4 Методы решения комбинаторно-оптимизационных задач

4.1 Стратегии декомпозиции пространства решений и дерево поиска

Можно выделить два основных подхода к поиску решения комбинаторно-оптимизационных задач.

Первая из них основана на двух идеях:

- *декомпозиция* пространства решений;
- *исследование* множества возможных решений, т. е. поиск оптимального на основе некоторой оценки в процессе декомпозиции.

Вторая стратегия реализует следующие три идеи:

- *разбиение* задачи на подзадачи;
- *определение* оптимального варианта подзадач;
- *получение* решения композицией (объединением) этих подзадач.

Декомпозиция пространства решений

Множество возможных вариантов решения M разбивается в соответствии с принципом формирования результата на подмножества M_i такие, что $\cup M_i = M$. Далее, используя тот же принцип, полученные подмножества вновь разбиваются на подмножества M_j , включающие в себя меньшее количество вариантов. После некоторого шага *k* разбиения каждое подмножество содержит по одному варианту решения.

Сопоставим каждому подмножеству вершину графа. Соединив ребром вершины, соответствующие подмножествам M_i и M_j, если подмножество M_j получено непосредственным разбиением подмножества M_i, придем к так называемому *дереву решений*.

Декомпозиция пространства решений

Принцип разбиения пространства решений связан с видом преобразований, которые необходимо выполнить над графом исходного описания для получения графа результата.

Существует *две стратегии декомпозиции* пространства решений, отличающиеся порядком получения вершин дерева решений:

- в ширину;
- в глубину с возвращением.

M_i^1

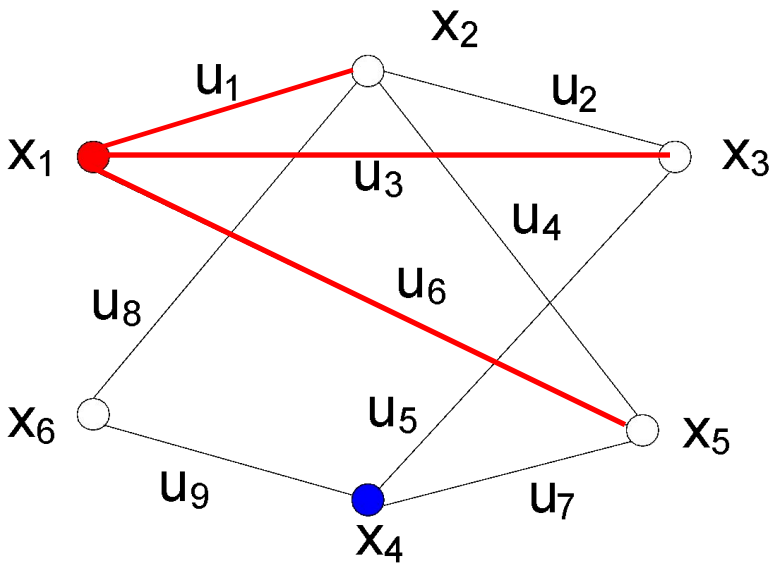
Декомпозиция в ширину

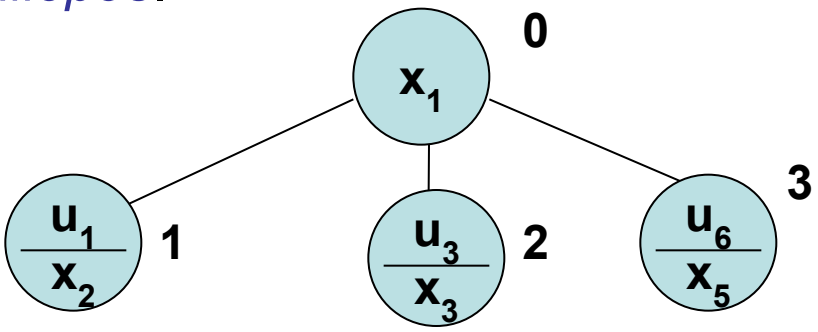
Все множество возможных вариантов решения M разбивается на подмножества первого уровня M_i^1 такие, что $\cup M_i^1 = M$. Затем каждое подмножество первого уровня M_i^1 разбивается на подмножества второго уровня M_j^1 такие, что $\cup M_j^1 = M_i^1$. Процесс продолжается до тех пор пока каждое подмножество не будет соответствовать одному варианту решения.

Декомпозиция в ширину

Рассмотрим задачу поиска всех простых цепей (маршрутов) из вершины x_1 в вершину x_4 в графе $G(X,U)$.

Принцип формирования маршрутов: начиная с вершины x_1, будем включать в маршруты ребра *в порядке возрастания их номеров*.





Добавляемое ребро

Достигнутая вершина

Вершина дерева решений с номером 1 соответствует включению в маршрут ребра u_1, вершины с номерами 2 и 3 – ребер u_3 и u_6.

Переход на следующий уровень дерева решений выполняется только после того, как *были получены все подмножества текущего уровня*.

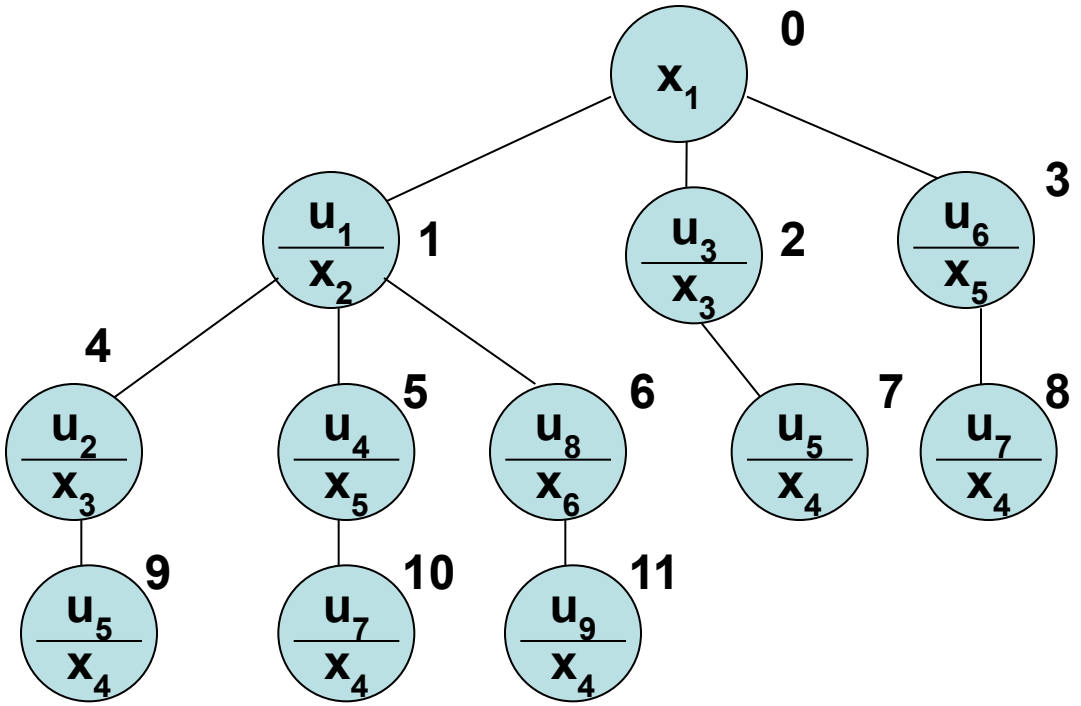
Декомпозиция в ширину

Таким образом на первом шаге (на 2-ом уровне дерева решений) множество *M* разбивается на три непересекающихся подмножества вариантов, одно из которых M_1, соответствующее вершине 1 дерева решений, порождает следующие три варианта простых цепей графа *G*:

- $\{u_1, u_2, u_5\}$,
- $\{u_1, u_4, u_7\}$,
- $\{u_1, u_8, u_9\}$.

Далее формируются вершины следующего третьего и т. д. уровней.

Процесс заканчивается после получения всех вариантов решения.



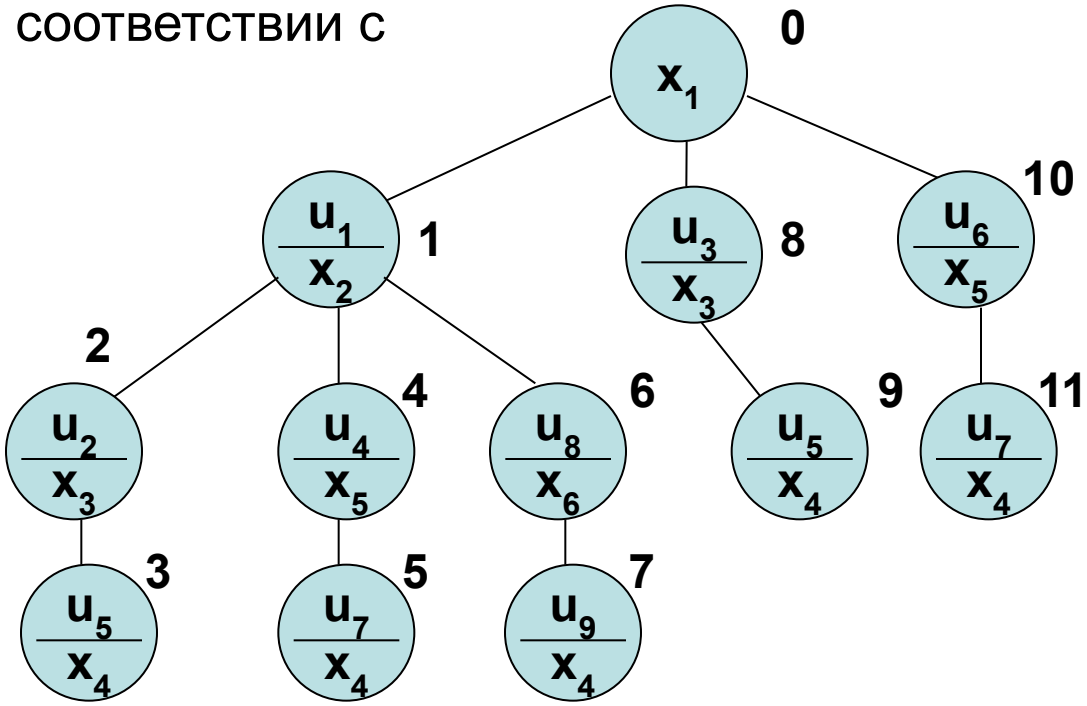
Описанный процесс реализует декомпозицию множества вариантов решения по *методу в ширину* с последовательным формированием состава вариантов и позволяет реализовать поиск решения *полным перебором*.

Декомпозиция в глубину с возвращением

Все множество возможных вариантов решения M разбивается на подмножества M_1 и $M \setminus M_1$. Затем подмножество M_1 разбивается на подмножества M_2 и $M_1 \setminus M_2$. Продвигаемся по этой ветви пока не получим подмножество, соответствующее одному варианту решения. Возвращаемся по ветви вверх, пока не придем в вершину, из множества вариантов которой можно выделить некоторое подмножество в соответствии с принятым принципом. Начиная с этого подмножества строим следующую ветвь, пока не получим подмножество, соответствующее одному варианту решения. Процесс повторяется, пока не будут получены все варианты.

Декомпозиция в глубину с возвращением

Выделим на 1-ом шаге из множества M подмножество M_1, включив в маршрут ребро u_1 из возможных $\{u_1, u_3, u_6\}$ в соответствии с минимальным индексом.



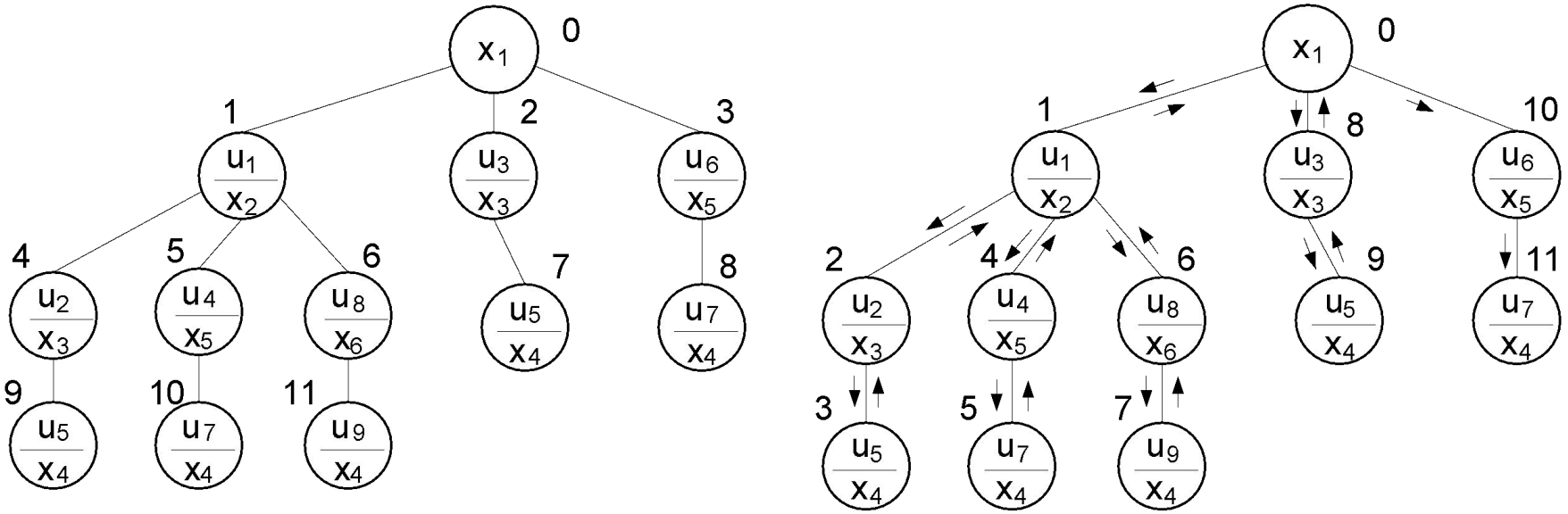
Таким образом разобьем множество M на подмножества M_1 и $M \setminus M_1$. Множество M_1 разобьем на подмножества M_2 и $M_1 \setminus M_2$, добавив в маршрут ребро u_2.

Продвигаясь по этой ветви получим вариант маршрута – x_1, x_2, x_3, x_4 или в виде последовательности ребер – u_1, u_2, u_5.

Возвращаемся по ветви вверх, пока не придем в вершину, из множества вариантов которой можно выделить некоторое подмножество в соответствии с принятым принципом и получаем следующий вариант.

Процесс повторяется, пока не будут получены все варианты.

Стратегии декомпозиции пространства решений



Отметим, что декомпозиция в ширину и в глубину с возвращением приводит к получению всех вариантов решения задачи, которые формируются последовательно, но в разном порядке. При поиске всех вариантов это не играет роли, однако может быть существенным при поиске одного решения в процессе декомпозиции. Например при решении задачи методом ветвей и границ, это может существенно влиять на количество анализируемых вариантов.

Стратегии декомпозиции пространства решений

Рассмотренные стратегии лежат в основе решения многих задач дискретной математики. Укажем некоторые из них.

- *Построение глубинного или d-дерева* (просмотра в глубину). Является частным случаем построения дерева декомпозиции в глубину с возвращением, а именно – дерева *просмотра* вершин графа. Вершины и ребра графа включаются в дерево (просматриваются) один раз. Глубинное или *d*-дерево строится с заданной начальной вершины – корня, затем среди смежных ей, а в дальнейшем смежных последней включенной в дерево, берется первая еще не просмотренная вершина и включается в строящуюся ветвь. Процесс продолжается в соответствии с описанной выше стратегией декомпозиции в глубину с возвращением.

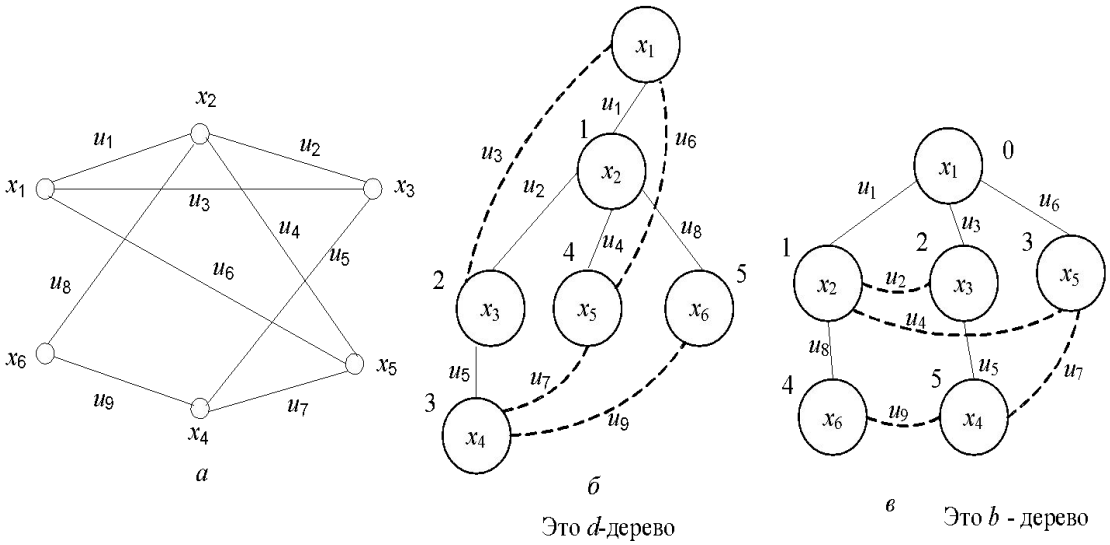
Глубинное или *d*-дерево

Вершина $x_j \in F_1 x_i$ называется *потомком* вершины x_i, сама вершина x_i – *отцом* (предком) вершины x_j, а ребро $u(x_i, x_j)$ – *древесным*. Вид глубинного дерева зависит от начальной вершины и порядка их перечисления в образах относительно предиката смежности $F_1 X = \{F_1 x_i / x_i \in X\}$. Для одной компоненты связности в виде неориентированного графа $G^{\sim}(X, U)$ глубинное дерево является остовным.

Количество потомков корневой вершины x_r равно $|F_1 x_r|$, а всех остальных вершин x_i – $|F_1 x_i \setminus X_d|$, где X_d – множество вершин, уже включенных в строящееся дерево.

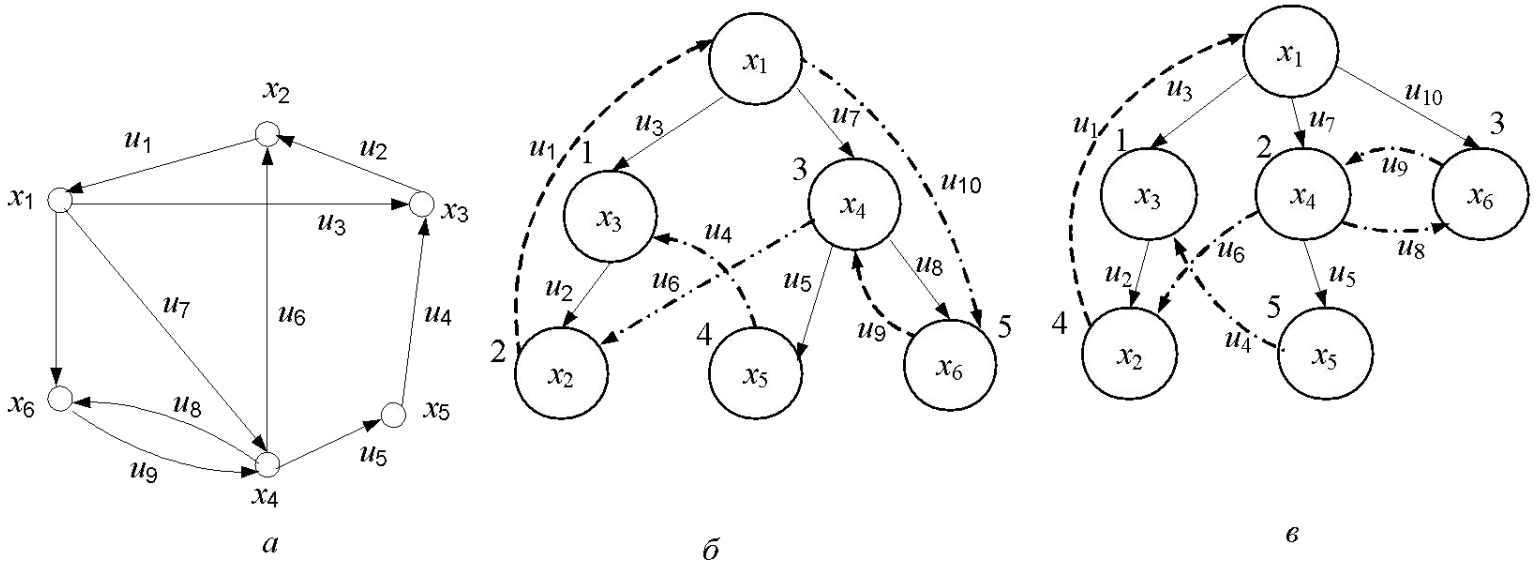
Глубинное или *d*-дерево

На рисунке (*б)* показано глубинное дерево неориентированного графа $G\tilde{}(X, U)$ (*а*) для случая, когда вершины в образах, т. е. в подмножествах $X_i = F_1 x_i$, перечислены в порядке возрастания номеров (их индексов в записи множества X). *Сплошными линиями изображены древесные ребра – U_T*, а *пунктирными – обратные $U_D = U \setminus U_T$*.



Глубинное или *d*-дерево ориентированного графа

Глубинное дерево может строиться и в ориентированном графе. На рис. *а* и *б* изображен ориентированный граф $G^{\rightarrow}(X, U)$ и его глубинное (ориентированное корневое) дерево. Построение этого дерева вводит на множестве вершин ориентированного графа отношение частичного порядка: $x_i < x_j$, если *xi* является предком вершины x_j.



Глубинное или *d*-дерево ориентированного графа

В этом дереве дуги графа разбиты на четыре класса:

- Древесные дуги, идущие от отца к потомку (u_2, u_3, u_5, u_7, u_8);
- Обратные, идущие от потомка к предку (u_1, u_9);
- Прямые, идущие от отца к потомку, но не являющиеся древесными (u_{10});
- Поперечные, соединяющие вершины, ни одна из которых не является потомком другой (u_4, u_6).

Отношение частичного порядка на множестве вершин ориентированного графа позволяет на основе построения глубинного остовного дерева решать, например, такие задачи как:

- распознавание сильной связности ориентированного графа, т. е. существования в нем пути из каждой вершины в любую другую [Асанов];
- отыскание блоков, мостов и расщепляющих вершин [Кормен];
- установления ацикличности ориентированного графа [Кормен];
- отыскание всех гамильтоновых циклов графа [Асанов].

