УДК 004.8

МЕТОД РАЗРЕШЕНИЯ КОНФЛИКТОВ ПРИ ПЛАНИРОВАНИИ ПРОСТРАНСТВЕННЫХ ТРАЕКТОРИЙ ДЛЯ ГРУППЫ БЕСПИЛОТНЫХ ЛЕТАТЕЛЬНЫХ АППАРАТОВ

А.А. Андрейчук (*andreychuk@mail.com*)
РУДН, Москва

К.С. Яковлев (*yakovlev@isa.ru*)
ФИЦ ИУ РАН, Москва

Аннотация. В работе рассматривается задача планирования совокупности траекторий для группы интеллектуальных агентов (беспилотных летательных аппаратов) в двухмерном случае. Исследуется децентрализованный подход к ее решению, когда процесс построения траекторий осуществляется независимо, а согласование и устранение конфликтов – централизовано. Предлагается новый метод разрешения конфликтов, использующий как механизмы задержки агентов, так и оригинальную процедуру локального перепланирования траектории.[1]

Ключевые слова: планирование траектории, мультиагентное планирование, разрешение конфликтов, беспилотный летательный аппарат, мультиагентные системы.

Введение

Задача планирования траектории в искусственном интеллекте и робототехнике обычно рассматривается как задача поиска пути на графе, вершинам которого соответствуют положения агента в пространстве, а ребрам – возможные переходы между ними (элементарные траектории). При этом задача планирования для коалиции агентов является NP-сложной [Yu, 2013]. Получение оптимальных решений этой задачи при достаточно большом числе агентов и размере графа весьма затруднительно, однако именно задачи с такими характеристиками обычно встречаются на практике, например, в области автоматизации управления коалициями беспилотных транспортных средств. Поэтому весьма распространен децентрализованный подход к решению задачи

[1] Работа выполнена при финансовой поддержке РФФИ (проект № 15-37-20893).

планирования, когда каждый агент строит свою траекторию независимо, а затем эти траектории централизованно согласуются между собой. Алгоритмы, реализующие этот подход, такие как, например, MAPP [Wang, 2011] или WHCA* [Silver, 2005], характеризуются высокой скоростью работы и активно применяются на практике. При этом эти алгоритмы способны функционировать лишь в условиях, когда конфликт между индивидуальными решениями (траекториями) возникает в определенной вершине графа. Однако известно, что в области планирования траектории для беспилотных транспортных средств, предпочтительнее использовать методы планирования, которые позволяют строить траектории, лишь частично привязанные к топологии графа. А именно – вершины графа используются в качестве опорных точек траектории, но сегменты траектории (ребра графа) генерируются на лету. К таким алгоритмам относятся, например, Theta* [Nash, 2007], Anya [Harabor, 2013], LIAN [Yakovlev, 2015]. Известные алгоритмы разрешения конфликтов не подходят для обработки траекторий, построенных этими алгоритмами, т.к. конфликты могут возникать в областях пространства, не соответствующих вершинам графовой модели окружающей среды. Таким образом, актуальной является задача разработки новых методов и алгоритмов планирования траектории для совокупности беспилотных транспортных средств, лишенных указанных недостатков. Именно эта задача решается в данной работе.

1 Модельная задача

Рассматривается задача автоматизации маловысотного полета группы малых беспилотных летательных аппаратов (в терминологии искусственного интеллекта – агентов) в городских условиях. Исследуется следующий сценарий: n идентичных по своим характеристикам БЛА находятся на земле, затем набирают определенную высоту и осуществляют полет в горизонтальной плоскости (облетая препятствия) до пункта назначения, в котором совершают посадку. Скорость перемещения всех БЛА полагается одинаковой. Известны траектории отдельных БЛА, которые могут быть конфликтными в совокупности, т.е. два (или более БЛА) могут оказаться в одной точке пространства одновременно. Требуется разрешить все конфликты и получить такие траектории, следование по которым позволит избежать столкновений, как с препятствиями, так и друг с другом.

