AA-SIPP: алгоритм планирования в среде с динамическими препятствиями*

К. С. Яковлев

Федеральный исследовательский центр «Информатика и управление» РАН, г. Москва, Россия

Аннотация. В работе рассматривается задача автоматического планирования траектории мобильного агента на плоскости в среде с динамическими и статическими препятствиями. Эта проблема формулируется как задача поиска пути на графе особого вида. Предлагается новый алгоритм ее решения, основанный на комбинации безопасно-интервального планирования (SIPP) и дополнения исходного графа допустимыми переходами между несмежными вершинами в процессе эвристического поиска решения – AA-SIPP. Описываются свойства этого алгоритма и проводится его экспериментальное исследование, результаты которого свидетельствуют о его превосходстве над предшественником – SIPP.

Ключевые слова: планирование траектории, эвристический поиск, безопасно-интервальное планирование, автономная навигация, мобильный робот.

DOI 10.14357/20718594200105

Введение

Автономная навигация – одна из ключевых функциональных возможностей мобильных интеллектуальных агентов, таких как мобильные роботы, беспилотные автомобили, персонажи компьютерных игр и др. Можно выделить два подхода к решению этой проблемы: реактивный и делиберативный. В рамках реактивного подхода не предполагается накопление и использование поступающей информации, а навигация осуществляется за счет принятия локальных решений (повернуть направо, продолжить движение прямо, ускориться и пр.) по текущим наблюдениям. Наиболее известными методами, реализующими такой подход, являются методы семейства Bug [1, 2]. Делиберативный подход, наоборот, опирается на накопление и обработку входных сенсорных (и других) данных для построения и обновления модели местности (карты) и планирования траектории с использованием этой модели. В данной работе предполагается использование второго подхода, в частности – исследуется задача планирования траектории по известной карте. Обычно в искусственном интеллекте эта задача рассматривается как поиск пути на графе, вершинам которого соответствуют различные положения агента в пространстве, а ребра – элементарным траекториям между положениями, т.е. таким траекториям, следование по которым, может быть выполнено в автоматическом режиме (обычно это отрезки прямых, кривые определенного радиуса, сплайны и пр.) [3].

Наиболее известным алгоритмом поиска путей на графе является алгоритм Дейкстры [4], однако в рассматриваемом случае достаточно легко дать нижнюю оценку длины путей между любыми вершинами графа (например, вычислив Евклидово расстояние между ними) и использовать ее в качестве эвристики для сокра-

* Работа выполнена при частичной поддержке РФФИ (проект № 18-37-20032)

✉ *Яковлев Константин Сергеевич E-mail: yakovlev@isa.ru*

щения пространства поиска и, соответственно, повышения вычислительной эффективности алгоритма. Впервые такой эвристический подход был описан в [5] и стал известен как алгоритм A*. В настоящее же время известны десятки, если не сотни, модификаций этого алгоритма, предназначенные для различных постановок задачи планирования. Известны и модификации этого алгоритма, предназначенные для планирования в среде не только со статическим, но и с динамическими препятствиями (именно этой проблема и посвящена работа). Например, в [6] предлагается подход, основанный на преобразовании динамических препятствий в статические, т.е. вершины графа, которые блокируются этими препятствиями в некоторый временной промежуток, считаются полностью заблокированными и поиск пути осуществляется с помощью обыкновенного A*, что положительно сказывается на вычислительной эффективности поиска. Однако такой подход ведет к неполноте, т.е. алгоритм не гарантирует нахождения решения задачи планирования, даже если оно существует.

В работе [7] вводится дискретное время, агенту разрешается ждать на месте, и таким образом для каждой вершины пространственного графа может быть сгенерировано $|T|$ состояний в пространстве поиска, где T – это временной горизонт, в котором осуществляется планирование, т.е. $T = \{0, 1, \ldots, |T|\}$. Такой метод позволяет гарантировать нахождение решения, однако плохо масштабируется к задачам с большим временным горизонтом и с большим числом вершин графа, описывающего возможные положения агента и препятствий в пространстве. Для устранения этого недостатка в [8-10] были предложены методы, подразумевающие работу с интервалами, а не с отдельными моментами времени, так называемое безопасно-интервальное планирование (safe-interval path planning). Это позволило существенно повысить скорость работы за счет значительного сокращения пространства поиска, так как теперь число состояний, которые потенциально может сгенерировать и рассмотреть алгоритм поиска, пропорционально не числу моментов времени, а числу динамических препятствий.

В данной работе подход безопасно-интервального планирования расширяется за счет возможности достраивать ребра исходного пространственного графа в ходе работы алгоритма, так называемое any-angle планирование [11]. Это особенно актуально в различных робототехнических задачах, где в качестве графовых моделей используются графы регулярной декомпозиции, изображенные на Рис. 1. Обычно при поиске пути на таком графе допускается переход между ортогонально и/или диагонально-смежными вершинами (клетками). Однако, очевидно, что при возможности достраивать ребра в процессе поиска, результирующая траектория получается более короткой и содержащей меньшее число поворотов.

