УДК 519.16

Алгоритмы решения задачи быстрого поиска пути на географических картах

© М.А. Басараб, А.Б. Домрачева, В.М. Купляков

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Рассмотрены алгоритмы, позволяющие для подготовленного ландшафта предоставить один из возможных вариантов пути из одной точки в другую на географической карте с учетом особенностей проходимости местности. Описаны методы, которые условно можно разделить на следующие классы: алгоритмы поиска кратчайшего пути (Дейкстры); алгоритмы поиска субоптимального пути (A и его модификации, в частности Theta*); алгоритмы постобработки маршрутов (удаление точек, лежащих на одной прямой; Line of Sight). Особое внимание уделено эвристическим алгоритмам, позволяющим найти близкий к оптимальному и в достаточной степени реалистично выглядящий маршрут. Описаны способы интерпретации ландшафта. Представлены выводы о применимости отдельных алгоритмов и их комбинаций. Сделан вывод о целесообразности использования различных алгоритмов на этапах построения предварительного варианта маршрута и его оптимизации. Произведен анализ различных методов поиска пути: их длины, сложности, числа точек поворота, суммарного угла отклонения.*

Ключевые слова: *поиск пути, алгоритм Дейкстры, алгоритм A*, алгоритм Theta*.*

Введение. При создании симуляторов, подразумевающих перемещение различных типов объектов по большим территориям с учетом текущей тактической обстановки, возникают проблемы с выбором алгоритма поиска оптимального пути, так как на его использование накладываются ограничения, вызванные следующими факторами:

- большой объем данных реальных карт местности, превышающий объем оперативной памяти, поэтому в большинстве случаев нет возможности хранить полную информацию о промежуточном состоянии маршрута в памяти;
- сложность представления ландшафта (территории), по которому перемещаются объекты, это требует минимизации числа запросов на определение проходимости определенного участка пути;
- большой разброс в сложности получаемого пути: оптимальным решением может оказаться как прямая, так и сильно изломанная линия.

Кроме указанных, при реализации конкретных алгоритмов может возникать ряд других проблем.

Существует большое количество алгоритмов, позволяющих определить маршрут, по которому можно попасть из одной точки в другую. Эти алгоритмы можно разбить на две группы:

- алгоритмы, позволяющие определить оптимальный путь [1];
- алгоритмы, позволяющие найти субоптимальный путь [2–5].

В первой группе для нахождения решения требуется полностью исследовать некоторую область. Самым простым способом поиска оптимального пути является полный перебор всех возможных маршрутов. В этом случае найденный путь будет кратчайшим. Однако такой способ неприменим в большинстве случаев из-за чрезмерных накладных расходов, так как требуется полное исследование всей карты и хранение ее в памяти.

В связи с этим на первый план выходит разработка алгоритмов поиска субоптимальных путей. Примером являются эвристические алгоритмы, которые на каждом шаге приближаются к конечной точке. Однако при поиске одного из близких к оптимальному пути следует учитывать, что изначально трудно точно предсказать, какой именно вариант будет выбран. К тому же, одним из требований к маршруту является его реалистичный внешний вид. В этом случае можно использовать различные алгоритмы при выборе направления на каждом шаге либо различные алгоритмы постобработки маршрутов.

Наряду с алгоритмами, позволяющими непосредственно определить путь, следует учитывать разнообразные алгоритмы постобработки полученного маршрута. Они позволяют разбить исходную задачу на несколько подзадач, например:

- определение возможного направления движения;
- определение ключевых точек маршрута и удаление точек, лежащих на одной прямой;
- спрямление отдельных участков пути;
- «огрубление» предварительного представления пути.

В данной работе на основе тестового анализа было предложено включить в библиотеку такие алгоритмы поиска пути:

- алгоритм Дейкстры;
- алгоритм A*;
- алгоритм Theta*.

В качестве алгоритмов постобработки выбраны следующие:

- удаление точек, лежащих на одной прямой;
- применение функции Line of Sight, используемой в алгоритме Theta*.

