05.05.2025

Локальный поиск. Рандомизированные алгоритмы

Филиппов Михаил Витальевич

m.filippov@g.nsu.ru

89232283872

Императивное программирование, 2024-2025

Новосибирский государственный университет

*НАСТОЯЩАЯ НАУКА

Давайте познакомимся

Филиппов Михаил Витальевич

- Окончил магистратуру ФФ НГУ
- Окончил аспирантуру ИТ СО РАН
- Являюсь м.н.с. ИТ СО РАН
- 7+ лет опыт в программировании C/C++

Адженда

Локальный поиск	Рандомизированные алгоритмы
45 минут	45 минут

Адженда

Локальный поиск

45 минут

Рандомизированные алгоритмы

45 минут

Введение

Локальный поиск — чрезвычайно универсальный прием; этим термином описываются любые алгоритмы, которые последовательно «исследуют» пространство возможных решений, перемещаясь за один шаг от текущего решения к другому, «ближнему». Общность и гибкость этого метода имеют свои преимущества: алгоритм на базе локального поиска можно без особых трудностей разработать почти для любой вычислительно сложной задачи; с другой стороны, часто бывает очень трудно сказать что-нибудь конкретное или доказуемое о качестве решений, находимых алгоритмом локального поиска, и, соответственно, очень сложно определить, хороший или плохой алгоритм локального поиска используется в каждом конкретном случае.

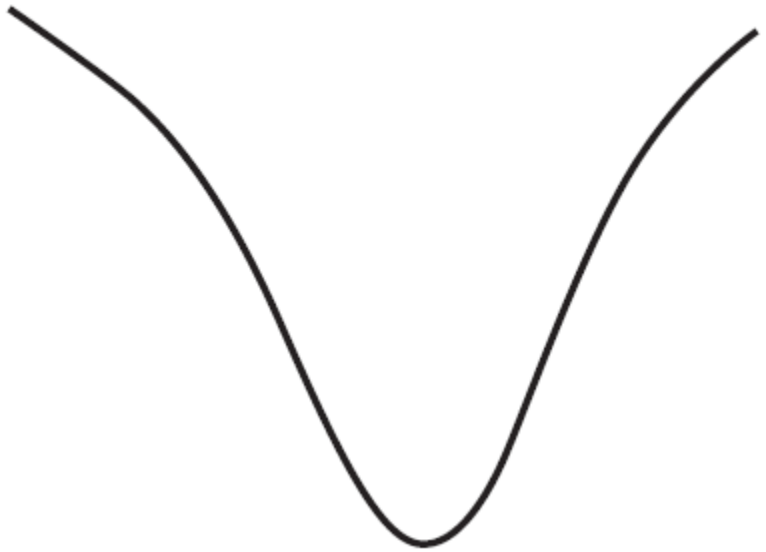
Алгоритмы локального поиска обычно представляют собой эвристики, предназначенные для нахождения хороших (но не обязательно оптимальных) решений вычислительных задач, и для начала мы поговорим о том, как выглядит поиск таких решений на глобальном уровне. Полезная интуитивно понятная аналогия существует в принципе минимизации энергии в физике, поэтому мы сначала исследуем этот вопрос. Впрочем, в отдельных случаях нам удастся доказать некоторые свойства алгоритмов локального поиска и связать их производительность с оптимальным решением.

Задача оптимизации в перспективе

Основные концепции локального поиска были разработаны учеными, мыслившими аналогиями с физикой. Рассматривая широкий спектр сложных вычислительных задач, требующих минимизации некоторой величины, они рассуждали следующим образом: физические системы постоянно решают задачу минимизации, стремясь к минимуму потенциальной энергии. Чему можно научиться, наблюдая за минимизацией в природе? Не подскажет ли она какие-нибудь новые алгоритмы?

Потенциальная энергия

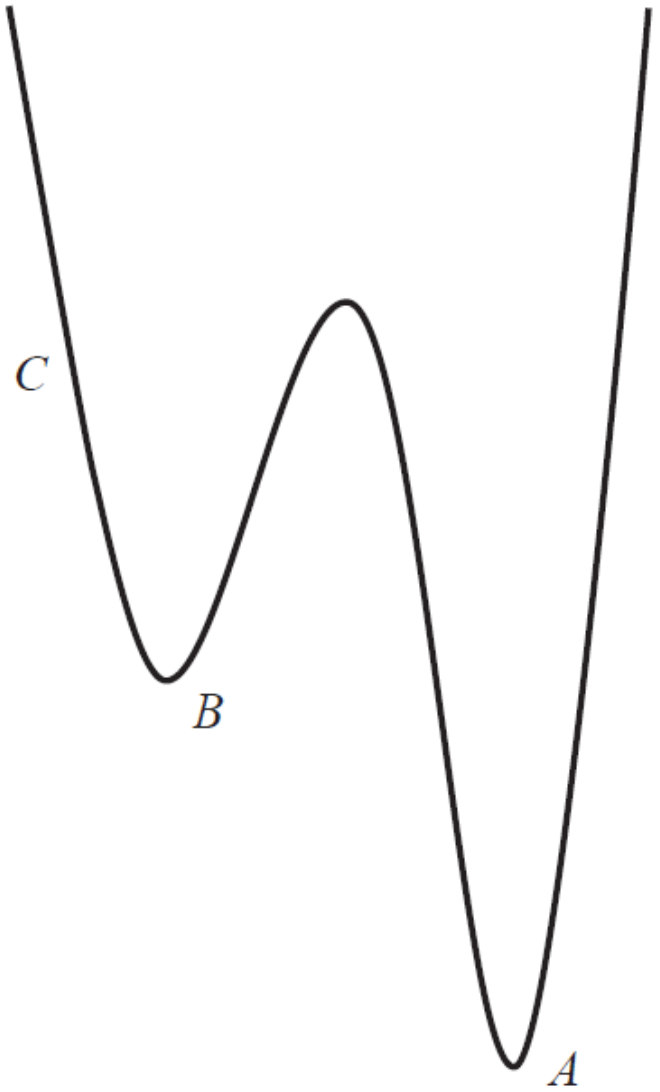
Если бы мир выглядел так, как нам рассказывают в начальном курсе механики, он состоял бы исключительно из хоккейных шайб, скользящих по льду, и шаров, скатывающихся по наклонным поверхностям. Шайба скользит, потому что ее толкнули; но почему шары скатываются вниз? Из ньютоновской механики известно, что шар стремится минимизировать свою потенциальную энергию. В частности, если шар имеет массу m и падает на расстояние h, он теряет потенциальную энергию, пропорциональную mh. Итак, если отпустить шар у верхнего края воронкообразного углубления (рис.), его потенциальная энергия будет минимальной в самой нижней точке.



Когда поверхность потенциальной энергии имеет структуру простой воронки, найти нижнюю точку достаточно легко

Потенциальная энергия

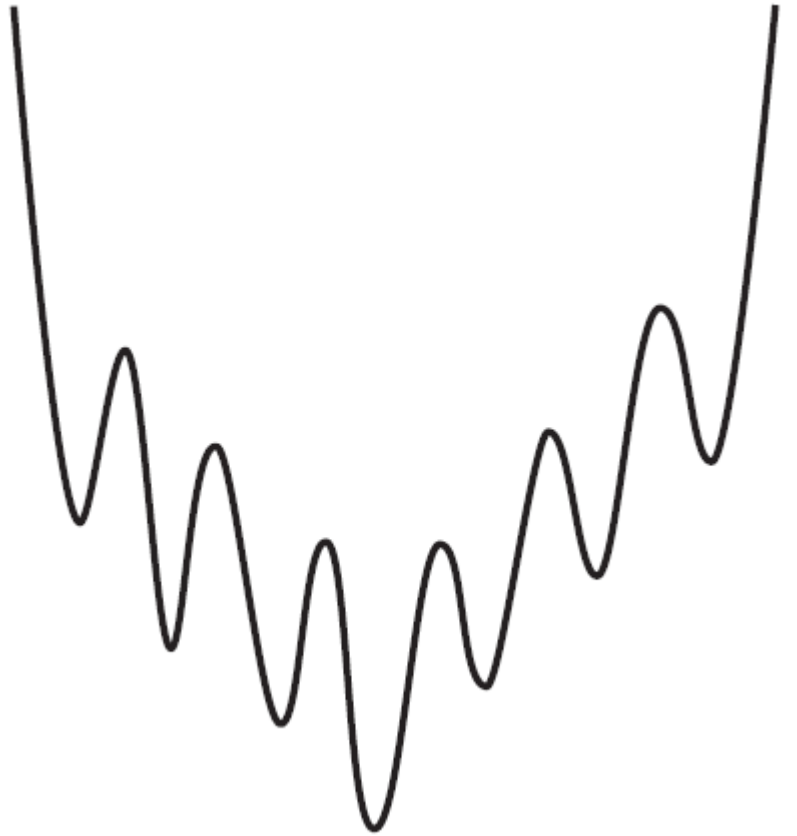
При усложнении структуры поверхности проявляются дополнительные сложности. Возьмем «двойную воронку» на рис. Точка А находится ниже точки В, поэтому она является предпочтительной для нахождения шара в состоянии покоя. Но если шар скатывается с точки С, он не сможет преодолеть барьер между двумя воронками и окажется в В. В таких случаях говорят, что шар попал в локальный минимум: он находится в самой нижней точке, если рассматривать окрестности текущего положения; но если отступить и взглянуть на ситуацию в целом, мы видим, что глобальный минимум не достигнут.



Большинство поверхностей сложнее простых воронок; например, в этой «двойной воронке» существуют большой глобальный минимум и меньший локальный минимум

Потенциальная энергия

Конечно, очень большие физические системы также должны стремиться к минимизации своей энергии. Представьте, например, что вы взяли несколько граммов однородного вещества, нагрели его и изучаете его поведение во времени. Чтобы точно отразить потенциальную энергию образца, теоретически необходимо представить поведение каждого атома вещества и его взаимодействие с ближними атомами. Но также будет полезно рассмотреть свойства системы в целом (то есть всего образца), и здесь на помощь приходит статистическая механика. Мы еще вернемся к статистической механике, а пока просто заметим, что концепция «поверхности потенциальной энергии» предоставляет полезную и наглядную аналогию для процесса, который используется в больших физических системах для минимизации энергии.



Обобщенная поверхность потенциальной энергии может содержать множество локальных минимумов, усложняющих поиск глобального минимума, как в изображенной на рисунке «гребенке»

Связь с оптимизацией

Этот подход к минимизации энергии в действительности базируется на нескольких основных положениях: физическая система может находиться в одном из многочисленных состояний; ее энергия является функцией текущего состояния; небольшое возмущение в заданном состоянии приводит к «соседнему» состоянию. Структура связи этих соседних состояний вместе со структурой энергетической функции определяет энергетическую поверхность.

Под этим углом мы снова взглянем на задачу вычислительной минимизации. В типичной задаче такого рода имеется большое (как правило, имеющее экспоненциальный размер) множество C возможных решений. Также понадобится функция стоимости $c(\cdot)$, оценивающая качество каждого решения; для решения $S \in C$ его стоимость записывается в виде $c(S)$. Целью является поиск решения $S^* \in C$ с наименьшим возможным значением (S^*). До настоящего момента мы рассматривали подобные задачи именно так. А теперь добавим в решения понятие соседских отношений, отражающих идею о том, что одно решение S' может быть получено незначительным изменением другого решения S. Запись $S \sim S'$ означает, что S' является соседним решением S, а множество соседей S, то есть $\{S' : S \sim S'\}$, будет обозначаться $N(S)$. Нас в первую очередь интересуют симметричные соседские отношения, хотя основные рассматриваемые принципы также распространяются и на асимметричные соседские отношения. Здесь принципиально то, что хотя множество C возможных решений и функция стоимости $c(\cdot)$ предоставляются в спецификации задачи, мы свободны выбирать соседские отношения по своему усмотрению.

Связь с оптимизацией

Алгоритм локального поиска получает эту конфигурацию, включая соседские отношения, и работает по следующей высокоуровневой схеме. Он постоянно поддерживает текущее решение $S \in C$. На каждом шаге он выбирает соседа S' решения S, объявляет S' новым текущим решением и повторяет процесс. На протяжении всего выполнения алгоритма запоминается решение с наименьшей стоимостью из всех встречавшихся; итак, по мере выполнения постепенно находятся все лучшие и лучшие решения. Суть алгоритма локального поиска заключается в выборе соседских отношений и в формулировке правила выбора соседнего решения на каждом шаге. Итак, соседские отношения можно рассматривать как определяющий (обычно ненаправленный) граф множества всех возможных решений, в котором ребра соединяют соседние пары решений. В этом случае алгоритм локального поиска может рассматриваться как обход графа с попыткой перемещения в направлении хорошего решения.

Локальный поиск в задаче о вершинном покрытии

Рассмотрим работу локального поиска на примере задачи о вершинном покрытии. Учтите, что задача о вершинном покрытии является хорошим примером, но существует много других задач оптимизации, которые с таким же успехом можно использовать в качестве примера.

