Лекция 14

Роевые алгоритмы. Задача о воре.

План лекции

- Роевые алгоритмы. Алгоритмы муравьиной колонии
  - Идея
  - Основные принципы работы
- Задача о воре (Travelling Thief Problem)

Роевые алгоритмы

Роевые алгоритмы (также «роевой интеллект», англ. „Swarm Intelligence“) - мультиагентные эвристические алгоритмы оптимизации.

Базовые принципы:

- Потенциальное решение задачи представляется в виде набора *элементов*.
- Коллектив агентов, каждый самостоятельно строит решение.
- Агенты обмениваются информацией, «помечая» перспективные элементы.

Т.е. РА — также представляют собой вариант случайного поиска с параллельным анализом множества решений.

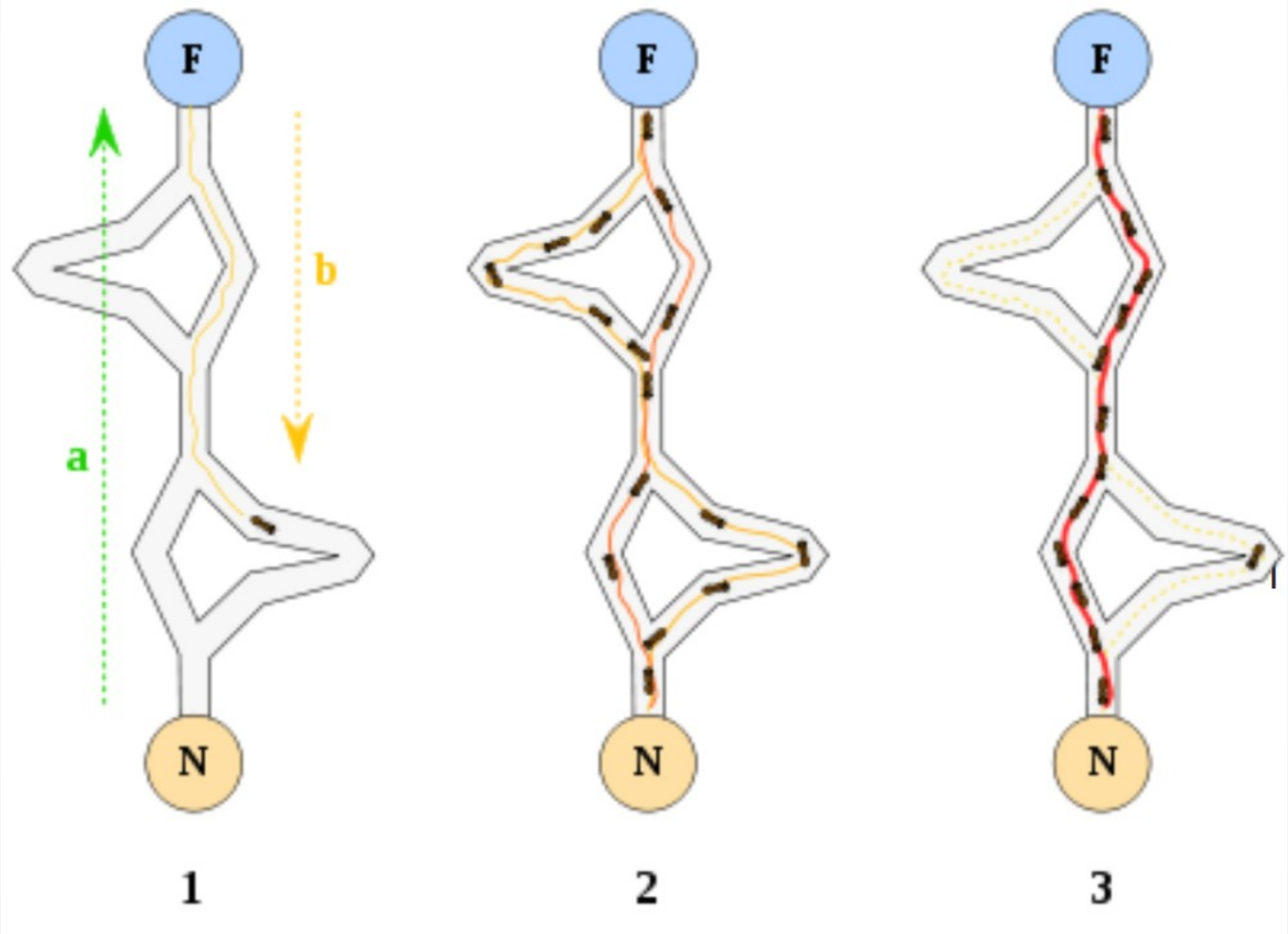
Наиболее популярный вариант: алгоритм муравьиной колонии/муравьиный алгоритм (Ant Colony Optimization, ACO).

Муравьиный алгоритм

- Каждый ***муравей*** изначально ищет путь от гнезда к источнику пищи самостоятельно (случайно).
- Проходя по пути, муравей помечает его ***феромоном***.
- Чем короче путь, тем быстрее его проходят муравьи, выбравшие этот путь => тем больше феромона останется на этом пути.
- Находясь на развилке, муравей с большей вероятностью выберет тот путь, на котором больше феромона.
- Феромон испаряется с течением времени. Тем самым а) снижается влияние случайности, и б) алгоритм способен адаптироваться к изменениям в исходных данных.