Дерево поиска в ширину (*b*-дерево)

Является частным случаем построения дерева декомпозиции в ширину, а именно – дерева *просмотра* вершин графа. Вершины и ребра графа включаются в дерево (просматриваются) один раз. Это дерево строится в соответствии с рассмотренной выше стратегией декомпозиции в ширину. Назначается некоторая исходная вершин *xr* – корень дерева. Первый уровень дерева просмотра составляют вершины, смежные корню, а все следующие – вершины-потомки вершин предыдущего уровня. Количество вершин-потомков определяется по тем же выражениям, что и для поиска в глубину.

Дерево может быть построено как для неориентированного, так и для ориентированного графа. Вид дерева зависит от назначения вершины-корня и порядка перечисления вершин в образах вершин-предков. Дерево содержит все достижимые из корня вершины.

Дерево поиска в ширину (*b*-дерево)

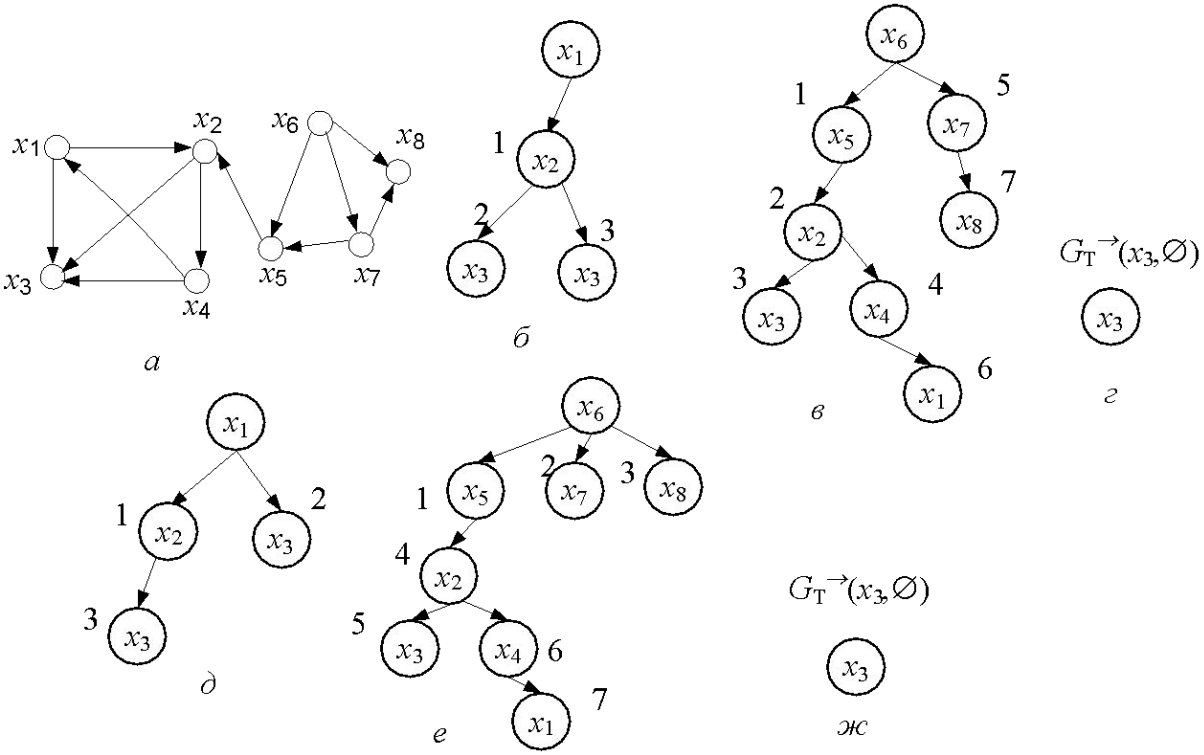
Для каждой вершины путь до нее из корня является одним из кратчайших, в смысле числа включенных в него ребер, путей в графе. Для неориентированного графа *b*-дерево – остовное. Выше были показаны *b*-деревья неориентированного и ориентированного графов соответственно.

Просмотр графа в ширину составляет основу алгоритмов решения таких задач как, например:

- определение длины кратчайшего пути (в указанном выше смысле) от некоторой вершины до каждой из достижимых [Кормен];
- построение остовного дерева минимального веса [Хидет.] и др.

Особенности просмотра в глубину с возвращением и в ширину в ориентированном графе

В ориентированных графах *d* - и *b*-деревья не обязательно являются остовными. Вид дерева зависит от структуры графа и начальной вершины. Как видно из рисунка в зависимости от начальной вершины в данном графе могут быть получены как остовные деревья, так и тривиальные $G^{\rightarrow}(x, \varnothing)$.



Вычислительная сложность просмотра в глубину с возвращением и в ширину

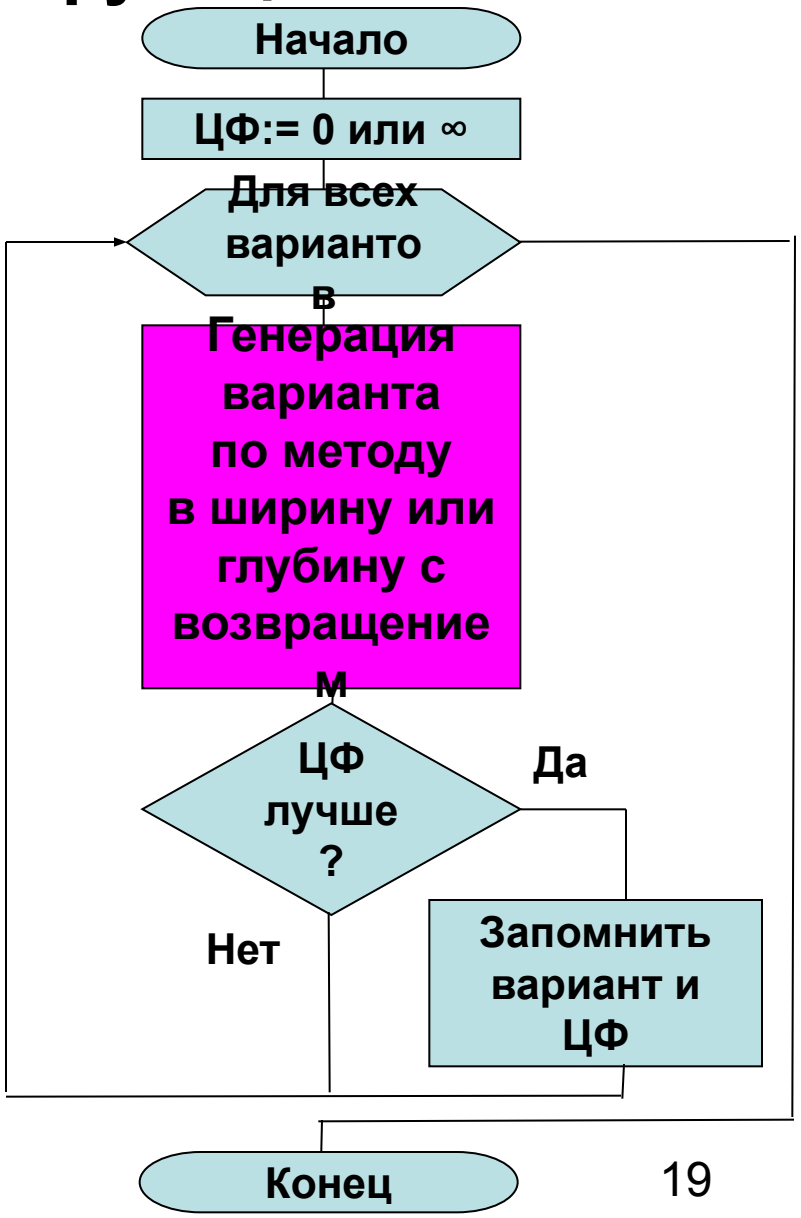
Просмотр графа как в глубину с возвращением, так и ширину требует ($n + m$) операций, где $n = |X|$, $m = |U|$. Если локальная степень вершин неориентированного графа (или сумма полустепеней исхода и захода в ориентированном) ограничена константой, то асимптотическая оценка построения d - и b-деревьев будет равна O (n).

4.2 Поиск решения с использованием оценочной функции

В практике проектирования обычно необходимо найти одно решение, наилучшее в смысле некоторого функционала – *целевой функции*.

Поиск такого решения при декомпозиции множества вариантов в ширину или в глубину с возвращением подразумевает вычисление значений целевой функции для каждого варианта, сравнение их и выбор оптимального, т. е. приводит *к полному перебору*.

Полный перебор требует слишком много времени либо просто нереализуем по этой причине при достаточно *большой размерности задачи*.



Оценка варианта в процессе генерации

Существование *некоторой оценки*, позволяющей с некоторой достоверностью судить о том, содержит ли *подмножество вариантов* оптимальное решение, обеспечивает *возможность избежать полного перебора*.

В общем случае оценка – это значение функции $F(M_i)$ на вершинах дерева решений, как правило исключая корень, равная в его конечных вершинах значению целевой функции для соответствующего варианта решения, а в остальных вершинах– нижней или верхней границе функции F для вариантов, входящих в это множество.

Особенности оценочных функций

Оценочные функции могут быть определены:

- на *любых* подмножествах вариантов;
- на *некоторых* подмножествах вариантов.

В первом случае выбор принципа разбиения множества *M* на подмножества не зависит от оценочной функции.

Во втором случае используемый принцип разбиения должен обеспечивать получение только таких подмножеств, на которых определена оценочная функция. Отсюда следует, что вид дерева решений зависит от оценочной функции.

Вид оценочной функции и та степень достоверности, с которой можно судить по ней о наличии в подмножестве оптимального варианта, порождает различные методы поиска по дереву решений.

Особенности оценочных функций

Различают два вида оценочной функции:

- *отсекающую оценку*, которая достоверно указывает, что исследуемое *подмножество не содержит оптимального решения* – подмножество можно исключить из процесса разбиения (*отсечь* ветви и вершины дерева решений, ему сопоставленные);

  отсекающей также является *оценка выбора подмножества*, гарантированно содержащего оптимальный вариант решения. В этом случае отсекаются все подмножества вариантов, которые порождаются остальными вершинами дерева решений.
- *оценку перспективности*, которая может служить для обоснования очередности разбиения подмножеств, т. е. выбора на каждом шаге построения дерева решений наиболее «перспективной» вершины, с большей вероятностью, чем другие, содержащей оптимальное решение.

Выбор оценочных функций

Выбор оценочной функции – один из самых сложных и ответственных этапов подготовки задачи к решению. *От него зависит возможность точного решения задачи..* В свою очередь возможности оценочной функции в значительной степени определяются *спецификой задачи и математическими свойствами графов* – объекта и результата проектирования.

- Оценочная функция в виде *верхней или нижней границы целевой функции* $F_в(M_i)$ и $F_н(M_i)$ (возможно ее текущего значения, например суммы длин ребер построенного участка маршрута) обычно является основанием для выбора более перспективного подмножества, но может служить и отсекающей оценкой. При этом $F_в(M_i)$ может только убывать, а $F_н(M_i)$ – только возрастать по мере разбиения M_i.

Выбор оценочных функций

В качестве отсекающей оценки для большинства комбинаторно-оптимизационных задач может выступать *опорное решение*, т. е. значение целевой функции какого-либо варианта решения. Несомненно, что, если для задачи на минимум для какого-то *подмножества* нижняя граница или текущее значение целевой функции больше или равно значению целевой функции некоторого решения, это подмножество не содержит оптимального варианта.

Отметим, что все точные комбинаторные методы решения задач дискретной оптимизации используют идею отсечения.

Способы отсечения ветвей дерева решений

Можно выделить *четыре основных способа отсечения ветвей*.

1. Для подножеств, соответствующих вершинам дерева (графа) решений, существует оценка выбора, гарантирующая, что оптимальное решение порождается определенным подмножеством. Например, выбор на каждом шаге ребра минимального веса обеспечивает получение точного решения задачи построения минимального остовного дерева.
2. Указанная выше оценка справедлива только для части подмножеств, для остальных она является оценкой перспективности. Например, если два фрагмента простой цепи приходят в одну и ту же вершину, то при решении задачи на минимум целевой функции фрагмент с большим весом не содержит оптимального решения.

Способы отсечения ветвей дерева решений

3. Сравнение оценки (нижней – Fн(M_i) или верхней – Fв(M_i) границы) со значением целевой функции для уже найденного (опорного) решения Fоп. Действительно, если при решении задачи на минимум целевой функции Fн(M_i)>Fоп, то подмножество M_i не содержит оптимального варианта, и соответствующая вершина дерева решений отсекается. Опорное решение может быть получено заранее приближенным алгоритмом, реализующим метод жадного выбора, либо в ходе решения задачи тем или иным методом.
4. По результатам сравнения двух оценок. Такое отсечение выполняется, если, например в задаче на минимум целевой функции, для подмножества вариантов соответствующей вершины можно найти оценку снизу Fн(M_i) и сверху Fв(M_i). Тогда, если для некоторого подмножества M_k окажется, что Fн(M_k) ≥ Fв(M_i), то ветвление в вершине, соответствующей M_k, прекращается.

Невычисляемая отсекающая оценка

Вырожденным случаем оценочной функции является невычисляемая отсекающая оценка, например, некоторое *заранее сформулированное условие*.

Примером может служить задача о кодовом замке. Пусть кодовый замок имеет четыре разряда, и каждый разряд может принимать значения «0» или «1». Известно, что код не может содержать подряд трех нулей или единиц – *условие отсечения*.

При генерации вариантов будем использовать декомпозицию по методу *в глубину с возвращением* или *с отходом назад*.

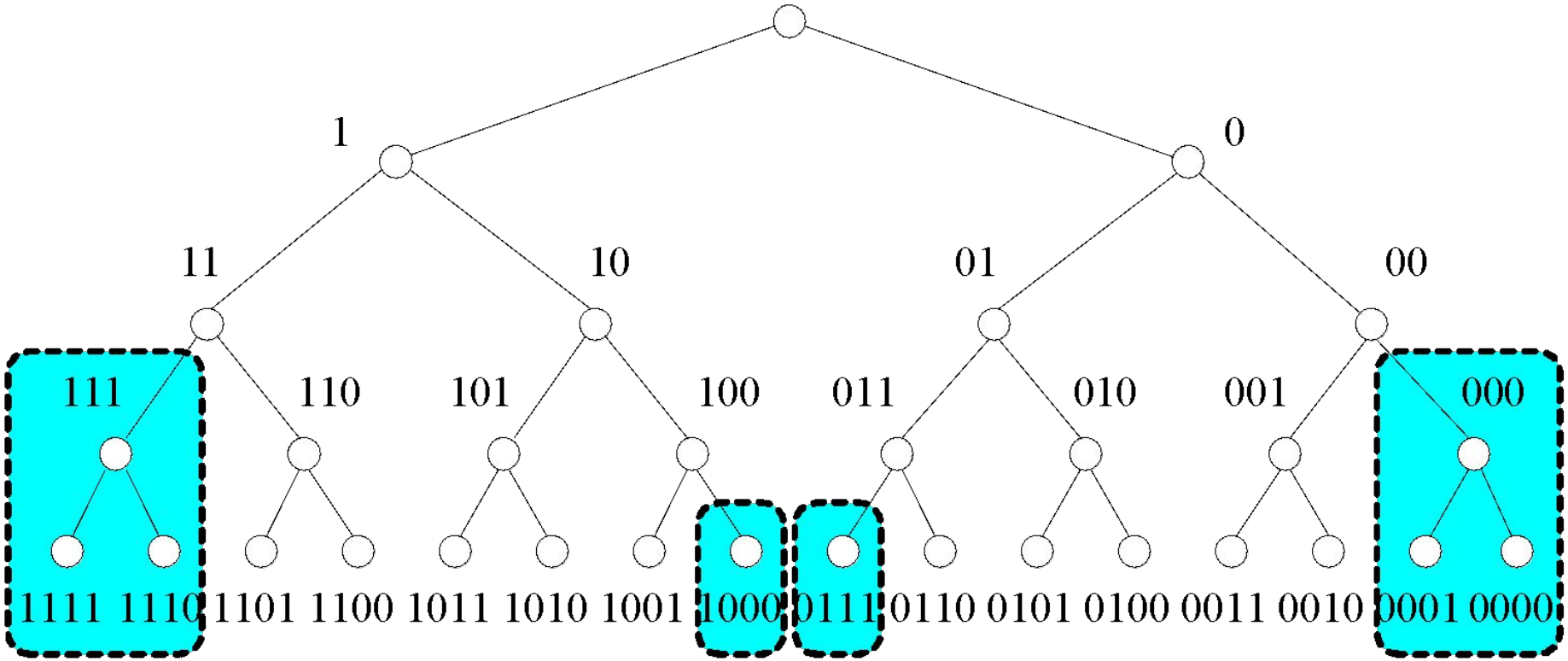
Примем направление обхода дерева решений слева направо. Принцип разбиения множества решений – присваивание соответствующему разряду значения «0» или «1» (количество сгенерированных разрядов равно номеру уровня дерева решений). Очевидно, что дерево будет бинарным. Тогда алгоритм можно сформулировать так: движемся вниз по дереву, придерживаясь левой ветви, пока не получим полной комбинации либо не выясним нецелесообразность этого.

Задача о кодовом замке

Если комбинация

- не подходит,
- либо ее или некоторое множество комбинаций нет смысла набирать (отсечение по условию),

то возвращаемся в ближайшую вершину и пробуем спускаться по другой ветви (левая ветвь – разряд ставим в положение «1», правая – в «0»).



4.3 Жадный метод

Этот метод осуществляет поиск решения задачи в процессе декомпозиции по методу в ширину. Для реализации метода задача разбивается на подзадачи, имеющие альтернативные варианты решения. Решение задачи в целом получают последовательным решением подзадач. Он заключается в том, что на каждом уровне дерева решений в соответствии с некоторой оценкой, определяемой спецификой задачи, выбирается одно подмно-жество, не формируя (отсекая) остальные, до тех пор, пока не будет получено решение. Отметим, что для этого метода часто используют термин «*жадный алгоритм*».Таким образом, здесь формируется только одна ветвь дерева решений.

Оценка метода: алгоритмы, основанные на этом методе, являются наиболее эффективными в смысле затрат времени и памяти ЭВМ;

Жадный метод

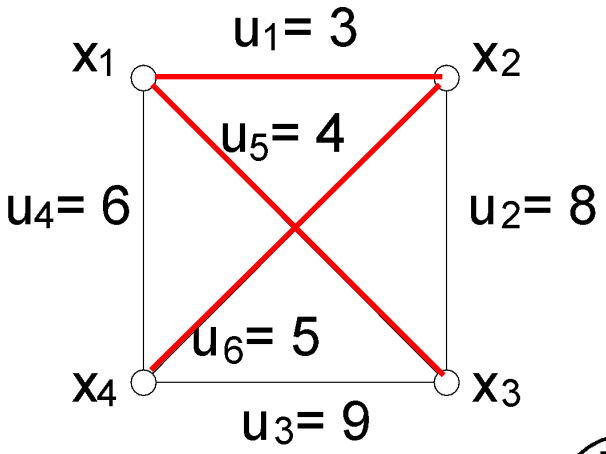
- если оценка выбора является отсекающей, т.е. с вероятностью, равной единице, позволяет судить о том, что оптимальный вариант находится в данном подмножестве, то метод обеспечивает точное решение. В противном случае гарантирует лишь приближенное.

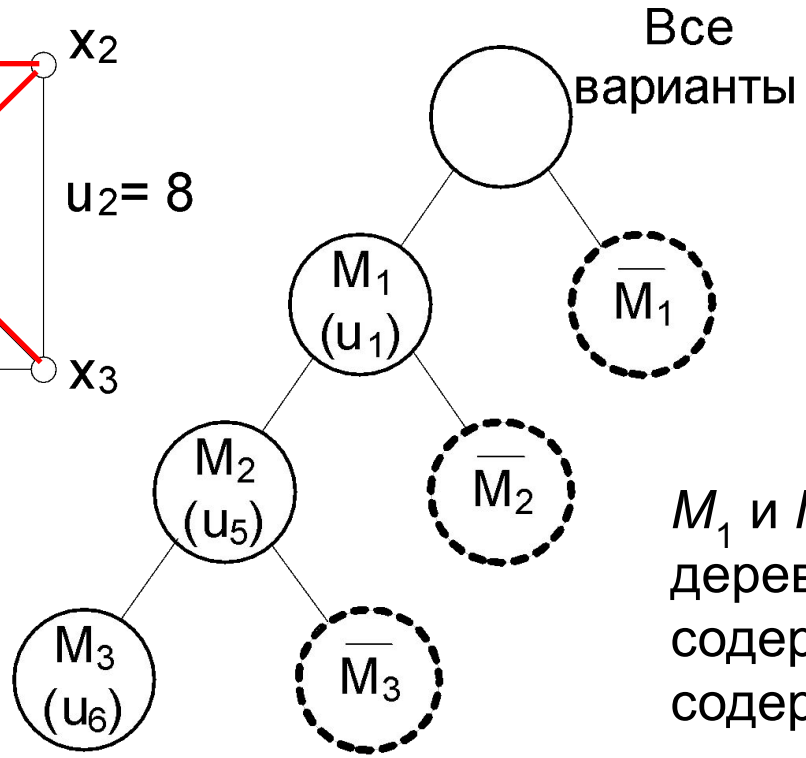
Другими словами условием возможности получения точного решения этим методом является свойство оптимальности задачи – локальные критерии оптимальности подзадач оптимизируют целевую функцию задачи.

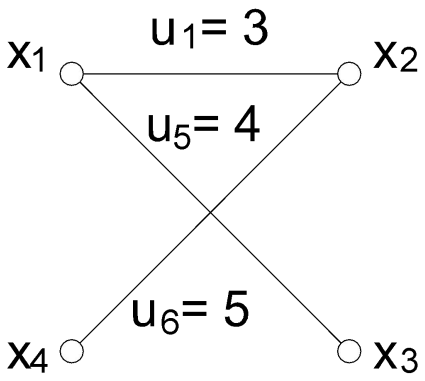
Последнее положение имеет принципиальное значение, учитывая, что в большинстве источников утверждается, что последовательные алгоритмы, реализующие метод поиска в глубину, могут гарантировать лишь приближенное решение.

Пример. Алгоритм Прима построения остовного дерева минимального веса

Алгоритм реализует жадный метод поиска решения: на первом шаге алгоритма ищется ребро минимального веса, далее, на втором шаге, к уже построенному поддереву присоединяется ребро минимального веса, не образующее с ним цикла, до тех пор, пока все вершины не будут включены в дерево.







M_1 и $\overline{M_1}$ - все варианты деревьев графа $G(X,U)$, содержащих и не содержащих ребро u_1

Пример. Алгоритм Прима построения остовного дерева минимального веса

Теоретически доказано, что *выбор на каждом шаге ребра минимальной длины обеспечивает получение точного решения*.

