УДК 004.8

LPLIAN: АЛГОРИТМ ПЛАНИРОВАНИЯ ТРАЕКТОРИИ С УЧЕТОМ ГЕОМЕТРИЧЕСКИХ ОГРАНИЧЕНИЙ В ДИНАМИЧЕСКОЙ СРЕДЕ

Н.А. Соболева (*nasoboleva@edu.hse.ru*)
Национальный исследовательский университет «Высшая Школа Экономики», Москва

К.С. Яковлев (*yakovlev@isa.ru*)
Федеральный исследовательский центр «Информатика и управление» Российской академии наук, Москва
Национальный исследовательский университет «Высшая Школа Экономики», Москва

Аннотация. Рассматривается задача планирования траектории, удовлетворяющей ограничениям на угол поворота в динамической среде, которая моделируется графом специального вида. Наивным решением этой задачи является повторный запуск алгоритма после каждого возникающего изменения (добавление, удаление препятствий и т.д.). Очевидно, что при незначительных изменениях среды такой подход характеризуется большим объемом повторяющихся вычислений. Для повышения эффективности перепланирования и сокращения числа повторяющихся вычислений предлагается использовать подход Lifelong Planning. В работе предложена модификация алгоритма планирования LIAN, основанная на этом подходе. Приведены результаты модельных экспериментальных исследований.[1]

Ключевые слова: планирование траектории, эвристический поиск, LIAN, Lifelong Planning.

Введение

Традиционно в искусственном интеллекте и робототехнике задача планирования траектории рассматривается как задача поиска пути на графе, вершинам которого соответствуют допустимые положения агента в пространстве, ребрам – элементарные траектории следования из одного положения в другое. При этом весьма распространено использование графа

[1] Работа выполнена при финансовой поддержке РНФ (проект 16-11-00048)

специального вида – графа регулярной декомпозиции (англ. – grid) [Яковлев, 2009][Yap, 2002], [Яковлев и др., 2013], который является простой и вместе с тем информативной графовой моделью окружающей агента среды. Граф регулярной декомпозиции (ГРД) – взвешенный неориентированный граф, каждый элемент (вершина) которого соответствует некоторой области пространства и может быть либо проходим, либо непроходим для агента. Путь на ГРД это последовательность проходимых секций, где секция – отрезок прямой соединяющей две проходимые вершины.

Для поиска пути на ГРД обычно применяются алгоритмы эвристического поиска семейства A* [Hart et al., 1968], такие как Theta* [Daniel et al., 2010], JPS [Harabor, et al., 2011] и другие. При этом подобные алгоритмы обычно минимизируют длину траектории и не накладывают никаких ограничений на ее форму, что может рассматриваться как недостаток при использовании алгоритмов в составе интеллектуальных систем управления мобильными роботами. Так, траектории содержащие резкие смены направления движения (повороты) не всегда являются исполнимыми (т.е. следование по ним не всегда возможно). Одним из способов решения этой проблемы является наложение геометрических ограничений. Методы решения задачи построения траектории с наложенными ограничениями были предложены ранее, в частности алгоритм, учитывающий ограничения на угол поворота, – LIAN [Андрейчук и др., 2017], [Yakovlev et al.,2015], однако эти методы предназначены для планирования в статической среде. Целью данной работы является разработка модификации алгоритма LIAN для функционирования в динамической среде.

1 Постановка задачи

Рассмотрим точечного агента на плоскости, осуществляющего перемещение внутри прямоугольной рабочей области, состоящей из свободного пространства и препятствий: $U = W_{free} \cup W_{obs}$ (здесь $W_{obs} = U \setminus W_{free}$). Дискретная модель этой области представляется в виде ГРД, который в свою очередь может быть представлен в виде матрицы $A_{m \times n}$. (i, j) – элемент матрицы соответствует квадратной области $u_{ij} \in U$, которая центрирована в (i, j), этот элемент помечен как заблокированный, если $u_{ij} \cap W_{obs} \neq \emptyset$ и является проходимым в противном случае. Задана также функция видимости (проходимости), которая определяет, какие перемещения разрешены на ГРД: $los: U \times U \rightarrow \{true, false\}$. Эта функция возвращает true, если отрезок линии, определенный двумя вершинами ГРД a, b – не пересекает ни одну из заблокированных клеток. Путь от начальной