Итак, имеется граф G = (V, E); множество C возможных решений состоит из всех подмножеств S множества V, образующих вершинные покрытия. Например, всегда выполняется V ∈ C. Стоимость c(S) вершинного покрытия S равна его размеру; таким образом, минимизация стоимости вершинного покрытия эквивалентна нахождению покрытия с минимальным размером. Наконец, в наших примерах алгоритмов локального поиска будет использоваться очень простое соседское отношение: мы говорим, что S ~ S', если решение S' может быть получено из S добавлением или удалением одного узла. Таким образом, наши алгоритмы локального поиска будут обходить пространство возможных вершинных покрытий, добавляя или удаляя из текущего решения узел на каждом шаге и пытаясь найти как можно меньшее вершинное покрытие.

Локальный поиск в задаче о вершинном покрытии

У этого соседского отношения есть одно полезное свойство:

Утверждение 1.1. Каждое вершинное покрытие S имеет не более n соседних решений. Утверждение доказывается попросту тем, что каждое соседнее решение S получается добавлением или удалением узла. Из (1.1) следует, что в процессе выбора можно эффективно исследовать все возможные соседние решения S.

Для начала рассмотрим очень простой алгоритм локального поиска — метод градиентного спуска. Градиентный спуск начинает с полного вершинного покрытия V и использует следующее правило выбора соседнего решения.

Обозначим S текущее решение. Если существует соседнее решение S' со строго меньшей стоимостью, то выбрать соседа, имеющего как можно меньшую стоимость. В противном случае завершить работу алгоритма.

Итак, метод градиентного спуска двигается строго вниз, пока может; когда дальнейшее движение становится невозможным, он останавливается.

Мы видим, что градиентный спуск завершается точно в тех решениях, которые являются локальными минимумами: то есть в таких решениях S, что для всех соседних S' выполняется $c(S) \leq c(S')$.

Это определение очень точно соответствует нашему понятию локального минимума на энергетических поверхностях: это те точки, для которых одношаговое возмущение не улучшает функции стоимости.

Локальный поиск в задаче о вершинном покрытии

Как наглядно представить поведение алгоритма локального поиска в контексте энергетических поверхностей, которые были представлены ранее? Начнем с градиентного спуска. Конечно, простейший экземпляр вершинного покрытия представляется n-узловым графом без ребер. Пустое множество является оптимальным решением (ребер для покрытия нет), а градиентный спуск превосходно справляется с нахождением этого решения: он начинает с полного множества вершин V и продолжает удалять узлы, пока не останется ни одного. В самом деле, множество вершинных покрытий для этого графа без ребер естественно соответствует воронке на рисунке потенциальной ямы: единственный локальный минимум также является глобальным минимумом, и к нему существует нисходящий путь от любой точки. Когда метод градиентного спуска работает неправильно? Рассмотрим «графзвезду» G, состоящий из узлов x_1, y_1, y_2, ..., y_{n-1}, в котором узел x_1 соединен с каждым из узлов y_i. Минимальное вершинное покрытие для G представляет собой одноэлементное множество $\{x_1\}$, а градиентный спуск может достигнуть этого решения, последовательно удаляя y_1, y_2, ..., y_{n-1} в любом порядке. Но если градиентный спуск начнет с удаления x_1, он немедленно оказывается в тупике: ни один узел y_i не может быть удален без разрушения свойства вершинного покрытия, поэтому единственным соседним решением является полное множество узлов V, обладающее более высокой стоимостью.

Локальный поиск в задаче о вершинном покрытии

Алгоритм «застревает» в локальном минимуме $\{y_1, y_2, ..., y_{n-1}\}$, обладающем очень высокой стоимостью по сравнению с глобальным минимумом. В графическом виде мы оказываемся в ситуации, соответствующей «двойной воронке». Глубокая воронка соответствует оптимальному решению {x1}, а мелкая воронка соответствует локальному минимуму $\{y_1, y_2, ..., y_{n-1}\}$. Спуск по неправильно выбранному склону, выбранному в самом начале, может привести к неправильному минимуму. Ситуация легко обобщается до двух минимумов с любыми относительными глубинами. Допустим, имеется двудольный граф G с узлами $x_1, x_2, ..., x_k$ и $y_1, y_2, ..., y_l$, where $k < l$ и существует ребро из каждого узла вида x_i в узел вида y_j. Тогда существуют два локальных минимума, соответствующих вершинным покрытиям$\{x_1,..., x_k\}$ и $\{y_1, ..., y_l\}$. Какой из этих минимумов будет обнаружен при выполнении градиентного спуска, зависит исключительно от того, будет ли первым удален элемент вида x_i или y_j.

Для более сложных графов часто бывает полезно подумать о том, какую поверхность они представляют; и наоборот, иногда можно взглянуть на поверхность и подумать о том, какой граф мог бы ее породить.

Локальный поиск в задаче о вершинном покрытии

Например, какой граф мог бы породить экземпляр задачи о вершинном покрытии с поверхностью вроде «гребенки»? Одним из таких графов мог бы быть n-узловой путь, в котором n — нечетное число, а узлы помечены в порядке v_1, v_2, ..., v_n. Уникальное минимальное вершинное покрытие S* состоит из всех узлов v_i, для которых i четно. Наряду с ним существует множество локальных оптимумов. Например, рассмотрим вершинное покрытие {v_2, v_3, v_5, v_6, v_8, v_9, ...}, в котором опущен каждый третий узел. Это вершинное покрытие существенно больше S*; при этом из него невозможно удалить ни один узел, сохранив покрытие всех ребер.

Как выясняется, найти минимальное вершинное покрытие S* методом градиентного спуска, начиная с полного вершинного покрытия V, очень трудно: после удаления одного узла v_i с четным значением i возможность нахождения глобального оптимума S* полностью теряется. Таким образом, различие между четными и нечетными узлами создает множество ошибочных путей для локального поиска, что и придает общей поверхности вид «гребенки». Конечно, между впадинами на рисунке и локальными оптимумами не существует прямого соответствия; как мы предупреждали ранее, рисунок всего лишь дает наглядное представление о происходящем. Но мы видим, что даже для графов с очень простой структурой градиентный спуск слишком прямолинеен для алгоритма локального поиска.

Алгоритм Метрополиса и имитация отжига

Первые идеи улучшенного алгоритма локального поиска были представлены в работе Метрополиса, Розенблата и Теллера (Metropolis, Rosenbluth, Teller, 1953). Они рассматривали задачу моделирования поведения физической системы в соответствии с принципами статистической механики. Базовая модель из этой области подразумевает, что вероятность нахождения физической системы в состоянии с энергией E пропорциональна функции Гиббса–Больцмана $e^{-E/(kT)}$, где T > 0 — температура, а k > 0 — константа. Присмотримся к этой функции. Для любой температуры T функция монотонно убывает по энергии E, поэтому из формулы следует, что физическая система находится в низкоэнергетическом состоянии с большей вероятностью, чем в высокоэнергетическом. Теперь рассмотрим эффект температуры T; при малых T вероятность низкоэнергетического состояния существенно выше вероятности высокоэнергетического состояния. Однако при высокой температуре разность между двумя вероятностями становится очень малой, и система почти с равной вероятностью может оказаться в любом состоянии.

Алгоритм Метрополиса

Метрополис вместе с другими авторами предложил следующий метод пошагового моделирования системы при фиксированной температуре T. В любой момент времени модель хранит текущее состояние системы и пытается сгенерировать новое состояние, применяя возмущение к текущему состоянию. Будем считать, что нас интересуют только состояния системы, «достижимые» из некоторого фиксированного исходного состояния через последовательность мелких возмущений; предполагается, что множество таких состояний C конечно. За один шаг сначала генерируется небольшое случайное возмущение в текущем состоянии S системы, приводящее к новому состоянию S'. Пусть E(S) и E(S') обозначают энергии S и S' соответственно. Если $E(S') \le E(S)$, то текущее состояние заменяется на S'. В противном случае пусть $\Delta E = E(S') - E(S) > 0$. Текущее состояние заменяется состоянием S' с вероятностью $e^{-\Delta E/(kT)}$, а в противном случае текущим состоянием остается S. Метрополис и др. доказали, что их моделирующий алгоритм обладает следующим свойством:

Утверждение 1.2 (без доказательства). Обозначим

$$Z = \sum_{S \in C} e^{-\frac{E(S)}{kT}}.$$

Пусть для состояния S запись $f_S(t)$ обозначает долю первых t шагов, в которых модель пребывает в состоянии S. Тогда предел $f_S(t)$ при стремлении t к ∞, с вероятностью, стремящейся к 1, равен $\frac{1}{Z} \cdot e^{-\frac{E(S)}{kT}}$.Именно такой факт нам и нужен; фактически он означает, что модель проводит приблизительно нужное время в каждом состоянии в соответствии с уравнением Гиббса–Больцмана.

Алгоритм Метрополиса

Чтобы использовать эту общую схему для разработки алгоритма локального поиска для задач минимизации, можно воспользоваться аналогией, в которой состояния системы соответствуют потенциальным решениям, а энергия соответствует стоимости. Работа алгоритма Метрополиса обладает парой свойств, очень полезных для алгоритма локального поиска: он склонен к «нисходящим» перемещениям, но также допускает небольшие «восходящие» перемещения с малой вероятностью. Такой алгоритм сможет двигаться дальше даже при заходе в локальным минимум. Более того, как выражено в (1.2), он глобально смещен в направлении решений с меньшей стоимостью. Ниже приводится конкретная формулировка алгоритма Метрополиса для задачи минимизации.

Начать с исходного решения S_0, констант k и T

За один шаг:

Пусть S — текущее решение

Случайно выбрать S' среди соседей S с равномерным распределением Если c(S') ≤ c(S)

Обновить S ← S'

Иначе С вероятностью $e^{-(c(S')-c(S))/(kT)}$

Обновить S ← S'

Иначе Оставить S без изменений

Конец Если

Алгоритм Метрополиса

```
void metropolis(void *initial_solution, double k, double T, int max_iterations) {
    srand(time(NULL)); // Инициализация генератора случайных ч
    void *current_solution = initial_solution;
    void *new_solution = NULL;
    for (int i = 0; i < max_iterations; i++) {
        new_solution = get_random_neighbor(current_solution);
        double current_cost = cost(current_solution);
        double new_cost = cost(new_solution);
        if (new_cost <= current_cost)
            copy_solution(current_solution, new_solution);
        else {
            double delta = new_cost - current_cost;
            double probability = exp(-delta / (k * T));
            double random = (double)rand() / RAND_MAX;
            if (random < probability)
                copy_solution(current_solution, new_solution);
        }
        free_solution(new_solution);
    }
}
```

Таким образом, для экземпляра задачи о вершинном покрытии, состоящего из графа-звезды (x_1 соединяется с каждым из узлов y_1, ..., y_{n-1}), мы видим, что алгоритм Метрополиса быстро выходит из локального минимума, возникающего при удалении x_1: соседнее решение, в котором x_1 возвращается обратно, будет сгенерировано и принято с положительной вероятностью. В более сложных графах алгоритм Метрополиса тоже может до некоторой степени исправить неверные решения, принятые в ходе выполнения.

Алгоритм Метрополиса

В то же время алгоритм Метрополиса не всегда ведет себя так, как нужно, причем даже в очень простых ситуациях. Вернемся к самому первому из рассмотренных графов — графу G без ребер. Градиентный спуск решает этот экземпляр без всяких проблем, последовательно удаляя узлы, пока не удалит все. Но хотя алгоритм Метрополиса начинает работать именно так, при приближении к глобальному максимуму он начинает буксовать. Рассмотрим ситуацию, в которой текущее решение содержит только с узлов, где с намного меньше общего количества узлов n. С очень высокой вероятностью соседнее решение, сгенерированное алгоритмом Метрополиса, будет иметь размер с + 1, а не с – 1, и с разумной вероятностью это перемещение вверх будет принято. Таким образом, по мере работы алгоритма сокращать размер вершинного покрытия становится все труднее и труднее; при приближении к дну воронки начинаются «метания».

Это поведение также проявляется и в более сложных ситуациях, и не столь очевидными способами; конечно, странно видеть его в таком простом случае. Чтобы понять, как справиться с проблемой, мы вернемся к физической аналогии, заложенной в основу алгоритма Метрополиса, и спросим: какой смысл имеет параметр температуры в контексте оптимизации? Т можно рассматривать как одномерную «рукоятку», поворот которой управляет готовностью алгоритма принимать повышающие перемещения. При очень больших Т вероятность принять повышающее перемещение стремится к 1, и алгоритм Метрополиса ведет себя как случайное блуждание, фактически не учитывающее функцию стоимости. С другой стороны, при приближении Т к 0 повышающие перемещения почти никогда не принимаются и алгоритм Метрополиса ведет себя практически идентично градиентному спуску.

