Илья И. Колыхматов, Антон В. Яковлев

Задача о минимальном разрезе

Введение

Основные понятия теории потоков входят в число вещей, которые хорошо бы знать любому грамотному программисту. Обычно про них коротко пишут в последних главах книг по computer science. А жаль – предмет достаточно интересен, чтобы рассказать о нем не торопясь.

Мы предполагаем, что читатель знаком с задачей о максимальном потоке и методом Форда–Фалкерсона [1]. Данная задача очень доступно изложена в книге [2]. Для закрепления своих знаний о методе Форда–Фалкерсона рекомендуется посмотреть визуализатор алгоритма Форда–Фалкерсона.

Сначала мы кратко опишем метод «проталкивания предпотока». Знакомство с ним необходимо, поскольку некоторые алгоритмы для задачи о минимальном разрезе (в частности, алгоритмы Гомору–Ху и Хао–Орлина) существенным образом используют идею проталкивания предпотока в сети.

Затем мы сформулируем задачу о минимальном разрезе и перейдем к рассмотрению некоторых ее приложений. Мы сосредоточимся на поиске минимального разреза в сети, в которой не указаны исток и сток. Классическая задача о максимальном потоке предполагает наличие выбранного истока s и стока t. Мы будем ссылаться на любой $(S, V \setminus S)$ разрез, где $s \in S$ и $t \in N \setminus S$, как на s-t разрез и называть задачу определения минимальной возможной пропускной способности s-t разреза задачей о минимальном s-t разрезе. Хотя задача о s-t разрезе имеет широкий круг приложений, во многих ситуациях желательно определить минимальный разрез в сети, в которой нет выделенного истока или стока. В таком случае желательно разделить множество вершин V сети на две непустых части S^* и $V \setminus S^*$ так, чтобы пропускная способность разреза $(S^*, V \setminus S^*)$ была наименьшей из возможных. Задачу в такой постановке мы будем называть задачей о нестрогом минимальном разрезе, чтобы подчеркнуть, что нет выделенного истока и стока.

Если в процессе изучения статьи возникнет необходимость уточнения какого-либо термина, определения или обозначения, обращайтесь к разделу «Определения и обозначения».

1 Алгоритм проталкивания предпотока

Напомним, что задача о максимальном потоке в сети состоит в следующем: для данной сети G с истоком s и стоком t требуется найти поток максимальной величины. В этом разделе мы опишем метод проталкивания предпотока для отыскания максимального потока в сети. Наиболее быстрые из известных алгоритмов для этой задачи используют именно его. Этот метод предложен Гольдбергом и Тарьяном в 1988 году [3].

1.1. Некоторые обозначения

В дальнейшем мы будем использовать следующие обозначения:

- $G = (V, E)$ – граф с множеством вершин V и множеством ребер E;
- $n = |V|$ – число вершин;
- $m = |E|$ – число ребер;
- s – исток;
- t – сток;
- $c(u, v)$ – пропускная способность ребра (u, v);
- $f(u, v)$ – поток вдоль дуги (u, v);
- $c_f(u, v) = c(u, v) - f(u, v)$ – остаточная пропускная способность;
- $G_f = (V, E_f)$, где $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$ – остаточная сеть;
- $\lambda_{s,t}(G)$ – величина минимального s-t разреза;
- $\lambda(G)$ – величина минимального разреза.

1.2. Начальные сведения

В этом разделе мы дадим формальные определения важных понятий.

Рассмотрим ориентированную сеть $G = (V, E)$ с истоком s и стоком t, в которой каждому ребру $(u, v) \in E$ поставлено в соответствие неотрицательная пропускная способность $c(u, v) \geqslant 0$. Мы будем искать максимальный поток от истока к стоку.

Определим избыток $e_f(v)$ для вершины $v \neq \{s, t\}$ как разницу входящего в v и исходящего из v потоков. Вершину $v \neq \{s, t\}$ с положительным избытком $e_f(v) > 0$ назовем активной.

Метки расстояний – это заданная на вершинах неотрицательная функция $d\colon V \to \mathbb{Z}_+$, удовлетворяющая следующим условиям:

- $d(t) = 0$;
- $d(s) = n$;
- $d(u) \leqslant d(v) + 1$ для всех дуг (u, v) остаточной сети G_f.

Дуга $(u, v) \in E_f$ остаточной сети $G_f = (V, E_f)$ называется допустимой, если $d(u) = d(v) + 1$. Вдоль допустимой дуги возможно проталкивание предпотока.

В процессе работы алгоритма мы не требуем выполнения закона сохранения потока, довольствуясь выполнением свойств предпотока. Предпотоком будем называть функцию $f\colon V \times V \to \mathbb{R}$, которая

- кососимметрична: $f(u, v) = -f(v, u)$ для всех u, v из V;
- удовлетворяет ограничениям, связанным с пропускными способностями: $f(u, v) \leqslant c(u, v)$ для всех u, v из V;
- удовлетворяет ослабленному закону сохранения: $\sum\limits_{v \in V} f(v, u) \geqslant 0$ для всех $u \in V \setminus \{s, t\}$.

1.3. Идея алгоритма

В методе Форда–Фалкерсона мы в каждый момент имеем дело, например, с потоком жидкости по трубам от истока к стоку; на каждом шаге мы увеличиваем этот поток, находя дополняющий путь. Жидкость никуда не проливается по дороге.

В алгоритме проталкивания предпотока избыток жидкости в каждой вершине (месте соединения труб) сливается. Кроме того, важную роль играет целочисленный параметр, который будет называться меткой расстояния. Метка расстояния истока всегда равна n, а стока – нулю. Все остальные вершины изначально имеют метки расстояний 0, и со временем эти метки увеличиваются.

Для начала мы отправляем из истока столько жидкости, сколько нам позволяют пропускные способности выходящих из истока труб (это количество равно пропускной способности разреза $(s, V \setminus s)$). Возникающий (в соседних с истоком вершинах) избыток жидкости сперва просто выливается, но затем он будет направлен дальше.

Рассмотрим какую-либо вершину u в ходе работы алгоритма. Может оказаться, что в ней есть избыток жидкости, но все трубы, по которым еще можно отправить жидкость из u куда-то (все ненасыщенные трубы), ведут в вершины с той же или большей меткой расстояния. В этом случае мы можем выполнить другую операцию, называемую изменением метки расстояния вершины u. После этого метка расстояния вершины u становится на единицу больше наименьшей метки расстояния ее соседа, куда ведет ненасыщенная труба – другими словами, мы увеличиваем метку расстояния u ровно настолько, чтобы появилась ненасыщенная труба.

В конце концов мы добьемся того, что в сток приходит максимально возможное количество жидкости (для данных пропускных способностей труб). При

этом предпоток может еще не быть потоком, то есть избыток жидкости сливается. Продолжая выполнять операции увеличения меток вершин (отметим, что у некоторых вершин метка расстояний могут стать больше, чем у истока), мы постепенно отправим избыток обратно в исток (другими словами, сократим поток жидкости от истока) – и превратим предпоток в поток, который и будет максимальным.

В ходе работы алгоритм манипулирует значением предпотока и меток расстояний, а операции проталкивания и изменения метки вершины применяются, пока есть активные вершины.

Отметим одно важное свойство функции меток расстояний d.

Лемма 1.1. *Пусть f – предпоток, а d – функция меток расстояний. Тогда в остаточной сети G_f не существует пути из истока s в сток t.*

Доказательство. Пусть это не так и такой путь существует. Устраняя циклы, можно считать, что он простой и потому его длина l меньше n: $l < n$. Поскольку d – метки расстояний, для каждой дуги (v_i, v_{i+1}) пути выполнено свойство $d(v_i) \leqslant \leqslant d(v_{i+1}) + 1$. Последовательно применяя это свойство, получим:

$$d(v_0) \leqslant d(v_1) + 1 \leqslant d(v_2) + 2 \leqslant \ldots \leqslant d(v_l) + l.$$

Известно, что $d(v_0) = d(s) = n$ и $d(v_l) = d(t) = 0$. Из этого следует, что $n \leqslant l$. Но $l < n$. Значит, в пути есть ребро, где метка расстояния уменьшается по крайней мере на 2 – а такое ребро не может входить в остаточную сеть. □

1.4. Инициализация алгоритма

Для «запуска» алгоритма необходимо инициализировать значение предпотока и меток расстояний. Сначала вдоль каждой дуги, исходящей из истока s, направим такое количество потока, которое совпадает с пропускными способностями соответствующих дуг. Поток по всем остальным дугам пока равен 0. Установим начальные значения меток расстояний: $d(s) = n$ и $d(v) = 0$ для всех остальных вершин. Заметим, что любая дуга из вершины s в вершину v не является дугой остаточной сети, поэтому мы не требуем, чтобы выполнялось свойство $d(s) \leqslant d(v) + 1$. Для любой дуги остаточной сети (v, w) метка расстояния $d(v) = 0 \leqslant 0 + 1 = d(w) + 1$, поэтому функция d меток расстояний задана корректно.

Теперь можно приступать к выполнению операций проталкивания и изменения меток вершин, а значит пришло время описать эти операции!

1.5. Операция проталкивания

Операция проталкивания заключается в перемещении избытка в соседнюю вершину с меньшей меткой расстояния. Для этого нам нужны активные вершины (вершины с положительным избытком) и допустимые дуги (дуги с положительной остаточной пропускной способностью, по которым можно направить увеличенный поток). Итак, нам нужна дуга остаточной сети $(v, w) \in G_f$ с избытком $e_f(v) > 0$, причем $d(v) = d(w) + 1$. При этом поток из вершины v в ее соседа w растет – его увеличение ограничено избытком $e_f(v)$ и остаточной пропускной способностью ребра (v, w). Мы можем увеличить $f(v, w)$ на величину $\delta = \min\{c_f(v, w), e_f(v)\}$. В результате избыток $e_f(w)$ возрастет на δ, а $e_f(v)$ и $f(w, v)$, наоборот, уменьшатся на δ. Если после проталкивания остаточная пропускная способность $c_f(v, w) = 0$, то проталкивание называется насыщающим, иначе – ненасыщающим.

Если больше нельзя выполнить операцию проталкивания, выполняют операции изменения меток вершин.

1.6. Операция изменения метки вершины

Данная операция выполняется для активной вершины v, когда все исходящие из v дуги не являются допустимыми. При выполнении этой операции метка расстояния $d(v)$ заменяется на $\min\limits_{(v,w)\in E_f} \{d(w) + 1\}$.

1.7. Свойства операций проталкивания и изменения меток вершин

Следующие леммы описывают важные свойства операций проталкивания и изменения меток вершин.

Лемма 1.2. *Во время выполнения операций проталкивания и изменения меток вершин ни одно из трех свойств предпотока не нарушается, а функция d остается корректной функцией меток расстояний.*

Доказательство. Операция изменения метки вершины v не изменяет значения потока. Мы заботимся о том, чтобы определение функции меток расстояний не нарушалось для остаточных ребер, исходящих из v. Что же касается входящих ребер, то с ними не может быть проблем, так как высота вершины v только возрастает.

Рассмотрим теперь операцию проталкивания. После ее применения свойства предпотока не нарушаются. Однако эта операция может привести к созданию новых ребер в остаточной сети за счет увеличения потока вдоль дуги (v, w), вдоль

которой до того не было потока. Поэтому мы требуем, чтобы после проталкивания $d(w) \leqslant d(v) + 1$, но операция проталкивания применяется только в случае, когда $d(v) = d(w)+1$: $d(w) = d(v)-1 \leqslant d(v)+1$. Итак, метки расстояний остаются корректными после выполнения операции проталкивания. □

Лемма 1.3. *Пусть d – метки расстояний, а вершина v является активной. Тогда можно выполнить операцию проталкивания из вершины v в соседнюю вершину, либо операцию изменения метки вершины v.*

Доказательство. Поскольку d – метки расстояний, то $d(v) \leqslant d(w) + 1$ для любого остаточного ребра (v, w). Если в v невозможно проталкивание, то для всех остаточных ребер (v, w) выполнено неравенство $d(v) < d(w) + 1$, из чего следует, что $d(v) \leqslant d(w)$, и для вершины v может быть выполнена операция изменения ее метки. □

1.8. Общая схема алгоритма

Сам алгоритм выглядит довольно просто: пока возможны операции проталкивания или изменения меток расстояний вершин, выполнять эти операции.

