8. Комбинаторная оптимизация

До этого момента мы рассматривали общие задачи с неким произвольным пространством поиска, которое могло быть сформировано перестановками переменных (векторов фиксированной длины), либо могло иметь достаточно разумную метрику или даже включать деревья или наборы правил.

Один тип пространства поиска заслуживает отдельного рассмотрения. Он относится к **задаче комбинаторной оптимизации** (***combinatorial optimization problem***)[1], в которой решение представляет собой комбинацию уникальных компонент, выбираемых из, как правило, конечного и часто малого набора. Целью является поиск оптимальной комбинации компонент.

Классической задачей комбинаторной оптимизации является **задача об упаковке ранца** (***knapsack problem***): имеется *n блоков* различной высоты и стоимости (не связанной с высотой), а также *ранец*[2] фиксированной высоты, большей, чем у любого блока, рис. .1. Цель: заполнение ранца блоками максимальной стоимости ($\$\$\$$, €€€или ¥¥¥), не переполнив ранец[3]. Блоки являются компонентами. На рис. .2 показаны различные варианты размещения блоков в ранце. Можно увидеть, что максимальная заполненность ранца не влечёт оптимальности: важна ценность содержимого, которое можно упаковать. Решение с переполнениями считаются **невыполнимыми** (***infeasible***) (или *некорректными*, *неверными*).

Данная задача не является тривиальной или надуманной. Ей посвящено большое число публикаций, и многие задачи можно к ней свести. Задачи подобного типа встречаются при определении очередности работы процессоров в операционной системе, составлении расписания движения грузовых автомобилей, или когда необходимо купить в ресторане аперитивов ровно на 15.05 $[4].

Еще одним примером является классическая **задача коммивояжера** (***traveling salesman problem, TSP)***), в которой имеется набор *городов*, попарно связанных между собой маршрутами (например, авиарейсами). Для каждого маршрута определена *цена*. Коммивояжер должен составить тур – последовательность посещения городов, начинающуюся с города A, при которой каждый город будет посещен один раз, и заканчивающуюся в A. При этом тур должен иметь минимально возможную цену. Другими словами, города являются вершинами, а маршруты – ребрами графа, веса которого равны весам, а целью является поиск цикла минимальной длины, проходящего по одному разу через каждую вершину. В этом примере компонентами являются не блоки, а ребра графа. Перестановва ребер местами играет большую роль, поскольку существует множество наборов ребер, не имеющих смысла, т.к. они не образуют цикл.

Цена и стоимость Несмотря на то, что в задаче коммивояжера имеется минимизируемая цена (соответствующая весу ребер), а в задаче об упаковке – максимизируемая стоимость, в действительности они играют одну и ту же роль, достаточно поменять знак или взять обратную величину от цену, чтобы получить стоимость. В большинстве алгоритмов комбинаторной оптимизации традиционно упоминается цена, однако мы будем рассматривать оба случая. Как бы то ни было, преобразование цены компоненты C_i в стоимость (и наоборот) может быть совершено наподобие:

$$\text{Стоимость}(C_i) = \frac{1}{\text{Цена}(C_i)}.$$

В текущем разделе будем подразумевать данное соотношение. Естественно, полагается, что цены (и стоимости) > 0, как обычно и бывает. Если цены и стоимости могут принимать как положительное, так и отрицательное значения, то поскольку некоторые описываемые далее методы

[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_8.pdf`

[1] Не путать с *комбинаторикой*, отдельным направлением, которое может в большей или меньшей степени включать практически все обсуждаемые до этого момента вопросы.

[2] Сюда же относятся и различные задачи об **упаковке** (***bin packing***), в которых необходимо найти способ расположить блоки в многомерной корзине так, чтобы они не вываливались.

[3] Существуют различные варианты постановки задачи о ранце. Например, в одном из них допускается иметь любое число блоков данного размера.

[4] `http://xkcd.com/287/`

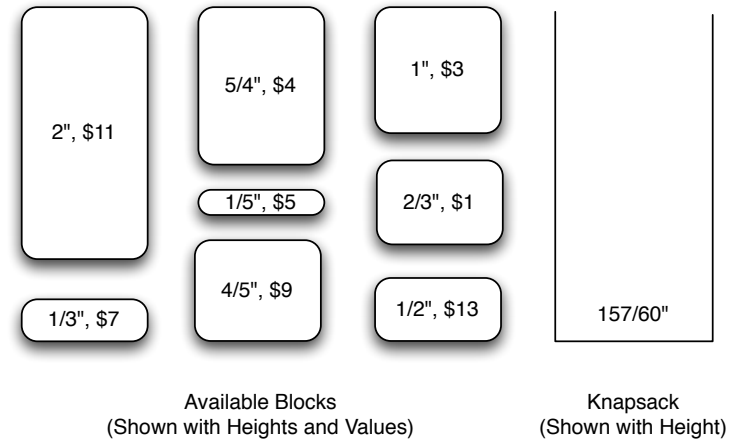

Рис. .1: Задача об упаковке ранца. Необходимо наполнить ранец содержимым максимальной стоимости ($$$), не превышая высоту ранца

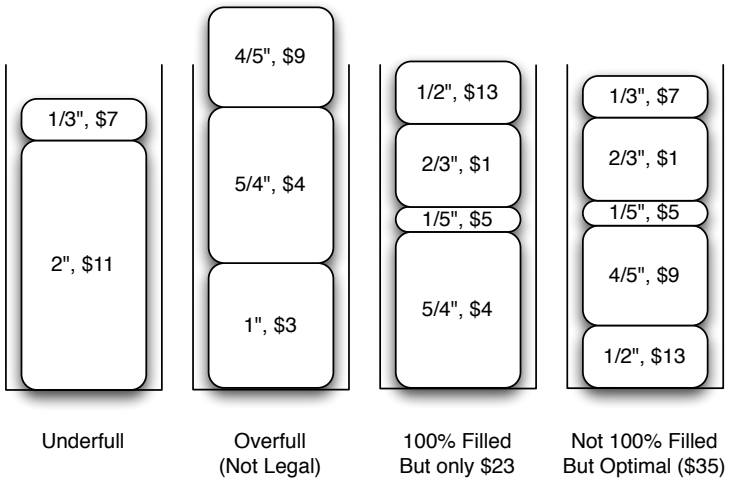

Рис. .2: Заполнение ранца

используют пропорциональную селекцию, необходимо увеличить эти значения на некоторую величину, чтобы избавиться от отрицательных значений. Кроме этого, существуют задачи, в которых все компоненты имеют одинаковую стоимость или цену. Можно также добавить в алгоритм **эвристику** (***heuristic***)[5], которая применяет пользовательское оценивание некоторых компонентов. В последнем случае можно использовать выражение: Стоимость(C_i) =ЭвристическаяСтоимость(C_i).