1 Формальная постановка задачи

Дан граф специального вида, МТ-Граф [Яковлев, 2013], который представлен в виде матрицы $A_{H \times W}=\{a_{ij}\}$, где i, j – индексы вершин (клеток)

графа, а H, W – его размеры. Даны n путей на этом графе: $\pi^{(1)}$, … $\pi^{(n)}$. Путь π – это последовательность секций $\pi=\{e_1, \ldots, e_v\}$, а секция $e=\langle a_{ij}, a_{kl}\rangle=\langle sp(e), ep(e)\rangle$, это проходимая прямая линия соединяющая центры соответствующих клеток – см. рис. 1.

Длиной секции, $len(e)$, будем называть расстояние между её концами. Секцию, длина которой после округления равна некоторому целому Δ, будем называть Δ-секцией. Длиной пути π является сумма длин всех секций, составляющих этот путь: $len(\pi)=len(e_1) + \ldots + len(e_v)$.

Здесь и далее будем рассматривать только пути, состоящие из Δ-секций. Построение таких путей может быть осуществлено, например, алгоритмом LIAN [Yakovlev et al., 2015] (который помимо прочего учитывает ограничение на угол отклонения между секциями) или незначительно модифицированным алгоритмом Theta* [Nash et al., 2007].

Потенциальным решением назовем набор частичных потенциальных решений (partial potential solutions) $PS=\{PPS^{(1)}, \ldots, PPS^{(n)}\}$, где $PPS^{(i)}$ это пара $\langle\pi^{(i)}, t^{(i)}\rangle$, $\pi^{(i)}$ –i-й путь, $t^{(i)}$ – i-я задержка.

Будем считать, что за 1 единицу времени агент преодолевает расстояние равное расстоянию между центрами двух горизонтально смежных вершин. В этих же единицах будем измерять величину $t^{(i)}$. Таким образом общая временная стоимость решения равна сумме стоимостей каждого частичного решения: $cost(PS) = cost(PPS^{(1)}) + \ldots + cost(PPS^{(n)})$, $cost(PPS^{(i)})=t^{(i)}+len(\pi^{(i)})$.

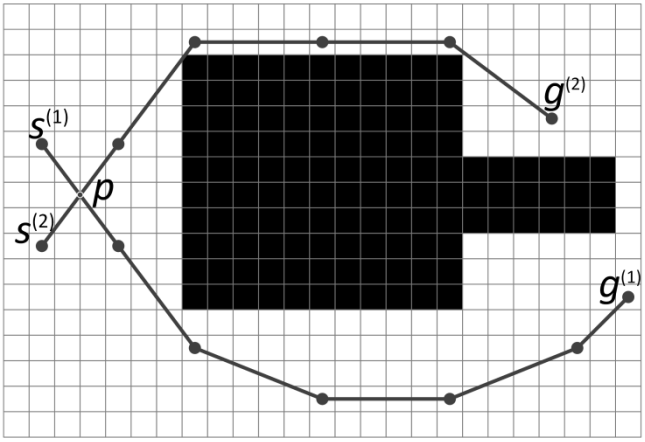

Рис.1 Примеры путей для двух агентов на MT-графе.

Пусть дано частичное решение $PPS^{(i)}=\langle\pi^{(i)}, t^{(i)}\rangle$ и секция $e_j^{(i)}\in\pi^{(i)}$. Далее в целях упрощения верхние индексы не используются. g-значением

секции e_j является сумма длин всех секций, составляющих частичный путь от вершины s до вершины $sp(e_j)$, а также задержка t: $g(e_j, PPS) = t + len(e_1) + len(e_2) + \ldots + len(e_{j-1})$. Пусть дана некоторая точка p, принадлежащая секции e_j. g-значением этой точки будем называть величину $g(p, e_j, PPS)=g(e_j, PPS)+dist(sp(e_j), p)$.

Две секции, принадлежащие различные частичным решениям являются потенциально конфликтными, если их пересечение (как отрезков) не пусто, то они. Секции являются конфликтными, если они являются потенциально конфликтными и существует точка, принадлежащая обеим секциям, такая что g-значения для обоих путей до этой точки равны. Два частичных решения являются конфликтными, если они содержат хотя бы одну пару конфликтных секций. Потенциальное решение является неконфликтным решением, если все его частичные решения являются неконфликтными

Задача согласования конфликтных состоит в преобразовании множества частичных решений, содержащих конфликты, в неконфликтное решение.