Приведем весь алгоритм и основные операции в псевдокоде:

```
procedure PREPROCESS
begin
    for each (для каждой) вершины u ∈ V do
        d(u) ← 0
        e_f(u) ← 0
    for each (для каждого) ребра (u, v) ∈ E do
        f(u, v) ← 0
        f(v, u) ← 0
    for each (для каждой) вершины {u: (s, u) ∈ E} do
        f(s, u) ← c(s, u)
        f(u, s) ← −c(s, u)
        e_f(u) ← c(s, u)
    d(s) ← n
end

procedure PUSH/RELABEL(i)
begin
    if сеть содержит допустимую дугу (i, j)
    then протолкнуть δ ← min{e_f(i), c_f(i, j)} единиц потока из i в j
    else d(i) ← min{d(j) + 1 : (i, j) ∈ E_f и c_f(i, j) > 0}
end
```

```
algorithm PUSH-RELABEL
begin
    PREPROCESS
    while (пока) есть активная вершина i do
        выбрать активную вершину i
        PUSH/RELABEL(i)
end
```

1.9. Анализ алгоритма

Сначала мы докажем, что если алгоритм остановится, то предпоток f в этот момент будет максимальным потоком. Затем мы докажем, что алгоритм действительно остановится. Чтобы убедиться, что алгоритм проталкивания предпотока останавливается, укажем верхние границы отдельно для числа операций изменения меток расстояний, насыщающих и ненасыщающих проталкиваний. После этого станет ясно, что время работы алгоритма есть $O(n^2m)$.

Лемма 1.4. *Если алгоритм проталкивания предпотока, примененный к сети $G = (V, E)$ с истоком s и стоком t, останавливается, то получающийся предпоток f будет максимальным потоком для G.*

Доказательство. Лемма 1.3 гарантирует, что в момент остановки активных вершин в сети нет (избыток каждой равен нулю). Значит, в этот момент предпоток является потоком. По лемме 1.2 функция d будет корректной функцией меток расстояний, и поэтому в силу леммы 1.1 в остаточной сети G_f нет пути из s в t. По теореме о максимальном потоке и минимальном разрезе поток f максимален. □

Лемма 1.5. *Пусть $G = (V, E)$ – сеть с истоком s и стоком t. Тогда для любой активной вершины v найдется простой путь из v в s в остаточной сети G_f.*

Доказательство. Пусть S – множество вершин, достижимых из v в остаточной сети G_f, а $\overline{S} = V \setminus S$ – все остальные вершины. Предположим, что $s \in \overline{S}$, то есть исток s не достижим из вершины v.

Поскольку вершина v активна, в ней есть избыток $e_f(v) > 0$. По свойству предпотока имеем: $e_f(w) \geqslant 0$, $\forall w \neq \{s, t\}$. Тогда для суммы избытков вершин множества S получаем неравенство $e_f(S) = \sum\limits_{w \in S} e_f(w) > 0$. Пусть (w, x) – дуга, для которой $w \in \overline{S}$ и $x \in S$. Если $f(w, x) > 0$, то $f(x, w) < 0 = c(x, w)$. Это означает, что ребро принадлежит остаточной сети: $(x, w) \in E_f$. Следовательно, $w \in S$. Получили противоречие, а значит $f(w, x) \leqslant 0$, $\forall w \in \overline{S}$, $x \in S$. Обозначим $f(W, X)$ суммарный поток из множества вершин W во множество вершин X.

В этих обозначениях $f(\overline{S}, S) \leqslant 0$. Тогда $e_f(S) = f(V, S) = f(S, S) + f(\overline{S}, S) = f(\overline{S}, S) \leqslant 0$. Но $e_f(S) > 0$. Полученное противоречие завершает доказательство леммы. □

Лемма 1.6. *При выполнении алгоритма проталкивания предпотока метка расстояния любой вершины $v \in V$ никогда не превзойдет $2n - 1$.*

Доказательство. Достаточно рассмотреть только активные вершины, так как метка расстояния $d(v)$ может увеличиваться только тогда, когда вершина v активна. Если v активна, то из нее достижим исток s в остаточной сети G_f по лемме 1.5. Следовательно, существует простой путь из v в исток s длины $l \leqslant n - 1$. Используя ту же технику, что и в доказательстве леммы 1.1, получаем: $d(v) \leqslant d(s) + l \leqslant d(s) + n - 1 = 2n - 1$. □

Лемма 1.7. *Во время выполнения алгоритма проталкивания предпотока метка расстояния любой вершины $v \in V$ никогда не убывает. После операции увеличения метки вершины v не существует допустимых дуг, входящих в v.*

Доказательство. Если мы покажем, что операция изменения метки расстояния вершины v приводит к увеличению $d(v)$, то поскольку эта операция является единственным способом изменения метки расстояния $d(v)$, из этого автоматически будет следовать утверждение о том, что метка расстояния $d(v)$ любой вершины $v \in V$ никогда не убывает. До операции изменения метки расстояния выполнено неравенство $d(v) \leqslant d(w)$ для каждой вершины w, достижимой из v в остаточной сети (иначе возможно проталкивание потока из v). После изменения метки расстояния найдется такая вершина w_0, что $d(v) = d(w_0)+1$, причем w_0 достижима из v в остаточной сети. Следовательно, метка $d(v)$ увеличилась хотя бы на 1. До ее увеличения выполнялось неравенство $d(u) \leqslant d(v)+1$ для всех остаточных ребер (u, v), а после увеличения $d(u) \leqslant d(v)$. Из этого следует, что операция изменения метки расстояния вершины v приводит к увеличению $d(v)$. □

Лемма 1.8. *Общее число операций изменения меток расстояний есть $O(n^2)$.*

Доказательство. Метка расстояния во время операции ее изменения увеличивается в силу леммы 1.7, но не может стать больше $2n-1$ по лемме 1.6. Итак, метку расстояния любой вершины $v \in V \setminus \{s, t\}$ можно увеличить самое большее $2n - 1$ раз. Всего таких вершин $n - 2$, поэтому для общего числа операций изменения меток расстояний верна оценка $O(n^2)$. □

Лемма 1.9. *При выполнении алгоритма проталкивания предпотока число насыщающих проталкиваний есть $O(nm)$.*

Доказательство. Рассмотрим насыщающее проталкивание вдоль ребра (v, w). После проталкивания ребро (v, w) исчезло из остаточной сети G_f. Для того, чтобы это ребро появилось вновь, необходимо протолкнуть поток из w в v, но этого нельзя сделать, пока не будет выполнено $d(w) = d(v) + 1$, то есть $d(w)$ необходимо увеличить по крайней мере на 2. В силу леммы 1.6 проталкивание вдоль дуги и соответствующее обратное проталкивание могут быть выполнены не более $n - 1$ раз. Поэтому и насыщающих проталкиваний вдоль каждой дуги не больше $n - 1$. Поскольку всего m дуг, для числа насыщающих проталкиваний справедлива оценка $O(nm)$. □

Лемма 1.10. *При выполнении алгоритма проталкивания предпотока число ненасыщающих проталкиваний есть* $O(n^2m)$.

Доказательство. Назовем потенциалом сумму меток расстояний активных вершин $\Phi = \sum_{v \in X} d(v)$, где X – множество активных вершин, и будем смотреть, как меняется потенциал в ходе выполнения алгоритма. Изначально $\Phi = 0$. Каждое ненасыщающее проталкивание из u в v уменьшает Φ по крайней мере на единицу, так как вершина u перестает быть активной и слагаемое $d(u)$ исчезает, а появиться может лишь слагаемое $d(v)$ (если вершина v не сток, не исток и не была активной), которое на единицу меньше: $d(v) = d(u) - 1$. Следовательно, число ненасыщающих проталкиваний ограничено значением потенциала Φ, который может увеличиваться в ходе насыщающих проталкиваний и операций изменения меток расстояний. Насыщающее проталкивание из v в w может увеличить Φ не более чем на $2n - 1$, так как никакая метка расстояния не изменяется, а только вершина w, метка расстояния которой не превосходит $2n - 1$, может стать активной. По лемме 1.9 число насыщающих проталкиваний $O(nm)$, поэтому увеличение Φ за счет насыщающих проталкиваний есть $O(n^2m)$. Каждая операция изменения метки вершины увеличивает Φ. Всего n вершин, причем метка расстояния каждой не превзойдет $2n - 1$. Следовательно, увеличение Φ за счет изменений меток расстояний ограничено числом $2n^2 - n$, или $O(n^2)$.

Таким образом, суммарное увеличение Φ за время работы алгоритма составляет $O(n^2m)$. Величина Φ остается неотрицательной, поэтому суммарное уменьшение Φ (и тем самым общее число ненасыщающих проталкиваний) не превосходит суммарного увеличения, то есть $O(n^2m)$. □

2 Задача о минимальном разрезе

2.1. Формулировка задачи

Задача о минимальном разрезе – задача разделения вершин взвешенного графа с n вершинами и m ребрами на два множества так, чтобы суммарный вес

множества ребер, концевые вершины которых лежат в разных множествах, был минимален.

Существует несколько вариантов постановки задачи: граф может быть ориентированным или неориентированным, взвешенным или невзвешенным. В случае неориентированного невзвешенного графа минимальный разрез будет являться наименьшим множеством ребер, при удалении которых граф распадается на две или более компонент связности.

2.2. Приложения задачи о минимальном разрезе

Ниже мы перечислим несколько практически важных задач, для конструктивного решения которых применяется техника поиска минимального разреза.

2.2.1. Получение информации

В области получения информации [4, 5] минимальные разрезы используются при решении задачи выделения совокупности документов, отвечающих теме запроса, в гипертекстовых системах. Если ссылки в наборе гипертекстовых документов трактовать как ребра графа, то минимальный и близкие к нему разрезы соответствуют группе документов, которые имеют мало ссылок между собой, и поэтому, скорее всего, не соответствуют нужной теме.

2.2.2. Компиляторы для параллельных языков программирования

Необходимость решения задачи о минимальном разрезе возникает и при разработке компиляторов для языков параллельного программирования [4, 6]. Параллельной программой будем считать такую программу, которую мы собираемся запускать на машине с распределенной памятью. В графе для этой программы вершины будут означать программные операции, а ребра – потоки данных между программными операциями. Когда программные операции распределены среди процессоров, ребра, соединяющие вершины из различных процессоров, являются плохими, поскольку они указывают на необходимость межпроцессорных связей. Поиск оптимального расположения программных операций требует повторного решения задачи о минимальном разрезе в графе.

2.2.3. Задачи комбинаторной оптимизации

Задача о минимальном разрезе также играет важную роль в крупных задачах комбинаторной оптимизации [4]. Сейчас лучшие методы поиска точного решения задачи о коммивояжере большой размерности основаны на технике отсекающих

плоскостей [7, 8, 9]. Множество правдоподобных туров бродячего торговца в графе приводит к выпуклому многограннику в многомерном векторном пространстве. Алгоритмы отсекающей плоскости находят оптимальный тур в результате повторной генерации линейных неравенств, которые отсекают нежелательные части многогранника до тех пор, пока не останется только оптимальный тур. Наиболее полезные неравенства – условия исключения фрагментов тура – были введены Данцигом, Фалкерсоном и Джонсоном в 1954 году [10]. Задача определения целостности тура может быть сформулирована как задача поиска минимального разреза в графе с вещественными весами на ребрах. Поэтому алгоритмы отсекающей плоскости для задачи о бродячем торговце должны решать огромное число задач о минимальном разрезе.

2.2.4. Теория устойчивости сети

Рассмотрим классическую задачу теории устойчивости сети [4, 11, 12]. Задана сеть из n вершин, каждый из m каналов связи которой может перестать действовать (выйти из строя) независимо от других с заданной вероятностью. Требуется определить вероятности некоторых событий в сети, относящихся к связности. Типичный вопрос многотерминальной сетевой устойчивости заключается в определении вероятности того, что сеть останется связной. Задача двухтерминальной устойчивости заключается в определении вероятности того, что две конкретные вершины останутся связанными. Эти задачи имеют очевидное применение в дизайне коммуникационных сетей. Поскольку задачи такого рода считаются трудно решаемыми точно, большой интерес представляет оценка вероятности сбоя. Такие оценочные задачи успешно решаются вероятностными алгоритмами, которые разрабатывались для задачи о минимальном разрезе.

2.2.5. Построение иерархического дерева детальности сегментации изображения

Векторизация изображения является важным этапом в анализе изображений. Векторизованное изображение используется в задаче совмещения и взаимной привязки нескольких изображений одной сцены или объекта. Основные приложения этой задачи – привязка аэрокосмических изображений к контурной карте, восстановление трехмерных объектов и сцен по серии снимков, создание круговой панорамы по набору изображений, полученных путем неравномерного поворота камеры вокруг какой-либо оси (с переменным увеличением), восстановление трехмерных моделей помещений после съемки их на камеру.