В задаче о ранце есть нечто, отсутствующее в задаче коммивояжера, а именно дополнительные **веса**[6] (высоты блоков), а также максимальный «вес», которые *не должен* быть превышен. В задаче коммивояжера подобное ограничение не используется для обнаружения бессмысленных решений.

8.1 Оптимизация общего назначения и жесткие ограничения

Задачи комбинаторной оптимизации может решать с применением большинства общих метаэвристических методов, встреченных ранее, и, действительно, определенные методы (итеративный локальный поиск, поиск с запретом и т.д.), часто используются для подобных задач. Однако необходимо помнить, что большинство метаэвристических методов созданы для решения задач оптимизации скорее в «открытых» пространствах поиска, нежели при наличии ограничений, что часто

[5] Эвристика – это эмпирическое правило, добавляемое разработчиком в алгоритм. Нередко оно может быть и ошибочным, однако часто оказывается полезным настолько, что может применяться в качестве руководства к действию.

[6] Согласен, можно запутаться между весами ребер в задаче коммивояжера и весами комбинируемых компонент. Но, увы, такова терминология.

присутствует в задачах комбинаторной оптимизации. Эти методы все равно можно адаптировать под задачу, но требуется учет имеющихся специфичных ограничений[7].

Для примера рассмотрим использование булевского вектора в метаэвристике, такой как имитация отжига или генетический алгоритм. Каждый элемент вектора – представляет компоненту, и если значение некоторого элемента равно Истина, то соответствующая компонента будет включена в решение. Так, на рис. .1 имеются блоки высотой 2, $\frac{1}{3}$, $\frac{5}{4}$, $\frac{1}{5}$, $\frac{4}{5}$, 1, $\frac{2}{3}$ и $\frac{1}{2}$. Потенциальное решение для этой задачи может быть представлено вектором из 8 переменных. Оптимальное решение, показанное на рис. .2 можно записать как ⟨Ложь, Истина, Ложь, Истина, Истина, Ложь, Истина, Истина⟩, что соответствует блокам $\frac{1}{3}$, $\frac{1}{5}$, $\frac{4}{5}$, $\frac{2}{3}$ и $\frac{1}{2}$.

Проблема данного подхода в том, что он допускает появление бессмысленных решений. В задаче о ранце было продекларировано, что решения, не «помещающиеся» в ранец, являются недопустимыми. Появление таких решений для этой задачи не является катастрофой, если они в дальнейшем способствуют нахождению оптимума, можно просто назначить дополнительные штрафы за такие решения (возможно, пропорционально степени переполненности ранца). Плюс можно еще как-нибудь их оштрафовать за сам факт нарушения ограничения. Но в задачах вроде задачи коммивояжера наш вектор может содержать по одной компоненте для каждого ребра в графе маршрутов. Здесь легко получить решение, которое в принципе не осуществимо, ведь неизвестно, как следует оценивать потенциальное решение, если оно даже не является циклом.

Загвоздка в том, что в подобных задачах имеют место **жесткие ограничения** (***hard constraints***), из-за которых в пространстве поиска имеются большие подобласти, содержащие некорректные решения. В конечном итоге необходимо получить все-таки допустимое решение, а во время его поиска хорошо бы работать с допустимым потенциальными решениями, чтобы можно было их адекватно оценить. Для достижения этого необходимы два шага: инициализация (конструирование) начального потенциального решения, и его улучшение (Tweaking).

Конструирование Итеративное конструирование компонент потенциального решения при наличии жестких ограничений может быть очевидным, а может и нет. Часто необходимо сделать следующее:

1. Выбор компоненты. Например, для задачи коммивояжера, можно выбрать ребро, связывающее города A и B. В задаче о ранце такой компонентой является начальный блок. Положим, что текущее (частичное) решение начинается с данной компоненты.

2. Выбор подмножества компонент, которые можно конкатенировать к компонентам текущего решения. В TSP, это подмножество может включать все ребра, выходящие из A или B, в то время как в задаче о ранце, это будут все блоки, которые можно добавить в ранец, не переполнив его.

3. Отбрасывание непривлекательных компонент. В задаче TSP можно отдавать больший приоритет ребрам, которые ведут в еще не посещенные города.

4. Добавление к частичному решению компоненты выбранной из списка оставшихся.

5. Если больше нельзя добавить ни одной компоненты, то выход. Иначе переход на шаг 2.

Данное описание нарочно сделано очень общим, т.к. в большинстве случаев итеративное конструирование решения зависит от решаемой задачи и требует множества экспериментов.

Улучшение Оператор Tweak сделать корректным еще сложнее, поскольку в пространстве поиска допустимые решения могут быть окружены недопустимыми. Вот четыре общих подхода:

- Разработка **замкнутого** (***closed***) оператора Tweak, которые автоматически создает корректные решения. Это может быть непросто, в частности если используется кроссинговер. И если создается замкнутый оператор, то можно ли с его помощью получить все возможные допустимые решения? Существуют ли предпочтения (*bias*)? Знаете ли вы их?

- Итеративно применяются различные варианты оператора Tweak пока не будет получен допустимый потомок. Это относительно несложно сделать, но может потребовать много вычислений.

[7] Есть хорошая обзорная статья на эту тему, написанная двумя столпами в данной области: Zbigniew Michalewicz and Marc Schoenauer, 1996, Evolutionary algorithms for constrained parameter optimization problems, *Evolutionary Computation*, 4(1), 1–32.

