Алгоритмы конфликтно-ориентированного поиска для задачи многоагентного планирования: CBS, CBS+PC, CBS+H, CBS+DS.

Им Евгений, Парамонов Антон, Эмдин Григорий

Постановка задачи

Вход

На вход задаче подается граф G(V, E) и набор агентов A. Для каждого a_i агента есть стартовая и конечная вершины графа - s_i и f_i соответственно.

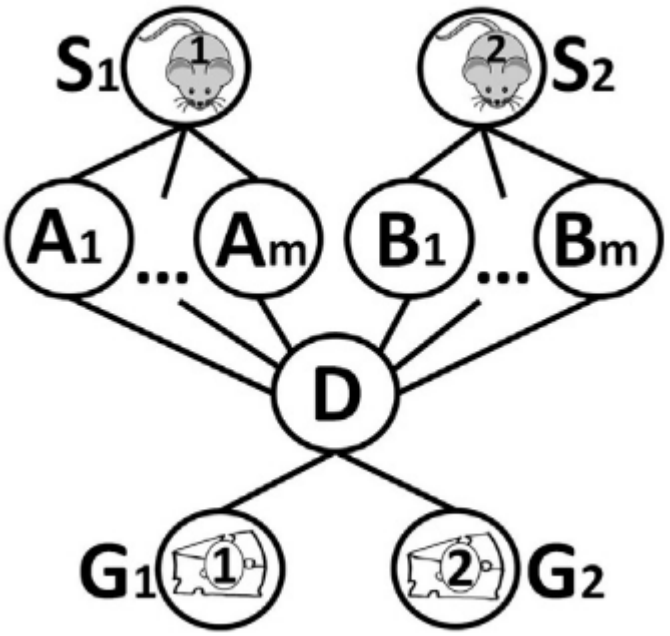
Выход

Для каждого агента алгоритм должен выдать маршрут в графе из начальной вершины агента в конечную его вершину, притом пути эти должны быть такими, чтобы во время движения агенты не столкнулись и суммарно потратили бы минимум времени.

Более формально

Будем считать время дискретным. В момент времени t_0 каждый агент a_i находится в своей стартовой вершине s_i.
Между последовательными моментами времени каждый агент может совершить действие: либо переместиться в соседнюю вершину графа, либо остаться на месте. Главное ограничение на действия агентов звучит так: в один момент времени двое агентов не могут находиться в одной вершине. Конфликтом будем называть ситуацию, когда главное ограничение нарушено.
С последовательностью действий естественным образом связано понятие времени, затрачиваемого на эту последовательность - это просто количество действий в последовательности.
Выходом алгоритма должен быть набор не конфликтующих друг с другом последовательностей действий агентов, приводящих каждого из них из его стартового состояния в конечное, и минимизирующий сумму по всем агентам времени, затрачиваемого ими на свою последовательность действий.

Пример



В данном примере есть две мышки-агента, стартующих из вершин S_1 и S_2 соответственно и желающих достичь каждая своего сыра (конечного состояния), в вершинах G_1 и G_2 соответственно.
Оптимальным путем для первой будет набор вершин {S_1, A_1, D, G_1}, для второй - {S_2, A_2, D, G_2}. Но тогда в момент времени t_2 обе мышки окажутся в вершине D, что нарушает главное ограничение. Поэтому оптимальными последовательностями действий будут {move A_1, wait, move D, move G_1} и {move B_1, move D, move G_2}, суммарное действие которых - 7. Заметим сразу, что “пропустить” могла и вторая мышка первую, так что оптимальных решений задачи может быть несколько.

Алгоритм conflict based search (CBS)

Введем ключевое понятие алгоритма - ограничение. **Ограничением** будем называть тройку (a, v, t), смысл которой - запрет агенту a находится в вершине v в момент времени t. **Конфликтом** будем называть четверку (a_i, a_j, v, t), ситуацию, когда агенты a_i и a_j находятся в вершине v одновременно в момент времени t.

Дерево ограничений

Алгоритм CBS строит дерево ограничений (constraint tree, CT) по следующему принципу. Вершина состоит из

- Набора ограничений
- Набора путей для агентов (приводящих каждого к цели, но, возможно, конфликтующих). Все наборы путей удовлетворяют всем ограничениям вершины

- Также в вершине будем хранить суммарное время ее путей

Если в вершине нет конфликта, то ее решение - оптимальное решение (это обеспечивается способом перебора вершин, который мы опишем далее). В противном случае пусть в вершине существует конфликт (a_i, a_j, v, t). Такой конфликт нужно разрешать, это можно сделать двумя способами: добавив ограничение (a_i, v, t) или (a_j, v, t), породив тем самым две дочерние вершины с множеством путей и ограничений, наследуемых от родителя + новое ограничение. Чтобы сохранить инвариант удовлетворения путей ограничениям, перестроим в дочерних вершинах решение для агента с новым ограничением (и, соответственно, пересчитаем суммарную стоимость). Корень дерева изначально не содержит ограничений, а решение для каждого агента - это просто оптимальное решение в предположении отсутствие других агентов. Раскрывать вершины будем в порядке возрастания их стоимости.