В статье предлагается метод интеграции безопасно-интервального планирования и построения дополнительных ребер в ходе поиска. Описывается алгоритм построения траектории мобильного агента в среде со статическими и динамическими препятствиями реализующий этот метод – AA-SIPP (any-angle safe interval path planning). Проводятся теоретические и

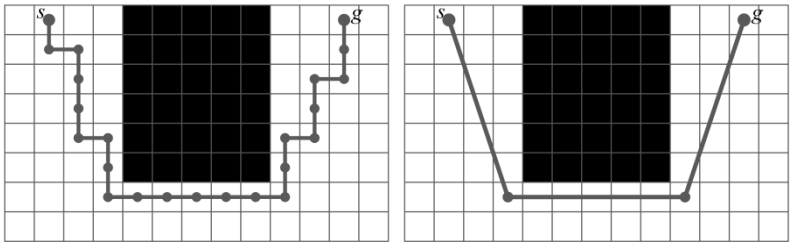

Рис. 1. Граф регулярной декомпозиции, моделирующий окружающую среду мобильного агента и два различных пути на нем

Слева – кратчайший путь, построенный в исходной топологии графа, когда разрешены переходы только между ортогонально-смежными вершинами. Справа – any-angle путь, когда разрешены переходы между парами вершин, которые исходно не образуют ребра. Эти ребра динамически достраиваются в процессе поиска решения

экспериментальные исследования алгоритма. Последние подтверждают перспективность применения предлагаемого алгоритма (в частности, его масштабируемость), а также превосходство над предшественником – алгоритмом SIPP [8].

1. Постановка задачи

Рассмотрим мобильного агента, который моделируется диском радиуса r, задачей которого является перемещение из одной точки евклидового пространства R^2 в другую. Рабочая область агента $W \subset R^2$ – это прямоугольник, часть из которого проходима для агента, а часть – нет: $W = W_{free} \cup W_{obs}$. Известны также начальные и конечные положения, а так же траектории k динамических препятствий, которые также моделируются дисками радиуса r. Формально траектория агента/препятствия – это отображение tr: $\mathfrak{I} \rightarrow W$, где $\mathfrak{I} = [0, \ldots)$ – непрерывное время. Здесь и далее в работе, когда будет идти речь о дискретных моментах времени, будет использоваться обозначение $T = \{0, 1, \ldots, |T|\}$. В рассматриваемой же постановке время непрерывно. Очевидно, при этом, что $T \subset \mathfrak{I}$

Траектория считается допустимой, если при следовании по ней агент/препятствие не пересекает непроходимые области рабочего пространства, т.е. $\rho(pos(tr(t)), W_{obs}) \geq r \ \forall t \in \mathfrak{I}$ где pos – положение центра агента/препятствия в момент времени t при следовании по траектории tr, а ρ – Евклидово расстояние до ближайшего препятствия. Траектория агента считается неконфликтной, если она допустима и при следовании по ней не происходит столкновений с динамическими препятствиями. Последнее условие можно записать как $\rho(pos(tr_0(t)), pos(tr_i)) \geq r \ \forall t \in \mathfrak{I}, i=1, \ldots, k$, где tr_0 – это траектория агента, а tr_i – траектория i-го препятствия. Задача планирования состоит в построении неконфликтной траектории для агента (Рис. 2). Критерием качества решения является время достижения агентом целевого положения, т.е. чем быстрее агент достигает цели, тем лучше.

Для эффективного решения задачи введем ряд допущений, позволяющих трансформировать поставленную задачу в задачу поиска пути на графе. Рабочая область W разбивается на квадратные ячейки (клетки) со стороной $2r$. Этот размер выбран из тех соображений, что агент/препятствие может полностью поместиться в одной клетке. Если внутри клетки содержится хотя-бы одна точка W_{obs}, то такая клетка целиком считается заблокированной. Вводится граф $G=(V, E)$ – граф регулярной декомпозиции (ГРД), вершинам которого соответствуют центры проходимых клеток, а ребрам – пары ортогонально и диагонально смежных вершин. В начальный момент времени агент и динамические препятствия расположены в различных вершинах, целевые состояния также задаются вершинами графа. Для агента (и препятствий) допускаются следующие типы перемещений: перемещение вдоль ребра графа G (перемещение в исходной топологии графа); перемещение по прямой между произвольными вершинами графами (any-angle-перемещения). Допускается также остановка и ожидание (произвольный период времени) в вершинах графа. Таким образом, траектория теперь может быть представлена в виде последовательности указанных выше перемещений и ожиданий. Понятия допустимости и неконфликтности для траекторий сохраняются. Задача также остается в отыскании неконфликтной траектории для агента.

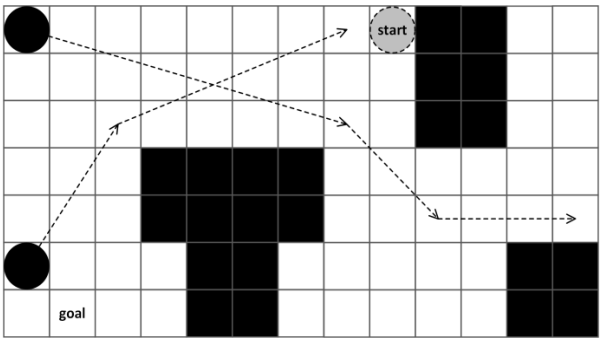

Рис. 2. Задача планирования траектории в среде со статическими и динамическими препятствиями

Траектории динамических препятствий (показаны пунктиром). Начальное положение агента помечено надписью “start”, целевое – “goal”.

