2.4 Алгоритм имитации отжига

Алгоритм имитации отжига (***Simulated Annealing***) был разработан различными исследователями в середине 80-х, однако его славная история берет начало от **алгоритма Метрополиса** (***Metropolis Algorithm***), созданного учеными, бывшими участниками проекта Манхэттен, Николасом Метрополисом (*Nicholas Metropolis*), Арианной и Маршаллом Розенблют (*Arianna and Marshall Rosenbluth*) и Автустой и Эдвардом Теллерами (*Augusta and Edward Teller*) в 1953 г.[1] Алгоритм отличается от алгоритма локального поиска (Алгоритм 4) на этапе принятия решения, когда необходимо заменить S, исходное потенциальное решение, новым вариантом R. В частности, если R лучше, чем S, то замена делается всегда, как обычно. Однако если R хуже S, все равно можно заменить S на R с вероятностью $P(t, R, S)$:

$$P(t, R, S) = \exp \frac{\mathsf{Quality}(R) - \mathsf{Quality}(S)}{t},$$

где $t \geq 0$. Другими словами, иногда алгоритм идет в обратную сторону. Для начала, заметим, что показатель степени – отрицательный, т.к. R хуже S. Это уравнение интересно по двум причинам. Во-первых, если R значительно хуже S, то модуль степени увеличивается, и вероятность становится близка к нулю. В случае, если R мало отличается от S, то и вероятность стремится к 1. Поэтому если R ненамного хуже S, то остается шанс выбрать R с разумной вероятностью.

Во-вторых, имеется настраиваемый параметр t. Если t очень мало, то степень увеличивается по модулю и вероятность также близка к 0. Если же t велико, то вероятность примерно равна 1. Идея заключается в том, чтобы изначально присвоить t большое значение, которое заставляет алгоритм «обращать внимание» на любое новое решение, вне зависимости от того, насколько оно хорошо. Получаются **случайные блуждания** в пространстве. Затем t медленно уменьшается, достигая в конце концов 0, после чего алгоритм вырождается в обыкновенный алгоритм локального поиска.

Алгоритм 13 Алгоритм имитации отжига

$t \leftarrow$ температура, большое начальное значение
$S \leftarrow$ начальное потенциальное решение
$Best \leftarrow S$
repeat
  $R \leftarrow$ Tweak (Copy (S))
  if Quality(R) $>$ Quality(S) ИЛИ случайное число из интервала $[0; 1] < \exp \frac{\mathsf{Quality}(R) - \mathsf{Quality}(S)}{t}$ **then**
    $S \leftarrow R$
  end if
  Уменьшить t
  if Quality(S) $>$ Quality($Best$) **then**
    $Best \leftarrow S$
  end if
until $Best$ – идеальное решение, или закончилось время поиска, или $t \leq 0$
return $Best$

Скорость, с которой уменьшается t, называется **расписанием** алгоритма (***schedule***). Чем больше по времени «растягивается» расписание, тем больше алгоритм напоминает случайные блуждания и тем дольше идет исследование пространства поиска.

[0] Перевод раздела из книги Luke S. Essentials of Metaheuristics. A Set of Undergraduate Lecture Notes. Zeroth Edition. Online Version 1.2. July, 2011 (`http://cs.gmu.edu/~sean/book/metaheuristics/`). Перевел – Юрий Цой, 2011 г. Любые замечания, касающиеся перевода, просьба присылать по адресу yurytsoy@gmail.com
Данный текст доступен по адресу: `http://qai.narod.ru/GA/meta-heuristics_2_4.pdf`

[1] Nicholas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller, 1953, Equation of state calculations by fast computing machines, Journal of Chemical Physics, 21, 1087–1091. И, да, Арианна и Маршалл – супруги, также как и Августа и Эдвард. Эта команда также разработала **Метод Монте-Карло** (***Monte Carlo Method***), широко используемый в моделировании. Эдвард Теллер в последующем стал известным защитником ядерных испытаний и считается одним из главных прототипов доктора Стрейнджлав. Чтобы еще сильнее завязать весь этот Гордиев узел, упомяну, что внук Августы и Эдварда Теллера, Эрик Теллер (*Eric Teller*), известный как Астро Теллер (*Astro Teller*), сделал приличный вклад в ранние исследования по генетическому программированию! (см. Раздел 4.3). Вышедшая позднее статья по имитации отжига, рассматривающая его как метод оптимизации, была: Scott Kirkpatrick, Charles Daniel Gelatt Jr., and Mario Vecchi, 1983, Optimization by simulated annealing, Science , 220(4598), 671–680.