2 Типизация конфликтов и процедура их выявления

Конфликты могут быть разделены на три типа: пересечение, лобовое столкновение, преследование (см. Рис.2). Конфликты типа «пересечение» возникают между неколлинеарными секциями, которые пересекаются в одной точке. Если g-значения в этой точке для обоих путей равны – то эти секции конфликтны. Конфликты второго типа возникают, когда две секции коллинеарны и разнонаправленны. Конфликты третьего типа возникают между секциями, которые являются коллинеарными и сонаправленными.

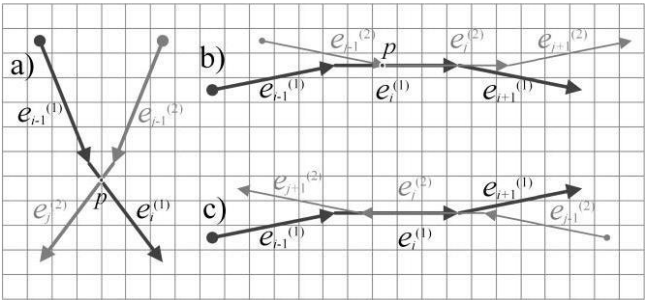

Рис.2 Типы конфликтов a) пересечение, b) преследование c) лобовое столкновение.

Пусть даны два частичных решения $PPS^{(1)}=\langle\pi^{(1)}, t^{(1)}\rangle$, $PPS^{(2)}=\langle\pi^{(2)}, t^{(2)}\rangle$, и секции $e_i^{(1)}\in\pi^{(1)}$, $e_j^{(2)}\in\pi^{(2)}$, которые необходимо проверить на конфликтность. Заметим, что секции не могут иметь конфликт, если

расстояние между концами секций больше чем сумма их длин. Если же это не так, то предлагается следующая процедура проверки на наличие конфликта.

Сначала необходимо проверить секции на коллинеарность. Если они не коллинарны, то возможен конфликт только первого типа. Используя геометрические формулы, можно найти точку пересечения, если она существует и посчитать g-значения для обоих путей. Строго говоря, чтобы считать две секции конфликтными, g-значения должны совпадать. Однако с практической точки зрения целесообразно использовать следующее выражение: $|g(p, e_i, PPS^{(1)}) - g(p, e_j, PPS^{(2)})| < r$, где r – минимальный безопасный радиус, который равен, к примеру, расстоянию между центрами двух горизонтально смежных вершин.

Если же две секции коллинеарны и находятся на одной прямой, необходимо проверить их на сонаправленность. Если секции разнонаправленны, между ними возможен конфликт второго типа. Необходимо проверить следующие соотношения: (1) $g(sp(e_i^{(1)}), PPS^{(1)}) + \text{dist}(sp(e_i^{(1)}), ep(e_j^{(2)})) \le g(ep(e_j^{(2)}), PPS^{(2)})$; (2) $g(sp(e_j^{(2)}), PPS^{(2)}) + \text{dist}(sp(e_j^{(2)}), ep(e_i^{(1)})) \le g(ep(e_i^{(1)}, PPS^{(1)})$. Если они выполняются, значит существует точка, принадлежащая обеим секциям, *g*-значение которой равно для обоих путей. В случае, когда секции сонаправленны, возможен конфликт третьего рода. Чтобы его проверить необходимо взять стартовую вершину секции, которая принадлежит общей части обеих секций, и посчитать её g-значения для обоих путей. Если они равны, значит эти две секции имеют конфликт.

Все вышеописанные проверки могут быть скомбинированы в одну функцию *FindFirstConflict*, которая станет одной из главных частей алгоритма разрешения конфликтов, который будет описан далее.

3 Локальное перепланирование

Пусть даны два частичных решения $PPS^{(1)}=\langle\pi^{(1)}, t^{(1)}\rangle$, $PPS^{(2)}=\langle\pi^{(2)}, t^{(2)}\rangle$, и секции $e_i^{(1)}\in\pi^{(1)}$, $e_j^{(2)}\in\pi^{(2)}$, которые являются конфликтными. Идея локального перепланирования заключается в том, чтобы построить незначительно отличающийся путь, изменив секции $e_j^{(2)}$ и $e_{j+1}^{(2)}$. Так как все пути являются Δ-путями, то можно предложить следующий подход к локальному перепланированию.