На примерах решения тестовых задач показана эффективность реализованной библиотеки.

Представление ландшафта. Большое влияние на качество нахождения пути оказывает способ представления ландшафта. В основном для этого используется представление карты в виде матрицы проходимостей. Выбор формы ячейки играет большую роль при реализации алгоритма — влияет на длину полученного пути. В таблице приведены основные используемые формы ячеек проходи-

мости и увеличение (в среднем) найденного пути по сравнению с оптимальным.

Формы ячейки сетки проходимости

Форма ячейки	Число соседних ячеек	Увеличение длины пути, %
Треугольник	3	100
	6	15
Квадрат	4	41
	8	8
Шестиугольник	6	15
	12	4

Исходя из данных, представленных в [2], наиболее выгодным является использование квадратных ячеек с рассмотрением 8 ближайших соседей (в случае шестиугольника и 12 соседей результат получился несколько лучше, однако такая реализация обычно оказывается на порядок сложнее).

Учитывая сказанное, при решении поставленной задачи мы в большинстве случаев будем интерпретировать карту набором квадратных ячеек, составляя из них сетку. Квадраты сетки будут маркироваться числами 0 или 1 (проходим или не проходим), т. е. для каждого типа объекта и типа местности, по которому он движется, определяется собственно факт проходимости. В перспективе задача может быть усложнена введением некоторого штрафа, отражающего различный характер проходимости ячейки.

Алгоритмы поиска оптимального пути. Алгоритмы этой группы позволяют найти оптимальный путь, но требуют большого ресурса памяти для хранения информации о том, какой удельный вес имеет каждая ячейка на карте, если она является частью оптимального пути. К тому же, время выполнения подобных алгоритмов также очень велико, поскольку требуется рассмотреть всю область, где потенциально может быть проложен маршрут. В основном подобного рода алгоритмы применяются либо на картах небольшого размера, где точность найденного пути играет большую роль, либо на этапе подготовки, когда требуется проложить некоторые не меняющиеся со временем магистрали.

Рассмотрим наиболее известные алгоритмы поиска оптимального пути [1].

Алгоритм поиска в ширину. При работе алгоритма поиска в ширину (Breadh-first search, BFS) сетка представляется в виде взвешенного графа, где из каждой вершины возможен переход в одну из 8 соседних вершин (кроме граничных). Веса вершин — 0 либо ∞ (для случая полной непроходимости). Для работы алгоритма требуется организовать очередь, в которую будут последовательно добав-

ляться вершины. Помимо этого, чтобы иметь возможность восстановить пройденный путь, понадобится массив «предков», где будет указано, из какой вершины ведет путь в данную. Необходим также номер стартовой вершины *s* (координаты стартовой ячейки). Сам алгоритм можно понимать как процесс «поджигания» графа: на нулевом шаге «поджигаем» только вершину *s*. На каждом следующем шаге «огонь» с уже «горящей» вершины перекидывается на всех ее соседей. Таким образом, за одну итерацию алгоритма происходит расширение «кольца огня» в ширину на единицу (отсюда и название алгоритма).

Более строго это можно представить следующим образом. Создадим очередь, в которую будут помещаться «горящие» вершины, а также заведем булевский массив, в котором для каждой вершины будем отмечать, «горит» она или нет (иными словами, была ли она посещена). Изначально в очередь помещается только вершина *s*. Затем реализуется цикл: пока очередь не пуста, достать из ее головы одну вершину, просмотреть все ребра, исходящие из этой вершины, и, если какие-то из просмотренных вершин еще не «горят», «поджечь» их и поместить в конец очереди.

В итоге, когда очередь опустеет, алгоритм BFS обойдет все достижимые из *s* вершины, причем до каждой дойдет кратчайшим путем. Также можно посчитать длины кратчайших путей (для чего просто надо завести массив длин путей) и компактно сохранить информацию, достаточную для восстановления всех этих кратчайших путей (завести массив «предков», в котором для каждой вершины необходимо хранить номер вершины, из которой мы в нее попали).

