К

УДК 004.75:004.021

А. А. Койбаш, Т. В. Завадская, С. В. Кривошеев
Государственное образовательное учреждение высшего профессионального образования
«Донецкий национальный технический университет», г. Донецк
83001, г. Донецк, ул. Артёма, 58

МОДИФИКАЦИЯ АЛГОРИТМА А* ДЛЯ ПРОГНОЗИРОВАНИЯ ТРАЕКТОРИИ ДВИЖЕНИЯ ПОДВИЖНОГО ОБЪЕКТА В РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

A. A. Koibash, T. V. Zavadskaya, S. V. Kryvosheev
State Educational Institution of Higher Education «Donetsk national technical University», Donetsk city
83001, Donetsk, Artema str., 58

MODIFICATION ALGORITHM A* FOR PREDICTION OF MOBILE OBJECT MOTION TRAJECTORY AT DISTRIBUTED SYSTEMS

О. А. Койбаш, Т. В. Завадська, С. В. Кривошеєв
Державна освітня установа вищої професійної освіти
«Донецький національний технічний університет», м. Донецьк
83001, м. Донецьк, вул. Артема, 58

МОДИФІКАЦІЯ АЛГОРИТМУ А * ДЛЯ ПРОГНОЗУВАННЯ ТРАЄКТОРІЇ РУХУ РУХОМОГО ОБ'ЄКТА В РОЗПОДІЛЕНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

В работе рассмотрен алгоритм А* и проведён анализ основных мест падения производительности алгоритма. Для повышения быстродействия поиска кратчайшего маршрута предложено использование сортирующего дерева и хеш-таблиц. Проведено исследование работы алгоритма с использованием этих структур. Результаты тестирования показали значительное увеличение скорости вычислений, которые будут использоваться далее в модуле для распределенной системы.
Ключевые слова: поиск кратчайшего маршрута, алгоритм А*, сортирующее дерево, хеш-таблица.

The paper the algorithm A * is considered and the analysis of the basic places of falling of productivity of algorithm is conducted. To improve the speed of searching for the shortest route, the use of a sorting tree and hash tables was proposed. The study of the algorithm with the use of these structures. The test results showed a significant increase in the speed of the calculations that will be used later in the module for a distributed system.
Keywords: search for the shortest route, A * algorithm, sorting tree, hash table.

В роботі розглянуто алгоритм А * і проведено аналіз основних місць падіння продуктивності алгоритму. Для підвищення швидкодії пошуку найкоротшого маршруту запропоновано використання сортуючого дерева і хеш-таблиць. Проведено дослідження роботи алгоритму з використанням цих структур. Результати тестування показали значне збільшення швидкості обчислень, які будуть використовуватися далі в модулі для розподіленої системи.
Ключові слова: пошук найкоротшого маршруту, алгоритм А *, сортуюче дерево, хеш-таблиця.

Введение

Вычислительная техника на сегодняшний день решает множество задач в самых разных отраслях. Однако существует необходимость подготовки квалифицированного персонала, способного управлять такой техникой. Для повышения эффективности этой задачи можно использовать распределённый симулятор, который позволит проводить тренировку, моделировать различные ситуации, а также давать возможность тщательного анализа ошибок, совершенных в процессе обучения.

Объект может перемещаться в место назначения, поэтому в процессе тренировки нужно знать, каким способом можно оптимально добраться до целевой точки. Следовательно, для работы симулятора необходим модуль прогнозирования траектории движения, который позволит рассчитать кратчайший маршрут в обход всех препятствий. Благодаря этому снизится время передвижения и затраты топлива, а также появится возможность сравнения путей, рассчитанных программной и оператором.

Цель работы. Выявить возможные направления модификации алгоритма A* для снижения времени поиска пути при прогнозировании траектории движения подвижного объекта в распределенных вычислительных системах.

Постановка задачи. Определить пути модификации алгоритма A* и оценить прирост производительности при поиске кратчайшего маршрута в распределенных вычислительных системах с использованием сортирующего дерева и хеш-таблиц.