Таким образом алгоритм естественным образом разделяется на два процесса

Высокоуровневый процесс

Решающий задачи связанные с пророщением дерева, а конкретно:

- Поддержание множества периферийных вершин (кроны дерева), с быстрым доступом к минимальной по стоимости
- Поиск конфликта в вершине
- Порождение дочерних вершин с новыми ограничениями

Низкоуровневый процесс

Перестраивающий пути в вершине, чтобы они удовлетворяли ограничениям вершины. Это можно делать любым известным алгоритмом поиска (например, A*), но только на графе с дополнительным измерением - временем.

Псевдокод

```
Input: MAPF instance
Root.constraints = ∅
Root.solution = find individual paths by the low level()
Root.cost = SIC(Root.solution)
insert Root to OPEN
while OPEN not empty do
    P ← best node from OPEN  // lowest solution cost
    Validate the paths in P until a conflict occurs.
    if P has no conflict then
        return P.solution // P is goal
    C ← first conflict (a_i, a_j, v, t) in P
    foreach agent a_i in C do
        A ← new node
        A.constraints ← P.constraints + (a_i, v, t)
        A.solution ← P.solution
        Update A.solution by invoking low level(a_i)
        A.cost = SIC(A.solution)
        if A.cost < ∞ // A solution was found then
            Insert A to OPEN
```

Расширения

Существует ряд эвристических улучшений описанного алгоритма. Мы хотим рассмотреть и сравнить три подхода:

CBS + Prioritization of conflicts (PC)

В оригинальном алгоритме не проводится подробного анализа по вопросу выбора вершины для раскрытия в случае нескольких вершин лучшей стоимости. Представим, что мы выбрали для раскрытия вершину N, стоимостью C. Имеется два варианта событий при раскрытии:

- Оба сына после разрешения в них конфликтов имеют стоимость > C. В таком случае, мы просто ищем наиболее дешевую вершину в кроне (выбираем случайно в случае ничьей).
- У какого-то сына S стоимость также оказалась равной C. В таком случае мы будем раскрывать именно S, в приоритете над другими вершинами стоимости C.

CBS + H

Это расширение заточено на раскрытие менее конфликтных вершин посредством выбора не вершин с наименьшим суммарным временем, а вершин с наименьшим значением одной из следующих эвристик:

- h_1: Количество конфликтов
- h_2: Количество агентов с хотя бы одним конфликтом
- h_3: Количество пар конфликтующих агентов
- h_4: Размер вершинного покрытия для графа, вершинами в котором являются агенты, а ребрами соединены те из них, между которыми есть хотя бы один конфликт

Поскольку эвристики действуют жадно, у нас пропадает теоретическая гарантия на оптимальность результата.

CBS + Disjoint Splitting (DS)

Поскольку после раздвоения вершины могут существовать решения, удовлетворяющие ограничениям в обоих детях, их перебор потенциально будет производиться дважды, что неэффективно. Во избежание этого применяется disjoint splitting: разрешая конфликт (a_i, a_j, v, t), мы не просто запрещаем в левом сыне проходит агенту a_j через (v, t), но еще и обязываем проходить a_i через (v, t) в правом сыне. Это называется позитивным ограничением, в отличие от негативного (классического) ограничения о запрете в CBS.

Экспериментальное исследование

Мы планируем проводить замеры с использованием бенчмарков movingai.
Сравнивать планируется нашу имплементацию CBS с имплементацией, приведенной в оригинальной статье, а также наше решение без расширений против решений с расширениями.
Сравнивать с решением из статьи мы будем по показателю success rate - количеству успешно решенных задач за не более чем 5 минут для разного количества агентов.
Сравнивать расширения будем по времени работы, количеству корректных решений (специально для CBS+H) и количеству созданных узлов дерева.

План реализации

Реализовывать проект планируется на языке Python3.
На данный момент мы видим следующие этапы работы над проектом:

1. Реализация классического CBS
2. Реализация системы тестирования алгоритма на бенчмарках
3. Тестирование CBS на корректность на маленьких собственных примерах
   Тестирование CBS на корректность на больших бенчмарках
4. Сравнение нашей реализации CBS с реализацией в статье

   Пункты 1-4 планируется сделать до 11 мая

5. Реализация CBS + H
6. Реализация CBS + PC
7. Реализация CBS + DS

   Пункты 5-7 планируется сделать до 18 мая
8. Замеры времени работы, стоимостей решения и количества узлов дерева для расширений и самого CBS, отрисовка графиков, анализ результатов

   До 1 июня