромосом.

Почти в каждой клетке любой особи есть набор хромосом, несущих информацию про эту особь. Основная часть хромосомы - нить ДНК, определяющая, какие химические реакции будут происходить в данной клетке, как она будет развиваться и какие функции выполнять.

Ген - это отрезок цепи ДНК, ответственный за определенное свойство особи, например за цвет глаз, тип волос, цвет кожи и т.д. Вся совокупность генетических признаков человека кодируется с помощью приблизительно 60 тыс. генов, длина которых составляет более 90 млн. нуклеотидов.

Вообразим себе искусственный мир, населенный множеством существ (особей), причем каждая особь - это некоторое решение задачи. Будем считать особь более приспособленной, чем лучше соответствующее решение (чем большее значение целевой функции оно дает). Тогда задача максимизации целевой функции сводится к поиску более приспособленной особи. Конечно, мы не можем поселить в наш виртуальный мир все особи сразу, так как их очень много. Вместо этого будем рассматривать множество поколений, которые сменяют друг друга. Теперь, если мы сумеем задействовать естественный отбор и генетическое наследование, полученная среда будет подчиняться законам эволюции. Целью этой искусственной эволюции будет создание наилучших решений. Очевидно, эволюция - бесконечный процесс, в ходе которого приспособленность особей постепенно повышается. Принудительно остановив этот процесс через длительное время после его начала и выбрав наиболее приспособленную особь в текущем поколении, получим не абсолютно точный, но близкий к оптимальному ответ. Такова идея генетического алгоритма.

Для того чтобы говорить о генетическом наследовании, нужно наделить наши особи хромосомами. В генетическом алгоритме хромосома - это некоторый числовой вектор, который отвечает подбираемому параметру, а набор хромосом данной особи определяет решение задачи. Какие именно векторы следует рассматривать в конкретной задаче, решает сам пользователь. Каждая из позиций вектора хромосомы называется геном.

Простой генетический алгоритм случайным образом генерирует начальную популяцию. Работа генетического алгоритма представляет итерационный процесс, который продолжается до тех пор, пока не выполнится заданное число поколений или любой другой критерий остановки. В каждом поколении генетического алгоритма реализуется отбор пропорциональной приспособленности, одноточечный кроссинговер и мутация. Сначала, пропорциональный отбор назначает каждой структуре вероятность $P_s(i)$ равную отношению ее приспособленности к суммарной приспособленности популяции:

$$P_s(i) = \frac{f(i)}{\sum_{i=1}^{n} f(i)}.$$

Потом происходит отбор (с замещением) всех n особей для дальнейшей генетической обработки, соответственно величине $P_s(i)$.

При таком отборе члены популяции с высокой приспособленностью с большей вероятностью будут выбираться чаще, чем особи с низкой приспособленностью. После отбора, n избранных особей случайным образом разбиваются на $n/2$ пары. Для каждой пары с вероятностью P_s может применяться кроссинговер. Соответственно, с вероятностью (1 - P_s) кроссинговер не происходит и неизмененные особи переходят на стадию мутации. Если кроссинговер происходит, полученные потомки заменяют родителей и переходят к мутации.

Определим теперь понятия, отвечающие мутации и кроссинговеру в генетическом алгоритме.

Мутация - это преобразование хромосомы, которое случайно изменяет одну или несколько ее позиций (генов). Наиболее распространенный вид мутаций - случайное изменение только одного из генов хромосомы.

Кроссинговер (в литературе по генетическим алгоритмам также употребляется название кроссовер или скрещивание) - это операция, при которой из двух хромосом порождается одна или несколько новых хромосом. Одноточечный кроссинговер работает следующим образом.

Сначала, случайным образом выбирается одна из точек разрыва (точка разрыва - участок между соседними битами в строке.) Обе родительские структуры в этой точке разрываются на два сегмента. Потом, соответствующие сегменты разных родителей склеиваются и выходят два генотипа потомков.