Использование специальных структур для повышения быстродействия

В работах [1-3] определено, что наилучшим решением является использование алгоритма A*, являющимся эвристическим подходом к алгоритму Дейкстры. Были исследованы многопоточные реализации алгоритма A*: просчет параметров для одного маршрута в несколько потоков и просчет сразу нескольких путей – каждый на отдельном потоке. Лучший результат дал подход, при котором считаются сразу несколько маршрутов. Сравнительная диаграмма показана на рис. 1 [4].

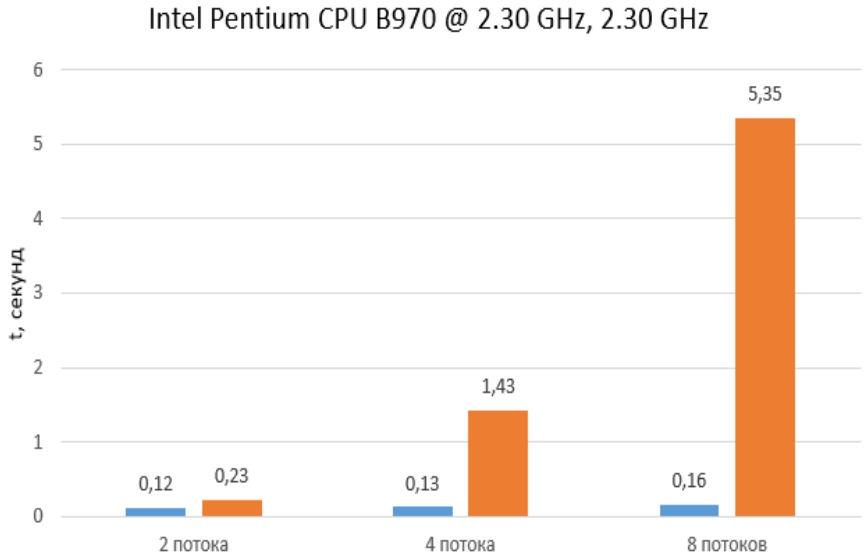

Рисунок 1 – Сравнительная диаграмма разных многопоточных реализаций

Левые столбцы – просчет сразу нескольких путей, правые – просчет одного пути в несколько потоков.

Однако для параллельной реализации необходимо максимально оптимизировать алгоритм A*. Сам алгоритм реализуется следующим образом:

1. Создаются открытый список (вершины, ожидающие рассмотрения) и закрытый список (уже рассмотренные вершины). Стартовая добавляется в закрытый.

2. Для каждой соседней вершины рассчитываются параметры F, G и H, где H – эвристическое приближение, G – путь в текущую клетку из стартовой, а F = G + H.

3. В открытом списке выбирается вершина с наименьшим F. Если она – финиш, то работа алгоритма завершается. Иначе продолжается поиск: вершина помечается текущей, удаляется из открытого списка и добавляется в закрытый.

4. Для каждой соседней с текущей производятся следующие действия:

a. Если соседняя клетка в закрытом списке или является препятствием, пропускается.

b. Если соседней клетки нет в закрытом списке, для неё рассчитываются параметры F, G и H, и она добавляется в открытый список. Текущая клетка становится для неё родительской.

c. Если соседняя клетка в открытом списке, сравнивается прежний параметр G с новым рассчитанным. Это показывает, не дешевле ли путь через данную клетку. Если новый параметр G меньше, то для этой клетки в открытом списке обновляются параметры F и G.

5. Открытый список пуст и финиш не найден, то пути к финишу нет. Для ускорения работы алгоритма вместо содержания закрытого списка в программе можно просто помечать клетки на карте закрытыми. Это существенно упростит поиск закрытых вершин – достаточно обратиться по координатам к нужной клетке и проверить её состояние. Однако при этом в алгоритме остаются следующие слабые места:

1) Поиск в открытом списке вершины с минимальным F (поиск минимума в массиве) – пункт 3.

