Севастопольский государственный университет
Институт информационных технологий

"МЕТОДЫ И СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА" (МиСИИ)

Бондарев Владимир Николаевич

Поиск в ширину

```
Procedure Breadth_First_Search;  {BFS}
Begin
        Поместить начальную вершину в список OPEN;
        CLOSED:=‘пустой список’;
        While OPEN<>‘пустой список’ Do
        Begin
                n:=first(OPEN);
                If n=‘целевая вершина’ Then Выход(УСПЕХ);
                Переместить n из списка OPEN в CLOSED;
                Раскрыть вершину n и поместить все ее
                дочерние вершины [, которых нет в списках
                CLOSED и OPEN,] в конец списка OPEN,
                связав с каждой дочерней вершиной
                указатель на n;
        End;
        Выход(НЕУДАЧА);
End.
```

В ЛР!

OPEN = util.Queue()

Сравнение методов слепого поиска

Критерий	BFS	DFS	DFID	Двунаправленный
Время	b^s	b^m	b^s	$b^{m/2}$
Память	b^s	bm	bs	$b^{m/2}$
Оптимальность	Да	Нет	Да	Да
Полнота	Да	Нет	Да	Да

Двунаправленный поиск возможен на неориентированных графах и предполагает движение из начального состояния в целевое и из целевого в исходное. Поиск прекращается, когда фронты поиска пересекутся (критерий: текущая вершина принадлежит другому дереву поиска).

Лекция 5

ИНФОРМИРОВАННЫЙ (ЭВРИСТИЧЕСКИЙ) ПОИСК РЕШЕНИЙ ЗАДАЧ В ПРОСТРАНСТВЕ СОСТОЯНИЙ

В.Бондарев

Общая характеристика методов эвристического поиска

Основная идея таких методов состоит в использовании *дополнительной информации* для ускорения процесса поиска. Эта дополнительная информация формируется на основе эмпирического опыта, догадок и интуиции исследователя, т.е. **эвристик.** Использование эвристик позволяет сократить количество просматриваемых вариантов при поиске решения задачи, что ведет к более быстрому достижению цели.

В алгоритмах эвристического поиска список **OPEN** упорядочивается по возрастанию некоторой ***оценочной функции***, состоящей из двух составляющих, одна из которых называется эвристической и характеризует близость текущей и целевой вершин. Чем меньше значение эвристической составляющей оценочной функции, тем ближе рассматриваемая вершина к целевой вершине.

Общая характеристика методов эвристического поиска

В зависимости от способа формирования оценочной функции выделяют следующие алгоритмы эвристического поиска:

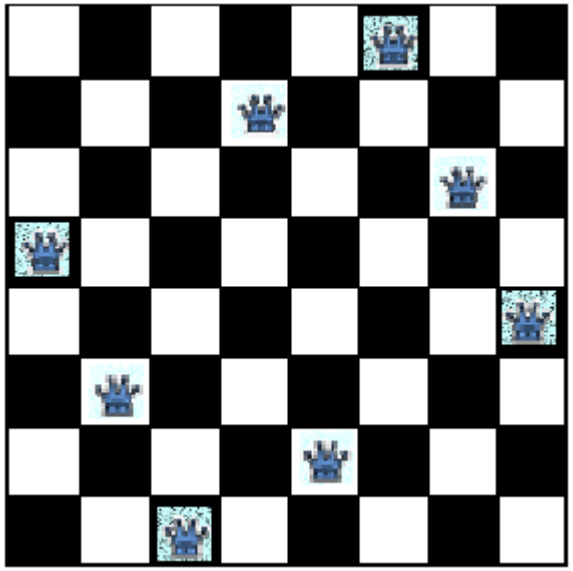
- “жадный” (greedy) ***локальный*** алгоритм (например, алгоритм “подъема на гору”);

- “жадный” алгоритм поиска по первому наилучшему совпадению (Best-First Search – BFS);

- A-алгоритм .

Локальные эвристические алгоритмы

При решении некоторых задач **путь к цели не представляет интереса.** Например, в задаче с восемью ферзями важна лишь окончательная конфигурация ферзей, а не порядок, в котором они были поставлены на доску.



Другие задачи: разработка планов размещений (БИС), составление расписания, автоматическое программирование, оптимизация сети связи.

Алгоритмы локального поиска **действуют с учетом единственного текущего состояния** (а не многочисленных путей) и рассматривают только переход в состояние, соседнее по отношению к текущему. Информация и пройденных путях часто не сохраняется.