Таким образом, минимальная длина очередного ребра является *достоверной оценкой*, а включение в строящееся дерево такого ребра отсекает все вершины дерева решений, которые соответствуют вариантам остовного дерева графа G, не содержащим этого ребра. Алгоритм относится к классу *полиномиальных* и имеет вычислительную сложность не хуже $O(n^2)$.

Разбиение задачи на подзадачи

Разбиение задачи на подзадачи определяется видом решения (остовное дерево, кусок графа и др.) и принципом его формирования. Например, в алгоритме Прима результатом решения подзадач может быть только одна компонента связности за счет выбора ребра только из ребер, смежных ребрам уже построенного поддерева, а в алгоритме Краскала возможно появление нескольких компонент связности за счет выбора из всех оставшихся ребер.

4.4 Поиск в ширину и в глубину с возвращением

Последовательное получение решений при декомпозиции как в ширину, так и в глубину с возвращением, позволяет реализовать поиск оптимального решения в процессе декомпозиции и, возможно, избежать полного перебора для *NP*-полных задач. Необходимым условием для этого является наличие некоторой оценки, которая является отсекающй для некоторого подмножества (подмножеств) вариантов, а для остальных она является оценкой перспективности.

Для многих комбинаторно-оптимизационных задач характерным является то, что в качестве отсекающей оценки может выступать только опорное решение, т.е. значение целевой функции уже полученного варианта решения. Например, если для задачи на минимум для какого-то подмножества нижняя граница или текущее значение целевой функции больше или равно значению целевой функции некоторого решения *F*оп, это подмножество не содержит оптимального варианта. Следовательно, значение целевой функции уже полученного решения может быть использовано в качестве отсекающей оценки для следующих вершин дерева декомпозиции.

Поиск в ширину и в глубину с возвращением

В ходе решения, если возможно, уточняется значение отсекающей оценки. Оптимальное решение будет найдено, когда в задаче на минимум целевой функции ее значение для некоторой конечной вершины меньше нижней границы $F_н$ для всех висячих вершин и меньше значения целевой функции для всех остальных конечных вершин (в задаче на поиск максимума целевой функции соответственно «больше»).

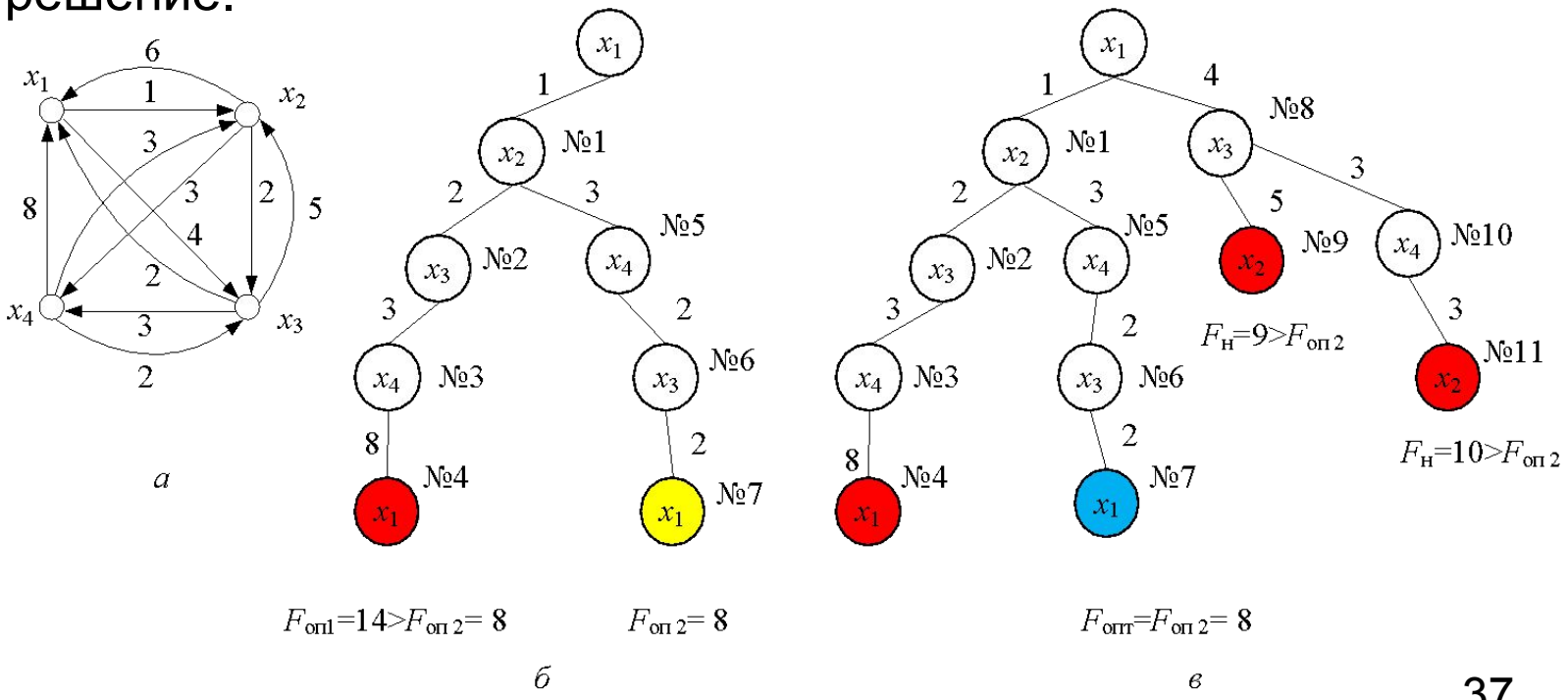
Нижняя $F_н$ или верхняя $F_в$ границы наиболее просто вычисляются в тех задачах, в которых необходимо найти оптимальное решение по минимуму или максимуму суммы весов ребер. Примером такой задачи является симметричная или несимметричная задача коммивояжера (поиск гамильтонова цикла). Как известно, для ее решения не существует алгоритма с полиномиальной оценкой вычислительной сложности. В связи с этим отсечение вариантов на основе использования опорного решения при поиске в ширину или глубину с возвращением может оказаться весьма полезным.

4.4.1 Поиск в глубину с возвращением

Поиск *в глубину с возвращением* рассмотрим на примере задачи нахождения в ориентированном графе $G^{\rightarrow}(X,U)$ гамильтонова цикла минимального веса [Асанов, Сигал]. Формальная постановка этой задачи была рссмотрена ранее. Принцип разибения множества вариантов гамильтонова цикла на подмножества – включение в формируемый цикл некоторого ребра, инцидентного достигнутой вершине и не образующего частичного цикла. В качестве нижней границы Fн будем использовать сумму весов ребер построенного фрагмента. Используя полученные решения, будем отсекать вершины (ветви) дерева решений, если Fн ≥ Fоп.

Поиск в глубину с возвращением

На рисунке показан ориентированный граф (*а*), две ветви дерева решений (*б*) и все дерево поиска гамильтонова цикла минимального веса, начиная с вершны x_1, (*в*). Около ребер указаны их веса. Вершина желтого цвета – текущее опорное решение, красного – отсеченная, голубого – оптимальное решение.



а

б

в

Поиск в глубину с возвращением

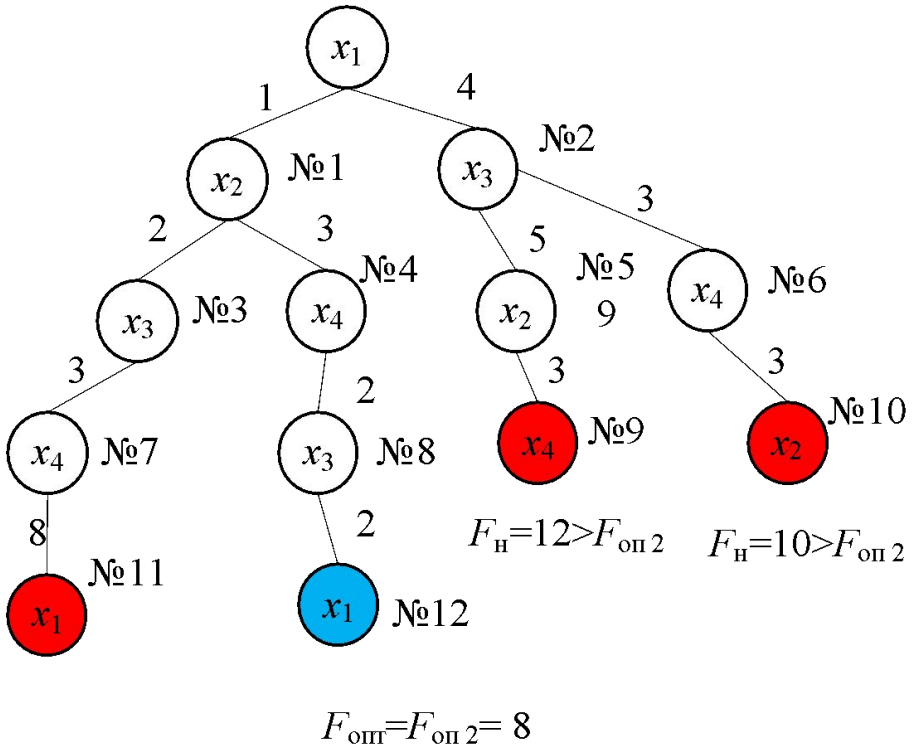
Ребро, соединяющее две вершины дерева решений, будет соответствовать ребру графа $G^{\rightarrow}$, инцидентному тем же вершинам. Около ребер справа проставлены их номера в порядке появления при построении дерева решений.

При построении дерева решений ребра графа $G^{\rightarrow}$ выбирались в соответствии с возрастанием индексов вершин, которые им инцидентны. При упорядочивании ребер по невозрастанию их весов эффективность отсечения может быть выше за счет более раннего нахождения опорного решения близкого к оптимальному решению.

4.4.2 Поиск в ширину

Задачу коммивояжера можно решать и методом *поиска в ширину*. На рисунке показано дерево поиска гамильтонова цикла минимального веса, начиная с вершны x_1, методом поиска в ширину. Вид дерева поиска решения зависит от метода и выбора начальной вершины.

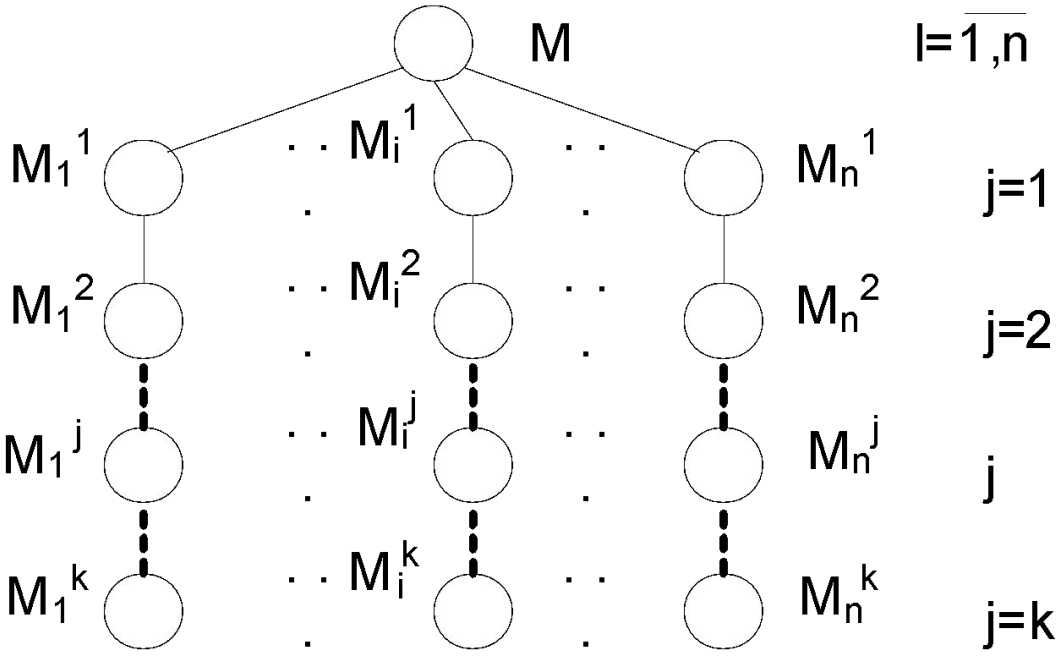
Очевидно, что степень сокращения количества перебираемых вариантов при применении данных методов непредсказуема – решение может быть найдено и за полиномиальное время и в результате полного перебора. Важным является то, что оба эти метода могут применяться для точного решения NP полных задач.



4.4.3 Метод параллельного поиска

Метод параллельного или одновременного поиска относится к классу эвристических и представляет собой комбинацию методов поиска в ширину и в глубину:

- множество решений разбивается на подмножества 1-го уровня, каждая вершина которого становится началом ветви, формируемой по *методу поиска в глубину*;
- переход к формированию подмножеств следующего уровня дерева решений выполняется после получения подмножеств *всех ветвей текущего уровня*.



Здесь n – количество искомых решений, следовательно, на j -ом уровне дерева n подмножеств M_i^j, $i = 1,n$ – множество индексов этих подмножеств (верхний индекс соответствует номеру уровня дерева решений).

Метод параллельного поиска

В зависимости от специфики задачи подмножества разных ветвей могут быть как пересекающимися, так и непересекающимися.

Рассмотрим задачу компоновки схемы в n конструктивных модулей. Множество элементов схемы $Э$ поставлено во взаимнооднозначное соответствие множеству вершин X гиперграфа $Э \leftrightarrow X$.

Указанная задача может быть решена алгоритмом, реализующим метод параллельного поиска. Результатом его работы будет разбиение множества X вершин гиперграфа на n подмножеств X_i, $i=\overline{1,n}$. Так как один и тот же элемент схемы не может входить в разные конструктивные модули и состав X_i определяется по методу поиска в глубину последовательным включением вершин $x \in X$ в X_j^i (X_j^i соответствует M_j^i), то $X_j^i \cap X_p^i = \varnothing$ для всех $j, p \in I = \{\overline{1,n}\}$. Таким образом, в данной задаче подмножества, принадлежащие разным ветвям дерева решений, должны быть непересекающимися.

Если подмножества вариантов 1-го уровня содержат все варианты решения, т.е. удовлетворяют условию $\cup M^1_i = M$ и оценка выбора подмножества в каждой ветви является отсекающей, то метод обеспечивает получение точного решения.

4.5 Метод ветвей и границ

Универсальный, хотя и достаточно сложный, метод точного решения широкого круга комбинаторно-оптимизационных задач посредством ***направленного*** перебора вариантов при условии, что их число – ***конечно***.

В худшем случае метод может вылиться в полный перебор.

В общем случае выполняется частичный перебор. Сокращение количества просмотренных вариантов достигается за счет:

- *организации ветвления* – разбиения множества вариантов на подмножества в наиболее перспективной вершине;
- *отсечения* подмножеств вариантов, не содержащих оптимального.

Ветвление – *по методу в ширину* или *в глубину с возвращением* с заданным порядком построения ветвей.

Принцип ветвления и вычисление условий отсечения существенно зависят от вида задачи, а степень сокращения перебора – от ее конкретных данных.

Метод ветвей и границ

В качестве *оценки перспективности* ветви используется *верхняя* $F_{в}(M_i)$ *или нижняя* $F_{н}(M_i)$ *границы целевой функции*, вычисляемые для каждого подмножества вариантов M_i. В ряде случаев они могут использоваться и для отсечения ветвей.

Рассматриваемый метод по сравнению с другими методами точного решения *NP* полных задач (поиск в ширину и глубину с возвращением) в большей степени направлен на сокращение полного перебора.

Способы отсечения

Можно выделить три основных способа отсечения ветвей:

1. *По результатам сравнения оценок сверху или снизу со значением целевой функции для уже найденного опорного решения $F_{оп}$,* например, если в задаче на минимум целевой функции для некоторого подмножества M_j оказалось, что $F_н(M_j) \geq F_{оп}$, то ветвление в вершине соответствующей подмножеству M_j следует прекратить.

   Опорное решение может быть получено приближенным алгоритмом заранее или в ходе решения задачи методом ветвей и границ. Причем при разбиении пространства решений при стратегии в глубину с возвращением опорное решение как правило получается на более ранних этапах построения дерева решений, чем при стратегии в ширину.

Способы отсечения

2. *По результатам сравнения оценок сверху и снизу*, например, если в задаче на минимум целевой функции для некоторого подмножества M_j оказалось, что $F_н(M_j) \geq F_в(M_i)$, то ветвление в вершине соответствующей подмножеству M_j следует прекратить.

3. Прекращение ветвления *в «особых точках»*, для которых откуда-либо известно, что полученное множество решений не содержит оптимального варианта.

Чем точнее будет получена оценка, т. е. чем ближе она будет к значению целевой функции для оптимального варианта подмножества, тем больше вершин будет отсечено и, следовательно, меньше вершин дерева решений будет построено и исследовано.

Выбор принципа разбиения и оценочной функции

Точность оценки зависит от принципа разбиения множества вариантов на подмножества и вида оценочной функции или способа ее вычисления.

- Чем сильнее отличается оценочная функция в различных вершинах дерева решений одного уровня или огибающей цепи, тем меньшее число вариантов будет рассмотрено.

Огибающая цепь – совокупность вершин, которые можно ветвить на данном шаге («висячих», т. е. не являющихся конечными или отсеченными).

Таким образом, лучшими являются тот принцип разбиения и такая оценочная функция, при которых разность между оценками подмножеств наибольшая, а сами оценки вычисляются с наибольшей точностью в смысле их близости к значению целевой функции для оптимального варианта подмножества. Число отсечений зависит и от способа ветвления, хотя никаких точных оценок и рекомендаций здесь не существует.

Способы ветвления

1. Разбиение множества вариантов на подмножества по методу в ширину и выбор вершины ветвления по минимуму (максимуму) оценочной функции.

На первом уровне строятся все вершины потомки исходной их или часть. Затем на каждом шаге выбирается вершина ветвления по минимуму нижней границы (максимуму верхней для задачи на максимум целевой функции) и разбивается на соответствующее ей подмножество вариантов. В ходе ветвления используется, если это возможно, тот или иной способ отсечения ветвей и вершин. Процесс выбора вершин и ветвления повторяется для всех висячих вершин.

2. Разбиение множества вариантов по методу в глубину с возвращением – последовательное построение ветвей.

Строим полностью одну ветвь дерева решений – находим опорный вариант. Полученное для этого варианта значение целевой функции может использоваться как отсекающая оценка. Далее ветвление выполняется последовательно от построенной ветви, начиная с вершины M_i, предшествовавшей конечной. При этом новая ветвь достраивается до конца, если не происходит отсечение. Процедура повторяется по отношению к вершинам новой ветви.

При последовательном способе ветвления построение дерева решений может выполняться, начиная с левой ветви – слева направо, или с правой – справа налево.

Количество построенных вершин дерева решений зависит от очередности развития ветвей.

Способы ветвления

3. Комбинация двух рассмотренных способов.

Например, строится одна ветвь.

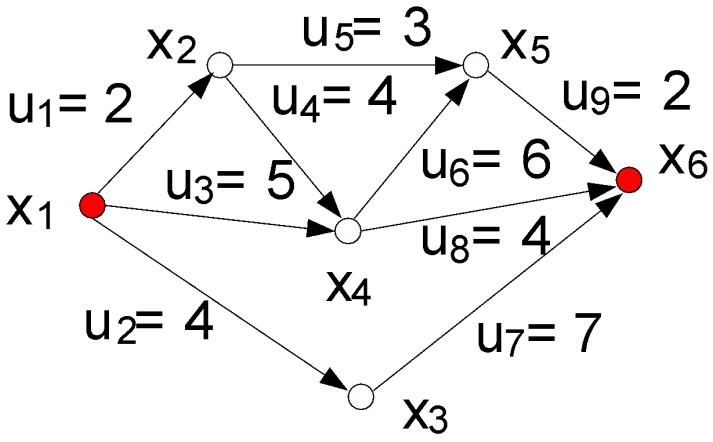
Далее развитие дерева решений происходит по методу в ширину, полученное опорное решение используется для отсечения, выбор вершины ветвления выполняется по минимуму нижней границы в задачах на минимум целевой функции (по максимуму верхней в задачах на максимум).

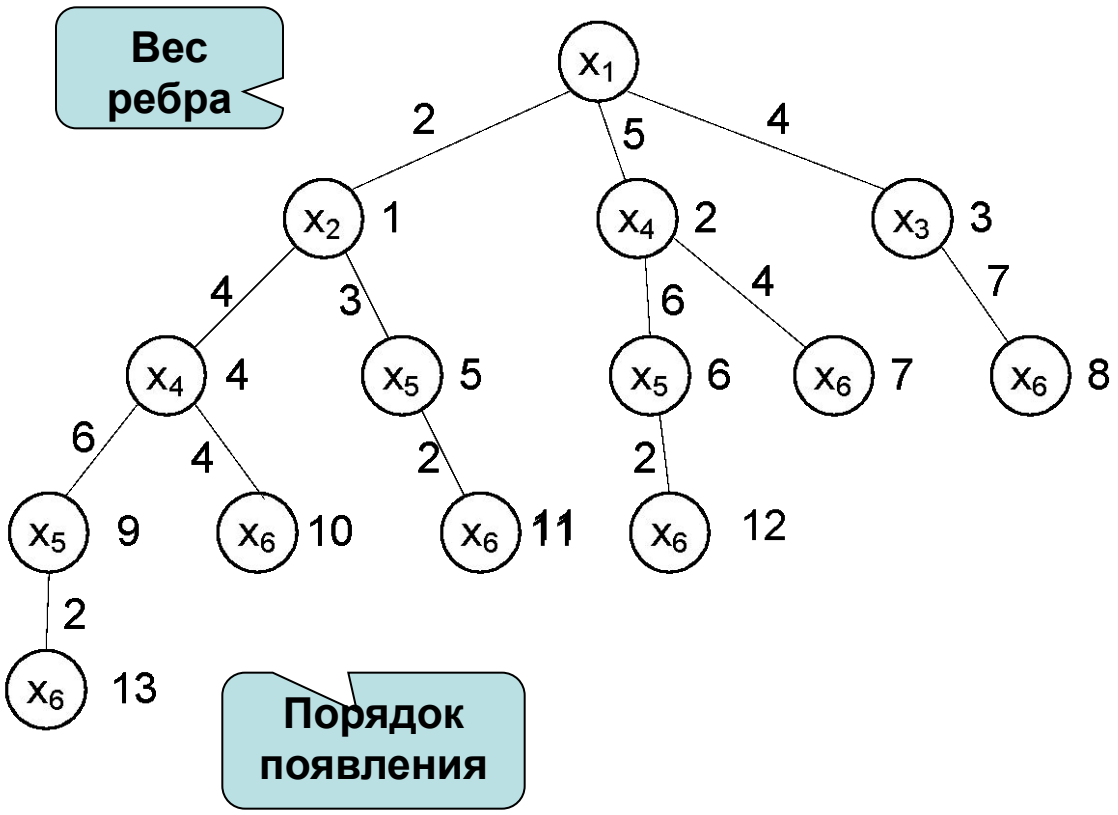
В ходе решения, если возможно, уточняется значение отсекающей оценки.