2) Поиск вершины в открытом списке по совпадению координат (прямой доступ к произвольному элементу массива) – пункт 4 c.

Эти участки алгоритма выполняются циклически, что имеет сильное влияние на быстродействие. На картах 50×50 и 100×100 падение производительности незначительное, так как открытый список в таких случаях не содержит достаточно большое количество элементов. В случае больших карт (например, 2048×2048) время вычисления маршрута возрастает во много раз. Это связано с размером открытого списка, который может достигать многих тысяч элементов. Соответственно, поскольку поиск и прямой доступ также будут выполняться гораздо большее количество раз из-за размеров карты, необходима оптимизация этих участков программы.

Поиск минимального элемента в массиве (открытом списке в данном случае) при размере N элементов оценивается в O(N). Сложность является линейной и для поиска необходим просмотр всего массива. Многократное выполнение этого участка в ходе работы алгоритма A* сильно замедлит время вычисления маршрута.

Для ускорения поиска минимума можно использовать сортирующее дерево, называемое двоичной кучей. Для такого дерева выполняется следующее условие: приоритет каждой вершины больше приоритетов её потомков. Пример такого дерева показан на рис. 2.

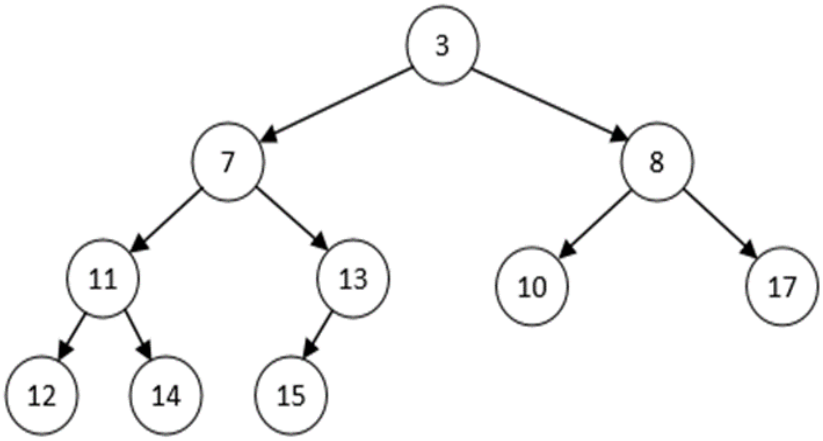

Рисунок 2 – Сортирующее дерево с минимальным приоритетом

При добавлении нового элемента происходит его циклическое сравнение с родительским. В случае большего приоритета новый меняется с родительским, поднимаясь вверх. Так происходит до тех пор, пока не будет соблюдено основное свойство сортирующего дерева.

Дерево имеет логарифмическую высоту и сложность вставки составляет O(log(N)). Поскольку самым приоритетным элементом всегда является корневой элемент дерева, нахождение минимального элемента составляет всего 1 операцию. Согласно алгоритму A* минимальный элемент должен удаляться из списка, то есть корневой элемент необходимо изымать. После такой операции вызывается процедура восстановления свойства двоичной кучи, сложность которой также оценивается как O(log(N)).

Таким образом, операции вставки и удаления минимального элемента двоичной кучи являются достаточно быстрыми. Данную структуру необходимо использовать, однако в программном представлении её удобнее хранить не связанным списком, а в виде одномерного массива. Это необходимо для того, чтобы избежать затратных операций выделения и высвобождения памяти. Тогда корнем дерева будет 0-й элемент, а дочерние элементы вычисляются по формулам: 2*i + 1 (левый) и 2*i + 2 (правый). Пример такого массива показан на рис. 3.

