Общие сведения о САПР

Проектирование — процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части (ISO 24765)

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.

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).
4. Исключение человека из цикла проектирования.
5. Облегчение получения документации и внесения в нее изменений.
6. Развитие средств ВТ.

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

Этапы жизненного цикла промышленных изделий

Проектирование можно разделить как во времени, так и по подразделениям проектной организации.

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.

Традиционно проектирование сложных технических систем подразделяют на следующие этапы или стадии разработки:


АСНИ
САПР
ТЗ на проектируемый объект
1) Пакетный поиск
2) Анализ выбранной информации и формулировка Технических Требований (ТТ) к проектируемому объекту
3) Выдача ТТ
НИР
1) Формирование критериев качества и управления
2) Управление научным экспериментом
3) Разработка математических моделей
4) Разработка новых технических средств
5) Отработка технологических процессов
6) Выдача Технических Предложений (ТП)
СКР
1) Эскиз проектируемой системы
2) Предварительные проектные решения
3) Первые проектные документы
Техническое проектирование
Уточнение решений, принятых при эскизном проектировании
Все документы – только исходные данные (ИД) для Рабочего Проекта
Рабочее проектирование
Оформление проектных документов и эксплуатационной документации
Изготовление и испытания спроектированного объекта. Внесение коррекции

Блочно-иерархический подход к проектированию

Распределение работ между подразделениями организации производится с использованием блочно-иерархического подхода.

Представления о сложных технических объектах в процессе их проектирования разделяются на аспекты и иерархические уровни. Разделение описаний проектируемых объектов на иерархические уровни по степени подробности отражения свойств объектов составляет сущность блочно-иерархического подхода к проектированию.

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.

Структурированное описание ЭВМ

Горизонтальные уровни	Вертикальные уровни			
	Функциональный	Алгоритмический	Конструктор-ский	Технологический
	Системный	Программирова-ние системы	Шкаф, стойка	Принципиальная схема технологи-ческого процесса
			Панель	
	Логический	Программирова-ние модулей	ТЭЗ	Маршрутная технология
			Модуль	
	Схемотехнический	Проектирование микропрограмм	Кристалл	Технологические операции
	Компонентный		Ячейка	

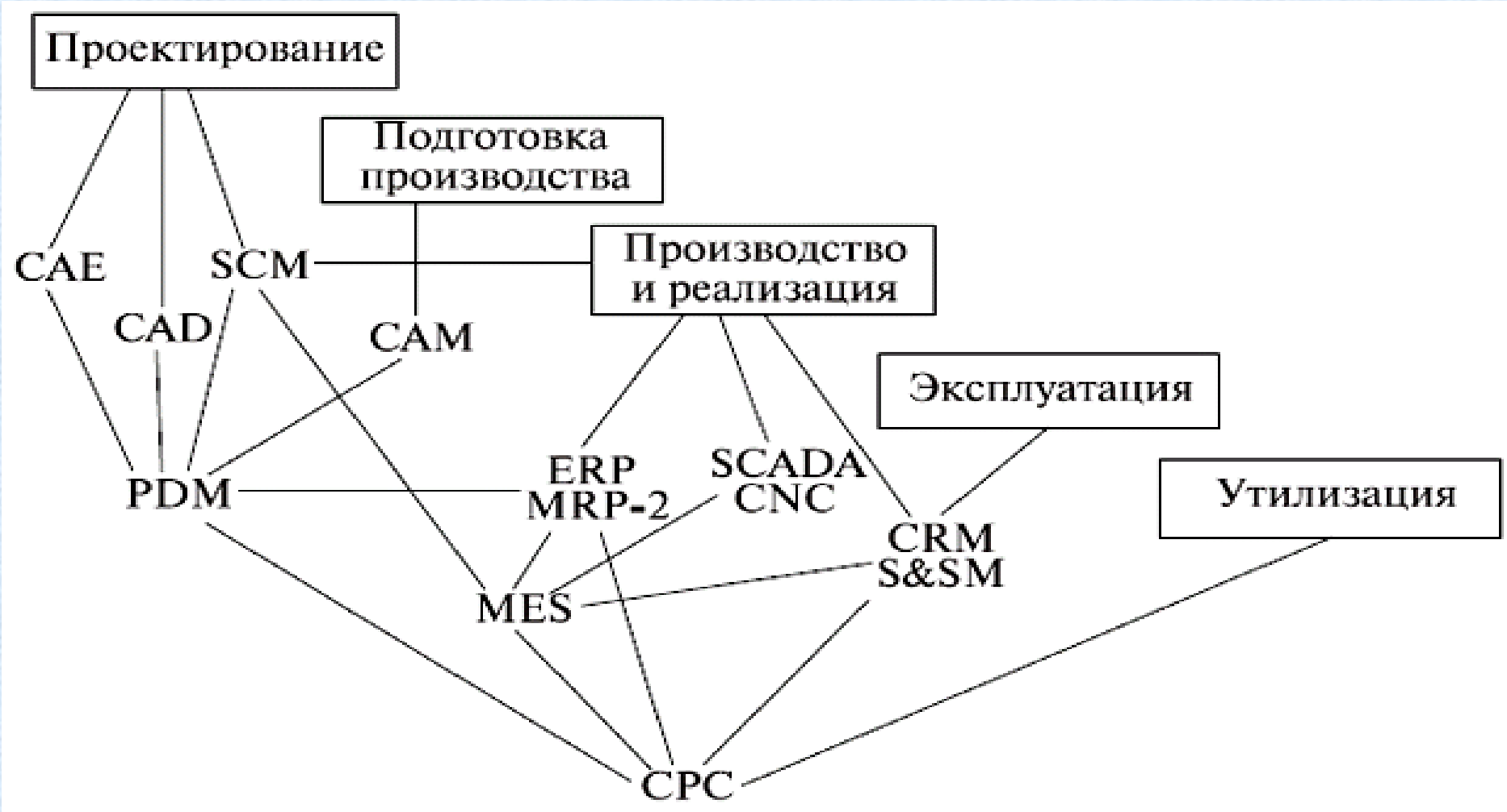
Такой подход позволяет на каждом иерархическом уровне формулировать задачи приемлемой сложности, поддающиеся решению с помощью имеющихся средств проектирования. Разбиение на уровни должно быть таким, чтобы документация на блок любого уровня была обозрима и воспринимаема одним человеком.

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

САПР ЭВМ и их место среди других автоматизированных систем

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*).



Уже на стадии проектирования требуются услуги системы управления цепочками поставок *SCM* (*Supply Chain Management*).

К автоматизированными системами управления предприятием (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*).

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.

Системы управления данными в интегрированном информационном пространстве *CPC* (*Collaborative Product Commerce*) или *PLM* (*Product Lifecycle Management*) обеспечивают 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.

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.

Математическое обеспечение САПР

Математическое обеспечение САПР состоит из математических моделей объектов проектирования, методов и алгоритмов выполнения проектных операций и процедур

Требования к математическому обеспечению

Свойства математического обеспечения оказывают существенное, а иногда и определяющее влияние на возможности и показатели САПР. Требования, предъявляемые к МО в САПР:

Универсальность. Под универсальностью МО понимается его применимость к широкому классу проектируемых объектов. Высокая степень универсальности МО нужна для того, чтобы САПР была применима к любым или большинству объектов, проектируемых на предприятии.

Алгоритмическая надежность. 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.

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.

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 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.

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, отражающая 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.

Элементы теории графов

Под *абстрактным графом* или просто графом $G(X,U)$ понимают совокупность непустого множества X и изолированного от него подмножества U, 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.

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

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 $\rho(x_i)$.

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

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

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*.

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*.

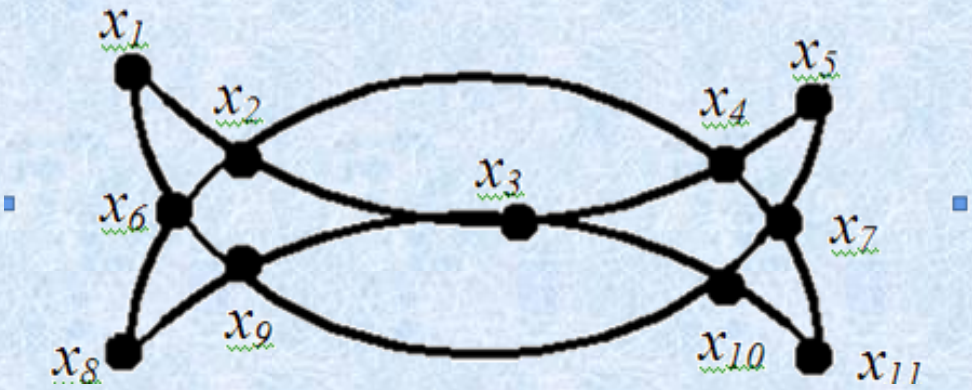
Циклом называют последовательность ребер $u_1=(x_1, x_i)$, ..., $u_k=(x_j, x_1)$, при которой в результате обхода вершин графа по этим ребрам возвращаются в исходную вершину x_1.

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.

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 изображен граф «Сабли Магомета». Это эйлеров граф, эйлеров цикл $x_1\, x_6\, x_9\, x_{10}\, x_{11}\, x_7\, x_4\, x_2\, x_6\, x_8\, x_9\, x_3\, x_{10}\, x_5\, x_4\, x_3\, x_2\, x_1$.



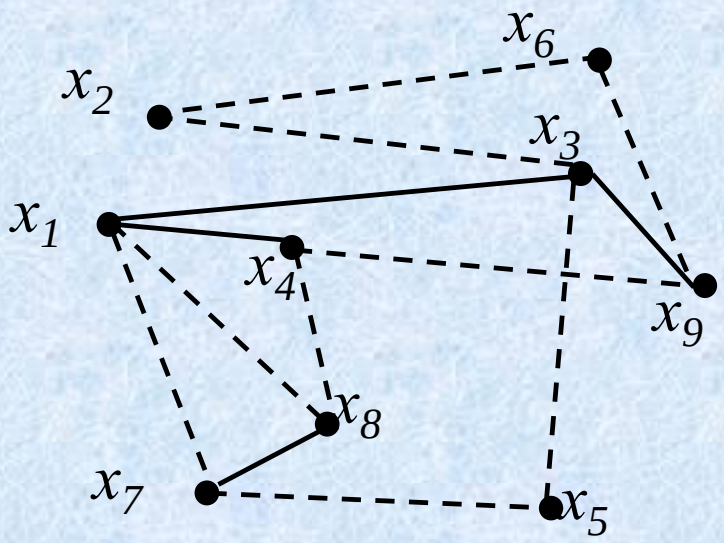
Необходимым и достаточным условием наличия в конечном связном графе эйлерова цикла является четность степеней всех его вершин (*теорема Эйлера*).

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

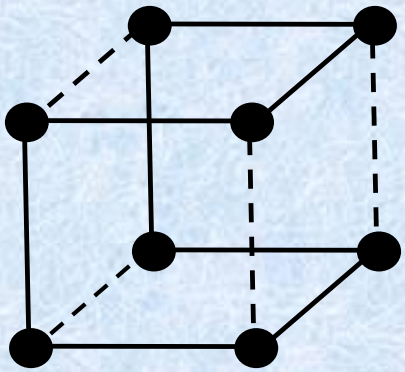
Известно (*теорема Оре – Дирака*), что граф имеет гамильтонов цикл, если сумма локальных степеней двух любых вершин графа больше или равна числу вершин графа, т.е. $\forall x_i\, x_j \in X\, [\rho(x_i) + \rho(x_j) \geq n]$.

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

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



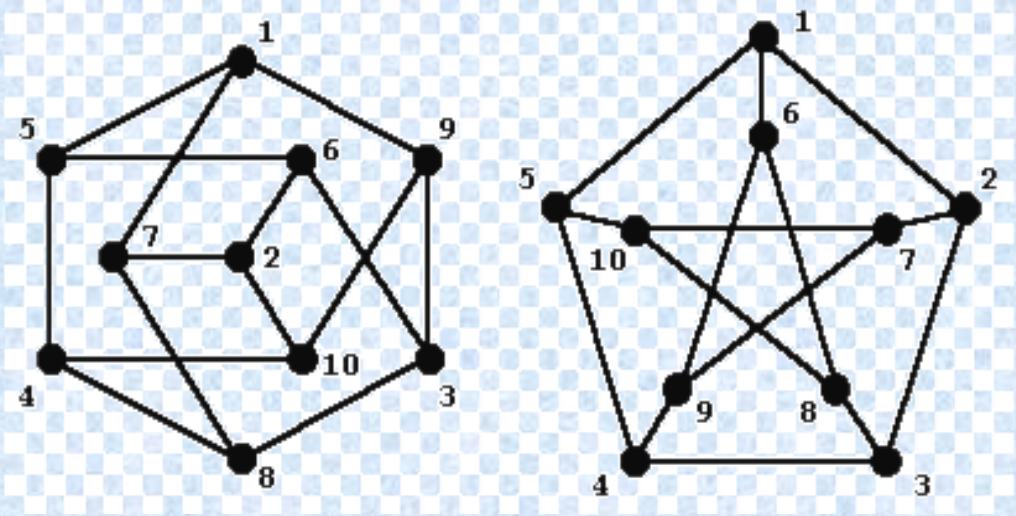
Для гамильтонова цикла критерий существования не известен. Существуют лишь частные критерии наличия в графе гамильтонова цикла. То, что критерии существования гамильтонова цикла в графе являются достаточными, но не необходимыми можно проиллюстрировать с помощью графа-куба



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

При размещении графа в виде *геометрической фигуры* существует 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.



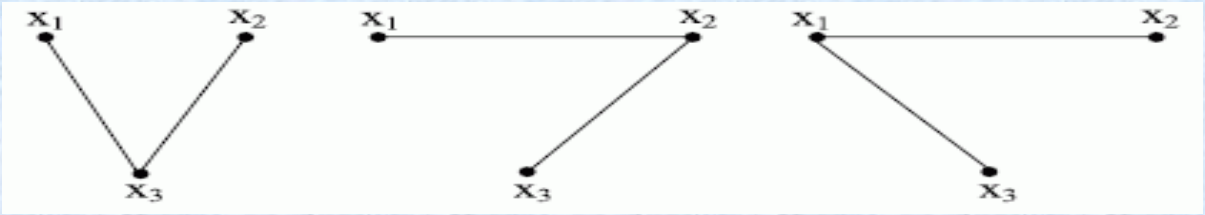
Если в графе $G(X, U)$ опущены некоторые ребра, а число вершин осталось прежним, то полученный граф $G(X, U')$ называют *частичным графом* $G(X, U)$ или *суграфом*.