Оптимальное решение будет найдено, когда в задаче на минимум целевой функции значение для некоторой конечной вершины $F(M_i^k)$ меньше нижней границы $F_н(M_j)$ для всех висячих вершин и меньше значения целевой функции $F(M_l^k)$ для всех остальных конечных вершин (в задаче на поиск максимума целевой функции соответственно «больше»).

Пример. Задача нахождения простой цепи (маршрута) минимальной длины между заданными вершинами графа

Принцип разбиения: последовательное включение в маршруты смежных ребер.





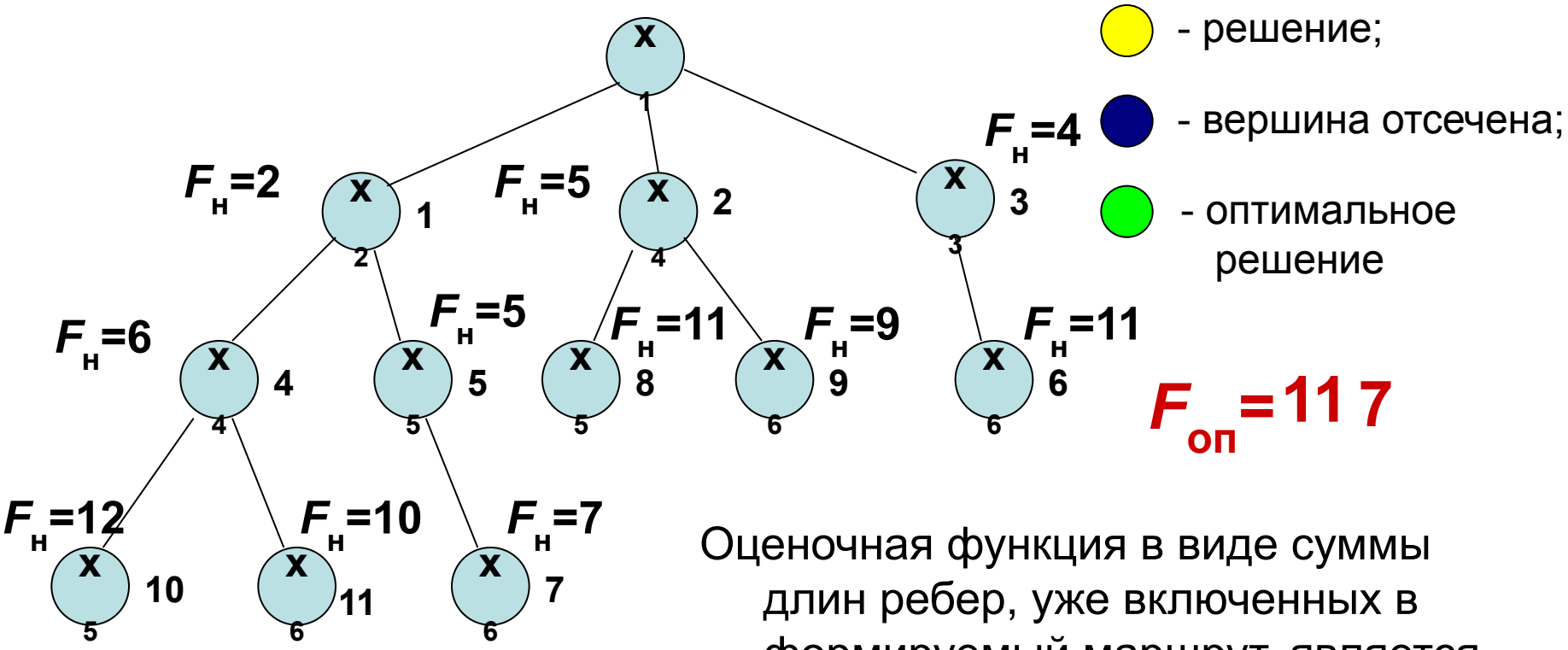
Маршруты:

$c_1 = \{x_1, x_2, x_4, x_5, x_6\}$;
$c_2 = \{x_1, x_2, x_4, x_6\}$;
$c_3 = \{x_1, x_2, x_5, x_6\}$;
$c_4 = \{x_1, x_4, x_5, x_6\}$;
$c_5 = \{x_1, x_4, x_6\}$;
$c_6 = \{x_1, x_3, x_6\}$.

Построение оценочной функции

Вначале в качестве нижней границы используем *сумму длин ребер построенного фрагмента*.

Ветвление по методу в ширину, выбирается вершина с минимальной оценкой:



Оценочная функция в виде суммы длин ребер, уже включенных в формируемый маршрут, является слабым значением нижней границы.

Построение оценочной функции (2)

Конструирование оценочных функций должно базироваться на анализе сущности задачи (в данном случае процесса построения маршрута) и свойств функций, которые могут быть использованы в качестве верхних и нижних границ.

Процесс построения маршрута – это *последовательное включение ребер* в него до тех пор пока не придем в конечную вершину.

Оценочная функция в конечной вершине должна быть равна значению целевой функции, т. е. *сумме длин ребер, составляющих соответствующий маршрут*.

Следовательно, одной из составляющих оценочной функции должна быть *сумма длин ребер, уже вошедших в формируемый маршрут* от корня до рассматриваемой вершины.

Нижняя граница для некоторой вершины дерева решений должна быть не больше, а верхняя не меньше, значения целевой функции для любой текущей и конечной вершины поддерева, начинающегося в ней. Очевидно, что вторая составляющая оценочной функции, которая должна *приблизить нижнюю или верхнюю границу к значению целевой функции оптимального варианта*, должна удовлетворять этому условию.

Построение оценки верхней границы

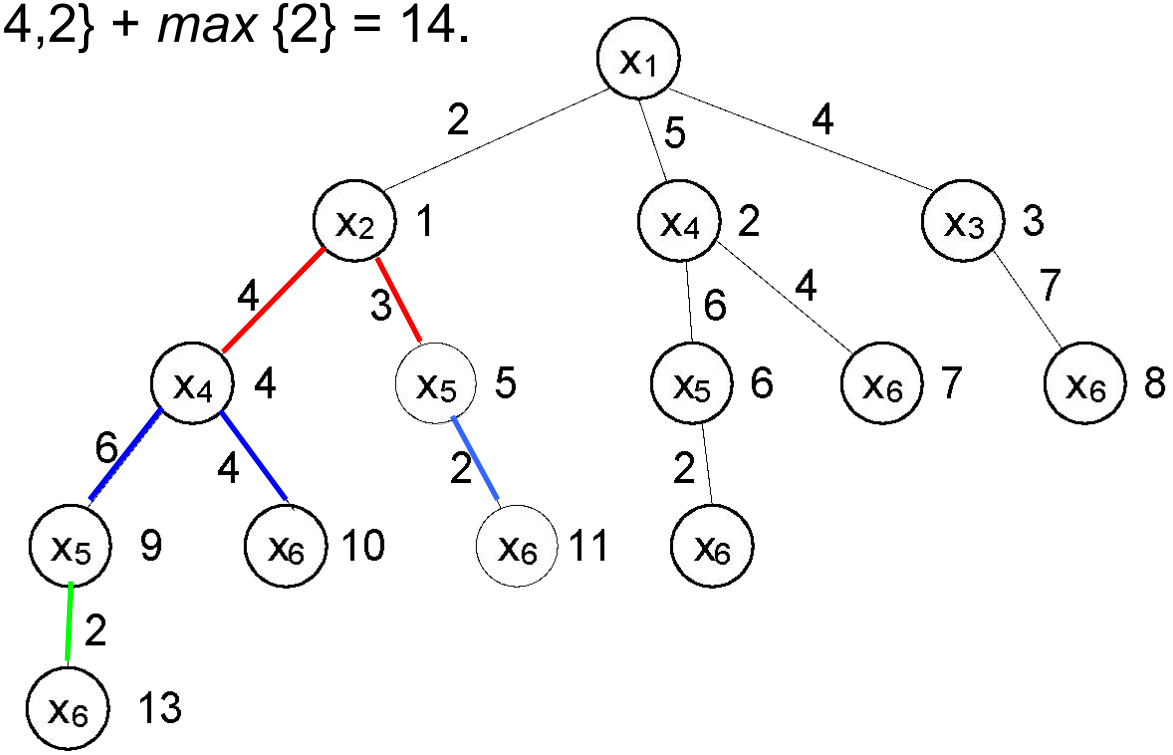
Верхняя граница для любой вершины дерева решений будет не меньше целевой функции маршрута максимальной длины, если в качестве второй составляющей оценочной функции будет выступать *сумма ребер максимальной длины среди ребер маршрутов для каждого уровня поддерева решений с корнем в этой вершине*.

Например, для вершины 1:

$F_в$=2 + *max* {4,3} + *max* {6,4,2} + *max* {2} = 14.

Сконструированная функция

- не возрастает;
- ее значение в конечных вершинах равно значению целевой функции для соответствующего варианта маршрута.



Построение оценки нижней границы

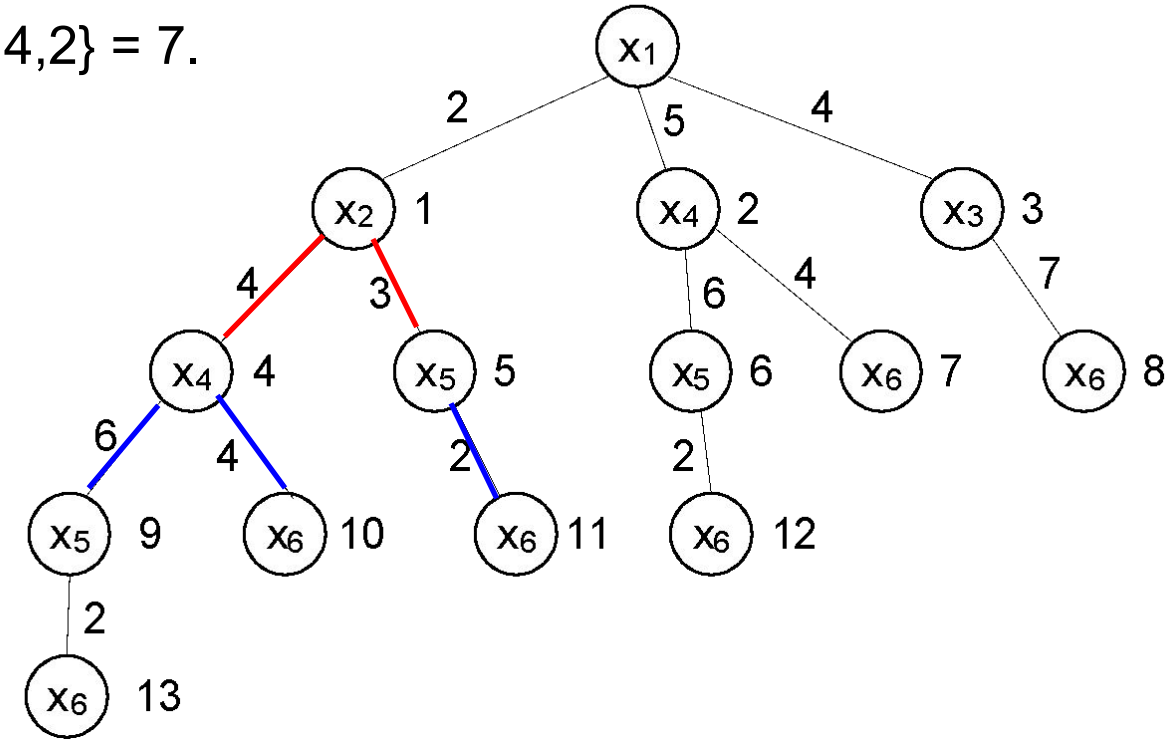
Компонентами второй составляющей для *нижней границы* должны быть *ребра минимальной длины для каждого уровня дерева решений среди ребер маршрутов, проходящих через эту вершину*. Суммируя эти величины от уровня рассматриваемой вершины до самой верхней, т. е. *ближайшей к ней* конечной вершины этих маршрутов, получим вторую составляющую.

Например, для вершины 1:

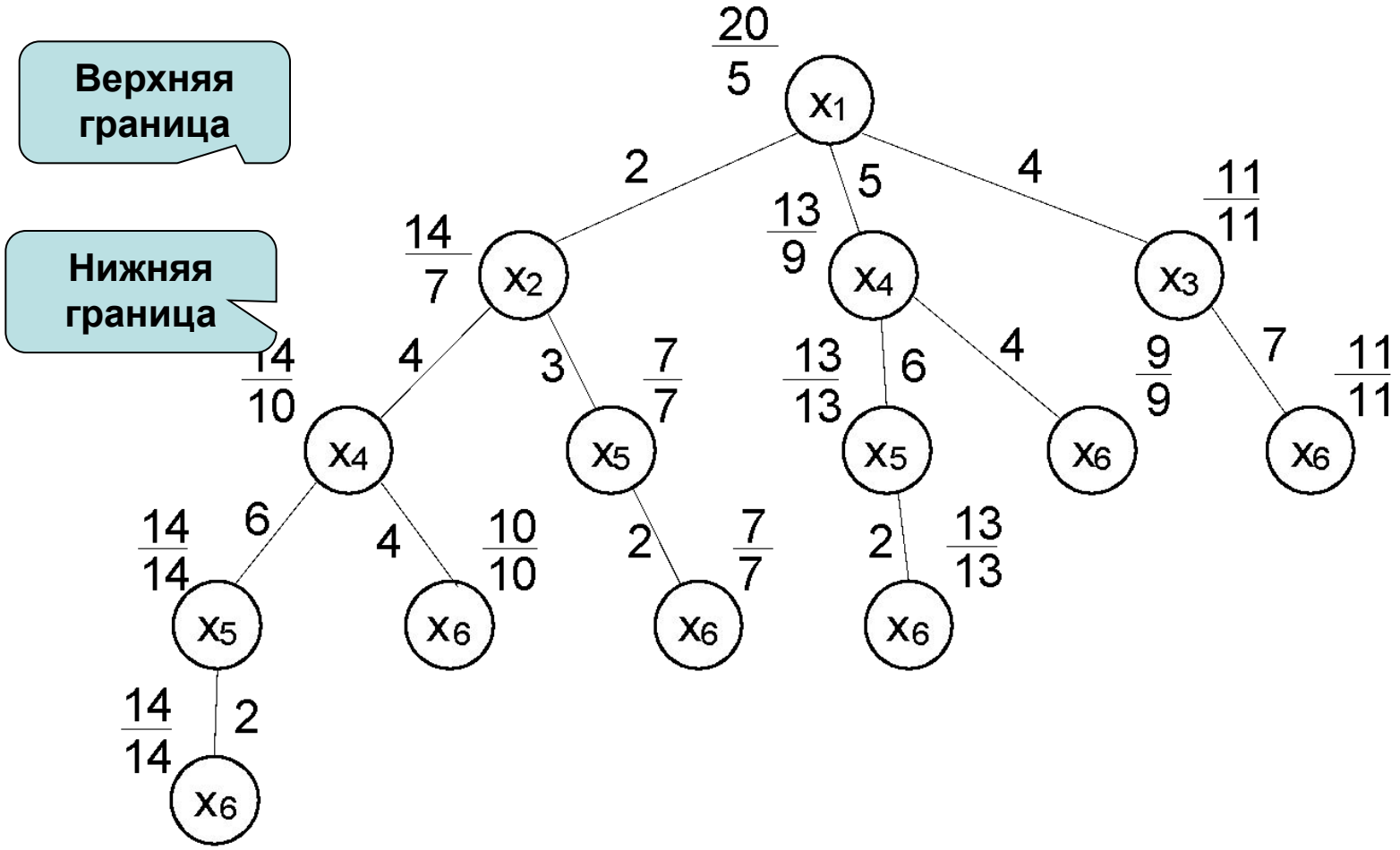
$F_н = 2 + min\ \{4,3\} + min\ \{6,4,2\} = 7.$

Сконструированная функция

- не убывает;
- ее значение в конечных вершинах равно значению целевой функции для соответствующего варианта маршрута.

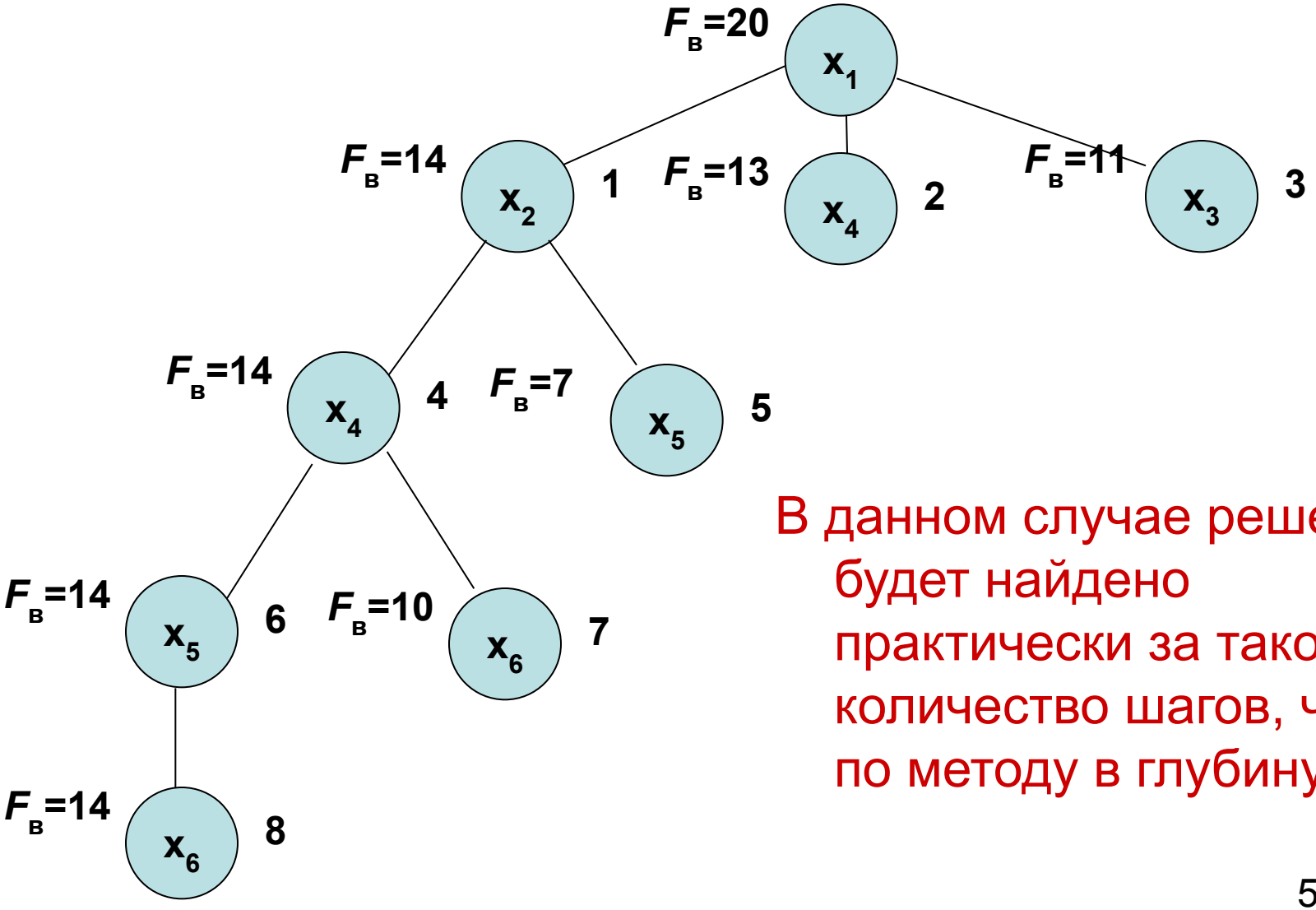


Дерево решений с оценками



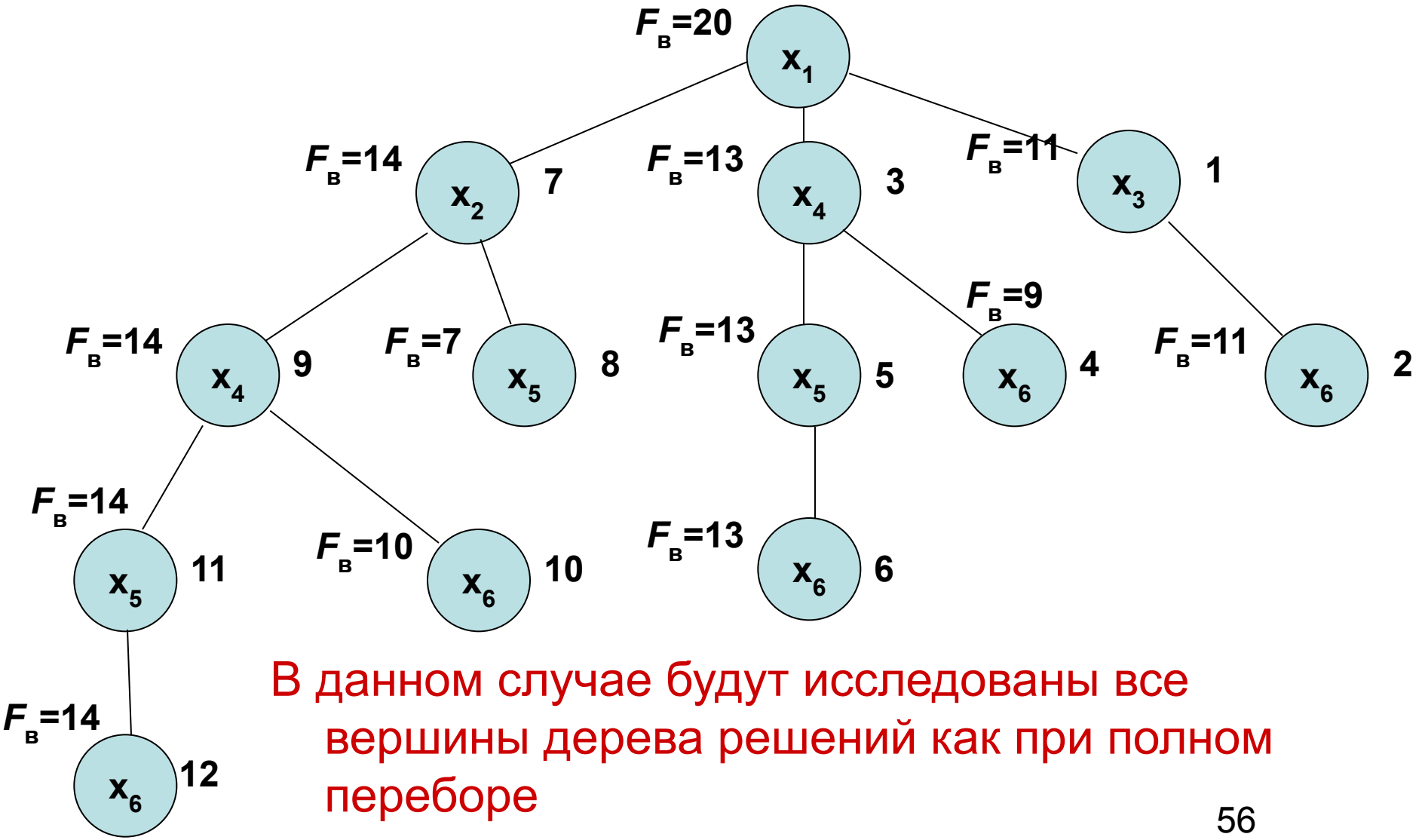
Для конкретного графа и выбранных оценочных функций количество просматриваемых вершин зависит от способа ветвления и направления построения ветвей.