Имитация отжига

Ни одна из крайних температур — ни очень низкая, ни очень высокая — не является эффективным методом решения задач минимизации в целом. Этот принцип также проявляется в физических системах: если взять твердое тело и нагреть его до очень высокой температуры, трудно ожидать сохранения стройной кристаллической структуры, даже если она предпочтительна с энергетической точки зрения; и это можно объяснить большим значением kT в выражении $e^{-E(S)/(kT)}$, с которым огромное количество менее выгодных состояний становится слишком вероятным. С этой же точки зрения можно рассматривать «метания» алгоритма Метрополиса в простом экземпляре задачи о вершинном покрытии: он пытается найти самое низкое энергетическое состояние при слишком высокой температуре, когда все конкурирующие состояния имеют слишком высокую вероятность. С другой стороны, если взять расплав и очень быстро заморозить его, не стоит также рассчитывать на формирование идеальной кристаллической структуры, скорее вы получите деформированную структуру с множеством дефектов. Дело в том, что при очень малых T мы слишком близко подходим к области градиентного спуска, и система захватывается в одном из многочисленных провалов своей гребенчатой энергетической поверхности. Интересно заметить, что при очень малых T утверждение (1.2) показывает, что в пределе большая часть времени случайных блужданий проводится в низшем энергетическом состоянии. Проблема в том, что случайное блуждание тратит слишком много времени, чтобы хотя бы отдаленно приблизиться к этому пределу.

Имитация отжига

В начале 1980-х годов ученые изучали связи между минимизацией энергии и комбинаторной оптимизацией. Киркпатрик, Гелатт и Веччи (Kirkpatrick, Gelatt, Vecchi, 1983) размышляли над обсуждаемой темой и задали следующий вопрос: как решить эту задачу для физических систем и какой тип алгоритма предполагает такое решение? В физических системах для перевода материала в кристаллическое состояние используется процесс, называемый отжигом: материал постепенно охлаждается с очень высокой температуры, что дает ему достаточно времени для достижения равновесия на промежуточных убывающих температурах. Таким образом удается избежать энергетических минимумов, с которыми материал сталкивается на всем протяжении процесса охлаждения, и в конечном итоге достичь глобального оптимума.

Этот процесс можно попытаться смоделировать на алгоритмическом уровне; так появился алгоритмический метод, известный как имитация отжига. Метод имитации отжига основан на выполнении алгоритма Метрополиса с постепенным снижением значения T в ходе выполнения. Конкретный способ обновления T называется по естественным причинам планом охлаждения; при его планировании учитывается целый ряд факторов. Формально план охлаждения представляет собой функцию т, отображающую {1, 2, 3, ...} на положительные вещественные числа; на итерации i алгоритма Метрополиса в определении вероятности используется температура T = т(i).

Имитация отжига

На качественном уровне очевидно, что имитация отжига допускает большие изменения в решении на ранних стадиях его выполнения, при высокой температуре. Затем в процессе поиска температура понижается, чтобы снизить вероятность отмены уже происшедших изменений. Имитация отжига также может рассматриваться как попытка оптимизации альтернативы, следующей из (1.2). Согласно (1.2), значения T, сколь угодно близкие к 0, обеспечивают наивысшую вероятность решений с минимальной стоимостью; однако (1.2) само по себе ничего не говорит о скорости сходимости используемых ей функций $f_S(t)$. Как выясняется, эти функции сходятся намного быстрее для больших значений T; чтобы быстро найти решения с минимальной стоимостью, полезно ускорить сходимость, начав процесс при больших T, с последующим сокращением его для повышения вероятности оптимальных решений. Хотя, насколько нам известно, физические системы достигают минимального энергетического состояния через процесс отжига, метод имитации отжига не предоставляет гарантий нахождения оптимального решения. Чтобы понять, почему это так, рассмотрим ситуацию с двойной воронкой. Если две воронки занимают равную площадь, то при высоких температурах система с равной вероятностью может оказаться в любой из двух. При снижении температуры переходы между двумя воронками постоянно усложняются. Нет никаких гарантий, что в конце процесса система окажется на дне более глубокой воронки.

В области имитации отжига остается много нерешенных задач, связанных как с доказательством свойств ее поведения, так и с определением диапазона настроек, в котором этот метод хорошо работает на практике.

Применение локального поиска в нейронных сетях Хопфилда

До настоящего момента мы рассматривали локальный поиск как метод поиска глобального оптимума в вычислительной задаче. Однако в отдельных случаях тщательный анализ спецификации задачи показывает, что в действительности требуется найти произвольный локальный оптимум. Следующая задача дает пример такой ситуации.

Задача

В этом разделе рассматривается задача поиска устойчивых конфигураций в нейронных сетях Хопфилда — простой модели ассоциативной памяти, в которой большое количество элементов объединяется в сеть, а соседние элементы пытаются согласовывать свои состояния. Конкретно сеть Хопфилда может рассматриваться как ненаправленный граф G = (V, E) с целочисленным весом для каждого ребра e; каждый вес может быть как положительным, так и отрицательным. Конфигурация сети S определяется как присваивание значения −1 или +1 каждому узлу u; это значение будет называться состоянием s_u узла u. Смысл конфигурации заключается в том, что каждый узел, представляющий элемент нейронной сети, пытается выбрать одно из двух возможных состояний («да» или «нет»; «истина» или «ложь»), и этот выбор зависит от состояния соседей. Каждое ребро сети устанавливает требование к своим конечным точкам: если u соединяется с v ребром отрицательного веса, то u и v находятся в одинаковом состоянии, а если u соединяется с v ребром положительного веса, то u и v стремятся находиться в противоположных состояниях. Абсолютное значение $|w_e|$ обозначает силу требования; мы будем называть $|w_e|$ абсолютным весом ребра e.

К сожалению, в системе может не быть единой конфигурации, соблюдающей требования всех ребер. Для примера рассмотрим три узла a, b и c, взаимно соединенных ребрами с весом 1. Какую бы конфигурацию мы ни выбрали, два ребра будут находиться в одинаковом состоянии, нарушая требование о противоположности состояний.

Задача

С учетом этого обстоятельства мы немного снизим уровень требований. В отношении заданной конфигурации ребро e = (u, v) называется хорошим, если устанавливаемое им требование соблюдается состояниями его конечных точек: либо $w_e < 0$ и $s_u = s_v$, либо $w_e > 0$ и $s_u \neq s_v$. В противном случае ребро e называется плохим. Заметим, что условие «ребро e является хорошим» очень компактно выражается в следующем виде: $w_e s_u s_v < 0$. Узел u в заданной конфигурации называется реализованным, если общий абсолютный вес всех хороших ребер, инцидентных u, по крайней мере не меньше общего абсолютного веса всех плохих ребер, инцидентных u. Это определение можно записать в виде

Наконец, конфигурация называется устойчивой, если все узлы в ней реализованы.

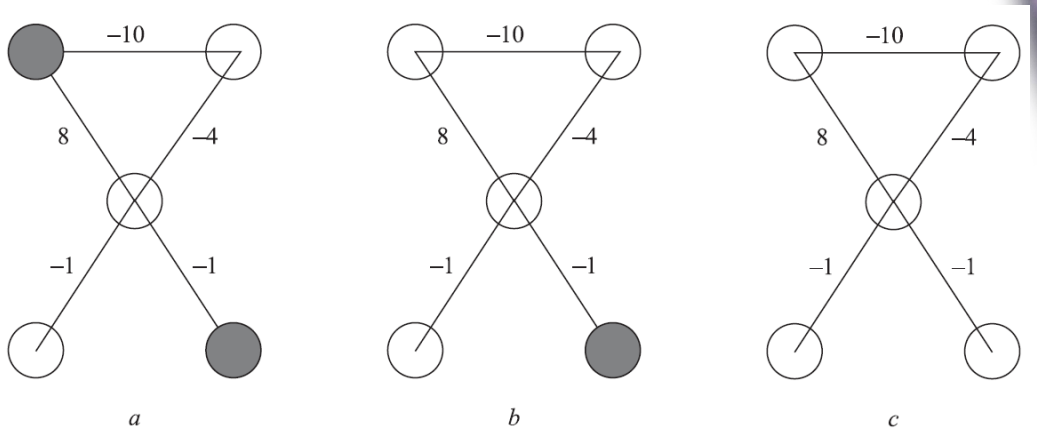
Почему такие конфигурации называются «устойчивыми»? Термин происходит из рассмотрения сети с точки зрения отдельного узла u. Сам по себе узел u имеет выбор только между состояниями −1 и +1; как и все узлы, он хочет учесть как можно больше требований ребер (в соответствии с метрикой абсолютного веса). Допустим, узел u спрашивает: следует ли мне перейти в противоположное состояние? Мы видим, что если u изменяет состояние (при сохранении состояний всех остальных узлов), то все хорошие ребра, инцидентные u, становятся плохими, а все плохие ребра, инцидентные u, становятся хорошими. Итак, чтобы максимизировать вес хороших ребер, находящихся под его прямым управлением, узел u должен перейти в обратное состояние в том, и только в том случае, если он не реализован. Другими словами, устойчивой конфигурацией является та, в которой нет отдельных узлов, для которых были бы причины перейти в противоположное состояние. А теперь мы приходим к основному вопросу: всегда ли в сети Хопфилда существует устойчивая конфигурация, и если да, то как ее найти?

Разработка алгоритма

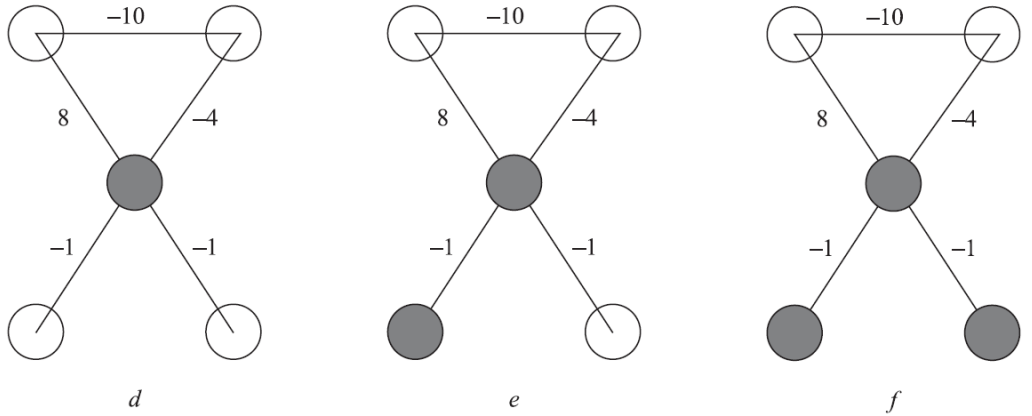
Алгоритм, который будет разработан в этом разделе, доказывает следующий результат.
Утверждение 1.3. У каждой сети Хопфилда имеется устойчивая конфигурация, которая может быть найдена за время, полиномиальное по n и $W = \sum_e |w_e|$.
Вы увидите, что устойчивые конфигурации крайне естественно встречаются в качестве локальных оптимумов определенных процедур локального поиска в сетях Хопфилда. Чтобы убедиться в том, что утверждение (1.3) не так уж тривиально, стоит заметить, что оно перестает быть истинным при некоторых естественных модификациях модели. Например, представьте, что мы определяем направленную сеть Хопфилда точно так, как описано выше, но с одним исключением — каждое ребро является направленным, а каждый узел определяет, является он реализованным или нет, проверяя только те ребра, для которых он является начальным. В этом случае сеть может и не иметь устойчивой конфигурации. Для примера возьмем направленную версию сети из трех узлов, упоминавшуюся ранее: имеются три узла a, b, c, соединенные направленными ребрами (a, b), (b, c), (c, a), все ребра имеют вес 1. Если все узлы находятся в одном состоянии, все они будут нереализованными; а если состояние одного узла отличается от состояния двух других, то узел, находящийся непосредственно перед ним, будет нереализованным. Следовательно, в этой направленной сети не существует конфигурации, в которой все узлы были бы реализованы.

Разработка алгоритма

Очевидно, доказательство (1.3) должно где-то зависеть от ненаправленной природы сети.
Чтобы доказать (1.3), мы проанализируем простую итеративную процедуру поиска устойчивой конфигурации, которую назовем алгоритмом переключения состояния.
Пока текущая конфигурация не является устойчивой
    Должен существовать нереализованный узел
    Выбрать нереализованный узел u
    Переключить состояние u
Конец Пока



Последовательность выполнения алгоритма переключения состояния для сети Хопфилда из пяти узлов, приводящая к устойчивой конфигурации. (Состояние узлов обозначается черным или белым цветом)



Анализ алгоритма

Очевидно, если только что определенный нами алгоритм переключения состояния завершился, была достигнута устойчивая конфигурация. Впрочем, не так очевидно то, что он действительно завершится. В предыдущем примере с направленным графом этот процесс будет просто перебирать три узла, бесконечно переключая их состояния. Сейчас мы докажем, что алгоритм переключения состояния всегда завершается, и приведем границу для количества итераций, необходимых для завершения. Тем самым будет доказано утверждение (1.3). Ключом к доказательству того, что этот процесс завершается, служит идея, использованная в нескольких предыдущих ситуациях: нужно найти метрику прогресса, то есть величину, которая строго увеличивается с каждым переключением состояния и имеет абсолютную верхнюю границу. Она может быть использована для ограничения количества итераций. Вероятно, самой естественной метрикой прогресса могло бы стать количество реализованных узлов: если оно увеличивается при каждом переключении нереализованного узла, процесс будет выполняться не более n итераций перед завершением в устойчивой конфигурации. К сожалению, эта метрика не подходит. Действительно, при переключении нереализованного узла v узел становится реализованным, но некоторые из ранее реализованных соседей могут стать нереализованными, что приведет к общему снижению количества реализованных узлов. В действительности это происходит в одной из итераций на рисунке: когда средний узел изменяет состояние, оба его (ранее реализованных) нижних соседа становятся нереализованными.

