**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ ФГБОУ ВО «БРЯНСКИЙ ГОСУДАРСТВЕНННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

## Кафедра «Информатика и программное обеспечение»

КУРСОВАЯ РАБОТА

**ПО ДИСЦИПЛИНЕ «СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА»** СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ НЕИНФОРМИРОВАННОГО ПОИСКА И ЭВРИСТИЧЕСКОГО ПОИСКА

**Всего листов 23**

Студент О-20-МОА-тп-Б Зач. Кн. №

Горбатков Д. Р.

« » 15 декабря 2023 г. Преподаватель

Бабурин А. Н.

« » 15 декабря 2023 г.

**БРЯНСК 2023**

**СОДЕРЖАНИЕ**

[Введение 3](#_bookmark0)

1. [Аналитическая часть 4](#_bookmark1)
   1. [Поиск в ширину 4](#_bookmark2)
   2. [Двунаправленный поиск 5](#_bookmark3)
   3. [Эвристический (информативный) поиск 5](#_bookmark4)
   4. [Сравнительный анализ методов поиска 7](#_bookmark5)
2. [Конструкторская часть 9](#_bookmark6)
   1. [Описание задачи 9](#_bookmark7)
   2. [Описание разработанной программы 9](#_bookmark8)
   3. [Описание разработанных алгоритмов поиска 10](#_bookmark9)
      1. [Поиск в ширину 12](#_bookmark10)
      2. [Двунаправленный поиск 13](#_bookmark11)
      3. [Эвристический поиск 14](#_bookmark12)

[2.3.5 Сравнение эвристических функций информативного поиска 16](#_bookmark13)

1. [Тестирование алгоритмов поиска 17](#_bookmark14)

[Заключение 22](#_bookmark15)

[Список литературы 23](#_bookmark16)

# ВВЕДЕНИЕ

В данной работе будет проведен сравнительный анализ алгоритмов неинформативного и эвристического поиска с целью определение более эффективного подхода для различных ситуаций. Данные для анализа были получены при использовании лабораторных работ, выполненных ранее. Чтобы получить объективные выводы о эффективности алгоритмов, введем ключевые параметры для их оценки:

1. Количество итераций цикла поиска до нахождения решения;
2. Максимальное количество узлов, хранимое единовременно в памяти во время поиска решения;
3. Эффективный коэффициент ветвления (обозначающийся как b\*).

Сравнение будет проводиться на задачах разной сложности, где сложность оценивается глубиной решения задачи.

# АНАЛИТИЧЕСКАЯ ЧАСТЬ

## Поиск в ширину

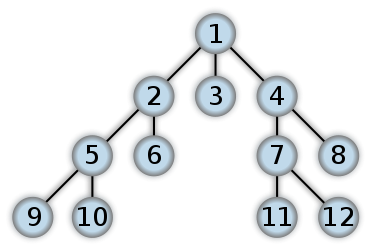
Поиск в ширину — один из простейших алгоритмов обхода графа, являющийся основой для многих важных алгоритмов для работы с графами. Он используется для исследования всех вершин графа путем поуровневого обхода каждой из них.

Поиск в ширину работает следующим образом: алгоритм начинает в заранее выбранной вершине и сначала «посещает» и отмечает всех соседей этой вершины (рис. 1). Потом он переходит к соседям посещенных вершин, затем — дальше по тому же принципу. Так продолжается до тех пор, пока все вершины не будут исследованы. Из-за характера распространения, похожего на волну, алгоритм еще называют волновым.

Для реализации алгоритма поиска в ширину принято использовать очередь: для каждой вершины, посещенной алгоритмом, все достижимые из нее соседи помещаются в очередь, из которой будет взята следующая вершина для дальнейшего обхода.

Данный алгоритм наиболее часто применяется для поиска пути между двумя вершинами графа, так как поиск в ширину гарантирует нахождение кратчайшего пути в графе, а его эффективность не зависит от структуры самого

графа.



*Рис. 1: Порядок рассмотрения вершин в поиске в ширину*

## Двунаправленный поиск

Двунаправленный поиск пути в ширину — усложнённый алгоритм поиска в ширину, идея которого заключается в формировании процесса поиска от начальной (прямой поиск) и от конечной вершины (обратный поиск) графа. При этом, внутренняя структура алгоритма не меняется.