до целевой клетки, от s до g, представляет собой последовательность проходимых сегментов:

$\pi\ (s, g) = \langle (s = a^0, a^1), (a^1, a^2), \dots, (a^k, a^{k+1} = g) \rangle$, где $a^i \in A_{m \times n}$ и $los(a^{i-1}, a^i) = true\ \forall i = 1, \dots, k+1$. Длина пути – сумма длин отрезков пути, $e_i = (a^{i-1}, a^i)$, а длина отрезка равна евклидову расстоянию между его конечными вершинами.

Теперь рассмотрим путь π, составленный из сегментов e_i, и значение $\alpha_m(\pi) = \max(|\alpha(e_1, e_2)|, |\alpha(e_2, e_3)|, \dots, |\alpha(e_{k-1}, e_k)|)$, где $\alpha(e_i, e_{i+1})$ – угол между двумя последовательными отрезками пути. Задача планирования в статической среде может быть сформулирована следующим образом. По данной начальной и целевой клеткам, а также ограничению $\alpha_{MAX} \in [0°, 180°]$ – найти путь π, такой что $\alpha_m(\pi) \leq \alpha_{MAX}$ (см. Рис. 1). Критерием оптимальности будем считать длину построенного пути. Пусть теперь среда изменилась из-за удаления/добавления некоторых препятствий, но расположение начальной и целевой вершин агента осталось неизменным. Задача теперь состоит в поиске пути на модифицированном ГРД.

2 LPLIAN

Наивный подход к поиску пути в условиях динамической среды заключается в повторном вызове алгоритма планирования. В этом случае некоторые выполняемые шаги полностью эквивалентны шагам первого запуска и соответственно некоторые вычисления будут продублированы. Таким образом, целесообразным представляется сохранять промежуточные данные первого запуска и использовать их повторно, с целью экономии вычислительных ресурсов. Lifelong Planning A* (LPA*) – алгоритм, который эффективно использует результаты вычислений, выполненных на первой итерации, для нахождения путей в динамической среде. В этой работе мы применяем подход Lifelong Planning для поиска путей с учетом геометрических ограничений, комбинируя его с алгоритмом LIAN. Результирующий алгоритм назван LPLian. Поскольку LPLian основан на оригинальном алгоритме LIAN и на подходе Lifelong Planning, предоставим информацию об этих двух методах.

2.1 LIAN

LIAN – это эвристический алгоритм поиска, который использует ту же стратегию обхода и обработки состояний, что и классический алгоритм A*. При этом допускаются движения в произвольном направлении (не нарушающие ограничений на угол поворота), также, как и в алгоритме Theta*. Используется идея множественных родителей алгоритма R* [Likhachev and Stentz, 2008].

LIAN, как и любой алгоритм семейства A*, оперирует списками – *OPEN*, в котором располагаются нерассмотренные элементы пространства состояний – кандидаты на дальнейшую обработку; и *CLOSED*, в котором содержатся рассмотренные элементы. В отличие от A*, элемент пространства состояний LIAN идентифицируется не только вершиной графа, то есть ячейкой сетки, но и его родительской вершиной. Это необходимо для учета ограничения на угол поворота. Подобно A*, имеются дополнительные данные, связанные с каждой вершиной (хранятся в памяти во время поиска): g-значение – длина кратчайшего пути к текущей вершине из начальной; *h*-значение – эвристическая оценкой длины кратчайшего пути от текущей вершины к целевой.