Анализ алгоритма

Также факт завершения невозможно доказать тем аргументом, что каждый узел изменяет состояние не более одного раза в ходе выполнения алгоритма: снова обратившись к примеру на рисунке, мы видим, что узел в правом нижнем углу изменяет состояние дважды.

Однако существует менее очевидная метрика прогресса, которая изменяется с каждым изменением состояния нереализованного узла. А именно: для заданной конфигурации S мы определяем Φ(S) как общий абсолютный вес всех хороших ребер в сети, то есть

$$\Phi(S) = \sum_{good\ e} |w_e|.$$

Очевидно, для любой конфигурации S выполняется Φ(S) ≥ 0 (так как Φ(S) — (так как в крайнем

сумма положительных целых чисел) и $\Phi(S) \leq W = \sum_e |w_e|$ случае каждое ребро является хорошим).

Теперь предположим, что в неустойчивой конфигурации S мы выбираем узел u, который является нереализованным, и изменяем его состояние, получая конфигурацию S'. Что можно сказать об отношениях между Φ(S') и Φ(S)? Вспомним, что при переключении состояния u все хорошие ребра, инцидентные u, становятся плохими; все плохие ребра, инцидентные u, становятся хорошими; а все ребра, для которых u не является конечной точкой, остаются без изменений. Если обозначить g_u и b_u общий абсолютный вес соответственно хороших и плохих ребер, инцидентных u, то имеем

$$\Phi(S') = \Phi(S) - g_u + b_u.$$

Анализ алгоритма

Но так как узел u был нереализованным в S, мы также знаем, что $b_u > g_u$; а поскольку и b_u, и g_u являются целыми числами, имеем $b_u \geq g_u + 1$. Следовательно,

$$\Phi(S') \geq \Phi(S) + 1.$$

Значение Ф начинается с некоторого неотрицательного целого числа, увеличивается на 1 с каждым изменением состояния и не может превысить W. Следовательно, процесс выполняется не более W итераций, и при его завершении мы должны иметь устойчивую конфигурацию. Кроме того, при каждой итерации нереализованный узел выявляется с использованием полиномиального по n количества арифметических итераций; из этого также следует, что граница времени выполнения полиномиальна по n и W.

Итак, сутью доказательства существования устойчивых конфигураций в конечном итоге оказывается локальный поиск. Сначала мы определили целевую функцию Ф, которую требуется максимизировать. Конфигурации были возможными решениями этой задачи максимизации, и мы определили, какие две конфигурации S и S' должны считаться соседними: S' должна получаться из S переключением одного состояния.

Анализ алгоритма

Затем мы изучили поведение простого итеративного алгоритма локального поиска («перевернутую» форму градиентного спуска, так как в данном случае используется задача максимизации); при этом было обнаружено следующее:

Утверждение 1.4. Любой локальный максимум в алгоритме переключения состояния, максимизирующий Ф, является устойчивой конфигурацией.

Следует заметить, что хотя наш алгоритм доказывает существование устойчивой конфигурации, время выполнения оставляет желать лучшего при больших абсолютных весах. А именно: по аналогии с тем, что мы видели в задаче о сумме подмножеств и в первом алгоритме максимального потока, полученный здесь алгоритм полиномиален только по фактической величине весов, а не по размеру их двоичных представлений. Для очень больших весов время выполнения может стать неприемлемым.

В настоящее время простые обходные решения неизвестны. Вопрос о существовании алгоритма, строящего устойчивые состояния за время, полиномиальное по n и log W (вместо n и W), или с количеством примитивных арифметических операций, полиномиальным только по n (независимо от W), остается открытым.

Аппроксимация задачи о максимальном разрезе с применением локального поиска

Теперь обсудим ситуацию, в которой алгоритм локального поиска применятся для нахождения доказуемой гарантии аппроксимации для задачи оптимизации. Для этого мы проанализируем структуру локальных оптимумов и установим границу качества локально оптимальных решений относительно глобального оптимума. Рассматриваемая задача о максимальном разрезе тесно связана с задачей поиска устойчивых конфигураций для сетей Хопфилда.

Задача

В задаче о максимальном разрезе имеется ненаправленный граф G = (V, E) с положительными целочисленными весами каждого ребра e. Для разбиения (A, B) множества вершин w(A, B) обозначает общий вес ребер, один конец которых принадлежит A, а другой принадлежит B:

$$w(A,B) = \sum_{\substack{e=(u,v) \\ u\in A, v\in B}} w_e .$$

Целью является поиск разбиения (A, B) вершинного покрытия, максимизирующего w(A, B). Задача о максимальном разрезе является NP-сложной в том смысле, что для взвешенного графа G и границы β принятие решения о том, существует ли разбиение (A, B) вершин G с w(A, B) ≥ β, выполняется с NP-сложностью.

И хотя задача нахождения устойчивой конфигурации сети Хопфилда не была оптимизационной задачей как таковой, что задача о максимальном разрезе с ней тесно связана. На языке сетей Хопфилда задача о максимальном разрезе представляет собой экземпляр, в котором все веса ребер положительны (а не отрицательны), а конфигурации состояний узлов S естественно соответствуют разбиениям (A, B): узлы имеют состояние –1 в том, и только в том случае, если они принадлежат множеству A, и состояние +1 — в том, и только в том случае, если они принадлежат множеству B. Целью является такое распределение состояний, при котором как можно большая часть веса приходится на хорошие ребра — те, у которых конечные точки находятся в разных состояниях. В такой формулировке задача о максимальном разрезе направлена на максимизацию величины Ф(S).

Разработка алгоритма

Алгоритм переключения состояния, использованный для сетей Хопфилда, предоставляет алгоритм локального поиска для аппроксимации целевой функции максимального разреза Ф(S) = w(A, B). В контексте разбиений он говорит следующее: если существует узел u, для которого суммарный вес ребер из u в узлы на соответствующей стороне разбиения превышает общий вес ребер из u в узлы на другой стороне разбиения, то узел u следует переместить на другую сторону разбиения. Введем понятие «ближнего соседства»: разбиения (A, B) и (A', B') называются ближними соседними решениями, если (A', B') можно получить из (A, B) перемещением одного узла с одной стороны разбиения на другую. Естественно задать два основных вопроса:

- Можно ли сказать что-то конкретное о качестве локальных оптимумов в окружении ближнего соседства?
- Так как ближнее соседство определяется предельно просто, какие варианты соседства могут предоставить более сильные алгоритмы локального поиска для максимального разреза?

Анализ алгоритма

Следующий результат отвечает на первый вопрос, показывая, что локальные оптимумы в ближнем соседстве предоставляют решения, соответствующие гарантированной границе аппроксимации.

Утверждение 1.5. Пусть (A, B) — разбиение, которое является локальным оптимумом для задачи о максимальном разделе в ближайшем соседстве, а (A*, B*) — глобально-оптимальное разбиение. Тогда w(A, B) ≥ 0.5 w (A*, B*).

Доказательство. Пусть $W = \sum_e w_e$. Мы также немного расширим систему обозначений: для двух узлов u и v запись w_{uv} будет обозначать w_e, если существует ребро e, соединяющее u и v, и 0 в противном случае.

Для любого узла u ∈ A должно выполняться условие

$$\sum_{u \in A} w_{uv} = \sum_{v \in B} w_{uv},$$

так как в противном случае узел u должен быть перемещен на другую сторону разбиения, и разбиение (A, B) не будет локально оптимальным. Просуммируем эти неравенства для всех u ∈ A; любое ребро, оба конца которого принадлежат A, будет находиться в левой части ровно двух таких неравенств, тогда как любое ребро, один конец которого принадлежит A, а другой принадлежит B, будет находиться в правой части ровно одного из таких неравенств.

Анализ алгоритма

Следовательно, мы получаем

$$2\sum_{\{u,v\}\subseteq A} w_{uv} \leq \sum_{u\in A, v\in B} w_{uv} = w(A,B). (1.1)$$

Аналогичные рассуждения можно применить к множеству B, получая

$$2\sum_{\{u,v\}\subseteq B} w_{uv} \leq \sum_{u\in A, v\in B} w_{uv} = w(A,B). (1.2)$$

Сложив неравенства (1.1) и (1.2) и разделив результат на 2, получаем:

$$\sum_{\{u,v\}\subseteq A} w_{uv} + \sum_{\{u,v\}\subseteq B} w_{uv} \leq w(A,B). (1.3)$$

В левой стороне неравенства (1.3) учитываются веса всех ребер, не переходящих из A в B; следовательно, если прибавить w(A, B) к обеим сторонам (1.3), левая сторона станет равна W. Правая сторона превращается в 2w(A, B), поэтому имеем W ≤ 2w(A, B), или w(A, B) ≥ 0.5 W. ■

Обратите внимание: в доказательстве (1.5) мы особенно не задерживались на оптимальном разбиении (A*, B*); в действительности было доказано более сильное утверждение о том, что в любом локально оптимальном решении в ближнем соседстве по крайней мере половина общего веса ребер в графе пересекает разбиение.

Анализ алгоритма

Утверждение (1.5) доказывает, что локальный оптимум представляет собой 2-аппроксимацию максимального разреза. Это наводит на мысль, что локальная оптимизация может быть хорошим алгоритмом для приближенной максимизации значения разреза. Тем не менее нужно рассмотреть еще одно обстоятельство: время выполнения. Алгоритм переключения состояния только псевдополиномиальный, и вопрос о том, возможно ли найти локальный оптимум за полиномиальное время, остается открытым. Однако в данном случае мы можем добиться практически такого же результата, просто остановив алгоритм при отсутствии «достаточно значительных» улучшений.

Пусть (A, B) — разбиение с весом w(A, B). Для фиксированного ε > 0 переключение одного узла будет называться значительным, если оно улучшает значение разреза минимум на $\frac{2\varepsilon}{n} w(A, B)$, где n = |V|. Теперь рассмотрим версию алгоритма переключения состояния, которая принимает только значительные переключения и завершается при отсутствии таких переключений, даже если текущее разбиение не является локальным оптимумом. Утверждается, что эта версия алгоритма приведет почти к не худшей аппроксимации и выполняется за полиномиальное время. Прежде всего можно расширить предыдущее доказательство и продемонстрировать, что полученный разрез почти не хуже. Достаточно добавить $\frac{2\varepsilon}{n} w(A, B)$ к каждому неравенству, так как нам известно лишь об отсутствии значительных переключений.

Анализ алгоритма

Утверждение 1.6. Пусть (A, B) — разбиение, для которого невозможно значительное переключение, а (A*, B*) — глобально-оптимальное разбиение. В этом случае (2 + ε) w(A, B) ≥ w(A*, B*).

Теперь обратимся ко времени выполнения.

Утверждение 1.7. Версия алгоритма переключения состояния, принимающая только значительные переключения, завершается максимум после $O(\varepsilon^{-1} n \log W)$ переключений (в предположении, что веса целочисленны, а $W = \sum_e w_e$.

Доказательство. Каждое переключение улучшает целевую функцию минимум на множитель $(1 + \varepsilon/n)$. Так как $(1 + 1/x)^x \geq 2$ для любого $x \geq 1$, мы видим, что $(1 + \varepsilon/n)^{n/\varepsilon} \geq 2$, а следовательно, целевая функция возрастает не менее чем вдвое каждые n/ε переключений. Вес не может превысить W, а следовательно, он может удваиваться не более log W раз. ■

Выбор соседского отношения

Алгоритмы локального поиска в действительности базируются на двух основных ингредиентах: выборе соседского отношения и правиле выбора соседнего решения на каждом шаге. Какие аспекты следует учитывать при выборе соседского отношения? Этот выбор может быть весьма непростым, хотя на высоком уровне альтернатива описывается достаточно тривиально.

(i) Соседское окружение решения должно быть достаточно широким, чтобы алгоритм не застревал в плохих локальных оптимумах.

(ii) Соседское окружение решения не должно быть слишком большим, потому что мы хотим иметь возможность эффективно искать в множестве соседей возможные локальные перемещения.

Если бы первый пункт был единственной проблемой, то все решения можно было бы просто сделать соседями друг друга — тогда локальных оптимумов вообще не будет, а глобальный оптимум всегда будет находиться в одном шаге! Второй пункт выявляет (очевидный) недостаток такого подхода: если бы соседское окружение текущего решения состояло из всех возможных решений, то парадигма локального поиска вообще не приносила никакой пользы и сводилась бы к простому перебору соседского окружения методом «грубой силы».

Вообще говоря, мы уже встречали один случай, в котором выбор правильного соседского отношения оказывал огромное влияние на разрешимость задачи, хотя этот факт тогда явно не отмечался: речь идет о задаче о двудольном паросочетании.

Выбор соседского отношения

Вероятно, простейшее соседское отношение для паросочетаний выглядит так: M' является соседом M, если M' может быть получено вставкой или удалением одного ребра в M. В соответствии с этим определением мы получаем «поверхности» с множеством зубцов, как и в примерах вершинного покрытия, которые приводились выше; и по этому определению можно получить локально оптимальные паросочетания, имеющие только половину размера максимального сочетания.