Двунаправленный поиск действует по такому принципу: алгоритм начинает свою работу одновременно из начального и конечного состояния в виде прямого поиска и обратного поиска, помечая всех достижимых соседей текущей вершины. В случае, если рассматриваемая вершина уже была помечена другим алгоритмом поиска, работа алгоритма завершается.

Несмотря на более высокую скорость алгоритма двунаправленного поиска, у него есть существенный недостаток: нужна дополнительная память для хранения дерева поиска для того, чтобы можно было выполнить проверку принадлежности проверяемой вершины другому дереву.

## Эвристический (информативный) поиск

Информативный поиск — стратегия поиска решений в пространстве состояний, в которой используются знания, относящиеся к конкретной задаче. Пространство состояний представляет собой граф, где узлы - состояния задачи, а ребра - операции, переводящие из одного состояния в другое. Информированные методы обычно обеспечивают более эффективный поиск по сравнению с неинформированными методами.

Информация о конкретной задаче формулируется в виде эвристической функции. Эвристическая функция на каждом шаге перебора оценивает альтернативы на основании дополнительной информации с целью принятия решения о том, в каком направлении следует продолжать перебор.

В качестве примера эвристического поиска можно рассмотреть алгоритм A\*. A\* на каждом шагу просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Как и все информированные алгоритмы поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. От жадного алгоритма, который тоже является алгоритмом поиска по первому лучшему совпадению, его отличает то, что при выборе вершины он учитывает, помимо прочего, весь пройденный до неё путь. Составляющая g(x) — это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме.

В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение f(x), после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа

— множеством частных решений, — которое размещается в очереди с приоритетом. Приоритет пути определяется по значению f(x) = g(x) + h(x). Алгоритм продолжает свою работу до тех пор, пока значение f(x) целевой вершины не окажется меньшим, чем любое значение в очереди, либо пока всё дерево не будет просмотрено. Из множества решений выбирается решение с наименьшей стоимостью.

## Сравнительный анализ методов поиска

Сравнительный анализ перечисленных выше алгоритмов представлен в таблице 1.

***Таблица 1***

**Сравнение алгоритмов поиска**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Поиск в глубину | Поиск в ширину | Эвристический поиск |
| Тип | Неинформативный | Неинформативный | Информативный |
| Оптимальнос ть | Гарантировано | Гарантировано | Гарантировано |
| Время | O(n+m) | O(n+m) | Зависит от эвристики и структуры пространства  состояний |
| Эффективнос ть | Неэффективен в случае больших деревьев или графов с большим  числом узлов | Эффективен для поиска в больших деревьях с неограниченной  памятью | Эффективен для задач с определенной целью и хорошей эвристикой |
| Хранение узлов | Очередь | Очередь | Очередь с приоритетом |
| Поиск пути | Гарантирует нахождение оптимального пути | Гарантирует нахождение оптимального пути | Гарантирует поиск оптимального пути, при условии допустимой эвристики |

Исходя из сравнительной таблицы, можно сделать следующие выводы:

1. Поиск в ширину подходит для простых задач, но может быть неэффективен для больших деревьев и графов;
2. Двунаправленный поиск является модификацией поиска в ширину, так как увеличивает его эффективность в больших деревьях и графах, однако затрачивает больше памяти;
3. A\* является информированным (эвристическим) алгоритмом, который обеспечивает гарантированное нахождение оптимального пути при условии использования допустимой эвристики. A\* также эффективен для задач с определенной целью и хорошей эвристикой.

Таким образом, выбор конкретного алгоритма зависит от особенностей конкретной задачи.