Сначала необходимо определить какие вершины находятся на расстоянии Δ от вершины e_j (для этого можно воспользоваться алгоритмом [Pitteway, 1985]). Обозначим такие вершины как *CIRCLE*. Вершины $a_{kl}\in CIRCLE$, которые не удовлетворяют заранее заданному ограничению на максимальный угол отклонения $\boldsymbol{\alpha_m}$, удаляются из *CIRCLE*. Удаляя такие вершины, мы отсекаем резкие смены направления. После того, как набор

вершин-кандидатов найден, из него выбирается вершина a_{kl} , такая что секции $e_{new1}=\langle ep(e_{j-1}), a_{kl}\rangle$ и $e_{new2}=\langle a_{kl}, ep(e_{j+1})\rangle$ проходимы. Это и есть секции обхода (см. Рис.3).

Обозначим данную процедуру как *ComputeLocalDetour*. Она будет использоваться в дальнейшем при построении алгоритма разрешения множества конфликтов.

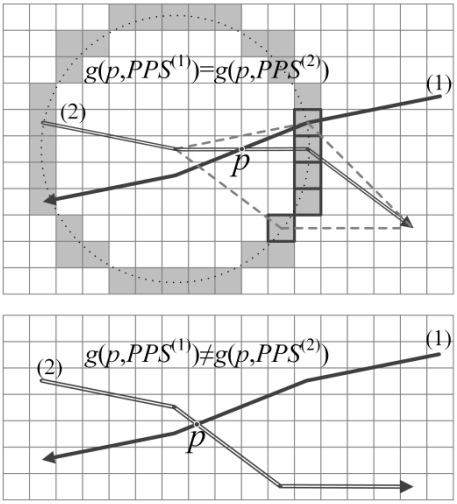

Рис.3 Процедура локального перепланирования. a) Вершины, формирующие Δ-окружность отмечены серым. Жирным отмечены вершины, удовлетворяющие ограничениям на максимальный угол отклонения. b) Результат процедуры перепланирования.

4 Алгоритм разрешения конфликтов

На вход алгоритму подается четверка <***PS***, **Δ**, $\boldsymbol{\alpha_m}$, ***wait***>, где ***PS*** – это начальное потенциальное решение (набор путей с нулевыми задержками), Δ – размер секции, $\boldsymbol{\alpha_m}$ **–** максимальный угол отклонения для функции *ComputeLocalDetour* и параметр ***wait*** **–** значение задержки (применяется в том случае, когда локальное перепланирование не смогло решить конфликт).

Псевдокод алгоритма представлен на Рис.4.

Algorithm 1 Conflicts Resolution

Input: ***PS***, Δ, **α**, ***wait***; **Output**: ***PS***$' \in$ NoCON

1: $\{HEAD, TAIL\} \leftarrow FormHeadAndTail(\boldsymbol{PS})$
2: **while** $TAIL \neq \varnothing$
3: $PPS^{(cur)}$=argmin$_{PPS \in TAIL}$ $NumberOfConficts(PPS, TAIL \cup HEAD)$
4: $TAIL$.remove($PPS^{(cur)}$)
5: **while** ($\{e_v^{(cur)}, e_w^{(k)}\} \leftarrow FindFirstConflict(PPS^{(cur)}, HEAD)) \neq \varnothing$
6: $\pi_{new} \leftarrow ComputeLocalDetour(PPS^{(cur)}, e_v^{(cur)}, PPS^{(k)}, \Delta, \boldsymbol{\alpha})$
7: **if** $\pi_{new} = \pi^{(cur)}$
8: $t^{(cur)}$+=***wait***
9: **else**
10: $\{e_t^{(new)}, e_s^{(m)}\} \leftarrow FindFirstConflict(PPS^{(new)}, HEAD)$
11: **if** $\{e_t^{(new)}, e_s^{(m)}\} = \varnothing$ or $t>v$
12: $\pi^{(cur)} = \pi_{new}$
13: **else**
14: $t^{(cur)}$+=***wait***
15: $HEAD$.add($PPS^{(cur)}$)
16: **return** $HEAD$

Рис.4 Псевдокод алгоритма разрешения конфликтов.