Но предположим, мы попытаемся определить более сложное (и даже асимметричное) соседское отношение: M' является соседом M, если при создании соответствующей потоковой сети M' может быть получено из M одним увеличивающим путем. Что можно сказать о паросочетании M, если оно является локальным максимумом с этим соседским отношением? В этом случае увеличивающего пути не существует, поэтому M в действительности должно быть (глобальным) максимальным паросочетанием. Другими словами, с таким соседским отношением единственными локальными максимумами являются глобальные максимумы, так что прямой градиентный подъем приведет к максимальному паросочетанию. Если поразмыслить над тем, что делает алгоритм Форда–Фалкерсона в нашем сведении от двудольного паросочетания к максимальному потоку, это выглядит логично: размер паросочетания строго возрастает на каждом шаге, и нам никогда не приходится «отступать» из локального максимума. Следовательно, тщательно выбирая соседское отношение, мы превратили зазубренную поверхность оптимизации в простую воронку.

Алгоритмы локального поиска при разбиении графов

Перейдем к соседским отношениям, порождающим соседские окружения большего размера по сравнению с правилом одного переключения и соответственно пытающимся сократить распространенность локальных оптимумов. Возможно, самым естественным обобщением является соседское окружение с k-переключением для $k \geq 1$: разбиения (A, B) и (A', B') называются соседними по правилу k-переключения, если (A', B') можно получить из (A, B) перемещением не более k узлов с одной стороны разбиения на другую.

Очевидно, если (A, B) и (A', B') являются соседями по правилу k-переключения, то они также являются соседями по правилу k'-переключения для всех $k' > k$. Следовательно, если разбиение (A, B) является локальным оптимумом по правилу k'-переключения, то оно также является локальным оптимумом по правилу k-переключения для всех $k < k'$. Но сокращение множества локальных оптимумов посредством повышения величины k обходится дорого: для просмотра множества соседей (A, B) по правилу k-переключения необходимо рассмотреть все $\Theta(nk)$ способов перемещения до k узлов на другую сторону разбиения. Затраты времени становятся неприемлемыми даже при небольших значениях k.

Керниган и Лин (1970) предложили альтернативный способ генерирования соседних решений; он обладает существенно большей вычислительной эффективностью, но все при этом позволяет выполнять крупномасштабные преобразования решений за один шаг. Их метод, который мы будем называть эвристикой К-Л, определяет соседей разбиения (A, B) по следующей n-фазной процедуре.

Алгоритмы локального поиска при разбиении графов

- В фазе 1 выбирается один узел для переключения — так, чтобы значение полученного решения было как можно больше. Переключение выполняется даже в том случае, если значение решения убывает относительно w(A, B). Узел, изменивший состояние, помечается, а полученное решение обозначается (A_1, B_1).
- В начале фазы k для k > 1 мы имеем разбиение (A_{k-1}, B_{k-1}), и k – 1 узлов помечены. Один непомеченный узел выбирается для переключения таким образом, что значение полученного решения является максимальным среди всех возможных. (И снова это происходит даже в том случае, если в результате значение решения уменьшится.) Переключенный узел помечается, а полученное решение обозначается (A_k, B_k).

После n фаз каждый узел окажется помеченным; это указывает на то, что он переключался ровно один раз. Соответственно последнее разбиение (A_n, B_n) в действительности является зеркальным отображением исходного разбиения (A, B): A_n = B и B_n = A. Наконец, эвристика К-Л определяет n – 1 разбиений (A_1, B_1), ..., (A_{n-1}, B_{n-1}) как соседей (A, B). Следовательно, (A, B) является локальным оптимумом по эвристике К-Л в том, и только в том случае, если $w(A, B) \geq w(A_i, B_i)$ для $1 \leq i \leq n - 1$. Итак, мы видим, что эвристика К-Л определяет очень длинную серию переключений, даже если ситуация на первый взгляд ухудшается, в надежде на то, что некоторое разбиение (A_i, B_i), сгенерированное по пути, окажется лучше (A, B). Но даже при том, что она генерирует соседей, очень отличных от (A, B), выполняется только n переключений и на каждое тратится время всего O(n).

Классификация на базе локального поиска

Рассмотрим более сложный пример применения локального поиска при разработке аппроксимирующих алгоритмов, связанный с задачей о сегментации изображения, которая рассматривалась среди практических применений сетевых потоков. Более сложная версия сегментации изображения, которая будет рассматриваться здесь, дает пример того, как для получения хорошего быстродействия алгоритма локального поиска приходится использовать довольно сложную структуру соседского окружения для множества решений. Как вы увидите, естественное соседское окружение «переключения состояния», описанное в предыдущем разделе, может приводить к очень плохим локальным оптимумам. Для получения хорошего быстродействия мы воспользуемся экспоненциально большим соседским окружением. Одна из проблем больших соседских окружений заключается в том, что последовательный поиск по всем соседям текущего решения для его улучшения становится неприемлемым. Потребуется более сложный алгоритм поиска улучшенного соседа (если он существует).

Задача

Вспомним базовую задачу сегментации изображения. Тогда задача сегментации изображения была сформулирована как задача разметки; наша цель заключалась в пометке (то есть классификации) каждого пиксела как принадлежащего к переднему плану или фону изображения. На тот момент было понятно, что это очень простая формулировка задачи и было бы неплохо иметь заняться более сложными задачами разметки, например сегментацией областей изображения в зависимости от их расстояния от камеры. По этой причине мы рассмотрим задачу разметки с более чем двумя метками. Попутно мы создадим инфраструктуру классификации, возможности применения которой не ограничиваются пикселами изображения. При определении задачи сегментации с двумя метками «передний план/ фон» мы в конечном итоге пришли к следующей формулировке. Имеется граф G = (V, E), в котором V соответствует пикселам изображения, а цель заключается в классификации каждого узла V по двум возможным классам: фону и переднему плану. Ребра представляют пары узлов, которые с большой вероятностью принадлежат одному классу (например, потому что они расположены вблизи друг от друга), и для каждого ребра (i, j) задан штраф $p_{ij} \geq 0$ за размещение i и j в разных классах. Кроме того, имеется информация о вероятности того, принадлежит узел или пиксел переднему плану или фону. Эти вероятности были преобразованы в штрафы за отнесение узла к классу, которому он принадлежит с меньшей вероятностью. Задача заключалась в нахождении разметки узлов, минимизировавшей общие штрафы за разделение и отнесение к классу. Мы показали, что задача минимизации решается посредством вычисления минимального разреза. В оставшейся части раздела эта формулировка задачи будет называться бинарной сегментацией изображения.

Задача

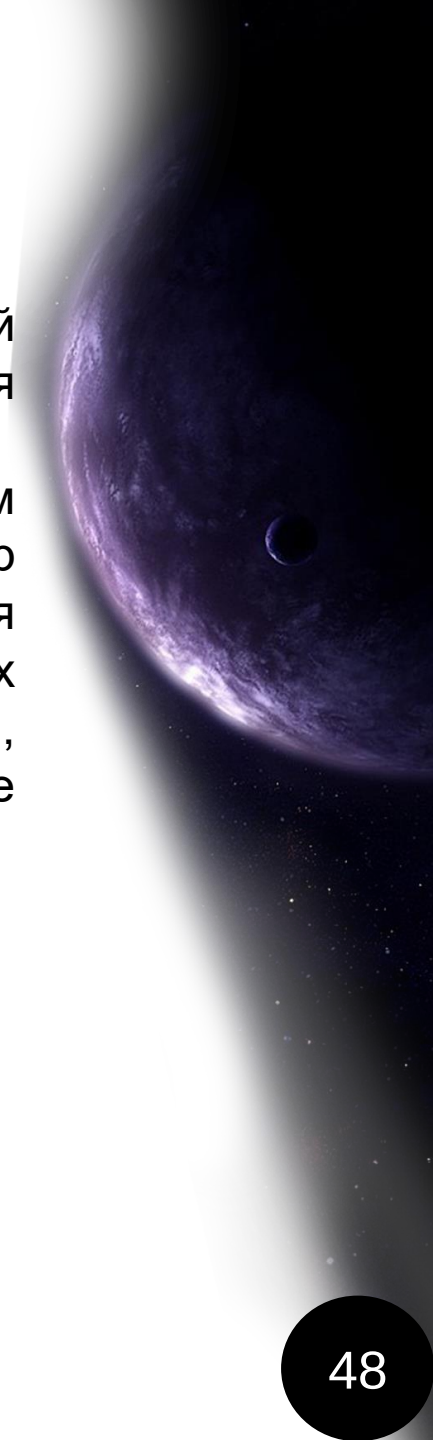
Сейчас будет сформулирована аналогичная задача классификации/разметки с более чем двумя классами (метками). Как выясняется, эта задача является NPсложной, и мы разработаем алгоритм локального поиска, в котором локальные оптимумы являются 2-аппроксимациями для лучшей разметки. Обобщенная задача разметки, рассматриваемая в этом разделе, формулируется следующим образом. Имеются граф G = (V, E) и множество L из k меток. Целью является пометка каждого узла в V одной из меток из множества L с целью минимизации некоторого штрафа. На выбор оптимальной разметки влияют два конкурирующих фактора. Для каждого ребра (i, j) ∈ E действует штраф за разделение $p_{ij} \geq 0$ за пометку двух узлов i и j разными метками. Кроме того, узлы с большей вероятностью имеют одни метки вместо других. Это условие выражается штрафом за назначение. Для каждого узла i ∈ V и каждой метки a ∈ L действует неотрицательный штраф $c_i(a) \geq 0$ за назначение метки a узлу i. (Эти штрафы являются аналогами вероятностей из задачи бинарной сегментации, не считая того, что здесь они рассматриваются как минимизируемые затраты.) Задача разметки заключается в нахождении разметки f: V → L, минимизирующей общий штраф:

$$\Phi(f) = \sum_i c_i\big(f(i)\big) + \sum_{(i,j)\in E: f(i)\neq f(j)} p_{ij}$$

Задача

Заметим, что задача разметки только с двумя метками в точности совпадает с задачей сегментации изображений. Для трех меток задача разметки уже является NP-сложной, хотя этот факт мы доказывать не будем.

Наша цель — разработать алгоритм локального поиска для этой задачи, в котором локальные оптимумы являются хорошими аппроксимациями оптимального решения. Пример также наглядно покажет, как важно выбирать хорошее соседское окружение для определения алгоритма локального поиска. Существует много возможных вариантов для соседских отношений, и вы увидите, что некоторые из них работают намного лучше других. В частности, для получения гарантий аппроксимации будет использовано довольно сложное определение соседского окружения.

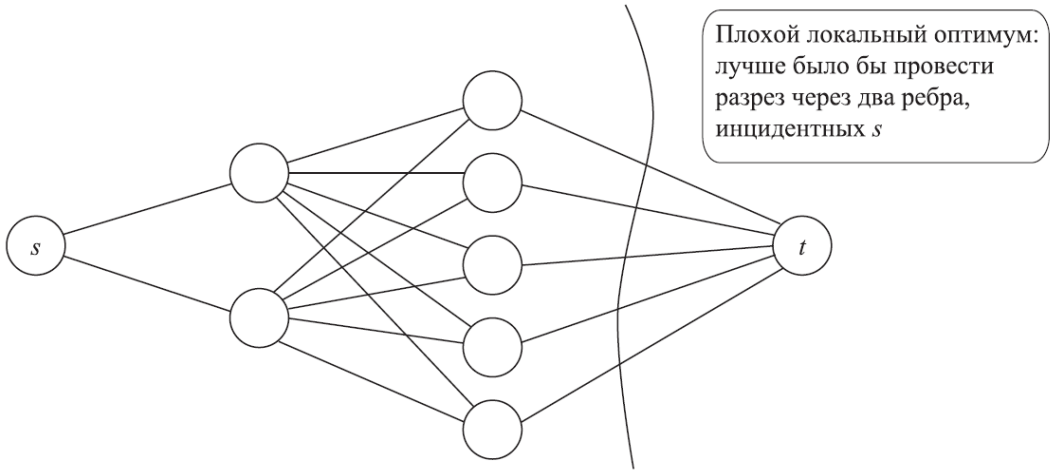


Разработка алгоритма

Первая попытка: правило одного переключения

Простейшим и, наверное, самым естественным выбором соседского отношения является правило одного переключения из алгоритма переключения состояния для задачи о максимальном разрезе: две разметки являются соседними, если одну из них можно получить из другой изменением метки одного узла. К сожалению, это соседство может привести к достаточно слабым локальным оптимумам в нашей задаче, даже всего с двумя метками.

Это выглядит несколько странно, потому что правило хорошо работало в задаче о максимальном разрезе. Однако наша задача связана с задачей о минимальном разрезе. Собственно, задача о минимальном разрезе s–t соответствует частному случаю, в котором используются всего две метки, а s и t — единственные узлы со штрафами назначения. Нетрудно увидеть, что этот алгоритм переключения состояния не является хорошим аппроксимирующим алгоритмом для задачи о минимальном разрезе.