Жадный локальный алгоритм поиска

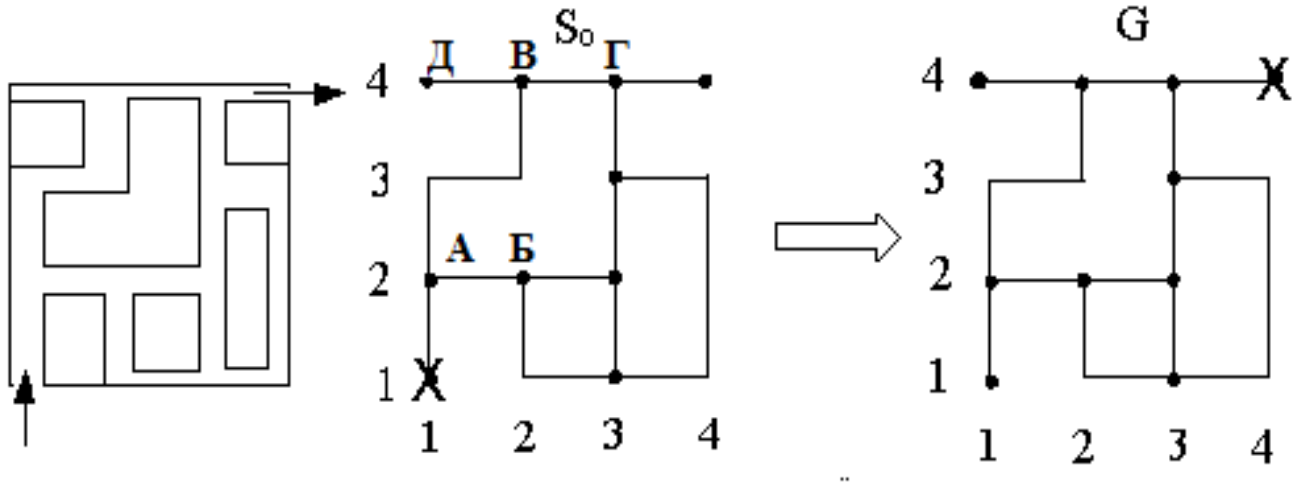
Алгоритм осуществляет целенаправленный поиск в направлении наибольшего убывания ***эвристической оценочной функции*** $\hat{h}(n)$. Данная функция обеспечивает оценку (прогноз) стоимости кратчайшего пути от текущей вершины ***n*** до ближайшей целевой вершины, т.е. *является мерой стоимости оставшегося пути.* Чем меньше значение этой функции, тем перспективнее путь, на котором находится вершина ***n***.

Подобный алгоритм используется при поиске экстремумов функции. Поиск экстремума осуществляют в направлении наибольшего возрастания (убывания) градиента. Поиск максимума функции в этом случае напоминает восхождение к вершине по наиболее крутому маршруту. Поэтому рассматриваемый алгоритм и называют алгоритмом “подъема на гору” (**hill – climbing**).

Жадный локальный алгоритм поиска

Procedure GreedySearch;
Begin
n:=‘начальная вершина’;
While n<>‘целевая вершина’ Do
Begin
Раскрыть вершину n и для всех дочерних вершин n_i вычислить оценки $\hat{h}(n_i)$;
Выбрать дочернюю вершину n_i с минимальным значением $\hat{h}(n_i)$;
If $\hat{h}(n_i) \geq \hat{h}(n)$ Then Выход(НЕУДАЧА);
n:=n_i;
End;
Выход(УСПЕХ);
End.

Жадный локальный алгоритм. Пример поиска пути в лабиринте



Эвристическую оценку $\hat{h}(n_i)$ будем вычислять по формуле

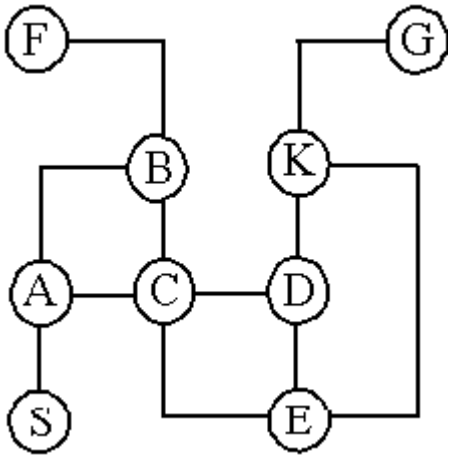
$$\hat{h}(n_i) = |x_g - x_n| + |y_g - y_n|,$$

где x_g, y_g – координаты целевой вершины; x_n, y_n – координаты текущей вершины. Данная оценка соответствует **манхэттенскому расстоянию** от вершины n_i до целевой вершины.

h(а)=|4-1|+|4-2|=5; h(б)=|4-2|+|4-2|=4; h(в)=|4-2|+|4-4|=2;

h(д)=|4-1|+|4-4|=3; h(г)=|4-3|+|4-4|=1. **Решение: А-В-Г-G**

Жадный локальный алгоритм. Пример поиска пути в лабиринте



Для лабиринта на рисунке:

h(a)=|4-1|+|4-2|=5; h(в)=|4-2|+|4-3|=3;

h(c)=|4-2|+|4-2|=4; h(f)=|4-1|+|4-4|=3;

Решение: Неудача, т.к. h(b)=h(f)=3

Таким образом, хотя данный алгоритм обеспечивает ускорение поиска, он **не гарантирует** достижение целевой вершины.