Например, предположим, один родитель состоит из 10 нулей, а другой - из 10 единиц. Пусть из 9 возможных точек разрыва выбрана точка 3. Родители и их потомки показаны ниже.

Кроссинговер

Родитель 1	000~0000000
Родитель 2	111~1111111
Потомок 1	111~0000000
Потомок 2	000~1111111

После того как заканчивается стадия кроссинговера, выполняются операторы мутации. В строке, к которой применяется мутация, каждый бит с вероятностью P_m изменяется на противоположный. Популяция, полученная после мутации, записывается поверх старой и цикл одного поколения завершается. Следующие поколения обрабатываются подобным образом: отбор, кроссинговер и мутация.

В настоящее время исследователи генов предлагают другие операторы отбора, кроссинговера и мутации. Ниже приведены наиболее распространенные.

Элитные методы отбора гарантируют, что при отборе обязательно будут выживать лучший или лучшие члены популяции. Наиболее распространена процедура обязательного сохранения только одной лучшей особи, если она не прошла как другие через процесс отбора, кроссинговера и мутации. Элитизм может быть введен практически в любой стандартный метод отбора.

Двухточечный кроссинговер и равномерный кроссинговер - достойные альтернативы одноточечному оператору. В двухточечном кроссинговере выбираются две точки разрыва, и родительские хромосомы обмениваются сегментом, находящемся между этими точками. В равномерном кроссинговере, каждый бит первого родителя наследуется первым потомком с заданной вероятностью, в противном случае этот бит передается второму потомку. И наоборот.

Схема генетического алгоритма изображена на рис. 11.1. Сначала генерируется начальная популяция особей (индивидуумов), то есть некоторый набор решений задачи. Как правило, это делается случайным образом. Потом моделируется размножение внутри популяции. Для этого, случайно отбирается несколько пар индивидуумов, происходит скрещивание между хромосомами в каждой паре, а полученные новые хромосомы переходят в популяцию нового поколения. В генетическом алгоритме сохраняется основной принцип естественного отбора - чем приспособленней индивидуум (чем больше соответствующее ему значение целевой функции), тем с большей вероятностью он будет участвовать в скрещивании. Теперь моделируются мутации - в нескольких случайно избранных особях нового поколения изменяются некоторые гены. Старая популяция частично или целиком уничтожается и рассматривается следующее поколение. Популяция следующего поколения в большинстве реализаций генетических алгоритмов содержит столько же особей, сколько начальная, но в силу отбора приспособленность в ней в среднем выше. Теперь описанные процессы отбора, скрещивания и мутации повторяются уже для этой популяции и т.д.

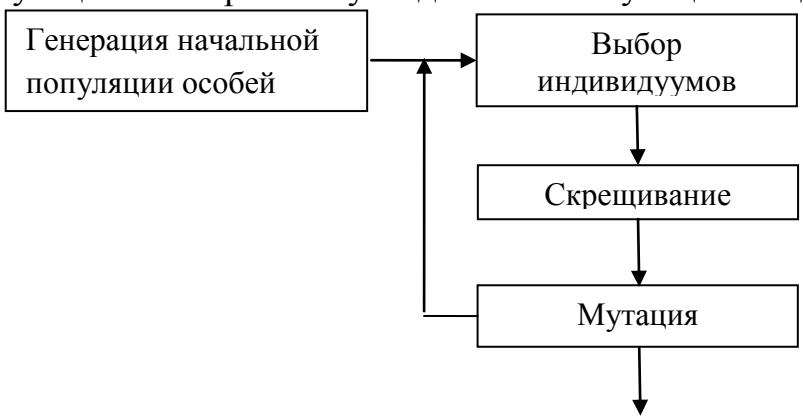


Рисунок 11.1. Схема генетического алгоритма

В каждом следующем поколении наблюдается возникновение новых решений задачи. Среди них будут как плохие, так и хорошие, но благодаря отбору, число приемлемых решений будет возрастать. Заметим, что в природе не бывает абсолютных гарантий, и приспособленный тигр может погибнуть от ружейного выстрела, не оставив потомков. Имитируя эволюцию на компьютере, можно избежать подобных нежелательных событий и всегда сохранять жизнь лучшему из индивидуумов текущего поколения - такая методика называется "стратегией элитизма".