Основная идея предлагаемого алгоритма заключается в разделении начального решения на два подмножества– *HEAD* и *TAIL*. В *HEAD* содержатся частичные решения, которые не конфликтуют между собой, а в *TAIL* – все остальные частичные решения, которые имеют, по крайней мере один конфликт с элементами из множества *HEAD*. На каждом шаге алгоритм выбирает частичное решения из *TAIL*, устраняет все конфликты, которые это решение имеет с решениями из *HEAD*, удаляет его из *TAIL* и добавляет в *HEAD*. Таким образом, на каждой итерации количество частичных решений в множестве *HEAD* увеличивается и в конечном итоге оно будет содержать все частичные решения. Т.к. все частичные решения множества *HEAD* не конфликтуют между собой, будет найдено искомое неконфликтное решение.

Первый шаг алгоритма заключается в формировании множеств *HEAD* и *TAIL* по исходным данным. После того, как эти множества сформированы, начинается основной цикл работы алгоритма (3-14). На каждой итерации выбирается потенциальное частичное решение с самым высоким приоритетом (приоритет определяется по количеству конфликтов, которое имеет это решение с множеством *HEAD*) из *TAIL* - $PPS^{(cur)}$ и удаляется из *TAIL*. В процессе работы цикла (6-13) устраняются все конфликты, и решение добавляется в *HEAD*.

Для того чтобы понять, что $PPS^{(cur)}$ не имеет конфликтов с *HEAD*

используется функция *FindFirstConflict*, которая возвращает пустое множество в случае, когда конфликтов между $PPS^{(cur)}$ и *HEAD* нет. Если же конфликты существуют, то функция вернет первый найденный конфликт.

После того как конфликт найден, алгоритм пытается его решить с помощью функции *ComputeLocalDetour*. Если изменить путь не удалось, задержка $t^{(cur)}$ увеличивается на значение ***wait***. Если функции *ComputeLocalDetour* удалось изменить путь, то необходимо проверить, не появился ли новый конфликт в текущей или более ранней секции. Если нет, считаем, что конфликт устранен и сохраняем новый путь. Если же возник конфликт в более ранней секции, то такое решение не используется по следующий причине. Допустим, был найден конфликт $con_a=\{e_v^{(cur)}\in\pi^{(cur)}, e_w^{(k)}\in\pi^{(k)}\}$. После того, как путь был локально перепланирован, был найден новый конфликт $con_b=\{e_t^{(new)}\in\pi^{(new)}, e_s^{(m)}\in\pi^{(m)}\}$ и при этом $t\leq v$. После этого, алгоритм пытается решить конфликт con_b и снова создает конфликт con_a. В результате возникает цикл, который не улучшает решение. Поэтому, в случае, когда после локального перепланирования был найден конфликт в той же самой или более ранней секции, алгоритм отменяет изменения, сделанные функцией *ComputeLocalDetour*, и добавляет задержку. В отличие от локального перепланирования, задержка не приводит к возникновению циклов, так как задержка работает только в одну сторону, то есть, задерживая одного агента несколько раз подряд невозможно создать конфликты, которые были у этого же агента с меньшим числом задержек.

5 Экспериментальные исследования

Тестирование проводилось на персональном компьютере Intel Q8300 2.5GHz, 2Gb RAM. В качестве заданий использовались фрагменты карт городской местности (Москвы), полученные из открытой базы данных OpenStreetMaps (www.openstreetmaps.org). Было использовано 100 фрагментов размером 1347x1347 м, каждый из которых был преобразован в MT-граф 501x501 вершин (непроходимыми считаются вершины, соответствующие зданиям).

Для каждого фрагмента было сгенерировано 4 задания 2-х разных типов. В случае заданий первого типа, стартовые вершины расположены случайном образом на краях карты, а целевые вершины – на противоположных. Задания второго типа характеризуются тем, что все стартовые вершины расположены в области 50x50 на одном из краев карты, а все целевые вершины – в области 50x50 на противоположной стороне.