Для каждого из контуров векторного изображения можно построить вектор параметров: длина, извилистость, фрактальные характеристики, точность аппроксимации типовыми формами (дугой, прямой, полиномом). По построенным

векторам параметров можно выполнить совмещение контуров двух или более изображений.

При совмещении изображений требуется учитывать наиболее характерные и заметные черты изображения, поэтому в векторном изображении должны быть только самые заметные контуры, составляющие «набросок» сцены. Здесь и возникает задача снижения детальности исходной сегментации [13] и выделения из множества обнаруженных границ наиболее заметных.

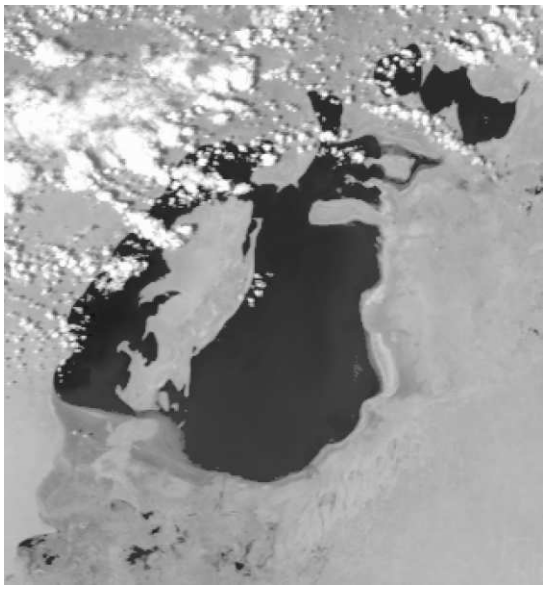
Часто используемый для векторизации алгоритм RSEG [14] имеет преимущество в том, что он выделяет даже малозаметные для человеческого глаза границы. Однако это приводит к тому, что количество выделяемых им сегментов очень велико (сегментация очень детальна).

Задача снижения детальности сегментации состоит в снижении числа сегментов путем их слияния. В результате на изображении должно остаться небольшое число крупных сегментов, разделенных хорошо заметными границами. Однако информация о более мелких сегментах также может понадобиться, поэтому целесообразно построить иерархическую сегментацию в виде пирамиды детальности, на самом верхнем уровне которой находятся крупные сегменты, а на более низких уровнях размер сегментов последовательно уменьшается.

Сначала по изображению формируется граф. Вершинами в нем выступают сегменты, а ребрами – границы между сегментами. Каждому ребру приписывается вес, который отражает степень «похожести» соседних сегментов и «заметность» границы. Чем сильнее отличаются сегменты и чем заметнее граница, тем меньше вес ребра.

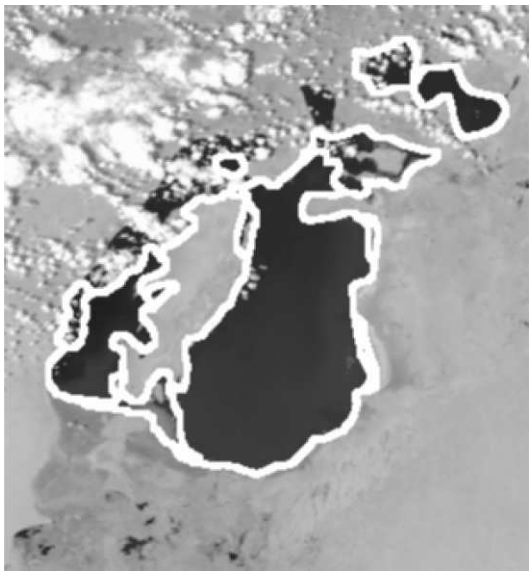
В этом графе выбираются сток и исток, после чего находится минимальный разрез. Этот разрез соответствует самой заметной границе, разделяющей выбранные в качестве стока и истока сегменты. После этого ребра, составляющие минимальный разрез, удаляются из графа, в результате чего образуются два подграфа. В каждом подграфе снова выбираются сток и исток, и находится минимальный разрез. Затем для полученных подграфов повторяется этот же алгоритм до тех пор, пока все вершины исходного графа не будут разъединены. В ходе работы алгоритма детальность сегментации постепенно увеличивается.

Для отражения иерархии детальности можно построить дерево с выделенным корнем, узлами которого будут все подграфы, образованные в процессе разрезания исходного графа. Исходный граф будет корневой вершиной. Из корня дерева будут идти два ребра к узлам, которые соответствуют подграфам, полученным на первой итерации. В свою очередь от этих узлов будут ребра к узлам следующего уровня детальности. Листья дерева будут соответствовать подграфам из одной вершины. Дерево детальности можно обойти, начиная с корня, так, что из возможных в каждый конкретный момент путей продолжения обхода будет выбираться ребро, которому соответствует разрез минимального веса.



(а)

(б)



(в)

Рис. 1. Алгоритм уменьшения детальности сегментации показал хорошие результаты для аэрокосмических изображений. Исходное изображение (а), результат сегментации (б) и результат уменьшения детальности (в) для аэрокосмического изображения Аральского моря, полученного путем совмещения трех спектральных каналов с прибора AVHRR спутника серии NOAA (из работы [13]). Видно, что выделились водоемы. На изображении остались в основном наиболее заметные границы – береговые линии.

2.2.6. Задача оптимального планирования открытых горных разработок

В ходе открытых горных разработок в земной коре формируется углубление, и горные породы выкапываются из земли для извлечения руды, содержащейся в этих пластах. Оптимальная форма этого углубления определяется еще до начала работ. Для определения такой формы, которая максимизирует чистую прибыль, область делится на части, после чего оцениваются запасы руды в каждом таком блоке (для этого используется геологическая информация). Задача оптимального планирования – максимизировать итоговую выгоду от ведения горных работ, причем должны выполняться условия наклонного бурения и ограничения, согласно которым глубоко залегающий блок может быть извлечен только после того, как будут выкопаны находящиеся над ним блоки.

Задача об открытых горных разработках [15] формально может быть сформулирована как задача теории графов, заданная на ориентированном графе $G = (V, E)$. Каждый блок соответствует вершине множества V с весом b_i, который означает величину прибыли от извлечения этого блока (значение b_i – это разница между оценкой стоимости руды и затратами на выкапывание этого блока). Вес b_i может быть как положительным, так и отрицательным. В графе G есть дуга (i, j), если блок i не может быть извлечен до блока j, т.е. блок j находится в пласте над блоком i. Решение задачи о том, какие блоки необходимо извлечь для максимизации прибыли, эквивалентно отысканию замкнутого подмножества вершин максимального веса в графе.

Можно показать, что задача поиска замкнутого подмножества вершин $V^* \subseteq V$, для которого сумма $\sum\limits_{i \in V^*} b_i$ максимальна, сводится к задаче о минимальном разрезе. Это легко продемонстрировать, сформулировав задачу в терминах целочисленного линейного программирования. Пусть x_i – двоичная переменная, принимающая значение 1, если вершина i принадлежит множеству V^*, и 0 – иначе. Тогда требуется

$$\text{максимизировать } \sum_{j \in V} b_j \cdot x_j$$

$$\text{при условии } x_j - x_i \geqslant 0, \quad \forall (i, j) \in E;$$

$$0 \leqslant x_j \leqslant 1, \ x_j \text{ – целое число}, \quad \forall j \in V.$$

Построим граф $G' = (V', E')$, добавив исток s и сток t, то есть $V' = V \cup \{s, t\}$. Обозначим $V^+ = \{j \in V : b_j > 0\}$ множество вершин с положительным значением веса, а $V^- = \{j \in V : b_j < 0\}$ – с отрицательным значением веса. Множество E' графа G' состоит из всех дуг множества E, дуг из истока s до каждой вершины с положительным весом $\{(s, v) : v \in V^+\}$ и дуг из вершин с отрицательным весом до

стока $\{(v,t)\colon v \in V^-\}$. Пропускная способность каждой исходной дуги множества E устанавливается равной ∞. Для каждой дуги вида (s,i) установим пропускную способность b_i, а для каждой дуги вида (j,t) – равную $-b_j$.

Лемма 2.1. *Если в построенном графе $G' = (V', E')$ найти минимальный s-t разрез (S, T), то множество S будет являться замкнутым подмножеством $V^* \subseteq V$ максимального веса.*

Доказательство. Сначала множество S, содержащее исток s, замкнуто, поскольку разрез должен иметь конечную величину, и поэтому не может содержать дуг множества E.

Пусть далее $(S, \overline{S})$ является конечным разрезом в графе G'. Пропускная способность этого разреза

$$\sum_{j \in V^- \cap S} |b_j| + \sum_{j \in V^+ \cap \overline{S}} b_j = \sum_{j \in V^- \cap S} |b_j| + \sum_{j \in V^+} b_j - \sum_{j \in V^+ \cap S} b_j = B - \sum_{j \in S} b_j,$$

где B – это сумма всех положительных весов. Поэтому минимизация пропускной способности разреза эквивалентна максимизации итоговой суммы весов вершин множества S разреза. □

2.3. Различные подходы к решению задачи

Классический алгоритм Гомору–Ху [16], предложенный в 1961 году, решает задачу о минимальном разрезе с использованием $n-1$ вычислений минимального s-t разреза. Наиболее быстрые из известных алгоритмов для задачи о минимальном s-t разрезе используют технику потоков, в частности алгоритм проталкивания предпотока, и имеют время работы $\omega(nm)$.

Для уменьшения числа вычислений максимального потока, требуемых алгоритмом Гомору–Ху, Падберг и Ринальди в 1990 году разработали комплекс эвристик [17], позволяющих стягивать определенные ребра в процессе вычислений. PR-эвристики часто приводят к большому увеличению производительности.

Для задачи о минимальном разрезе Хао и Орлин в 1992 предложили алгоритм [18], основанный на методике проталкивания предпотока Гольдберга–Тарьяна, и показали, как выполнить вычисление $n-1$ минимального s-t разреза за время, асимптотически эквивалентное затратам на вычисление одного минимального s-t разреза. Этот алгоритм работает за время $O(nm \log(n^2/m))$.

Некоторые новые алгоритмы, предложенные позднее, теоретически более эффективны – оценки времени работы для них сравнимы или превосходят лучшие оценки времени работы для задачи о минимальном s-t разрезе.

Один из подходов к решению задачи без привлечения потоков разработал Габов в 1991 году [19]. С помощью данного подхода, основанного на технике матроидов, можно вычислить минимальный разрез за время $O(m + c^2 n \log(n^2/m))$, где c – величина минимального разреза.

Другой новый подход заключается в повторном выявлении и стягивании ребер, которые не входят в минимальный разрез, до тех пор, пока не останутся две крупные метавершины и минимальный разрез будет виден. Эта техника применима только к неориентированным графам, но допускает и взвешенные графы.

Рандомизированный алгоритм Каргера–Штейна [20, 21] выполняется за время $O(n^2 \log^3 n)$ и находит минимальный разрез с большой вероятностью.

Нагамочи и Ибараки [22] в 1992 году предложили процедуру, которая определяет и стягивает ребро, гарантировано не входящее в минимальный разрез, за время $O(m + n \log n)$. Построенный на основе этой процедуры алгоритм вычисляет минимальный разрез за время $O(n(m + n \log n))$.

2.4. Приступаем к решению задачи

Самый простой подход к вычислению минимального разреза состоит в следующем: необходимо вычислить s-t разрез для всех пар вершин (s, t) и выбрать разрез минимальной величины. Пусть $M(n, m)$ обозначает наилучшее время решения задачи о максимальном потоке в графе с n вершинами и m дугами. Тогда для «наивного» алгоритма справедлива оценка $O(C_n^2 M(n, m)) = O(n^2 M(n, m))$. Однако можно довольно легко сделать лучше:

Теорема 2.2 (Гомору–Ху). *Минимальный разрез может быть найден с использованием $n - 1$ вычислений минимального s-t разреза в случае неориентированного графа, или $2(n - 1)$ вычислений – в случае ориентированного графа.*

Доказательство. Рассмотрим случай неориентированного графа. Зафиксируем вершину s. Мы будем вычислять минимальный s-t разрез для всех вершин, отличных от s. Покажем, что один из этих разрезов будет минимальным. Пусть $(S, \overline{S})$ – минимальный разрез, где $s \in S$. Поскольку множество $\overline{S}$ непусто, оно содержит вершину t. Теперь $(S, \overline{S})$ можно рассматривать как s-t разрез. Ясно, что величина минимального s-t разреза между вершинами s и t не превзойдет величины разреза $(S, \overline{S})$. Конечно же, величина минимального s-t разреза не может быть меньше величины разреза $(S, \overline{S})$. Итак, мы нашли минимальный разрез.

