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.

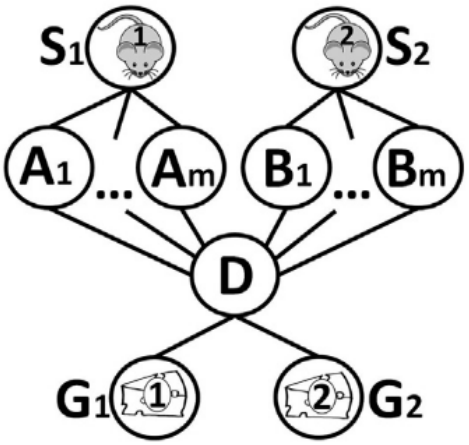
Выход

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.

Уточнения условий MAPF

В нашем проекте мы рассматривали задачу MAPF со следующими соглашениями

- Реберные конфликты учитываются
- Агенты не исчезают при попадании в конечную точку
- Агент может выйти из конечной точки
- Стоимость не начисляется с момента, когда агент не выходит из конечной точки
- 4-связная карта
- Агент может перейти в вершину, из которой только что ушел другой агент

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.

Расширения

Существует ряд эвристических улучшений описанного алгоритма. Мы рассмотрели и сравнили 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.

Введем понятие кардинального конфликта: конфликт называется кардинальным, если после создания детей у дерева ограничений с учетом этого конфликта у обоих детей стоимость пути увеличивается. Назовем конфликт полу-кардинальным, если в результате разделения стоимость пути у ровно одного увеличивается, а у другого нет. Если же стоимость у обоих детей не увеличивается, то конфликт будем называть не кардинальным.
Теперь по ходу алгоритма мы будем проверять конфликты на кардинальность, и в первую очередь рассматривать кардинальные, если таких нет то полу-кардинальные, а если таких нет, то не кардинальные.

CBS + H

Это расширение улучшает оценку стоимости вершины CT, тем самым позволяя проращивать потенциально более близкие к ответу вершины. Делается это за счет добавления к оригинальной *sum of costs* допустимой эвристики *h.* Эта эвристика считается следующим образом: строится граф, вершинами в котором являются агенты, а ребра проводятся для тех, что находятся в кардинальном конфликте. Теперь *h* = размер минимального реберного покрытия в этом графе, который считается жадно (такой подход сохраняет допустимость).

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.

Эксперименты

Benchmarks

Бенчмарки для замеров брались с
https://movingai.com/benchmarks/mapf

Мы рассматривали три карты:
- Berlin, размера 256x256
- empty, размера 8x8
- room, размера 32x32

Схема эксперимента

Чтобы изучить показатели конкретной модификации алгоритма, мы делали следующее:

Для каждой карты:
    Для количества *n* агентов из [5, 20]:
        Проделать 20 раз:
            Случайно просэмплирвоать *n* агентов
            Запуститься на этой случайной выборке

Метрики

Для каждого запуска мы отслеживали следующие метрики:

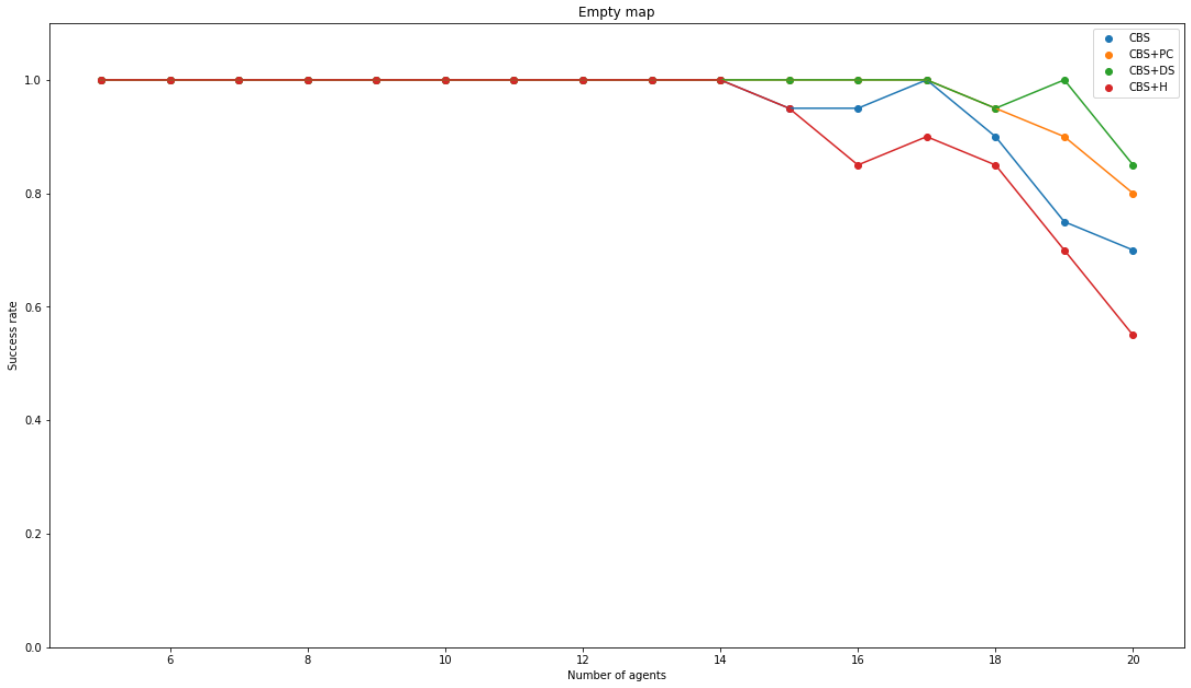
- Доля успешных* запусков среди сэмплов
- Среднее по сэмплам количество вершин Constraint Tree
- Среднее по сэмплам время работы