# КОНСТРУКТОРСКАЯ ЧАСТЬ

## Описание задачи

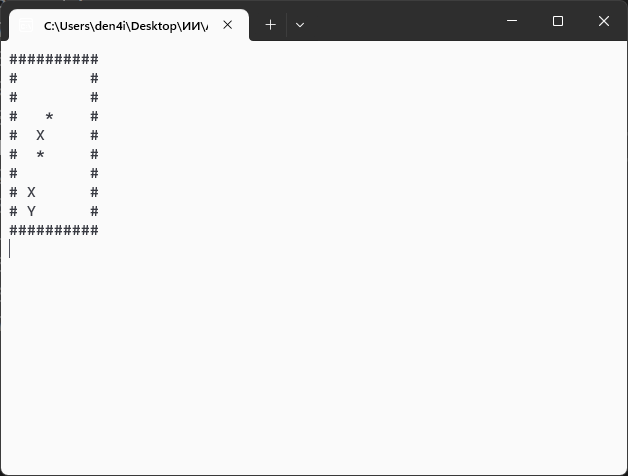
Необходимо разработать программу для нахождения решение в игре

«Сокобан».

Игра представляет из себя поле, размером до 10\*10 клеток. Имеется 3 вида клеток: свободная, стена, метка. Метка – это клетка, на которой нужно разместить ящик. На поле должно иметься N меток, при этом на свободных клетках должно быть расположено N ящиков. На одной из свободных клеток должен находиться грузчик (игрок). Его задача – расположить ящики на метках. Ящики можно только толкать, тянуть нельзя. Ходить грузчик может только вверх, вниз, влево и вправо, и только по свободным клеткам или меткам. Можно толкать ящик, только если он не упирается в стену. Толкать два ящика друг за другом нельзя. Цель – разместить все ящики по меткам.

## Описание разработанной программы

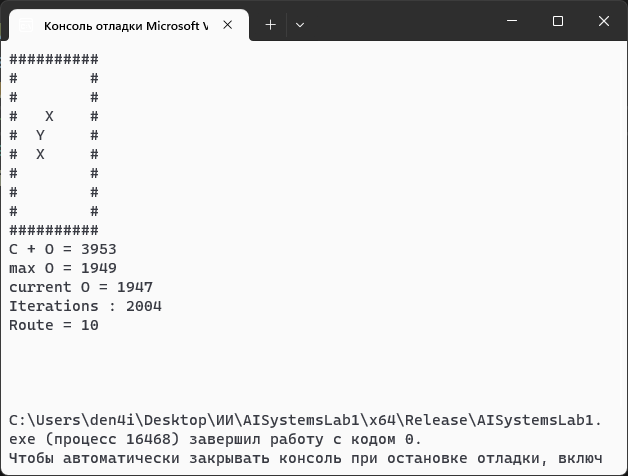
Была разработана программа с графическим интерфейсом, представленным на рис. 2.



*Рис. 2: Графический интерфейс игры*

Для задания игровой карты необходимо изменить текстовый файл, идущий вместе с игрой, либо добавить новый и вписать его в коде программы.

При нахождении решения программа покажет путь от начального до конечного состояния, а так же выведет всю необходимую для анализа информацию (рис. 3).



*Рис. 3: Результат поиска решения*

## Описание разработанных алгоритмов поиска

В ходе выполнения работы были разработаны 3 алгоритма поиска: 2 неинформативных и информативный (эвристический) с 2 различными эвристическими функциями.

Был разработан базовый класс Sokoban (см. листинг 1), который содержит в себе общие методы по типу проверки текущего состояния на совпадение с конечным, переход между состояниями и т.п. Все разработанные алгоритмы поиска расширяли базовый класс.

***Листинг 1***

***Базовый класс игры Sokoban***

class Sokoban

{

public:

Sokoban() = default;

~Sokoban() = default;

void loadFieldFromFile(const string& filename); void makeMoveWithoutPrint(Move move);

void startGame();

vector<vector<int>> getField() const; void makeField(const string&);

string getCurrentFieldState(); void setField(vector<vector<int>>);

void setTargets(vector<vector<int>>); void printCondition();

bool isGamePassed(); private:

void setPlayerPosition();

bool isMoveAvailable(Move move);

string readFromFile(const string& filename); vector<vector<int>> makeTargets(const string); void resizeField(int& cols, int& rows);

void moveLeft(); void moveRight(); void moveTop(); void moveBottom(); bool checkLeft(); bool checkRight(); bool checkTop(); bool checkBottom();

private:

vector<vector<int>> field; vector<vector<int>> targets; int numberOfBoxes = 0;

pair <int, int> playerPosition;

const int TARGET = 5, EMPTY = 0, BOX = 1, WALL = 2, PLAYER = 3;

};

## Поиск в ширину

Реализация алгоритма поиска в ширину представлена в листинг 2. Для данного алгоритма были разработаны методы раскрытия узла, проверки на наличие рассматриваемого состояния в списках O и С.