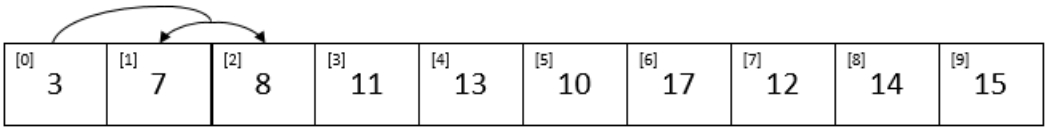

Рисунок 3 – Пример сортирующего дерева на основе массива

В алгоритме A* открытый список можно построить с приоритетом по полю F каждой вершины. Таким образом, сложность вставки каждого соседнего элемента будет составлять O(log(N)) в худшем случае, сложность удаления элемента с последующим восстановлением свойств кучи также составит O(log(N)). Такие операции гораздо быстрее, чем полный просмотр массива. Повышение производительности оказывается намного существеннее при больших размерах открытого списка – когда поиск пути выполняется на больших картах.

Второе падение производительности происходит из-за поиска конкретного элемента в списке. Оценка временной сложности поиска в худшем случае составляет O(N). Необходимо также учесть, что элементы будут добавляться не в стандартном

порядке, как в обычный список, а в отсортированном по полю F–в двоичную кучу. Поэтому для организации прямого доступа по координатам к элементам открытого списка можно модифицировать двоичную кучу при помощи соответствующих структур.

Элемент находится по совпадению координат (x; y). Однако для облегчения прямого доступа координаты можно представить хеш-значениями. В результате определены два способа оптимизации: использование красно-черного дерева или хеш-таблиц. Каждый из этих методов имеет свои сильные стороны, а также разные хеш-функции.

Красно-черное дерево является видом двоичного дерева поиска, для которого выполняется то же условие: каждая вершина больше левого потомка и меньше правого. Однако обычное дерево в худшем случае может разрастись в одну из сторон, что сделает процедуру поиска линейной. Поэтому красно-черное дерево самобалансирующееся – все узлы окрашены в красные или черные цвета. Это обеспечит сложность поиска и вставки в O(log(N)) [5]. Пример красно-черного дерева изображен на рис. 4.

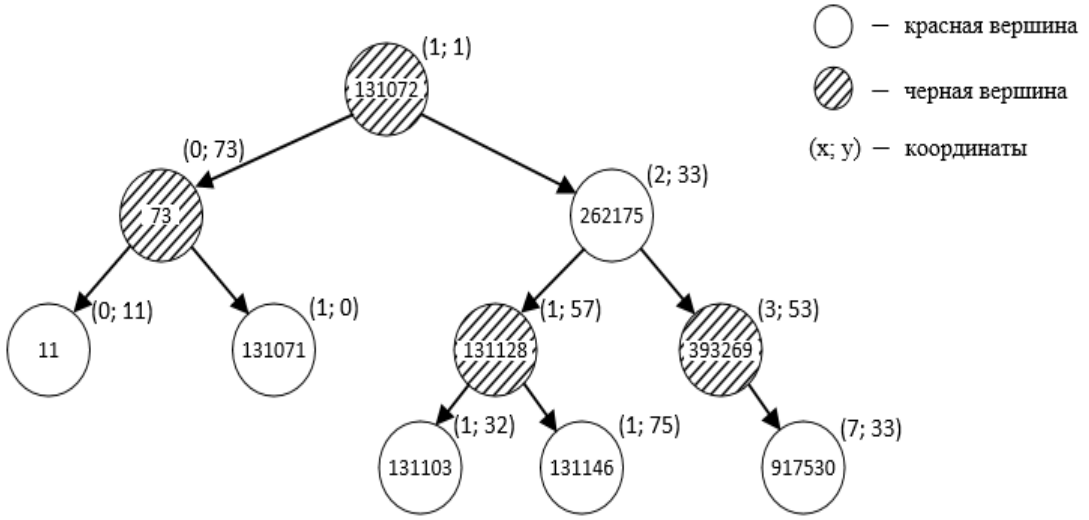

Рисунок 4 – Пример красно-черного дерева

Значения узлов заданы хеш-кодом. Координаты могут хешироваться функцией вида 131 071 * x + y, которая при x∈ [0, 2047], y∈ [0, 2047] не даёт коллизий (совпадений хеш-значений при разных координатах). Сами узлы могут содержать ссылку на элемент в двоичной куче. Таким образом, при необходимости доступа к элементу по координатам будет вычислено хеш-значение и начнется обход дерева. После нахождения нужного узла по ссылке можно получить доступ к элементу двоичной кучи.