Экземпляр задачи о минимальном разрезе s–t, в котором пропускные способности всех ребер равны 1

Разработка алгоритма

Вторая попытка: рассмотрение двух меток за раз

Здесь мы разработаем алгоритм локального поиска, в котором окружение определяется намного сложнее. Интересная особенность этого алгоритма заключается в том, что он позволяет каждому решению иметь экспоненциальное количество соседей. На первый взгляд это противоречит общему правилу «соседское окружение решения не должно быть слишком большим. Однако здесь мы будем работать с окружением более элегантно. Сохранение малого размера соседского окружения хорошо работает тогда, когда вы планируете проводить поиск улучшающего локального шага методом «грубой силы»; а на этот раз мы воспользуемся вычислением минимального разреза с полиномиальным временем, чтобы определить, встречается ли улучшение среди экспоненциально многочисленных соседей решения.

Идея локального поиска заключается в применении алгоритма с полиномиальным временем для бинарной сегментации изображения с целью улучшения локальных шагов. Начнем с простейшей реализации этой идеи, которая не всегда дает хорошую гарантию аппроксимации. Для разметки f выберем две метки a, b ∈ L и ограничим внимание узлами, имеющими метки a или b в разметке f. За один локальный шаг любое подмножество этих узлов может переключить метки из a в b или из b в a. Или в более формальном определении, две разметки f и f ' считаются соседними, если существуют две метки a, b ∈ L, такие, что для всех остальных меток c ∉ {a, b} и всех узлов i ∈ V условие f (i) = c выполняется в том, и только в том случае, если f '(i) = c. Следует заметить, что состояние f может иметь сколь угодно много соседей, так как произвольное подмножество узлов с метками a и b может переключать свою метку.

Разработка алгоритма

Однако при этом действует следующее утверждение:

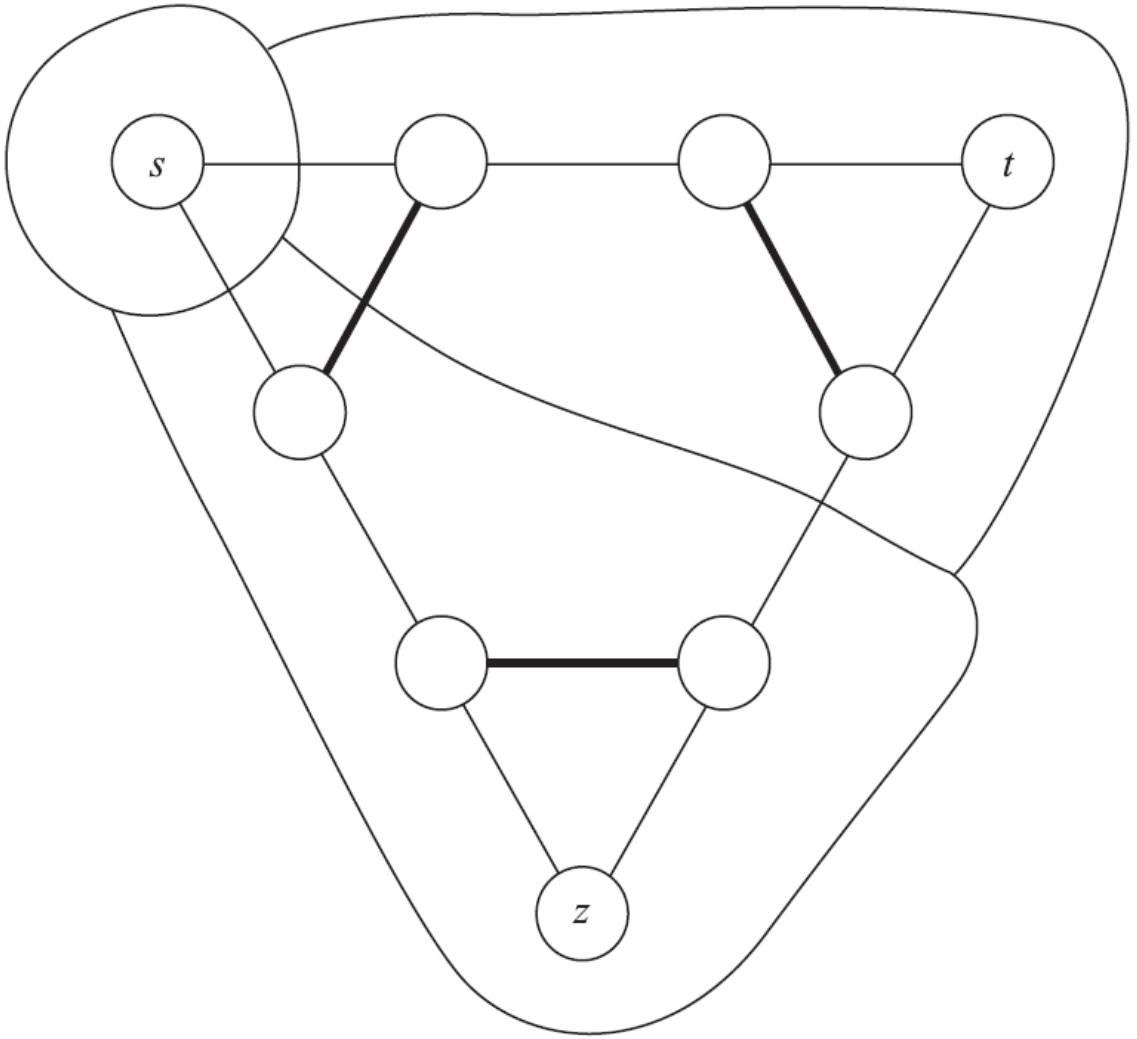
Утверждение 1.8. Если разметка f не является локально оптимальной для описанного выше соседского окружения, то сосед с меньшим штрафом может быть найден за k^2 вычислений минимального разреза.

Доказательство. Количество пар разных меток меньше k^2, поэтому каждую пару можно проверить по отдельности. Для пары меток a, b ∈ L рассмотрим задачу нахождения улучшенной разметки посредством перестановки меток узлов между a и b. Она в точности совпадает с задачей сегментации для двух меток в подграфе узлов, которым в f назначены метки a или b. Воспользуемся алгоритмом, разработанным для бинарной сегментации изображений, для поиска лучшей из таких измененных разметок. ■

Это соседское окружение намного лучше варианта с одним переключением, который был рассмотрен в начале.

Разработка алгоритма

Например, оно предоставляет оптимальное решение случая с двумя метками. Однако даже с улучшенным окружением локальные оптимумы могут быть плохими, как показано на рисунке. В этом примере есть три узла s, t и z, которые должны сохранить свои исходные метки. Все остальные узлы лежат на одной из сторон треугольника; они должны сохранить одну из двух меток, связанных с узлами на концах этой стороны. Эти требования легко выражаются назначением каждому узлу очень большого штрафа за назначение для неразрешенных меток. Штрафы за разделение определяются следующим образом: тонким ребрам на рисунке соответствует штраф 1, а толстым — большой штраф M. Теперь заметим, что разметка на иллюстрации имеет штраф M + 3, но при этом является локально оптимальной. У (глобально) оптимальной разметки штраф равен всего 3, а для ее получения на рисунке достаточно изменить метки обоих узлов рядом с s.



Плохой локальный оптимум для алгоритма локального поиска, учитывающего только две метки

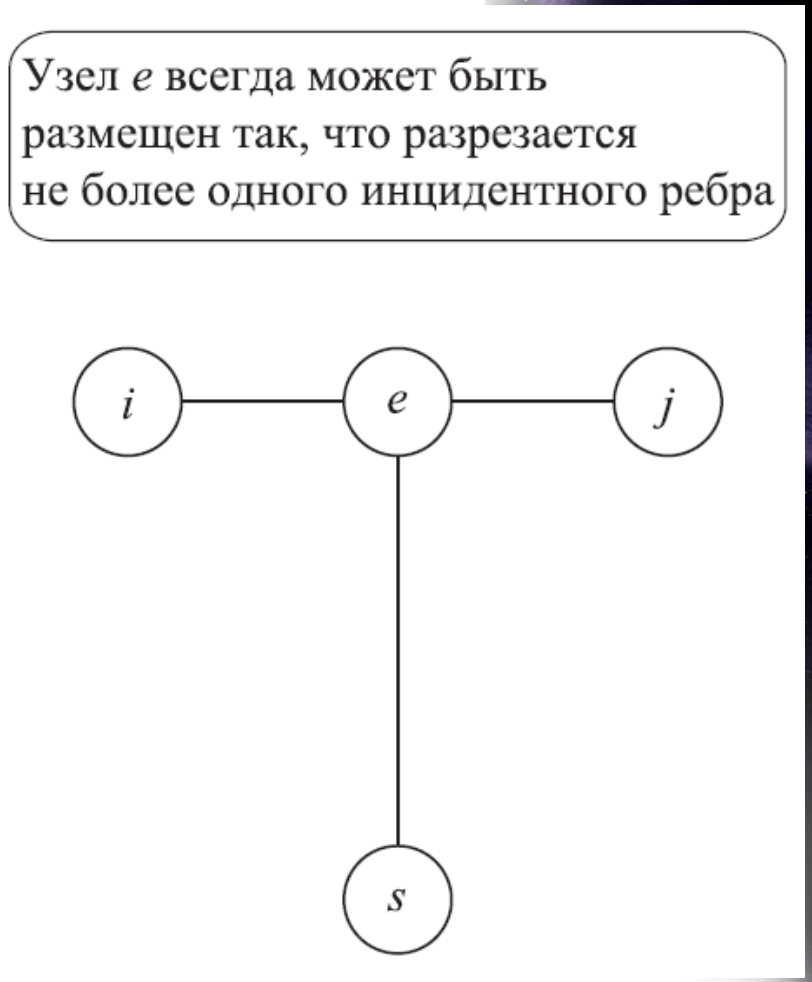
Поиск хорошего соседа

Чтобы найти лучшего соседа, мы проверяем каждую метку по отдельности. Начнем с метки a. Утверждается, что лучшее переназначение, в котором узлы могут изменить свои метки на a, может быть найдено посредством вычисления минимального разреза. Построение графа минимального разреза G' = (V', E') аналогично вычислению минимального разреза, разработанному для бинарной сегментации изображений. Тогда мы определили источник s и сток t для представления двух меток. Здесь мы тоже введем источник и сток, но источник s будет представлять метку a, а сток t будет фактически представлять другой вариант, имеющийся у узлов, а именно сохранение их исходных меток. Идея заключается в том, чтобы найти минимальный разрез в G' и заменить метки всех узлов на стороне s разреза меткой a, тогда как все узлы на стороне t сохранят свои исходные метки.

Для каждого узла G в новом множестве V' имеется соответствующий узел, а в E' добавляются ребра (i, t) и (s, i). Ребро (i, t) имеет пропускную способность $c_i(a)$, так как разрезание ребра (i, t) приводит к размещению узла i на стороне источника, а следовательно, соответствует пометке узла i меткой a. Ребро (i, s) будет иметь пропускную способность $c_i(f(i))$, если $f(i) \neq a$, или она будет выражаться очень большим числом M (или $+\infty$), если $f(i) = a$. Разрезание ребра (i, t) помещает узел i на сторону стока, а следовательно, соответствует сохранению узлом i исходной метки $f(i) \neq a$. Большая пропускная способность M предотвращает размещение узлов i с f (i) = a на стороне стока.

Поиск хорошего соседа

В построении для задачи с двумя метками мы добавляли ребра между узлами V и использовали штрафы за разделение как пропускные способности. Этот способ хорошо работает для узлов, разделенных разрезом, или узлов на стороне источника, одновременно имеющих метку a. Но если i и j находятся на стороне стока, то соединяющее их ребро еще не разрезано, но i и j разделены, если f (i) ≠ f (j). Чтобы решить эту проблему, мы дополним построение G' следующим образом. Для ребра (i, j), если f (i) = f (j) или один из узлов i или j имеет метку a, в E' добавляется ребро (i, j) с пропускной способностью p_{ij}. Для ребер e = (i, j), у которых f (i) ≠ f (j) и ни один узел не имеет метку a, чтобы правильно закодировать через граф G', что i и j остаются разделенными, даже если находятся на стороне стока, придется действовать иначе. Для каждого такого ребра e в V' добавляется дополнительный узел e, соответствующий ребру e, и ребра (i, e), (e, j) и (e, s) с пропускной способностью p_{ij}. Эти ребра изображены на рисунке.



Рисункок. Построение для ребра e = (i, j) с a ≠ f (i) ≠ f (j) ≠ a

Поиск хорошего соседа

Утверждение 1.9. Для заданной разметки f и метки a минимальный разрез в графе G' = (V', E') соответствует соседу разметки f с минимальным штрафом, полученному заменой меток подмножества узлов на a. В результате сосед f с минимальным штрафом может быть найден посредством k вычислений минимального разреза, по одному для каждой метки в L.

Доказательство. Пусть (A, B) — разрез s–t в G'. Большое значение M гарантирует, что разрез с минимальной пропускной способностью не разрежет никакие из этих ребер с высокой пропускной способностью. Теперь рассмотрим узел e в G', соответствующий ребру e = (i, j) ∈ E. Узел e ∈ V' имеет три прилегающих ребра, каждое из которых имеет пропускную способность p_{ij}. Для любого разбиения остальных узлов e можно разместить так, что разрезано будет не более одного из этих трех ребер. Назовем разрез хорошим, если никакое ребро с пропускной способностью M не разрезается и для всех узлов, соответствующих ребрам в E, разрезается не более одного из прилегающих ребер. К настоящему моменту мы обосновали, что все разрезы с минимальной пропускной способностью являются хорошими.