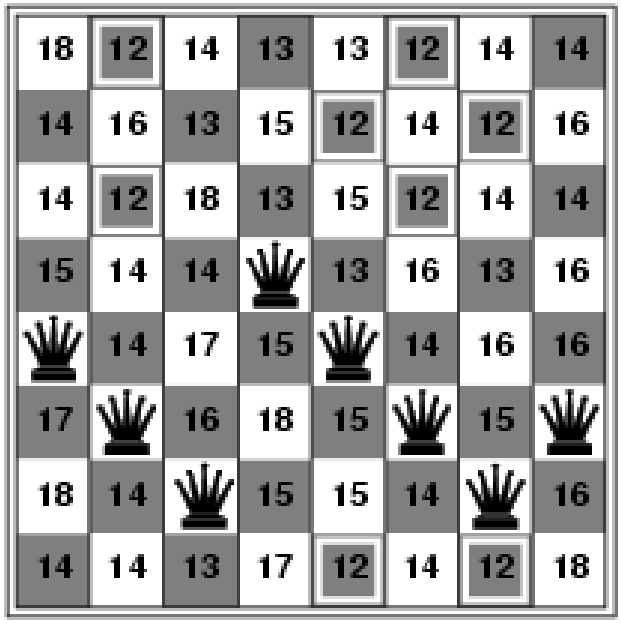
Преждевременное завершение алгоритма происходит в следующих случаях: если в процессе поиска встречаются **локальные минимумы** оценочной функции ; если для группы соседних вершин эвристические оценки равны между собой (проблема **“плато”**); если оценочная функция имеет **хребет.**

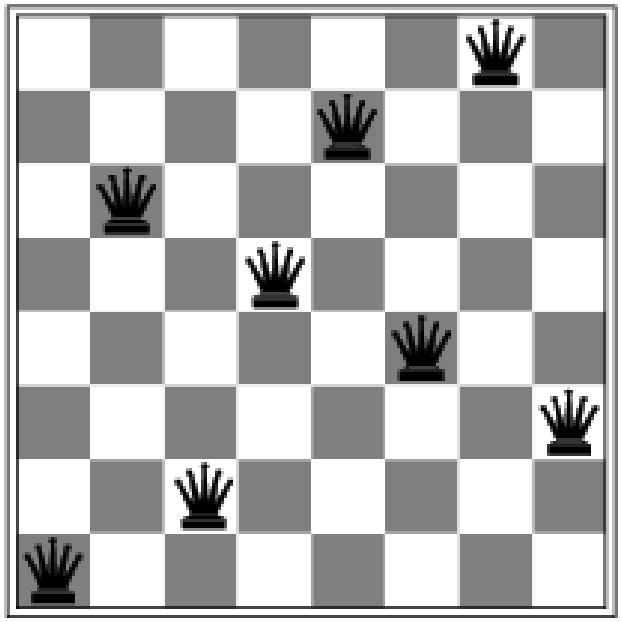
Задача с восемью ферзями

Состояние: 8 ферзей на доске, по одному в каждом столбце.

Оператор: Перемещение отдельного ферзя в другую клетку этого же столбца (каждое состояние имеет 8*7=56 приемников).

Эвристическая функция: кол-во пар ферзей, которые атакуют друг друга (глобальный минимум этой функции равен нулю и это соответствует целевому состоянию).





h=17(лучшие ходы *h*=12)

h=1 (лок. минимум)

Задача с восемью ферзями

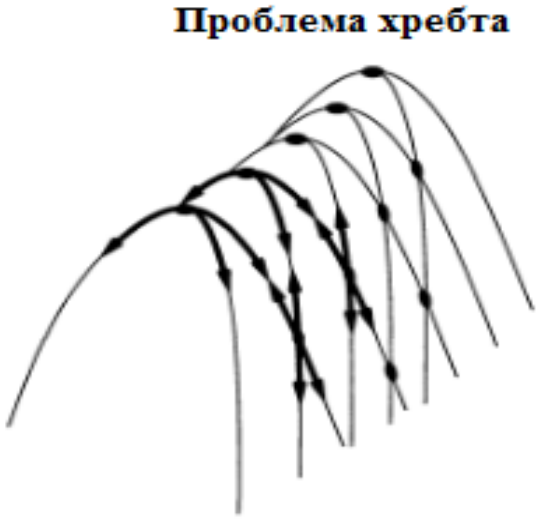
Если приемников более одного, то предусматривается случайный выбор в множестве наилучших приёмников.