- Допускается присутствие некорректных решений, но для них создается функция приспособленности, учитывающая расстояние до ближайшего допустимого решения или оптимума. Для одних задач это выполнить проще, чем для других. Например, в задаче о ранце это просто: качество решений с переполнением можно оценивать на основе того, насколько сильно переполнен ранец (также как и решения с недозаполнением).

- Недопустимым решениям присваивается низкое качество. Это фактически удаляет такие решения из популяции, и в итоге эффективный размер популяции уменьшается. Здесь есть другая проблема: переход *через* границу между допустимыми и недопустимыми решениями приводит к существенному падению оценки качества, как в Хеммингове пике (*Hamming Cliff*) (см. Представление данных, Раздел 4). Так в задаче о ранце, лучшие решения очень близко расположены к недопустимым, поскольку они (почти) полностью заполняют ранец. Поэтому небольшая мутация оптимального решения и вуаля, решение уже некорректно и получает большой штраф к оценке качества. Данный эффект превращает оптимизацию в окрестности лучших решений похожей на хождение по веревке над пропастью.

Ни один из этих вариантов не является наилучшим. Несмотря на то, что часто удается создать корректный оператор конструирования решений, разработка хорошего оператора Tweak, который является замкнутым, может быть очень сложной. Другие методы либо требуют больших вычислений, либо приводят к появлению в популяции недопустимых решений.

Компонентно-ориентированные методы Оставшаяся часть данного раздела посвящена методам, разработанным для работы в специфичных пространствах поиска, часто присутствующими в задачах комбинаторной оптимизации, и использующим тот факт, что решения являются *комбинацией компонент* из некоторого обычно *фиксированного набора*. Благодаря этому можно воспользоваться подобием жадного, локального поиска сохраняя оценку «исторической ценности» каждой компоненты, вместо (или в дополнение к) качества всего решения в целом. Тому есть две причины:

- Необходимость отдавать предпочтение «хорошим» компонентам во время конструирования решения.

- Необходимость изменять компоненты, которые тянут решение в сторону локального оптимума во время работы оператора Tweak.

Для начала рассмотрим метаэвристику под названием **Жадные рандомизированные процедуры случайного поиска** (***Greedy Randomized Adaptive Search Procedures, GRASP***), в которых рассматривается простой вариант конструирования комбинаторных решений из отдельных компонент, а затем их улучшение. Затем будет описан близкий метод, **Алгоритм муравьиной колонии** (***Ant Colony Optimization***), в котором компонентам присваиваются значения «исторической ценности», чтобы в более агрессивной манере конструировать решения из исторически более «хороших» компонент. После этого мы изучим вариант поиска с запретом, известный как **Управляемый локальный поиск** (***Guided Local Search***), в котором основной упор делается на операцию Tweak, при этом штрафуются компоненты, использование которых ухудшает работу алгоритма.

В некоторых из этих методов используется, хотя и по-разному, «историческая ценность» отдельных компонент. В алгоритме муравьиной колонии награждаются более успешные компоненты, тогда как в направленном локальном поиске собирается информация, позволяющая определить, какие из неуспешных компонент часто встречаются в локальном оптимуме.

Значение качества или приспособленности Поскольку задачи комбинаторной оптимизации могут быть сформулированы в терминах стоимости или цены, то и смысл *качества* или *приспособленности* становится размытым. Если в задаче используется стоимость (как в задаче о ранце), то качество или приспособленность можно определить просто как общую стоимость $\sum_i \text{Стоимость}(C_i)$, всех компонент C_i, формирующих потенциальное решение. Если в задаче присутствует цена (как в задаче TSP), то все не так просто: необходимо наличие многих недорогих компонент, которые совместно составляют высококачественное решение. Общепринятым подходом является определение качества или приспособленности как $1/(\sum_i \text{Цена}(C_i))$, для каждой компоненты C_i потенциального решения.

8.2 Жадные рандомизированные процедуры случайного поиска

В любом случае, будем начинать с простой метаэвристики с одним состоянием, в которой используется и конструирование, и улучшение допустимых решений, но не рассматривается «историческое качество» на уровне компонент. Это все – **Жадные рандомизированные процедуры случайного поиска** (***Greedy Randomized Adaptive Search Procedures, GRASP***), созданные Томасом Фео (*Thomas Feo*) и Маурисио Ресенде (*Mauricio Resende*)[8]. Сам алгоритм действительно прост: создается допустимое решение путем его конструирования из компонент с максимальной стоимостью (минимальной ценой) (используя подход, описанный выше), а затем осуществляется локальный поиск в окрестности этого решения.

Алгоритм 108 Жадные рандомизированные процедуры случайного поиска, GRASP

$C \leftarrow \{C_1, \ldots, C_n\}$ компоненты
$p \leftarrow$ доля компонент в процентах, добавляемых на каждой итерации
$m \leftarrow$ продолжительность локального поиска

$Best \leftarrow \square$
repeat
  $S \leftarrow \{\}$ {Потенциальное решение.}
  repeat
    $C' \leftarrow$ компоненты из $C - S$, которые можно добавить и сохранить решение допустимым
    if C' пусто **then**
      $S \leftarrow \{\}$ {Еще одна попытка.}
    else
      $C'' \leftarrow p\%$ компонент с наибольшей стоимостью (наименьшей ценой) из C'
      $S \leftarrow S \cup \{$случайно выбранная компонента из $C''\}$
    end if
  until S – полное решение
  for m раз **do**
    $R \leftarrow$ Tweak(Копия(S)) {Операция Tweak должна быть замкнутой, т.е. должна создавать допустимые решения.}
    if Качество(R) $>$ Качество(S) **then**
      $S \leftarrow R$
    end if
  end for
  if $Best = \square$ или Качество(S) $>$ Качество($Best$) **then**
    $Best \leftarrow S$
  end if
until $Best$ – идеальное решение, или закончилось время поиска
return $Best$

Вместо выбора $p\%$ наилучших доступных компонент в некоторых версиях GRASP выбираются компоненты ценность которых не меньше (или цена не больше) заданной. В GRASP в той или иной степени применяется селекция усечением среди компонент для формирования начального решения. Можно использовать и что-нибудь другое, например, турнирную или же пропорциональную селекцию компонент (описание этих методов см. в Главе 3).