Поиск хорошего соседа

Хорошие разрезы s–t в G' находятся в однозначном соответствии с переназначениями меток f, полученными заменой метки подмножества узлов на a. Рассмотрим пропускную способность хорошего разреза. Ребра (s, i) и (i, t) добавляют в пропускную способность разреза в точности штраф за назначение. Ребра (i, j), напрямую соединяющие узлы в V, вносят в точности штраф за разделение узлов в соответствующей разметке: p_{ij}, если они разделены, и 0 в противном случае. Наконец, рассмотрим ребро e = (i, j) с соответствующим узлом e ∈ V'. Если оба узла i и j находятся на стороне источника, ни одно из трех ребер, прилегающих к e, не будет разрезано, а во всех остальных случаях разрезается ровно одно из этих ребер. Итак, три ребра, прилегающий к e, добавляют к разрезу в точности величину штрафа за разделение между i и j в соответствующей разметке. В результате пропускная способность хорошего разреза в точности совпадает со штрафом соответствующей разметки, а следовательно, разрез с минимальной пропускной способностью соответствует лучшему переназначению меток f. ■

Анализ алгоритмов

Утверждение 1.10. Для разметки f и ее соседа f_a

$$\Phi(f_a) - \Phi(f) \le \sum_{i \in V_a^*} \left[c_i\left(f^*(i)\right) - c_i\left(f(i)\right)\right] + \sum_{\substack{(i,j) \\ leaving V_a^*}} p_{ij} - \sum_{\substack{(i,j) in\ or\ leaving V_a^* \\ f(i) \neq f(j)}} p_{ij}\,.$$

Утверждение 1.11. Для любой локально оптимальной разметки f и любой другой разметки f * — Φ(f) ≥ 2Φ(f *).

Утверждение 1.12. Для любого фиксированного ε > 0 версия алгоритма локального поиска, принимающая только значительные улучшения, завершается за полиномиальное время и приводит к такой разметке f, что Φ(f) ≤ (2 + ε) Φ(f*) для любой другой разметки f*.

Доказательства – дз!

Адженда

Локальный поиск	Рандомизированные алгоритмы
45 минут	45 минут

Введение

Идея «случайности» процесса не нова; само понятие возникло давно в истории человеческой мысли. Оно отражено в азартных играх и страховом деле — и то и другое происходит из древних времен. Но хотя столь же интуитивно понятные дисциплины, такие как геометрия или логика, изучаются математическими методами уже несколько тысяч лет, область математического изучения вероятности на удивление молода; первые попытки ее серьезной формализации были предприняты в XVII веке. Конечно, компьютерная наука существует на много меньшем историческом отрезке, и случайности в ней уделяется внимание с первых дней.

Темы рандомизация и вероятностного анализа проникают во многие области компьютерной науки, включая разработку алгоритмов, и когда речь заходит о случайных процессах в контексте вычислений, обычно имеется в виду одна из двух точек зрения. Первая рассматривает случайное поведение реального мира: здесь часто изучаются традиционные алгоритмы, сталкивающиеся со случайно сгенерированными входными данными. Такие методы часто называются анализом среднего случая, потому что поведение алгоритма изучается на «средних» входных данных (подвергнутых воздействию некоторого случайного процесса) вместо худшего случая.

Введение

Вторая точка зрения связана со случайным поведением самих алгоритмов: мир всегда поставляет одни и те же входные данные худшего случая, но алгоритму разрешено принимать случайные решения при обработке ввода. Таким образом, рандомизация в этом методе действует исключительно внутри алгоритма и не требует новых допущений относительно природы входных данных. Именно этой концепции рандомизированных алгоритмов посвящена данная глава.

Чем может быть полезен алгоритм, принимающий случайные решения? Во-первых, рандомизация расширяет возможности используемой модели. Эффективные детерминированные алгоритмы, которые всегда дают правильный ответ, могут рассматриваться как особый случай эффективных рандомизированных алгоритмов, которые дают правильный ответ с очень высокой вероятностью; они также являются особым случаем рандомизированных алгоритмов, которые всегда работают правильно и при этом, как ожидается, выполняются эффективно. Даже при получении худших входных данных алгоритм, выполняющий свою «внутреннюю» рандомизацию, может компенсировать некоторые нежелательные аспекты худшего случая.

Первое применение: разрешение конфликтов

Начнем с первого применения рандомизированных алгоритмов — разрешения конфликтов в распределенных системах. Этот пример демонстрирует общий стиль анализа, который будет использоваться во многих дальнейших алгоритмах. В частности, он дает возможность попрактиковаться в основных операциях, относящихся к событиям и их вероятностям, анализу пересечений событий с использованием независимости и объединений событий.

Задача

Допустим, имеется n процессов P_1, P_2, ..., P_n, конкурирующих за доступ к одной базе данных. Будем считать, что время делится на кванты. В одном кванте времени с базой данных может работать не более чем один процесс; если два и более процесса пытаются одновременно обратиться к базе данных, эти процессы «блокируются» до конца кванта. Следовательно, хотя каждый процесс хочет как можно чаще обращаться к базе данных, пытаться обращаться к базе данных всем процессам в каждом кванте бессмысленно; в этом случае все процессы будут постоянно находиться в заблокированном состоянии. Нужен справедливый механизм распределения квантов между процессами, чтобы все процессы имели возможность регулярно работать с базой данных.

Если передача данных между процессами реализуется достаточно просто, можно представить себе всевозможные средства для прямого разрешения конфликта. Но предположим, что процессы вообще не могут передавать информацию друг другу; как определить протокол, по которому они смогут «поочередно» работать с базой данных?

Разработка рандомизированного алгоритма

Рандомизация предоставляет естественный протокол для этой задачи, который определяется очень просто. Для некоторого числа $p > 0$, которое будет определено ниже, каждый процесс пытается обратиться к базе данных в каждом кванте с вероятностью p независимо от решений других процессов. Итак, если в каком-либо кванте ровно один процесс выдает такое обращение, его попытка завершается успешно; если обращения поступают от двух и более процессов, они блокируются; и если ни одной попытки не было, то квант «пропадает». Такая стратегия, рандомизирующая поведение каждого процесса из множества одинаковых процессов, лежит в основе уже упоминавшейся парадигмы нарушения симметрии: если все процессы действуют синхронно, многократно пытаясь обратиться к базе данных в одно и то же время, никакого прогресса не будет; рандомизация же позволяет «сгладить» конкурентную борьбу.

Анализ алгоритма

Как и во многих других примерах использования рандомизации, алгоритм в этом случае формулируется очень просто; основной интерес представляет прежде всего анализ его эффективности.

Определение основных событий

Анализ вероятностных систем такого рода полезно начать с определения основных событий и их вероятностей. Первое событие: для заданного процесса P_i и заданного кванта времени t событие A[i, t] обозначает попытку обращения P_i к базе данных в кванте t. Известно, что каждый процесс пытается обратиться к базе данных с вероятностью p, так что вероятность этого события для любых i и t равна Pr[A[I, t]] = p. Для каждого события также существует дополняющее событие, которое означает, что основное событие не произошло; в данном случае дополняющее $\overline{A[i,t]}$ событие означает, что процесс P_i не пытался обратиться к базе данных в кванте t, а его вероятность равна

$$\Pr\left[\overline{A[i,t]}\right] = 1 - \Pr\left[\mathrm{A[I,t]}\right] = 1 - \mathrm{p}.$$

Главный вопрос — удастся ли процессу обратиться к базе данных в заданном кванте. Обозначим это событие S[i, t]. Очевидно, процесс P_i должен хотя бы попытаться обратиться к базе данных в кванте t. Успешное обращение означает, что процесс P_i попытался обратиться к базе данных в кванте t, а все остальные процессы не пытались обращаться к базе данных в кванте t. Следовательно, S[i, t] равно пересечению события A[i,t] с дополняющими событиями $\overline{A[j,t]}$ для j ≠ i:

$$S[i,t] = A[i,t] \cap \left(\bigcap_{j \neq i} \overline{A[j,t]} \right).$$

Все события в этом пересечении независимы по определению протокола разрешения конфликтов.

Определение основных событий

Следовательно, вероятность S[i, t] вычисляется умножением вероятностей всех событий в пересечении:

$$\Pr[S[i,t]] = \Pr[A[i,t]] \cdot \prod_{j \neq i} \Pr\left[\overline{A[j,t]}\right] = p(1-p)^{n-1}.$$

Мы получили удобное выражение в замкнутой форме для вероятности того, что P_i успешно обратится к базе данных в кванте t; теперь уместно задаться вопросом, как выбрать p для максимизации вероятности успеха. Сначала заметим, что вероятность успеха равна 0 для крайних случаев p = 0 и p = 1 (в которых либо процессы вообще не пытаются обратиться к базе данных, либо каждый процесс пытается обратиться к базе данных в каждом кванте, так что в итоге все процессы блокируются). Функция $f(p) = p(1-p)^{n-1}$ положительна для значений p из диапазона от 0 до 1, а ее производная $f'(p) = (1-p)^{n-1} - (n-1)p(1-p)^{n-2}$ равна нулю только в точке p = 1/n, где и достигается максимум. Таким образом, вероятность успеха максимизируется при выборе p = 1/n. (Стоит заметить, что 1/n — естественный и интуитивный вариант в том случае, если в каждом кванте попытка обращения должна исходить ровно от одного процесса.)

Определение основных событий

Выбирая p = 1/n, получаем $\Pr[S[i,t]] = 1/n\left(1-\frac{1}{n}\right)^{n-1}$. Полезно примерно представить асимптотическое поведение этого выражения с помощью следующего факта из курса математического анализа.

Утверждение 2.1. (a) Функция $\left(1-\frac{1}{n}\right)^{n}$ монотонно сходится от ¼ к 1/e при увеличении n, начиная с 2.

(b) Функция $\left(1-\frac{1}{n}\right)^{n-1}$ монотонно сходится от 1/2 к 1/e при увеличении n, начиная с 2.

Из (2.1) следует, что $\frac{1}{en} \leq \Pr[S[i,t]] \leq \frac{1}{2n}$, а следовательно, значение $\Pr[S[i,t]]$асимптотически равно $\theta(\frac{1}{n})$.

Ожидание успешного обращения со стороны конкретного процесса

Рассмотрим этот протокол с оптимальным значением вероятности обращения p = 1/n. Допустим, нас интересует, сколько времени должно пройти, чтобы процесс P_i успешно обратился к базе данных хотя бы один раз. Из предыдущих вычислений видно, что вероятность успеха в любом отдельном кванте не очень хороша при достаточно больших n. Но как насчет серии из нескольких квантов?

Обозначим $\mathcal{F}$[i, t] «событие неудачи», при котором процесс P_i не добивается успеха во всех квантах с 1 до t. Очевидно, оно представляет собой обычное пересечение дополняющих событий $\overline{S[i,t]}$ для r = 1, 2, …, t. Кроме того, поскольку все эти события независимы, вероятность $\mathcal{F}$[i,t] вычисляется умножением:

$$\Pr\left[\mathcal{F}[i,t]\right] = \Pr\left[\bigcap_{r=1}^{t}\left[\overline{S[i,t]}\right]\right] = \prod_{r=1}^{t}\Pr\left[\overline{S[i,t]}\right] = \left[1-\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1}\right]^{t}.$$

Эти вычисления дают значение искомой вероятности, но если и дальше так продолжать, придется иметь дело с чрезвычайно сложными выражениями, поэтому важно переходить на асимптотическое мышление. Вспомните, что вероятность успеха после одного кванта равна $\theta(\frac{1}{n})$, а точнее, она лежит в границах от 1/(en) до 1/(2n). Используя приведенное выше выражение, имеем

$$\Pr\left[\mathcal{F}[i,t]\right] = \prod_{r=1}^{t}\Pr\left[\overline{S[i,t]}\right] \leq \left[1-\frac{1}{ne}\right]^{t}.$$

Ожидание успешного обращения со стороны конкретного процесса

Теперь заметим, что если задать t = en, то мы получим выражение, которое можно напрямую подставить в (2.1). Конечно, en не будет целым числом, поэтому мы возьмем t=|en| и получим:

$$\Pr\left[\mathcal{F}[i,t]\right] \le \left[1-\frac{1}{ne}\right]^{|en|} \le \left[1-\frac{1}{ne}\right]^{en} \le \frac{1}{e}.$$

Это чрезвычайно компактное и полезное асимптотическое утверждение: вероятность того, что процесс P_i не добьется успеха ни в одном из квантов с 1 по |en|,
ограничивается сверху константой e^{-1} независимо от n. Теперь при увеличении t с очень малым коэффициентом вероятность того, что P^i не добьется успеха ни в одном из квантов с 1 по t, резко падает: если задать t = |en| · (c ln n), то получаем

$$\Pr\left[\mathcal{F}[i,t]\right] \le \left[1-\frac{1}{ne}\right]^{t} \le \left[\left(1-\frac{1}{ne}\right)^{|en|}\right]^{c\ ln\ n} \le e^{-c\ ln\ n} = n^{-c}.$$

Итак, асимптотически ситуацию можно рассматривать следующим образом: квантов вероятность того, что процесс P_i еще не смог обратиться к базе после эта вероятность данных, ограничивается константой; а с этого момента до падает до очень малой величины, ограничиваемой обратной полиномиальной зависимостью от n.