Существует также разновидность алгоритма, где мы «склеиваем» вершины s и t в одну большую вершину (при этом кратные дуги не удаляются) после вычисления минимального s-t разреза между этими вершинами. Мы берем полученную новую вершину в качестве истока, выбираем новый сток, и повторяем эту процедуру до тех пор, пока не закончатся вершины (производится $n - 1$ вычислений s-t разреза). Отметим два факта:

- если две вершины лежат на одной стороне разреза $(S, \overline{S})$ (то есть обе вершины, например, одновременно принадлежат множеству S), то «склеивание» (объединение) их вместе не приведет к изменению величины разреза (мы не изменили веса ребер, входящих в разрез);

- «склеивание» двух вершин никогда не может привести к уменьшению числа ребер, входящих в минимальный разрез, поскольку любой разрез в новом графе также будет разрезом в исходном графе.

Покажем, что описанный алгоритм найдет минимальный разрез. Пусть $(S, \overline{S})$ – минимальный разрез, а t – первый выбранный сток, причем вершины s и t лежат на разных сторонах разреза. Все предыдущие операции «склеивания» не изменили величину разреза $(S, \overline{S})$, поэтому найденный s-t разрез будет минимальным.

Теперь перейдем к случаю ориентированного графа. Мы снова фиксируем вершину s, и замечаем, что минимальный разрез будет иметь вид $(S, \overline{S})$ или $(\overline{S}, S)$, где $s \in S$. С помощью алгоритма, описанного выше, найдем минимальный разрез с вершиной $s \in S$ (это первая фаза). Для поиска минимального разреза в случае, когда $s \in \overline{S}$, мы просто изменим направление всех дуг G на противоположное и запустим алгоритм снова. Минимальным разрезом в графе G будет разрез наименьшей величины из этих двух. Заметим, что одна и та же начальная вершина s используется в обеих фазах. □

2.5. Алгоритм Хао–Орлина

В этом разделе мы опишем алгоритм, с помощью которого задача о минимальном разрезе может быть решена с использованием примерно тех же затрат, что необходимы для вычисления максимального потока в сети с использованием метода проталкивания предпотока.

Ключевая идея алгоритма Хао и Орлина: аккуратно выбирать следующий сток, чтобы можно было использовать информацию от предыдущих вычислений максимального потока. По-прежнему, фиксируем исходную вершину s, выполняем описанный выше алгоритм в две фазы для ориентированного случая. В ходе первой фазы будет найден минимальный разрез, в котором $s \in S$, затем мы изменим на противоположное направление всех дуг в графе и запустим вторую фазу для отыскания минимального разреза, но уже при условии, что $s \in \overline{S}$. Важно отметить, что вершина s остается фиксированной для обеих фаз.

В ходе каждой фазы выбирается последовательность стоков $t_1, t_2, \ldots, t_{n-1}$. Отметим, что эта последовательность будет различна для двух фаз (в общем случае). Когда вершина t_i выбрана в качестве стока, мы начинаем проталкивать поток к вершине t_i. Этот процесс завершится, когда будет найден максимальный предпоток, то есть в сток не удастся отправить больше потока. Мы можем тут же

указать минимальный s-t_i разрез $(S, \overline{S})$. Для этого образуем множество $\overline{S}$ вершин, из которых есть путь до стока t_i в остаточной сети. Наконец мы объединим s и t_i вместе и выберем новый сток.

Приведем изменения в стандартном алгоритме проталкивания предпотока, которые необходимо сделать.

- При инициализации выполняется присваивание $d(s) = n$. Как будет показано ниже, это гарантирует, что никогда не будет пути из вершины s в сток t_i в остаточной сети.

- Когда необходимо будет «склеить» (объединить) вершины t_i и s, мы просто присвоим $d(t_i) = n$ и произведем насыщение всех дуг, исходящих из t_i, для сохранения корректности меток расстояний. Это эффективным образом делает вершину t_i также вершиной-истоком. Если в реализации алгоритма проталкивания предпотока используются динамические деревья, то для всех дуг динамического дерева в t_i нужно выполнить операцию отсечения до того, как она станет еще одной вершиной-истоком (это сохраняет свойство допустимости всех дуг динамического дерева).

- Далее, мы укажем способ выбора следующего стока t_i. Однако для корректности работы алгоритма проталкивания предпотока в течение времени, пока t_i является стоком, нужно обеспечить, чтобы разница $d(s) - d(t_i)$ была достаточно велика. Это ограничение необходимо, чтобы не было пути из вершины s в t_i в остаточной сети G_f.

  Обозначим $\overline{d(t_i)}$ метку расстояния вершины t_i в тот момент, когда она выбрана в качестве нового стока. Для последовательности вершин-стоков будут выполнены следующие условия:

$$\overline{d(t_1)} = 0 \tag{1}$$

$$\overline{d(t_i)} \leqslant 1 + \max_{j<i}\{\overline{d(t_j)}\}, \quad \forall i \geqslant 2 \tag{2}$$

  Исходя из этого по индукции легко показать, что $\overline{d(t_i)} < i$, $\forall i \geqslant 1$, и потому $d(s) - d(t_i) > n - i$, пока t_i является стоком.

  Теперь предположим, что существует простой путь P из s в t_i в остаточной сети G_f. Отметим, что граф G содержит только $n-i+1$ вершин (поскольку предыдущие $i-1$ стоков были объединены вместе с s), поэтому и путь P состоит не более чем из $n-i$ дуг. По определению меток расстояний для каждой дуги (v, w) пути P выполнено неравенство $d(v) \leqslant d(w) + 1$, поэтому $d(s) \leqslant d(t_i)+(n-i)$. Получили противоречие. Это соображение подсказывает способ того, как выбирать сток t_i так, чтобы выполнялось (2).

- Ниже мы также незначительно изменим определение активной вершины и допустимой дуги.

2.5.1. Выбор следующего стока

Для решения данной задачи будем использовать правило изменения меток при создании пропусков. Определим массив уровней, который содержит ячейки $S[i]$ для всех возможных меток расстояний ($0 \leqslant i \leqslant n$). В ячейке i хранится число вершин с меткой расстояния i. Ясно, что поддержание этой структуры не приведет к увеличению времени работы алгоритма.

Рассмотрим операцию изменения метки какой-нибудь вершины v, где $d(v) = i$. Предположим, что после этого будет $S[i] = 0$ (образуется пробел).

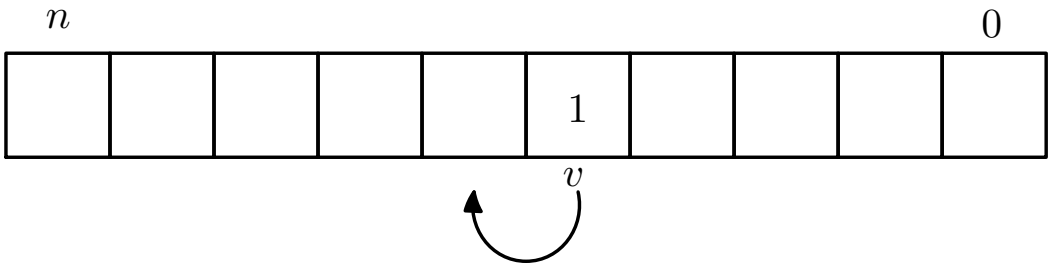


Рис. 2. Создание пробела в массиве уровней.

Тогда для любой вершины w с меткой $d(w) > i$ (включая и v) не может быть пути из w до стока в остаточной сети G_f, поскольку каждая дуга пути уменьшает метку расстояния по крайней мере на 1. Следовательно, все такие вершины w могут быть проигнорированы в ходе вычисления максимального потока.

Алгоритм будет использовать это свойство для разделения вершин на две группы: множество D «дремлющих» («замороженных») вершин и множество $W = V \setminus D$ «бодрствующих» вершин [18, 23]. Впоследствии мы распределим «дремлющие» вершины по классам $D_0, D_1, \ldots, D_k$, где $D_0 = S$ (множество вершин-истоков). Множества D_i будем хранить в стеке.

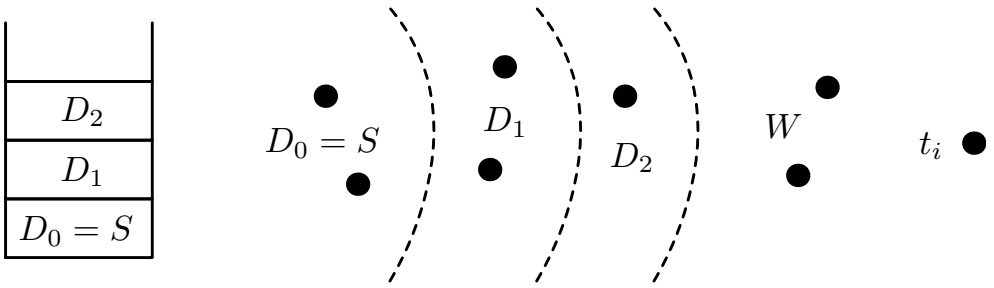


Рис. 3. Стек «дремлющих» вершин.

Когда операция изменения метки для вершины v будет приводить к пробелу, вместо обновления метки для v мы «заморозим» множество активных вершин $w \in W$ с метками расстояний $d(w) \geqslant d(v)$. Будет создано новое множество «дремлющих» вершин, которое мы добавим в стек (увеличим k и сформируем множество D_k). Эти вершины будут удалены из массива уровней и будут игнорироваться до тех пор, пока не придет время «разморозить» их. В частности, мы изменим определение активных вершин и допустимых дуг для работы только с множеством W «бодрствующих» вершин.

- Вершину v назовем активной, если $v \in W \setminus \{t\}$ и $e_f(v) > 0$.
- Дугу (v, w) будем называть допустимой, если $v, w \in W$, $c_f(v, w) > 0$ и $d(v) = = d(w) + 1$.

Теперь мы можем объяснить способ выбора следующего стока. Изначально мы работаем с вершинами множества W до тех пор, пока они не закончатся, затем мы переключимся на множество «дремлющих» вершин, которое находится на вершине стека. Итак, выбор следующего стока проходит в три этапа.

1. Переместить t_i из W в S так, как описано выше.
2. Если $W = \emptyset$, то положить $W = D_k$ и уменьшить k (извлечь множество «дремлющих» вершин из стека).
3. Назначить в качестве следующего стока t_{i+1} элемент множества W «бодрствующих» вершин с минимальной меткой расстояния.

2.5.2. Алгоритм в псевдокоде

Приведем алгоритм поиска минимального разреза в псевдокоде.

```
procedure INITIALIZE
begin
    for each (для каждой) вершины s ∈ S do
        послать c(s, j) единиц потока вдоль дуги {(s, j): s ∈ S, j ∉ S}
    D_0 ← S
    d*_max ← 0    // d*_max = max{i: D_i ≠ ∅}
    W ← V \ S
    d(t') ← 0
    for each (для каждой) вершины j ∈ V \ {t'} do
        d(j) ← 1
end
```

procedure PUSH/RELABEL(i)
begin
  if сеть содержит допустимую дугу (i, j)
  then протолкнуть $\delta \leftarrow \min\{e_f(i), c_f(i, j)\}$ единиц потока из i в j
  else
    if $d(j) \neq d(i)$ для любой вершины $j \in W \setminus \{i\}$ **then**
      $d^\star_{\max} \leftarrow d^\star_{\max} + 1$
      $R \leftarrow \{j \in W : d(j) \geqslant d(i)\}$
      $D_{d^\star_{\max}} \leftarrow R$
      $W \leftarrow W \setminus R$
    else if в G_f нет дуги $\{(i, j) : j \in W\}$ **then**
      $d^\star_{\max} \leftarrow d^\star_{\max} + 1$
      $D_{d^\star_{\max}} \leftarrow \{i\}$
      $W \leftarrow W \setminus \{i\}$
    else $d(i) \leftarrow \min\{d(j) + 1 : (i, j) \in E_f,\ j \in W,\ c_f(i, j) > 0\}$
end

procedure SELECTNEWSINK
begin
  // вершина t' обозначает старый сток
  $W \leftarrow W \setminus \{t'\}$
  $S \leftarrow S \cup \{t'\}$
  $D_0 \leftarrow D_0 \cup \{t'\}$
  if $S = N$ **then quit else continue**
  for each (для каждой) дуги $\{(t', k) : k \in V \setminus S\}$ **do**
    послать $c_f(t', k)$ единиц потока вдоль дуги (t', k)
  if $W = \emptyset$ **then**
    $W \leftarrow D_{d^\star_{max}}$
    $d^\star_{max} \leftarrow d^\star_{max} - 1$
  $t' \leftarrow \{j \in W : d(j) = \min_k d(k)\}$
end

```
procedure FindMinCut
begin
    S ← {1}
    t' ← {2}
    Initialize
    while (пока) S ≠ N do
        while (пока) сеть содержит активную вершину i do
            выбрать активную вершину i
            Push/Relabel(i)
        if BestValue > c(D, W) then
            Cut ← (D, W)
            BestValue ← c(D, W)
        SelectNewSink
    return Cut
end
```

При инициализации алгоритм делает насыщенными все дуги, исходящие из вершин множества S, поэтому в остаточной сети не будет дуги из вершины множества S в вершину из $V \setminus S$. Множеству D_0 «дремлющих» вершин в качестве начального значения присваивается S, а множеству «бодрствующих» вершин W – $V \setminus S$.