При достижении конечной вершины алгоритм прекращает свою работу и восстанавливается общий путь до конечной вершины.

Одна из возможных оптимизаций алгоритма BFS заключается в добавлении поиска направленности движения. Зачастую путь между двумя вершинами оказывается заключенным в соответствующий прямой угол, поэтому добавлять вершину в очередь можно, не обходя последовательно все соседние с ней, а начиная от биссектрисы соответствующего прямого угла и расходясь в стороны, при этом беря поочередно вершины с каждой из сторон по отношению к биссектрисе.

Другая возможность оптимизации — двунаправленный поиск в ширину. Это улучшает простой поиск BFS тем, что запускаются два одновременных поиска в ширину из стартового и конечного узлов, и процесс останавливается, когда узел из одного фронта поиска находит соседний узел из другого фронта. Это может улучшить простой поиск в ширину (обычно в 2 раза), который остается при этом не очень эффективным.

Одним из главных достоинств алгоритма BFS является то, что он гарантированно находит кратчайший путь из начальной вершины в конечную. Однако недостатком является необходимость исследования большого числа сторонних точек, что требует излишних затрат памяти для хранения информации, а также дополнительного времени.

Алгоритм поиска в глубину. Поиск в глубину (Depth-first search, DFS) проверяет каждый возможный путь целиком, прежде чем перейти к другому возможному маршруту. При обходе дерева маршрутов каждый раз выбирается левая ветвь, пока не будет достигнут конечный узел или найдена цель. Если текущий узел — конечный, совершается подъем на один уровень вверх и выбирается правая ветвь дерева, а из нее продолжается движение по левым ветвям до тех пор, пока не встретится цель или конечный узел. Эта процедура повторяется, пока не будет обнаружена цель или пройден последний узел пространства.

Одной из возможных оптимизаций алгоритма DFS является алгоритм последовательных приближений при поиске в глубину (Iterative deeping depth-first search, IDDFS). В действительности в алгоритме поиска в глубину существует еще одна проблема — выбор правильной глубины остановки. Если она будет слишком маленькой, то путь не будет найден, если слишком большой, то можно потратить много времени и денег впустую. Эти проблемы решаются итеративным углублением — приемом, при котором выполняется DFS с увеличением глубины до тех пор, пока путь не будет найден. При поиске пути можно не начинать с глубины, равной единице, а сразу начать с глубины, равной расстоянию по прямой от старта до цели. Этот поиск является асимптотически оптимальным среди всех переборных алгоритмов по времени и памяти.

Алгоритм Дейкстры. Введем массив $d[\ ]$, в котором для каждой вершины v будем хранить текущую длину $d[v]$ кратчайшего пути из начальной вершины s в конечную вершину v. Для s эта длина равна 0, а для всех остальных — бесконечности (в качестве бесконечности обычно выбирают достаточно большое число, заведомо большее возможной длины пути): $d[v] = \infty$, $v \neq s$. Кроме того, для каждой вершины v будем хранить информацию о том, помечена она или нет, т. е. заведем булевский массив $u[\ ]$. Изначально все вершины не помечены $u[v] = \text{false}$.

Сам алгоритм Дейкстры состоит из n итераций. На очередной итерации выбирается вершина v с наименьшей длиной $d[v]$ среди еще не помеченных (на первой итерации будет выбрана стартовая вершина s). Выбранная таким образом вершина v считается «помеченной». Далее, на текущей итерации, из вершины v производятся релаксации:

просматриваются все ребра (v, t_0), исходящие из вершины v, и для каждой вершины t_0 алгоритм пытается улучшить значение $d[t_0]$. Пусть длина текущего ребра равна len, тогда шаги алгоритма (релаксации) выглядят так:

$$d[t_0] = \min(d[t_0], d[v] + \text{len}).$$

На этом текущая итерация заканчивается, алгоритм переходит к следующей итерации (снова выбирается вершина с наименьшей длиной $d[v]$, из нее производятся релаксации и т. д.). При этом после n итераций все вершины графа станут помеченными и алгоритм свою работу завершает. Утверждается, что найденные значения $d[v]$ и есть искомые длины кратчайших путей из s в v.