*под успешными мы понимаем запуски, завершившиеся за не более чем 5 минут

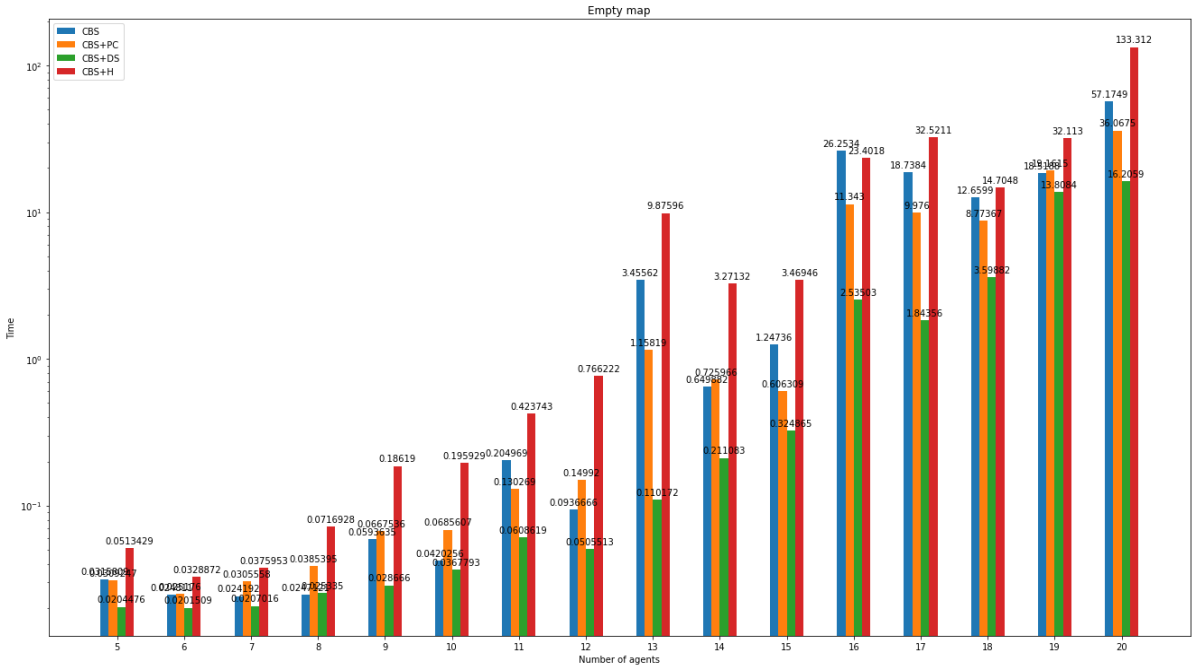
Результаты

Empty 8x8

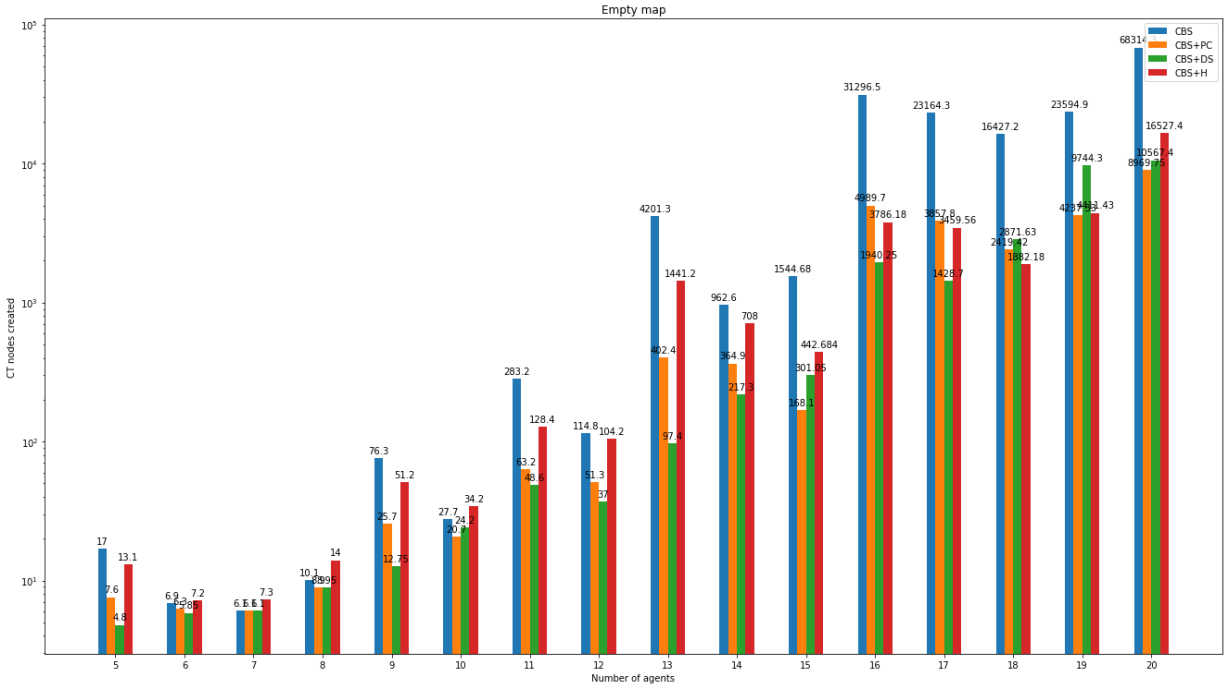
Success rate



Time



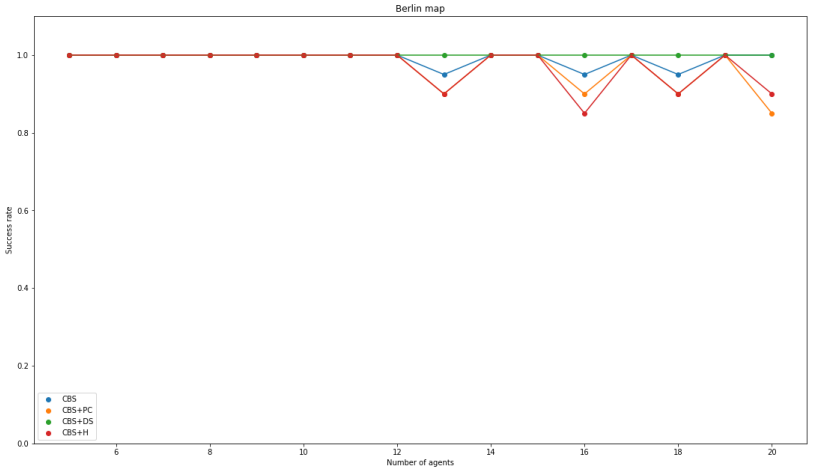
CT nodes created



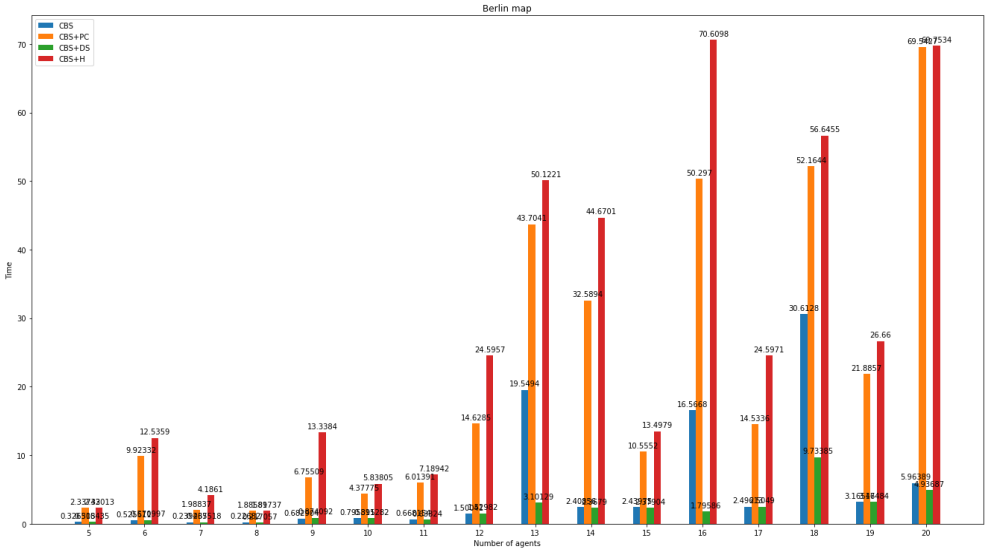
Хочется отметить преимущество CBS+DS и проседание CBS+H.
Не смотря на то, что в некоторых случаях эвристика *h* делает свое дело и выводит CBS+H на первое место по количеству созданных nodes, время, затраченное на нахождение всех кардинальных конфликтов и впоследствии паросочетания, существенно негативно сказывается на производительности CBS+H в целом.