В GRASP демонстрируется один из способов конструирования потенциальных решений путем итеративного перебора компонент. Однако здесь, так же как и в эволюционном подходе, остается загадка, связанная с операцией Tweak: необходимо придумать способ, гарантирующий замкнутость.

[8] Первая статья по GRASP: Thomas A. Feo and Mauricio G. C. Resende, 1989, A probabilistic heuristic for a computationally difficult set covering problem, *Operations Research Letters*, 8, 67–71. Многие текущие публикации Ресенде по GRASP можно найти на сайте: `http://www.research.att.com/~mgcr/doc/`.

8.3 Оптимизация муравьиной колонией

Оптимизация муравьиной колонией (ОМК) (*Ant Colony Optimization, ACO*)[9], созданный Марко Дориго (*Marco Dorigo*), воплощает подход к комбинаторной оптимизации, при котором операция Tweak является необязательной. Вместо этого решению конструируются из путем выбора конкурирующих между собой компонент.

Алгоритм ОМК является популяционным, однако в нем присутствуют популяции двух видов. Первая представляет набор *компонент*, составляющих потенциальное решение задачи, например, в задаче о ранце в нее будут входить все блоки, а в задаче TSP – все ребра. Этот набор постоянен, но у различных компонент в популяции с течением времени изменяется их «приспособленность» (называемая **феромоном** (***pheromone***)).

В каждом поколении формируется одно или несколько потенциальных решений, называемых в терминах ОМК **муравьиной тропой** (***ant trail***), путем поочередного выбора компонент в соответствии с количеством их феромона. Данный процесс порождает «популяцию» второго вида для ОМК: набор троп. Для каждой тропы оценивается приспособленность, и на основе полученной оценки обновляются компоненты: часть приспособленности добавляется к феромону каждой компоненты. Напоминает своеобразный алгоритм кооперативной коэволюции с одной популяцией, не правда ли?

Простейший абстрактный алгоритм ОМК:

Алгоритм 109 Абстрактный алгоритм оптимизации муравьиной колонией (ОМК)

$C \leftarrow \{C_1, \ldots, C_n\}$ компоненты
popsize $\leftarrow$ количество троп, формируемых на каждой итерации {«Муравьиные тропы» в АМК соответствуют потенциальным решениям.}
$\overrightarrow{p} \leftarrow \langle p_1, \ldots, p_n \rangle$ феромоны компонент, изначально равны нулю
$Best \leftarrow \square$
repeat
  $P \leftarrow$ *popsize* троп, сформированных путем итеративного выбора компонент на основе их феромона и значений цены или ценности
  for $P_i \in P$ **do**
    $P_i \leftarrow$ Опциональное локальное улучшение P_i
    if $Best = \square$ или Приспособленность(P_i) > Приспособленность($Best$) **then**
      $Best \leftarrow P_i$
    end if
  end for
  Обновить вектор $\overrightarrow{p}$ для компонент на основании оценок приспособленности решений $P_i \in P$, в которые эти компоненты были включены
until $Best$ – идеальное решение, или закончилось время поиска
return $Best$

Я постарался подчеркнуть общие черты с алгоритмом GRASP: в обоих алгоритмах сначала итеративно происходит конструирование потенциальных решений, а затем их локальное улучшение. Однако есть и очевидные различия. Во-первых, в ОМК конструируется сразу *popsize* потенциальных решений. Во-вторых, в ОМК локальное улучшение является опциональным, и в действительности часто не осуществляется. Если построение замкнутого оператора Tweak сопряжено с трудностями для выбранного способа кодирования, то при необходимости можно полностью проигнорировать шаг с локальным улучшением.

В-третьих, что наиболее важно компоненты выбираются не только по их ценности и цене, но и с учетом *феромона*. Феромон в сущности представляет «историческую ценность» компоненты и часто вычисляется как сумма (или среднее, и т.д.) приспособленностей всех троп, содержащих эту компоненту. Феромон указывает насколько хорошей является компонента вне зависимости от ее (возможно низкой) ценности или (высокой) цены. После определения приспособленности троп величина феромона обновляется таким образом, чтобы отражать последние значения приспособленностей для уменьшения или увеличения вероятности выбора компоненты на последующих итерациях.

[9] АМК известны с приблизительно 1992 г., когда Дориго представил их в своей диссертации: Marco Dorigo, 1992, *Optimization, Learning and Natural Algorithms*, Ph.D. thesis, Politecnico di Milano, Milan, Italy. В настоящей книге приведено свободное изложение этих алгоритмов из замечательной недавней книги Дориго и Томаса Штюцле (*Thomas Stützle*): Marco Dorigo and Thomas Stützle, 2004, *Ant Colony Optimization*, MIT Press.

А где муравьи? А вот где. Алгоритм ОМК появился под влиянием более ранних исследований муравьиных алгоритмов сбора пищи и поиска пути на основе феромонов, хотя связь между ОМК и настоящими муравьями ... весьма тонка. Люди, применяющие ОМК любят рассказывать следующую историю о решении задачи коммвояжера: возьмем муравья, высадим его в Сиэтле и отправим гулять по графу, от города к городу, в результате чего рано или поздно образуется цикл. Муравей выбирает ребра (маршруты из текущего города в соседние города), которые имеют наибольший феромон и относительно хорошую (низкую) цену. После перемещения по маршруту, на нем остается некоторое фиксированное количество феромона. Если муравьиная тропа короткая (имеет меньшую цену), то естественно, что феромон будет распределен вдоль ее ребер более плотно, что повышает их привлекательность для других муравьев.

Такая вот история. На самом деле муравьев нет, а есть только компоненты с их историческим качеством («феромоном») и потенциальные решения, сформированные из этих компонент («тропы»), с присвоенными значениями приспособленности, которые в дальнейшем будут перераспределены по соответствующим компонентам.