Генетические алгоритмы в разных формах применяются для решения многих научных и технических проблем. Генетические алгоритмы используются при создании других вычислительных структур, например, автоматов или сетей сортировки. В машинном обучении они используются при проектировании ней-

ронных сетей или управлении роботами. Они также применяются при моделировании развития в разных предметных областях, включая биологические (экология, иммунология и популярная генетика) и социальные (такие как экономика и политика) системы.

Тем не менее, возможно популярное применение генетических алгоритмов - оптимизация многопараметрических функций. Большинство реальных задач могут быть сформулированы как поиск оптимального значения, где значение - сложная функция, зависящая от определенных входных параметров. В некоторых случаях, нужно найти те значения параметров, при которых достигается точное значение функции. В других случаях, точный оптимум не нужен - решением может считаться любое значение, лучшее по определенной заданной величине. В этом случае, генетические алгоритмы - приемлемый метод для поиска "приемлемых" значений. Сила генетического алгоритма состоит в его способности манипулировать одновременно многими параметрами, которая используется в сотнях прикладных программ, включая проектирование самолетов, СБИС, настраивании параметров алгоритмов и поиске стойких состояний систем нелинейных дифференциальных уравнений.

Генетические алгоритмы являются эффективной процедурой поиска, которая конкурирует с другими процедурами. Эффективность генетических алгоритмов сильно зависит от таких деталей, как метод кодирования решений, операторов, настраивания параметров, отдельных критериев успеха.

11.2. Биоинспирированные методы в оптимизации

11.2.1. Муравьиные методы и алгоритмы

Одним из биоинспирированных подходов является метод роевого интеллекта включающий в себя муравьиные алгоритмы, пчелиные алгоритмы, метод роя частиц, алгоритм капель воды и др.

Роевой интеллект описывает коллективное поведение децентрализованной самоорганизующейся системы, которая состоит из множества агентов, локально взаимодействующих между собой и с окружающей средой. Агенты обычно довольно просты, но, локально взаимодействуя, вместе создают, так называемый, роевой интеллект.

Муравьи относятся к социальным насекомым, живущим внутри некоторого коллектива – колонии. Основу «социального» поведения муравьев составляет самоорганизация – множество динамических механизмов, обеспечивающих достижение системой глобальной цели в результате низкоуровневого взаимодействия ее элементов. Принципиальной особенностью такого взаимодействия является использование элементами системы только локальной информации. При этом исключается любое централизованное управление и обращение к глобальному образу, репрезентирующему систему во внешнем мире. Самоорганизация является результатом взаимодействия следующих четырех компонентов: случайность; многократность; положительная обратная связь; отрицательная обратная связь.

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученного из предыдущих решений.

В сравнении с генетическими алгоритмами муравьиные алгоритмы имеют некоторые преимущества: опираются на память всей колонии вместо памяти только о предыдущем поколении и меньше подвержены неоптимальным начальным решениям (из-за случайного выбора пути и памяти колонии).

Ряд экспериментов показывает, что эффективность муравьиных алгоритмов растёт с ростом размерности решаемых задач оптимизации. Хорошие результаты получаются для нестационарных систем с изменяемыми во времени параметрами.

Важным свойством муравьиных алгоритмов является неконвергентность: даже после большого числа итераций одновременно исследуется множество вариантов решения, вследствие чего, не происходит длительных временных задержек в локальных экстремумах. Перспективными путями улучшения муравьиных алгоритмов являются on-line адаптация параметров с помощью базы нечетких правил, а также их гибридизация с другими методами природных вычислений, например генетическими алгоритмами. Гибридизация может осуществляться по островной схеме, когда различные алгоритмы решают задачу параллельно и автономно, или по принципу «мастер-подмастерье», когда основной алгоритм − «мастер» передает решение типовых подзадач «подмастерью».