В дальнейшем без ограничения общности будем считать, что скорость движения агента/препятствий составляет 1 кл./с, т.е. за 1 секунду (или – условную единицу времени) агент/препятствие преодолевает расстояние равное расстоянию между центрами ортогонально смежных клеток (т.е. $2r$). Считается, что до этой скорости агент и препятствия ускоряются/замедляются мгновенно.

2. Метод решения

2.1. Идея алгоритма

Идея предлагаемого алгоритма заключается в комбинации безопасно-интервального планирования и планирования с достраиванием сегментов траектории вне начальной топологии графа. Опишем эти подходы более подробно.

Планирование с достраиванием сегментов траектории. При эвристическом поиске пути без учета динамических препятствий пространство поиска (дерево решений) индуцируется вершинами графа, то есть каждому элементу дерева решений (состоянию) соответствует определенная вершина графа, а переход от одного состояния к другому возможен только при наличии соответствующего ребра в исходном графе. В задачах же планирования траектории, в особенности, когда речь идет о моделировании рабочего пространства агента графом регулярной декомпозиции, часто возможен переход и между несмежными вершинами графа, так как следование по такому переходу не ведет к столкновению со статическими препятствиями. Это свойство может быть использовано на этапе поиска следующим образом. На каждой итерации алгоритма проверяется – достижимо ли очередное состояние s' не только из непосредственно предшествующего (того состояния, которому соответствует смежная вершина графа) s, но из родителя предшествующего $parent(s)$. Если да, то очевидно, что путь к s' напрямую от $parent(s)$ не длиннее, чем через s, поэтому, несмотря на то, что в исходном графе не было ребра соединяющего s' и $parent(s)$, оно создается в пространстве поиска. На практике, при планировании траектории, это приводит к снижению ее длины и уменьшению числа поворотов. При этом вычислительная сложность алгоритма увеличивается незначительно, так как обычно проверка проходимости сегмента траектории, соединяющего напрямую две несмежные вершины исходного графа, осуществляется достаточно быстро.

Безопасно-интервальное планирование. При планировании в среде с динамическими препятствиями для каждой вершины исходного графа может быть создано несколько состояний пространства поиска, различающихся моментами времени нахождения агента в этой вершине. Это необходимо, так как зачастую агент должен подождать определенное время без движения, чтобы пропустить препятствие и лишь затем продолжить движение к цели. Таким образом, если время дискретно, то для каждой вершины графа может быть создано $|T|$ состояний в дереве поиска, где T – множество рассматриваемых дискретных моментов времени. Если оно не ограничено, то дерево поиска может быть бесконечным (агент может ждать бесконечно долго), поэтому на практике обычно устанавливают временной лимит – либо агент достигает цели к этому моменту, либо задача считается не решенной. Однако даже в этом случае число состояний, соответствующих отдельной вершине графа, которые потенциально могут быть созданы весьма велико, что заметно снижает эффективность поиска. Более того в рассматриваемой постановке, когда время не дискретно, а непрерывно, подобный подход, основанный на перечислении отдельных временных моментов, в которых агент может находиться в вершине не применим, т.к. таких моментов – бесконечное множество. Идея же безопасно-интервального планирования состоит в том, чтобы объединить множество смежных временных моментов, в течение которых агент может находиться в определенной вершине графа, в один безопасный интервал и в процессе поиска рассматривать лишь пары (вершина графа, безопасный интервал). Это позволяет достичь двух целей. Во-первых, это повышает эффективность поиска при дискретном времени, во-вторых, это – единственный способ корректно работать с непрерывным временем.

Предлагаемый подход состоит в том, что безопасно-интервальное планирование сочетается с достраиванием сегментов траектории во время поиска. Метод, реализующий этот подход, назван AA-SIPP, от англ. Any-Angle Safe Interval Path Planning. Перейдем к его более подробному описанию.

2.2. Алгоритм AA-SIPP

Безопасные интервалы. Безопасный интервал для вершины графа – это временной интервал, в течение которого агент может находиться в данной вершине (например, ждать) без столкновения с динамическими препятствиями. Очевидно, что если через вершину графа не проходит ни одно динамическое препятствие, то для нее задан один интервал $[0, +\infty)$. Если через вершину проходит k препятствий, то (за исключением частного случая, когда препятствия проходят через вершину подряд одно за другим) число безопас-

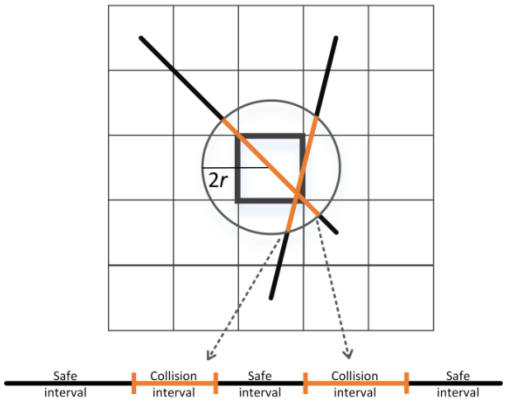

Рис. 3. Множество временных интервалов для вершины графа (центра клетки)

черным показаны безопасные интервалы (можно находиться в клетке), серым – не безопасные (нахождение в клетке ведёт к столкновению с динамическими препятствиями).