Рассматриваемый поиск называют **жадным локальным поиском**, поскольку в процессе его выполнения происходит захват самого хорошего состояния без предварительных рассуждений о том, куда следует двигаться затем.

Во время такого поиска зачастую происходит очень **быстрое продвижение в направлении к решению.** Например, из состояния h=17 достаточно сделать 5 ходов, чтобы получить состояние h=1, которое очень близко к одному из решений (но каждый последующий ход будет характеризоваться большим значением оценочной функции).

Поиск с подъемом на гору заходит в тупик из-за следующих проблем (для задачи с 8 ферзями 86% случаев):

Иллюстрация проблем алгоритма подъема на "гору"



Если для задачи с 8 ферзями разрешить 100 случайных ходов для равных значений оценочной функции, то алгоритм будет находить решение в 94% случаев.

Разработано много вариантов поиска с восхождением на гору. Для задачи с 8 ферзями эффективным является – **восхождение к вершине с перезапуском случайным образом** (начальное состояние формируется случайным образом, требуется примерно 7 перезапусков).

Жадный поиск по первому наилучшему совпадению

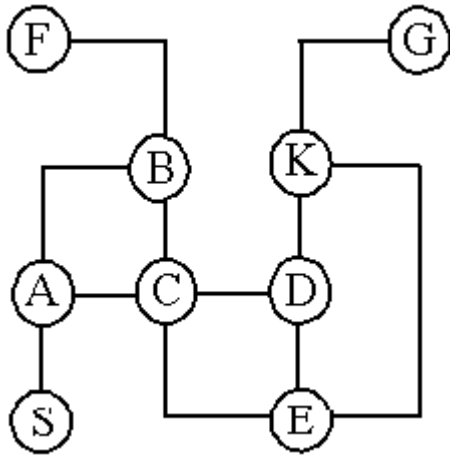
Этот алгоритм формально похож на алгоритм равных цен (**UCS**), но в качестве оценочной функции используется не стоимость уже пройденного пути *g(n)*, а **эвристическая оценка** *h(n)*, т.е. учитываются только предстоящие затраты. Для этого список открытых вершин сортируется в порядке возрастания значений *h(n)*. Наилучшая вершина для продолжения пути будет находиться на первом месте в списке **OPEN**. Поэтому данный алгоритм называют также алгоритмом поиска по первому наилучшему совпадению (**Best-First**).

Благодаря тому, что здесь на каждом этапе выполняется не локальное сравнение вершин, полученных при раскрытии текущей вершины, а **глобальное сравнение** всех вершин-кандидатов, находящихся в списке **OPEN**, удается успешно решать проблему локальных экстремумов оценочной функции.

Жадный поиск GBFS

Procedure Greedy_Best_First_Search;
Begin
Поместить начальную вершину в OPEN;
CLOSED:=‘пустой список’;
While OPEN<>‘пустой список’ Do Begin
n:=first(OPEN);
If n=‘целевая вершина’ Then Выход(УСПЕХ);
Переместить вершину n из списка OPEN в список CLOSED;
Раскрыть вершину n, для каждой дочерней вершины вычислить $\hat{h}(n_i)$ **;**
Поместить дочерние вершины, которых нет в списках OPEN и CLOSED, в список OPEN, связав с каждой дочерней вершиной указатель на вершину n;
Упорядочить список OPEN по возрастанию значений $\hat{h}(n_i)$
End;
Выход(НЕУДАЧА);
End.

Пример применения алгоритма GBFS



Для лабиринта на рисунке состояния списка OPEN будут следующими:

(S(6))→(A(5)) → (B(3) C(4)) → (F(3) C(4)) → (C(4)) → (D(3) E(4)) → → (K(2) E(4)) → (G(0) E(4)).

Таким образом, алгоритм позволяет успешно справляется с проблемой локальных минимумов. Однако при этом снижается эффективность самого процесса поиска. Поиск будет выполняться **быстро**, если оценка стоимости кратчайшего пути из вершины *n* в целевую вершину будет как можно ближе к реальной стоимости *h(n)* .

В общем случае **GBFS не является оптимальным и полным. Пространственная и временная сложности - $O(b^m)$,** где максимальная глубина пространства поиска. Хорошая эвристика позволяет значительно снизить сложность.

A-алгоритм

Достоинством алгоритма равных цен является оптимальность.

Достоинством алгоритмов, использующих оценку предстоящих затрат $\hat{h}(n)$, является возможность усечения дерева поиска, что повышает эффективность поиска.