***Листинг 2***

***Основные методы алгоритма поиска в ширину***

int BroadFirstSearch::findSolution(Sokoban& sokoban) { while (not openStates->empty()) {

addAllMoves(sokoban); if (isPassed) break; }

return closedStates->size() + maxO;

}

void BroadFirstSearch::addAllMoves(Sokoban& sokoban) { string parent = openStates->front().parent;

string currentState = openStates->front().current; sokoban.makeField(currentState);

if (sokoban.isGamePassed()) isPassed = true; else {

addMove(sokoban, currentState, Move::LEFT); addMove(sokoban, currentState, Move::RIGHT); addMove(sokoban, currentState, Move::TOP); addMove(sokoban, currentState, Move::BOTTOM);

}

closedStates->insert({ currentState, parent }); openStates->pop\_front();

iterations++;

}

void BroadFirstSearch::addMove(Sokoban& sokoban, const string& currentState, const Move move) {

sokoban.makeField(currentState); sokoban.makeMoveWithoutPrint(move);

string newState = sokoban.getCurrentFieldState(); if (newState != currentState) {

if (not isInLists(newState)) {

openStates->push\_back({ newState, currentState }); maxO = max(maxO, openStates->size()); checkedStates->insert(newState);

} } }

## Двунаправленный поиск

Для реализации данного алгоритма класс поиска решений был дополнен обратным поиском AntiSokoban.

Пример методов алгоритма представлены в листинг 3.

***Листинг 2***

***Примеры методов алгоритма двунаправленного поиска***

void BidirectionalSearch::addMove(Sokoban& sokoban, const string& currentState, const

Move move) {

sokoban.makeField(currentState); sokoban.makeMoveWithoutPrint(move);

string newState = sokoban.getCurrentFieldState(); if (newState != currentState) {

if (not isInSokobanLists(newState)) { sokobanOpenStates.push\_back({ newState, currentState });

maxO=max(maxO,antisokobanOpenStates.size()+sokobanOpenStates.size()); sokobanCheckedStates.insert({ newState, currentState });

}

}

}

void

BidirectionalSearch::addMove(AntiSokoban&

antisokoban,

const

string&

currentState, const Move move) {

antisokoban.makeField(currentState); antisokoban.makeMoveWithoutPrint(move);

string newState = antisokoban.getCurrentFieldState(); if (newState != currentState) {

if (not isInAntiSokobanLists(newState)) { antisokobanOpenStates.push\_back({ newState, currentState }); maxO=max(maxO,antisokobanOpenStates.size()+sokobanOpenStates.size()); antisokobanCheckedStates.insert({ newState, currentState });

}

}

}

## Эвристический поиск

Эвристический поиск потребовал изменений структуры данных для хранения открытых состояний в вычисления приоритетов для каждого состояния. Модифицированный метод вставки состояния и метод вычисления приоритета приведены в листинг 3.

***Листинг 3***

***Пример метода алгоритма эвристического поиска***

void HeuristicSearch::addMove(Sokoban& sokoban, const string& currentState, int depth, const Move move) {

sokoban.makeField(currentState); sokoban.makeMoveWithoutPrint(move);

string newState = sokoban.getCurrentFieldState(); if (newState != currentState) {

if (not isInLists(newState)) {

double heuristics = getHeuristics(newState, sokoban, ++depth); openStates.push({ -heuristics, newState, currentState, depth }); maxO = max(maxO, openStates.size()); checkedStates.insert(newState);

}

}

}

double HeuristicSearch::getHeuristics(const string& current, Sokoban& sokoban, int depth) {

int heuristics;

heuristics = (isManhattan ? sokoban.getManDist() : sokoban.getUpManDist()); return heuristics + depth;

}

Для вычисления эвристической функции разработано 2 метода: первая из них рассчитывает расстояние от ящиков до целевых состояний; (листинг 4), другая же также учитывает расстояние от грузчика до ящиков (листинг 5).