Алгоритм выполняет операцию проталкивания сходным с оригинальным алгоритмом проталкивания предпотока Гольдберга–Тарьяна образом, однако предлагаемый алгоритм налагает ограничение на выбор вершины для выполнения проталкивания, которое заключается в требовании, чтобы выбранная вершина принадлежала множеству W «бодрствующих» вершин.

Будем говорить, что вершина из множества D «дремлющих» вершин удовлетворяет свойству дремоты, если для любой вершины $i \in D$ не существует дуги $(i, j) \in G_f$, где $j \in W$. Будем говорить также, что вершины множества $D = \bigcup_k D_k$ удовлетворяют расширенному свойству дремоты, если в дополняющей сети не существует дуги из вершины множества D_i в вершину множества D_j, где $i < j$.

Алгоритм выполняет операцию изменения меток расстояний вершин так же, как и оригинальный алгоритм проталкивания предпотока, за исключением двух случаев. В первом случае алгоритм определяет активную вершину i, метку расстояния которой нужно изменить, с требованием, чтобы метка расстояния никакой другой вершины не равнялась $d(i)$. Далее алгоритм полагает $R = \{j \in \in W\colon d(j) \geqslant d(i)\}$, увеличивает $d^\star_{\max}$, и пересылает вершины множества R из W в $D_{d^\star_{\max}}$. Поскольку не существует дуги из вершины множества R в вершину множества $W \setminus R$, модифицированные множества «дремлющих» вершин будут продолжать удовлетворять свойству дремоты и расширенному свойству дремо-

ты в предположении, что D удовлетворяло этим двум свойствам до перемещения множества R из W в D.

Рассмотрим еще одну ситуацию, которая может возникнуть при изменении меток расстояний. Предположим, что у вершины i нужно изменить метку расстояния, а в остаточной сети не существует дуги из i в вершину множества W. В этом случае можно добавить вершину i в D и удалить ее из W. Легко видеть, что новое множество $D \cup \{i\}$ также удовлетворяет свойству дремоты.

Поскольку, как будет показано в следующем разделе (смотри лемму 2.3), не существует дуги из вершины множества D в вершину из W, вершины из D не должны рассматриваться на текущем шаге процедуры поиска минимального S-t' разреза, однако эти вершины могут рассматриваться на более поздней стадии процедуры FindMinCut, так как алгоритм решает последовательность задач о минимальном S-t' разрезе, где каждая вершина, кроме s, однажды будет стоком.

После определения минимального S-t' разреза алгоритм пересылает t' в множество S и выбирает новый сток. Он выбирает вершину $j \in W$ с минимальной меткой расстояния. Процесс продолжается до тех пор, пока есть такая вершина $j \in W$, для которой верно $d(j) = d(t') + 1$. Однако, возможно, что t' была единственной вершиной в W в конце процедуры FindMinCut поиска минимального S-t' разреза, и что благодаря последующему перемещению t' в S, не останется вершин в W. Если W пусто, алгоритм пересылает множество $D_{d^\star_{max}}$ в W, что приводит к тому, что W перестает быть пустым. Пока W не является пустым, алгоритм выбирает вершину из W с минимальной меткой расстояния и начинает новую процедуру поиска минимального разреза.

Теперь мы видим, почему вершины множества D мы назвали «дремлющими». Каждая вершина из D (кроме вершин из S) будет в конце концов «пробуждена» и переслана обратно в W. В действительности, каждая из этих вершин будет стоком для некоторой последующей подзадачи.

2.5.3. Анализ алгоритма Хао–Орлина

Корректность работы алгоритма и временные оценки основаны на нескольких свойствах, которым удовлетворяет алгоритм. Ниже мы приведем наиболее важные свойства.

Лемма 2.3. *В остаточной сети G_f никогда не будет дуги из вершины множества D в вершину из W, а также из вершины множества D_i в вершину из D_j, где $i < j$.*

Доказательство. Когда множество вершин D_i «заморожено», в остаточной сети нет дуг из D_i в W (существует пробел между их метками расстояний, и для любой дуги в остаточной сети метки расстояний d уменьшаются не больше чем на 1).

Мы модифицировали определение допустимых дуг так, что выполнять проталкивание потока можно только среди вершин множества W. Следовательно, это свойство сохраняется, пока множество D_i является множеством «дремлющих» вершин. □

Это означает, что при поиске минимального s-t_i разреза нужно рассматривать только вершины из W, поскольку только из этих вершин может существовать путь до стока t_i в остаточной сети. Отметим, что в реализации с динамическими деревьями не будет дуг, исходящих из вершин множества D, так как дуги динамического дерева всегда являются допустимыми.

Лемма 2.4. *Для каждого стока t_i выполнено* $\overline{d(t_i)} \leqslant 1 + \max\limits_{j<i}\{\overline{d(t_j)}\}$.

Доказательство. Случай 1. $W \neq \emptyset$ (после удаления t_{i-1}). Заметим, что метка расстояния стока никогда не меняется. Пусть t_i – вершина W с наименьшей меткой расстояния. Предположим, что $\overline{d(t_i)} > \overline{d(t_{i-1})} + 1$. Поскольку массив уровней содержит только вершины из W, должен быть пробел. Всякий раз, когда при смене меток будет создан пробел в W, для предотвращения этого мы удаляет множество вершин из W и «замораживаем» их. Поэтому такого пробела не существует и $\overline{d(t_i)} \leqslant \overline{d(t_{i-1})} + 1$.

Случай 2. $W = \emptyset$. В этом случае присвоим $W = D_k$, «разморозим» множество «дремлющих» вершин. Пусть $v \in W$ – вершина с наименьшей меткой расстояния, то есть вершина, для которой выполнение операции изменения метки расстояния привело к «заморозке» множества D_k. В это время множество W должно содержать хотя бы одну вершину w с меткой $d(w) = d(v) - 1$, поскольку иначе был бы пробел. Метки расстояний никогда не уменьшаются, поэтому когда w становится стоком, $\overline{d(w)} \geqslant d(v) - 1$ также будет выполнено. Поскольку каждая вершина множества W будет выбрана в качестве стока до того, как v будет «размножена», будет выполнено $\max\limits_{j<i} \overline{d(t_j)} \geqslant d(v) - 1$. Присваивание $t_i = v$ дает желаемую оценку. □

2.5.4. Время работы

Время работы алгоритма складывается из затрат на:

- инициализацию;
- выбор активных вершин;
- выбор допустимых дуг;
- насыщающие проталкивания;

- ненасыщающие проталкивания;
- увеличение меток расстояний в ходе операции изменения меток;
- создание множеств «дремлющих» вершин и пересылку вершин из W в D;
- выбор новых стоков.

Время, затраченное на инициализацию, составляет $O(m)$. Для выбора активной вершины необходимо $O(n^2 + \text{число проталкиваний})$ шагов, а для выбора допустимых дуг – $O(nm + \text{число проталкиваний})$. Согласно приведенной в лемме 1.9 оценке, затраты на выполнение насыщающих проталкиваний составляют $O(nm)$. Метка расстояния любой вершины v не превосходит $O(n)$, поэтому будет $O(n^2)$ операций изменения меток, что справедливо и для оригинального алгоритма проталкивания предпотока Гольдберга–Тарьяна в силу леммы 1.8. Отметим, что операции «замораживания» и «размораживания» требуют $O(n^2)$ шагов, поскольку мы «замораживаем» вершину не более одного раза для конкретного стока. Затраты на выбор нового стока в сумме составляют $O(n^2)$ ($O(1)$ на однократную операцию плюс время на пересылку вершин из D в W), так как не более n множеств вершин будут перемещены из D в W, а каждая пересылка требует $O(n)$ шагов. Теперь можно подвести итог. На основе указанных соображений получаем алгоритм со временем работы $O(n^3)$ для задачи о минимальном разрезе, но с использованием динамических деревьев оценка времени работы может быть улучшена до $O(nm \log(n^2/m))$.

В дополнение к хорошим гарантиям производительности отметим, что алгоритм Хао–Орлина очень хорош на практике. В частности, он не требует выполнения операций стягивания ребра. Хотя концепция стягивания ребер проста и понятна, она трудно реализуема на практике.

2.6. Рандомизированный алгоритм Каргера–Штейна

В этом разделе мы рассмотрим вероятностный алгоритм для неориентированных мультиграфов (графов с кратными ребрами). В этом случае идея алгоритма становится интуитивно понятной и ее легко анализировать. Можно также показать, что такое ограничение не уменьшает общности результата.

Пусть дан мультиграф $G = (V, E)$ с n вершинами и m ребрами. Вероятностный алгоритм основан на фундаментальной операции стягивания ребра между двумя вершинами. После стягивания ребра (u, v) получим новый граф без вершины v, в котором каждое ребро вида (v, x) заменено ребром вида (u, x) (петли также удаляются).

Возникает желание применить следующий алгоритм:

повторить $n-2$ раза
  выбрать случайно ребро e
  стянуть ребро e

Когда алгоритм завершает работу, каждая исходная вершина уже объединена с одной из двух оставшихся крупных метавершин. Легко указать минимальный разрез в исходном графе: каждая часть его соответствует вершинам, содержащимся в одной из метавершин.

Теорема 2.5. *Вероятностный алгоритм вычисляет минимальный разрез с вероятностью* $P \geqslant \frac{2}{n(n-1)}$.

Доказательство. Пусть минимальный разрез C состоит из k ребер. Тогда все вершины имеют степень не меньше k. Действительно, если бы какая-нибудь вершина v имела степень меньше k, то разрез $(v, V \setminus v)$ состоял бы менее чем из k ребер, что противоречит предположению о минимальности разреза C. Поскольку каждое ребро, соединяющее две вершины, учитывается при подсчете степеней обоих вершин, а суммарная степень вершин не меньше nk, то граф должен содержать хотя бы $nk/2$ ребер.

Будем трактовать событие E_i следующим образом: «стянутое на итерации i ребро не входит в минимальный разрез C». Тогда вероятность получить минимальный разрез на финальной стадии алгоритма – это вероятность того, что на протяжении всех $n-2$ итераций (операций стягивания ребра) не было стянуто ни одного ребра, входящего в минимальный разрез C. Итак, требуется оценить вероятность «успеха» (вычисления минимального разреза) $P(\bigcap\limits_{i=1}^{n-2} E_i)$.

Заметим, что $P(\bigcap\limits_{i=1}^{n-2} E_i) = P(E_1)P(E_2|E_1)P(E_3|E_2 \cap E_1)\dots P(E_{n-2}|\bigcap\limits_{i=1}^{n-3} E_i)$. Теперь осталось оценить вероятности всех сомножителей и подставить значения этих вероятностей в данную формулу.

Посмотрим, что происходит на первой итерации. По крайней мере k из $nk/2$ ребер входят в минимальный разрез C, поэтому вероятность ошибки на первой итерации (выбора и стягивания ребра из C) не превосходит $\frac{k}{nk/2}$. Следовательно, для вероятности «успеха» на первой итерации справедлива оценка $P(E_1) \geqslant 1 - \frac{k}{nk/2} = 1 - \frac{2}{n}$.

В результате стягивания ребра на первой итерации число вершин уменьшилось на одну. Тем не менее, осталось хотя бы $(n-1)k/2$ ребер. Для условной вероятности стягивания ребра, не входящего в минимальный разрез C, на второй итерации алгоритма получаем выражение: $P(E_2|E_1) \geqslant 1 - \frac{k}{(n-1)k/2} = 1 - \frac{2}{n-1}$.