Алгоритм имитации отжига берет название от процесса остывания расплавленного металла. Если остудить металл очень быстро, его атомы не успеют сформировать плотную решетку и останутся в случайной конфигурации, из-за чего металл будет хрупким. А если уменьшать температуру очень медленно, то у атомов будет достаточно времени, чтобы образовать крепкий кристалл. Поэтому не удивительно, что t называют **температурой** (***temperature***).

2.5 Поиск с запретом

Алгоритм **поиска с запретом** (***Tabu Search***), разработанный Фредом Гловером[2] (*Fred Glover*), использует оригинальный подход к исследованию пространства поиска: в нем сохраняется информация о недавно сгенерированных потенциальных решениях (называемый *список запретов*, *tabu list*), возврат к которым на последующих этапах невозможен пока не кончится временное ограничение. Таким образом, если алгоритм «взбирается» по холму, то у него нет иного выбора, как перейти на другую сторону, поскольку остаться на вершине нельзя.

Простейшим способом реализации поиска с запретом является поддержание **списка запретов** L с максимальной длиной l, включающего уже найденные потенциальные решения. Каждый раз, когда создается новое решение, оно записывается в список запретов. Если список становится слишком большим, то из него исключается самое «старое» решение, которое перестает быть запретным. Поиск с запретом обычно напоминает вариант наискорейшего подъема с замещением (алгоритм 6). В приведенном ниже варианте создается n измененных потомков, из которых рассматриваются только те, которые отсутствуют в списке запретов. Это требует нескольких дополнительных проверок в коде:

Алгоритм 14 Поиск с запретом

$l \leftarrow$ требуемая длина списка запретов
$n \leftarrow$ количество модификаций, необходимых для сбора информации о градиенте

$S \leftarrow$ начальное потенциальное решение
$Best \leftarrow S$
$L \leftarrow \{\}$ список запретов длины l. {Реализуется в виде FIFO очереди.}
Записать S в очередь L
repeat
  if Length(L) $> l$ **then**
    Удалить самый старый элемент из L
  end if
  $R \leftarrow$ Tweak (Copy (S))
  for $n-1$ раз **do**
    $W \leftarrow$ Tweak (Copy (S))
    if $W \notin L$ и (Quality(W) $>$ Quality(R) или $R \in L$) **then**
      $R \leftarrow W$
    end if
  end for
  if $R \notin L$ и (Quality(R) $>$ Quality(S)) **then**
    $S \leftarrow R$
    Записать R в очередь L
  end if
  if Quality(S) $>$ Quality($Best$) **then**
    $Best \leftarrow S$
  end if
until $Best$ – идеальное решение, или закончилось время поиска
return $Best$

Поиск с запретом предназначен для работы в дискретных пространствах. Что делать, если пространство поиска – вещественное? Проблема в том, что в вещественном пространстве посетить одну

[2] «Tabu» является альтернативной формой слова «taboo». Гловер также внес вклад в использование слова «метаэвристика» и создал Распределенный поиск с переопределением путей (см. Раздел 3.3.5). Впервый поиск с запретом был описан в публикации Fred Glover, 1986, Future paths for integer programming and links to artificial intelligence, Computers and Operations Research, 5, 533–549.

и ту же точку дважды практически нереально, что делает список запретов бесполезным. В этом случае можно считать, что решение принадлежит списку, если оно «достаточно близко» к какому-либо элементу из списка. Выбор меры схожести целиком зависит от исследователя. Некоторые идеи изложены в разделе 6.4.

Даже если исключить эту проблему, большой трудностью для поиска с запретом являются случаи очень больших пространств поиска, в частности, в пространстве большой размерности очень легко застрять в локальной окрестности, принадлежащей одному пику, даже если использовать очень большой список запретов. Возможных решений может быть слишком много. Альтернативой является создание списка запретов не для потенциальных решений, встреченных ранее, а для *изменений значений параметров*. К примеру, необходимо найти решение для задачи коммивояжера (см. Раздел 8). Существующее решение можно улучшить, если удалить ребро и добавить ребра B и C, а затем оценить новое решение. Вместо того, чтобы записать в список запретов новое решение, туда можно записать изменения для ребер A, B и C. Тогда при переборе новых вариантов улучшения текущего решения нельзя будет рассмотреть удаление или добавление этих ребер. Другими словами, они теперь тоже «попадают под запрет».