Другим вариантом является использование хеш-таблиц. Это специальная структура данных для хранения пар (ключ, значение). В среднем все три операции (вставка, поиск и удаление) выполняются за время O(1)[5].Такая таблица может содержать некоторый массив H, тогда хеш-значение i = hash(key) является индексом массива H. То есть объект хранится в ячейке H[i] и для прямого доступа к нему достаточно использовать хеш-функцию. Однако при таком подходе могут возникать коллизии, которые минимизируются выбором подходящей хеш-функции и разрешаются различными методами.

Поскольку карта 2048×2048 содержит максимум 4 194 304 вершины, нужен баланс между допустимым количеством коллизий и затратами памяти. В качестве хеш-

функции можно взять формулу k*x⊕y, где k – степень двойки. Такая функция обеспечит хорошее распределение [6].

Коллизии можно разрешать методом цепочек. При этом в каждой ячейке будет храниться ссылка на вершину сортирующего дерева и ссылка на следующий элемент таблицы. При 32-разрядной адресации каждая ссылка занимает в памяти 4 байта. Тогда весь массив займёт 2*4*(k*2048) = 16384*k байт. Пример таблицы изображен на рис. 5.

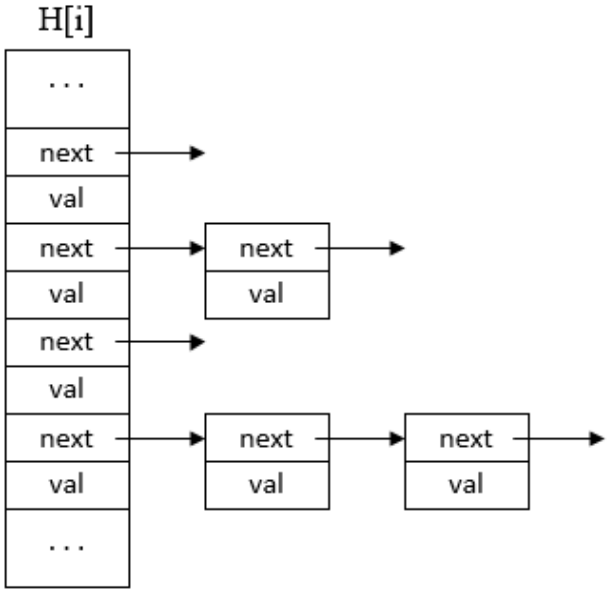

Рисунок 5 – Пример хеш-таблицы

Для выбора параметра k необходимо учесть несколько факторов. При больших значениях k уменьшается вероятность появления коллизий, однако возрастает потребление памяти и времени для выделения этой памяти. При малых k ресурсов нужно меньше, однако повышается вероятность коллизий.

При k = 64 количество совпадений в худшем случае составляет 31 элемент на одну ячейку таблицы. Поскольку всего в таблице 131 072 элементов, она занимает в памяти 1 048 576 байт = 1 МБ. Однако тесты показали, что максимальный уровень коллизий не превышает двух вершин на одну ячейку при поиске, направленность которого примерно совпадает с распределением коллизий. Это связано с удалением вершин из открытого списка в ходе работы алгоритма A*. При других направлениях поиска количество коллизий снижается или исчезает.

Методы оптимизации алгоритма A* были протестированы на процессоре Intel Pentium CPUG2020 @ 2.90GHz 2.90GHz. Было произведено 3 тестовых поиска: без оптимизаций, с использованием сортирующего дерева, а также с использованием сортирующего дерева и хеш-таблиц. Поиск пути производился из точки (0; 0) в точку (1023; 2047). Результаты тестов показаны на рис. 6.