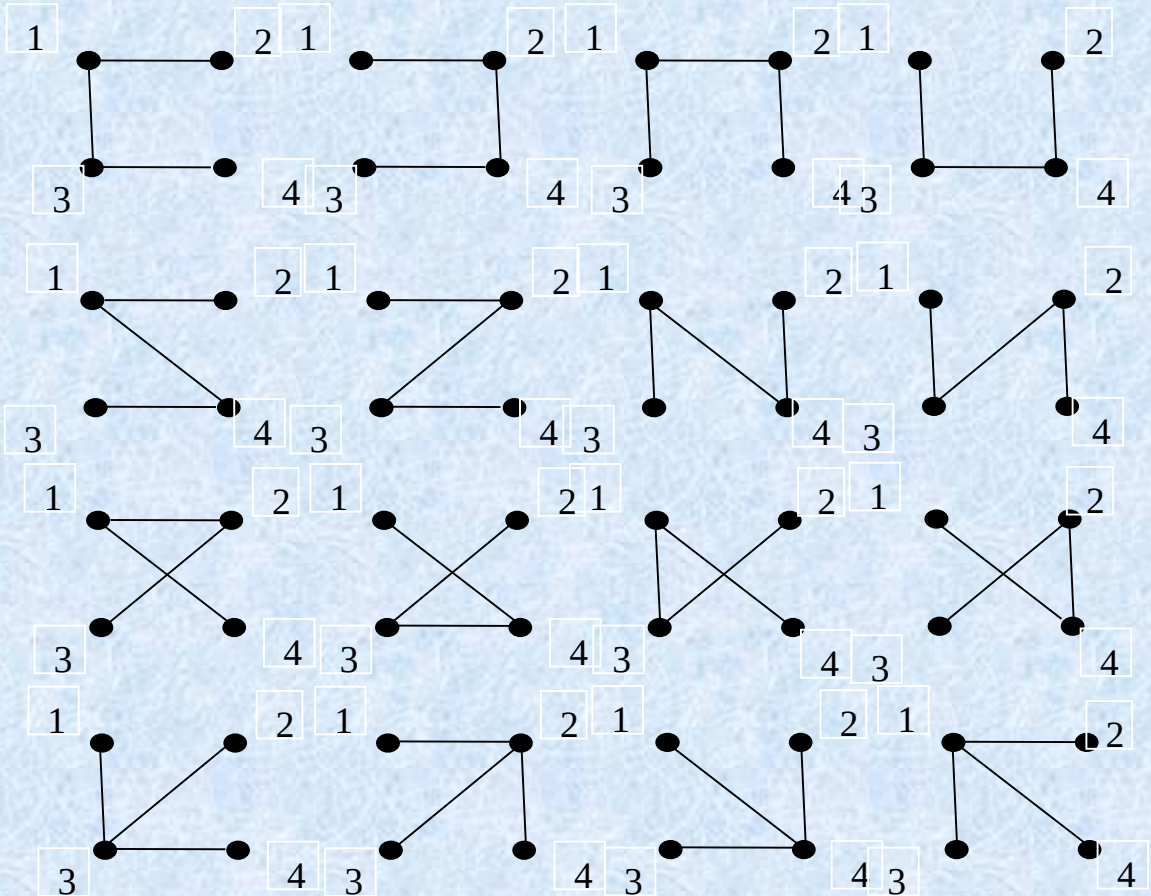
Если в графе $G(X, U)$ опущены некоторые ребра и инцидентные им вершины, то полученный граф $G(X', U')$ называют *подграфом* $G(X,U)$.

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 $|W| = n - p$.

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, для n=5 – 125 деревьев и так далее.





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.

Цикломатическое число графа равно максимальному числу независимых циклов.

Хроматическое число. Пусть, задан граф $G(X,U)$ без петель. Разобьем множество его вершин на k непересекающихся подмножеств $X_1, X_2, \ldots, X_k$, $X = \bigcup_{i=1}^{k} X_i, \forall X_i, X_j \subset X [X_i \cap X_j = \varnothing]$ 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 $G(X_1, X_2, U)$.

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

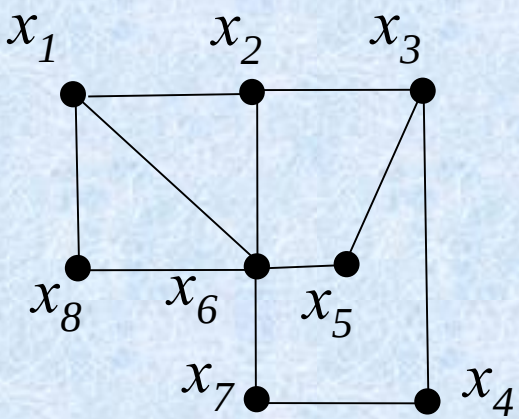
Число планарности. Граф $G(X, U)$ называют *плоским* тогда и только тогда, когда он ***геометрически реализован*** на плоскости так, что все его ребра пересекаются только в вершинах X графа. Граф $G(X,U)$ называют *планарным*, если он ***может быть геометрически реализован*** на плоскости без пересечения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

На рисунке приведены непланарные графы Понтрягина-Куратовского (полный граф K_5 и полный двудольный граф K_{33} . 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 $n \geq 4$ $\Theta(G) = (n - 3)(n - 4)/2$. Минимальное число плоских суграфов, на которые разбивается граф $G(X,U)$ называется *толщиной графа* $t(G)$.

В 1976 г. американские математики К. Аппель и В. Хейкен доказали гипотезу четырех красок, существенно используя при этом компьютерные вычисления.

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, $\alpha(G)=4$.

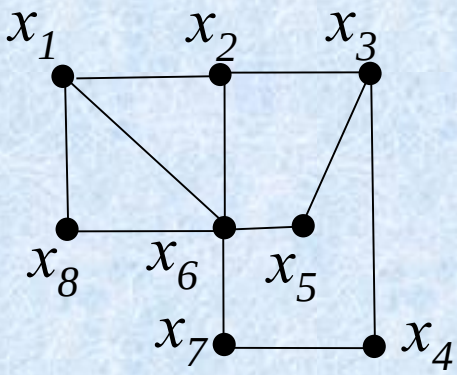
Заметим, что при раскраске графа, множество вершин, окрашенных в один цвет – внутренне устойчивое.

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 $\beta(G)$, $\beta(G) = \min|X_s|$.

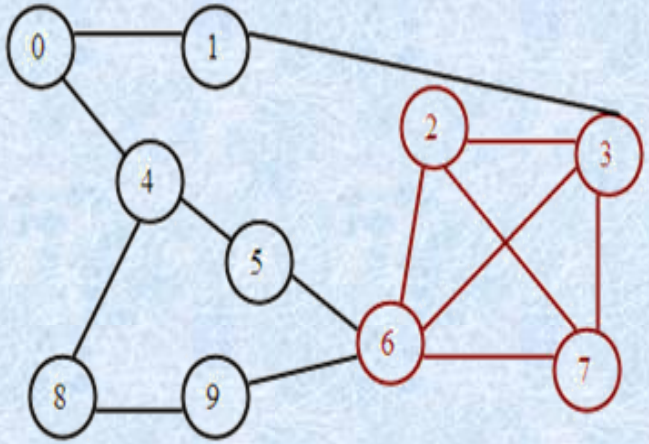


Для графа $G(X,U)$, изображенного на рисунке, множества вершин $\{x_1, x_3, x_7\}$, $\{x_2, x_4, x_5, x_8\}$ являются минимальными, но не наименьшими.
Множества $\{x_3, x_6\}$ и $\{x_4, x_6\}$ является наименьшими, $\beta(G)=2$.

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.

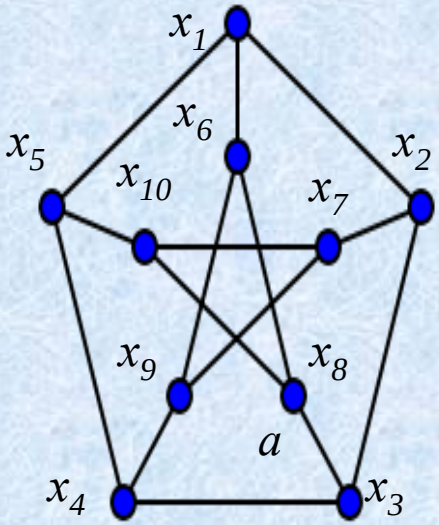
Число вершин клики графа называется *плотностью графа* и обозначается $f(G)$.

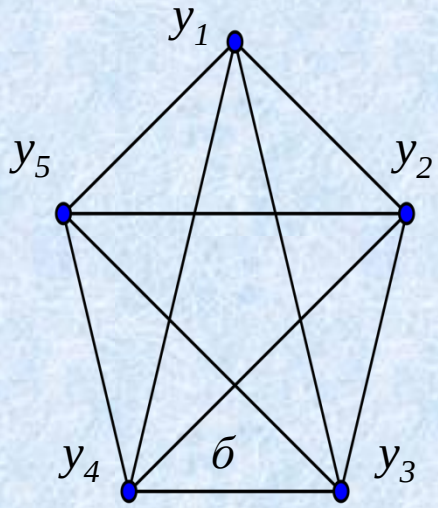


Для графа $G(X,U)$, изображенного на рис. клику образуют вершины (2, 3, 6, 7). Плотность этого графа $f(G)=4$.

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 рисунке *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.*б*).

Число Хадвигера графа Петерсена $\eta(G)=5$.

Алгоритмы автоматизированного проектирования ЭВМ

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.* 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.

Элементы теории сложности

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

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.

В таблице 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.

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.

$O(n)$	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	$6{,}64 + n_4$	$9{,}97 + n_4$

На конструкторском этапе проектирования, решают задачи:

1. компоновки схем;
2. размещения элементов и узлов;
3. реализация печатных и проводных соединений для электронных изделий всех уровней;
4. выпуска конструкторской документации.

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:

- алгоритмы, использующие методы целочисленного программирования;
- алгоритмы, основанные на методе ветвей и границ;
- последовательные алгоритмы;
- итерационные алгоритмы;
- 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 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.

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, $n_1, n_2, \ldots, n_k$ $(n_1 + n_2 + \ldots + n_k = n)$, 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

$$\forall G_i(X_i,U_i), G_j(X_j,U_j) \subset G(X,U)[G_i(X_i,U_i) \neq G_j(X_j,U_j) \Rightarrow (X_i \cap X_j = \varnothing$$

$\& U_i \bigcap U_j = U_{ij})]$ $\bigcup_{i=1}^{k} G_i(X_i, U_i) = G(X, U)$ $i, j =$ 1, 2, …, 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.

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 $X^*=X \setminus X_1$, $U^*= U \setminus U_1$.

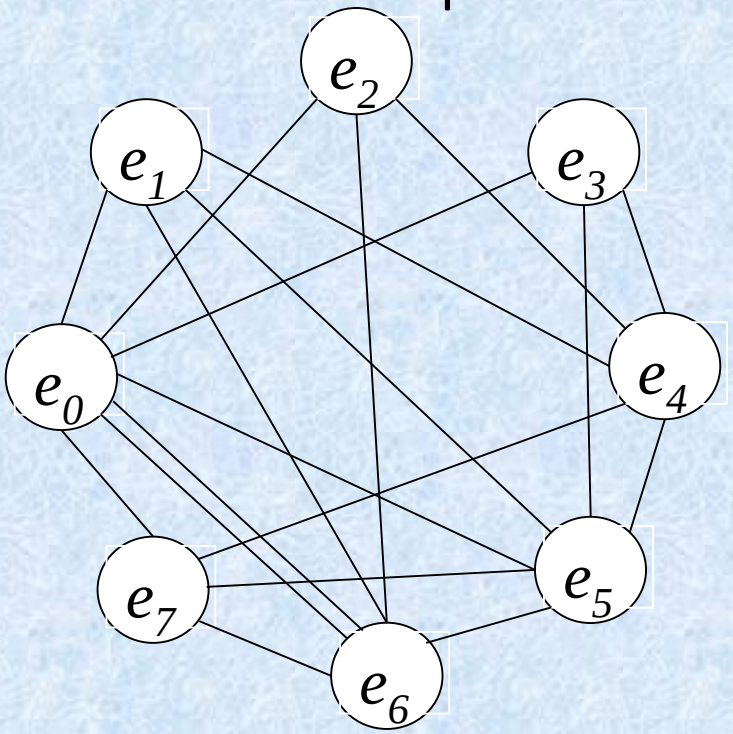
Далее формируется второй подграф G_2 и т.д.

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 не-способность находить глобальный минимум числа внешних связей

Наибольшая эффективность метода последовательного разбиения графа имеет место, когда число вершин графа *G* значительно больше числа вершин в любой части разбиения $n >> n_1, n_2, .., n_k$.



$R =$

	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

$$\delta(x_j) = \rho(x_j) - 2\sum_{x_s \in X_1} r_{js} \quad .$$

$\delta(x_0) = 7 - 6 = 1$, $\delta(x_4) = 5 - 2 = 3$, $\delta(x_6) = 6 - 6 = 0$.

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

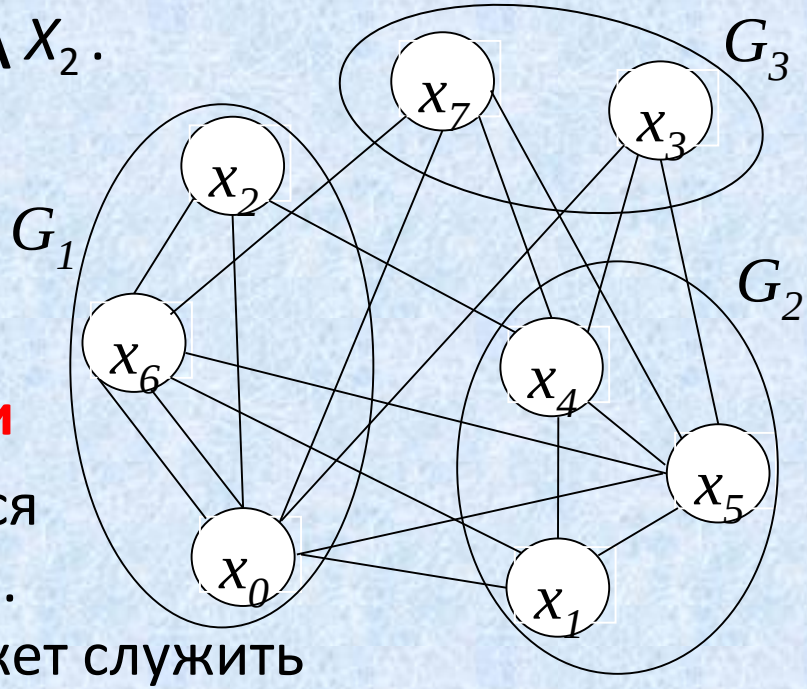
7. Находим вершину $x_i \in X^*$ с минимальной локальной степенью $\rho(x_i)$, это вершины x_1, x_3 и x_7.

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.



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

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

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 $b_{ij} = \alpha(x_i) + \alpha(x_j) - 2r_{ij}$.