Для реализации подобного подхода необходимо изменить принцип организации списка запретов. Простая очередь, работающая по правилу FIFO, уже не подойдет, поскольку в очередь будут записываться объекты состоящие из различного количества элементов. Вместо этого очередь можно реализовать в виде множества пар $\langle X, d\rangle$, где X – измененный параметр (например, «ребро A»), а d – метка времени, когда было произведено изменение. Кроме этого, простой проверки принадлежности объекта очереди будет недостаточно. Необходимо передавать очередь запретов в операцию Tweak, чтобы сообщать о недоступных изменениях, поэтому в измененной версии будет использовано: Tweak (Copy (...)), L. Я назвал новый алгоритм **Поиск с запретом, основанный на параметрах** (***Feature-based Tabu Search***).

Алгоритм 15 Поиск с запретом, основанный на параметрах

$l \leftarrow$ требуемая длина списка запретов
$n \leftarrow$ количество модификаций, необходимых для сбора информации о градиенте

$S \leftarrow$ начальное потенциальное решение
$Best \leftarrow S$
$L \leftarrow \{\}$ {L будет содержать пары вида $\langle X, d\rangle$, где X – параметр, d – временная метка.}
$c \leftarrow 0$
repeat
  $c \leftarrow c + 1$
  Удалить из L все пары вида $\langle X, d\rangle$, в которых $c - d > l$ {«Старые» объекты.}
  $R \leftarrow$ Tweak (Copy (S), L)
  for $n - 1$ раз **do**
    $W \leftarrow$ Tweak (Copy (S, L))
    if Quality(W) $>$ Quality(R) **then**
      $R \leftarrow W$
    end if
  end for
  $S \leftarrow R$
  for каждый параметр X измененый операцией Tweak для создания R из S **do**
    $L \leftarrow L \cup \{\langle X, c\rangle\}$
  end for
  if Quality(S) $>$ Quality($Best$) **then**
    $Best \leftarrow S$
  end if
until $Best$ – идеальное решение, или закончилось время поиска
return $Best$

Поиск с запретом, основанный на параметрах отличается от описанных в этой главе методов тем, что он в существенной степени опирается на свойства идентифицируемости и разделимости *параметров* потенциальных решений, вместо рассмотрения каждого решения как атомарной сущности, за исключением операции Tweak. Далее мы увидим, что такой подход очень активно используется в комбинаторной оптимизации (Раздел 8).

2.6 Итеративный локальный поиск

Сейчас так называется концепция, которая нашла свою реализацию в различных методах с как минимум 80-х годов [3]. Она представляет более эффективную версию локального поиска со случайными перезапусками. При каждом случайном перезапуске локальный поиск оказывается в некотором (возможно новом) локальном оптимуме. Поэтому случайный поиск с перезапусками можно рассматривать как случайный поиск в *пространстве локальных оптимумов*. Сначала будет найден один локальный оптимум, затем другой, потом третий и т.д. и в конечном итоге будет выбрано наилучшее найденное решение (в идеале – глобальный оптимум!).

Итеративный локальный поиск (ИЛП) (***Iterated Local Search (ILS)***) пытается работать более интеллектуально: он осуществляет *стохастический локальный поиск в пространстве локальных оптимумов*. Т.е. ИЛП находит локальный оптимум, а затем производит поиск другого локального оптимума в окрестности и возможно делает его текущим, потом ищет новый локальный оптимум в окрестности нового и т.д. Здесь используется эвристика, что всегда можно найти более «хороший» локальный оптимум рядом с текущим, и подобные перемещения должны быть более эффективными, чем совершенно случайные перезапуски.

В ИЛП используются два приема. Первый заключается в том, что координаты для перезапуска выбираются не случайно. Вместо этого применяется своеобразный «домашний» локальный оптимум и новые точки для перезапуска как правило, хотя и не всегда, попадают в окрестность этого «домашнего» локального оптимума. Необходимо произвести перезапуск достаточно далеко от текущего оптимума, чтобы найти новый, но не сильно далеко, чтобы не оказаться в совершенно случайной окрестности. Нужен маршрут, а не случайные блуждания.

Второй прием состоит в принятии решения, оставлять ли текущий «домашний» локальный оптимум, или присвоить этот статус только что найденному. Если всегда делать переходы, то получим случайные блуждания (подобие мета-исследования). Если переходить в локальные оптимумы только если они «лучше» текущего, то это будет локальный поиск. В ИЛП часто выбирают нечто среднее между этими вариантами, как будет описано далее.

Если отвлечься от данных двух приемов, то ИЛП выглядит очень просто. Единственная сложность заключается в том, чтобы определить, что был найден локальный оптимум. Поскольку это всегда сложно, то вместо этого будем использовать тот же подход, что применялся при случайных перезапусках: установим таймер. Будем производить локальный поиск, а когда закончится время, сделаем перезапуск. Очевидно, что это не дает гарантий нахождения локального оптимума, однако если таймер заведен на достаточно большое время, то мы с большой вероятностью окажемся в окрестности оптимума.