ных интервалов равно k+1. Например, если через вершину проходит одно препятствие, то безопасных интервалов два: один до того, как препятствие проходит через вершину, другой – после. На Рис. 3 изображен фрагмент графа регулярной декомпозиции, а также сегменты траекторий двух динамических препятствий. Эти два препятствия задевают в определенные моменты времени центральную вершину (центр клетки, которая отмечена жирным). Таким образом, для этой вершины определены три безопасных интервала (safe intervals), когда агент может находиться в этой вершине. Вне этих интервалов агент не может этого делать, так как это приведет к столкновению с одни из динамических препятствий.

Для расчета безопасных интервалов в рассматриваемом случае, когда и агент, и препятствия моделируются диском радиуса r (при этом r равен половине размера клетки графа регулярной декомпозиции) используется следующий подход. Сначала вычисляются координаты точек пересечения сегментов траекторий динамических препятствий с окружностью радиуса $2r$, ее центр расположен в вершине, для которой вычисляются интервалы. Зная эти координаты, а также, в какие моменты времени препятствия их проходят, вычисляем временные интервалы, которые приводят к столкновениям. Безопасные интервалы теперь могут быть вычислены как их инверсия. Т.е., если интервал столкновений (t_1, t_2), то безопасные интервалы: $[0, t_1]$, $[t_2, +\infty)$.

Заметим также, что безопасные интервалы для каждой вершины графа вычисляются один раз в начале работы алгоритма и в дальнейшем не меняются, а лишь используются для поиска неконфликтной траектории.

Пространство состояний. Также как и в алгоритме SIPP пространство поиска индуцируется парами (вершина графа, безопасный интервал). Таким образом, в отличие от планирования в статической среде, в пространстве состояний может быть несколько элементов, соответствующих одной и той же вершине графа, но различающихся безопасными интервалами. Также для каждого состояния s сохраняются: $time(s)$ – минимальный момент времени, в который возможно достижение этого состояния при переходе из предшествующего – $parent(s)$; $g(s)$ – время достижения этого состояния от начального (в данном случае $g(s)=time(s)$); $h(s)$ – эвристическая оценка времени достижения целевого состояния из s (в рассматриваемом случае она равна отношению расстояния к скорости движения агента). Таким образом, состояние – это кортеж $s=[cfg, interval, g, time, parent, h]$, при этом его идентификатором является пара $[cfg, interval]$, где $cfg(s)$ – это определенная вершина исходного графа, а $interval(s)$ – целое число, являющееся идентификатором интервала (при этом сами границы интервала вычисляются так, как это было описано ранее).

Поиск в пространстве состояний осуществляется следующим образом. Создается список кандидатов на дальнейшее исследование – OPEN, изначально содержащий лишь состояния, соответствующие начальной вершине графа. Таких состояний, напомним, может быть несколько, так как для одной вершины в общем случае может быть определено несколько безопасных интервалов. На каждой итерации из OPEN выбирается элемент с минимальным f-значением, где $f(s)=g(s)+h(s)$. Далее, этот элемент раскрывается (expand). Под этим понимается генерация новых состояний, для которых текущее является родительским, корректное заполнение их g- и $time$-значений, добавление/обновление этих элементов в список OPEN. Так продолжается до тех пор, пока из списка OPEN не будет извлечен для раскрытия элемент, соответствующий целевой вершине графа (целевое состояние). В этот момент алгоритм поиска завершает свою работу, а искомая траектория может быть восстановлена с помощью родительских указателей, начиная от целевого состояния к начальному. Если же список OPEN исчерпан, а целевое состояние так и не было раскрыто, алгоритм возвращает *failure*,

т.е. сигнализирует о невозможности построения решения.

Ключевым отличием алгоритма AA-SIPP от предшественника является процедура генерации новых состояний в процессе поиска. При раскрытии состояния s алгоритм AA-SIPP сначала создает непосредственных последователей s стандартным образом, т.е. рассматривает все исходящие ребра из вершины исходного графа $cfg(s)$ и для каждого конца ребра создает состояние s', соответствующее переходу из вершины $cfg(s)$ – в смежную вершину $cfg(s')$. Далее для каждого s' алгоритм рассматривает возможность перехода из $parent(s)$ и, если это возможно, добавляет и такое состояние в OPEN.

Возможность перехода напрямую из $cfg(parent(s))$ в $cfg(s')$ определяется с помощью модифицированного алгоритма By [12]. Это алгоритм компьютерной графики, исходно предназначенный для визуализации линий на растровом дисплее. Он определяет, какие клетки пересекает отрезок прямой, соединяющий центры клеток графа регулярной декомпозиции (который в данном случае является аналогом растрового дисплея в предположении, что пиксели, образующие изображение, являются квадратными по форме). Модификация же заключается в том, что мы определяем все клетки, расстояние до которых от сегмента траектории меньше r – это те клетки, которые задевает агент при следовании по этому сегменту.