На итерации алгоритма происходит извлечение элемента из списка *OPEN*, минимизирующего значение *f*, где $f = g + h$. Этот элемент обрабатывается (раскрывается) следующим образом. Сначала он удаляется из *OPEN* и вставляется в *CLOSED*. Затем создаются элементы-преемники, для этого LIAN использует ячейки, лежащие на заданном входным параметром расстоянии Δ от текущего. Таким образом, LIAN работает только с фрагментами определенной длины, так называемыми Δ-секциями (см. Рис. 1 справа). Вершины-преемники, которые нарушают ограничение достижимости (функция *los*) или ограничение на угол поворота удаляются. Оставшиеся вершины вставляются/обновляются в *OPEN* согласно логике A*. Остановка происходит, если вершина, извлеченная (на очередной итерации) из списка *OPEN*, соответствует целевой клетке графа (в этом случае путь найден) или же если список *OPEN* становится пуст (в этом случае алгоритм возвращает failure, что означает невозможность простроить пути при заданном Δ). В первом случае искомый путь найден и может быть восстановлен с использованием родительских указателей, в противном случае алгоритм сообщает об ошибке, решение не было найдено. На рис.2 слева показан путь, построенный LIAN на пустой сетке со следующими параметрами: $\alpha_{MAX} = 45°, \Delta = 5$.

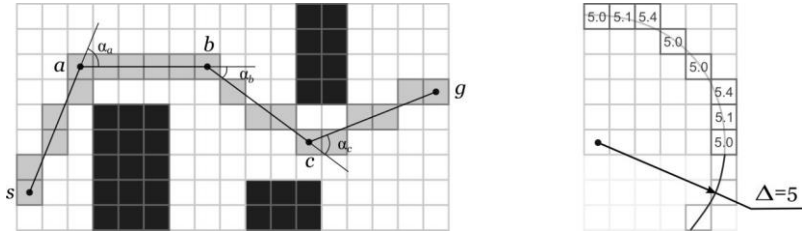

Рис. 1. Граф регулярной декомпозиции и путь на нем (слева), Δ-секция (справа).

2.2 LifeLong Planning

Пусть после того, как путь для данного состояния ГРД был найден, добавляется/удаляется препятствие, при этом начальная и целевая вершины остаются неизменны. Эти изменения влияют на такое свойство вершин как устойчивость (consistency): вершина является устойчивой, если ее g-значение точно равно длине кратчайшего пути в нее из начальной вершины. Если клетка ГРД в процессе появления препятствий становится заблокированной и путь к соответствующей вершине перестает существовать (формально g-значение такой вершины становится $+\infty$), устойчивость пропадает. То же самое верно для клеток, которые стали проходимыми в результате изменений. Более того, клетки, которые находятся рядом с ними могут также утратить свойство устойчивости. В то же время, если удается восстановить устойчивость всех вершин-состояний, второй поиск не потребуется. Это основная идея метода LifeLong Planning [Likhachev et. al., 2001] – восстанавливать устойчивость вершин вместо генерации новых с нуля.

Напрямую применение метода LifeLong Planning применяемого для алгоритма A* к рассматриваемой проблеме планирования траектории с ограничением на угол поворота невозможно, так как пространство поиска в алгоритме LIAN отличается от A*. Описание модификации (метода LifeLong Planning для алгоритма LIAN) представлено далее.

2.3 LPLIAN

Первый запуск LPLian на исходном ГРД ничем не отличается от прохода алгоритма LIAN. Результатом является путь, а также списки *OPEN* и *CLOSED*, заполненные элементами пространства поиска, которые были сгенерированы для построения пути (см. Рис. 2 слева). При обнаружении изменений в графе алгоритм выявляет вершины, потерявшие свою устойчивость, а также генерирует некоторые новые вершины. То есть вместо генерации всех вершин из обоих списков с нуля, обрабатываются только вершины, которые потеряли устойчивость или еще не рассматривались на предыдущем запуске алгоритма. После такой обработки все неустойчивые состояния идентифицированы и помещены в *OPEN* (см. Рис. 2 по центру). Наконец, стандартный цикл алгоритма LIAN возобновляется, то есть вершины извлекаются из списка OPEN и происходит их раскрытие (*expand*) (см. Рис. 2 справа).