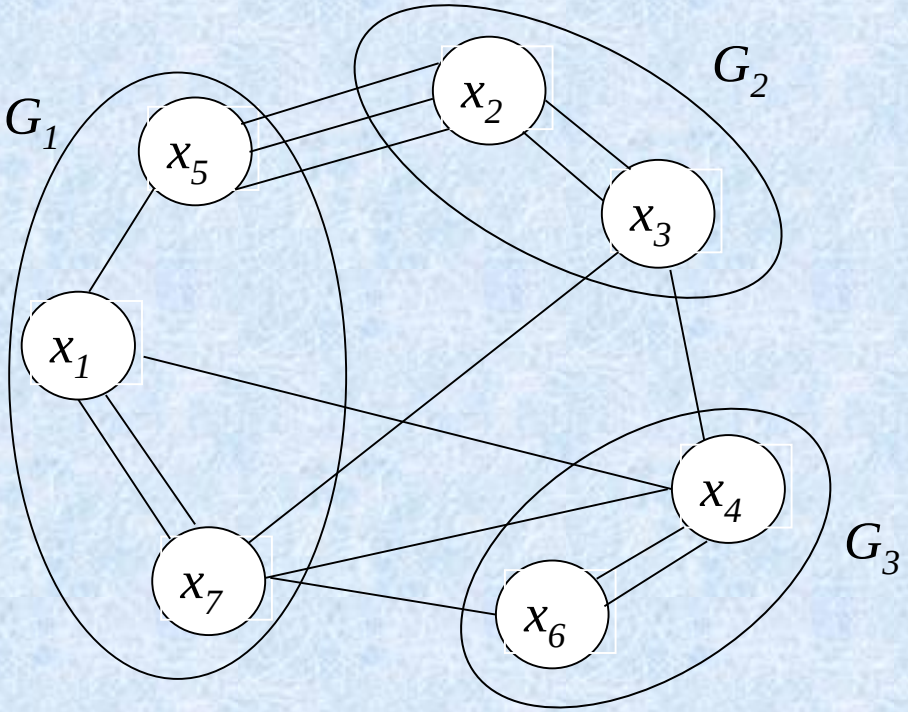
4. Выбирается пара вершин x_i и x_j, для которых b_{ij} максимально. Эти вершины переставляются.

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 $b_{ij} \leq 0$, т.е. никакая перестановка не уменьшит количество 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.



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

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

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

$R'=$

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

$b_{12}= \alpha(x_1) + \alpha(x_2) - 2r_{12} = -3 + 1 - 0 = -2;$
$b_{13}= \alpha(x_1) + \alpha(x_3) - 2r_{13} = -3 + -1 - 0 = -4;$
$b_{52}= \alpha(x_5) + \alpha(x_2) - 2r_{52} = 2 + 1 - 6 = -3$ и т.д.

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

$B =$

	x_2	x_3
x_1	-2	-4
x_5	-3	1
x_7	0	-4

Пересчитаем b_{ij}

$B =$

	x_2	x_5
x_1	-2	-5
x_3	0	-1
x_7	-3	-4

$R_{12} =$

	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

Все $b_{ij} \le 0$, 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 $\alpha(x_i)$.

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

$$B = \begin{array}{c|cc} & x_4 & x_6 \\ \hline x_1 & -1 & -2 \\ x_3 & -1 & 0 \\ x_7 & -2 & -3 \end{array}$$

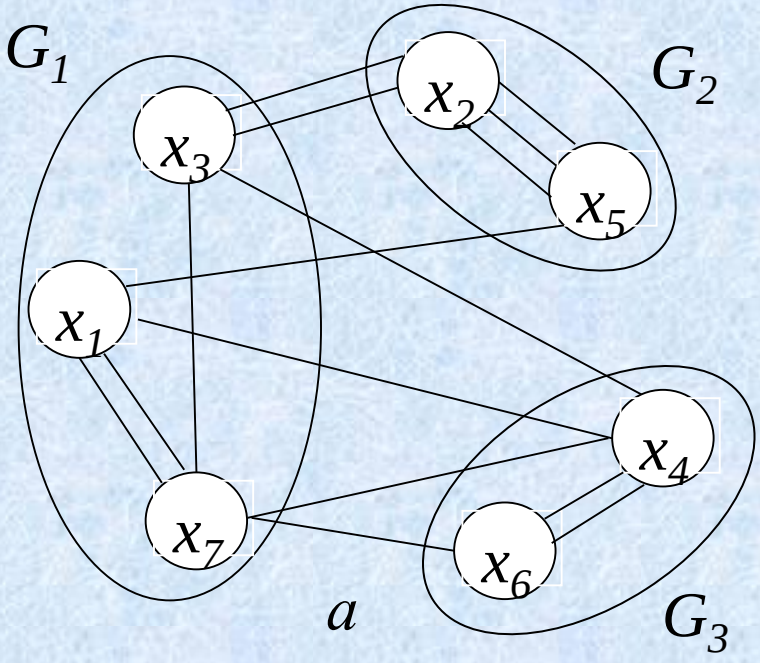
$$R_{13} = \begin{array}{c|ccccc|c} & x_1 & x_3 & x_7 & x_4 & x_6 & \alpha(x_i) \\ \hline x_1 & 0 & 0 & 2 & 1 & 1 & 0 \\ x_3 & 0 & 0 & 1 & 1 & 0 & 0 \\ x_7 & 2 & 1 & 0 & 1 & 1 & -1 \\ \hline x_4 & 1 & 1 & 1 & 0 & 2 & 1 \\ x_6 & 1 & 0 & 1 & 2 & 0 & 0 \end{array}$$

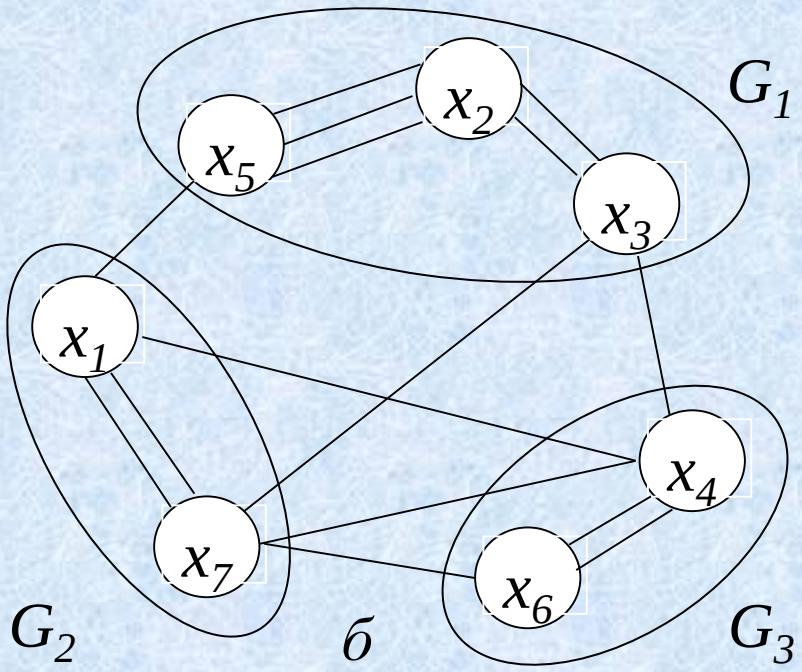
Все $b_{ij} \le 0$. Выберем нерассмотренные куски $G_2(X_2, U_2)$ и $G_3(X_3, U_3)$ и вычислим $\alpha(x_i)$.

$$R_{23} = \begin{array}{c|cccc|c} & x_2 & x_5 & x_4 & x_6 & \alpha(x_i) \\ \hline x_2 & 0 & 3 & 0 & 0 & -3 \\ x_5 & 3 & 0 & 0 & 0 & -3 \\ \hline x_4 & 0 & 0 & 0 & 2 & -2 \\ x_6 & 0 & 0 & 2 & 0 & -2 \end{array}$$

Все $\alpha(x_i)$ и, соответственно, $b_{ij} \le 0$, работа алгоритма заканчивается.

В результате получили разрезание (рисунок)





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. (*б*).

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, …, R_{k-1}.
8. Разбиение получено.

Продемонстрируем работу алгоритма на примере предыдущей компоновки

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

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

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

$$R = \begin{array}{c|ccccccc|c} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & \alpha(x_i) \\ \hline x_1 & 0 & 0 & 0 & 1 & 1 & 0 & 2 & 4 \\ x_2 & 0 & 0 & 2 & 0 & 3 & 0 & 0 & 1 \\ x_3 & 0 & 2 & 0 & 1 & 0 & 0 & 1 & 0 \\ \hline x_4 & 1 & 0 & 1 & 0 & 0 & 2 & 1 & -1 \\ x_5 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 4 \\ x_6 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & -3 \\ x_7 & 2 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array}$$

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

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

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 $\alpha(x_i)$.

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

$B =$

	x_6	x_7
x_4	-1	2
x_1	2	-1

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

$R' =$

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

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

$R' =$

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

Получено разрезание графа G, совпадающее с оптимальным

Алгоритмы размещения элементов

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:
• алгоритмы решения математических задач, являющихся моделями задач размещения;
• конструктивные алгоритмы начального размещения;
• итерационные алгоритмы улучшения начального размещения;
• непрерывно - дискретные методы размещения.

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

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.

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 на множестве всех перестановок P, соответствует расположению составляющих 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 $m(m-1)/2$ 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

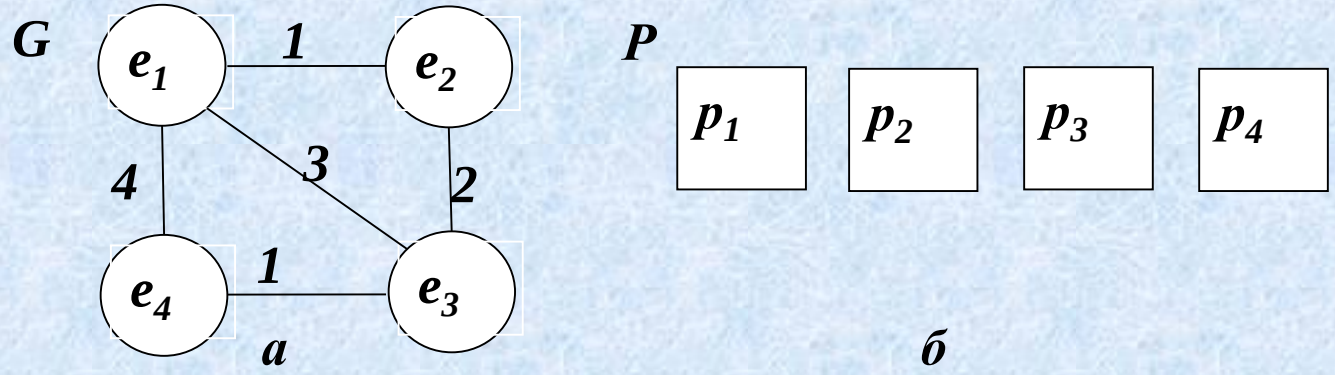
$F(P) = F(q) + w(P) + v(P)$, где

$$F(q) = \sum_{e_i \in E_k} \sum_{e_s \in E_k} r_{is} d_{p(i)p(s)}$$ - 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. (*а*)) на множестве позиций *P* (рис. (*б*)).



Составим матрицы соединений *R* графа и расстояний *D* множества позиций.

$$R = \begin{array}{c|cccc} & e_1 & e_2 & e_3 & e_4 \\ \hline e_1 & 0 & 1 & 3 & 4 \\ e_2 & & 0 & 2 & 0 \\ e_3 & & & 0 & 1 \\ e_4 & & & & 0 \end{array} \qquad D = \begin{array}{c|cccc} & p_1 & p_2 & p_3 & p_4 \\ \hline p_1 & 0 & 1 & 2 & 3 \\ p_2 & & 0 & 1 & 2 \\ p_3 & & & 0 & 1 \\ p_4 & & & & 0 \end{array}$$

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 $v(P) = r \times d = 2 + 1 + 0 = 3$.

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 $\{p_1, p_3, p_4\}$.

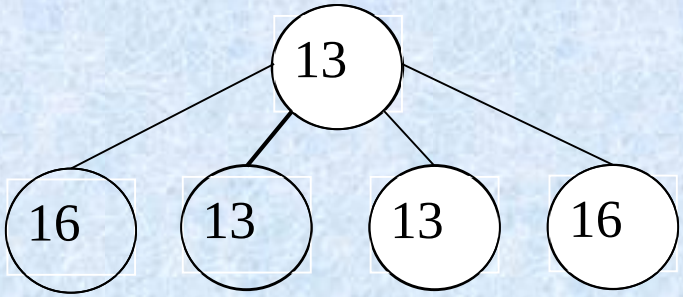
Составим вектор, соответствующий первой строке матрицы R $r_1 = \{4\ 3\ 1\}$, и вектор, соответствующий второй строке матрицы D $d_2 = \{1\ 1\ 2\}$, суммарная длина соединений между размещенным и неразмещенными элементами

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

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 $v(P) = r \times d = 2 + 2 + 0 = 4$. 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 рисунке представлены результаты расчета нижних границ для первого яруса дерева решений.

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 $\{p_1, p_3\}$;

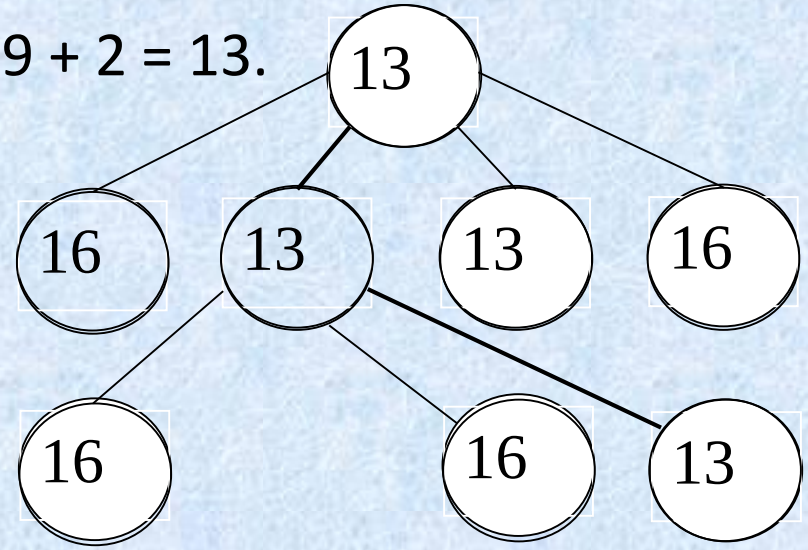
$r_1 = \{4\ \ 3\}$ и $d_2 = \{1\ \ 1\}$, $r_1 \times d_2 = 4 + 3 = 7$;

$r_2 = \{2\ \ 0\}$ 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×d = 2. $F(P) = 2 + 9 + 2 = 13$.