Принцип работы алгоритма проиллюстрирован на Рис. 4. На рисунке жирным отмечены клетки, которые идентифицирует оригинальный алгоритм By. В каждом столбце он выделяет ровно две клетки, расстояние от которых до сегмента прямой – наименьшее. Дополнительная проверка расширяет область на одну клетку вверх и одну клетку вниз (такие клетки отмечены квадратными метками в одном из углов). Далее для этих клеток вычисляются координаты углов, расположенных ближе всего к сегменту, и определяется – превышает ли расстояние от них до прямой r. Если да, то соответствующая клетка также помечается (на рисунке такие дополнительно идентифицированные клетки помечены светло-серым цветом). Если все идентифицированные клетки проходимы, значит, следование по сегменту траектории возможно и состояние, соответствующее этому перемещению, может быть сгенерировано. Временная сложность алгоритма может быть оценена как $O(\Delta)$, где $\Delta=max(\Delta_x, \Delta_y)$, а Δ_x, Δ_y – модуль разницы x, y координат начала и конца сегмента траектории.

Итак, при раскрытии состояния алгоритм AA-SIPP создает два множества потенциальных последователей: одно содержит непосредственных последователей, т.е. состояния, соответствующие переходам между смежными вершинами графа, другое – состояния, соответствующих переходам по достроенным в процессе поиска ребрам графа. Псевдокод алгоритма представлен на Рис. 5.

Строки 1-15 – это стандартный цикл алгоритма эвристического поиска (извлечение лучшего состояния из OPEN, раскрытие состояния, обновление OPEN). Эти шаги идентичны для SIPP и AA-SIPP за исключением шагов 7-8, которые отсутствуют в оригинальном алгоритме. В строках 16-30 (функция getSuccesors) происходит создание элементов пространства поиска, соответствующих переходам из текущего состояния s либо из $parent(s)$, если такое движе-

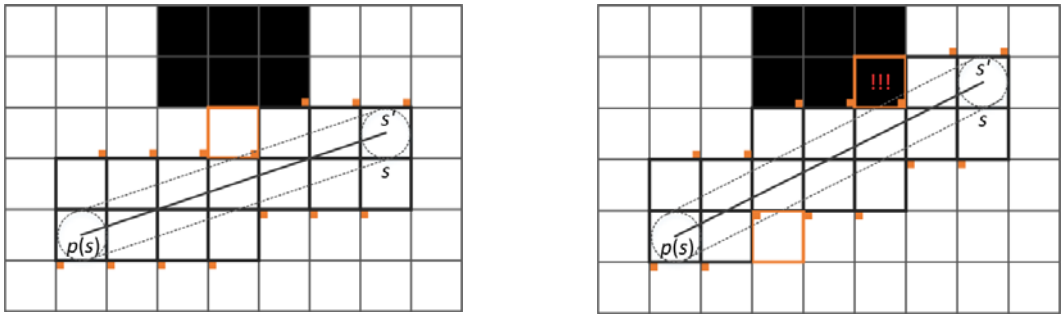

Рис. 4. Проверка проходимости сегмента траектории

Жирным черным цветом выделены клетки, идентифицируемые алгоритмом By. Небольшими квадратами в углу отмечены клетки, требующие дополнительной проверки. Светло-серые клетки – клетки, идентифицированные в ходе такой проверки. На левой части рисунка показана ситуация, когда переход возможен, справа – не возможен

Algorithm 1: AA-SIPP

```
1  g(s_start) = 0; OPEN = ∅;
2  insert s_start into OPEN with f(s_start) = h(s_start);
3  while s_goal is not expanded do
4      remove s with the smallest f-value from OPEN;
5      for each cfg in NEIGHBORS(s.cfg) do
6          successors = getSuccessors(cfg, s);
7          if cfg is reachable from parent(s).cfg then
8              successors = successors ∪
                 getSuccessors(cfg, parent(s));
9          for each s' in successors do
10             if s' was not visited before then
11                 f(s') = g(s') = ∞;
12             if g(s') > g(s) + c(s, s') then
13                 g(s') = g(s) + c(s, s');
14                 updateTime(s');
15                 insert s' into OPEN with
                     f(s') = g(s') + h(s');
16 Function getSuccessors(cfg, s)
17     successors = ∅;
18     m_time = time to reach cfg from s.cfg;
19     start_t = time(s) + m_time;
20     end_t = endTime(interval(s)) + m_time;
21     intervals =get all safe intervals for cfg;
22     for each safe interval i in intervals do
23         if startTime(i) > end_t or
             endTime(i) < start_t then
24             continue;
25         t=earliest arrival time from s to cfg during
             interval i with no collisions;
26         if t does not exist then
27             continue;
28         s'=state of configuration cfg with interval i
             and time t;
29         insert s' into successors;
30     return successors;
```

Рис. 5. Псевдокод алгоритма AA-SIPP

ние возможно по результатам описанной выше проверки. Покажем процедуру генерации последователей более подробно.