Ожидание успешного обращения всех процессов

Наконец, мы добрались до вопроса, неявно подразумевавшегося в общей постановке задачи: сколько квантов должно пройти, чтобы все процессы с достаточно высокой вероятностью обратились к базе данных хотя бы один раз?

Чтобы ответить на него, будем считать, что выполнение протокола привело к неудаче после t квантов, если какой-то процесс так и не смог обратиться к базе данных. Обозначим $\mathcal{F}_t$ событие неудачи протокола после t квантов; наша цель — найти разумно малое значение t, для которого значение $\Pr[\mathcal{F}_i]$ мало.

Событие $\mathcal{F}_t$ происходит в том, и только в том случае, если происходит одно из событий $\mathcal{F}[i,t]$; мы можем записать

$$F_t = \bigcup_{i=1}^{n} F[i,t]\,.$$

Ранее мы рассматривали пересечения независимых событий, с которыми было очень просто работать; на этот раз речь идет об объединении событий, которые независимыми не являются. Точное вычисление вероятностей подобных объединений может быть очень сложным, и во многих случаях достаточно воспользоваться простой границей объединения, которая гласит, что вероятность объединения событий ограничивается сверху суммой их вероятностей:

Ожидание успешного обращения всех процессов

Утверждение 2.2. (Граница объединения) Для заданных событий $\mathcal{E}_1$, $\mathcal{E}_2$, …, $\mathcal{E}_n$ выполняется

$$\Pr\left[\bigcup_{i=1}^{n} \mathcal{E}_i\right] \leq \sum_{i=1}^{n} \Pr[\mathcal{E}_i] .$$

Обратите внимание: речь не идет о равенстве; однако верхняя граница достаточно хороша в подобных ситуациях, когда объединение в левой части представляет «плохое событие», которого мы пытаемся избежать, и мы хотим ограничить его вероятность в контексте составляющих «плохих событий» в правой части.

Для текущей задачи вспомните, что $\mathcal{F}_i = \bigcup_{i=1}^{n} \mathcal{F}[i,t]$, а следовательно,

$$\Pr[\mathcal{F}_i] \leq \sum_{i=1}^{n} \Pr[\mathcal{F}[i,t]] .$$

Выражение в правой части представляет собой сумму n слагаемых с одинаковыми значениями; чтобы вероятность Фt была достаточно малой, необходимо убедиться в том, что каждое слагаемое в правой части существенно меньше 1/n. Из предшествующего обсуждения мы видим, что выбор $t = \theta(n)$ недостаточно хорош, поскольку каждое слагаемое в правой части ограничивается только константой. Если выбрать t = |en| · (c ln n), то $\Pr\left[\mathcal{F}[i,t]\right] \leq n^{-c}$ для всех i — именно то, что нужно.

Ожидание успешного обращения всех процессов

А конкретно, выбор t = 2|en| ln n дает нам

$$\Pr[\mathcal{F}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{F}[i,t]\right] \leq n \cdot n^{-2} = n^{-1}.$$

Таким образом, нам удалось показать следующее:

Утверждение 2.3. С вероятностью не менее $1-n^{-1}$ всем процессам удастся успешно обратиться к базе данных по крайней мере один раз за квантов.

Интересно заметить, что если бы мы выбрали значение t, равное qn ln n для очень малого значения q (вместо фактически использованного коэффициента 2e), тогда для $\Pr\left[\mathcal{F}[i,t]\right]$ была бы получена верхняя граница, большая n^{-1}, а следовательно, соответствующая верхняя граница для общей вероятности неудачи была бы больше 1, — очевидно, такая верхняя граница полностью бесполезна. Однако, как мы видели, выбирая все большие и большие значения для коэффициента q, можно снизить верхнюю границу $\Pr[\mathcal{F}_i]$ до n^{-c} для любой нужной константы c —это очень малая верхняя граница. Итак, в некотором смысле все «действие» в границе объединения быстро происходит в период, когда $t = \theta(n \ln n)$; с изменением скрытой константы внутри $\theta(\cdot)$ граница объединения переходит от нулевой полезной информации до предоставления в высшей степени сильной границы вероятности.

Ожидание успешного обращения всех процессов

Может, это просто артефакт применения границы объединения к нашей верхней границе, или данное свойство присуще наблюдаемому процессу? Мы не будем приводить (довольно громоздкие) вычисления, но при желании можно показать, что для t, равного малой константе, умноженной на n ln n, существует заметная вероятность того, что какому-то процессу так и не удалось обратиться к базе данных. Следовательно, быстрое падение значения $\Pr[\mathcal{F}_i]$ действительно происходит в диапазоне $t = \theta(n \ln n)$. Для этой задачи, как и для многих задач такого типа, мы в действительности находим асимптотически «правильное» значение t, несмотря на использование слабой на первый взгляд границы объединения.

Нахождение глобального минимального разреза

Идея рандомизации естественным образом возникает в приведенном примере — модели с множеством процессов, которые не могут взаимодействовать напрямую. Теперь рассмотрим задачу для графов, в которой тема рандомизации возникает довольно неожиданно, так как для этой задачи существуют вполне разумные детерминированные алгоритмы.

Задача

Для заданного ненаправленного графа G = (V, E) разрез определяется как разбиение V на два непустых подмножества A и B. Ранее, при рассмотрении сетевых потоков, мы работали с похожим определением разреза s–t: тогда для направленного графа G = (V, E) с выделенными узлами источника и стока s и t, разрез s–t определялся как разбиение V на такие множества A и B, что $s \in A$ и $t \in B$. Наше новое определение отличается от того: граф стал ненаправленным, и в нем нет ни источника, ни стока.

Для разреза (A, B) в ненаправленном графе G размер (A, B) равен количеству ребер, один конец которых принадлежит A, а другой принадлежит B. Глобальным минимальным разрезом называется разрез минимального размера. Термин «глобальный» указывает на то, что разрешены любые разрезы графа; нет ни источника, ни стока. Таким образом, глобальный минимальный разрез может рассматриваться как естественная характеристика «надежности»; это минимальное количество ребер, удаление которых приводит к потере связности графом. Сначала мы убедимся в том, что методы сетевого потока действительно достаточны для нахождения глобального минимального разреза.

Утверждение 2.4. Существует алгоритм с полиномиальным временем для нахождения глобального минимального разреза в ненаправленном графе G.

Доказательство. Начнем со сходства разрезов в ненаправленных графах и разрезов s–t в направленных графах и с того факта, что нам известен оптимальный способ нахождения последних.

Заданный ненаправленный граф G = (V, E) необходимо преобразовать так, чтобы в нем были направленные ребра, источник и сток. Каждое ненаправленное ребро $e = (u, v) \in E$ заменяется двумя противоположно ориентированными направленными ребрами, e' = (u, v) и e'' = (v, u), каждое из которых имеет пропускную способность 1. Обозначим полученный направленный граф G'.

Задача

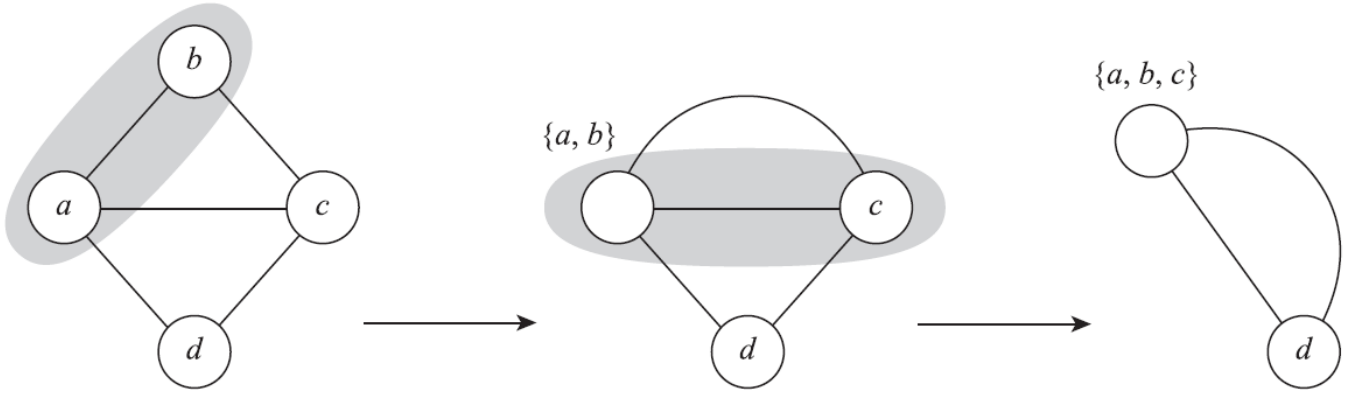
Теперь допустим, что мы выбрали два произвольных узла s, t ∈ V и нашли минимальный разрез s–t в G'. Легко убедиться в том, что если (A, B) является минимальным разрезом в G', то (A, B) также является разрезом минимального размера в G среди всех разрезов, отделяющих s от t. Но мы знаем, что глобальный минимальный разрез в G должен отделять s от чего-то, так как обе стороны A и B не пусты, и s принадлежит только одной из них. Соответственно мы фиксируем любой узел s ∈ V и вычисляем минимальный разрез s–t в G' для всех остальных узлов t ∈ V − {s}. Это требует n − 1 вычислений направленного минимального разреза, лучшим из которых будет глобальный минимальный разрез графа G. ■

Алгоритм в (2.4) создает сильное впечатление, что нахождение глобального минимального разреза в ненаправленном графе в каком-то смысле является более сложной задачей, чем поиск минимального разреза s–t в потоковой сети, так как нам приходится n – 1 раз вызывать процедуру для решения последней задачи в нашем методе для решения первой. Но оказывается, это всего лишь иллюзия. Серия все более простых алгоритмов, разработанных в конце 1980-х и начале 1990-х годов, показала, что глобальные минимальные разрезы в ненаправленных графах могут вычисляться так же эффективно, как разрезы s–t, и даже еще эффективнее, причем для этого не нужны ни улучшающие пути, ни даже концепция потока. Кульминационным моментом в этой области стало открытие Дэвидом Каргером в 1992 году алгоритма стягивания — рандомизированного метода, качественно более простого по сравнению со всеми предшествующими алгоритмами нахождения глобальных минимальных разрезов. В самом деле, алгоритм настолько прост, что на первый взгляд даже трудно поверить, что он действительно работает.

Разработка алгоритма

Сейчас мы опишем алгоритм стягивания в его простейшей форме. Эта версия, хотя и работает за полиномиальное время, не принадлежит к числу самых эффективных алгоритмов для глобальных минимальных разрезов. Впрочем, последующие оптимизации алгоритма существенно улучшили его время выполнения.

Алгоритм стягивания работает со связным мультиграфом G = (V, E); это ненаправленный граф, который может иметь несколько «параллельных» ребер между одной парой узлов. Сначала он случайным образом выбирает в G ребро e = (u, v) и «стягивает» его, как показано на рисунке. В результате появляется новый граф G', в котором u и v порождают новый узел w; все остальные узлы сохраняют свою исходную «личность». Все ребра, у которых один конец равен u, а другой равен v, удаляются из G'. Все остальные ребра e остаются в G', но если один конец ребра равен u или v, то этот конец заменяется новым узлом w. Обратите внимание: даже если в G любые два узла соединены не более чем одним ребром, в G' могут появиться параллельные ребра.



Применение алгоритма стягивания к графу из четырех узлов

Разработка алгоритма

Алгоритм стягивания продолжает рекурсивно выполняться с G', случайно выбирая ребра (с равномерным распределением) и стягивая их. По мере продолжения рекурсии вершины G' должны рассматриваться как суперузлы: каждый суперузел w соответствует подмножеству $S(w) \subseteq V$, «поглощенному» в результате стягиваний, породивших w. Алгоритм завершается при достижении графа G', состоящего из двух суперузлов v_1 и v_2 (предположительно соединенных несколькими параллельными ребрами). Каждый суперузел v_i имеет соответствующее подмножество $S(v_i) \subseteq V$, которое состоит из стянутых в него узлов, и эти два множества $S(v_1)$ и $S(v_2)$ образуют разбиение V. Далее $(S(v_1), S(v_2))$ выводится как разрез, найденный алгоритмом.
Применение алгоритма стягивания к мультиграфу G = (V,E):

Для каждого узла v хранится множество S(v) узлов, стянутых в v

В исходном состоянии $S(v) = \{v\}$ для каждого v

Если G содержит два узла v_1 и v_2, вернуть разрез $(S(v_1), S(v_2))$

Иначе случайно-равномерно выбрать ребро e = (u, v) of G

Пусть G' — граф, полученный в результате стягивания e, в котором новый узел z_{uv} заменяет u и v

Определить $S(z_{uv}) = S(u) \cup S(v)$

Рекурсивно применить алгоритм стягивания к G'

Конец Если

Анализ алгоритма

Утверждение 2.5. Алгоритм стягивания возвращает глобальный минимальный разрез графа G с вероятностью не менее $1/\binom{n}{2}$.

Утверждение 2.6. Ненаправленный граф G = (V,E) из n узлов содержит не более $\binom{n}{2}$ глобальных минимальных разрезов.

Доказательство - дз