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



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 рисунке представлены результаты расчета нижних границ для трех ярусов дерева решений.

Назначаем элемент e_3 в позицию p_3.

8. Неразмещенный элемент $\{e_4\}$,

свободная позиция $\{p_1\}$.

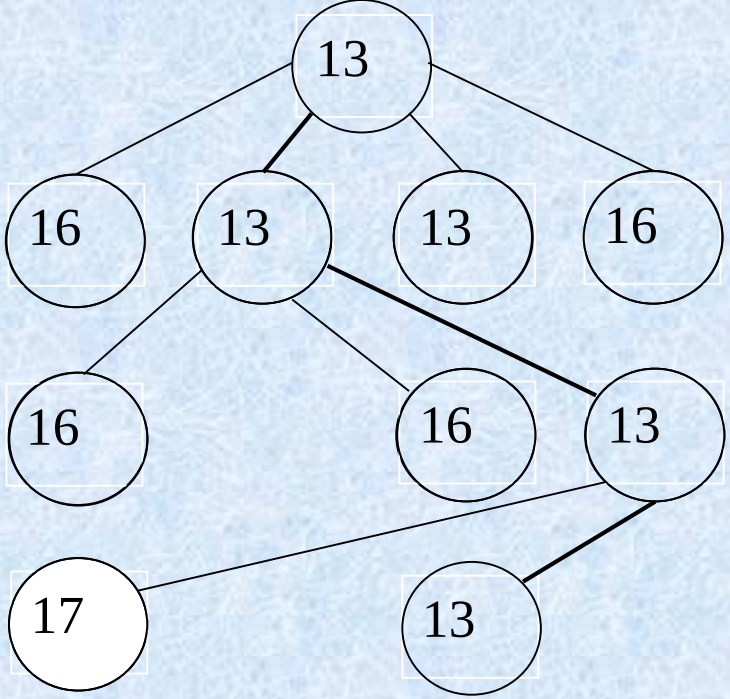
Помещаем $\{e_4\}$в позицию $\{p_1\}$.

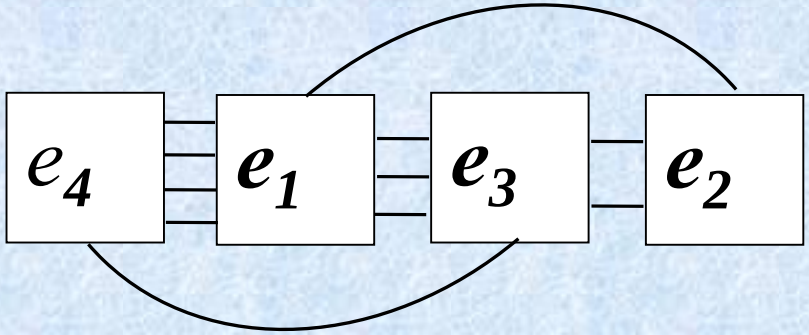
$F(q) = r_{12}d_{24} + r_{13}d_{23} + r_{23}d_{43} + r_{14}d_{21} + r_{24}d_{41}$

$+ r_{34}d_{31} = 2 + 3 + 2 + 4 + 0 + 2 = 13.$

$w(P) = v(P) = 0.$ $F(p) = 13.$

Получено размещение





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.

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

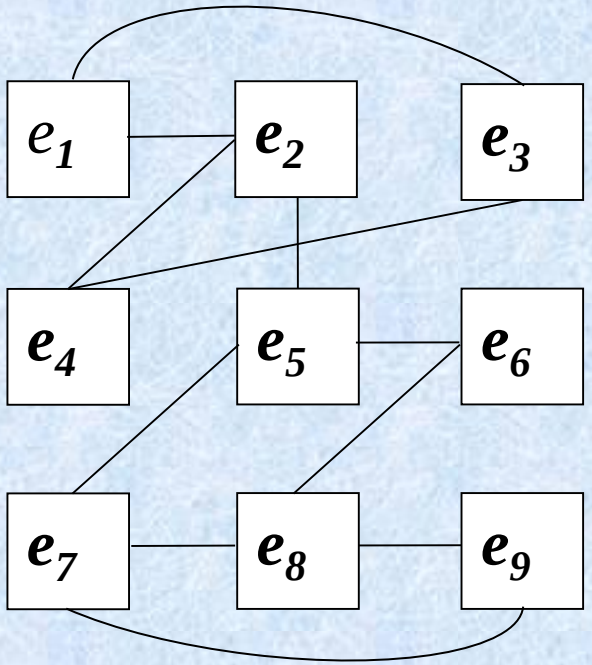
Значение функции $F(p)=18$.

1. Для каждой пары соединенных вершин e_i и e_j вычислим l_{ij}:

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

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

2. Меняем местами вершины e_3 и e_2

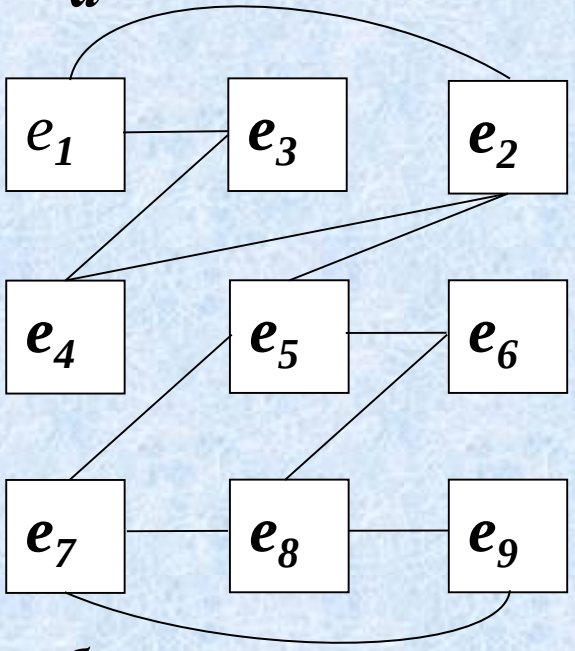


а

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

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

$max l'_{ij} = l_{24} = 3$. Максимальная длина не уменьшилась, возвращаем размещение



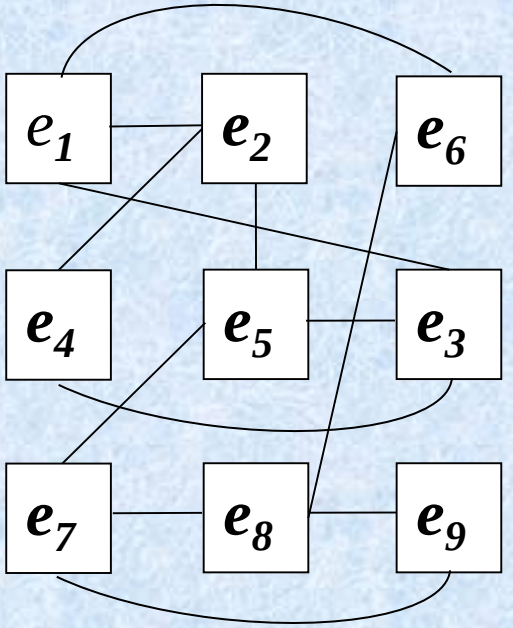
б

4. Меняем местами вершины e_3 и e_6

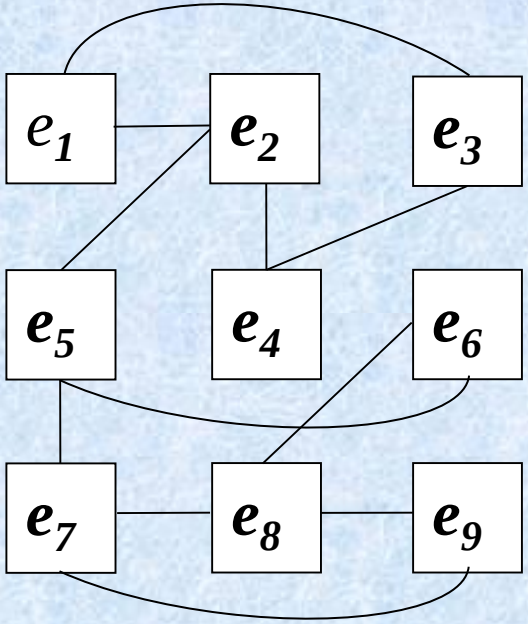
5. Для каждой пары соединенных вершин e_i и e_j вычислим новые l'_{ij}:

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.

$max l'_{ij} = l_{13} = l_{68}$ =3. Максимальная длина не уменьшилась, возвращаем размещение



а



б

6. Меняем местами вершины e_4 и e_5

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.

$max l'_{ij} = l_{13} = l_{25} = l_{34} = l_{56} = l_{68} = l_{79}$=2.

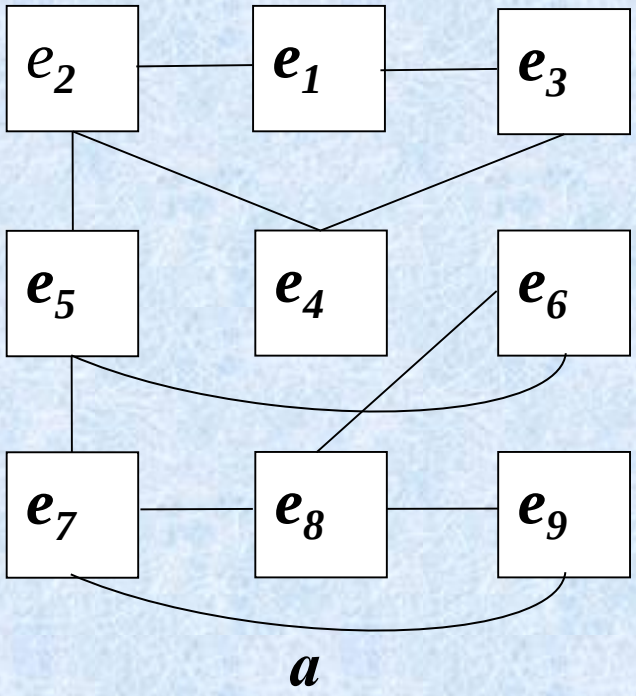
Максимальная длина уменьшилась.

Сохраняем полученное размещение.

Количество максимумов k = 6.

8. Меняем местами вершины e_1 и e_2

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



а

10. $max l'_{ij} = l_{24} = l_{34} = l_{56} = l_{68} = l_{79}$=2. Меняем местами e_2 и e_1 и возвращаемся к предыдущему размещению

11. Меняем местами e_2 и e_5 и т.д.

Дальнейшие попытки уменьшить максимальную длину результата не дадут.
Значение функции $F(p)$=16.

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 $W=\{e_1, e_2, \ldots, e_k\}$ - некоторое несвязное множество элементов, а $L=\{p_1, p_2, \ldots, p_k\}$ – 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 $n > 5$.

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 ($n \leq 20$).

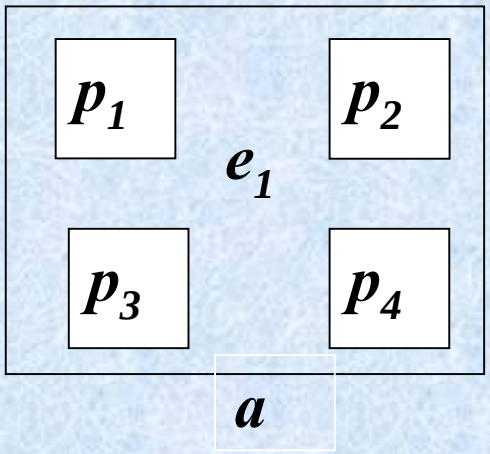
Достаточные результаты достигаются сочетанием конструктивного алгоритма (сложность $\sim O(n)$) с улучшающим итерационным (сложность $\sim O(n^2) \div O(n^4)$).

Размещение разногабаритных элементов

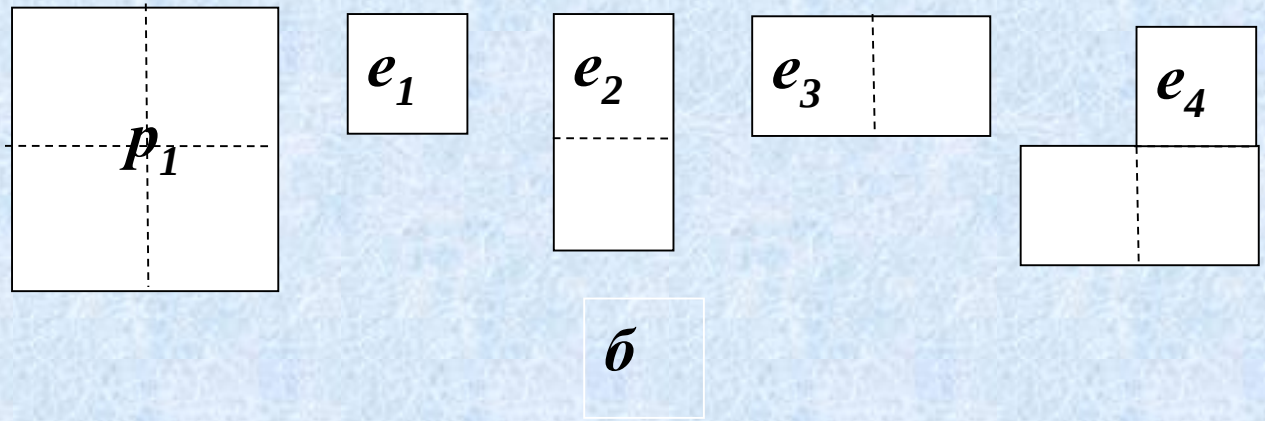
Рассмотренные ранее алгоритмы не учитывали габариты размещаемых элементов. Задача размещения существенно усложняется при необходимости размещения разногабаритных элементов на плате при нефиксированных позициях для установки элементов.

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. (*а*)) или в одну позицию может быть помещено несколько элементов (рис. (*б*)).



а



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.