Следует отметить, что если не все вершины графа достижимы из вершины s, то значения $d[v]$ для них так и останутся бесконечными. Понятно, что несколько последних итераций алгоритма будут как раз выбирать эти вершины, но никакой полезной работы производить не будут (поскольку бесконечное расстояние не сможет прорелаксировать другие, пусть и бесконечные расстояния). Поэтому алгоритм можно останавливать, как только будет выбрана вершина с бесконечным расстоянием.

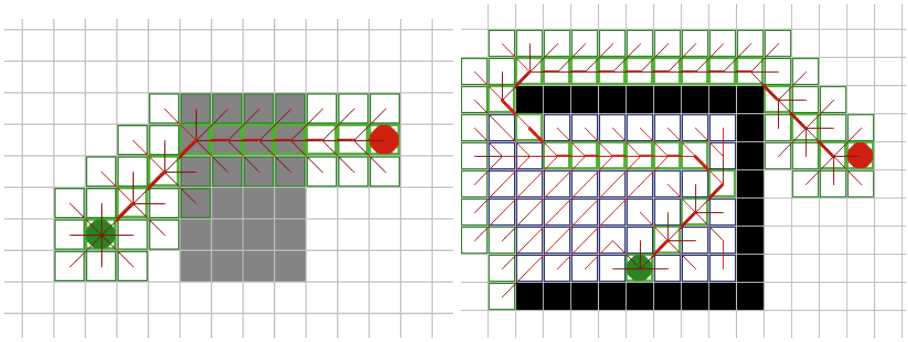
Обычно нужно знать не только длины кратчайших путей, но и получить сами пути. Покажем, как сохранить информацию, достаточную для последующего восстановления кратчайшего пути из s до любой вершины. Для этого достаточно так называемого массива «предков»: массива $p[\ ]$, в котором для каждой вершины $v \neq s$ хранится номер вершины $p[v]$, являющейся предпоследней в кратчайшем пути до вершины v. Здесь используется тот факт, что если мы возьмем кратчайший путь до какой-то вершины v, а затем удалим из этого пути последнюю вершину, то получится путь, оканчивающийся некоторой вершиной $p[v]$, и этот путь будет кратчайшим для вершины $p[v]$. Если имеется массив «предков», то кратчайший путь можно будет восстановить по нему, просто каждый раз беря «предка» от текущей вершины, пока мы не придем в стартовую вершину s — так получится искомый кратчайший путь, но записанный в обратном порядке. Итак, кратчайший путь P до вершины v равен: $P = (s, \ldots, p[p[p[v]]], p[p[v]], p[v], v)$.

Массив «предков» строится просто: при каждой успешной релаксации, т. е. когда из выбранной вершины v происходит улучшение расстояния до некоторой вершины t_0, записываем, что «предком» вершины t_0 является вершина v: $p[t_0] = v$.

Алгоритм Дейкстры учитывает веса при продвижении по графу, к тому же он обновляет данные в ранее достигнутых узлах пути. Таким образом, гарантированно будет найден кратчайший путь, если он су-

ществует. Однако у этого алгоритма есть слабая сторона — поиск в ширину: он игнорирует направление к цели.

Алгоритм «Лучший — Первый». Этот эвристический алгоритм принимает во внимание знания о пространстве поиска для направления своих усилий. Он похож на алгоритм Дейкстры. Отличие заключается в том, что узлы в списке оцениваются по приблизительному оставшемуся расстоянию до цели, кроме того «Лучший — Первый» не требует наличия обновлений. Этот алгоритм достаточно быстрый и направляется по прямой к цели, однако он имеет и недостатки. На рис. 1, *а* показано, что он не принимает во внимание накопленную стоимость пути, направляясь по прямой через труднопроходимую зону. На рис. 1, *б* можно увидеть, что найденный путь изгибается вокруг препятствия аналогично пути, полученному алгоритмом трассировки. Недостатком является и то, что вершина, до которой ищется путь, может оказаться в поддереве, рассматриваемом в последнюю очередь.