8.3.1 Алгоритм муравья

Первая версия алгоритма ОМК являлась **алгоритмом муравья** (**AM**) (***Ant System, AS***). В настоящее время этот алгоритм используется нечасто, однако на его примере удобно излагать основные понятия. Решения в алгоритме муравья формируются на основе подобия пропорциональной селекции, учитывающей как цену или ценность, так и феромоны (объяснение см. далее). Величина приспособленности *всегда* прибавляется к феромону компонент, и, чтобы избежать неограниченного роста, каждый раз производится *уменьшение* (**испарение**) феромона.

Основные 5 шагов алгоритма муравья:

1. Формирование некоторых троп (потенциальных решений) путем выбора компонент.
2. (опция) Локальное улучшение троп.
3. Вычисление приспособленности полученных троп.
4. Небольшое испарение феромона.
5. Обновление феромона компонент, использованных при формировании троп на основе их приспособленности.

В оригинальном AM локальное улучшение отсутствует, здесь оно добавлено мной. В последующих версиях ОМК оно появилось. Вот вариант алгоритма (заметьте некоторую схожесть с алгоритмом GRASP):

Ценности или цены компонент и выбор компонент Тропа формируется путем итеративного выбора среди компонент, добавление которых не приведет к недопустимости тропы. Для задачи о ранце это легко обеспечить: нужно просто выбирать блоки до тех пор, пока оставшиеся блоки не будут переполнять ранец. Но для задачи коммивояжера ситуация более сложная. Например, в TSP можно выбирать ребра до тех пор, пока не будет получен полный тур, однако в этом случае есть риск получить тур с неиспользуемыми ребрами, либо чрезмерно усложнить сам тур. При другом подходе можно начать с некоторого города и выбирать среди ребер, ведущих в ранее не посещенный город (если есть такая возможность), затем выбирать среди ребер, инцидентных *этому* городу и т.д. Однако бывает, что оптимальный тур требует повторного посещения ряда городов. Кроме этого, что если для формирования тура обязательным является перелет из Солт-Лейк Сити в Денвер, который имеет очень высокую цену (и низкую ценность), поэтому алгоритм будет выбирать более дешевые варианты, но рано или поздно ему придется выбрать и это ребро? Таким образом, можно получить весьма отвратительные туры. Как бы то ни было, суть в том, что формирование тропы может потребовать предварительных размышлений.

В AM используется величина, которую я называю **желательностью** (***desirability***) компоненты, комбинирующей ценность и величину феромона:

$$\text{Желательность}(C_i) = p_i^{\delta} \times (\text{Ценность}(C_i))^{e},$$

либо если в задаче используется цена

$$\text{Желательность}(C_i) = p_i^{\delta} \times (\frac{1}{\text{Ценность}(C_i)})^{e},$$

Алгоритм 110 Алгоритм муравья (АМ)

$C \leftarrow \{C_1, \ldots, C_n\}$ компоненты
$e \leftarrow$ константа для испарения, $0 < e \leq 1$
popsize $\leftarrow$ количество троп, формируемых на каждой итерации
$\gamma \leftarrow$ начальные величины феромонов
$t \leftarrow$ количество итераций для локального поиска

$\overrightarrow{p} \leftarrow \langle p_1, \ldots, p_n \rangle$ феромоны компонент, изначально равные γ
$Best \leftarrow \square$
repeat
  $P \leftarrow \{\}$ {Исходные тропы (потенциальные решения).}
  {Формирование троп.}
  for *popsize* раз **do**
    $S \leftarrow \{\}$
    repeat
      $C' \leftarrow$ компоненты из $C - S$, которые можно добавить и сохранить решение допустимым
      if C' пусто **then**
        $S \leftarrow \{\}$ {Еще одна попытка.}
      else
        $S \leftarrow S \cup$ {компонента, выбранная из C' на основе величин феромонов и ценности или цены}
      end if
    until S – полное решение
    $S \leftarrow$ Локальный поиск HillClimb(S) в течение t итераций {Опция, по умолчанию не выполняется.}
    ВычислениеПриспособленности(S)
    if $Best = \square$ или Приспособленность(S) > Приспособленность($Best$) **then**
      $Best \leftarrow S$
    end if
    $P \leftarrow P \cup \{S\}$
  end for
  for каждое значение $p_i \in \overrightarrow{p}$ **do**
    $p_i \leftarrow (1 - e)p_i$ {Небольшое уменьшение феромона («испарение»).}
  end for
  for $P_j \in P$ **do**
    for каждая компонента C_i **do**
      if C_i использовалась в P_j **then**
        $p_i \leftarrow p_i +$ Приспособленность(P_j)
      end if
    end for
  end for
until $Best$ – идеальное решение, или закончилось время поиска
return $Best$

δ и e – настроечные параметры[10]. Заметим, что при увеличении феромона качества также растет, так же, как и при увеличении ценности (уменьшении цены). В результате в АМ можно применять пропорциональную селекцию среди компонент, основанную на «желательности», как в Алгоритме 30. При желании можно использовать и другой вариант селекции, например, турнирную или наподобие усечения до $p\%$ как в алгоритме GRASP, тоже использующие желательность.

Инициализация феромонов Можно установить все значения $\gamma = 1$. Для задачи TSP исследователи АМК часто используют $\gamma = popsize \times (1/\text{Цена}(D))$, где D некоторые очень дорогой бессмысленный тур, вроде тура по принципу ближайшего соседа (тур для TSP конструируется жадным образом, всегда выбирая ребро с наименьшей ценой).