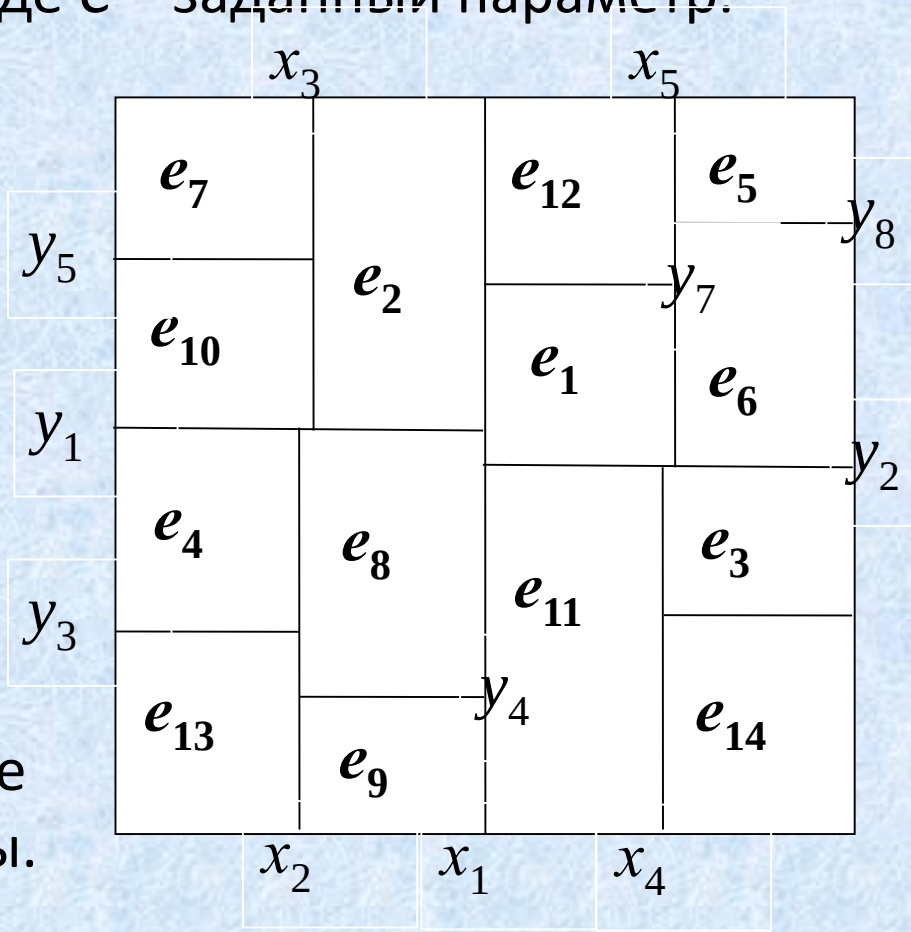
Рассмотрим алгоритм нисходящего проектирования размещения разногабаритных элементов, основанный на последовательном разбиении КП. Алгоритм является вариантом алгоритма дихотомического деления.

Пусть задано множество элементов $E=\{e_1, e_2, \ldots, e_m\}$ с размерами l_x^i и l_y^i. Вначале рассчитываются размеры поля для размещения элементов

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.

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.

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. приведены основные критерии трассировки в зависимости от технологии изготовления.

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

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

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:

Что? (Определение перечня соединений);

Где? (Распределение соединений по слоям);

Когда? (Определение порядка проведения соединений);

Как? (Трассировка проводников).

А. 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).

Алгоритмы построения минимальных связывающих деревьев

Для построения МСД разработан ряд полиномиальных алгоритмов.

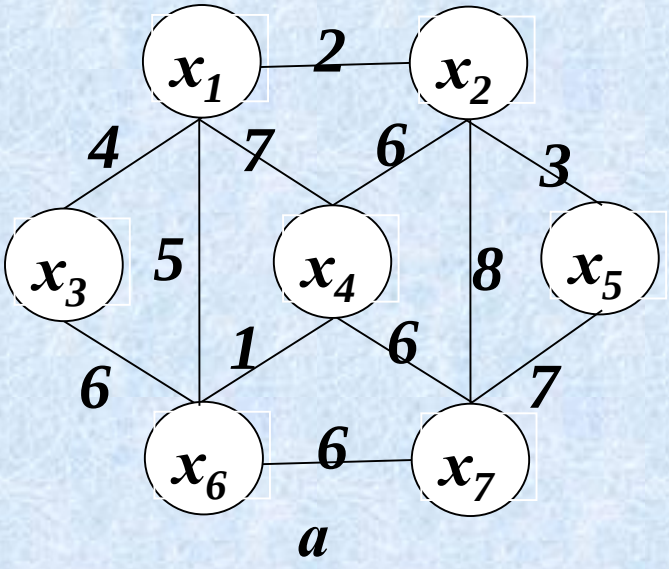
Алгоритм Прима

Алгоритм Прима (*R.C. Prim*) 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.

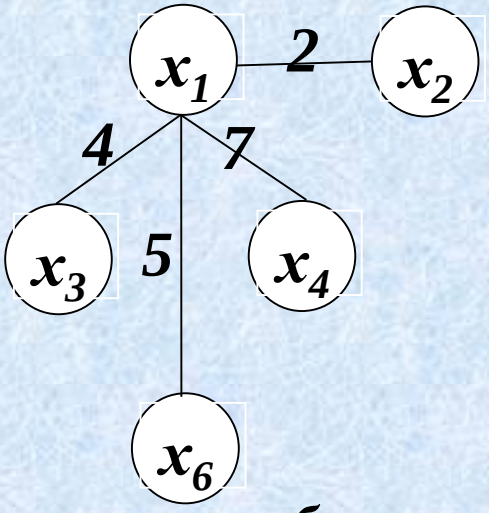
Начнем с произвольной вершины графа x_i и включим ее в остовное дерево. Из всех вершин, соединенных с данной ($x_j \in Гx_i$), ищем вершину, соединенную ребром с наименьшим весом.

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 рисунке.



а



б

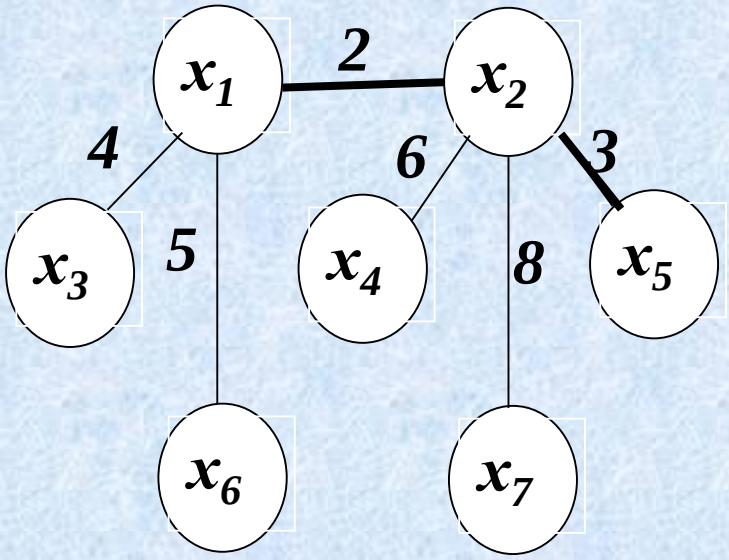
В начале процесса выбираем произвольную вершину, например, x_1. Выбираем вершины, непосредственно связанные с x_1 .

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

Наименьший вес из пяти ребер - кандидатов имеет ребро $(x_2\ x_5)$, поэтому к дереву нужно добавить его и вершину x_5

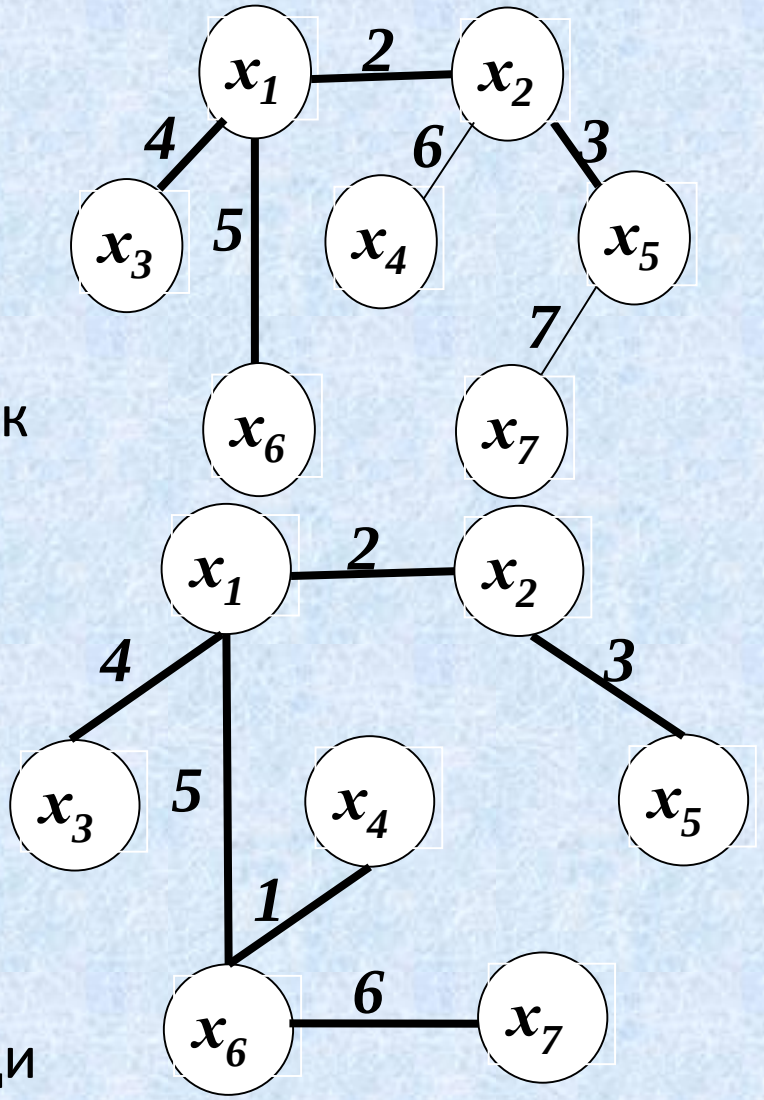


Вес ребра ($x_5\ x_7$) меньше веса ребра ($x_2\ x_7$), поэтому оно замещает последнее.

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

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.



Ребро ($x_4\ x_6$) имеет наименьший вес среди оставшихся, и оно добавляется следующим.

Теперь осталось добавить к дереву всего одно ребро ($x_6\ x_7$)

После этого процесс завершается, построено МСД с суммарным весом ребер 21.

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

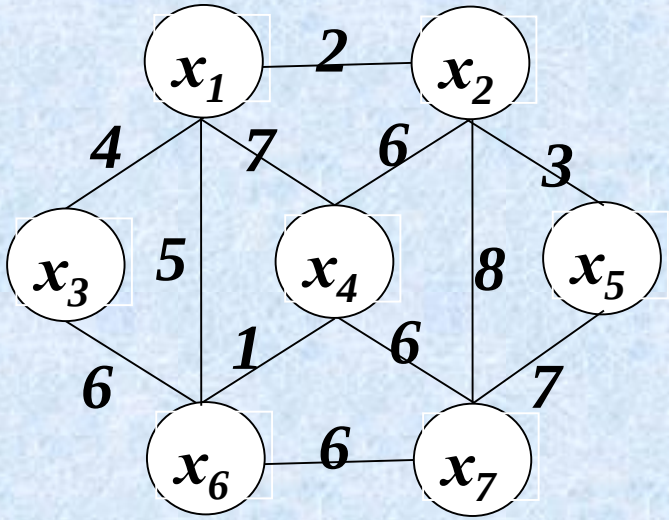
Алгоритм Краскала

Алгоритм заключается в следующем.

Упорядочим все ребра исходного графа по не убыванию 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.

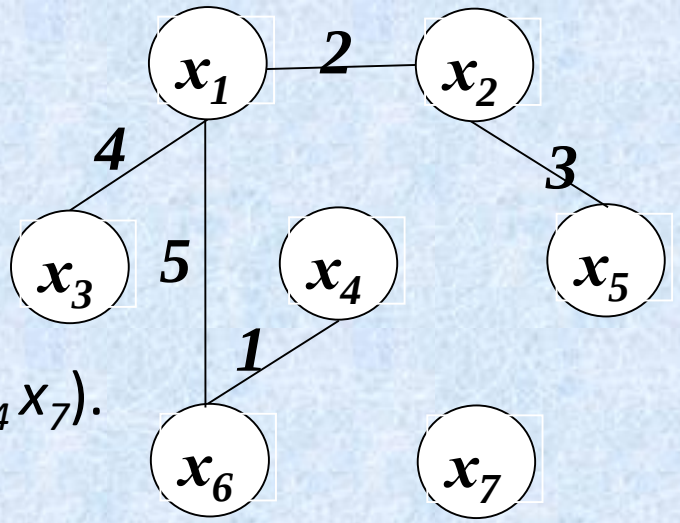
Построим МСД для графа



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

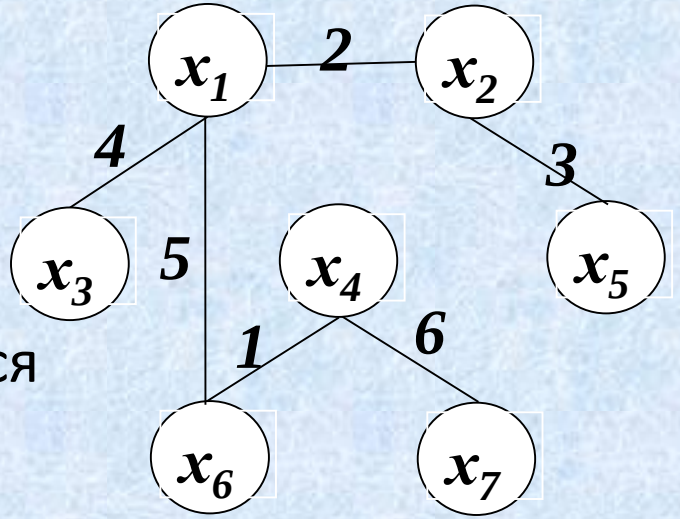
Ребра ($x_2 x_4$) и ($x_3 x_6$) образуют цикл с уже построенными ребрами, поэтому в дерево не включаются.



Следующее включаемое в дерево ребро ($x_4 x_7$). Дерево построено.

Суммарный вес ребер МСД равен 21.

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



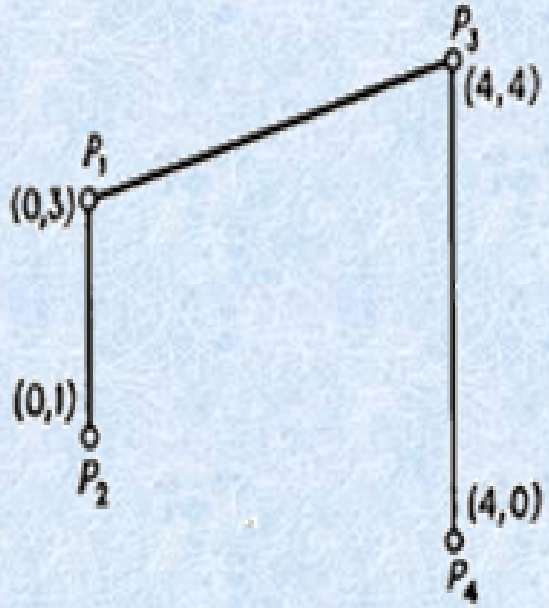
Задача Штейнера

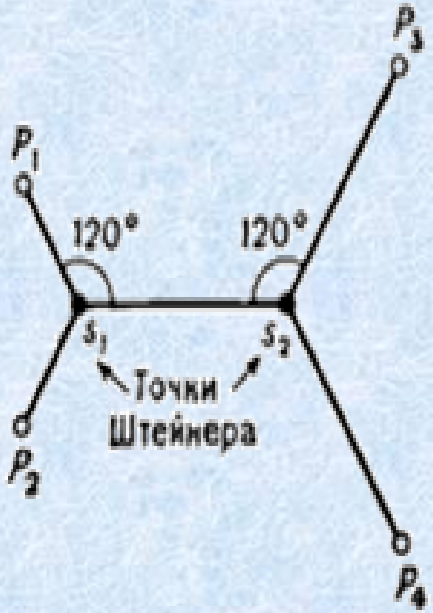
Виды используемых деревьев определяются технологией выполнения соединений и схемотехническими требованиями. При автоматизированном конструировании схем проводного и печатного монтажа ставятся разные задачи построения минимальных деревьев соединений.