Для всех последующих итераций получаем аналогичные выражения для вероятности «успеха»: $P(E_i|\bigcap_{j=1}^{i-1} E_j) \geqslant 1 - \frac{2}{n-i+1}$.

В результате, когда алгоритм завершит работу, вероятность нахождения минимального разреза составит $P(\bigcap_{i=1}^{n-2} E_i) \geqslant \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \prod_{i=1}^{n-2} \left(\frac{n-i-1}{n-i+1}\right) = \frac{2}{n(n-1)}$. □

Внимательный читатель, возможно, уже заметил, что алгоритм, дающий правильный ответ с вероятностью $O(\frac{1}{n^2})$ не внушает оптимизма... На самом деле это обстоятельство лишь кажется непреодолимым. Повторение увеличивает вероятность успеха. Действительно, повторим алгоритм случайного стягивания ребер $7n^2$ раз. Тогда для вероятности сделать ошибку после $7n^2$ испытаний справедлива оценка:

$$\begin{aligned} P(\text{общая ошибка}) &= P(\text{ошибиться } 7n^2 \text{ раз}) \\ &= [P(\text{ошибиться однажды})]^{7n^2} \\ &= \left(1 - \frac{2}{n^2}\right)^{7n^2} \leqslant 10^{-6} \end{aligned}$$

Еще раз вернемся к оценке для вероятности дать правильный ответ в результате одного испытания. Из формулы для $P(\bigcap_{i=1}^{n-2} E_i)$ следует, что вероятность ошибки велика на последних итерациях алгоритма. Поэтому для улучшения производительности вероятностного алгоритма изменяют его поведение на последних итерациях так, чтобы модифицированный вариант был надежнее.

2.7. Алгоритм Нагамочи–Ибараки

Сейчас мы дадим обзор оригинального алгоритма поиска минимального разреза [22]. Общая идея проста: нужно $n - 2$ раза стянуть ребро, которое не будет входить в искомый минимальный разрез (такие ребра мы будем называть безопасными). Наибольшая трудность при таком подходе заключается в выборе безопасного ребра. Решение, предложенное Нагамочи и Ибараки

- достаточно эффективно;
- просто в реализации;
- позволяет стягивать сразу несколько ребер на одном шаге (это приводит к существенному снижению среднего времени работы);

- удобно для применения целого семейства эвристик (как вы уже догадались, речь идет о PR-тестах).

Отметим, что версия алгоритма Нагамочи–Ибараки с применением эвристик Падберга–Ринальди часто встречается под названием гибридного алгоритма Нагамочи.

2.7.1. Общая схема алгоритма

Мы уже обсудили основную идею алгоритма: на каждом шаге выявляются и стягиваются безопасные ребра. Напомним, что величину минимального разреза в графе G мы обозначили λ, а величину минимального s-t разреза – $\lambda_{s,t}(G)$.

Приведем схему алгоритма в псевдокоде:

```
algorithm NAGAMOCHI–IBARAKI
begin
    λ ← min{c(w, V \ w): w ∈ V}
    while (пока) |V| ⩾ 2 do
        (G, λ) ← CONTRACTSAFE(G, λ)
    return λ
end
```

На каждом шаге обновляется оценка сверху λ на величину минимального разреза. Ее значение задается минимумом на множестве величин всех тривиальных разрезов графа. Далее, пока в графе имеется достаточное количество вершин (не менее двух), мы выбираем две, являющиеся концами безопасного ребра, и стягиваем их. Последнее действие осуществляется процедурой ContractSafe, получающей на вход граф и текущую оценку на величину минимального разреза и выдающей обновленную оценку и граф, в котором стянуто хотя бы одно безопасное ребро.

2.7.2. Обновление оценки на величину минимального разреза

Процедуру ContractSafe мы реализуем при помощи следующего обхода графа. Изначально все ребра не просмотрены, а все вершины не посещены. Алгоритм поддерживает свойство $r(v)$ для каждой вершины и $q(e)$ для каждого ребра, где

- $r(v)$ – сумма пропускных способностей ребер между v и уже посещенными вершинами;
- $q(e)$, где $e = (v, w)$, равно $r(w)$, когда ребро e просмотрено в направлении от v к w.

При обходе всегда выбирается непосещенная вершина с наибольшим $r(v)$, и просматриваются все исходящие из этой вершины ребра. Пусть x и y – следующая за последней и последняя просмотренная вершины соответственно. Как можно заметить, тривиальный разрез $(y, V \setminus y)$ – это минимальный x-y разрез, так что мы можем обновить оценку $\lambda = \min\{\lambda, c(y, V \setminus y)\}$.

В целях оптимизации применяются эвристики. Мы рассмотрим две из них: первая поможет стягивать дополнительные ребра, в то время как вторая позволяет дополнительно уменьшать оценку на величину λ.

Суть первой эвристики – стягивать каждое ребро $e = (v, w)$, для которого $q(e) \geqslant \lambda$, так как $q(e)$ – это на самом деле нижняя оценка на $\lambda_{v,w}(G)$. Это наблюдение сильно ускоряет алгоритм, поскольку каждое стянутое ребро уменьшает количество фаз на 1. Фрагмент кода для этой эвристики представляет собой цикл, начало которого отмечено символом $(\star)$ в приводимом ниже тексте процедуры ContractSafe в псевдокоде.

Вторая эвристика базируется на наблюдении: если мы просматриваем вершину v, то мы уже просмотрели связное множество вершин V'. Мы можем наблюдать за V' и обновлять λ, если $c(V', V \setminus V') < \lambda$. Код, реализующий эту эвристику (часто называемую α-эвристикой), отмечен символом $(\star\star)$.

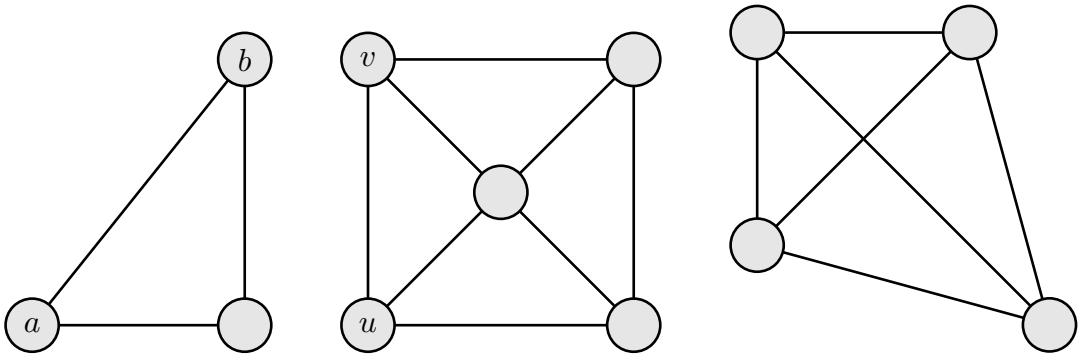
```
procedure ContractSafe(G, λ)
begin
    for each (для каждой) вершины v ∈ V do
        r(v) ← 0
        visited(v) ← false
    for each (для каждого) ребра e ∈ E do
        scanned(e) ← false
    while (пока) есть непосещенная вершина do
        v ← непосещенная вершина с наибольшим r(v)
        α ← α + c(v, V \ v) − 2r(v)     // (⋆⋆)
        λ ← min{α, λ}
        for each (для каждого) непросмотренного ребра e = (v, w) do
            r(w) ← r(w) + c(v, w)
            q(e) ← r(y)
            scanned(e) ← true
        visited(v) ← true
    for each (для каждого) ребра {e = (v, w): q(e) ⩾ λ} do    // (⋆)
        G ← G \ (v, w)
        λ ← min{c(v, V \ v), λ}
    return λ
end
```

3 Теория Менгера

Мы не раз в процессе доказательства утверждений неявно использовали результаты теории Менгера [24, 25]. В этом разделе мы установим достаточно сильный результат: какой бы граф ни был, как только степень каждой вершины будет хотя бы k, так сразу в нем найдутся k путей между любыми двумя наперед заданными вершинами таких, что их средняя длина есть $O\left(\frac{n}{\sqrt{k}}\right)$.

3.1. Основные определения

Определение 3.1. Пусть $G = (V, E)$ – простой (без циклов и параллельных ребер) неориентированный граф. Будем говорить, что две вершины w и v k-реберно связаны, если для любого подмножества $V_1 \subset V$ такого, что $w \in V_1$ и $v \notin V_1$, величина разреза (то есть количество ребер, у которых один конец лежит в V_1, а другой – в $V \setminus V_1$) не меньше k. Будем говорить, что граф k-реберно связен, если любые две различные вершины k-реберно связаны.



(а) (б) (в)

Рис. 4. Примеры k-связанных вершин и k-связного графа. (а) Вершины a и b 2-связаны. (б) Вершины u и v 3-связаны. (в) 3-связный граф.

Определение 3.2. Пусть задан граф $G = (V, E)$. Под длиной $\text{length}(p)$ пути p в графе G будем понимать число, равное количеству ребер в этом пути. Пусть $P_{u,v} = \{p_1, p_2, \ldots, p_k\}$ – набор реберно-непересекающихся путей между вершинами u и v. Определим

- среднюю длину пути: $\text{Avg}(u, v) = \frac{1}{k} \sum_{i=1}^{k} \text{length}(p_i)$;
- длину наибольшего пути: $\text{Long}(P_{u,v}) = \max\{\text{length}(p_i) \colon 1 \leqslant i \leqslant k\}$;
- $P_{avg}(u, v)$ – набор путей, минимизирующий $\text{Avg}(P_{u,v})$;

- $P_{long}(u, v)$ – набор путей, минимизирующий $\mathrm{Long}(P_{u,v})$;
- k-среднее расстояние: $B^{(k)}_{avg}(u, v) = \mathrm{Avg}(P_{avg}(u, v))$;
- k-подавляющее расстояние: $B^{(k)}_{long}(u, v) = \mathrm{Long}(P_{long}(u, v))$.

Определение 3.3. Пусть P – набор реберно-непересекающихся путей между вершинами u и v. Назовем uv-ориентацией P граф, полученный из P заменой каждого ребра на дугу, ориентированную в соответствии с направлением прохода по ней вдоль пути из u в v.

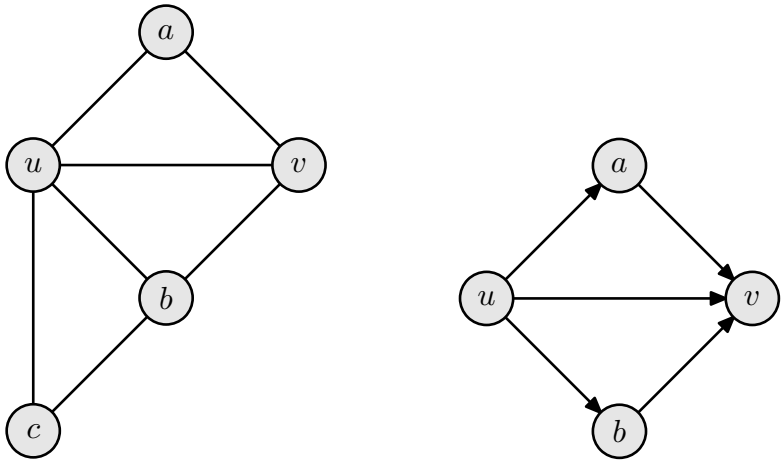


Рис. 5. Построение ориентации. (а) Исходный граф, в котором рассматривается набор путей $P_{avg}(u, v)$. (б) uv-ориентация $P_{avg}(u, v)$.

3.2. Короткие пути между k-связанными вершинами

Пусть D – uv-ориентация $P_{avg}(u, v)$.

Лемма 3.4. *D ациклична.*

Доказательство. В силу оптимальности P_{avg} каждый путь из набора P_{avg} не содержит цикла. Предположим, что D содержит цикл $C = s_1 s_2 \dots s_l$ и докажем лемму от противного.

Разделим C на фрагменты, каждый из которых будет содержаться ровно в одном пути из набора. Естественно, разные фрагменты могут оказаться в одинаковых путях. Заметим, что конечная вершина одного фрагмента является начальной вершиной другого. Кроме того, поскольку u может быть только началом дуги, а v – только концом, то они не могут находиться в одном и том же цикле.

Для каждого сегмента найдется дуга, входящая в начальную вершину фрагмента и исходящая из конечной вершины фрагмента. Обозначим за r_i и t_i ($i = 1, \ldots, l$) входящие и исходящие ребра соответствующих сегментов. Тогда мы можем преобразовать k путей так, что они останутся реберно-непересекающимися, но не будут содержать дуг из C, что противоречит свойству оптимальности $P_{avg}(u, v)$.