На вход getSuccesors передается состояние s, которое будет считаться родительским для создаваемых состояний s', а также вершина графа *cfg*, в которую совершается перемещение. Для вершины *cfg* в общем случае определенно k безопасных интервалов. Соответственно, всего может быть создано k новых состояний поиска s'. Каждое такое состояние соответствует перемещению из s в определенный безопасный интервал s'. Для генерации каждого состояния последователя выполняется цикл по всем k безопасным интервалам *cfg*. Для каждого интервала i необходимо установить, может ли агент достичь вершину *cfg* в безопасный интервал i, осуществляя переход из s.

Заметим, что вершина *cfg* может достигаться из s минимум в момент времени $start_t = time(s) + m_{time}$, где m_{time} – время перемещения по сегменту траектории (длина сегмента, деленная на скорость агента). Другими словами, агент после достижения s (этот момент известен и сохранен в $time(s)$) сразу же продолжает движение в *cfg*. Максимальное же время достижения равно $end_t = endtime(interval(s)) + m_{time}$, где первое слагаемое – это последний момент времени безопасного интервала состояния s. End_t достигается, если агент насколько возможно долго ждет в s (до конца безопасного интервала), а затем совершает перемещение. Очевидно, что если *start* или end_t лежат вне интервала i, то агент не может достигнуть *cfg* в безопасный интервал i, и соответствующее состояние не создается. Если же достижение *cfg* возможно, то создается состояние s', идентифицируемое как [*cfg*, i], при этом в качестве $time(s')$ устанавливается минимально возможное время достижения *cfg*. Для расчета этого времени необходим учет динамических препятствий, пересекающих соответствующий сегмент траектории агента. Когда речь идет об агенте и препятствиях, моделируемых дисками, движущихся прямолинейно и равномерно, то применим следующий подход. Изначально полагается, что агент начинает движение без задержки, т.е. в момент времени $time(s)$. Используя уравнения движения, можно вычислить приведет ли такое движение к столкновению. Если да, то добавляется задержка δ и проверка повторяется. Так происходит до тех пор, пока не будет выполнено одно из следующих условий: либо при начале движения в момент $time(s) + n\delta$ столкновения не происходит (здесь n – число итерации применения задержки), либо $time(s) + n\delta > EndTime(interval(s))$. В последнем случае движение невозможно, так как агент не может ожидать так долго в вершине $cfg(s)$ (формально, момент начала движения лежит вне безопасного интервала), и новое состояние поиска не создается. В первом случае выполняется дополнительная проверка, что после ожидания в исходной вершине и движения в текущую вершину агент ее достигает в рассматриваемый безопасный интервал. Если это выполняется, то создается состояние s', для которого $time(s') = time(s) + n\delta + m_{time}$.

2.3. Свойства алгоритма AA-SIPP

Свойство 1. Алгоритм AA-SIPP гарантирует нахождение решения, если такое существует в исходной топологии графа. Выполнимость этого свойства напрямую следует из того, что алгоритм AA-SIPP на каждом шаге работы создает теже состояния поиска, что и алгоритм SIPP, плюс некоторые дополнительные. Поскольку известно, что алгоритм SIPP гарантирует нахождение решения, если оно существует, то, используя то же множество элементов пространства поиска (плюс дополнительные состояния, отвечающие за переходы между несмежными вершинами) и ту же стратегию их исследования, алгоритм AA-SIPP также найдет решение.

Свойство 2. Если решение задачи существует в исходной топологии графа, то алгоритм AA-SIPP гарантирует отыскание траектории, которая не хуже (с точки зрения времени следования по ней), чем оптимальная траектория в исходной топологии графа.

Выполнимость этого свойства гарантирует отыскание оптимального решения в исходной топологии графа, а дополнительные состояния, которые вводит в рассмотрение AA-SIPP, соответствуют более коротким переходам между вершинами этого графа и не могут, очевидно, приводить к построению более длинных траекторий (соответственно, и время следования по ним не будет больше).

В более общем виде Свойство 1 и Свойство 2 могут быть сформулированы следующим образом. Алгоритм AA-SIPP гарантирует нахождение решения, если такое существует в исходной топологии графа, при этом время достижения агентом цели минимально, т.е. оно не может быть уменьшено за счет следования по другой траектории к цели, состоящей из ребер исходного графа.

3. Экспериментальное исследование

Экспериментальное исследование проводилось на двух картах, представленных в виде графов регулярной декомпозиции размером 64 × 64 клетки. Ребра графа были заданы ортогональными переходами между центрами клеток. Первая карта (empty grid) не содержала ни одного статического препятствия. Вторая карта (warehouse grid) моделировала складское помещение и содержала 10 прямоугольных препятствий размером 20 × 2 (Рис. 6). Число динамических препятствий в обоих случаях варьировалось от 0 до 300 с шагом 50, их начальные и целевые позиции выбирались случайно. При создании отдельного задания динамические препятствия вводились в рабочее пространство последовательно одно за другим следующим образом. Для каждого числа препятствий было случайным образом сгенерировано 100 заданий для агента. Заметим, что при таком подходе не все задания являются априори разрешимыми, так как при высокой плотности динамических препятствий возможна ситуации, когда они блокируют цель агента.