Berlin

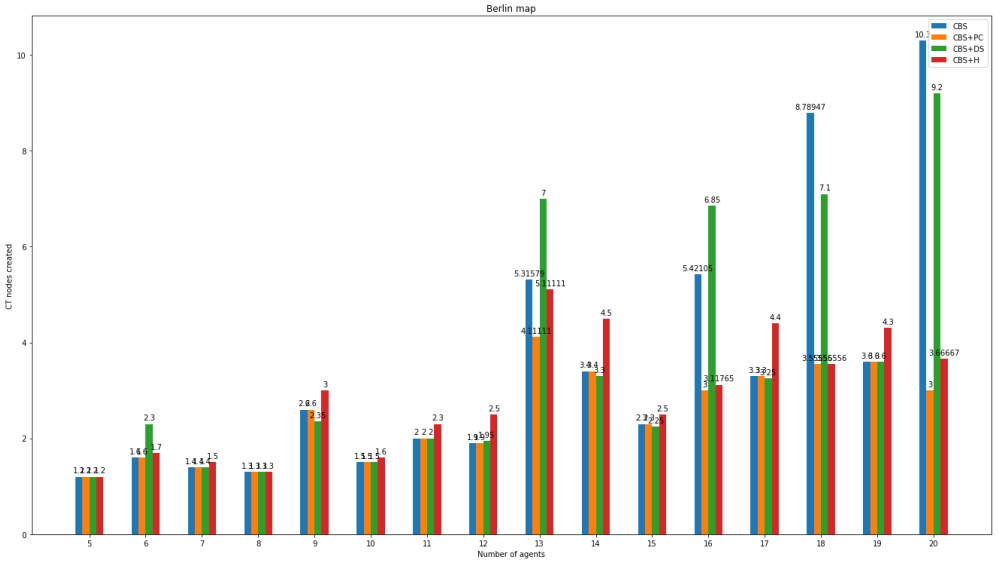
Success rate



Time



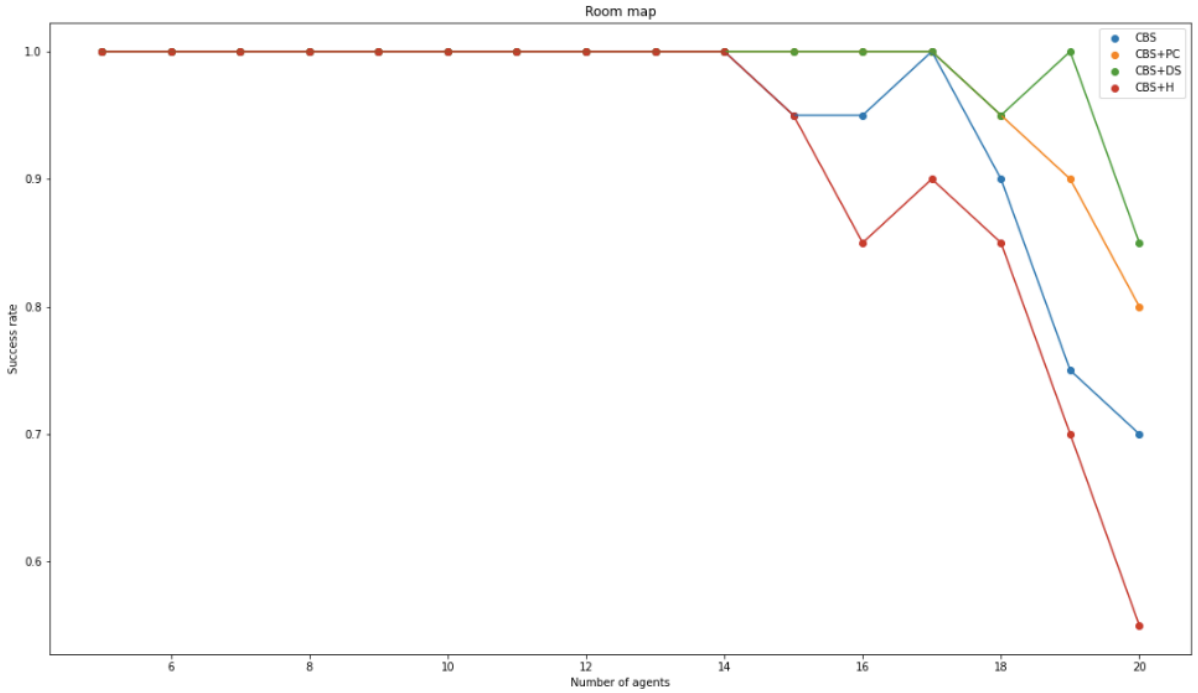
CT nodes created



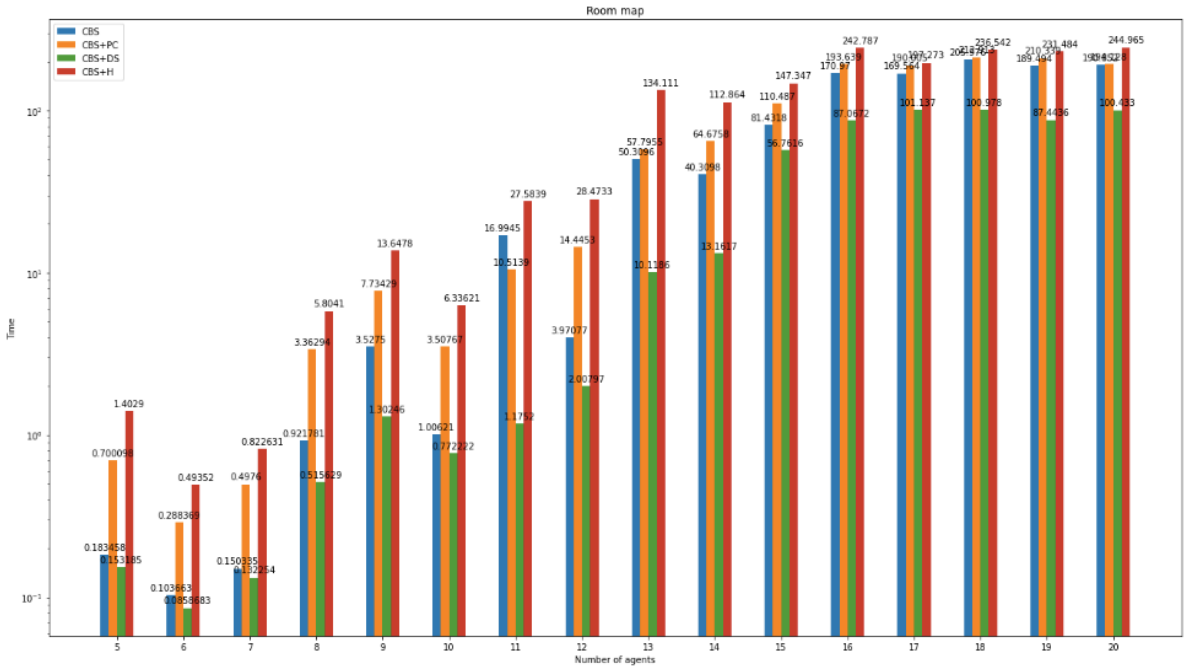
Особенность карты Berlin в том, что она большая, а значит агенты разбросаны далеко друг от друга и редко конфликтуют. Это видно по тому, как мало создается CT nodes. В такой ситуации становится крайне невыгодно применять эвристики, связанные с уменьшением количества CT node, т.к. алгоритм страдает именно на низком уровне. Отсюда и проседание CBS+H и CBS+PC, ведь их низкоуровневые процессы становятся накладными на больших картах, а выигрыш в количестве раскрываемых CT node не актуален.