Новые пути можно строить, например, так. Возьмем старый путь. Новый пойдет вдоль него до тех пор, пока не встретится некоторая дуга r_i. Теперь, вместо того, чтобы войти в цикл, он пойдет вдоль t_{i-1} (если $i = 1$, то используется t_l) и так далее. Отметим, что до преобразования у каждой дуги была уникальная предшествующая ей дуга. Поскольку преобразование переставляет и удаляет некоторых предшественников, то уникальность останется и после преобразования.

Теперь осталось показать, что:

- каждый новый путь достигнет v (справедливо в силу свойства уникальности предшественника: если какой-то путь не приходит в v, то он содержит цикл);
- новые пути будут реберно-непересекающимися (справедливо в силу свойства уникальности предшественника).

□

Поскольку D ациклична, ее можно представить в виде графа слоев, используя топологическое упорядочивание:

1) слой H_0 состоит из единственной вершины u;
2) для получения нового слоя удаляем все вершины и ребра, находящиеся в уже составленных слоях;
3) очередной слой H_i составят вершины, не имеющие входящих дуг.

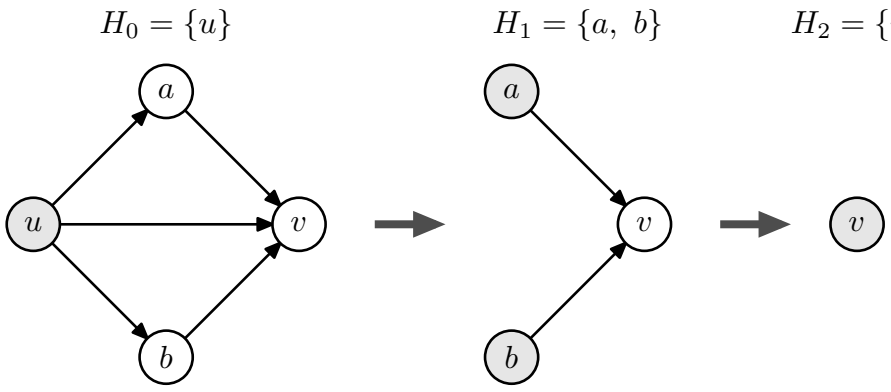


Рис. 6. Последовательное построение слоев.

Определение 3.5. Пусть граф слоев D имеет $l+1$ слой. Обозначим H_i множество вершин на i-м слое, где $0 \leqslant i \leqslant l$.

- Интервал $H[i,j]$ определяется как подграф, полученный сужением D на множество вершин $\bigcup\limits_{i \leqslant h \leqslant j} H_h$.
- Интервал $H[i,j]$ называется s-интервалом, если $j - i + 1 = s$.
- Будем говорить, что дуга a покрывает интервал $H[i,j]$, если $a \in H_i \times H_j$.
- Дуга a называется базовой, если найдется такое i, что a покрывает интервал $H[i, i+1]$.
- Разрез C_i, где $0 \leqslant i < l$, определяется как множество ребер, начинающихся в $H[0,i]$ и заканчивающихся в $H[i+1,l]$.

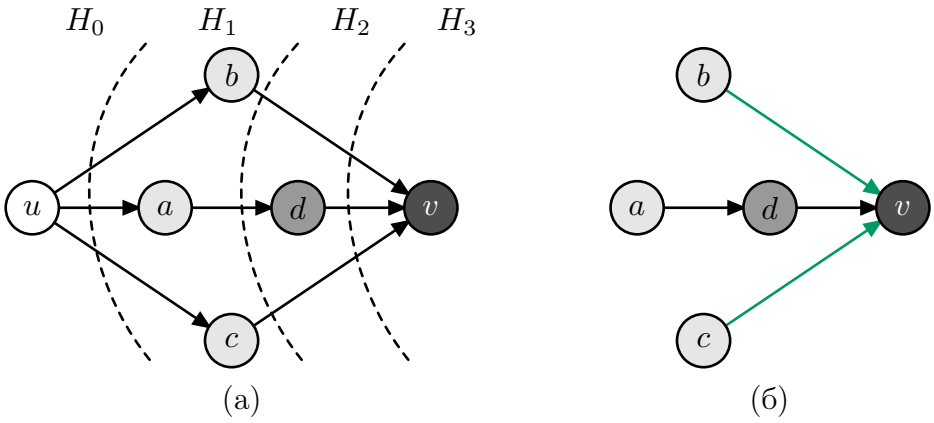


Рис. 7. (а) uv-ориентация $P_{avg}(u,v)$ со слоями $H_0 = \{u\}$, $H_1 = \{a,b,c\}$, $H_2 = \{d\}$, $H_3 = \{v\}$. (б) Интервал $H[1,3]$, (b,v) и (c,v) – покрывающие интервал $H[1,3]$ дуги.

Отметим некоторые свойства графа слоев.

- Наивысший слой $H_0 = \{u\}$.
- Самый низкий слой $H_l = \{v\}$.
- Все дуги ориентированы сверху вниз, то есть начальная вершина любой дуги расположена в более высоком (с меньшим номером) слое, чем конечная. Разрез C_i имеет величину ровно k для любого i, $0 \leqslant i < l$.
- $\mathrm{Long}(P_{avg}(u,v)) \leqslant l$.

- Для любой вершины не из H_0 найдется входящая в нее базовая дуга. Заметим, однако, что возможны вершины не из H_l, имеющие исходящую дугу, не являющуюся базовой.
- За исключением u и v, любая вершина имеет входящую степень, равную исходящей.

Далее, мы определим операцию SINK, которая позволит преобразовать один граф в другой с таким же количеством ребер. Причем мы сможем оценить число ребер в новом графе.

Определение 3.6. Будем говорить, что

- интервал $H[i, j]$ s-насыщен, где s удовлетворяет неравенству $1 \leqslant s \leqslant j - i + 1$, если $i \leqslant j$ и, кроме того, любой s-подынтервал формирует полный s-дольный граф при отброшенной ориентации дуг, то есть между любыми двумя вершинами из разных слоев в подынтервале имеется дуга;
- наибольший номер s, для которого интервал $H[i, j]$ s-насыщен, называется степенью насыщения интервала $H[i, j]$;
- интервал $H[i, j]$ полностью насыщен, если $s = j - i + 1$;
- дуга, покрывающая интервал $H[i, j]$, может быть «утоплена», если степень насыщения $H[i, j]$ строго меньше $j - i$.

Определение 3.7 (операция SINK**).** В ходе операции SINK(e) дуга e заменяется дугой, покрывающей наивысший ненасыщенный $(s + 1)$-подынтервал в $H[i, j]$ (предполагается, что он указан заранее).

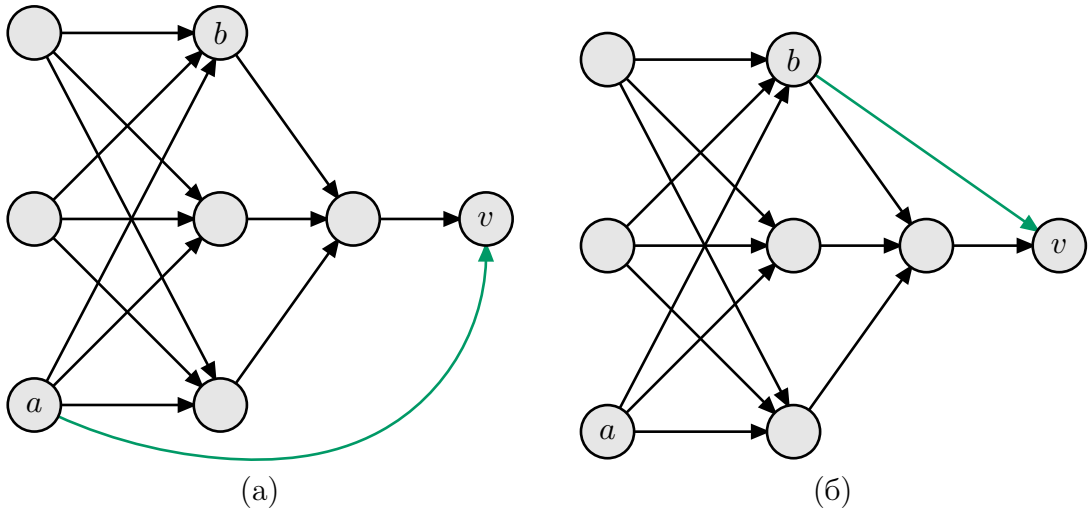


Рис. 8. (а) 2-насыщенный интервал $H[1, 4]$. Дугу $e = (a, v)$ можно «утопить». (б) После выполнения операции SINK(e) для $e = (a, v)$ появилась дуга (b, v).

Определение 3.8 (**операция** FS)**.** Пусть задан граф G. Определим операцию $\mathrm{FS}(G)$ следующим алгоритмом.

```
procedure FS(G)
begin
    for each (для каждой) дуги {e: e можно «утопить»} do
        SINK(e)
    return G    // такие графы иногда называют «утопленными»
end
```

Свойство 1. *После выполнения операции* SINK *в графе сохраняется*

- *количество слоев*;
- *множество вершин в каждом слое*;
- *суммарное количество дуг.*

Свойство 2. *После выполнения операции* SINK

- *число ребер в C_i не возрастает для всех i, $0 \leqslant i \leqslant l$;*
- *как только дуга покрывает интервал $H[i,j]$, так сразу $H[i,j]$ будет $(j-i)$-насыщен.*

Теорема 3.9. *Пусть G – простой неориентированный граф, u, v – k-реберно связанная пара вершин. Тогда $B^{(k)}_{avg}(u,v) = O\left(\frac{n}{\sqrt{k}}\right)$.*

Доказательство. Пусть $W = \sum\limits_{i=1}^{k} \mathrm{length}(p_i)$, тогда $B^{(k)}_{avg}(u,v) = \frac{W}{k}$. Заметим, что W равно количеству ребер в D или, что эквивалентно, в $F = \mathrm{FS}(D)$. Ниже мы докажем для W оценку $O(n\sqrt{k})$.

Для множества слоев A положим $\|A\| = \sum\limits_{H_i \in A} |H_i|$. Разделим слои на два множества:

- множество S слоев H_i таких, что $|H_i| < \sqrt{k}$, за исключением первого и последнего слоя;
- множество L, содержащее все слои, не вошедшие в множество S.

По построению $|L| \leqslant \frac{\|L\|}{\sqrt{k}+2}$. Для любого $H_i \in L$ в силу $|C_{i-1} \cup C_i| \leqslant 2k$ справедлива оценка на число X дуг, касающихся либо проходящих через слои в L:

$$X \leqslant \sum_{H_i \in L} |C_{i-1} \cup C_i| = O(k|L|) = O(\sqrt{k}\|L\|) = O(n\sqrt{k}).$$

Оставшаяся часть дуг может только касаться либо проходить через слои в S. Чтобы проанализировать этот случай, мы введем последовательность номеров слоев

$$0 = l_0, l_1, \ldots, l_h = l \in F$$

таких, что для i, $0 \leqslant i < h$, $H_{l_{i+1}}$ – это самый дальний слой, в который может попасть любая дуга из H_{l_i}. Поскольку F не содержит дуг, которые можно утопить, интервал $H[l_i, l_{i+1}]$ $(l_{i+1} - l_i)$-насыщен и не одна дуга из слоев выше H_{l_i} не сможет достигнуть слоев ниже $H_{l_{i+1}}$. Интервал $H[l_i, l_{i+1}]$ назовем усечением и обозначим T_i.

Рассмотрим два слоя $H_x, H_y \in L$ таких, что все слои между ними находятся в S. Пусть $S_1 = \{H_i \colon x < i < y\}$, а $T_a, T_{a+1}, \ldots, T_b$ – минимальная последовательность усечений, которая покрывает S_1. Обозначим $G_i = S_1 \cap T_{a+i-1}$, $1 \leqslant i \leqslant q$, $q = b - a + 1$.