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 ребер (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 (точки Штейнера) и суммарный вес ребер в T' будет минимальным.

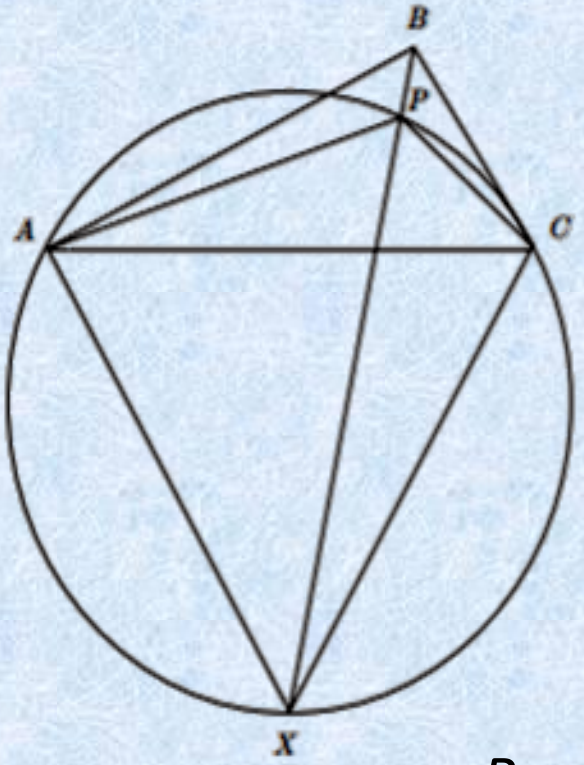




У МСД суммарная длина ребер - 10,123, а у дерева Штейнера суммарная длина ребер – 9,196.

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.

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, описанной 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.

Трехточечная задача используется как составная часть алгоритмов решения задачи Штейнера.

Задача Штейнера относится к классу так называемых NP-полных задач, поэтому алгоритмы, дающие точные решения не могут быть использованы в САПР из-за неприемлемой временной сложности.

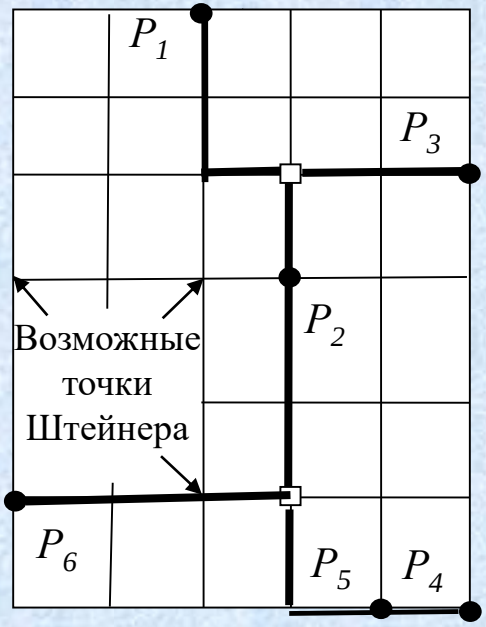
Это обстоятельство послужило стимулом для разработки многочисленных эвристических алгоритмов. Наибольший практический интерес представляет алгоритм последовательного введения дополнительных вершин в дерево Прима-Краскала.

Одним из важных практических применений задачи Штейнера является конструирование интегральных электронных схем. Более короткая сеть проводящих линий на интегральной схеме повышает 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.

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. 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.

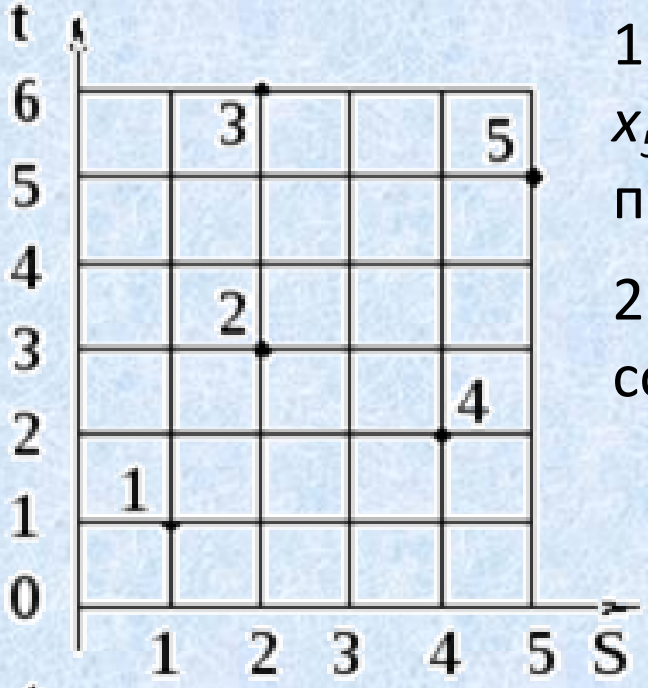
□ Точки Штейнера



□ Точки Штейнера

Метод Ханана

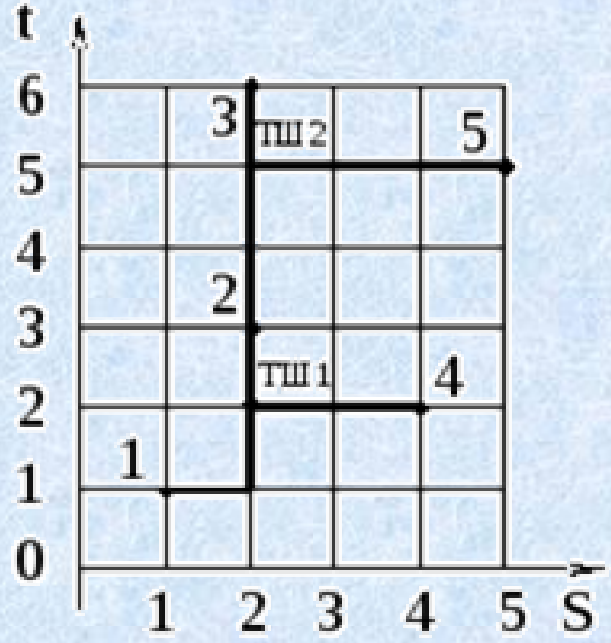
1. Все множество вершин с координатами s разбивается на классы $S_1, S_2, \ldots, S_m$ в порядке не убывания координаты s_i так, чтобы у всех вершин одного класса была одинаковая координата s.
2. Анализируется множество вершин класса S_1 и соединяется между собой прямой.
3. Образуется множество I, состоящее 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.



1. Разобьем множество вершин $X = \{x_1, x_2, ..., x_5\}$ на классы по не убыванию 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.

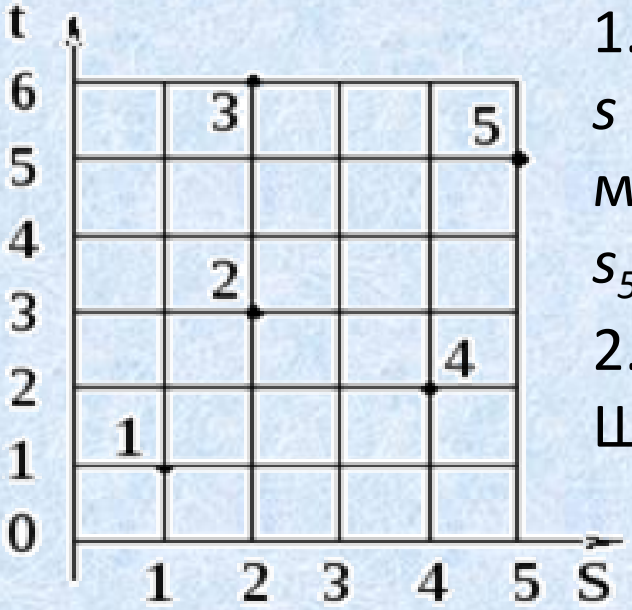
6. Вершину $x_5 \in S_4$ соединяем по кратчайшему расстоянию с ближайшей точкой построенного дерева, получаем ТШ 2.

Построено дерево Штейнера с суммарной длиной ребер L=11

Столб Штейнера

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

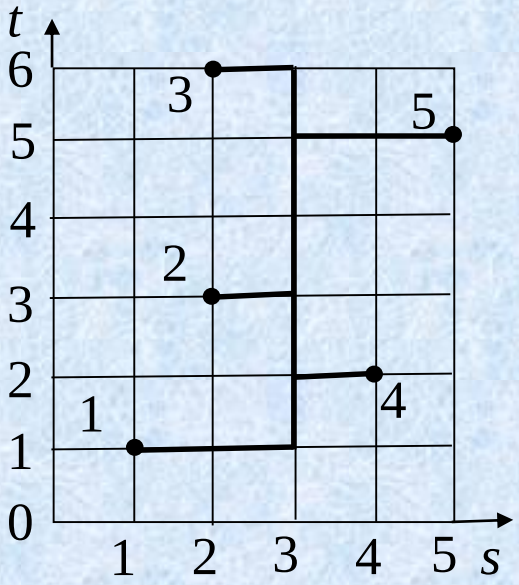
1. Все вершины проецируются на ось *s*.
2. Расстояние от наименьшей до наибольшей координаты *s* делится пополам, и из этой точки проводится перпендикуляр (столб Штейнера).
3. Из каждой вершины опускается перпендикуляр до пересечения со столбом Штейнера.

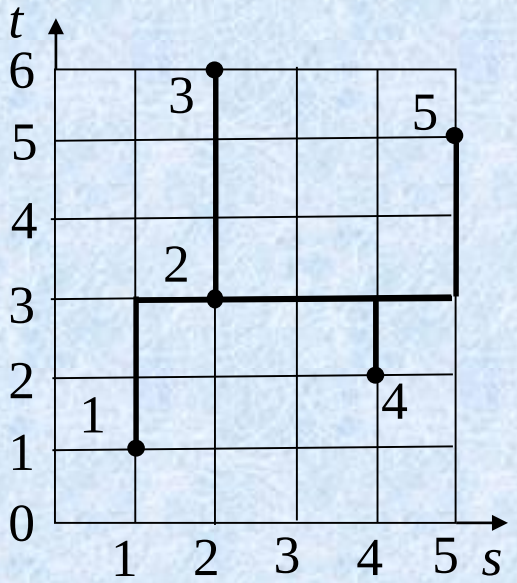


1. Находим среди множества проекций на ось *s* (s_1=1, s_2= s_3=2, s_4=4, s_5=5) минимальное и максимальное значения: $\min s_i = s_1 = 1$, $\max s_i = s_5 = 5$.
2. Вычисляем координату проведения столба Штейнера: $s_i = (s_1 + s_5) / 2 = 3$.

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.





Аналогично строится горизонтальный столб, L=12.

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.

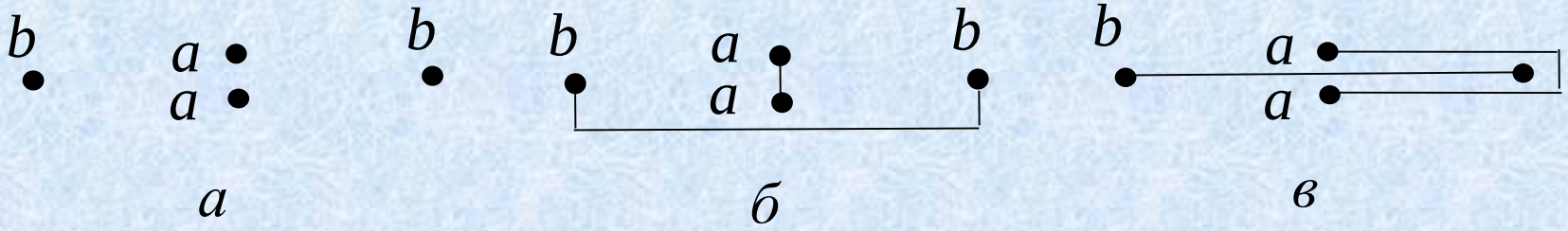
Если расслоение проводится после трассировки, то получают совмещенную топологию. Строят граф пересечений, в котором вершины – проводники, а ребра их соединяют, если проводники пересекаются. Граф пересечений раскрашивается в минимальное число цветов.

Расслоение в процессе трассировки будет рассмотрено позже.

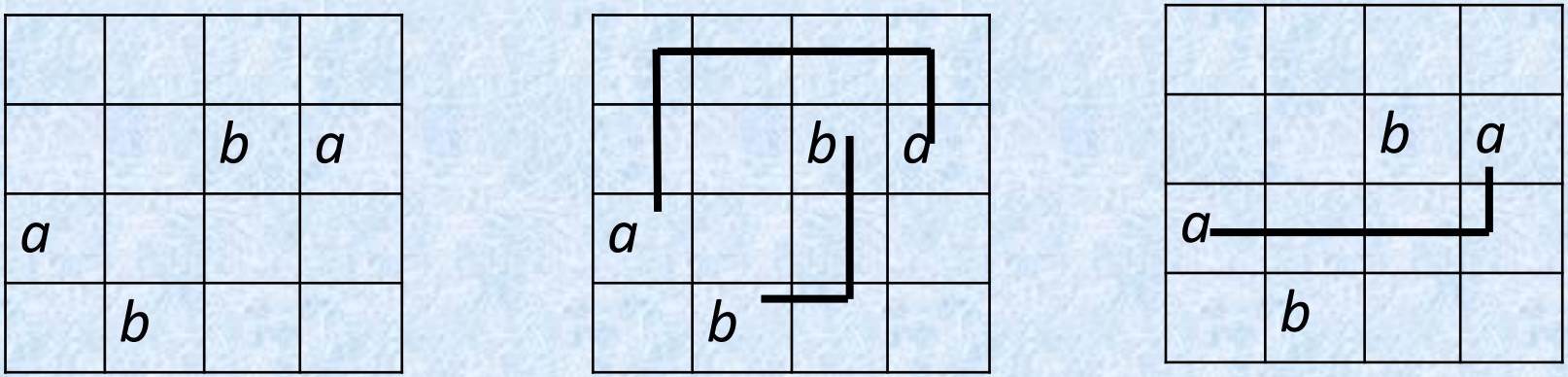
С. На третьем этапе определяется последовательность проведения соединений, т. е. указывается, когда каждый проводник будет проведен.