11.2.2. Пчелиные методы и алгоритмы

Пчелиный алгоритм − это оптимизационный алгоритм, в основе которого лежит поведение пчёл в живой природе.

Применительно к задаче оптимизации в пчелином алгоритме каждое решение представляется в виде пчелы, которая знает (хранит) расположение (координаты или параметры многомерной функции) какого-то участка. Выделим два варианта поведения.

В первом варианте две пчелы нашли два разных пересекающихся участка, и оба этих участка следует отметить как лучшие или выбранные. Во втором варианте будем считать, что это один участок, центр которого находится в точке, соответствующему большему значению целевой функции.

Второй вариант поведения менее подвержен попаданию в локальные оптимумы за счет просмотра перспективных мест и их окрестностей. Причем, на каждой итерации алгоритма область просмотра уменьшается.

В обычной колонии пчел, например *Apis mellifera* (медоносная пчела домашняя), предполагается, что пчелы со временем выполняют разные роли. В типичном улье может быть от 5000 до 20 000 особей. Взрослые особи (в возрасте от 20 до 40 дней), как правило, становятся фуражирами. Фуражиры обычно выполняют одну из трех ролей: активные фуражиры, фуражиры-разведчики и неактивные фуражиры.

Активные фуражиры летят к источнику нектара, обследуют соседние источники, собирают нектар и возвращаются в улей.

Разведчики обследуют местность вокруг улья (площадью до 130 км^2) в поисках новых источников нектара. Примерно 10% пчел-фуражиров в улье задействованы в качестве разведчиков.

В любой момент некоторое количество пчел-фуражиров неактивно. Они ждут неподалеку от входа в улей. Когда активные фуражиры и разведчики возвращаются в улей, то — в зависимости от качества источника нектара, который они только что посетили, — они могут исполнять виляющий танец перед ждущими неактивными пчелами. Есть довольно веские доказательства того, что этот виляющий танец несет информацию неактивным пчелам о местонахождении и качестве источника нектара. Неактивные фуражиры извлекают из виляющего танца эту информацию об источниках нектара и могут становиться активными фуражирами.

В целом, активный фуражир продолжает собирать нектар из конкретного источника до тех пор, пока он не истощится, после чего эта пчела становится неактивным фуражиром.

Работа пчелиного алгоритма зависит от следующих основных параметров: общего числа пчёл-разведчиков (N); общего числа участков (m); числа элитных участков (e); числа пчёл-разведчиков на элитных участках (n); числа пчёл (l) на остальных ($m-e$) участках; начального размера участков, т.е. размера участков вместе с их окрестностями (S); максимального числа итераций (I).

Основная идея пчелиного алгоритма заключается в том, что все пчёлы на каждом шаге будут выбирать как элитные участки для исследования, так и участки в окрестности элитных, что позволит, во-первых, разнообразить популяцию решений на последующих итерациях, во-вторых, увеличить вероятность обнаружения, близких к оптимальным решения. После чего в окрестности остальных участков ($m-e$), в зависимости от значения их целевой функции, отправляются рабочие пчёлы ($l = N-n$).

Приведём словесное описание алгоритма пчёл. Сначала создается популяция пчёл и производится оценка значений их целевых функций, затем выбор участков для поиска в их окрестностях и отправка пчёл-разведчиков, далее производится выбор пчёл с лучшими значениями целевой функции с каждого участка и отправка рабочих пчёл, осуществляющих случайный поиск с оценкой их целевой функции. Затем формируется новая популяция и производится проверка условия окончания работы алгоритма. Таким образом, ключевой операцией алгоритма пчёл является совместное исследование перспективных областей и их окрестностей. В конце работы алгоритма популяция решений будет состоять из двух частей: пчёлы с лучшими значениями целевой функции элитных участков, а также группы рабочих пчёл со случайными значениями функции. Отличительной особенностью алгоритма является способность динамически разбивать поисковое пространство на области, что уменьшает время работы алгоритма. Данный алгоритм иллюстрирует стратегию поиска «Разделяй и властвуй». Главным преимуществом является тот факт, что резко снижается вероятность попадания в локальный оптимум, а за счет распараллеливания уменьшается время.