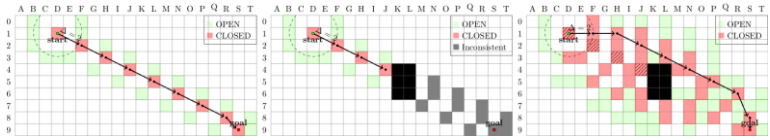

Рис 2. Этапы работы алгоритма LPLIAN (слева-направо): 1) построение пути на первоначальном ГРД, 2) нахождение неустойчивых (отмечены серым) вершин при возникновении препятствия, 3) поиск нового пути с учетом известных списков OPEN и CLOSED.

Опишем этапы работы алгоритма более формально. LPLian использует следующие характеристики вершин: метка вершины g – длина кратчайшего найденного пути в эту вершину, направление движения – определяется вершиной-родителем, метка устойчивости которая задается следующим образом:

$$rhs(a) = \begin{cases} 0, & \text{если} \quad a = start, \\ \min\limits_{a' \in pred(a)} g(a') + cost(a', a), & \text{иначе.} \end{cases}$$

Здесь *cost*(a, b) – вес перехода из a в b, а *pred*(a) – множество вершин, переход из которых в вершину a является допустимым, будем называть их предшественниками вершины a. Вершина a является устойчивой, если выполняется равенство $rhs(a) = g(a)$. Порядок рассмотрения вершин определяется по соответствующему ключу:

$key(a) = [k_1 = \min(g(a), rhs(a) + h(a)), k_2 = \min(g(a), rhs(a))]$.

Вершины добавляются в очередь и извлекаются из нее в порядке минимальности ключа, то есть вершина a должна быть рассмотрена раньше вершины a', если $k_1(a) < k_1(a')$ или $k_1(a) = k_1(a')$ и $k_2(a) < k_2(a')$. Ключ используется такой же, как и в алгоритме LPA*, где его выбор обусловлен следующим фактором: при введении такого ключа алгоритм не требует ручного переключения между состоянием поиска неустойчивых вершин и состоянием стандартной обработки новых вершин. То есть появившаяся неустойчивая вершина будет иметь более высокую приоритетность, чем вершины, для которых она может являться предшественником. Следовательно, если среди ее последователей есть неустойчивые, то они будут найдены на следующем шаге.

Ниже представлен псевдокод алгоритма LPLian, который во многом повторяет этапы алгоритма LPA*. Новизна заключается в добавлении функции *ResetParent*, которая для неустойчивой вершины a находит всех потенциальных предшественников и выбирает из них наилучшего. Особенность заключается в том, что функция ищет вершины, лежащие на расстоянии Δ от текущей вершины и при этом не нарушающие дальнейший угол поворота пути, то есть угол поворота между a и его последователем a'.

Algorithm 1 LPLIAN-Main

```
1: function UpdateVertex(v)
2:     if v ∈ OPEN then
3:         OPEN.remove(v)
4:     end if
5:     if g(v) ≠ rhs(v) then
6:         OPEN.insert(v, key(v))
7:     end if
8: end function
9:
10: function ComputeShortestPath( )
11:     while OPEN.top.key() < key(goal) or rhs(goal) ≠ g(goal) do
12:         current = OPEN.top()
13:         if g(current) > rhs(current) then
14:             g(current) = rhs(current)
15:             expand(current)
16:         else
17:             g(current) = ∞
18:             for v in successors(current) and current do
19:                 ResetParent(v, current)
20:                 UpdateVertex(v)
21:             end for
22:         end if
23:     end while
24:     if g(goal) < ∞ then
25:         return GetPath(goal)
26:     else
27:         return path-not-found
28:     end if
29: end function
30:
31: function FindPath( )
32:     Set all g rhs-values to ∞
33:     rhs(start) = 0
34:     OPEN.insert(start)
35:     while true do
36:         ComputeShortestPath()
37:         wait for the changes
38:         for v: ⟨parent.parent(v), parent(v)⟩ is affected by changes do
39:             ResetParent(v, parent(v))
40:             if parent(v) ≠ NULL then
41:                 UpdateVertex(v)
42:             end if
43:         end for
44:     end while
45: end function
```

Листинг 1. Основной цикл алгоритма LPLIAN.