Естественным является стремление комбинировать эти алгоритмы в оценочной функции вида

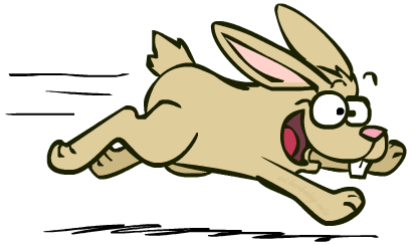
$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n),$$

где $\hat{g}(n)$ – оценка стоимости пути из начальной вершины в вершину n; $\hat{h}(n)$ – оценка стоимости кратчайшего пути из вершины n в целевую вершину.

Алгоритм, базирующийся на использовании оценочной функции $\hat{f}(n)$, называется **A-алгоритмом**. В общем случае оценки $\hat{f}(n)$ меняют свои значения в процессе поиска. Это приводит к тому, что вершины из списка CLOSED могут перемещаться обратно в список OPEN:

A* Поиск

UCS



Greedy

A*

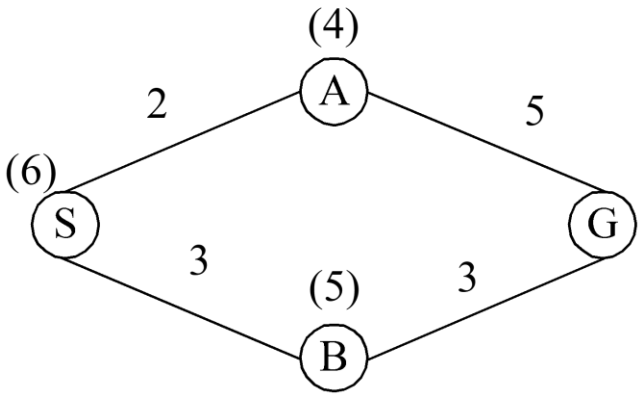
A-алгоритм

1. Поместить начальную вершину *s* в список OPEN: $\hat{f}(s) = \hat{h}(s)$;
2. CLOSED:=‘пустой список’;
3. While OPEN<>‘пустой список’ Do
   n:=first(OPEN);
   If n=‘целевая вершина’ Then Выход(УСПЕХ);
   Переместить n из OPEN в CLOSED;
   Раскрыть n и для всех ее дочерних вершин вычислить оценку $\hat{f}(n, n_i) = \hat{g}(n, n_i) + \hat{h}(n_i)$;
   Поместить дочерние вершины, которых нет в списках CLOSED и OPEN, в список OPEN, связав с каждой вершиной указатель на вершину n;
   Для дочерних вершин , которые уже содержатся в OPEN, сравнить оценки $\hat{f}(n, n_i) < \hat{f}(n_i)$, при выполнении условия связать с вершиной n_i новую оценку $\hat{f}(n, n_i)$ и указатель на вершину n;
   Если вершина содержится в списке CLOSED и $\hat{f}(n, n_i) < \hat{f}(n_i)$, то связать с вершиной n_i новую оценку $\hat{f}(n, n_i)$, переместить её в список OPEN и установить указатель на n;
   Упорядочить список OPEN по возрастанию $\hat{f}(n_i)$;
4. Выход(НЕУДАЧА);

Свойства A-алгоритма

1. Если для всех вершин $\hat{h}(n) = 0$, то **A**-алгоритм будет соответствовать алгоритму равных цен.

2. **A**-алгоритм не даёт гарантии, что найденный путь будет оптимальным. Путь может быть неоптимальным в случае, если оценка затрат на пути из вершины *n* в целевую вершину будет преувеличена (переоценена), т.е. если $\hat{h}(n) > h(n)$. Сказанное можно пояснить с помощью рисунка



При раскрытии вершины S получаем дочерние вершины A и B. Оценки для этих вершин *f(A)*=2+4, *f(B)*=3+5. На следующем шаге раскрывается вершина A, и получается *f(G)*=7+0. Решением будет путь S→A→G, который для данного графа не является оптимальным.

Свойства А-алгоритма

3. **А**-алгоритм обеспечит нахождение оптимального пути ***при поиске по дереву***, если выполняется условие **допустимости (гарантированности)**