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.



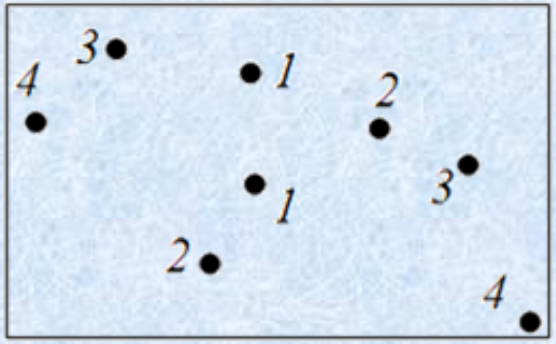
а *б* *в*

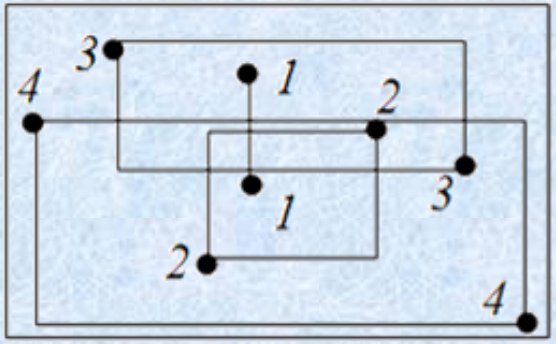


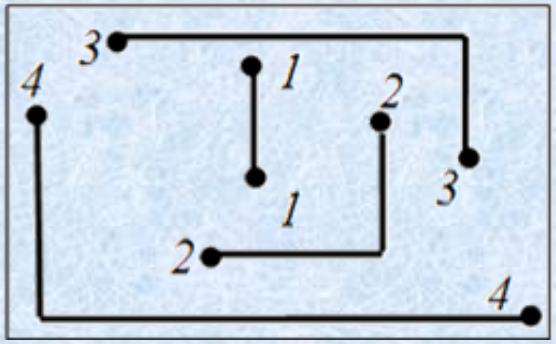
Идея алгоритмов определения порядка проведения соединений заключается в том, что первыми проводятся проводники, меньше «мешающие» другим проводникам.

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 не убыванию этих чисел;
4. Получен порядок проведения соединений.







1. Строим минимальные охватывающие прямоугольники;
2. Число контактов других соединений, попавших в данный прямоугольник *1* – 0; *2* – 1; *3* – 2; *4* – 4;
3. Упорядочим соединения *1, 2, 3, 4*; Проводим соединения.

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.

На четвертом этапе дается ответ, каким образом (каким алгоритмом) должно быть проведено каждое соединение.

Трассировка соединений

Это основной и наиболее трудоемкий этап, определяющий эффективность и качество трассировки в целом.

Алгоритмы трассировки можно разбить на следующие группы:

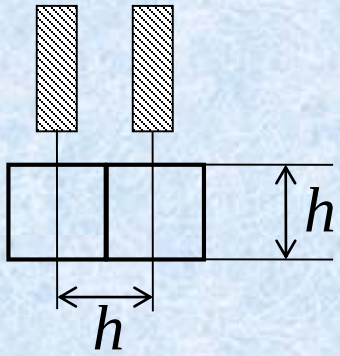
1. Волновые;
2. Лучевые;
3. Канальные.

Волновой алгоритм трассировки

Многие алгоритмы трассировки основаны на волновом алгоритме (алгоритме, предложенном Ли (Lee C.Y.)).

Коммутационное поле разбивается на дискреты. Размеры дискретов определяются шириной проводников и расстоянием меду ними.

Получаем дискретное рабочее поле (ДРП).



Волновой алгоритм состоит из двух этапов:

1. Распространение волны;
2. Проведение трассы.

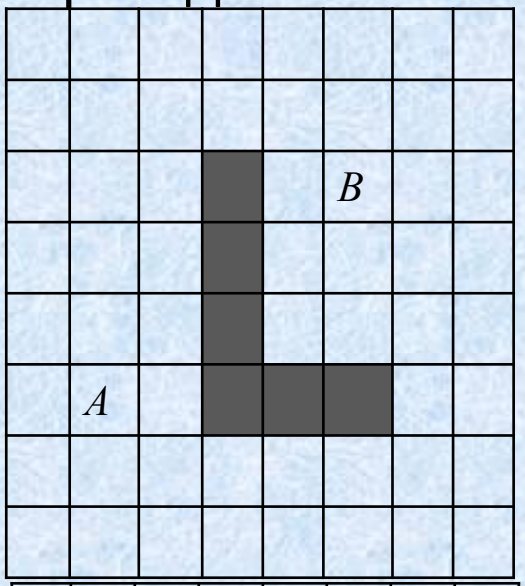
В первой части моделируется процесс распространения волны от источника ячейки A по свободным ячейкам ДРП. Алгоритм последовательно строит $\Phi_1(A)$, $\Phi_2(A)$, ..., $\Phi_k(A)$ – 1, 2, ..., k фронты. Множество ячеек ДРП, входящих в i-ый фронт A называется i-й окрестностью ячейки A – $O_i(A)$. Если 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.

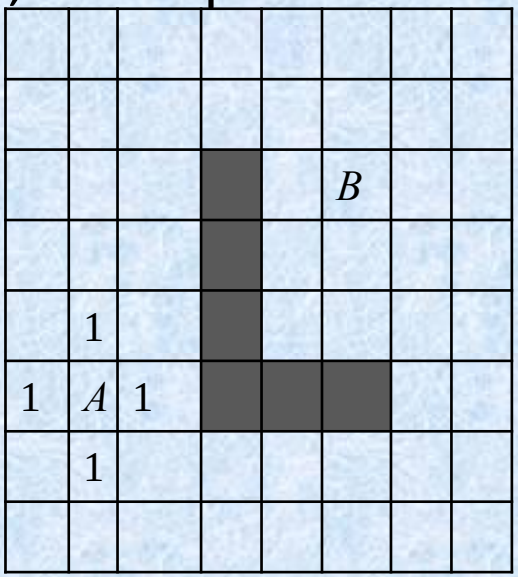
Если же в следующий фронт не удается включить ни одной ячейки, т. е. $O_{i-1}(A) = O_i(A)$, то пути между A и B не существует.

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.





	4						
4	3	4	■		*B*		
3	2	3	■				
2	1	2	■				
1	*A*	1	■	■	■		
2	1	2	3	4			
3	2	3					

6	5	6	7				
5	4	5	6	7			
4	3	4	■		*B*		
3	2	3	■				
2	1	2	■				
1	*A*	1	■	■	■	7	
2	1	2	3	4	5	6	7
3	2	3	4	5	6	7	

6	5	6	7	8	9		
5	4	5	6	7	8	9	
4	3	4	■	8	*B*		
3	2	3	■	9		9	
2	1	2	■		9	8	9
1	*A*	1	■	■	■	7	8
2	1	2	3	4	5	6	7
3	2	3	4	5	6	7	

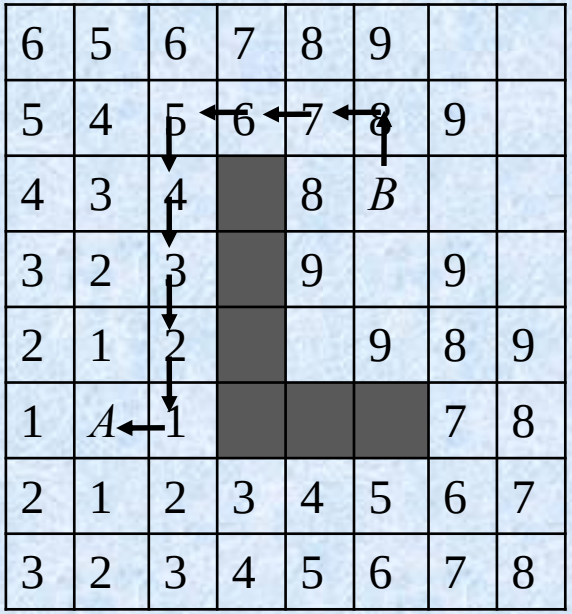
На девятом шаге волна достигла ячейку *B*.

При построении пути возможны два выхода из ячейки *B*.

Направление движения определяется правилом приоритетов – вес какой соседней ячейки проверяется.

6	5	6	7	8	9		
5	4	5	6	7	8	9	
4	3	4		8	*B*		
3	2	3		9		9	
2	1	2			9	8	9
1	*A*	1				7	8
2	1	2	3	4	5	6	7
3	2	3	4	5	6	7	8

Пусть ячейки проверяются в следующем порядке: ↑→↓←.



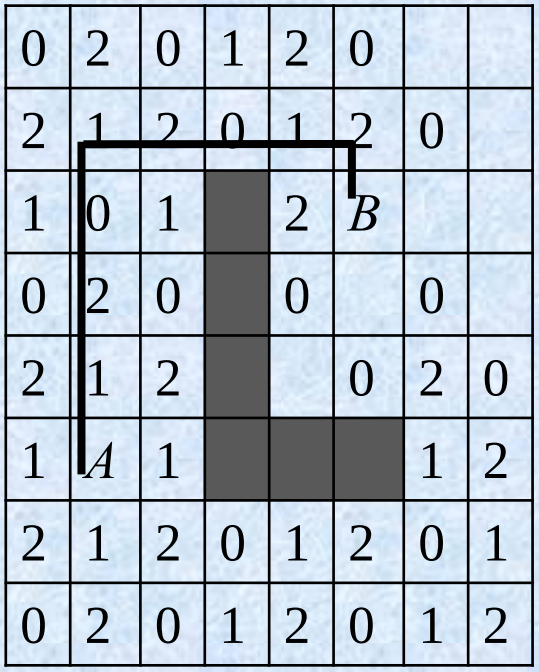
Вычислительная сложность волнового алгоритма близка к $O(n^2)$.

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 число.

Поэтому при реализации волнового алгоритма важная проблема – сокращение объема памяти, необходимой для хранения ДРП.

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

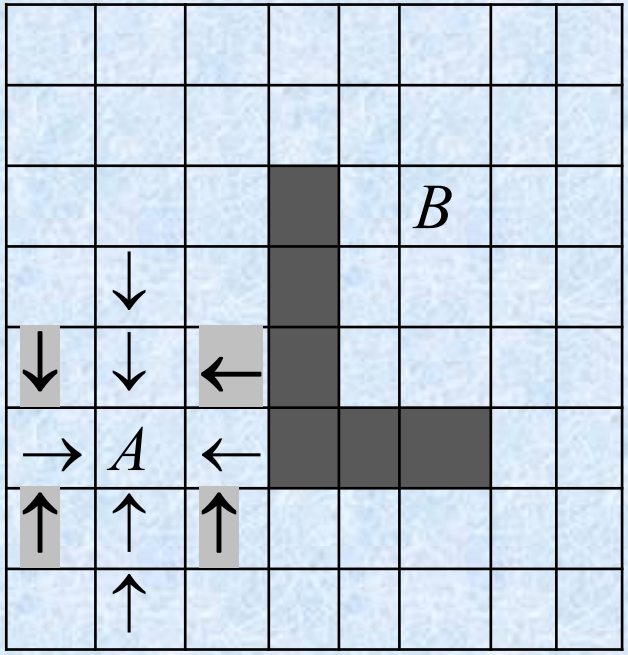


Для проведения пути необходимо выбирать последовательность 0, 1, 2 в обратном порядке. При кодировании по *mod* 3 каждая ячейка ДРП в процессе работы может быть в одном из следующих состояний: свободная, занятая или содержать весовую метку 0, 1, 2. Таким образом, требуемое число разрядов памяти на одну ячейку ДРП составляет $N = \lceil \log_2(5) \rceil = 3$. Важно, что N не зависит от длины пути.

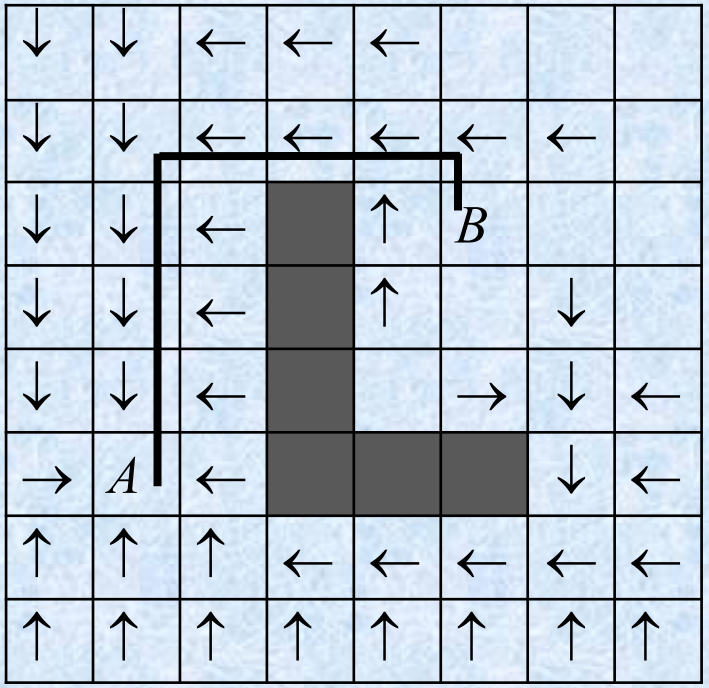
Метод путевых координат

Путевой координатой ячейки c_i фронта $Ф_k$ будем называть ту, соседнюю с ней ячейку фронта $Ф_{k-1}$, от которой она получает свой вес. Для рассматриваемой окрестности соседства имеем четыре возможные путевые координаты $\uparrow, \leftarrow, \downarrow, \rightarrow$. Если имеется несколько соседних с c_i ячеек фронта $Ф_{k-1}$, то назначение путевой координаты производится согласно выбранному правилу приоритетов, т.е. порядку просмотра соседних ячеек.