При тестировании использовались следующие значения параметров $\Delta=5$, $\boldsymbol{wait}=5$, $\boldsymbol{\alpha_m}=25$. В качестве алгоритмов планирования траектории

были использованы Theta* и LIAN.

В таблице 1 представлена подробная статистика, касающаяся процесса разрешения конфликтов.

Таблица 1. Статистика разрешенных конфликтов.

	Theta*		LIAN	
	Тип-1	Тип-2	Тип-1	Тип-2
Агенты				
Задержаны	12.07	64.685	15.3	72.335
Перепланированы	14.73	58.565	7.77	64.09
Не изменены	79.54	34.33	80.56	26.44
Количество разрешенных конфликтов				
Задержкой	21.025	746.085	26.575	1176.19
Перепланированием	30.935	1442.185	12.7	638.3

В случае заданий первого типа в среднем 80% агентов вообще не подвергаются никаким изменениям, в то время как для разрешения всех конфликтов в заданиях второго типа требуется остановить и перепланировать 60-70% всех агентов. Так же видно, что количество конфликтов по секциям в несколько десятков раз выше у второго типа заданий.

В таблице 2 представлены результаты эксперимента, касающиеся времени работы и стоимости решения после обоих этапов планирования.

Таблица 2. Время работы алгоритма и стоимость найденного решения.

		Theta*		LIAN	
		Тип-1	Тип-2	Тип-1	Тип-2
Этап-1	Время(с)	7.3783	6.9426	9.5827	5.387
	Стоимость	46926	46722	49636	49651
Этап-2	Время(с)	0.1466	0.804	0.1441	0.5926
	Стоимость	47034 (+0.23%)	50477 (+8.04%)	49772 (+0.27%)	55566 (+11.91%)

Полученные результаты свидетельствуют о том, что для заданий первого типа ухудшение стоимости решения составляет менее 1%, а для заданий второго типа – 10-15%. Такая большая разница связана с тем, что в заданиях второго типа возникает значительно большее число конфликтов, так как все агенты фактически движутся вместе из одной области в другую, в результате чего возникают конфликты на протяжении всего пути.

Заключение

В работе рассмотрена задача согласования множества конфликтных траекторий возникающих при планировании перемещений группы интеллектуальных агентов на примере беспилотных летательных аппаратов, совершающих маловысотный полет в городских условиях. Предложены процедуры идентификации конфликтов и их разрешения путем локального перепланирования. На основе этих процедур разработан новый алгоритм, который позволяет разрешать конфликты и строить совокупность неконфликтных траекторий. Экспериментальные исследования показали его применимость к задаче, описанной в работе. В дальнейшем планируется теоретическое исследование разработанного алгоритма.

Список литературы

[Яковлев, 2013] Яковлев К.С., Баскин Е.С. Графовые модели в задаче планирования траектории на плоскости // Искусственный интеллект и принятие решений, 1, 2013. С.5-12.

[Harabor, 2013] Daniel Harabor and Alban Grastien. An Optimal Any-Angle Pathfinding Algorithm. In ICAPS, 2013.

[Nash, 2007] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta*: Any-Angle Path Planning on Grids. In Proceedings of the National Conference on Artificial Intelligence (Vol. 22, No. 2, p. 1177), 2007. Menlo Park, Calif.: AAAI Press.

[Pitteway, 1985] M. L. V. Pitteway. Algorithms of conic generation. In Fundamental algorithms for computer graphics, 219-237. Springer Berlin Heidelberg, 1985.

[Silver, 2005] David Silver. Cooperative pathfinding. In AIIDE, pages 117–122, 2005.

[Wang, 2011] Ko-Hsin Cindy Wang and Adi Botea. Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. Journal of Artificial Intelligence Research, 42:55-90, 2011.

[Yakovlev, 2015] Konstantin Yakovlev, Egor Baskin, and Ivan Hramoin. Grid-based angle-constrained path planning. In Proceedings of The 38th Annual German Conference on Artificial Intelligence (KI’2015), pages 208-221, 2015. Springer International Publishing.

[Yu, 2013] Yu, J. and LaValle, S.M., Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In AAAI. 2013