Поиск маршрута максимальной длины при ветвлении в ширину и выборе вершины ветвления по максимуму верхней границы



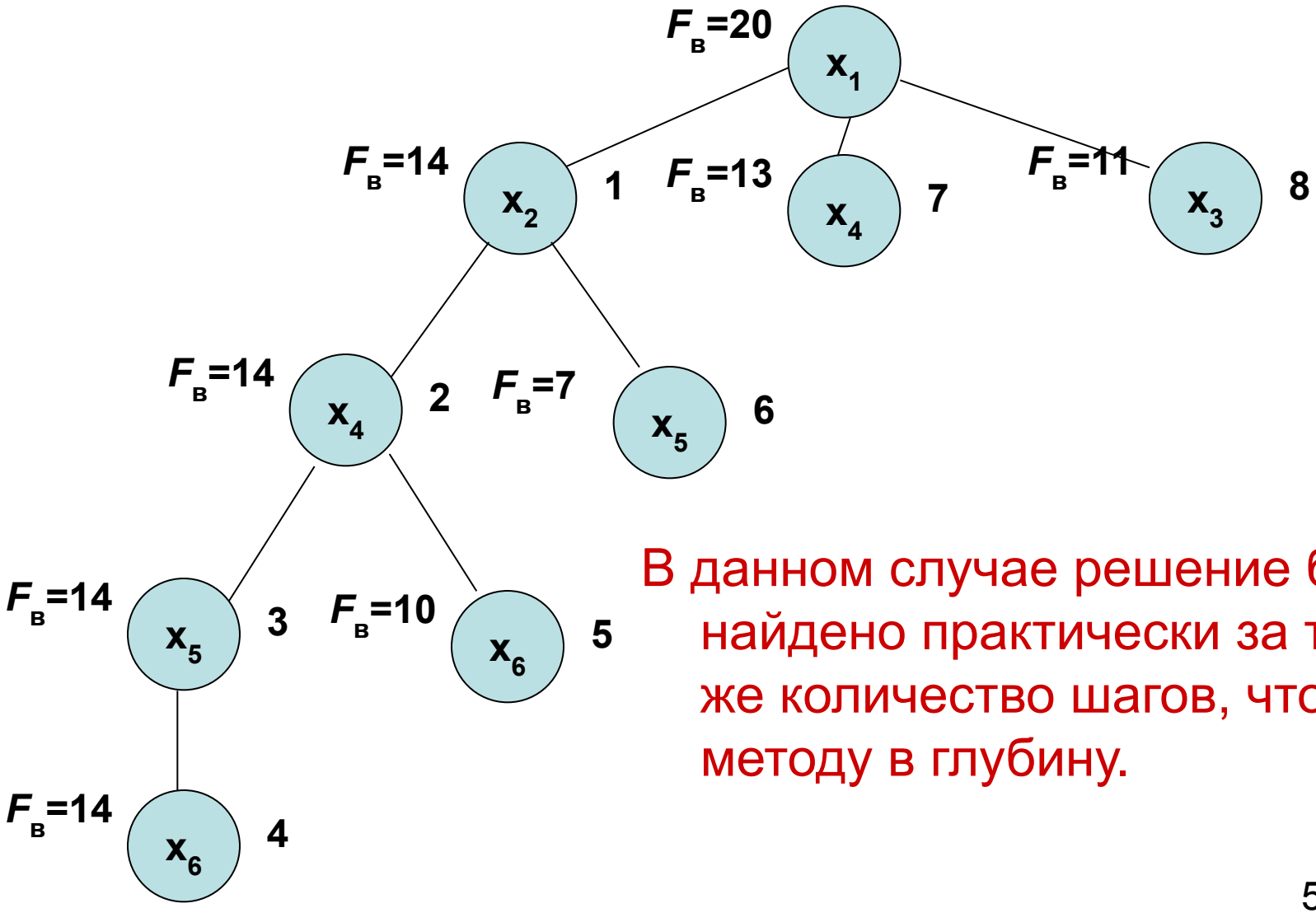
В данном случае решение будет найдено практически за такое же количество шагов, что и по методу в глубину.

Поиск маршрута максимальной длины при последовательном ветвлении и порядке построения ветвей справа налево



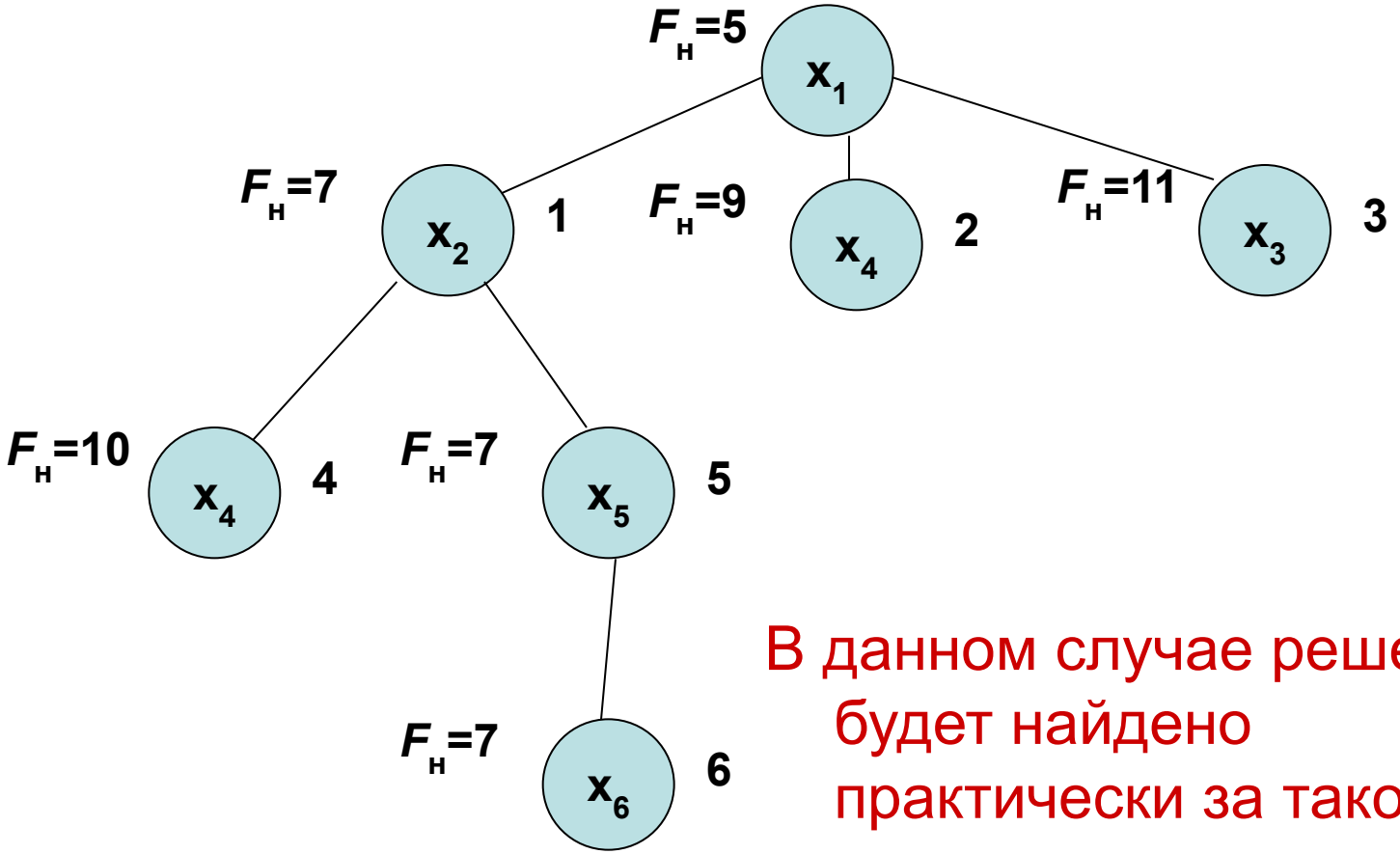
В данном случае будут исследованы все вершины дерева решений как при полном переборе

Поиск маршрута максимальной длины при последовательном ветвлении и порядке построения ветвей слева направо



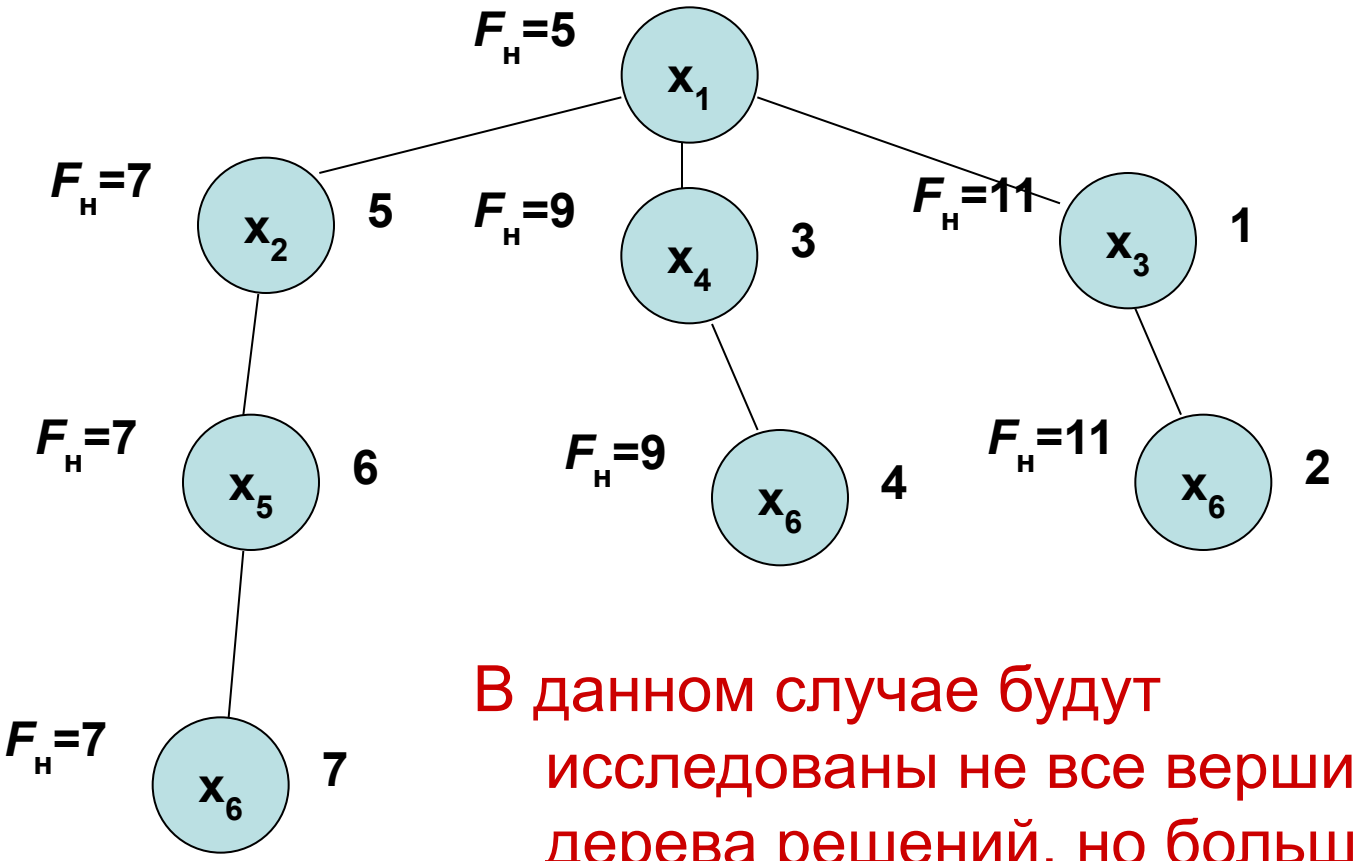
В данном случае решение будет найдено практически за такое же количество шагов, что и по методу в глубину.

Поиск маршрута минимальной длины при ветвлении в ширину и выборе вершины ветвления по минимуму нижней границы



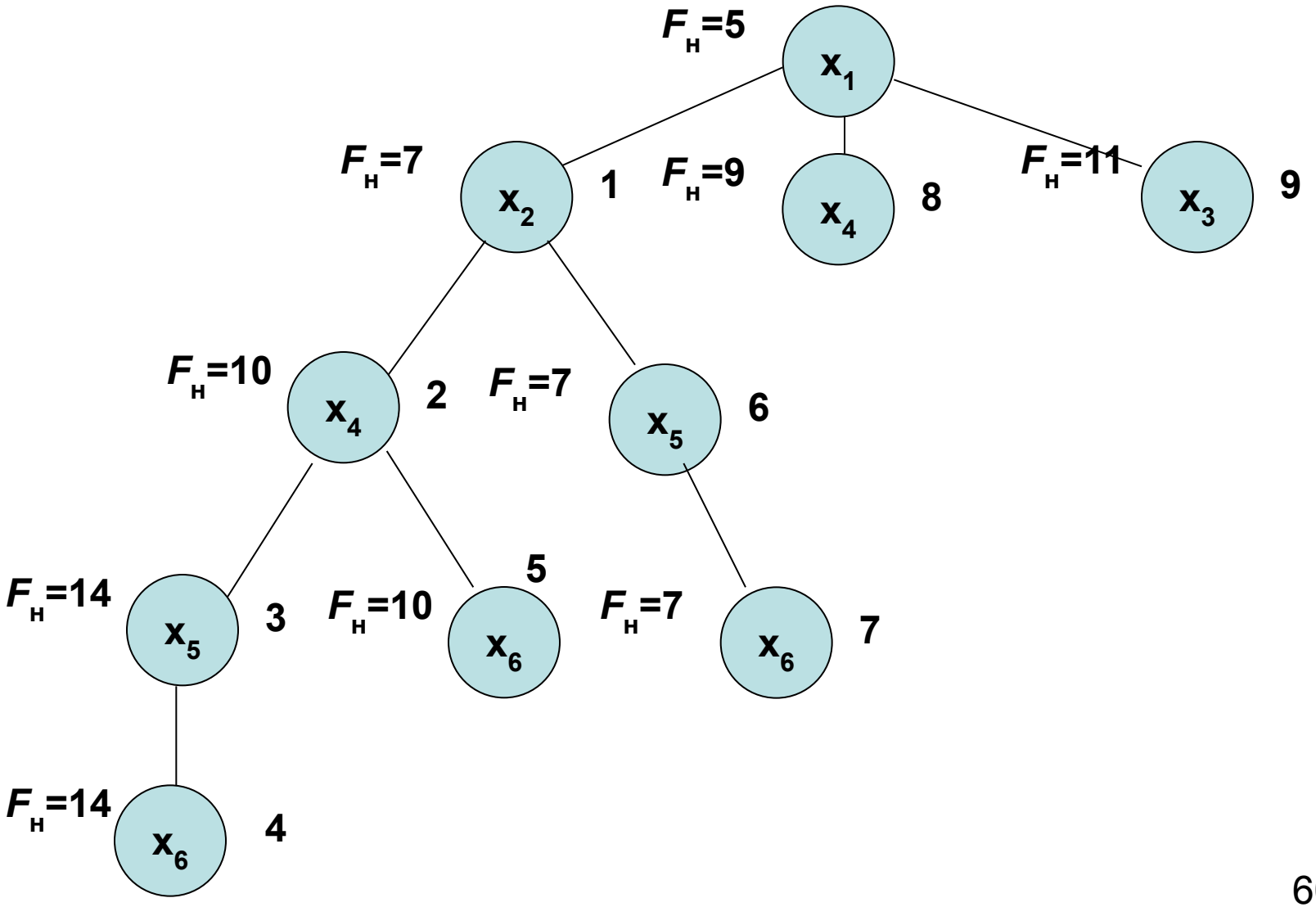
В данном случае решение будет найдено практически за такое же количество шагов, что и по методу в глубину.

Поиск маршрута минимальной длины при последовательном ветвлении справа налево



В данном случае будут исследованы не все вершины дерева решений, но большее количество, чем по методу в глубину.

Поиск маршрута минимальной длины при последовательном ветвлении слева направо



4.6 Метод Дейкстры

В настоящее время в специальной литературе с разной степенью детализации описан алгоритм Дейкстры поиска маршрута минимальной длины и как метод не трактуется. Поскольку идеи этого алгоритма могут быть использованы для решения довольно широкого круга задач дискретной оптимизации по мнению автора его общую схему следует трактовать как метод.

Метод Дейкстры использует схему метода ветвей и границ и позволяет найти за полиномиальное время точное решение таких задач, для которых характерны следующие особенности:

1. Решение задачи можно разбить на подзадачи (в задаче определения кратчайшего пути подзадачами являются части пути, приходящие в вершину);

Метод Дейкстры

2. Подзадачи не могут быть сформированы заранее, так как зависят от структуры исследуемого графа и порождаются в ходе реализации метода выбранным вариантом решения подзадачи предыдущего уровня;
3. Количество вариантов решения задачи экспоненциально зависит от размера входа, а количество подзадач, среди которых выполняется выбор перспективной, – полиномиально;
4. Задача обладает свойством оптимальности;
5. Существуют подзадачи как имеющие, так и не имеющие альтернативной реализации;
6. Для подзадач с альтернативной реализацией есть отсекающая оценка, однако она не распространяется на другие подзадачи.

Метод Дейкстры

Задача: поиск маршрута (простой цепи) минимальной длины из некоторой исходной точки в заданную конечную.

Стратегия декомпозиции множества решений – по методу в ширину *при получении вершин первого уровня графа решений, далее – ветвление в перспективной вершине.*

Принцип разбиения – включение во фрагмент пути некоторого ребра.

Вершинами графа решений являются вершины графа – модели схемы соединения компонентов объекта. Будем представлять решение в виде последовательности вершин. Таким образом, подзадачей является включение вершины в строящийся фрагмент маршрута, а решением задачи – последовательность решенных подзадач.

Метод Дейкстры

Оценочная функция в каждой вершине дерева – суммарная длина ребер уже построенного фрагмента маршрута – нижняя граница целевой функции.

Поскольку гарантия, что эта оценка является отсекающей отсутствует, она может использоваться как оценка *перспективности*, т. е. для выбора очередной вершины ветвления.

Однако в данном случае эта оценка может выступать в качестве *отсекающей* в «**особых**» вершинах дерева решений.

Этот факт основывается на свойстве графа – результата решения:

маршруты, как простые цепи, могут проходить через одну и ту же вершину графа и иметь разную длину до этой вершины.

Метод Дейкстры

Докажем,что задача обладает свойством оптимальности – любая часть кратчайшего пути сама есть кратчайший путь. Предположим, что последовательность локально оптимальных подзадач не является кратчайшим маршрутом. Следовательно, в этой последовательности существует подзадача – некоторый фрагмент маршрута меньшей длины. Предположение неверно, так как фрагменты выбирались по критерию минимума длины.

Покажем, что количество вариантов решения задачи экспоненциально зависит от размера входа, а количество подзадач (*вершин дерева решений, среди которых выполняется выбор перспективной*) – *линейно*. Очевидно, что максимальное количество вариантов маршрутов будет, если каждый пункт соединен с каждым, т.е. моделью будет полный граф.

Метод Дейкстры

Обозначим начальную вершину маршрута $x_н$, а конечную $x_к$.

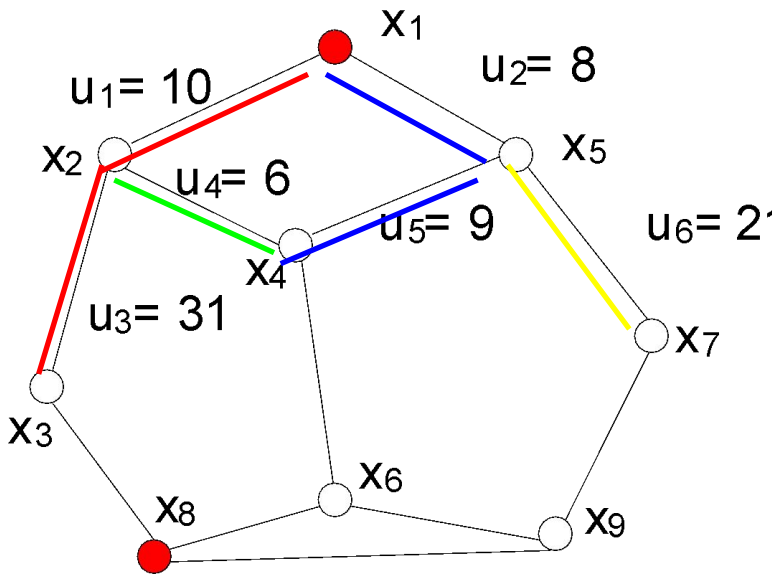
Количество только тех маршрутов, которые проходят через каждую из вершин $X \setminus \{ x_н, x_к\}$, определяется количеством перестановок элементов этого множества, т. е. $(n–2)!$ Однако выбор перспективной может происходить не более чем из $(n–1)$ подзадач. Этот факт основывается на свойстве графа – результата решения, а именно: *маршруты, как простые цепи, могут проходить через одну и ту же вершину графа и иметь разную длину до этой вершины.* Таким образом, в данной задаче локальный критерий оптимальности является *отсекающей* оценкой для альтернативных подзадач – двух и более различных фрагментов маршрутов, заканчивающихся в одной и той же вершине. Поскольку количество вершин маршрута без $x_н$ и $x_к$ не более $(n–2)$ вычислительная сложность алгоритма, реализующего данный метод, не более $O(n^2)$.