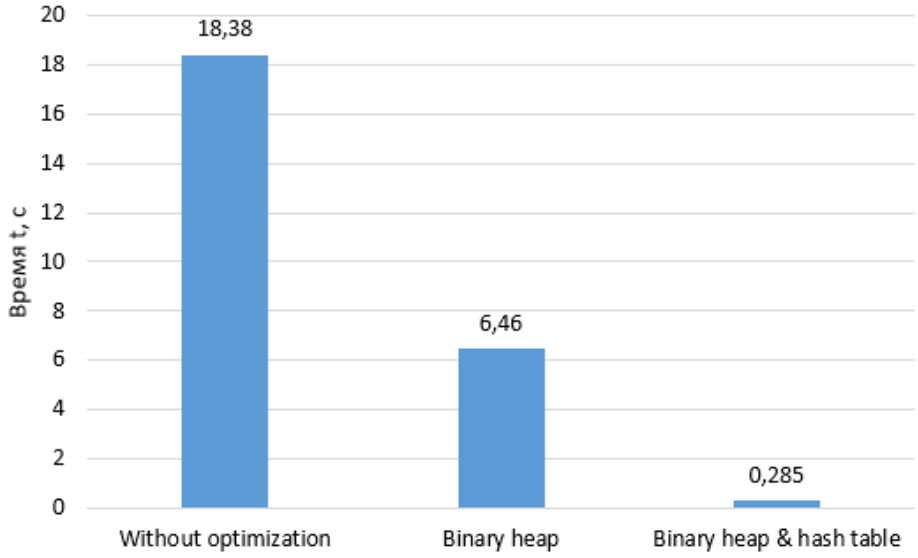

Рисунок 6 – Диаграмма времени поиска пути

Как видно из диаграммы, время выполнения с помощью оптимизирующих структур сильно уменьшилось, что даёт очень сильный прирост производительности. Следовательно, использование этих алгоритмов допустимо при поиске кратчайшего маршрута в распределенных вычислительных системах.

Выводы

Таким образом, многопоточный маршрут с оптимизирующими структурами может дать колоссальный прирост производительности. Для своевременного прогнозирования траектории движения можно вычислять путь изо всех точек, куда может попасть подвижный объект через определённый интервал времени. Поскольку таких точек может быть много, для параллельных вычислений целесообразно использовать архитектуру NVidiaCUDA, что позволит вычислять одновременно десятки маршрутов.

Список литературы

1. Койбаш А. А. Прогнозирование траектории движения подвижного объекта распределенного симулятора тяжелой инженерной техники [Текст] / А. А. Койбаш, С. В. Кривошеев // Информатика, управляющие системы, математическое и компьютерное моделирование (ИУСМКМ – 2016): материалы VII междунар. науч.-техн. конф., Донецк, 2016. / редкол. А. Ю. Харитонов и др. – Донецк : ДонНТУ, 2016. – С. 343–346.
2. Койбаш А. А. Подсистема прогнозирования траектории движения подвижного объекта распределенного симулятора тяжелой инженерной техники [Текст] / А. А. Койбаш, С. В. Кривошеев // Программная инженерия: методы и технологии разработки информационно-вычислительных систем (ПИИВС – 2016): сборник научных трудов I научно-практической конференции (студенческая секция), Донецк, 2016. – Донецк : ДонНТУ, 2016. – С. 262–265.
3. Кривошеев С. В. Пути снижения времени прогнозирования траектории движения подвижного объекта распределенного симулятора тяжелой инженерной техники [Текст] / С. В. Кривошеев // Современные тенденции развития и перспективы внедрения инновационных технологий в машиностроении, образовании и экономике: материалы IV междунар. науч.-практ. конф., Азов 2017. / редкол. С. В. Жуков и др. [Текст] / Кривошеев С.В., Койбаш А.А. – Азов : Технологический институт (филиал) ДГТУ, 2017. С. 51–54.
4. Койбаш А. А. Эффективность использования многоядерных систем для прогнозирования траектории движения подвижного объекта распределенного симулятора тяжелой инженерной техники [Текст] /