Испарение феромонов В алгоритме муравья испарение феромонов необходимо, чтобы избежать их неограниченного увеличения. Но есть и более хорошее средство: величина феромона увеличивается либо уменьшается в зависимости от среднего качества. Т.е. вместо моделирования испарения и обновления, продемонстрированных в алгоритме муравья, можно скорректировать величину p феромона следующим образом:

Алгоритм 111 Обновление феромона с учетом скорости обучения

$C \leftarrow \{C_1, \ldots, C_n\}$ компоненты
$\overrightarrow{p} \leftarrow \langle p_1, \ldots, p_n \rangle$ феромоны компонент
$P \leftarrow \{P_1, \ldots, P_m\}$ компоненты
$\alpha \leftarrow$ скорость обучения

$\overrightarrow{r} \leftarrow \langle r_1, \ldots, r_n \rangle$ общая желательность компонент, изначально 0
$\overrightarrow{c} \leftarrow \langle c_1, \ldots, c_n \rangle$ счетчики использования компонент, изначально 0
{Вычисление общей желательности компонент и количества их использований.}
for $P_j \in P$ **do**
  for каждая компонента C_i **do**
    if C_i использовалась в P_j **then**
      $r_i \leftarrow r_i +$ Желательность(P_j)
      $c_i \leftarrow c_i + 1$
    end if
  end for
end for
for каждое значение $p_i \in \overrightarrow{p}$ **do**
  if $c_i > 0$ **then**
    $p_i \leftarrow (1-\alpha)p_i + \alpha \frac{r_i}{c_i}$ $\{\frac{r_i}{c_i}$ – средняя желательность.$\}$
  end if
end for
return $\overrightarrow{p}$

$0 \leq \alpha \leq 1$ – **скорость обучения** (***learning rate***). В алгоритме для каждой компоненты вычисляется средняя желательность, в зависимости от приспособленности троп, в которых эта компонента была использована. Затем отбрасывается часть уже имеющейся информации (за счет умножения на $1-\alpha$) и добавляется новая, полученная на текущей итерации, о том, насколько хороша компонента (пропорционально α). При больших α новая информация усваивается быстрее, в ущерб исторически накопленной. Возможно, лучше выбирать малые значения для α[11].

Опциональный локальный поиск: увеличение эксплуатации По умолчанию в АМ локальный поиск отсутствует. Однако его можно использовать применительно к муравьиной тропе S сразу же после этапа ВычислениеПриспособленности, так же, как в алгоритме GRASP. И здесь так же приходится сталкиваться со все той же проблемой: необходимостью гарантировать, что в результате операции Tweak для улучшения тропы полученный потомок будет допустимым. Для одних задач это не составляет труда, а для других – сложно осуществить. В любом случае, локальный поиск

[10] Их использование не является жестко ограниченным. Например, можно было записать Желательность$(C_i) = p_i^{\delta} + (\text{Ценность}(C_i))^e$ или Желательность$(C_i) = \delta p_i + (1-\delta)$Ценность$(C_i)$

[11] Метафора скорости обучения в форме $1-\alpha$vs.α будет встречаться при обсуждении Систем обучающихся классификаторов, а также является общим понятием для обучения с подкреплением.

усиливает эксплуатационные свойства алгоритма, направляя его прямиком к локальному решению. Это часто оказывается преимуществом для задач вроде TSP, которым эксплуатация обычно идет только на пользу.

8.3.2 Алгоритм муравьиной колонии: Более эксплуатирующая версия алгоритма

С момента, когда был представлен алгоритм муравья, было предложено множество его усовершенствованных версий (ряд которых был рассмотрен ранее). Здесь я расскажу о наиболее известной: **алгоритме муравьиных колоний** (***Ant Colony System, ACS***)[12]. АМК похож на АМ, но имеет следующие отличия:

1. При обновлении феромонов применяется **элитаризм**: повышается величина феромона только для тех компонент, которые использованы в лучшем найденном решении. В некотором смысле, это похоже на схему $(1 + \lambda)$.
2. Использование скорости обучения при обновлении феромонов.
3. Немного иной способ испарения феромонов.
4. Сильная тенденция к использованию компонент, присутствующих в наилучшем найденном решении.

Элитаризм В АМК обновляются только феромоны компонент, использованных в лучшей найденной тропе (которая хранится в переменной $Best$), с применением обновления скорости обучения по методу, подсмотренному в алгоритме 111. Т.е. если компонента используется в лучшей тропе, то величина ее феромона увеличивается согласно: $p_i \leftarrow (1 - \alpha)p_i + \alpha$Приспособленность$(Best)$.

Такой подход обладает существенными эксплуатирующими свойствами, поэтому величина всех феромонов еще и *сокращается*, вне зависимости от их использования в наилучшем решении, чтобы уменьшить желательность их использования в будущих решениях, чтобы дать алгоритму возможность немного больше исследовать пространство поиска. В частности, если компонента C_i используется в решении, то ее феромон обновляется по правилу: $p_i \leftarrow (1 - \beta)p_i + \beta\gamma$, где β нечто вроде испарения или «скорости разобучения», а γ – начальное значение феромона. Если не вмешиваться, то со временем величина всех феромонов сойдется к γ.

Элитарный отбор компонент Выбор компонент также обладает сильными эксплуатационными свойствами. С вероятностью q подбрасывается монета, и если выпадет орел, то выбирается компонента с наибольшей желательностью. В противном случае выбор осуществляется как и в АМ, хотя в АМК этот алгоритм упрощен, путем избавления от δ (полагается равной 1).

Итак, определим алгоритм муравьиной колонии, который по структуре не так уж сильно отличается от АМ:

Как и ранее было бы нелишним осуществить локальный поиск сразу после шага ВычислениеПриспособленности.

На этом этапе алгоритм ОМК может показаться несколько странным. Используется жадный выбор компонент для добавления в потенциальное решение, на основании того, как эта компонента проявила себя в высококачественных решениях (а может быть лишь в *наилучшем*). Здесь не рассматривается возможность того, что компонента должна всегда сочетаться с другой, иначе ее качество ухудшится, т.к. без второй компоненты оно может оказаться ужасным. Т.е. ОМК *полностью игнорирует связанность между компонентами.*

Это очень смелое допущение, которое теоретически ведет к проблемам, встречаемым в коэволюционных алгоритмах: появлению мастеров на все руки. АМК пытается обойти это ограничение сильно полагаясь на наилучшее найденное решение, аналогично коэволюционным подходам на основе выбора лучшего из n варианта и с использованием архива, которые пытаются анализировать компоненту в контексте ее лучшего результата. На мой взгляд, в ОМК есть много общего с коэволюцией, хотя такая точка зрения не была подробно исследована. В некотором смысле можно рассматривать ОМК как алгоритм **псевдо-кооперативной коэволюции с одной популяцией** (***one population pseudo-cooperative coevolution***).

[12] И снова авторами являются Марко Дориго и Люка Гамбарелла. Не подумайте чего, в ОМК помимо Дориго работает еще много людей!