а *б*

Рис. 1. Иллюстрация работы алгоритма «Лучший — Первый»: зеленый маркер — начало пути; красный маркер — конец пути; зеленый контур — анализируемые ячейки; красные линии — труднопроходимая зона; черные ячейки — непроходимая зона.

Эвристические алгоритмы определения субоптимального пути. Субоптимальные алгоритмы работают по принципу пошагового улучшения текущего результата. Выбирается некоторая эвристическая функция, с ее помощью на каждом шаге можно выбрать ячейку сетки, расстояние от которой до конечной точки будет иметь минимальное значение (на основании значения эвристической функции). Преимуществом подобных алгоритмов является меньшее количество используемых ресурсов по сравнению с алгоритмами определения оптимального пути. Многие эвристические алгоритмы позволяют выдать точку, являющуюся предположительно наиболее близкой к конечной, и маршрут до нее, что важно для

алгоритмов моделирования, работающих в реальном времени, когда требуется получить не конечный путь, а некоторый его участок.

Алгоритм A*. Алгоритм A* является одним из наиболее известных алгоритмов поиска субоптимального пути [2]. Он находит маршрут от начальной вершины к конечной с наименьшей стоимостью. Порядок обхода определяется эвристической функцией «расстояние + стоимость»: $f(x) = g(x) + h(x)$. Функция $h(x)$ должна быть допустимой эвристической оценкой, т. е. не должна переоценивать расстояние к целевой вершине. Она может представлять собой расстояние до цели по прямой, так как это наименьшее расстояние между двумя точками.

Алгоритм A* пошагово просматривает все пути от начальной вершины до конечной, пока не найдет минимальный. Как и все информированные алгоритмы поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. От жадного алгоритма (который также является алгоритмом поиска по первому лучшему совпадению) его отличает то, что при выборе вершины он учитывает, помимо прочего, весь пройденный до нее путь (составляющая $g(x)$ — это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме).

Алгоритм A* является полным, т. е. всегда находит решение, если оно существует. Он также оптимально эффективен для заданной эвристики h. Это значит, что любой другой алгоритм исследует не меньше узлов, чем алгоритм A* (за исключением случаев, когда существует несколько частных решений с одинаковой эвристикой, точно соответствующей стоимости оптимального пути).

В то время как алгоритм A* оптимален для «случайно» заданных графов, нет гарантии, что он сделает свою работу лучше, чем более простые, но и более информированные относительно проблемной области алгоритмы. Например, в некотором лабиринте может потребоваться сначала идти по направлению от выхода, и только потом повернуть назад. В этом случае обследование вначале тех вершин, которые расположены ближе к выходу (по прямой), будет потерей времени.

Рассмотрим модификации алгоритма A* и другие эвристические алгоритмы субоптимального поиска.

Beam search (поиск по лучу). В случае обычной реализации алгоритма A* в списке задействованных вершин сохраняются все вершины, которые могут потребоваться для поиска пути. Алгоритм Beam search [5] накладывает ограничение на размер этой очереди: если множество задействованных вершин становится слишком большим, вершины с наименьшей вероятностью получения лучшего пути удаляются из множества. Единственным ограничением является

требование поддерживать упорядоченность вершин в множестве, что, в свою очередь, накладывает ограничения на используемые виды контейнеров для хранения вершин.

Iterative deepening (итеративное погружение). Алгоритм Iterative deepening [5] позволяет производить более точные вычисления. Идея его заключается в том, чтобы проверить следующие несколько шагов алгоритма. Если при этом не достигается улучшение результата, то делается предположение, что текущее решение уже является достаточно оптимальным, и вряд ли удастся его улучшить при проведении аналогичных операций в том направлении. В алгоритме IDA* производится регулировка глубины по уровню f: если значение f слишком велико, узел не будет рассматриваться. Таким образом, при первом проходе будет проверено малое количество узлов. При последующих проходах количество посещаемых узлов будет увеличиваться. В случае обнаружения улучшения пути следует продолжить исследование, в противном случае вычисления можно прерывать.