А. А. Койбаш, А. Г. Кравченко // Информатика, управляющие системы, математическое и компьютерное моделирование (ИУСМКМ – 2017) : материалы VIII междунар. науч.-техн. конф., Донецк, 2017. / редкол. Ю. К. Орлов и др. – Донецк : ДонНТУ, 2017. – С. 622–625.

5. Томас Х. Кормен. Алгоритмы: построение и анализ [Текст] / Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн; 3-е изд.; Пер. с англ. Красиков И.В. – М. : «Вильямс», 2013. – 1328 с.
6. Койбаш А. А. Прогнозирование траектории движения подвижного объекта распределенного симулятора тяжелой инженерной техники. [Текст] / А. А. Койбаш, Т. В. Завадская, С. В. Кривошеев // Информатика, управляющие системы, математическое и компьютерное моделирование (ИУСМКМ – 2018) : материалы IXмеждунар. науч.-техн. конф., Донецк, 2018. / редкол. Р. В. Мальчева и др. – Донецк : ДонНТУ, 2018. – С. 187–191.

References

1. Koibash A.A., Kryvosheev S.V. Prognozirovanie traektorii dvizheniya podvizhnogo ob"ekta raspredelennogo simulyatora tyazheloj inzhenernoj tekhniki. [Prediction of the motion trajectory of a moving object of a distributed simulator of heavy engineering equipment] V kn.: *Informatika, upravlyayushchie sistemy, matematicheskoe i komp'yuternoe modelirovanie* (IUSMKM – 2016): materialy VII mezhdunar. nauch.-tekhn. konf., Doneck, 2016. / redkol. A. YU. Haritonov i dr. [Computer science, control systems, mathematical and computer modeling] Doneck: DonNTU, 2016. pp. 343–346.
2. Koibash A.A., Kryvosheev S.V. Podsistema prognozirovaniya traektorii dvizheniya podvizhnogo ob"ekta raspredelennogo simulyatora tyazheloj inzhenernoj tekhniki. [Subsystem for predicting the motion trajectory of a moving object of a distributed heavy engineering simulator] V kn: *Programmnaya inzheneriya: metody i tekhnologii razrabotki informacionno-vychislitel'nyh sistem* (PIIVS – 2016): sbornik nauchnyh trudov I nauchno-prakticheskoj konferencii (studencheskaya sekciya), Doneck, 2016. [Software Engineering: Methods and Technologies for Developing Information-Computing systems] / Doneck: DonNTU, 2016. pp. 262–265.
3. Kryvosheev S.V., Koibash A.A. Puti snizheniya vremeni prognozirovaniya traektorii dvizheniya podvizhnogo ob"ekta raspredelennogo simulyatora tyazheloj inzhenernoj tekhniki [Ways to reduce the time to predict the motion trajectory of a moving object of a distributed heavy engineering simulator]. V kn.: *Sovremennye tendencii razvitiya i perspektivy vnedreniya innovacionnyh tekhnologij v mashinostroenii, obrazovanii i ehkonomike*: materialy IV mezhdunar. nauch.-prakt. konf., Azov 2017. / redkol. S. V. ZHukov i dr. [Modern development trends and perspectives of implementation innovative technologies in mechanical engineering, education and economy] Azov: Tekhnologicheskij institut (filial) DGTU [Technological Institute DGTU], 2017. pp. 51–54.
4. Koibash A.A., Kravchenko A.G. Effektivnost' ispol'zovaniya mnogoyadernyh sistem dlya prognozirovaniya traektorii dvizheniya podvizhnogo ob"ekta raspredelennogo simulyatora tyazheloj inzhenernoj tekhniki [Efficiency of using multi-core systems for predicting the trajectory of movement of a moving object of a distributed simulator of heavy engineering equipment]. V kn.: *Informatika, upravlyayushchie sistemy, matematicheskoe i komp'yuternoe modelirovanie* (IUSMKM – 2017): materialy VIII mezhdunar. nauch.-tekhn. konf., Doneck, 2017.[Computer science, control systems, mathematical and computer modeling] / redkol. YU. K. Orlov i dr. Doneck: DonNTU, 2017. pp. 622–625.
5. Tomas H. Kormen. *Algoritmy: postroenie i analiz* [Algorithms: construction and analysis]/ Tomas H. Kormen, CHarl'z I. Lejzerson, Ronal'd L. Rivest, Klifford SHtajn; 3-e izd.; Per. s angl. Krasikov I.V., M., Vil'yams, 2013, 1328 s.
6. Koibash A.A., Zavadskaya T.V., Kryvosheev S.V. *Prognozirovanie traektorii dvizheniya podvizhnogo ob"ekta raspredelennogo simulyatora tyazheloj inzhenernoj tekhniki*.[Prediction of the motion trajectory of a moving object of a distributed simulator of heavy engineering equipment.] V kn.:Informatika, upravlyayushchie sistemy, matematicheskoe i komp'yuternoe modelirovanie (IUSMKM – 2018): materialy IXmezhdunar. nauch.-tekhn. konf., [Computer science, control systems, mathematical and computer modeling] Doneck, 2018. / redkol. R. V. Mal'cheva i dr. Doneck: DonNTU, 2018. pp. 187–191.