Алгоритм 112 Алгоритм муравьиной колонии (АМК)

$C \leftarrow \{C_1, \ldots, C_n\}$ компоненты
$popsize \leftarrow$ количество троп, формируемых на каждой итерации
$\alpha \leftarrow$ скорость обучения для элиты
$\beta \leftarrow$ скорость испарения
$\gamma \leftarrow$ начальные величины феромонов
$\delta \leftarrow$ настроечный параметр для эвристик для отбора компонент {Обычно $\delta = 1$.}
$t \leftarrow$ количество итераций для локального поиска
$q \leftarrow$ вероятность выбора элитарной компоненты

$\overrightarrow{p} \leftarrow \langle p_1, \ldots, p_n \rangle$ феромоны компонент, изначально равные γ
$Best \leftarrow \square$
repeat
  {Формирование троп.}
  for *popsize* раз **do**
    $S \leftarrow \{\}$
    repeat
      $C' \leftarrow$ компоненты из $C - S$, которые можно добавить и сохранить решение допустимым
      if C' пусто **then**
        $S \leftarrow \{\}$ {Еще одна попытка.}
      else
        $S \leftarrow S \cup \{$компонента, выбранная из C' на основе элитарного отбора компонент$\}$
      end if
    until S – полное решение
    $S \leftarrow$ Локальный поиск HillClimb(S) в течение t итераций {Опция, по умолчанию не выполняется.}
    ВычислениеПриспособленности(S)
    if $Best = \square$ или Приспособленность(S) > Приспособленность($Best$) **then**
      $Best \leftarrow S$
    end if
  end for
  for каждое значение $p_i \in \overrightarrow{p}$ **do**
    $p_i \leftarrow (1 - \beta)p_i + \beta\gamma$ {Небольшое уменьшение феромона («испарение»).}
  end for
  {Обновление феромонов только для компонент из $Best$.}
  for каждая компонента S_i **do**
    if S_i использовалась в $Best$ **then**
      $p_i \leftarrow (1 - \alpha)p_i + \alpha$Приспособленность($Best$)
    end if
  end for
until $Best$ – идеальное решение, или закончилось время поиска
return $Best$

Вероятно интересным является другое применение ОМК, в котором популяция содержит не компоненты, а (допустим) все возможные пары компонент. Можно выбирать из них качественные пары, что должно помочь при учете связанности компонент, хотя потребует сущенно большей популяции.

ОМК имеет также много общего с алгоритмами оценки распределений с одной переменной (обсуждается в Разделе 9.2)[13] Вот почему так получается: приспособленности компонент можно рассматривать как **вероятности**, тогда вся популяция представляет **распределение вероятностей** в компонентном базисе. Сравните с эволюционной моделью, в которой популяцию можно считать выборочным распределением из *совмещенного* пространства всех возможных потенциальных решений, другими словами, из пространства всех возможных *комбинаций* компонент. Очевидно, что по сравнению с эволюционной моделью ОМК работает в существенно более простом (возможно, упрощенном) пространстве. Для общих задач оптимизации это может порождать проблемы, но во многих комбинаторных задачах такой подход часто является оправданным.

8.4 Управляемый локальный поиск

Существует и другой способ воспользоваться преимуществом организации компонентного пространства поиска в задачах комбинаторной оптимизации: можно помечать определенные компоненты, использование которых часто ведет к локальным оптимумам, и избегать их в дальнейшем.

Вспомним, что в **Поиске с запретом, основанном на параметрах**, (Алг. 15, Раздел 2.5) определялись значения параметров, присутствующие в хороших решениях, на использование которых в дальнейшем объявлялось «табу», в результате которого эти значения временно не использовались функцией Tweak. Идея метода заключалась в том, чтобы предотвратить возвращение алгоритма в одни и те же локальные экстремумы, для которых характерны найденные значения параметров.

В случае, если удается создать хороший замкнутый оператор Tweak, можно легко адаптировать поиск с запретом, основанный на параметрах, к решению комбинаторных задач, путем простой замены «параметров» компонентами. К примеру, можно проводить локальный поиск для задачи коммивояжера с помощью такого метода, если запретить использовать некоторые хорошие ребра, чтобы заставить алгоритм выйти из локального оптимума.

Разновидность поиска с запретом, основанного на параметрах, названная **Управляемый локальный поиск** (**УЛП**) (***Guided Local Search, GLS***), выглядит отлично подходящей для решения задач комбинаторной оптимизации: в ней компонентам присваиваются величины «исторического качества» (*historical quality*), как в алгоритме оптимизации муравьиной колонией. Но что интересно, данный алгоритм применяет полученную информацию о качестве не для того, чтобы определить и использовать наилучшие компоненты, а чтобы обозначить ряд проблемных компонент запретными и принудительно усилить исследование пространства поиска.

Алгоритм УЛП разработан Крисом Вудорисом (*Chris Voudouris*) и Эдвардом Цангом (*Edward Tsang*)[14]. Сам по себе алгоритм представляет вариант локального поиска, в котором производится попытка найти компоненты, который слишком часто соответствуют локальному оптимуму, и в результате решения, использующие такие компоненты, штрафуются, чтобы заставить усилить исследовательскую составляющую поиска.

Для этого в управляемом локальном поиске используется вектор феромонов[15], по одному на компоненту, чтобы определить, как часто каждая компонента встречается в хороших решениях. Вместо локального поиска по функции Качество в УЛП применяется функция СогласованноеКачество, учитывающая как Качество, так и величины феромонов[16]. Для данного потенциального решения S и набора компонент C из рассматриваемой задачи, а также вектору $\overrightarrow{p}$ феромонов, по одному элементу на каждую компоненту, согласованное качество определяется как:

$$\text{СогласованноеКачество}(S, C, \overrightarrow{p}) = \text{Качество}(S) - \beta \sum_i \begin{cases} p_i, & \text{если компонента } C_i \text{ присутствует в } S, \\ 0, & \text{в противном случае.} \end{cases}$$

[13] Этот факт был замечен ранее, т.е. не только мной, см. с. 57 в Marco Dorigo and Thomas Stützle, 2004, Ant Colony Optimization, MIT Press. Таким образом, есть сходство и с коэволюционными алгоритмами, и с АОР ...хммм ....