Dynamic weighting (использование переменных весов). В случае использования изменяющихся весов [4] мы предполагаем, что на первых шагах алгоритма требуется как можно быстрее достичь области, содержащей конечную точку пути. В финальной стадии поиска более важным является достижение конкретной точки. Предлагается следующая модификация функции веса:

$$f(p) = g(p) + w(p)\, h(p).$$

При приближении к конечной точке вес уменьшается. Это снижает значимость эвристики и увеличивает относительную важность фактической стоимости пути.

Bidirectional search (двунаправленный поиск). В данной модификации [5] алгоритма A* поиск пути ведется одновременно из двух вершин навстречу друг другу. Когда эти два пути встречаются, алгоритм завершает свою работу. Основной недостаток — отсутствие универсальности: алгоритм не будет работать на всех типах карт. В то же время он работает не с одним большим деревом поиска, а с двумя маленькими, что позволяет уменьшить количество используемой памяти.

Theta*. Одна из основных проблем алгоритма A* (как и многих других) заключается в том, что получаемые с его помощью пути не выглядят реалистичными. Эта проблема решается в алгоритме Theta*

[2, 4]. Ключевой момент, отличающий эти два алгоритма, заключается в том, что Theta* позволяет в качестве «предка» для каждой вершины выбрать любую вершину, в отличие от A*, где «предком» может быть только ближайший видимый сосед.

Алгоритмы постобработки путей. В общем случае алгоритмы постобработки разделяются на две группы: алгоритмы, требующие дополнительных обращений к ландшафту, и алгоритмы, основывающиеся на уже имеющейся информации.

К первой группе можно отнести алгоритм Theta*, использующий функцию Line of Sight для определения факта наличия непроходимых областей на прямой, которая соединяет две ключевые точки пути.

В алгоритмах второй группы происходит удаление лишних точек, лежащих на одной прямой. Такая постобработка может потребоваться для исследования результатов, полученных в ходе работы алгоритма A*. Этот алгоритм может использоваться также для «грубой» обработки полученного пути: обычно выбирается тройка последовательных ключевых точек пути и исследуется расстояние между центральной точкой и прямой, соединяющей две крайние точки. Варьируя это расстояние, можно добиться уменьшения общего числа точек результирующего маршрута, однако при этом в некоторых случаях соединительные отрезки между ключевыми точками могут проходить через заблокированные участки. Такая ситуация допустима при работе с географическими картами, особенно в случае болотистой местности или различных предгорий, где точная граница препятствия не является фиксированной.

Сравнение алгоритмов. Многие алгоритмы имеют схожую проблему: получаемые с их помощью пути выглядят нереалистично. Для ее решения требуется либо использовать последующую оптимизацию пути, либо применять алгоритм, в котором уже используются функции, позволяющие получить реалистичную картину.

На рис. 2 приведен пример поиска пути между двумя точками с помощью алгоритмов A* и Theta*. Из рисунка видно, что путь, полученный с помощью Theta*, короче и, кроме того, выглядит более реалистичным. Однако алгоритм Theta* оказывается достаточно тяжеловесным из-за большого количества обращений в ландшафту. Если используемую в нем функцию определения наличия препятствий на прямой между двумя точками применить к результату работы алгоритма A*, получится путь, по реалистичности близкий к результату работы алгоритма Theta*, при этом накладные расходы будут на порядок меньше.