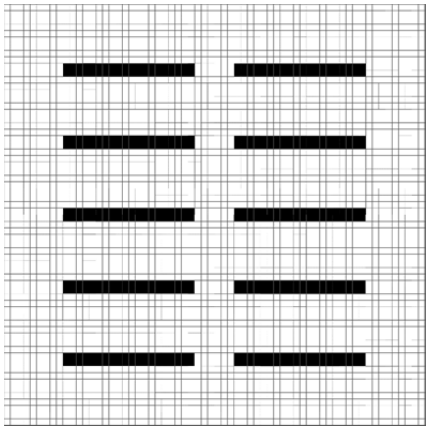

Рис. 6. Карта складского помещения (представленная в виде графа регулярной декомпозиции размером 64 на 64 клетки), используемая в тестах

Одиночный эксперимент заключался в построении неконфликтной траектории в заданной динамической среде. Между собой сравнивались алгоритмы SIPP и предлагаемый в работе алгоритм AA-SIPP. Первый осуществлял поиск в исходной топологии графа, второй – с достраиванием сегментов траектории между несмежными вершинами. Для корректности сравнения оба алгоритма были программно реализованы с использованием одних и тех же приемов программирования и структур данных. Тестирование осуществлялось на персональном компьютере конфигурации AMD FX-8350 @ 4.0 GHz, 16 Gb RAM.

В Табл. 1 и Табл. 2 приведены усредненные показатели времени работы алгоритмов и стоимости найденных решений (т.е. времени прохождения агентом спланированной траектории) – столбцы Runtime, Cost. В столбце Success Rate указан процент успешно решенных заданий.

Табл.1. Результаты эксперимента на пустой карте, содержащей 0-300 динамических препятствий

	64x64 Empty Grid					
	AA-SIPP			**SIPP (Cardinal only)**		
Obstacles	*Success Rate*	*Cost*	*Runtime*	*Success Rate*	*Cost*	*Runtime*
0	100,00%	45,440	0,00248	100,00%	58,291	0,00160
50	100,00%	46,955	0,00995	100,00%	59,122	0,00339
100	99,60%	48,341	0,01669	99,60%	59,814	0,00525
150	98,00%	49,452	0,02179	98,00%	60,569	0,00683
200	97,60%	51,146	0,02756	97,60%	61,929	0,00893
250	96,40%	53,333	0,03513	96,40%	64,066	0,01146
300	94,80%	55,524	0,04257	94,80%	66,385	0,01407

Табл. 2. Результаты эксперимента на карте, содержащей 10 статических и 0-300 динамических препятствий

	64x64 Warehouse Grid					
	AA-SIPP			**SIPP (Cardinal only)**		
Obstacles	*Success Rate*	*Cost*	*Runtime*	*Success Rate*	*Cost*	*Runtime*
0	100,00%	48,245	0,00489	100,00%	57,567	0,00180
50	100,00%	51,455	0,01640	100,00%	59,859	0,00450
100	99,20%	54,280	0,02505	99,20%	62,573	0,00703
150	98,80%	58,346	0,03685	98,80%	66,395	0,01039
200	98,00%	62,573	0,04898	98,00%	70,836	0,01378
250	97,20%	66,066	0,05958	97,20%	74,286	0,01629
300	95,20%	69,413	0,06924	95,20%	78,132	0,01984

В результате анализа полученных данных можно сделать следующие выводы. Во-первых, совпадение метрики SR является эмпирическим подтверждением свойства полноты алгоритма AA-SIPP, т.е. алгоритм AA-SIPP, так же как и предшественник гарантирует нахождение решения, если оно существует, либо возвращает failure и корректно завершается, если решения нет. Во-вторых, алгоритм AA-SIPP позволяет заметно улучшить качество решения – до 26% по сравнению с SIPP. Наибольший эффект достигается на пустой карте при малом числе динамических препятствий. На карте, содержащей 300 динамических и 10 статических препятствий, разница в показателях составляет 12,5%. В-третьих, время работы обоих алгоритмов в зависимости от числа динамических препятствий растет одинаково, при этом очевидно, что время работы алгоритма AA-SIPP в среднем больше, чем у предшественника. Это связано с рассмотрением дополнительных состояний поиска, а также с проверками на проходимость сегментов траектории между несмежными вершинами графа. В целом, полученные результаты свидетельствуют о превосходстве представленного алгоритма над предшественником по качеству отыскиваемых решений и о его практической применимости для решения задач с большим числом динамических препятствий (порядка сотен).

Заключение

В работе задача планирования траектории сформулирована как задача эвристического поиска пути на графе. Предложен новый алгоритм ее решения, предназначенный для динамических сред, т.е. сред, содержащих как статические, так и динамические препятствия (траектории, движения которых известны планировщику) – AA-SIPP. Алгоритм развивает идеи безопасно-интервального планирования и динамического дополнения пространства состояний поиска за счет построения сегментов траекторий, отсутствующих в изначальной топологии графа. Описаны теоретические свойства алгоритма и проведено его эмпирическое исследование. К дальнейшим направлениям исследований можно отнести разработку модификаций алгоритма, основанных лишь на частичном исследовании пространства состояний, что положительно повлияет на быстродействие, разработку модификаций, учитывающих более реалистичную кинематическую модель движения агента (ускорение, замедление), проведение экспериментов на реальных робототехнических платформах.