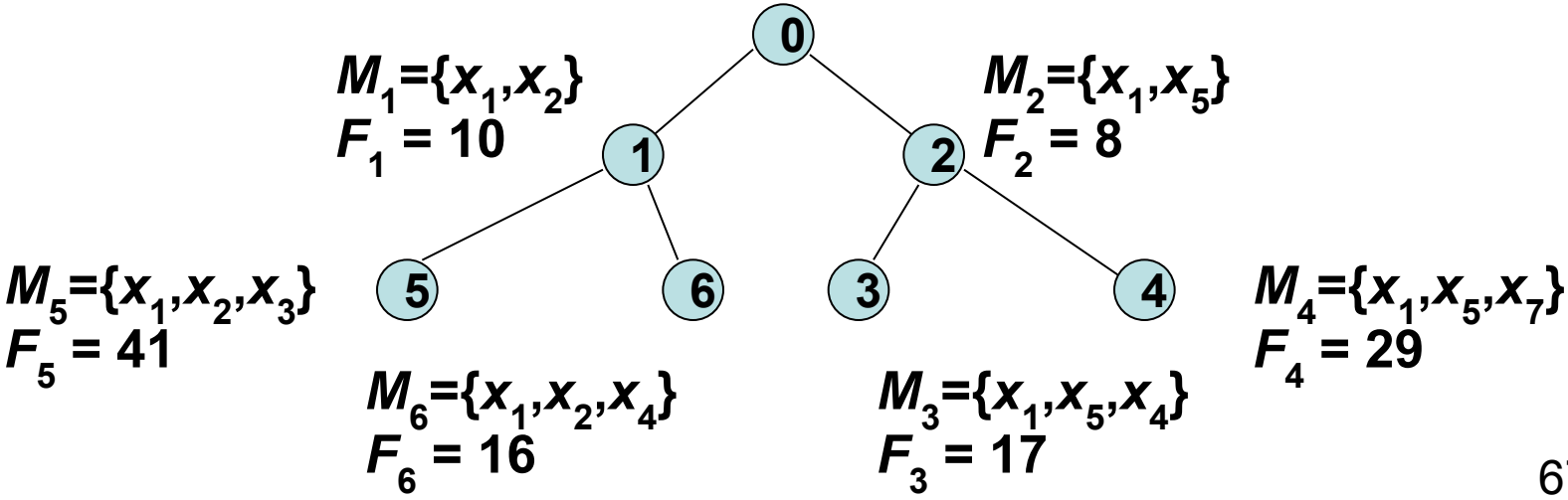
Идея алгоритма Дейкстры

Тогда, если x_i – начальная вершина, x_j – конечная вершина, x_k –промежуточная вершина, и ∃ $M_i'(X_i')$ и $M_i''(X_i'')$,

такие что $L(M_i') > L(M_i'')$, где $X_i' = \{x_i, \ldots x_r, x_k\}$, $X_i'' = \{x_i, \ldots x_t, x_k\}$ и $X_i' \neq X_i''$,

то все варианты маршрутов, порождаемые множеством M_i' можно отбросить.





4.7 Метод Форда-Фалкерсона

Этот метод является обобщением широко известного алгоритма того же названия. Метод является основой алгоритмов решения задач о максимальном потоке в сети, о максимальном паросочетании в двудольном графе и др.

Определение сети, содержательная и формальная постановки задачи отыскания максимального потока в ней были приведены в разделе????. Там же были указаны условия выполнения ограничений по пропускной способности, сохранения потока в сети и в ее вершинах.

Метод Форда-Фалкерсона гарантирует нахождение максимального потока только в сетях с целочисленными и рациональными пропускными способностями (последние могут быть приведены к целочисленным). Для случая иррациональных пропускных способностей итерационный процесс поиска решения может не сходиться

Метод Форда-Фалкерсона

Моделью сети является ориентированный граф $G^{\rightarrow}(X, <U, f, c>)$, где f функция, значение которой показывает количество сообщений по каналам в данном состоянии системы; c – функция, задающая пропускную способность каналов. Основой метода является определение дополняющего пути в графе остаточной сети $G_f^{\rightarrow}(X, <U_f, f, c>)$.

Дополняющий путь – это простая цепь из истока s в сток t в остаточной сети. Моделью остаточной сети является граф $G_f^{\rightarrow}(X, <U_f, f, c>)$, множество ребер которого определяется следующим образом:

- если $c(u_j) > f(u_j)$, где $\Gamma_1 u_j=\{x_i\}$ и $\Gamma_2 u_j=\{x_k\}$, то ребру $u_j(x_i, x_k)$ присваивается вес $c_f(u_j) = c(u_j) – f(u_j)$ и добавляется ребро $u_r \notin U$ & $ur \in U_f$ с весом $f(u_j)$ такое, что $\Gamma_1 u_r=\{x_k\}$ и $\Gamma_2 u_r=\{x_i\}$ или $u_r(x_k, x_i)$;

Метод Форда-Фалкерсона

- если $c(u_j) = f(u_j)$, то удаляется ребро u_j, соединяющее вершину x_i с вершиной x_k, и добавляется ребро $u_r \notin U$ & $u_r \in U_f$ с весом $c(u_j)$, соединяющее вершину x_k с вершиной *xi*.

Веса ребер $u_r \in U_f$ показывают на сколько можно уменьшить поток от вершины x_i к вершине x_k, если появится необходимость перераспределения потоков в сети. Отметим, что $U_f \cap U \neq \varnothing$, $U_f \not\subset U$ и $|U_f| \leq 2|U|$. В классическом изложении метода [2. Форд Л.Р., Фалкерсон Д.Р. Потоки в сетях. – М.: Мир, 1966] способ определения дополняющего пути не декларируется.

Ребра множества *U* будем называть прямыми, а множества $U_f \setminus U$ - обратными.

Метод Форда-Фалкерсона

Метод в целом заключается в следующем:

1. Задается начальное состояние сети: в графе сети $G \rightarrow (X, <U, f, c>)$ значение передаваемого потока $f(u_j)$ для всех ребер устанавливается равным нулю. В соответствии с правилами определения ребер графа остаточной сети полученный граф $G^{\rightarrow}$ $(X, <U, 0, c>)$ является иходным графом остаточной сети, в котором $Uf = U$.
2. Методом поиска в ширину (возможно в глубину с возвращением) в графе $G_f^{\rightarrow}$ ищется дополняющий путь, пропускная способность которого Δf, равная минимуму пропускных способностей составляющих его ребер, максимальна.

Метод Форда-Фалкерсона

3. В графе сети $G^{\rightarrow}$ суммарный поток F увеличивается на Δf. Значения передаваемого потока $f(u_j)$ ребер графа сети изменяются следующим образом:

– если ребро $u_j(x_i, x_k)$ дополняющего пути принадлежит множеству U, то в графе сети поток через него увеличивается на Δf;

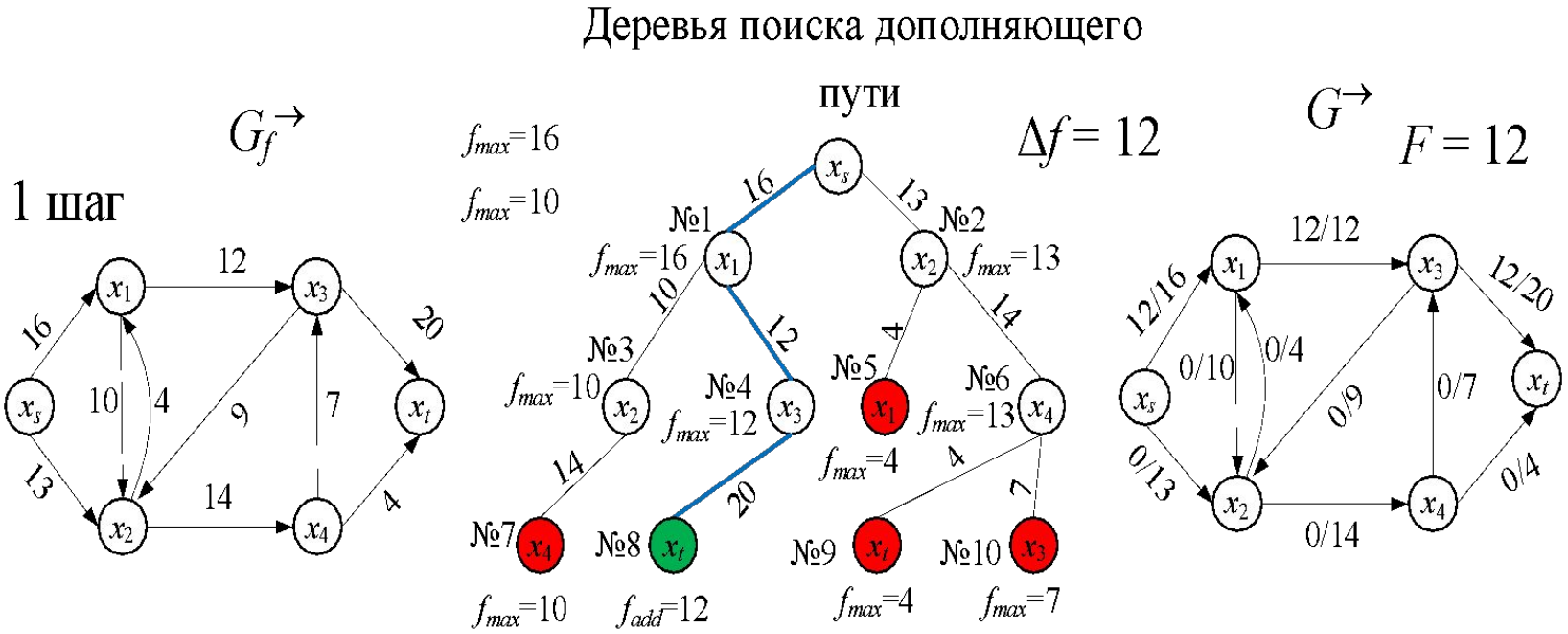
– если ребро $u_r(x_k, x_i)$ дополняющего пути принадлежит множеству $U_f \setminus U$, то в графе сети поток через ребро $u_j(x_i, x_k)$ уменьшается на Δf.

4. Определяется граф остаточной сети G_f.

Процесс повторяется с п. 2 до тех пор, пока существует дополняющий путь.

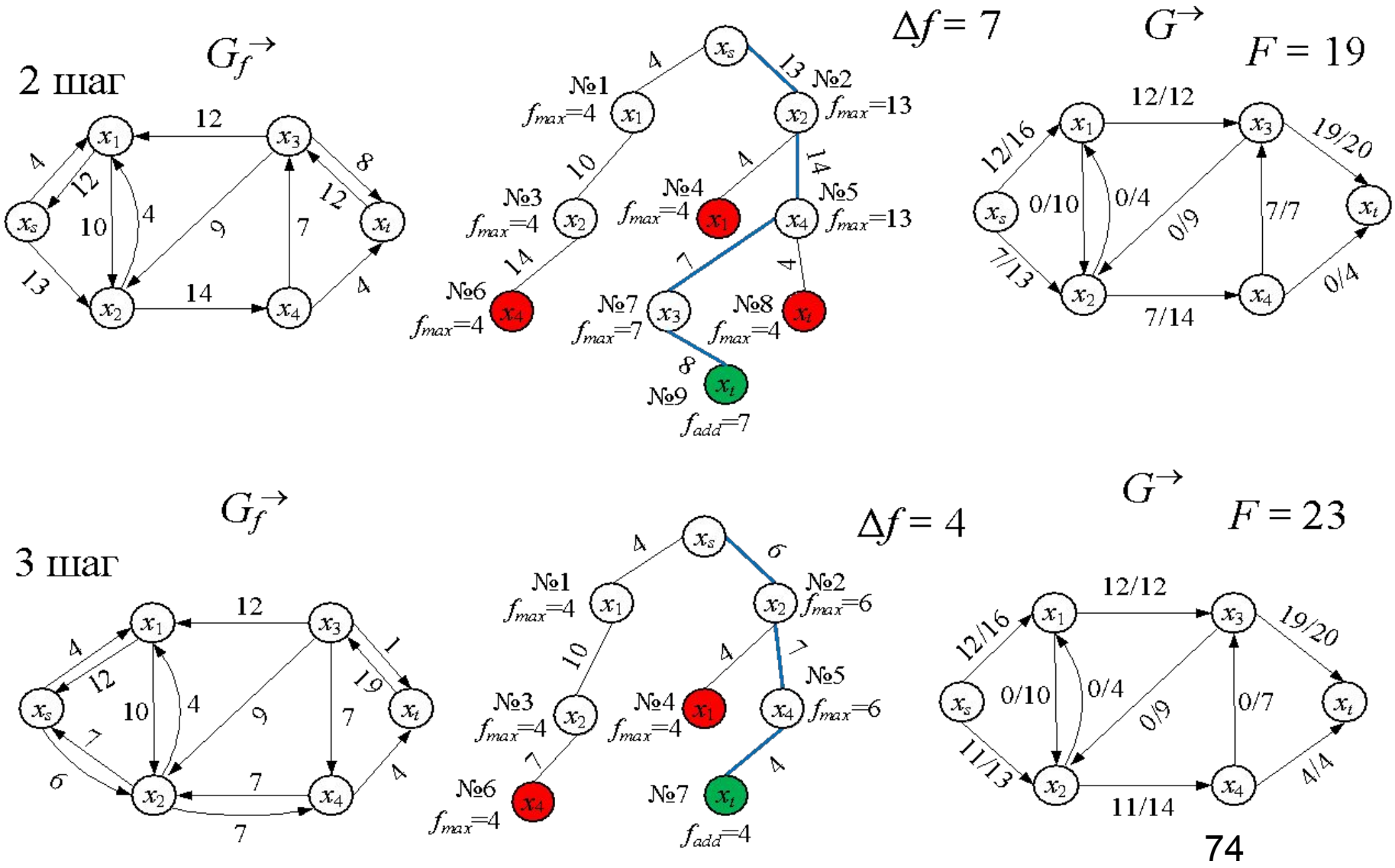
Метод Форда-Фалкерсона

Пример поиска максимального потока в сети. Рисунок иллюстрирует поиск максимального потока в сети, приведенной в [Кормен]. Около ребер в круглых скобках через наклонную черту указаны поток через ребро $f(u_j)$ и его пропускная способность $c(u_j)$.



Метод Форда-Фалкерсона

Деревья поиска дополняющего пути



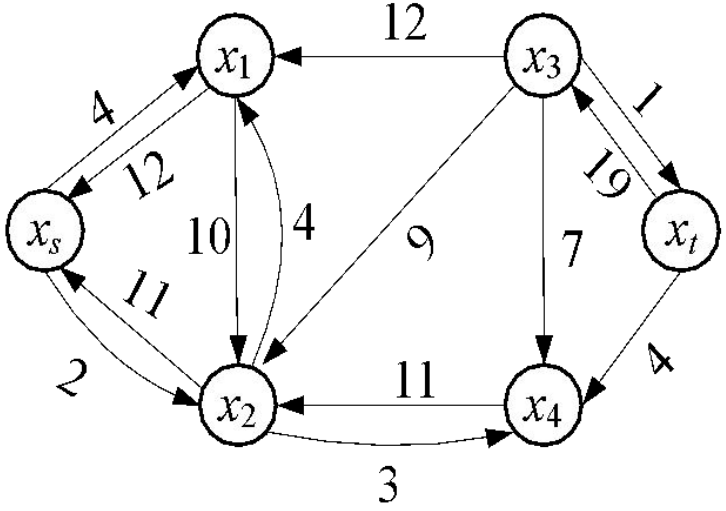
Метод Форда-Фалкерсона

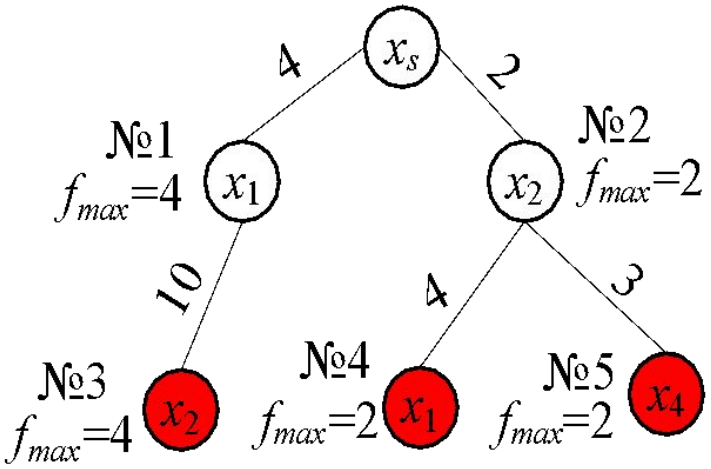
Дерево поиска

дополняющего пути

$G_f^{\rightarrow}$

4 шаг





Дополняющего пути нет - $\Delta f = 0$

Методом Форда-Фалкерсона за полиномиальное время может быть решена, например, задача о максимальном паросочетании в двудольном графе.

4.8 Метод динамического программирования

Динамическое программирование решает задачу, объединяя решение подзадач и выбирая лучший вариант из *альтернативных*.

Метод *применим*, если:

- Решение задачи можно разбить на подзадачи, у которых имеются общие подзадачи;
- структура подзадач может быть определена заранее, а их количество полиноминально зависит от размера входа (эта особенность разбиения определяется свойствами задачи);
- количество вариантов их объединения экспоненциально зависит от размера входа;
- для каждой подзадачи существует оценка, позволяющая выбирать из её альтернативных решений то, которое оптимизирует целевую функцию, ***отсекая другие***, - свойство оптимальности (например, любая часть кратчайшего пути сама есть кратчайший путь).

Метод динамического программирования

***Идея метода*:**

- решение идет от малых подзадач к большим;
- возможные (допустимые) решения получаются объединением решений предыдущих шагов, которые могут входить в разные варианты допустимых;
- оценки рассчитываются для всех допустимых решений (подзадач) ***один раз по полученным ранее оценкам решения объединяемых подзадач***.
- *Метод обеспечивает получение точного решения за полиномиальное время.*

Если оценки для подзадач рассчитывать для каждого варианта их объединения, это потребует экспоненциального времени, так как количество вариантов объединения подзадач экспоненциально.

Пример. Вычисление произведения матриц

Рассмотрим вычисление произведения n матриц

$$M = M_1 \times M_2 \times \ldots \times M_n,$$

где M_i – матрица с p строками и q столбцами. Порядок, в котором эти матрицы перемножаются, может существенно сказаться на общем количестве операций, требуемых для вычисления M, ***независимо от алгоритма, применяемого для умножения матриц.***

Требуется определить порядок перемножения матриц, при котором количество операций будет минимальным.

В данной задаче допустимые решения определяются возможным порядком умножения матриц.

Ограничимся $n = 4$ и рассмотрим произведение:

$$M = M_1[10,20] \times M_2[20,50] \times M_3[50,1] \times M_4[1,100].$$

Умножение матриц $M_i[p,q]$ на $M_j[q,r]$ требует $m_{i,j} = pqr$ операций.

Если вычислять M в порядке

$$M_1 \times (M_2 \times (M_3 \times M_4)),$$ то потребуется 125000 операций,

тогда как вычисление M в порядке

$$(M_1 \times (M_2 \times M_3)) \times M_4$$ осуществляется за 2200 операций.

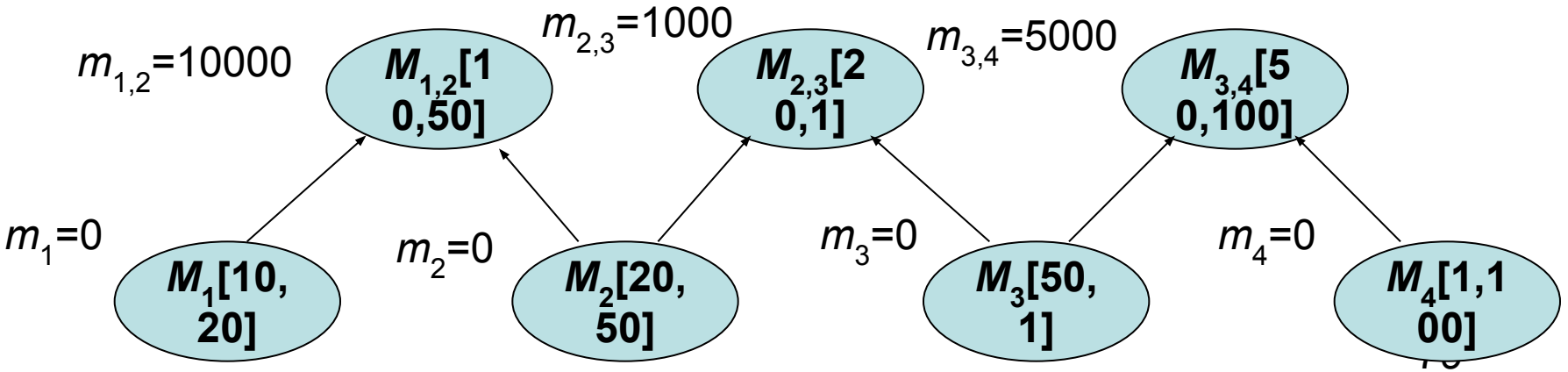
Пример. Вычисление произведения матриц

Процесс перебора всех порядков, в которых можно вычислить произведения всех матриц с целью минимизировать число операций, имеет *экспоненциальную сложность*.

Построим *дерево получения решений*, руководствуясь изложенной выше идеей метода динамического программирования.

Исходными подзадачами будем считать «умножение» каждой матрицы M_i на саму себя. Примем, что количество операций, необходимых для этого $m_i = 0$.

Возможные порядки умножения двух матриц с учетом их размеров (и число операций): $M_1 \times M_2$ ($m_{1,2} = 10 \cdot 20 \cdot 50$), $M_2 \times M_3$ ($m_{2,3} = 20 \cdot 50 \cdot 1$), $M_3 \times M_4$ ($m_{2,3} = 50 \cdot 1 \cdot 100$).