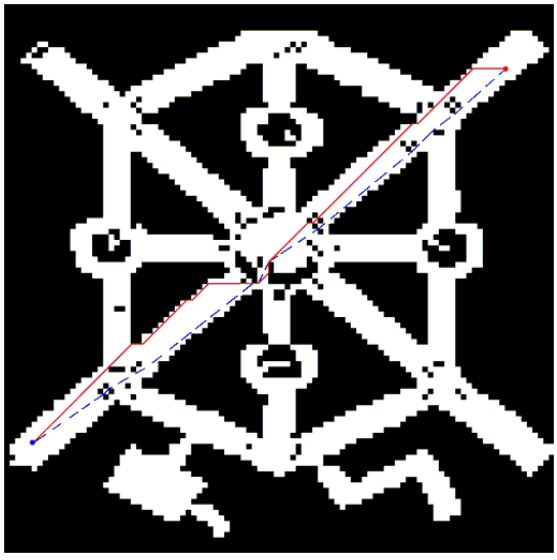


Рис. 2. Сравнение пути A* (красная линия) с путем Theta* (синяя линия)

Заключение. Главная проблема задачи поиска пути заключается в том, что не существует какого-либо универсального алгоритма ее решения. Вместе с тем, на основании проведенных исследований можно сделать вывод, что в случае поиска пути на географических картах необходимо выбирать один из следующих способов решения задачи. Если требуется получить наиболее реалистично выглядящий субоптимальный путь, рекомендуется использовать алгоритм Theta*. Когда затраты на обращение к ландшафту оказываются критичными, рекомендуется использовать следующую комбинацию алгоритмов:

- применяем алгоритм A* для получения маршрута;
- удаляем точки, лежащие на одной прямой;
- для каждой пары из небольшого множества полученных ключевых точек применяем алгоритм проверки наличия пути по прямой. Можно применять этот алгоритм только к соседним отрезкам пути, разбивая их путем внедрения фиктивных точек.

В общем случае необходимо строить систему алгоритмов, имеющих схожие входные и выходные данные, что позволит обмениваться данными на отдельных шагах, комбинируя различные подходы к решению задачи. Помимо этого, немаловажным окажется введение иерархии: объединяя отдельные области ландшафта, можно прокладывать пути сначала между крупными областями, потом, применяя другие алгоритмы, — на отдельных участках, и дальше по аналогии [3].

ЛИТЕРАТУРА

[1] Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. *Алгоритмы. Построение и анализ.* Москва, Вильямс, 2005.

[2] Nash A. *Any-Angle Path Planning.* Dis. … Doctor of Philosophy (Computer Science). University of South California. August 2012.

[3] Botea A., Muller M., Schaeffer J. Near Optimal Hierarchical Path-Finding. *Journal of Game Development*, 2004, vol. 1, issue 1, pp. 7–28.

[4] Daniel K., Nash A., Koenig S., Felner A. Theta*: Any-Angle Path Planning on Grids. *Journal of Artificial Intelligence Research*, 2010, vol. 39, pp. 533–579.

[5] Variants of A*, *Amit Patel's Home Page*. http://theory.stanford.edu/~amitp/GameProgramming/Variations.html (дата обращения 16.04.2013).

Статья поступила в редакцию 24.06.2013 г.

Ссылку на эту статью просим оформлять следующим образом:

Басараб М.А., Домрачева А.Б., Купляков В.М. Алгоритмы решения задачи быстрого поиска пути на географических картах. *Инженерный журнал: наука и инновации*, 2013, вып. № 11. URL: http://engjournal.ru/catalog/it/hidden/1054.html

Басараб Михаил Алексеевич окончил Харьковский авиационный институт им. Н.Е. Жуковского в 1993 г. Д-р физ.-мат. наук, профессор кафедры «Теоретическая информатика и компьютерные технологии» МГТУ им. Н.Э. Баумана. Автор пяти монографий и более 100 научных работ в области прикладной математики, информатики, цифровой обработки сигналов, радиофизики. e-mail: bmic@mail.ru

Домрачева Анна Борисовна окончила МГТУ им. Н.Э. Баумана в 1993 г. Доцент кафедры «Теоретическая информатика и компьютерные технологии» МГТУ им. Н.Э. Баумана. Автор более 30 работ в области цифровой обработки сигналов, математического моделирования, геоинформатики, информационной безопасности. e-mail: annd70@mail.ru

Купляков Виталий Михайлович — студент кафедры «Теоретическая информатика и компьютерные технологии» МГТУ им. Н.Э. Баумана. Область научных интересов: поисковые алгоритмы оптимизации на графах.