***Листинг 4***

***Вычисление первой эвристической функции***

vector<vector<double>> distances(n, vector<double>(n)); for (int i = 0; i < n; i++) {

Point fst, snd;

for (int j = 0; j < n; j++) {

fst = { boxes[i].first, boxes[i].second }; snd = { targets[j].first, targets[j].second }; distances[i][j] = fst.calculateDist(&snd);

}

}

vector<int> currPerm(n);

for (int i = 0; i < n; i++) currPerm[i] = i; do {

double currentSum = 0.0;

for (int i = 0; i < n; i++) currentSum += distances[i][currPerm[i]]; if (currentSum < sum) sum = currentSum;

} while (next\_permutation(begin(currPerm), end(currPerm)));

return sum;

***Листинг 5***

***Вычисление второй эвристической функции***

vector<vector<double>> distances(n, vector<double>(n)); for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

fst = { boxes[i].first, boxes[i].second }; snd = { targets[j].first, targets[j].second }; distances[i][j] = fst.calculateDist(&snd);

}

}

vector<int> currPerm(n);

for (int i = 0; i < n; i++) currPerm[i] = i; do {

double currentSum = 0.0;

for (int i = 0; i < n; i++) currentSum += distances[i][currPerm[i]]; if (currentSum < sum) sum = currentSum;

} while (next\_permutation(begin(currPerm), end(currPerm)));

return sum;

## Сравнение эвристических функций информативного поиска

Обе эвристики для вычисления своего значения используют расстояния от ящиков до целевых точек, которое всегда будет меньше, чем количество ходов, требующееся для их перемещения в конечное состояние, поэтому обе эвристики будем считать допустимыми.

Эвристика, учитывающая расстояние от грузчика до ящиков является более продуманной, так как даже при движении без изменения положения ящика приоритет состояния будет изменяться, из-за чего количество ходов «в холостую» будет сведено к минимуму.

# ТЕСТИРОВАНИЕ АЛГОРИТМОВ ПОИСКА

На основе разработанной программы были проведены тесты для глубины решения d = 6-15.

Для каждой глубины были рассмотрены 10 различных состояний, полученных с помощью генератора состояний с заданной глубиной и взяты средние результаты, приведенные в таблицах 2, 3, 4.

***Таблица 2***

**Результаты стоимости поиска**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Стоимость поиска N | | | |
| d | BFS | BDS | A\*(h1) | A\*(h2) |
| 6 | 1769 | 865 | 419 | 260 |
| 7 | 3430 | 1119 | 690 | 446 |
| 8 | 5937 | 1789 | 1449 | 804 |
| 9 | 8937 | 2276 | 1819 | 718 |
| 10 | 17462 | 2780 | 2826 | 1268 |
| 11 | 45802 | 5480 | 5286 | 1368 |
| 12 | 68502 | 8281 | 9065 | 1692 |
| 13 | 107172 | 12236 | 7589 | 5017 |
| 14 | 149817 | 16120 | 9128 | 3045 |
| 15 | 264206 | 17035 | 22454 | 3911 |

***Таблица 3***

**Результаты количества итераций**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Количество итераций | | | |
| d | BFS | BDS | A\*(h1) | A\*(h2) |
| 6 | 959 | 386 | 180 | 88 |
| 7 | 1942 | 530 | 345 | 160 |
| 8 | 3407 | 872 | 718 | 290 |
| 9 | 5114 | 1136 | 907 | 236 |
| 10 | 10390 | 1424 | 1525 | 485 |
| 11 | 27274 | 2870 | 2850 | 485 |
| 12 | 36496 | 4330 | 4805 | 628 |
| 13 | 65232 | 6832 | 3873 | 1755 |
| 14 | 92163 | 9278 | 5433 | 1165 |
| 15 | 162182 | 9164 | 12164 | 1416 |