Пример. Вычисление произведения матриц

Возможны следующие порядки умножения трех матриц:

1) M_1, M_2, M_3:

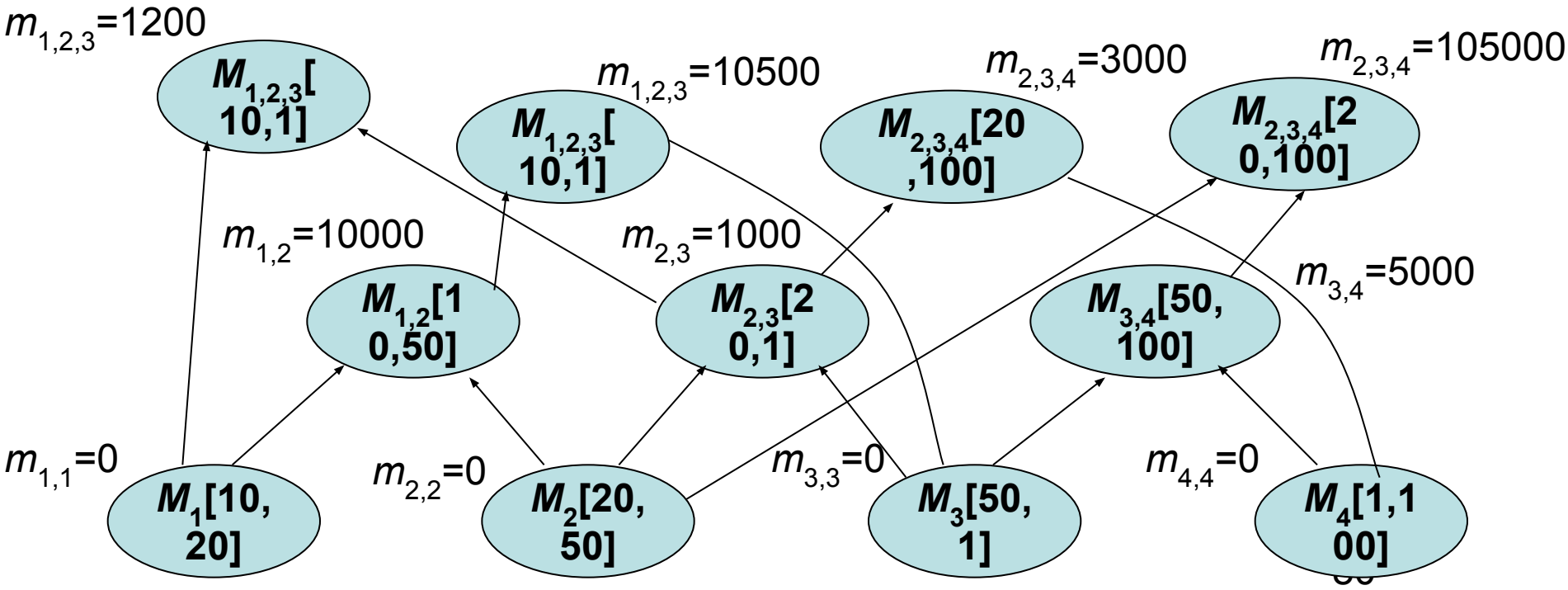
$M_1 \times (M_2 \times M_3) = M_1[10,20] \times M_{2,3}[20,1]$, $m_{1,2,3} = m_{2,3} + 10 \cdot 20 \cdot 1 = 1200$;

$(M_1 \times M_2) \times M_3 = M_{1,2}[10,50] \times M_3[50,1]$, $m_{1,2,3} = m_{1,2} + 10 \cdot 50 \cdot 1 = 10500$.

2) M_2, M_3, M_4:

$(M_2 \times M_3) \times M_4$ с оценкой $m_{2,3,4} = 3000$;

$M_2 \times (M_3 \times M_4)$ с оценкой $m_{2,3,4} = 105000$.

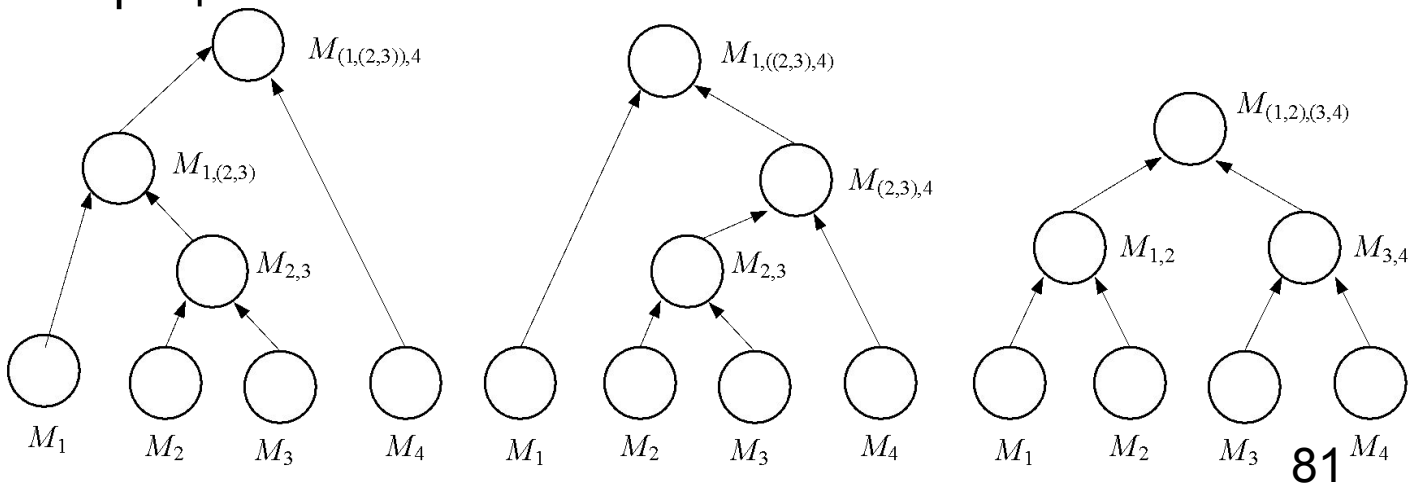


Пример. Вычисление произведения матриц

Возможные порядки умножения четырех матриц:

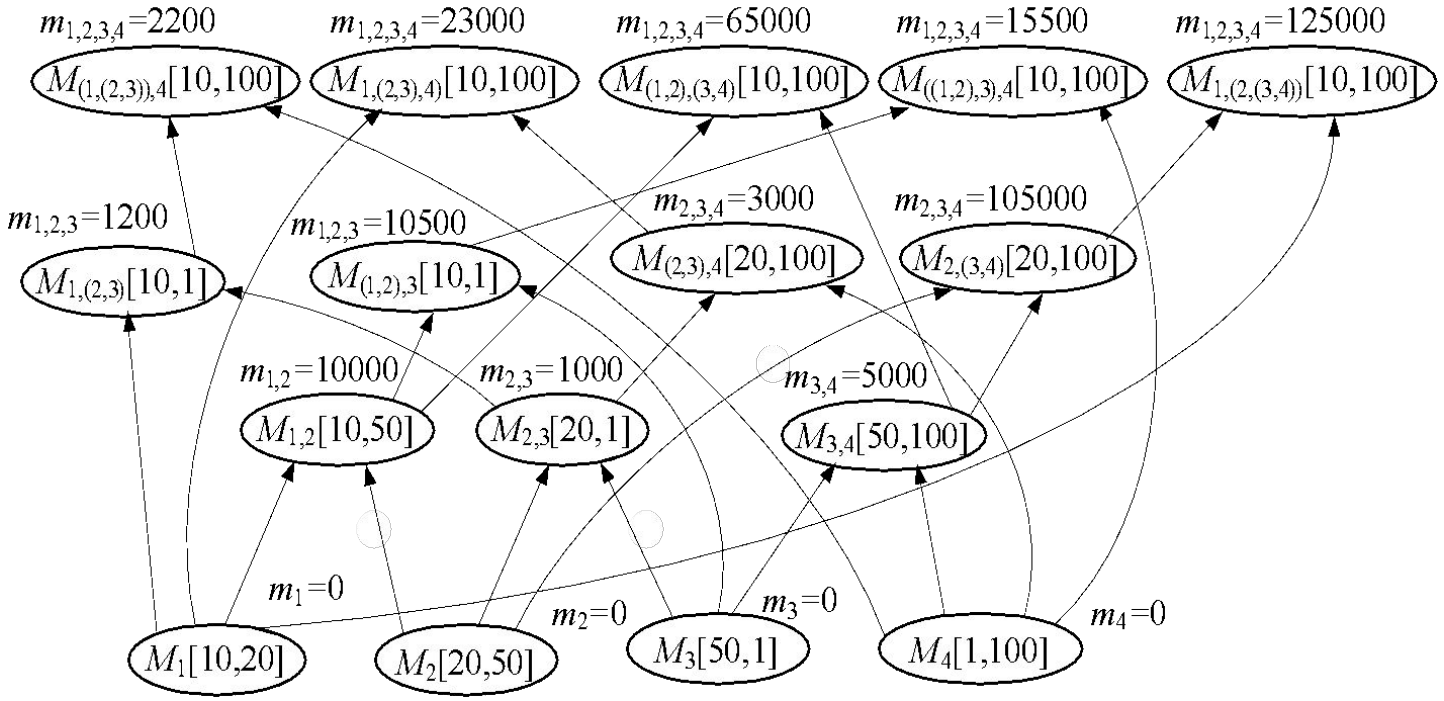
- $(M_1 \times (M_2 \times M_3)) \times M_4$ с оценкой $m_{1,2,3,4} = 2200$;
- $M_1 \times ((M_2 \times M_3) \times M_4)$ с оценкой $m_{1,2,3,4} = 23000$;
- $(M1 \times M2) \times (M3 \times M4)$ с оценкой $m1,2,3,4 = 65000$;
- $((M1 \times M2) \times M3) \times M4$ с оценкой $m1,2,3,4 = 15500$;
- $M1 \times (M2 \times (M3 \times M4))$ с оценкой $m1,2,3,4 = 125000$;

Каждому варианту порядка умножения матриц соответствует дерево свертки. На рисунке показаны деревья свертки для первых трех из указанных выше порядков умножения последовательности из четырех матриц.



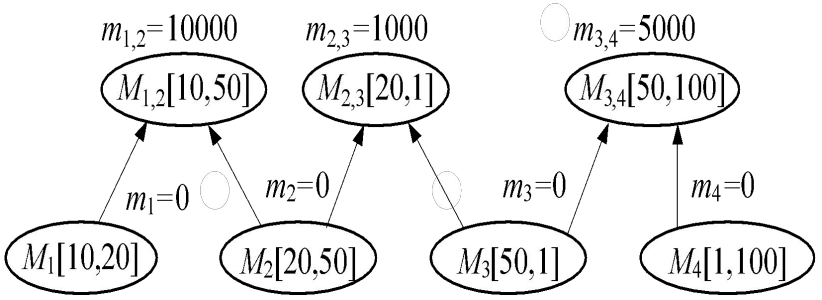
Пример. Вычисление произведения матриц

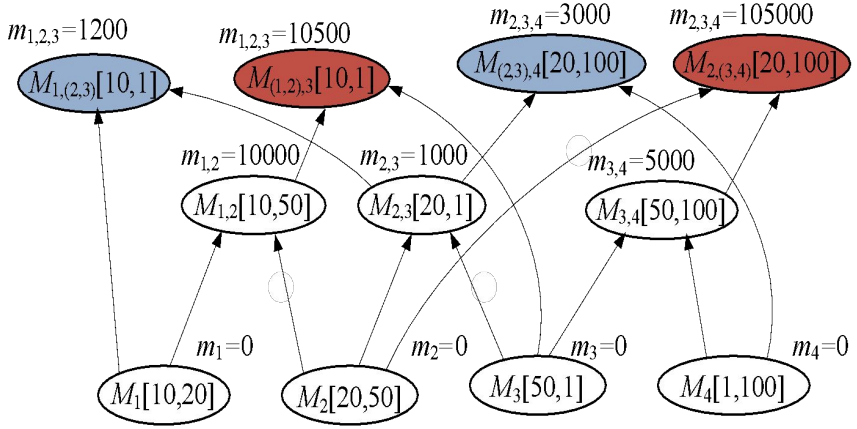
Объединением деревьев свертки будет граф возможных порядков умножения матриц последовательности. На рисунке показан такой граф для последовательности из четырех матриц и число операций умножения, необходимых для реализации вариантов подзадач и задачи в целом.

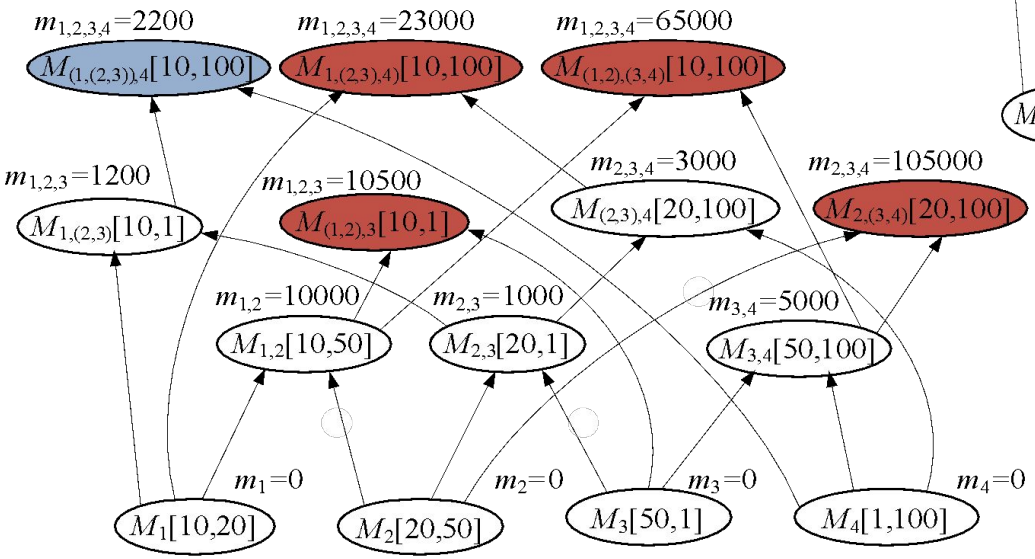


Пример. Вычисление произведения матриц

На последующих рисунках показаны первый, второй и третий шаги процесса решения задачи. Красным цветом залиты отсекаемые вершины (варианты решения подзадачи), синим – оптимальное решение..







Количество операций вычислений произведений $M_i \times M_{i+1} \times \ldots \times M_j$

Таблица оценок строится следующим образом:

$m_1 = 0$	$m_2 = 0$	$m_3 = 0$	$m_4 = 0$
$m_{1,2}$=10000	$m_{2,3}$=1000	$m_{3,4}$=5000	
$m_{1,2,3}$=1200	$m_{2,3,4}$=3000		
$m_{1,2,3,4}$ =2200			

Разбиение исходной задачи на подзадачи, количество которых полиноминально зависит от размера входа, наличие отсекающей оценки для формируемых на данном шаге допустимых вариантов решения, запоминание оценок и вычисление следующих на их основе приводят к получению алгоритмов *с полиномиальной сложностью*. Вычислительная сложность алгоритма определения оптимального порядка умножения матриц, реализующего метод динамического программирования, не хуже $O(n^3)$.

4.9 Метод итерационного улучшения

Метод выполняет *ограниченный* и нередко *упорядоченный перебор* вариантов решения, генерируя их из исходного варианта в процессе своей работы. Метод относится к классу *эвристических*.

Применяется, например для разбиения множества вершин графа на подмножества или определения некоторого варианта взаимнооднозначного соответствия элементов двух множеств, например, решения задачи компоновки – разрезания схемы на подсхемы или позиционирования – размещение микросхем на плате субблока.

Если имеется исходное приближенное решение, то генерация других вариантов решения может быть выполнена путем допустимых по смыслу задачи перестановок элементов в парах взаимно однознач-ных соответствий (перестановка или обмен). Обмены осуществляют такие, которые приводят к улучшению значения целевой функции с учетом заданных ограничений.

В общем случае обмены могут быть:

- парными,
- групповыми.

Метод итерационного улучшения - парные обмены

Положим для определенности, что лучшему варианту решения задачи соответствует минимум целевой функции F. Пусть результатом решения задачи является разрезание графа $G(X,U)$ на два куска $G_1(X_1,U_1)$ и $G_2(X_2,U_2)$, для множества вершин которых X_1 и X_2 в соответствии с определением понятия «кусок графа» справедливо: $X = X_1 \cup X_2$ и $X_1 \cap X_2 = \varnothing$. Обозначим исходный вариант решения k_0, а значение целевой функции – F_0. Предположим, что выбрана пара элементов $x_i \in X_1$ и $x_j \in X_2$ таких, что их перестановка приводит к уменьшению значения целевой функции.

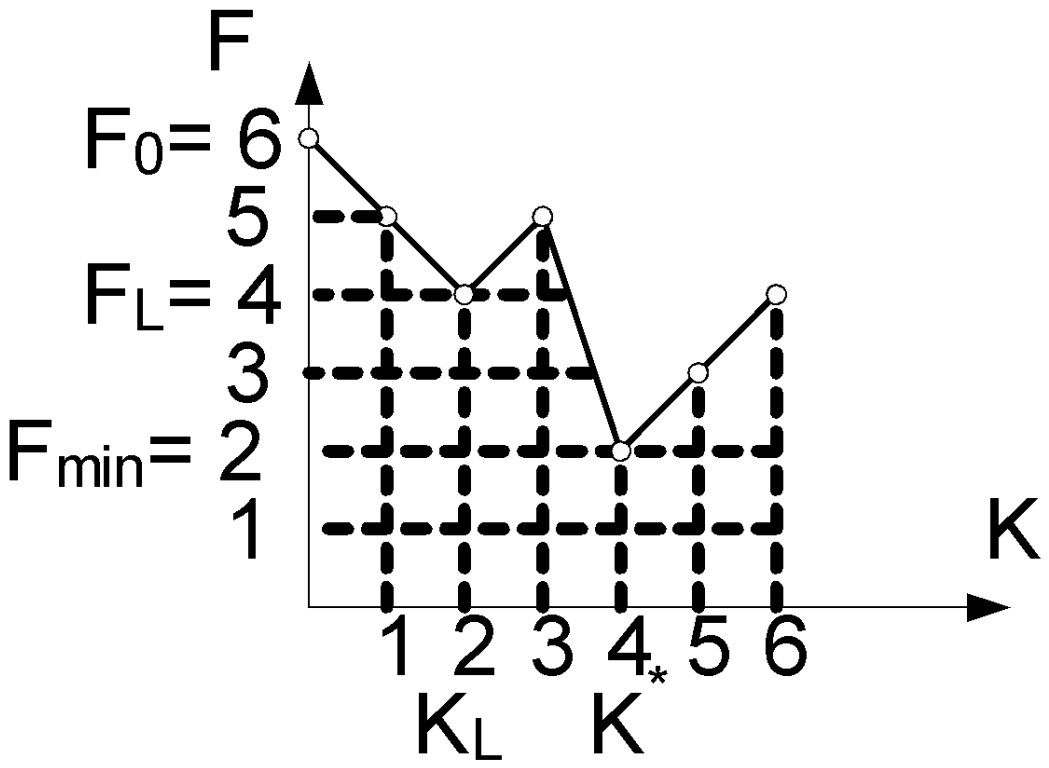
Парные обмены

После перестановки элемента x_i в X_2, а x_j в X_1, получим вариант решений k_1 и значение целевой функции F_1, причем $\Delta F_1 = F_0 - F_1 > 0$. Процесс продолжают до тех пор, пока существуют перестановки, уменьшающие значение F.

Обычно для выбора переставляемых элементов используется приращение ΔF целевой функции. Из множества возможных перестановок выбирают ту, для которой этот показатель имеет экстремальное значение. *Следует иметь в виду, что оценка этого показателя, как правило, требует значительного количества операций преобразования данных, а сами его значения могут полностью или частично меняться после выполнения каждого обмена.*

Парные обмены

Такой итерационный процесс может привести к нахождению только локального оптимума (F_L,k_L).



Пройти локальный оптимум позволяет метод групповых перестановок.

Групповой обмен

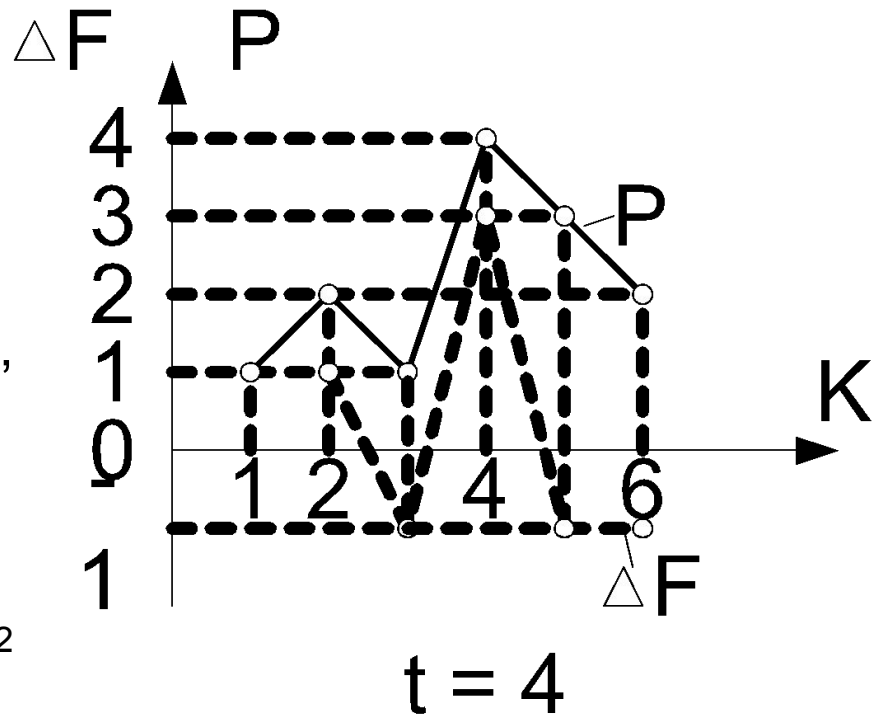
Рассмотрим один из способов определения группы.

Для всех пар вершин $x_i \in X_1$ и $x_j \in X_2$ определяют приращение ΔF.

Выбирают пару вершин с максимальным значением ΔF (оно может быть положительным, отрицательным или равным нулю), временно осуществляют перестановку этих вершин и включают их в множества X_1 и X_2.