RESUME

A. A. Koibash, T. V. Zavadskaya, S.V. Kryvosheev

Modification Algorithm A for Prediction of Mobile Object Motion Trajectory at Distributed Systems*

Background: The paper the algorithm A * is considered and the analysis of the basic places of falling of productivity of algorithm is conducted. To improve the speed of searching for the shortest route, the use of a sorting tree and hash tables was proposed. The study of the algorithm with the use of these structures. The test results showed a significant increase in the speed of the calculations that will be used later in the module for a distributed system.

Materials and methods: development environment Microsoft Visual Studio and programming languages C ++ and C #.

Results: A modified algorithm for finding the shortest route is proposed.

Conclusion: A multi-threaded route with optimizing structures can give a huge performance boost. For timely prediction of the trajectory of motion, it is possible to calculate the path from all points where a moving object can fall at a certain time interval. Since there may be many such points, for parallel computing, it is advisable to use the NVidiaCUDA architecture, which will allow you to simultaneously calculate dozens of routes.

РЕЗЮМЕ

А. А. Койбаш, Т. В. Завадская, С.В. Кривошеев

Модификация алгоритма A для прогнозирования траектории движения подвижного объекта в распределенных вычислительных системах*

История вопроса. В работе рассмотрен алгоритм A* и проведён анализ основных мест падения производительности алгоритма. Для повышения быстродействия поиска кратчайшего маршрута предложено использование сортирующего дерева и хеш-таблиц. Проведено исследование работы алгоритма с использованием этих структур. Результаты тестирования показали значительное увеличение скорости вычислений, которые будут использоваться далее в модуле для распределенной системы.

Материалы и методы. В работе используется среда разработки Microsoft Visual Studio. А также языки программирования C++ и C#.

Результаты. Предложен модифицированный алгоритм поиска кратчайшего маршрута.

Заключение. Многопоточный маршрут с оптимизирующими структурами может дать колоссальный прирост производительности. Для своевременного прогнозирования траектории движения можно вычислять путь изо всех точек, куда может попасть подвижный объект через определённый интервал времени. Поскольку таких точек может быть много, для параллельных вычислений целесообразно использовать архитектуру NVidiaCUDA, что позволит вычислять одновременно десятки маршрутов.

Статья поступила в редакцию 05.09.2018.