$$\hat{h}(n) \quad \leq \quad h(n).$$

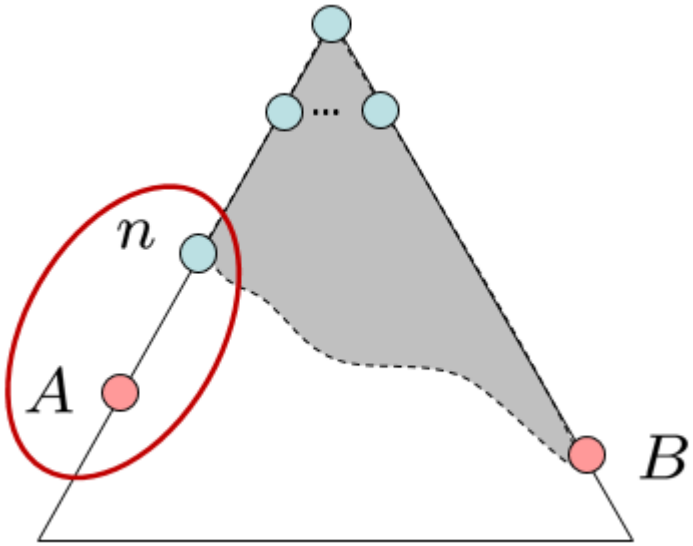
Алгоритм, учитывающий это условие, называют **А*- алгоритмом или гарантирующим алгоритмом.** *А*-алгоритм недооценивает затраты на пути из промежуточной вершины в целевую вершину или оценивает их правильно, но никогда не переоценивает.*

Теорема: Если эвристическая функция при поиске *по дереву* удовлетворяет условию допустимости, то А*-алгоритм обеспечивает нахождение оптимального решения

Свойства A-алгоритма

Доказательство оптимальности A алгоритма:*

Пусть имеются два достижимых целевых состояния: A – оптимальное и B – субоптимальное. Имеется некоторый родительский узел *n* для A, находящийся в списке открытых вершин. Узел B также в OPEN. Утверждается, что узел A будет раскрыт раньше узла B.



Для доказательства оптимальности покажем, что:

1. *n* будет раскрыт ранее B, т.е. f(n)<f(B):
   - а) f(n) меньше или равно f(A);
   - б) f(A) меньше, чем f(B);
   - с) *n* раскрывается раньше B;
2. Все предки A раскрываются раньше B;
3. A раскрывается раньше B;
4. A* является оптимальным.

Свойства A-алгоритма

Доказательство оптимальности A алгоритма :*

$g(A)<g(B)$, т.к. A –оптимально, то затраты для достижения A меньше
$h(A)=h(B)=0$, т.к. для любого узла n, если он целевой $h(n)=0$
а) Покажем, что если n предок A, то $f(n) \le f(A)$:

$$f(A)=g(A)+h(A)=g(A)=g(n)+h(n,A) \ge g(n)+\hat{h}(n)=f(n)$$

б) Покажем, что $f(A) < f(B)$:

$$f(A)=g(A)+h(A)=g(A)<g(B)=g(B)+h(B)=f(B)$$

в) Покажем, что $f(n) < f(B)$. Т.к. $f(A) < f(B)$ и $f(n) \le f(A)$, то

$$(f(n) \le f(A)) \cap (f(A) < f(B)) \rightarrow f(n) < f(B)$$

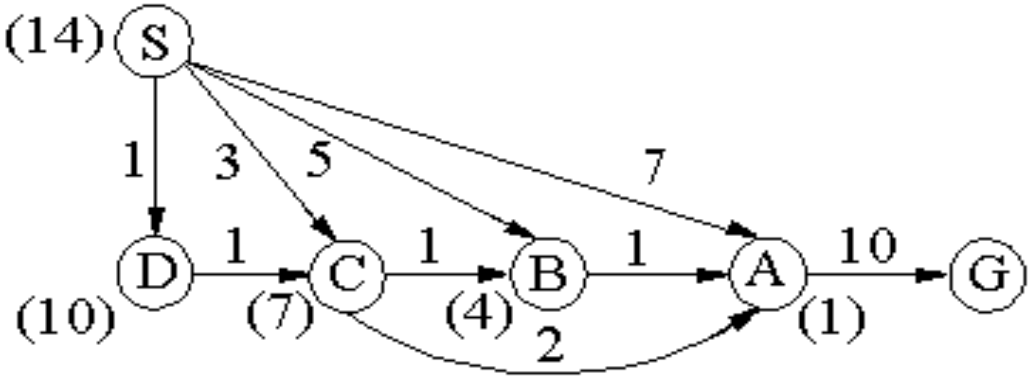
т.е. n будет раскрыта раньше B.
Так как это доказано для произвольного узла n, то все предки A будут раскрыты ранее B.
Следовательно A раскрывается раньше B.
Значит A* алгоритм оптимальный.