Room

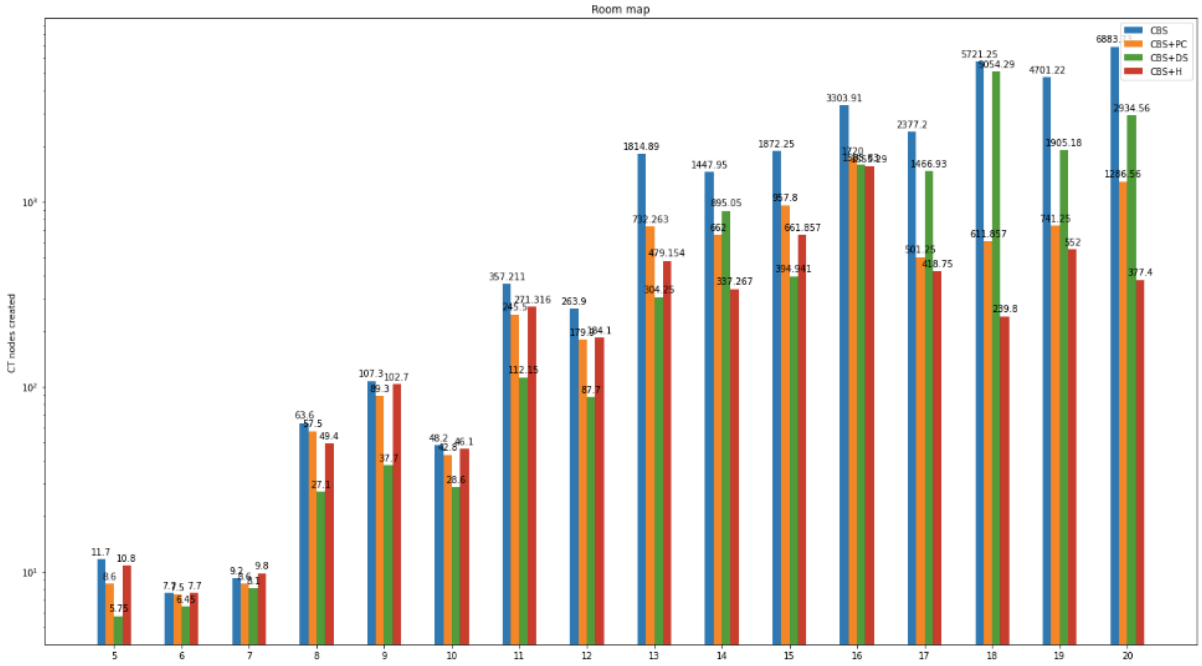
Success rate



Time



CT nodes created



На карте Room много узких мест, поэтому потенциально на ней может быть очень много конфликтов. Из-за этого становится важно, какие конфликты мы выбираем, и графики для CBS+PC это показывают. При этом на большом количестве агентов на алгоритмах (кроме CBS+DS) довольно маленький success rate, это как раз можно обусловить большим количеством конфликтов. При этом CBS+DS все равно достаточно быстрый чтобы это ему не мешало завершаться с успехом
На лицо эффективность стратегии DS. Это можно объяснить тем, что “двери” - популярные вершины для конфликта и часто пересчитываются в случае отсутствия дизъюнктивности.
Не смотря на примерное равенство по раскрытым вершинам, видно, что найти один кардинальный конфликт (PC) быстрее, чем искать все (H).

Общие выводы

- На больших картах “бутылочным горлышком” алгоритма является low-level
- В случае когда конфликтов становится много, время, затраченное на поиск кардинального конфликта оправдывается
- Стратегия Disjoint Splitting “бесплатна” на низком уровне, что компенсирует не оптимальные решения на верхнем
- Время, затраченное на поиск всех кардинальных конфликтов не оправдывается уменьшением узлов CT