[14] Одно из ранних описаний алгоритма встречается в отчете Chris Voudouris and Edward Tsang, 1995, Guided local search, Technical Report CSM-247, Department of Computer Science, University of Essex. Позднее этот отчет был доработан в виде публикации Chris Voudouris and Edward Tsang, 1999, Guided local search, *European Journal of Operational Research*, 113(2), 469–499

[15] В данном случая я заимствую терминологию из ОМК, в УЛП используется термин **штрафы** (***penalties***).

[16] Ради единообразия я отхожу от стандартной формулировки задачи для УЛП, который традиционно решает задачи минимизации, а не максимизации.

Таким образом, алгоритм ищет высококачественные решения, которые являются относительно *новыми*, т.е. используют компоненты, редко встречающиеся в хороших решениях до этого. В данном контексте большие величины феромонов – это *плохо*. Параметр β определяет вклад новизны в вычисление результирующего качества и его необходимо настраивать осторожно.

После этапа локального поиска в полученном пространстве с согласованном критерием качества алгоритм увеличивает величины феромонов для компонент, используемых в текущем потенциальном решении S, которое предположительно располагается в окрестности локального оптимума. Вероятность увеличения феромона компоненты зависит от трех свойств. Во-первых, компонента должна присутствовать в текущем решении, т.е. она несет частичную ответственность за попадание в локальный оптимум и ее следует в некоторой степени избегать. Во-вторых, компонента должна обладать меньшей ценностью или большей ценой, т.к. первым делом в решении избавляются от менее важных компонент. В-третьих, у нее должно быть меньше феромона. Дело в том, что в УЛП одна и та же компонента не штрафуется вечно, алгоритм переключает внимание на другие компоненты, для исследования пространства поиска. Поэтому если величина феромона компоненты достаточно велика, она более не будет повышаться. Миром правит любовь!

Для определения компонент, величина феромона которых должна увеличиться, в УЛП сначала вычисляется *штрафуемость* (*penalizability*) каждой компоненты C_i в зависимости от текущей величины феромона p_i[17]:

$$\text{Штрафуемость}(C_i, p_i) = \frac{1}{(1 + p_i) \times \text{Ценность}(C_i)}$$

. . . либо если в задаче используются цены . . .

$$\text{Штрафуемость}(C_i, p_i) = \frac{\text{Цена}(C_i)}{(1 + p_i)}.$$

Управляемый локальный поиск выбирает компоненту с *наибольшей* штрафуемостью, среди *всех компонент в текущем решении* S, и увеличивает ее феромон на 1. Если таких компонент больше одной (одинаковое значение штрафуемости), то у них всех изменяется величина феромона.

Сравнивая функцию штрафа с функцией желательности из Раздела , отметим, что компоненты с высоким значением желательности обычно имеют низкую штрафуемость и обратно. В то время как ОМК старается сконструировать новое потенциальное решение из исторически желаемых компонент, УЛП штрафует компоненты, часто появляющиеся в локальных оптимумах, хотя сильнее всего штрафуются те, у которых желательность меньше.

Итак, зная способ согласования качества решений, основанный на величинах феромонов, и способ увеличения феромонов компонент, часто присутствующих в локальных оптимумах, алгоритм целиком становится достаточно очевидным: это обычный локальный поиск с дополнительным случайным изменением текущих феромонов компонент. И никакого испарения (что весьма удивительно!).

В алгоритме управляемого локального поиска не определяется, как выяснить, что алгоритм застрял в локальном оптимуме и необходимо модифицировать феромоны, чтобы выбраться оттуда. Как правило, общепринятого теста на локальную оптимальность нет, поэтому я заимствовал подход из Алг. 10 (Локальный поиск со случайными перезапусками, Раздел 2.2), в котором локальный поиск ограничен случайным таймером, после чего производится обновление феромонов, в предположении, что отведенного времени было достаточно, чтобы глубоко засесть в локальном оптимуме.

[17] В УЛП традиционно используется слово *полезность* (*utility*) вместо выдуманной мной *штрафуемости*. Однако полезность – это очень перегруженный смыслами термин, который обычно имеет совсем другое значение, см. например Раздел 10, поэтому я стараюсь его избегать.

Алгоритм 113 Управляемый локальный поиск (УЛП) со случайными обновлениями

$C \leftarrow \{C_1, \ldots, C_n\}$ набор компонент, которые могут составлять потенциальное решение
$T \leftarrow$ распределение для временных интервалов

$\overrightarrow{p} \leftarrow \langle p_1, \ldots, p_n \rangle$ феромоны компонент, изначально равные 0
S начальное потенциальное решение
$Best \leftarrow S$
repeat
  $time \leftarrow$ случайное время в недалеком будущем, распределенное согласно T
  {Сначала осуществляется локальный поиск в пространстве компонент с согласованным качеством.}
  repeat
    $R \leftarrow$ Tweak(Копия(S))
    if Качество(R) $>$ Качество($Best$) **then**
      $Best \leftarrow R$
    end if
    if СогласованноеКачество($R, C, \overrightarrow{p}$) $>$ СогласованноеКачество($S, C, \overrightarrow{p}$) **then**
      $S \leftarrow R$
    end if
  until $Best$ – идеальное решение, или закончилось время $time$, либо полное время поиска
  $C' \leftarrow \{\}$
  for каждая компонента $C_i \in C$, присутствующая в S **do**
    {Поиск наиболее штрафуемых компонент.}
    if для всех $C_j \in C$, присутствующих в S: Штрафуемость(C_i, p_i) $\geq$ Штрафуемость(C_j, p_j) **then**
      $C' \leftarrow C' \cup \{C_i\}$
    end if
  end for
  for каждая компонента $C_i \in C$, присутствующая в S **do**
    {Штрафование компонент путем увеличения их феромона.}
    if $C_i \in C'$ **then**
      $p_i \leftarrow p_i + 1$
    end if
  end for
until $Best$ – идеальное решение, или закончилось время поиска
return $Best$