Литература

1. Lumelsky V., Stepanov A. Dynamic path planning for a mobile automaton with limited information on the environment // IEEE transactions on Automatic control. 31(11). 1986. pp. 1058-1063.
2. McGuire K.N. et al. A comparative study of bug algorithms for robot navigation // Robotics and Autonomous Systems. 121. 2019. p.103261.
3. Яковлев К.С., Баскин Е.С. Графовые модели в задаче планирования траектории на плоскости // Искусственный интеллект и принятие решений. 2013. №1.С.5-12.
4. Dijkstra E.W. A note on two problems in connexion with graphs // Numerische mathematic.1(1). 1959. pp.269-271.
5. Hart P.E., Nilsson N.J. and Raphael B. A formal basis for the heuristic determination of minimum cost paths // IEEE transactions on Systems Science and Cybernetics. 4(2). 1968. pp.100-107.
6. Rufli M., Ferguson, D., Siegwart R. Smooth path planning in constrained environments // In 2009 IEEE International Conference on Robotics and Automation (ICRA 2009). 2009. pp. 3780-3785.
7. Silver D. Cooperative Pathfinding // In Proceedings of The First Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2005). 2005. pp.117-122.
8. Phillips M., Likhachev M. SIPP: Safe interval path planning for dynamic environments // In Proceedings of The 2011 IEEE International Conference on Robotics and Automation (ICRA 2011). 2011. pp. 5628–5635.
9. Van Den Berg J.P., Overmars M.H. Roadmap-based motion planning in dynamic environments // IEEE Transactions on Robotics. 21(5). 2005. pp.885-897.
10. ter Mors A.W., Witteveen C., Zutt J., Kuipers F.A. Context-Aware Route Planning // In Proceedings of The 8th German Conference on Multi Agent Systems Technologies (MATES 2010). 2010. pp. 138-149.
11. Nash A., Daniel K., Koenig S., Felner A. Theta*: Any-angle path planning on grids // In Proceedings of The 22nd AAAI Conference on Artificial Intelligence (AAAI 2007). 2007. pp. 1177–1183.
12. Wu X. An efficient antialiasing technique // In Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-1991).1991. pp. 143–152.

AA-SIPP: any-angle Path Finding Amidst Static and Dynamic Obstacles

K. S. Yakovlev

Federal Research Center for Computer Science and Control of RAS, Moscow, Russia

Abstract. A problem of path planning for an agent moving amidst static and dynamic obstacles is considered in the paper as a graph search problem. A novel method to solve it is introduced. The method is based on the combination of two complimentary ideas: any-angle path finding, i.e. augmenting the graph with the edges that connect initially non-adjacent vertices, and safe interval path planning, i.e. grouping the timesteps into the continuous intervals and operating them during heuristic search for a solution. Theoretical and empirical investigation of the algorithm are performed.

Keywords: Path Panning, Path Finding, Heuristic Search, Safe Interval Path Planning, Any-angle Path Planning.

DOI 10.14357/20718594200105

References

1. Lumelsky V., Stepanov A. Dynamic path planning for a mobile automaton with limited information on the environment // IEEE transactions on Automatic control, 31(11), 1986. pp. 1058-1063.
2. McGuire K.N. et al, 2019] McGuire K.N. de Croon G.C.H.E., Tuyls K. A comparative study of bug algorithms for robot navigation // Robotics and Autonomous Systems, 121, 2019. p.103261.
3. Yakovlev K.S., Baskin E.S. Graph Models for Path Planning // Artificial Intelligence and Decision Making, 1, 2013. Pp.5-12.
4. Dijkstra E.W. A note on two problems in connexion with graphs // Numerische mathematik, 1(1), 1959. pp.269-271.
5. Hart P.E., Nilsson N.J. and Raphael B. A formal basis for the heuristic determination of minimum cost paths // IEEE transactions on Systems Science and Cybernetics, 4(2), 1968. pp.100-107.
6. Rufli M., Ferguson, D., Siegwart R. Smooth path planning in constrained environments // In 2009 IEEE International Conference on Robotics and Automation (ICRA 2009), 2009. pp. 3780-3785.
7. Silver D. Cooperative Pathfinding // In Proceedings of The First Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2005), 2005. pp.117-122.
8. Phillips M., Likhachev M. SIPP: Safe interval path planning for dynamic environments // In Proceedings of The 2011 IEEE International Conference on Robotics and Automation (ICRA 2011), 2011. pp. 5628–5635.
9. Van Den Berg J.P., Overmars M.H. Roadmap-based motion planning in dynamic environments // IEEE Transactions on Robotics, 21(5), 2005. pp.885-897.
10. ter Mors A.W., Witteveen C., Zutt J., Kuipers F.A. Context-Aware Route Planning // In Proceedings of The 8th German Conference on Multi Agent Systems Technologies (MATES 2010), 2010. pp. 138-149.
11. Nash A., Daniel K., Koenig S., Felner A. Theta*: Any-angle path planning on grids // In Proceedings of The 22nd AAAI Conference on Artificial Intelligence (AAAI 2007), 2007. pp. 1177–1183.
12. Wu X. An efficient antialiasing technique // In Proceedings of The 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-1991), 1991. pp. 143–152.