Этап проведения пути состоит в отслеживании путевых координат в размеченном ДРП, начиная от ячейки - приемника *B*



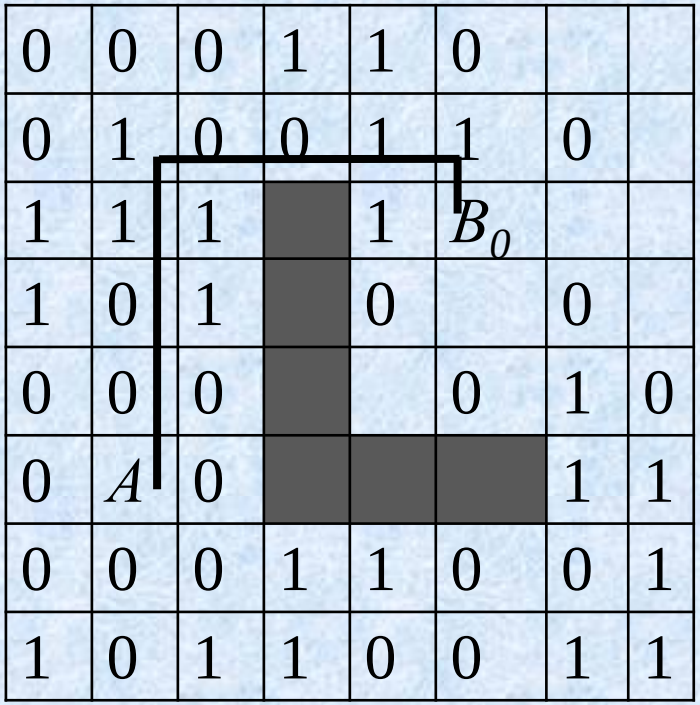
$\uparrow \leftarrow \downarrow \rightarrow$



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 $\uparrow, \leftarrow, \downarrow, \rightarrow$. Таким образом, требуемое число разрядов памяти на одну ячейку ДРП составляет $N = \lceil \log_2(6) \rceil = 3$.

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 (Akers S. B) предложил последовательность 00110011..., которая обладает этим свойством.



Для построения пути необходимо выбрать последовательность в обратном порядке.

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 $N = \lceil \log_2(4) \rceil = 2$.

Оптимизация пути по нескольким параметрам

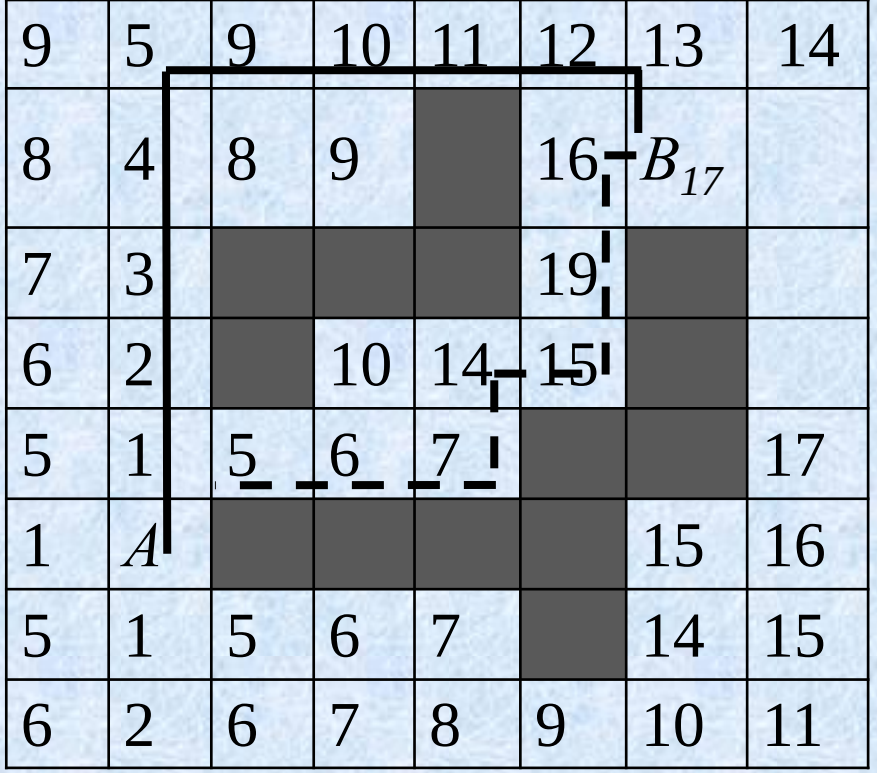
Волновой алгоритм может одновременно учитывать несколько критериев: длина пути, число перегибов, число переходов со слоя на слой и т.д. Тогда вес ячейки *k*-го фронта будет вычисляться по формуле $p_k = p_{k-1} + \sum_{i=1}^{n} a_i f_i(k)$, где a_i - весовой коэффициент, учитывающий важность *i*-го критерия; $f_i(k)$ – значение *i*-го параметра для рассматриваемой ячейки; *n* –количество учитываемых критериев.

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, $p_k = p_{k-1} + 1 + 3p_2$.

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



На втором шаге алгоритма в ячейку после изгиба ставится метка 5, 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.

Построен путь с двумя перегибами и длиной – 11. Минимальная возможная длина - 9 при пяти перегибах (пунктирный путь).

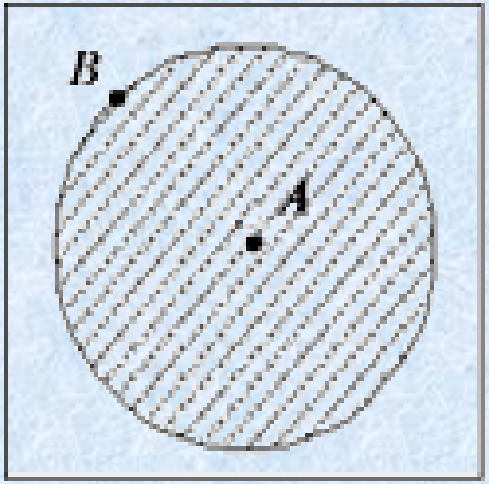
Волновой алгоритм характеризуется высокой эффективностью нахождения пути, но требует значительных временных затрат. Причем 90% времени тратится на распространение волны и 10% - на проведение пути.

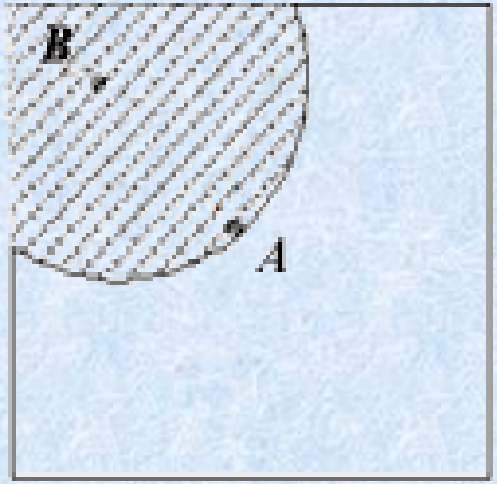
Методы повышения быстродействия волнового алгоритма

Методы сокращают время распространения волны. Это методы:

1. Выбор источника волны.

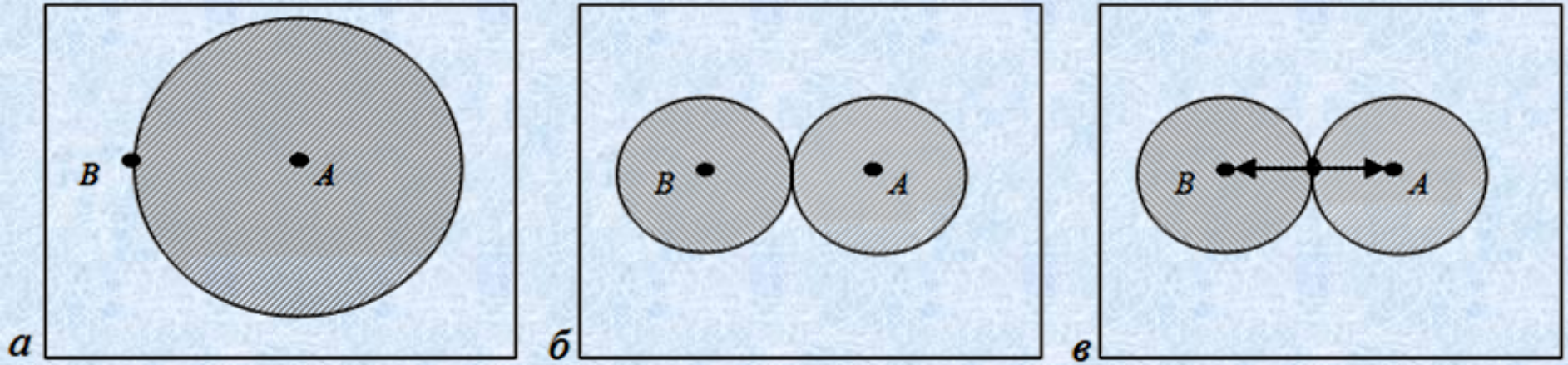
Время распространения волны пропорционально площади заштрихованных фигур. В качестве источника волны выбирается ячейка, ближе расположенная к краю КП.





2. Метод встречной волны.

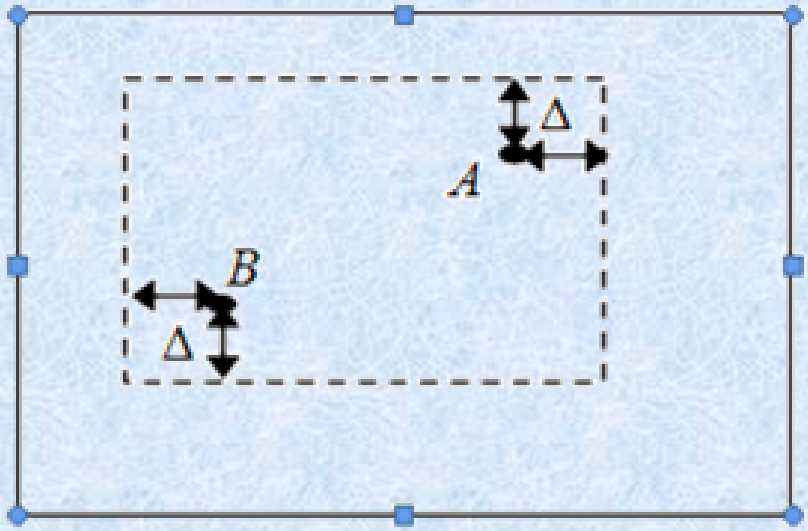
Волна попеременно распространяется из двух источников (*A* и *B*) до тех пор, пока волны не встретятся. Путь строится, начиная с ячейки встречи, до источников.



Площадь заштрихованной фигуры рис. (*а*) $S_1 = \pi L^2_{AB}$, а фигуры рис. (*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.

3. Обрамление соединяемых точек.

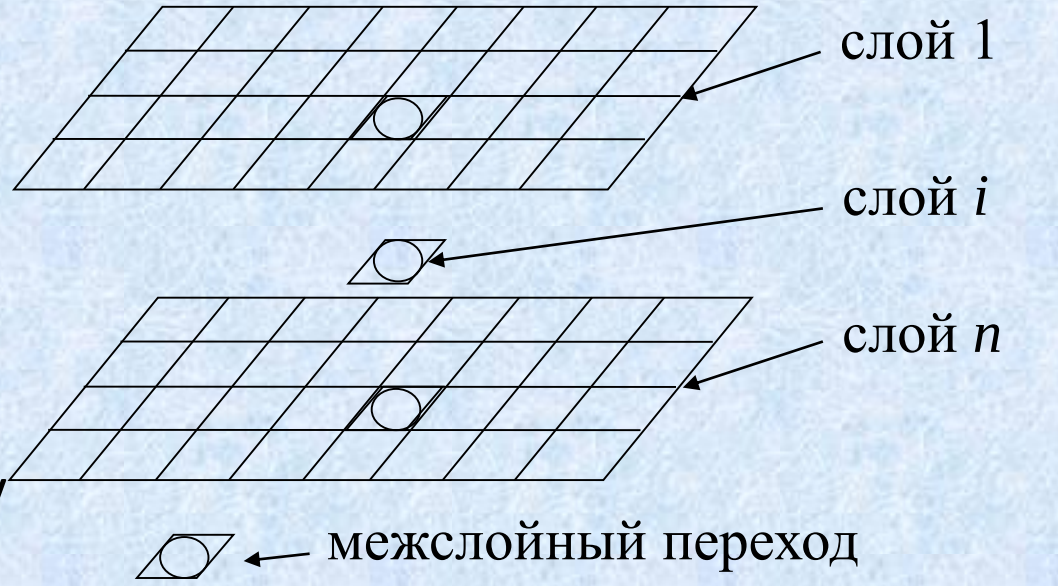
Путь ищется не во всем коммутационном поле а в окрестности ячеек *A* и *B*



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.

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

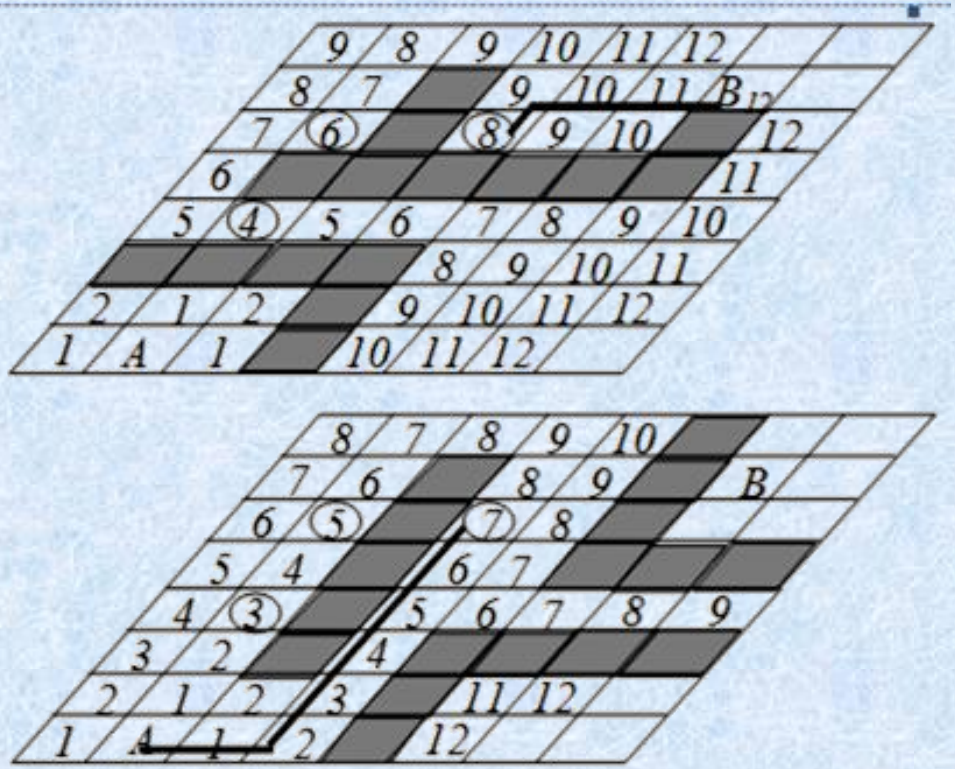


Одним из таких алгоритмов является алгоритм многослойной трассировки Хейса (S. Heiss). При распространении волны свободным ячейкам ДРП$_i$ присваивается индекс длины пути p_i и индекс количества межслойных переходов v_i.

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



В алгоритме Хейса волна распространяется независимо в каждом слое, что приводит к значительным затратам времени и памяти при реализации.

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.