Пчелиный алгоритм легко распределяется на несколько параллельных процессов, за счёт чего значительно увеличится его скорость. По сравнению с генетическим алгоритмом, операторы которого могут быть реализованы различным образом, пчелиный алгоритм имеет лишь один оператор, что делает его более простым в использовании. Концепция этих методов основывается на двух совершенно разных природных процессах: пчелиный алгоритм основывается на социальном поведении роя, а генетический алгоритм имитирует процесс эволюции и естественного отбора. За счёт этого есть возможность объединения этих методов.

Заключение

Системы автоматизированного проектирования являются одним из видов наиболее сложных искусственных технических систем.

Разработка МО является самым сложным этапом создания САПР, от которого при использовании условно одинаковых технических средств в наибольшей степени зависят производительность и эффективность функционирования САПР в целом.

Знание математического обеспечения САПР позволяет правильно сформулировать решаемую задачу, выбрать необходимый метод ее решения и разработать модель, эффективно использовать программно-технические средства САПР. Именно поэтому МО САПР считается одним из наиболее важных видов обеспечения с точки зрения функционирования САПР.

При написании пособия авторами ставилась цель рассмотреть модели, методы и алгоритмы конструкторского этапа проектирования, которые используются в САПР ЭВМ.

Авторы придерживаются убеждения, что нельзя написать ни одной книги, не прочитав, хотя бы одну, другую.

Так, и с алгоритмами: нельзя придумать ни одного стоящего алгоритма, если ты, не знаешь ни одного эффективного алгоритма.

Надеемся, что данное пособие стимулирует обучаемых на разработку эффективных алгоритмов конструкторского этапа проектирования ЭВМ и, в частности, задач теории графов.

Список литературы

1. Абрайтис, Л.Б. Автоматизация проектирования топологии цифровых интегральных микросхем / Л.Б. Абрайтис. - М.: Радио и связь, 1985. – 198 с.
2. Алексеев, О.В. Автоматизация проектирования радиоэлектронных средств: учеб. пособие для студентов радиотехнических специальностей вузов / О.В. Алексеев. – М.: Высшая школа, 2000. – 430 с.
3. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы: Учебное пособие.– 2-е изд.–М.: Физматлит, 2006. –320 с.
4. Гладков Л.А, Курейчик В.В., Курейчик В.М. Генетические алгоритмы. – М.: Физматлит, 2010. – 366 с
5. Гладков Л. А., Курейчик В. В, Курейчик В. М. и др. Биоинспирированные методы в оптимизации: монография. – М.: Физматлит, 2009. –384 с.
6. Гладков Л. А., Курейчик В. В, Курейчик В. М. Дискретная математика. Учебник/ Таганрог: Изд-во ТТИ ЮФУ, 2011. –312 с.
7. Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И. Лекции по теории графов. / Изд.2, испр. М.: УРСС, 2009. –392 с.
8. Зыков А.А. Основы теории графов. М.: Вузовская книга, 2004. – 664 с. - ISBN: 5-9502-0057-8
9. Иванова Н.Ю., Петров А.С., Поляков В.И., Романова Е.Б. Технология проектирования печатных плат в САПР P-CAD-2006 / СПб: СПбГУ ИТМО, 2009 – 168 с.
10. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS. – 3-е изд. – М.: «Вильямс», 2013. –1324 с.
11. Корячко, В.П. Теоретические основы САПР / В.П. Корячко, В.М. Курейчик, И.П. Норенков. – М.: Энергоиздат, 1987. – 400 с.
12. Кристофидес Н, Теория графов: Алгоритмический подход: пер. с англ. / Н. Кристофидес . – М. : Мир, 1978 . – 432 с.
13. Мелихов А.Н., Берштейн Л.С., Курейчик В.М. Применение графов для проектирования дискретных устройств - М.:Наука, 1974. –304 с.
14. Муромцев, Д. Ю. Математическое обеспечение САПР : учебное пособие / Д. Ю. Муромцев, И. В. Тюрин . – 2-е изд., перераб. и доп . – СПб.: Лань, 2014 . – 464 с.
15. Муромцев, Ю.Л. Задачи компоновки радиоэлектронной аппаратуры: лабораторные работы / Ю.Л. Муромцев, В.Н. Грошев, В.В. Трейгер. – Тамбов: ТИХМ, 1987. – 36 с.
16. Муромцев, Ю.Л. Задачи размещения радиоэлектронной аппаратуры: лабораторные работы / Ю.Л. Муромцев, В.В. Трейгер, В.Н. Грошев. – Тамбов: ТИХМ, 1988. – 32 с.
17. Муромцев, Ю.Л. Задачи трассировки печатных плат: лабораторные работы / Ю.Л. Муромцев, В.В. Трейгер, В.Н. Грошев. – Тамбов: ТИХМ, 1990. – 32 с.