***Таблица 4***

**Результаты эффективного коэффициента ветвления**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Эффективный коэффициент ветвления b\* | | | |
| d | BFS | BDS | A\*(h1) | A\*(h2) |
| 6 | 3,27 | 2,86 | 2,52 | 2,3 |
| 7 | 3,02 | 2,54 | 2,35 | 2,2 |
| 8 | 2,8 | 2,4 | 2,31 | 2,13 |
| 9 | 2,6 | 2,2 | 2,15 | 1,91 |
| 10 | 2,52 | 2,07 | 2,07 | 1,9 |
| 11 | 2,53 | 2,06 | 2,05 | 1,79 |
| 12 | 2,42 | 2 | 2,01 | 1,73 |
| 13 | 2,33 | 1,95 | 1,88 | 1,81 |
| 14 | 2,24 | 1,89 | 1,81 | 1,66 |
| 15 | 2,21 | 1,82 | 1,85 | 1,63 |

Из данных таблиц можно сделать вывод, что эвристические алгоритмы поиска гораздо превосходят неинформированные алгоритмы.

Экспериментальным путём выяснено, что алгоритм A\*(h2) лучше, чем A\*(h1), что и ожидалось на этапе разработки, но всё же оба эти алгоритма лучше BDS (двунаправленный поиск в ширину).

Для наглядности можно привести следующие графики (рис 6, 7, 8):

300000

250000

200000

150000

100000

BFS BDS

A\*(h1)

A\*(h2)

50000

0

1 2 3 4 5 6 7 8 9 10

*Рис. 4: График стоимости пути*

180000

160000

140000

120000

100000

80000

60000

BFS BDS

A\*(h1)

A\*(h2)

40000

20000

0

1 2 3 4 5 6 7 8 9 10

*Рис. 5: График количества итераций*

3,5

3

2,5

2

1,5

1

BFS BDS

A\*(h1)

A\*(h2)

0,5

0

1 2 3 4 5 6 7 8 9 10

*Рис. 6: График эффективного коэффициента ветвления b\**

Из графиков видны колоссальный рост стоимости поиска и количества итераций для стандартного метода поиска в ширину, небольшой рост для его модификации и стабильность информированных методов. Также для информированных алгоритмов поиска эффективный коэффициент ветвления b\* стремится к 1.

# ЗАКЛЮЧЕНИЕ

В заключении работы можно сделать следующие вывод: аналитическое сравнение алгоритмов неинформированного и эвристического поиска позволило выявить превосходство эвристического подхода в решении поставленной задачи.

Исследование основывалось на данных из предыдущих лабораторных исследований, анализ был обоснован и объективен.

При сравнении алгоритмов они оценивались по трем ключевым параметрам: количество итераций цикла поиска, максимальное количество узлов, хранимых в памяти, и эффективный коэффициент ветвления. Эти параметры позволили точно оценить эффективность каждого алгоритма в различных ситуациях.

Сравнение было проведено на задачах разной сложности, где сложность определялась глубиной оптимального решения. Это позволило проверить применимость каждого алгоритма в разных сценариях.

В результате исследования напрашивается вывод, что эвристический подход является широко применимым в решении задач. Этот вывод подкреплен полученными экспериментальными данными и аналитическими вычислениями. Таким образом, результаты работы говорят в пользу использования эвристического поиска в решении задач поиска кратчайшего пути и являются

основанием для дальнейших исследований в этой области.

# СПИСОК ЛИТЕРАТУРЫ

* + - 1. Расселл, С., & Норвиг, П. (2016). Искусственный интеллект: Современный подход. Pearson Education.
      2. Нильссон, Н. Й. (1998). Искусственный интеллект: Новый синтез. Morgan Kaufmann.
      3. Лугер, Г. Ф., & Стаблфилд, У. А. (2004). Искусственный интеллект: Структуры и стратегии для решения сложных проблем. Pearson Education.
      4. Перл, Дж. (1984). Эвристика: Интеллектуальные стратегии поиска решений компьютерных проблем. Addison-Wesley.
      5. Харт, П. Е., Нильссон, Н. Й., & Рафаэль, Б. (1968). Формальная основа эвристического определения путей минимальной стоимости. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107.
      6. Ньюэлл, А., & Саймон, Х. А. (1972). Человеческое решение проблем. Prentice-Hall.