Bi-Objective A*

Егор Горбачев & Александра Новикова

25 мая 2021

1 Вступление

Алгоритм $A*$ лежит в основе многих алгоритмов эвристического поиска для решения задач поиска кратчайшего пути. В таких задачах нужно найти путь с минимальной стоимостью от заданного начального состояния до заданного целевого состояния.

Однако в реальной жизни часто бывает несколько параметров стоимости пути. Например, правительственные учреждения, которые перевозят опасные материалы, должны найти маршруты не только с минимальным расстоянием, но ещё и с минимальным воздействием на жителей. Безусловно, эти параметры могут конфликтовать, поэтому необходимо находить компромисс.

И из-за этого алгоритм $A*$ был расширен для решения и оптимизации многокритериальных задач поиска кратчайшего пути, в которых надо найти набор оптимальных по Парето путей от начального состояния до целевого.

2 Формальная постановка задачи

- **Вход**
  На вход подаётся граф $G = (S, E, c, s_{start}, s_{goal})$.

  Всё по аналогии с классическими алгоритмами поиска:
  S - множество состояний графа;
  $E \subseteq S \times S$ — множество рёбер графа;
  s_{start}, s_{goal} — стартовое и целевое состояния соответственно.

  Отличается только функция стоимости рёбер:
  $c = (g_1, g_2) : E \to \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$.
  То есть на каждом ребре теперь написано две функции расстояния — g_1 и g_2.

- **Выход**
  На выход мы хотим получить множество оптимальных по Парето путей.

  Пояснение:
  Путь p_1 со стоимостью $c_1 = (a_1, b_1)$ доминирует путь p_2 со стоимостью $c_2 = (a_2, b_2)$ тогда и только тогда, когда $a_1 \leq a_2 \ \wedge \ b_1 \leq b_2 \ \wedge \ (a_1, b_1) \neq (a_2, b_2)$.

Множество оптимальных по Парето путей — это множество всех путей, которые не доминирует ни один другой путь.

3 Алгоритмы

Есть несколько алгоритмов, решающих данную задачу:

- Bi-Objective Dijkstra
- Bidirectional Bi-Objective Dijkstra
- NAMOA*
- NAMOA*dr
- BOA*

Наша главная задача написать и изучить алгоритм BOA* как потенциально наиболее эффективный.

Во всех алгоритмах требуется использовать монотонную эвристику, чтобы гарантировать нахождение истинного оптимального по Парето множества.

4 NAMOA*

Идея алгоритма NAMOA* заключается в следующем:
Вместо обычного $f - value$ у нас теперь пара (f_1, f_2).
Далее алгоритм производит действия, аналогичные классическому A*, но из OPEN состояния достаются в лексикографическом порядке по $f - value$.

Поддерживаем множество $sols$ - текущее множество никем не доминируемых путей в целевое состояние.

И также для каждой вершины s поддерживаем 2 множества с g-значениями: $G_{cl}(s)$ (которое состоит из g-значений раскрытых вершин) и $G_{op}(s)$ (которое состоит из g-значений вершин, которые были сгенерированы, но ещё не были раскрыты).

При обработке текущей вершины s и её ребёнка t с f-значением (f_1, f_2) мы пробегаемся по всем путям в $sols$ и сравниваем их с (f_1, f_2): если (f_1, f_2) кто-то оттуда доминирует, то просто пропускаем этот путь.

Также пробегаемся по $G_{op}(t) \cup G_{cl}(t)$ и если кто-то оттуда доминирует g_t, то тоже пропускаем этот путь.

В противном случае удаляем все пути из $G_{cl}(t)$ и $G_{op}(t)$ которые доминируются текущем путем через t и добавляем t в OPEN.

Соответственно, на каждом шаге доминирование для текущего пути проверяется за линейное от $|sols| + |G_{op}(t)| + |G_{cl}(t)|$ время

5 BOA*

Данный алгоритм является усовершенствованием NAMOA*. В нём проверка на доминирование происходит константное время вместо линейного.
Идея алгоритма заключается в следующем: мы так же, как и в NAMOA* обрабатываем вершины в лексикографическом порядке по f-значению.

Однако мы больше не храним $G_{cl}(s)$ и $G_{op}(s)$, а вместо этого отдельно для каждой вершины храним минимальное значение g по второй координате среди всех путей в множестве Парето, которое мы назовем $g_2^{min}(s)$ для каждой вершины s.

И тогда при очередной обработке вершины s мы знаем, что текущий путь точно не меньше всех предыдущих путей по первой координате (так как из OPEN мы достаём по f-значению в лексикографическом порядке, то есть первая координата всегда монотонно увеличивается).

Соответственно чтобы добавить этот путь в множество Парето, нам нужно только сравнить вторую координату с $g_2^{min}(s)$, что происходит за константное время.

6 Тестирование

Мы нашли отличный датасет. Там есть карты разных размеров и областей. На них мы и планируем сравнить все 5 алгоритмов многокритериального поиска, которые мы упомянули.