Замечание 1. *Для любого i, $1 \leqslant i \leqslant q$, справедлива оценка $\|G_i\| = O(\sqrt{k})$.*

Доказательство. Пусть $\|G_i\| > 5\sqrt{k}$ для фиксированного $1 \leqslant i \leqslant q$. Поскольку наивысший слой $H_f \in G_i$ и самый низкий слой $H_g \in G_i$ имеют размер, меньший $\sqrt{k}$, найдется хотя бы $3\sqrt{k}$ вершин в промежуточных слоях. Поскольку каждый промежуточный слой имеет размер, меньший $\sqrt{k}$, найдется промежуточный слой $H_j \in G_i$ такой, что $\|H[f+1, j] \cap G_i\| > \sqrt{k}$ и $\|H[j+1, g-1] \cap G_i\| > \sqrt{k}$. В силу того, что $H[f+1, g-1]$ полностью насыщен, найдется не более чем $\sqrt{k}\sqrt{k} = k$ дуг в C_j, что ведет к противоречию, то есть $\|G_i\| \leqslant 5\sqrt{k}$. □

Оценим число Y дуг, которые не были учтены при подсчете X (это дуги, которые не касаются и не пересекают слои в L). В силу свойства слоев, любая такая дуга либо в G_i (такие дуги мы назовем внутренними), либо она ведет из G_i в G_{i+1} (такие дуги мы назовем внешними). Число внутренних дуг не превосходит $\|G_i\|^2$. Число внешних дуг из G_i в G_{i+1} не превосходит $\|G_i\|\|G_i + 1\|$, что в свою очередь ограничено $\frac{\|G_i\|^2 + \|G_{i+1}\|^2}{2}$. Таким образом,

$$Y \leqslant 2 \sum_{G_i} \|G_i\|^2,$$

где G_i пробегает множество слоев между большими слоями H_x и H_y, описанными выше. Поскольку два таких G_i перекрываются не более чем в одном слое, общий размер G_i меньше, чем $2\|S\|$. В ограничении $\|G_i\| \leqslant 5\sqrt{k}$ можно показать, что Y достигает своего максимума, когда каждый G_i имеет наибольший размер – $5\sqrt{k}$. В этом случае имеем:

$$Y \leqslant 2(5\sqrt{k})^2 \cdot \left(\frac{2}{5}\right) \cdot \frac{\|S\|}{\sqrt{k}} = O(\sqrt{k}\|S\|) = O(n\sqrt{k}).$$

Итак, получаем: $W = X + Y = O(n\sqrt{k})$. □

Аналогичное доказательство имеет место и в случае ориентированных графов.

Следствие 1. *Пусть вершина и k-реберно связана с v в простом ориентированном графе. Тогда* $B^{(k)}_{avg}(u, v) = O\left(\frac{n}{\sqrt{k}}\right)$.

Замечание 2. *Оценка, установленная в теореме 3.9, точна и имеет место только лишь для средних путей. Для длинных путей оценка может оказаться неверной.*

Следствие 2. *Пусть вершина и k-реберно связана с v в простом ориентированном графе. Тогда существует $\Omega(k)$ реберно-непересекающихся путей между u и v таких, что каждый из них имеет длину* $O\left(\frac{n}{\sqrt{k}}\right)$.

4 Определения и обозначения

- **m** – число ребер в исходном графе $G = (V, E)$: $m = |E|$.
- **n** – число вершин в исходном графе $G = (V, E)$: $n = |V|$.
- **s-t разрез** – разбиение множества вершин V на два дизъюнктных подмножества S и T, причем $s \in S$ и $t \in T$.
- **PR-тесты (PR-эвристики)** – комплекс эвристик Падберга–Ринальди [17], позволяющих стягивать определенные ребра в процессе вычисления минимального разреза.
- **push-relabel algorithm, preflow-push algorithm** – алгоритм проталкивания предпотока.
- **Активной** называется вершина $v \neq \{s, t\}$ с положительным избытком $e_f(v) > 0$.
- **Безопасным** ребром называется ребро, которое не будет входить в искомый минимальный разрез.
- **Величина** потока f определяется как сумма потоков по всем ребрам, ведущим в сток: $|f| = \sum\limits_{v \in V} f(v, t)$.

- **Динамические деревья**. Для улучшения производительности алгоритма проталкивания предпотока была предложена идея выполнения сразу серии проталкиваний вдоль пути в одном направлении с использованием динамических деревьев [26, 27]. Эту структуру данных разработали Слитор и Тарьян. Данная структура данных представляет собой лес вершинно-непересекающихся деревьев.

- **Дополняющий путь** – простой путь из истока s в сток t в остаточной сети G_f.

- **Допустимой** называется дуга $(v, w) \in E_f$ остаточной сети, если $d(v) = d(w) + 1$.

- **Задача о максимальном потоке в сети**: для данной сети G с истоком s и стоком t найти поток максимальной величины.

- **Задача о минимальном разрезе** – задача разделения взвешенного графа с n вершинами и m ребрами на два множества так, чтобы суммарный вес ребер, концевые вершины которых лежат в разных множествах, был минимален.

- **Избыток** $e_f(v)$ для вершины $v \neq \{s, t\}$ – разница входящего в v и исходящего из v потоков.

- **Метки расстояний** – это функция $d\colon V \to \mathbb{Z}_+$, для которой выполнено: $d(t) = 0$, $d(s) = n$, и $d(v) \leqslant d(w) + 1$ для всех дуг остаточной сети (v, w).

- **Минимальным разрезом в сети** называется разрез наименьшей пропускной способности среди всех разрезов данной сети.

- **Насыщающим** называется проталкивание потока из вершины u в вершину v, в результате которого ребро (u, v) становится насыщенным. В противном случае проталкивание считают **ненасыщающим**.

- **Насыщенным** называется ребро (u, v), остаточная пропускная способность которого обращается в нуль: $c_f(u, v) = 0$ (ребро исчезает из остаточной сети).

- **Остаточная сеть** состоит из тех ребер, поток по которым можно увеличить: $G_f = (V, E_f)$, где $E_f = \{(u, v) \in V \times V\colon c_f(u, v) > 0\}$.

- **Остаточной пропускной способностью из u в v** называется величина, определяющая, сколько еще потока можно направить из u в v: $c_f(u, v) = c(u, v) - f(u, v)$.

- **Остаточные ребра** – ребра остаточной сети G_f.
- **Потоком** в сети $G = (V, E)$ называется функция $f\colon V \times V \to \mathbb{R}$, обладающая следующими тремя свойствами:
  1) кососимметричность: $f(u, v) = -f(v, u)$ для всех u, v из V;
  2) ограничение, связанное с пропускной способностью: $f(u, v) \leqslant c(u, v)$ для всех u, v из V;
  3) сохранение потока: $\sum\limits_{v \in V} f(u, v) = 0$ для всех $u \in V \setminus \{s, t\}$.
- **Правило изменения меток при создании пропусков**. Перед изменением метки расстояния вершины v сначала выполняется проверка, имеет ли какая-нибудь другая вершина метку расстояния $d(v)$. Если ответ – да, то выполняется обычная процедура изменения метки расстояния для v. Иначе, все вершины с превосходящими $d(v)$ метками расстояний удаляются из графа, так как сток недостижим из этих вершин. Эта эвристика часто ускоряет алгоритм проталкивания предпотока и существенна при анализе алгоритма Хао–Орлина.
- **Предпоток** – это функция $f\colon V \times V \to \mathbb{R}$, которая
  1) кососимметрична: $f(u, v) = -f(v, u)$ для всех u, v из V;
  2) удовлетворяет ограничениям, связанным с пропускными способностями: $f(u, v) \leqslant c(u, v)$ для всех u, v из V;
  3) удовлетворяет ослабленному закону сохранения: $\sum\limits_{v \in V} f(v, u) \geqslant 0$ для всех $u \in V \setminus \{s, t\}$.
- **Пропускной способностью разреза** (S, T) называется сумма $c(S, T)$ пропускных способностей пересекающих разрез ребер:
  $c(S, T) = \sum\limits_{u \in S,\ v \in T,\ (u,\ v) \in E} c(u, v)$.
- **Разрезом** сети $G = (V, E)$ называется разбиение множества вершин V на две части S и $T = V \setminus S$, для которых $s \in S$ и $t \in T$.
- **Сеть** – ориентированный граф $G = (V, E)$, каждому ребру $(u, v) \in E$ которого поставлено в соответствие число $c(u, v) \geqslant 0$, называемое **пропускной способностью** ребра.
- **Стягивание** ребра (v, w) выполняется путем удаления вершины w и замены каждого ребра вида (w, x) на ребро вида (v, x). Если это приводит

к появлению кратных ребер, то они объединяются в одно ребро, при этом их пропускные способности складываются. Возникающие петли удаляются. Граф G, в котором было стянуто ребро (v, w), обозначается $G \setminus (v, w)$.

- **Теорема о максимальном потоке и минимальном разрезе**. Пусть f – поток в сети $G = (V, E)$ с истоком s и стоком t. Тогда следующие утверждения равносильны:
  - поток f максимален (является потоком максимальной величины) в сети G;
  - остаточная сеть G_f не содержит дополняющих путей;
  - для некоторого разреза (S, T) сети G выполнено равенство $|f| = c(S, T)$ (в этом случае разрез является минимальным s-t разрезом).

Литература

[1] L. R. Jr. Ford, D. R. Fulkerson. Maximum flow through a network. *Canadian Journal of Mathematics*, Vol. 8, pp. 399–404, 1956.

[2] Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 1999. – 960 с., 263 ил.

[3] A. V. Goldberg, R. E. Tarjan. A New Approach to the Maximum Flow Problem. *Journal of the ACM*, Vol. 35, pp. 921–940, 1988.

[4] C. S. Chekuri, A. V. Goldberg, D. R. Karger, M. S. Levine, C. Stein. Experimental study of minimum cut algorithms. *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pp. 324–333, 1997.

[5] R. A. Botafogo. Cluster analysis for hypertext systems. *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 116–125, 1993.

[6] S. Chatterjee, R. Schreiber, T. J. Sheffler, J. R. Gilbert. Array Distribution in Data-Parallel Programs. *Proceedings of the 7th International Workshop on Languages and Compilers for Parallel Computing*, pp. 76–91, 1994.

[7] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, No. 64, pp. 275–278, 1958.

[8] M. W. Padberg, M. Grötschel. Polyhedral computations. The Traveling Salesman Problem. Wiley, Chichester, pp. 307–360, 1995.

[9] M. W. Padberg, G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, Vol. 33, pp. 60–100, 1991.

[10] G. B. Dantzig, R. Fulkerson, S. M. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, Vol. 2, pp. 393–410, 1954.

[11] D. R. Karger. A Randomized Fully Polynomial Time Approximation Scheme for the All Terminal Network Reliability Problem. *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pp. 11–17, 1995.

[12] A. Ramanathan, C. J. Colbourn. Counting almost minimum cutsets with reliability applications. *Mathematical Programming: Series A*, Vol. 39, No. 3, pp. 253–261, 1987.

[13] P. V. Nepomnyaschy, D. V. Yurin. Creation of hierarchical detail tree of image segmentation via solving minimum cut problem. *Proceedings of the 13th International Conference on Computer Graphics GraphiCon*, 2003.

[14] M. V. Mintchenkov, D. V. Yurin, A. V. Helvas. Unsupervised Algorithm for Images Segmentation on the Base of the Rayleigh 2D Objects Boundaries Detector. *Proceedings of the 12th International Conference on Computer Graphics GraphiCon*, pp. 243–250, 2002.

[15] D. S. Hochbaum, A. Chen. Performance analysis and best implementations of old and new algorithms for the Open-Pit Mining Problem. To appear in Operations Research.

[16] R. E. Gomory, T. C. Hu. Multi-terminal network flows. *SIAM Journal on Discrete Mathematics*, Vol. 9, pp. 551–570, 1961.

[17] M. W. Padberg, G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming: Series A and B*, Vol. 47, No. 1, pp. 19–36, 1990.

[18] J. Hao, J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pp. 165–174, 1992.

[19] H. N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, Vol. 50, No. 2, pp. 259–273, 1995.

[20] D. R. Karger, C. Stein. An $\widetilde{O}(n^2)$ algorithm for minimum cuts. *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pp. 757–765, 1993.

[21] D. R. Karger, C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, Vol. 43, No. 4, pp. 601–640, 1996.

[22] H. Nagamochi, T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, Vol. 5, No. 1, pp. 54–66, 1992.

[23] R. K. Ahuja, T. L. Magnanti, J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1993.

[24] Z. Galil, X. Yu. Short length versions of Menger's theorem. *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pp. 499– 508, 1995.

[25] H. Ripphausen-Lipa, D. Wagner, K. Weihe. The vertex-disjoint menger problem in planar graphs. *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pp. 112–119, 1993.

[26] D. D. Sleator, R. E. Tarjan. A Data Structure for Dynamic Trees. *Journal of Computer and System Sciences*, Vol. 26, pp. 362–391, 1983.

[27] D. D. Sleator, R. E. Tarjan. Self-adjusting Binary Search Trees. *Journal of the ACM*, Vol. 32, pp. 652–686, 1985.