Более формально функция описывается следующим образом. Изначально, идентифицируется множество потенциальных вершин-кандидатов (множество *U*), которые лежат на Δ-расстоянии от родителя текущей вершины. Затем выбирается узел, удовлетворяющий следующим двум критериям: 1) угол между ячейками ГРД *parent*(*u*), *u*, *parent*, а также угол между *u*, *parent*, *current* удовлетворяют наложенному ограничению; 2) использование сегмента (*u*, *parent*) способствует минимизации длины пути.

Algorithm 2 LPLIAN-ResetParent

```
1: function ResetParent(current, parent)
2:     U - set of nodes at the Δ-distance from parent
3:     for u in U do
4:         node = (parent, u)
5:         if ∠(u, node) < α_max and ∠(node, current) < α_max then
6:             if rhs(parent) > rhs(u) + cost(u, parent) then
7:                 parent(parent) = u
8:                 rhs(parent) = rhs(u) + cost(u, parent)
9:             end if
10:        end if
11:    end for
12:    rhs(current) = rhs(parent) + cost(parent, current)
13: end function
```

Листинг 2. Функция ResetParent

3 Экспериментальные исследования

Для проведения экспериментальных исследований в качестве входных данных использовались фрагменты карт реальной городской местности, полученных из коллекции Openstreetmaps. Всего было использовано 100 фрагментов размером ~ 1,2 на 1,2 км. Указанные фрагменты были преобразованы в ГРД размера 501x501 вершин. Для каждого ГРД было сгенерировано 200 заданий таким образом, что расстояние между начальным и целевым положениями превышало 400 клеток (960 метров).

Одиночный эксперимент состоял в сравнении работы динамической модификации LIAN и нескольких запусков статического LIAN на карте до изменений и после. Тестирование происходило следующим образом, запускалась первая итерация динамического алгоритма, далее карта менялась – на уже построенном пути помещалось препятствие размером 20x10. После изменений, динамический алгоритм LPLian достраивал путь. Для сравнения, на новой карте с препятствием так же запускался статический алгоритм LIAN и подсчитывалось суммарное время работы алгоритмов на обеих картах – 2xLIAN. Так как сложность восстановления пути зависит от того, в каком месте карты произошли изменения, было исследованы различные значения параметра obstacle position, отвечающего за то, на каком расстоянии от клетки старта/финиша было размещено препятствие. Значение параметра obstacle position находится в диапазоне от 0 до 1, и порядковый номер вершины построенного пути, вокруг которой строится препятствие, вычисляется как $i = round(obstacle\_position \times length(path))$. То есть при obstacle position = 0.2 препятствие находится ближе к стартовой вершине, при 0.8 к целевой. Препятствие представляет из себя прямоугольник размера 20x10.

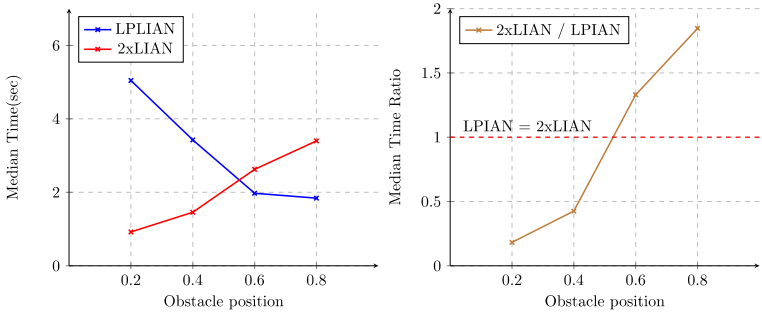

Рис. 5. Графики сравнения работы алгоритмов LPLIAN и двух проходов алгоритма LIAN.