7 Распределение ролей

Алгоритмы BDijkstra, BBDijkstra, BOA* пишет Егор, тестирует Аля;
Алгоритмы NAMOA*, NAMOA*dr пишет Аля, тестирует Егор.

Анализ результатов будет происходить совместно.

8 Результаты работы

В ходе проекта были написаны и проанализированы все заявленные алгоритмы. Однако помимо этого мы еще столкнулись с той проблемой, что в датасете есть только сама карта, но нет заданий

и ответов к ним. Из этого вытекла необходимость написать решение полным перебором всех путей графа и тестирования его с алгоритмом BDijkstra. После того, как мы убедились, что они работают одинаково, мы использовали далее уже алгоритм BDijkstra, чтобы сравнивать с ним ответы других алгоритмов. Задания генерировались взятием двух случайных вершин графа.

Кроме того очень важно было выбрать правильные эвристики для решения задачи. Мы использовали варианты с расстояниями Евклида и Чебышева. Также мы рассматривали вариацию алгоритма BOA* без использования эвристики (что можно понимать как эвристику, равную нулю). Такой алгоритм мы назвали BOA*Dijkstra.

Эвристика Евклида показала себя лучше всех других на всех картах и алгоритмах. Она работает в среднем в 1.8 раз быстрее, чем эвристика Чебышева. BOA*Dijkstra работает еще в 1.2 раза медленнее, но, что удивительно, это весьма хороший результат. Стоит отметить, что BOA*Dijkstra — это далеко не то же самое, что и алгоритм BDijkstra, потому что BOA* сам по себе использует много методов по уменьшению времени работы вне зависимости от эвристики (константная проверка доминации и ранняя терминация).

Также мы столкнулись с той проблемой, что длины ребер графа иногда были короче евклидова расстояния между вершинами на плоскости, что, разумеется, ломает эвристики. Для решения этой проблемы мы воспользовались масштабированием: нашли максимальное отношение реального расстояния к длине ребра и домножили эвристику на это значение, в результате чего эвристика стала корректной.

Другой интересной проблемой явилось то, что если для расстояния эвристики нам известны, то для времени придумать эвристику совсем не просто. В итоге мы пришли к следующему решению: мы нашли максимальную скорость в городе (то есть максимум отношения длины ребра к времени проезда по ребру по всем ребрам), после чего эвристика времени бралась как эвристика расстояния, деленная на максимальную скорость. Очевидно, что эта эвристика корректна, потому что она является просто обычной эвристикой, домноженной на константу.

Также мы тестировали различные способы разрешения ничьих, но в итоге отказались от этого потому, что это не давало никакой разности во времени работы. Вероятно, это совсем не является узким местом алгоритма.

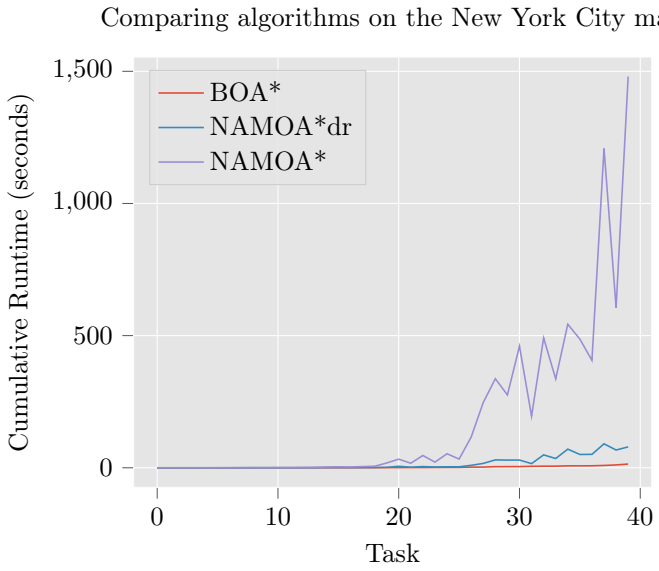
Представим результаты работы алгоритмов на картах различных городов США. Всего мы выбрали 3 карты: Нью-Йорк, Сан-Франциско (с окресностями) и Колорадо (штат). Первые две карты меньшего размера, вторая побольше.

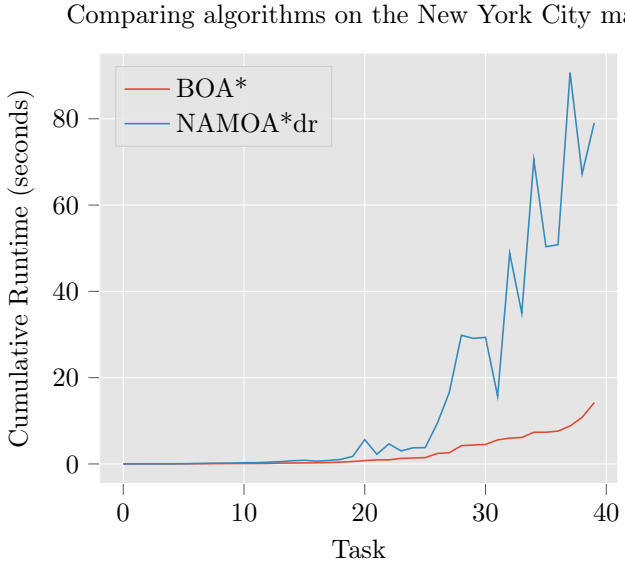
8.1 New York

Ниже представлены статистики времени работы различных алгоритмов на датасете из 40 случайных заданий (обе вершины выбирались случайно равновероятно из всех вершин графа). Все алгоритмы «точные», то есть обязательно находят правильный ответ, так что все алгоритмы выполнили все задания верно. На карте 264,346 вершин и 733,846 ребер. Средний размер множества Парето на этой карте — 167.