Получившийся алгоритм работает очень просто: осуществляем локальный поиск; затем (когда закончится время) принимаем решение, оставаться в текущем «домашнем» оптимуме или переходить в новый (функция NewHomeBase[4]); после этого делаем очень большой Tweak (либо Perturb), который настроен таким образом, чтобы имелась *достаточная* вероятность оказаться в новом оптимуме. Алгоритм выглядит так:

Как правило, выбор функций Perturb и NewHomeBase напоминает шаманство и определяется в основном природой решаемой задачи. Но можно дать ряд рекомендаций.

Целью Perturb является совершить очень большой Tweak, достаточный, чтобы покинуть область притяжения текущего локального оптимума, но не настолько радикальный, чтобы выродиться в случайный поиск. Помните, что необходимо оказаться в *соседнем* оптимуме. Поэтому реальное значение слова «достаточный» может очень сильно меняться от задачи к задаче.

Задача функции NewHomeBase состоит в интеллектуальном выборе новых стартовых позиций. Точно так же, как алгоритмы глобальной оптимизации балансируют между двумя крайностями: исследованием (случайным поиском) и эксплуатацией (локальный поиск), также и функция NewHomeBase находится между подобными границами при анализе *локальных оптимумов*[5]. С одной стороны алгоритм может всегда принимать новый локальный оптимум:

$$\mathsf{NewHomeBase}(H, S) = S.$$

Результатом очевидно являются случайные блуждания от одного локального оптимума к другому.

[3] Хороший обзор метода можно найти в статье Helena Lourenço, Olivier Martin, and Thomas Stützle, 2003, Iterated local search, in Fred Glover and Gary Kochenberger, editors, *Handbook of Metaheuristics*, pages 320–353, Springer. В ней отслеживается история метода с публикации ohn Baxter, 1981, Local optima avoidance in depot location, Journal of the Operational Research Society, 32, 815–819.

[4] Я сам придумал это имя.

[5] Вот эта функция действительно *мета*-эвристическая. Наконец-то правильное использование термина!

Алгоритм 16 Итеративный локальный поиск со случайными перезапусками

```
1: T ← распределение для интервалов времени

2: S ← начальное потенциальное решение
3: H ← S {Текущий «домашний» оптимум.}
4: Best ← S
5: repeat
6:    time ← случайный момент времени в будущем, определенный с использованием T
7:    repeat
8:       R ← Tweak (Copy (S))
9:       if Quality(R) > Quality(S) then
10:         S ← R
11:      end if
12:   until S – идеальное решение, или закончилось время time, либо закончилось общее время
      поиска
13:   if Quality(S) > Quality(Best) then
14:      Best ← S
15:   end if
16:   H ← NewHomeBase (H, S)
17:   S ← Perturb (H)
18: until Best – идеальное решение, или закончилось общее время поиска
19: return Best
```

С другой стороны, алгоритм может принимать новый локальный оптимум только если его качество не хуже качества текущего оптимума, то есть:

$$\mathsf{NewHomeBase}(H,S) = \begin{cases} S & \text{если } \mathsf{Quality}(S) \geq \mathsf{Quality}(H), \\ H & \text{в противном случае.} \end{cases}$$

В итоге получится в некотором роде аналог локального поиска среди локальных оптимумов. Большинство реализаций ИЛП пытаются использовать промежуточный подход. К примеру, ИЛП может осуществлять локальный поиск до тех пор пока будут находиться новые более «хорошие» решения, если таковых не будет, то алгоритм переключается на некоторое время на случайные блуждания. Есть, конечно, и другие варианты: в функции NewHomeBase можно применить алгоритм имитации отжига или поиск с запретом.

Сочетания и комбинации Алгоритмы, описанные в этой главе не являются жестко заданными. Существует большой число их сочетаний и комбинаций, включая и нерасмотренные подходы. К примеру, не будет бессмысленным использовать локальный поиск со случайными перезапусками с операцией Tweak, работающей по принципу (1+1). Или можно придумать различные версии случайных перезапусков использующие алгоритм наискорейшего подъема. Поиск с запретом можно осуществлять в виде $(1, \lambda)$ алгоритма. Либо рассмотреть процедуру Tweak, в которой параметр σ^2 для Гауссовской свертки убывает наподобие температуры в алгоритме имитации отжига. И так далее. Творите.