В процессе проведения эксперимента был установлен верхний порог на время выполнения 200 сек, все задания, которые были не решены тем или иным алгоритмом в пределах отведенного времени (470 из 4000) не участвовали в общем усреднении. Сравнивались скорости решения задачи после добавления нового препятствия, то есть время работы стандартного алгоритма LIAN на новой карте с препятствием и время, затраченное на восстановление пути после добавления препятствия для LPLian. Для уменьшения разброса значений, полученных в результате экспериментов, было выбрано медианное значение вместо усредненного. На рисунке 5 (слева) показано взаимное расположение кривых изменения медианы времени работы алгоритмов в зависимости от расположения препятствия и (справа) – изменение отношение времени работы 2xLIAN к LPLIAN. Из графиков видно, что при достаточно близком (0.2. 0.4) расположении препятствия к начальной вершине, алгоритму LPLian требуется больше времени, чтобы перестроить путь, чем построить его просто заново (как это делает 2xLIAN). Это связано с тем, что LPLian совершает два прохода по оставшейся части пути (вершинам, которые потенциально могли потерять устойчивость) – первый проход детектирует неустойчивые вершины, и второй – повторное раскрытие. При перемещении препятствия ближе к целевой вершине время работы LPLian значительно сокращается. А время построения нового пути алгоритмом LIAN становится почти в два раза больше времени, которое тратит LPLian на то, чтобы восстановить путь.

В таблице 1 приведены сравнения медианных значений рассматриваемых выходных параметров обоих алгоритмов. В случае, когда obstacle position = 0.8, получаем выигрыш по времени равный 84.6 %.

Таблица 1. Результаты алгоритмов LPLIAN и двух алгоритмов LIAN.

obstacle position	0.2	0.4	0.6	0.8
LPLIAN	5.045	3.425	1.974	1.842
2xLIAN	0.921 (- 81.7 %)	1.457 (- 57.5 %)	2.626 (+ 33 %)	3.4 (+ 84.6 %)

3 Заключение

Использование предложенного алгоритма оправдано при навигации в динамической среде, если изменения происходят ближе к целевой вершине. Это особенно важно, т.к. при планировании траектории в ограниченно-наблюдаемой среде обычно старт и цель меняют местами исходя из алгоритмических соображений и почти все изменения происходят рядом с агентов (т.е. у цели).

Список литературы

[Андрейчук и др., 2017] Андрейчук А.А, Яковлев К.С. Методы планирования траектории на плоскости с учетом геометрических ограничений // Известия РАН. Теория и системы управления, 2017. № 6, с. 125–156.

[Яковлев, 2009] Яковлев К.С. Графы специальной структуры в задачах планирования траектории. Труды III международной конференции «Системный анализ и информационные технологии САИТ-2009». М: ИСА РАН, 2009.

[Яковлев и др., 2013] Яковлев К.С., Баскин Е.С. Графовые модели в задаче планирования траектории на плоскости // Искусственный интеллект и принятие решений. 2013. №1. С. 5-12.

[Daniel et al., 2010] Daniel K., Nash A., Koenig S., Felner A. Theta*: Any-angle Path Planning on Grids // J. Artificial Intelligence Research. 2010. V. 39. P. 533–579.

[Harabor and Grastien, 2011] Harabor, D.D., Grastien, A.: Online graph pruning for pathfinding on grid maps. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011). pp. 1114–1119 (2011)

[Hart et. al., 1968] Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), 100–107 (1968)

[Likhachev et. al., 2001] Likhachev, M., Koenig, S.: Lifelong planning A* and dynamic A*lite: The proofs. In: Technical report, College of Computing, Georgia Institute of Technology, Atlanta (Georgia) (2001)

[Likhachev and Stentz, 2008] Likhachev, M., Stentz, A.: R* search. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence, AAAI-2008. pp. 344–350 (2008)

[Yap, 2002] Yap, P.: Grid-based path-finding. In: Proceedings of 15th Conference of the Canadian Society for Computational Studies of Intelligence. pp. 44–55. Springer (2002)