Algo \time per task	average	minimum	maximum	median	95 percentile	90 percentile	80 percentile
BDijkstra	1336.7185	0.00085	8712.546	95.661	6301.4324	4597.381	3640.29
BBDijkstra	1032.183	0.00261	5146.581	69.472	4973.119	2185.125	1749.288
NAMOA*	185.955	0.007113	1480.21	17.9997	1207.93	543.033	406.44
NAMOA*dr	16.3427	0.00657	90.6902	2.23327	79.0694	67.2311	34.9844
BOA*	2.54078	0.002363	14.1972	0.778755	10.8015	7.61164	5.99125

Самой важной характеристикой, безусловно, является среднее время работы, однако не стоит забывать и про другие статистики. К примеру, если выполняется большое количество запросов на сервере, то, разумеется, нас интересует в первую очередь среднее время работы, однако если задания выполняются на клиенте по одному, то бывает важно, чтобы каждое отдельное задание работало быстро. К примеру, 95% заданий выполняются не дольше 10 секунд (как у алгоритма BOA*).





Графики показывают время выполнения различных алгоритмов в зависимости от сложности этих заданий. Сложность задания определяется временем работы на самом эффективном алгоритме — BOA*. На втором графике убран алгоритм NAMOA*, чтобы можно было подробнее изучить разницу во времени работы более эффективных алгоритмов: BOA* и NAMOA*dr.

Можно заметить, что все алгоритмы кроме NAMOA*dr и BOA* невероятно долго работает, поэтому нами было принято решение не использовать их в анализе результатов далее, потому что

с одной стороны это займет очень много времени, а с другой стороны не даст никакой полезной информации из-за такого невероятного разрыва во времени работы. При этом можно заметить, что если задания очень простые (minimum и в районе его), BDijkstra справляется лучше всех просто из-за того, что в ней нет никаких эвристик и дополнительных проверок, на которые тратится время в эвристических алгоритмах. Однако когда задания становятся сложнее, алгоритм уже совсем не справляется.

8.2 San Francisco Bay Area

На этой карте мы тоже тестировали работу алгоритмов на 40 случайных заданиях (обе вершины выбирались случайно равновероятно из всех вершин графа). Эта карта по размеру больше Нью-Йорка, однако судя по результатам, она проще. Вероятно, дело в том, что достаточно большая часть карты — это окраины, на которые существует очень малое количество разных маршрутов, но это только гипотетическое предположение. На карте 321,270 вершин и 800,172 ребер. Средний размер множества Парето на этой карте — 148.

Algo \time per task	average	minimum	maximum	median	95 percentile	90 percentile	80 percentile
NAMOA*dr	9.46984	0.002613	78.6086	2.06969	63.1638	44.1086	14.66686
BOA*	2.33046	0.001109	16.8218	0.731742	12.5197	6.40846	4.72066

8.3 Colorado

На этой карте мы тоже тестировали работу алгоритмов на 40 случайных заданиях (обе вершины выбирались случайно равновероятно из всех вершин графа). Это большая карта целого штата. На этой карте относительная и абсолютная разница во времени работы NAMOA*dr и BOA* становится наиболее значительной. Если BOA* справился со всеми заданиями в течение 6 минут, то NAMOA*dr понадобился почти час. На карте 435,666 вершин и 1,057,066 ребер. Средний размер множества Парето на этой карте — 303.

Algo \time per task	average	minimum	maximum	median	95 percentile	90 percentile	80 percentile
NAMOA*dr	79.8516	0.008393	1973.48	6.61664	413.095	99.572	50.1545
BOA*	9.78043	0.005097	169.747	2.5116	62.2044	20.3503	9.54405

9 Выводы

Из результатов исследования становится очевидно, что алгоритм BOA* является самым быстрым на данный момент алгоритмом двукритериального поиска с сильным запасом. Он в несколько раз обгоняет NAMOA*dr по эффективности, а про остальные алгоритмы даже не приходится и говорить. Разумеется, в основном это достигается как раз благодаря тому, что проверка нового пути на доминирование текущим множеством Парето происходит в этом алгоритме за константное время вместо линейного. В случаях задач, в которых множество Парето имеет на порядок меньший размер, разница во времени работы этих алгоритмов будет уже не так заметна.

10 Ссылки

Основная статья - https://ojs.aaai.org/index.php/ICAPS/article/view/6655/6509

GitHub репозиторий - https://github.com/Peltorator/BIOA-Project