Муравьиный алгоритм

Идея



Муравьиный алгоритм

Муравьиная *метаэвристика* применяется для решения оптимизационных задач (канонический вариант - в форме минимизации), в которых решение представляется в виде последовательности/вектора значений (*компонент*). Могут присутствовать ограничения на допустимые значения последовательностей.

Перебор потенциальных решений часто представляют в виде графа, на котором компоненты решения представляют собой дуги или вершины, а допустимые решения — пути.

Естественный вариант такой задачи: задача коммивояжёра. Компоненты решения = дуги графа.

Муравьиный алгоритм

Общая схема:

1. Инициализация
   - Создать колонию *муравьёв*
   - Поместить на дугах небольшое количество *феромона*
2. Пока не выполнено условие остановки:
   - Построить решения
   - (опционально) Улучшить решения с помощью локального поиска
   - Изменить концентрацию феромона.

Разработано много вариантов реализации этой схемы. Самый первый - «муравьиная система» (Ant System).

Ant System

Фаза «Построить решения»

Каждый муравей начинает из некоторой вершины графа и строит гамильтонов цикл/путь, перемещаясь из текущей вершины *i* в смежную с ней вершину *j* с вероятностью

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}$$

Ant System

Фаза «Построить решения»

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}$$

$\tau_{i,j}$ - количество феромона на дуге (*i,j*)

$\eta_{i,j}$ - «эвристическая оценка» («привлекательность») дуги (*i,j*); например = 1/c(*i,j*)

Параметры α и β регулируют влияние эвристической оценки (при α=0 алгоритм превращается в жадный) и коллектива (при β=0 муравей учитывает только опыт других муравьёв).

Ant System

Фаза «Изменить концентрацию феромона»

Каждый муравей *k* рассчитывает стоимость найденного решения (L_k). Концентрация феромона на каждой дуге (i,j) изменяется по формуле:

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \sum_k \Delta\tau_{i,j}(k)$$

$\tau_{i,j}$ - концентрация феромона на дуге (i,j)

ρ - скорость испарения феромона

$\Delta\tau_{i,j}(k)$ - рассчитывается как Q/L_k для дуги, входящей в *k*-е решение, и 0 для остальных дуг

Задача о воре

Недостатки большинства классических алгоритмов (как точных, так и приближённых): они слишком «привязаны» к специфике конкретной задачи. Практические задачи как правило не совпадают на 100% с одной из классических оптимизационных задач (задача о рюкзаке, задача коммивояжёра и т. д.), но часто могут быть сведены к ***набору взаимосвязанных*** подзадач.

Две важные характеристики:

- Комплексность — реальные проблемы часто состоят или нескольких (двух или более) подзадач, каждая со своим ограничением или критерием оптимальности.
- Взаимозависимость — ограничения и критерии оптимальности одной подзадачи зависят от решения другой подзадачи.

Пример: задача о проектировании печатных плат (Printed Circuit Board).

Задача о воре

Для того чтобы при разработке и сравнении алгоритмов можно было учесть эти характеристики, в качестве модельной была предложена [Bonyadi, Michalewicz, Barone, 2013] задача о воре (Travelling Thief Problem, TTP).

Имеется n городов (заданы попарные расстояния между ними d_{ij}). В этих городах расположены m предметов; для каждого предмета задан его вес w_k и стоимость c_k.

У вора имеется рюкзак вместимостью b.

Задача: обойти все города по одному разу, «собрав» (=украв) в них предметы таким образом, чтобы суммарный вес предметов не превышал b.

Цель (критерий оптимизации) можно установить разными способами. Предложены два варианта (TTP_1, TTP_2).

Задача о воре

Задача TTP_1

Цель: максимизировать общий доход.

Доход складывается из стоимости украденных предметов за вычетом стоимости аренды рюкзака.

Стоимость аренды рюкзака определяется ценой за единицу времени (заданный входной параметр) и временем прохождения по маршруту. Время прохождения зависит от расстояния между городами и текущей скорости. В свою очередь, скорость перемещения вора зависит от текущего веса его добычи:

$$v_c = v_{max} - \mathrm{W}_c \frac{v_{max} - v_{min}}{b}$$

Задача о воре

Задача TTP_2

Двухкритериальная задача: максимизировать стоимость украденных предметов; минимизировать время прохождения по маршруту.

Скорость перемещения между городами определяется так же, как в TTP_1.

Дополнение: стоимость каждого предмета уменьшается с течением времени. Время считается с момента кражи данного предмета до момента окончания марширута.

Задача о воре

Доказано, что ни одна из задач не может быть эффективно решена путём последовательного решения подзадач (сначала построить оптимальный цикл, потом для него найти оптимальный набор предметов; или наоборот).

Для TTP_1: такое решение может быть сколь угодно далеко от оптимального решения полной задачи.

Для TTP_2: такое решение может не лежать во множестве Парето полной задачи.