18. Новиков Ф.А. Дискретная математика для программистов: Учебник для вузов. 3- е изд. — СПб.: Питер, 2009. – 384 с.
19. Норенков, И.П. Основы автоматизированного проектирования : учебник для вузов / И.П. Норенков. – - 3-е изд., перераб. и доп . – М.: Изд-во МГТУ им. Н.Э. Баумана, 2006. – 448 с.
20. Петухов, Г.А. Алгоритмические методы конструкторского проектирования узлов с печатным монтажом / Г.А. Петухов, Г.Г. Смолич, Б.И. Юлин. – М.: Радио и связь, 1987. – 152 с.
21. Разевиг, В.Д. Применение программ PCAD и Pspice для схемотехнического моделирования на ПЭВМ / В.Д. Разевиг. – М.: Радио и связь, 1992. – Вып. 4. 71 с.
22. Селютин, В. А. Машинное конструирование электронных устройств / В. А. Селютин. – М. : Советское радио, 1977. – 384 с.
23. Советов, Б.Я. Информационные технологии: учебник для вузов / Б.Я. Советов. – М.: Высшая школа, 2003. – 352 с.
24. Харари Ф. Теория графов (3-е издание). М.: КомКнига, УРСС, 2006. – 259 с.
25. Шапорев С.Д. Дискретная математика. – СПб.: БХВ-Петербург, 2006. – 400 с.

УНИВЕРСИТЕТ ИТМО

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра вычислительной техники Университета ИТМО создана в 1937 году и является одной из старейших и авторитетнейших научно-педагогических школ России. Заведующими кафедрой в разное время были выдающиеся деятели науки и техники М.Ф. Маликов, С.А. Изенбек, С.А. Майоров, Г.И. Новиков. Многие поколения студентов и инженеров в Советском Союзе и за его пределами изучали вычислительную технику по учебникам С.А. Майорова, Г.И. Новикова, О.Ф. Немолочного, С.И. Баранова, В.В. Кириллова, А.А. Приблуды, Т.И Алиева, Б.Д.Тимченко и др.

Основные направления учебной и научной деятельности кафедры в настоящее время включают в себя встроенные управляющие и вычислительные системы на базе микропроцессорной техники, информационные системы и базы данных, сети и телекоммуникации, моделирование вычислительных систем и процессов, обработка сигналов.

Выпускники кафедры успешно работают не только в разных регионах России, но и во многих странах мира: Австралии, Германии, США, Канаде, Германии, Индии, Китае, Монголии, Польше, Болгарии, Кубе, Израиле, Камеруне, Нигерии, Иордании и др. Выпускник, аспирант и докторант кафедры ВТ Аскар Акаев был первым президентом Киргизии.

Зыков Анатолий Геннадьевич

Поляков Владимир Иванович

Алгоритмы конструкторского проектирования ЭВМ

Учебное пособие по дисциплине
«Конструкторско-технологическое обеспечение производства ЭВМ»

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49