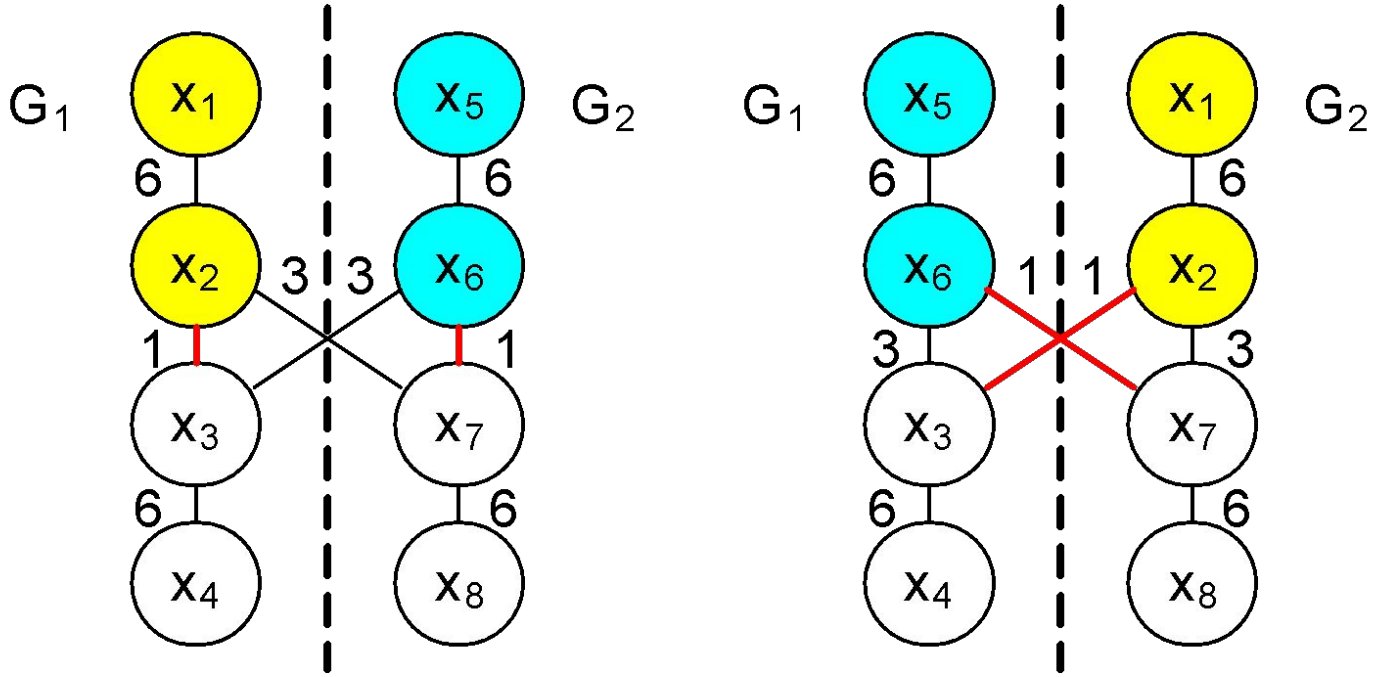
Процесс повторяют до тех пор, пока все элементы подмножеств X_1 и X_2 не поменяются местами. Строят зависимость ΔF от шага обмена. По полученной кривой определяют шаг обмена $t=k^*$, при котором значение

$$P = \sum_{k=1}^{t} \Delta F_k > 0$$



и максимально. Выполняют обмен подмножества X_{1t} из множества X_1 на подмножество X_{2t} из множества X_2.

Групповой обмен (2)



Обмен любой пары вершин не улучшает значение критерия.

Перенос вершин x_1 и x_2 в кусок графа G_2, а вершин x_5 и x_6 в кусок G_1, приведет к уменьшению значения целевой функции с 6 до 2.

Варианты окончания итерационного процесса:

- не существует перестановки, улучшающей целевую функцию ($\Delta F_k \le 0$);
- достигнута требуемая точность итераций $\Delta F_k / F_k \le \varepsilon$,
- получено удовлетворительное значение целевой функции $F_k \le F_{доп}$;
- выполнено допустимое количество итераций.

Характеристика метода итерационного улучшения

Алгоритмы, реализующие итерационный метод, требуют значительно больше машинного времени, чем алгоритмы по методу поиска в глубину.

Сокращение затрат машинного времени возможно за счет уменьшения числа возможных перестановок. Существуют различные способы упорядочивания переборов, приводящие к снижению количества обменов.

Один из таких способов заключается в следующем. Для всех $x_i \in X_1$ и $x_j \in X_2$ или $x_i \in X$ подсчитывают величину вклада каждого элемента в значение целевой функции данного решения. Элементы множества X_1 и X_2 или множества X упорядочивают по не возрастанию этой величины, если оптимум целевой функции соответствует ее минимальному значению. Обозначим после упорядочивания X_1 и X_2 как Y и Z, а X как X^y. В качестве кандидатов на обмен на k-ом шаге рассматривают элементы, расположенные в соответствующих по порядку позициях упорядоченных множеств X_1 и X_2, т.е. пары (y_k, z_k), (y_{k+1}, z_{k+1}) и т. д. или для X - пары, составленные x^y_k с элементами подмножества $\{x^y_{k+1}, x^y_{k+2}, \ldots x^y_n\}$. После окончания итерационного процесса можно подсчитать новые значения вкладов элементов в целевую функцию и повторить процесс.

4.10 Генетический метод

Все рассмотренные ранее методы основаны на детерминированном поиске решения. Если оценочная функция не является отсекающей, то гарантируется поиск локально оптимального решения.

Генетический метод сочетает *направленный поиск, основанный на эвристиках, с элементами случайности*, цель которых – исключение «застревания» поиска в точках локального оптимума.

Терминология:

- *хромосома* – любое решение задачи синтеза, т.е. определенным образом организованная совокупность выходных параметров, характеризующих решение задачи;
- *гены* (*аллели*)– поля хромосом, содержащие значения проектных параметров;
- *начальная популяция* – исходное множество решений;
- *кроссинговер* (*кроссовер*) – оператор, осуществляющий *скрещивание хромосом,* т. е. взаимный обмен генами между хромосомами предварительно выбранной пары *родителей*, порождающий пару новых решений;
- *мутация* – случайное перестроение генов отдельных решений – порождает новые решения и служит для исключения «застревания» поиска в точках локального оптимума.

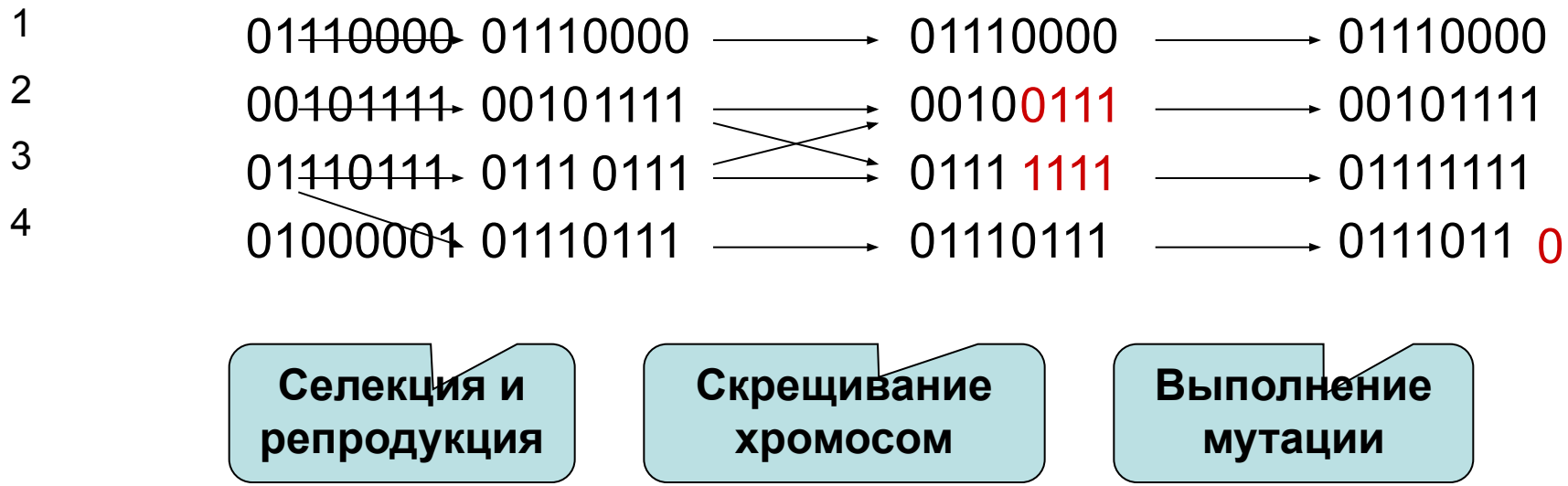
Сущность генетического метода

1. Задается начальная популяция и рассчитываются значения локальной целевой функции.
2. Выбирается лучшее решение и копируется на место худшего (селекция и репродукция).
3. Определяется пара родителей:
   - случайным образом, при этом вероятность выбора хромосом с лучшими значениями целевой функции должна быть выше;
   - по лучшему значению целевой функции (детерминированно).
4. Применяется кроссинговер и рассчитывается целевая функция для полученной пары решений. Гены, подлежащие обмену:
   - могут выбираться случайно с учетом получения допустимых решений, при этом вероятность выбора может зависеть от значения целевой функции;
   - назначаться детерминированно.
5. Выполняется замещение родительской пары полученной парой или исключение из нового множества решений пары с худшими значениями целевой функции.
6. Осуществляется мутация случайного или выбранного гена и оценка полученного решения.

Для завершения процесса могут использоваться те же условия, что и у итерационного метода.

Пример генерационного цикла получения строки из 8 двоичных символов с максимальным количеством единиц

Популяция состоит из четырех хромосом, ее размер остается неизменным:



Эффективное применение генетического метода невозможно без использования содержательных сведений о сути решаемой задачи.

Эти сведения позволяют сформулировать локальную целевую функцию или совокупность локальных критериев и разработать эффективные эвристики селекции, скрещивания и мутации.

Характеристика генетического метода

Алгоритмы, реализующие генетический метод, гарантируют получение лишь *приближенного* решения.

Обладая достоинствами детерминированных алгоритмов (большей эффективностью по сравнению с вероятностными), генетические алгоритмы за счет использования вероятностного подхода при селекции, выборе пары родителей, работы кроссинговера и мутации должны быть лишены таких недостатков, как попадание в локальные экстремумы и зацикливание.

Максимальное усиление любого из достоинств генетического метода способно свести остальные его преимущества на нет, превращая его в чисто вероятностный или полностью детерминированный метод.

Простейшим примером вырожденного случая является итерационный алгоритм парных обменов, в котором репродуцируется одно начальное решение и выполняется парный обмен генов, т. е. скрещивание хромосом.

4.11 Метод параллельно-последовательной *n*-арной свертки

Метод предназначен для объединения элементов множества X в некоторые подмножества.

В графе в качестве свертываемых множеств могут рассматриваться как множество вершин X, так и ребер U.

До начала свертки в качестве кандидатов на объединение рассматривают все элементы множества X. На первом шаге объединяют удовлетворяющие заданному критерию подмножества множества X. В общем случае подмножества могут состоять из одного, двух или более элементов. Сформированные подмножества рассматривают как элементы нового множества, по отношению к которому процедура может быть повторена.

Включение в состав формируемого подмножества элементов или частей других подмножеств *невозможно*, поскольку если X_1 и X_2 – подмножества, сформированные к некоторому шагу свертки, то возможно получение нового подмножества $Z = \{X_1, X_2\}$ и невозможно – $Z = \{X'_1, X'_2\}$, в котором $X'_1 \subset X_1$ и $X'_2 \subseteq X_2$ или $X'_1 \subseteq X_1$ и $X'_2 \subset X_2$.

Следовательно *формируются* только *непересекающиеся* подмножества.

Метод параллельно-последовательной свертки

При объединении подмножеств (элементов) происходит отсечение других вариантов состава подмножеств. Очевидно, что *точность получаемого решения* определяется тем, *является ли оценка* выбора кандидатов на объединение *отсекающей*, т. е. точной, или оценкой выбора перспективного решения. В случае, если это оценка выбора перспективного решения, то метод гарантирует лишь получение приближенного решения, так как он не предусматривает сохранение и рассмотрение других вариантов состава подмножеств.

Метод параллельно-последовательной свертки

Возможны два варианта окончания процесса свертки:

- процесс заканчивается, когда все элементы множества свернуты в один;

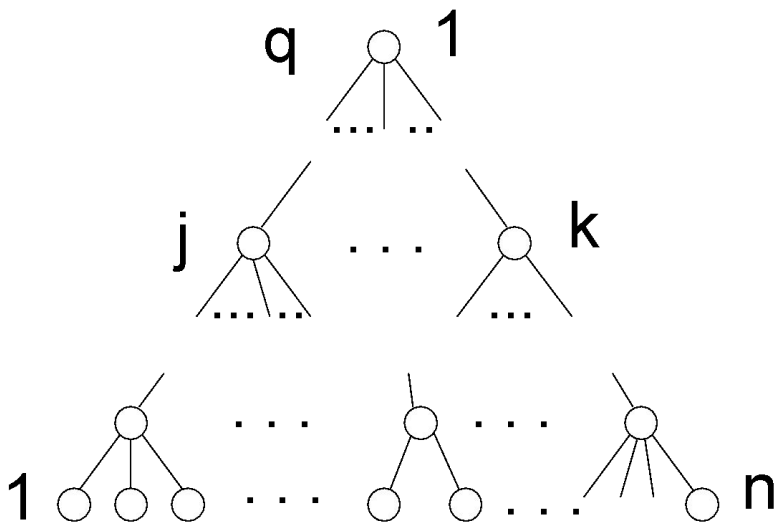
  С практической точки зрения это требуется, если такое событие означает установление того факта, что множество в целом обладает некоторым определяющим свойством, например, что все множество вершин гиперграфа некоторой схемы является минимальным массивом.

- в тех случаях, когда на подмножества наложены ограничения (например, количество подмножеств, их элементов, суммарный вес элементов), процесс свертки заканчивается при удовлетворении этих условий.

По мере получения подмножеств, для которых выполняются заданные ограничения, эти подмножества исключаются из рассмотрения.

Метод параллельно-последовательной свертки

Процесс свертки можно наглядно представить в виде дерева, в котором вершины *j*-го уровня сопоставлены подмножествам, являющимся кандидатами на объединение, а ребра указывают на вхождение двух или более подмножеств в новое подмножество. На нижнем уровне вершины соответствуют элементам исходного множества.



Формирование на каждом шаге свертки из *n* элементов множества *X* подмножеств, включающих 2, 3 и т.д. элементов, требует перечисления всех соответствующих сочетаний, вычисления для них значения заданного критерия и выбора оптимальных вариантов. Это в пределе приводит к *полному перебору*. Избежать полного перебора позволяет попарное объединение элементов на каждом шаге свертки. Такая модификация называется *методом двоичной свертки*.

Двоичная свертка

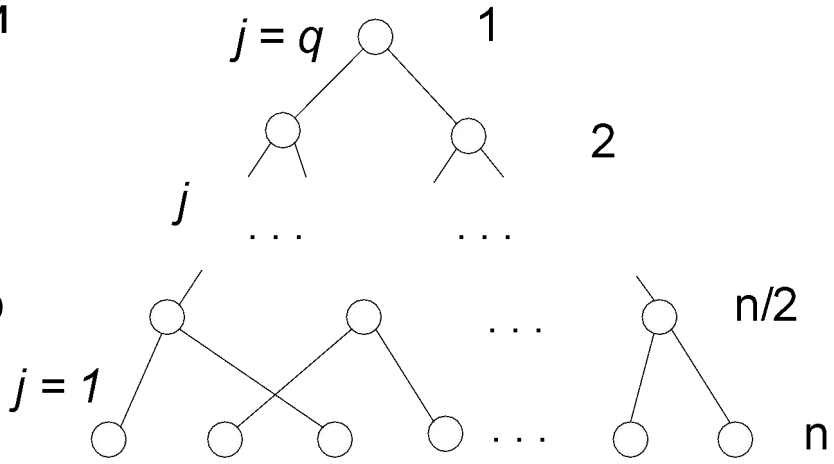
Различают уравновешенную и неуравновешенную двоичную свертку.

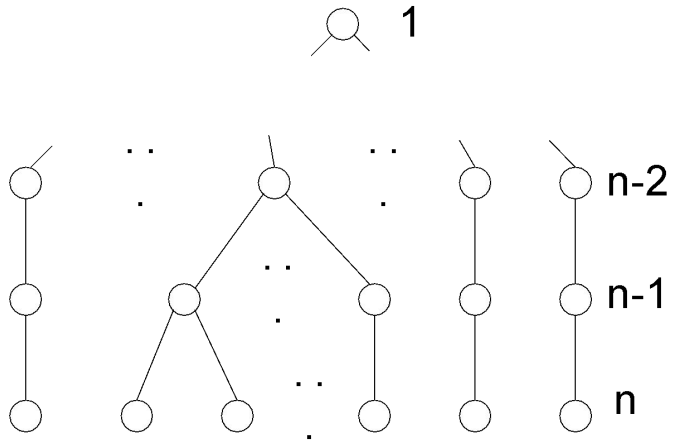
При *уравновешенной* или параллельной двоичной свертке вновь образованный (факторизованный) элемент переходит на следующий уровень дерева, т. е. сформированное подмножество становится элементом нового множества. Свертка на текущем уровне продолжается для оставшихся элементов. Очевидно, что на каждом уровне свертки число элементов вдвое меньше, чем на предыдущем.

Количество элементов на уровне j определяется по формуле:

$$n_j = n/2^{j-1},\ j = \overline{1,q},\ \text{где}\ q = ln_2\ n.$$

При *неуравновешенной* двоичной свертке элементы или подмножества, вошедшие в новое подмножество, исключаются из множества, а это новое подмножество включается в него и участвует в процессе свертки.





Характеристика метода двоичной свертки

Метод двоичной свертки *не гарантирует нахождение точного решения*.

Действительно, пусть на некотором шаге свертки имеется подмножество $X_k \subset X$, ни одна из пар x_i, x_j элементов которого не может быть свернута, так как значения заданного критерия не удовлетворяет требованию к нему, и ни один из элементов $x_i \in X_k$ не может быть объединен ни с каким из сформированных подмножеств. Если среди элементов подмножества X_k существуют подмножества, например, из 3-х элементов, которые удовлетворяют требованиям, предъявляемым к значению соответствующего критерия, то такие подмножества не будут найдены.

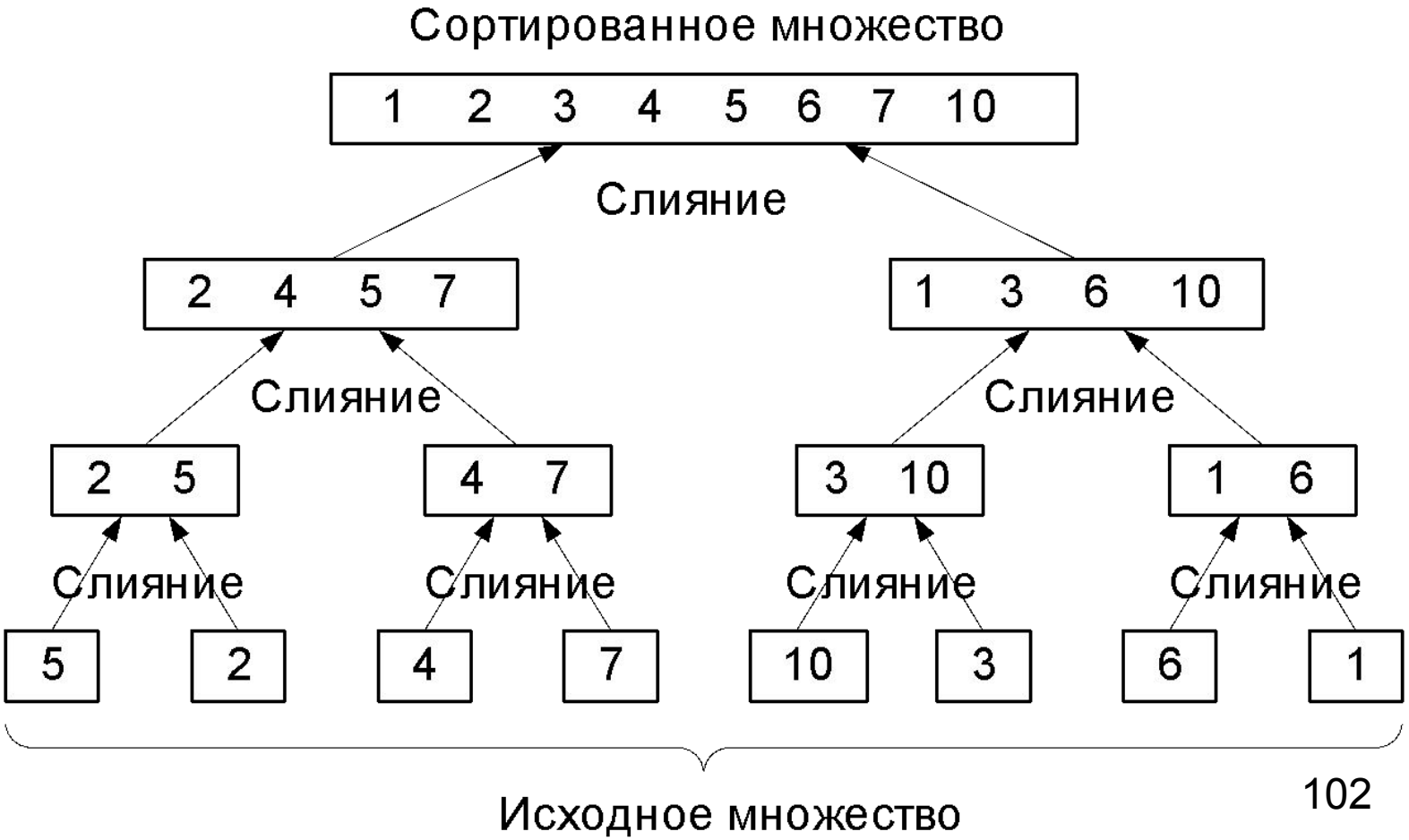
Примером может служить задача выделения всех минимальных массивов гиперграфа. В гиперграфе может вообще не существовать минимальных массивов из 2-х вершин и существовать минимальные массивы из 3-х, 4-х и т.д. вершин.

Сказанное однако не означает, что на основе метода двоичной свертки нельзя построить точный алгоритм.

Алгоритм сортировки слияниями

Алгоритм реализует метод двоичной свертки и является точным.

Попарно сравнивая числа исходного множества, их заносят в подмножества, состоящие из двух, четырех и т.д. чисел, *записывая сначала меньшие числа* (для сортировки по возрастанию).



Характерные особенности метода двоичной свертки при сортировке слияниями

- критерий слияния подмножеств – их принадлежность к одному уровню свертки;
- сортировка обеспечивается упорядочиваем элементов каждого подмножества;
- окончание процесса свертки означает реализацию на множестве чисел X отношения порядка.

Оценим количество операций сравнения N_c, необходимых для упорядочивания n элементов множества X алгоритмом слияния:

- если n – степень двойки, то количество подмножеств (вершин) на уровне j – $n_j = n/2^{j-1}$, где $j = \overline{1,q}$, $q = log_2 n$;
- число элементов в подмножестве уровня j – 2^{j-1};
- количество сравнений элементов двух подмножеств уровня j - $2 \cdot 2^{j-1} - 1 = 2^j - 1$.

Суммарное количество сравнений на уровне j – $(2^j - 1)n_j/2 = n(1 - 2^{-j})$.

Суммируя по $j = \overline{1,q}$, получим

$$N_c = n \sum_{j=1}^{q} (1 - 2^{-j}).$$

Очевидно, что $N_c < n \log_2 n$, в то время как алгоритм сортировки выбором требует $N_в = n(n-1)/2 - 1$ операций сравнения чисел.