Свойства A-алгоритма

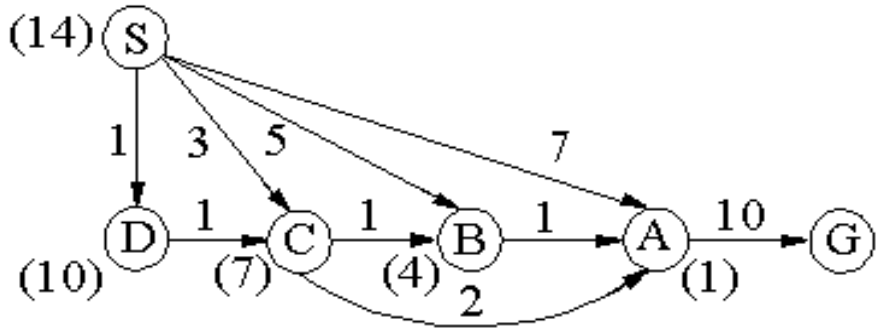
4. Пусть **A1*** и **A2*** – произвольные гарантирующие алгоритмы и на всех вершинах $\hat{h}_1(n) > \hat{h}_2(n)$. Тогда информированность **A1*** выше, чем **A2*.** Говорят, что в этом случае **A1*** имеет большую **эвристическую силу,** чем **A2* (A1* доминирует** над **A2*).** Эвристически более сильный алгоритм **A1*** в большей степени сокращает пространство поиска.

Свойства A-алгоритма

5. Эффективность поиска с помощью **A***-алгоритма может снижаться из-за того, что вершина, находящаяся в списке CLOSED, может попадать обратно в список OPEN и затем повторно раскрываться. Следующий пример демонстрирует случай многократного раскрытия вершин



Свойства A-алгоритма



№	OPEN	CLOSED	Комментарий
1	S(14)	–	
2	A(8) B(9) C(10) D(11)	S(14)	
3	B(9) C(10) D(11) G(17)	**A(8)** S(14)	
4	A(7) C(10) D(11) G(17)	B(9) S(14)	Возврат: A
5	C(10) D(11) G(16)	**A(7)** B(9) S(14)	
6	A(6) B(8) D(11) G(16)	C(10) S(14)	Возврат: A, B
7	B(8) D(11) G(15)	**A(6)** C(10) S(14)	
8	D(11) G(15)	B(8) A(6) C(10) S(14)	
9	C(9) G(15)	D(11) B(8) A(6) S(14)	Возврат: C
10	A(5) B(7) G(15)	C(9) D(11) S(14)	Возврат: A, B
11	B(7) G(14)	**A(5)** C(9) D(11) S(14)	
12	G(14)	B(7) A(5) C(9) D(11) S(14)	

Свойства А-алгоритма

6. Чтобы **A***-алгоритм не раскрывал несколько раз одну и ту же вершину необходимо, чтобы выполнялось ***условие монотонности***

$$\hat{h}(n_i) - \hat{h}(n_j) \le c(n_i, n_j) \ ,$$

где n_i – родительская вершина; n_j – дочерняя вершина; $c(n_i, n_j)$ – стоимость пути между вершинами n_i и n_j. Монотонность предполагает, что не только не переоценивается стоимость $h(n)$ оставшегося пути из n до цели, но и **не переоцениваются стоимости ребер между двумя соседними вершинами**. Условие монотонности поглощает условие гарантированности (допустимости).

Если условие монотонности соблюдается для всех дочерних вершин графа, то можно доказать, что в тот момент, когда раскрывается некоторая вершина n, ***оптимальный путь*** *к ней уже найден.* Следовательно, оценочная функция для данной вершины в дальнейшем не меняет своих значений, и никакие вершины из списка CLOSED в список OPEN не возвращаются

Свойства A-алгоритма

Если соблюдается условие монотонности, то значения оценочной функции *f(n)* вдоль любого пути являются **неубывающими**.

Действительно:

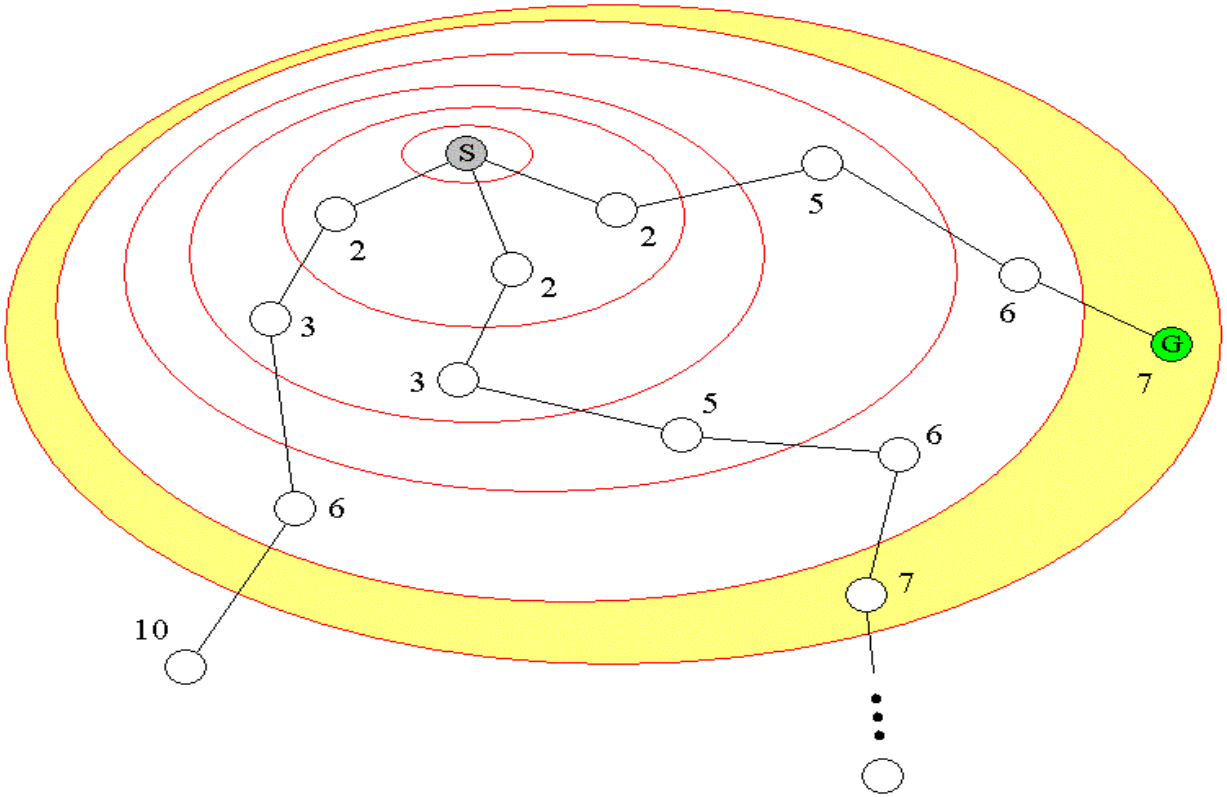
$$f(n_j) = g(n_j) + h(n_j) = g(n_i) + c(n_i, n_j) + h(n_j) \geq g(n_i) + h(n_i) = f(n_i)$$

Т.е. $f(n_j) \geq f(n_i)$

Тот факт, что стоимости вдоль любого пути являются не убывающими, означает что в пространстве состояний могут быть очерчены **контуры равных стоимостей**. Поэтому **A*-** алгоритм проверяет все узлы в контуре меньшей стоимости, прежде чем перейдет к проверке узлов следующего контура.

В **A*-**поиске могут раскрываться некоторые дополнительные узлы, находящиеся на целевом контуре, прежде чем будет выбран целевой узел.

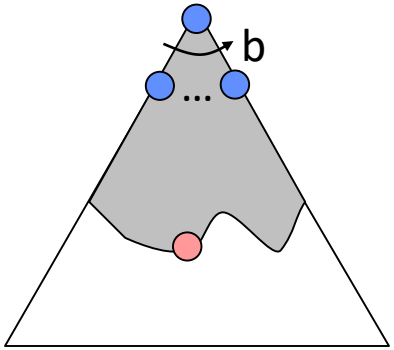
Контуры равных стоимостей



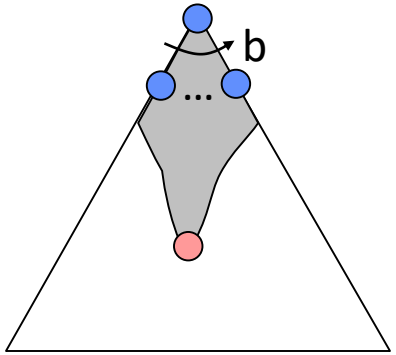
Т.о **A*-**алгоритм (при выполнении условия монотонности) является **полным, оптимальным и оптимально эффективным** (не гарантируется развертывание меньшего кол-ва узлов с помощью иного алгоритма, выполняющего поиск пути от корня и использующего ту же эвристическую информацию).

Свойства A*

Uniform-Cost

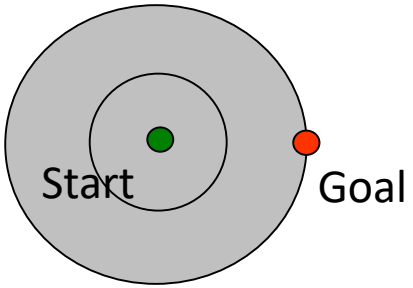


A*

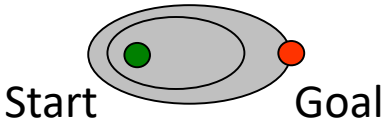


Контуры UCS и A*

- UCS расширяется равномерно во всех «направлениях»



- A* расширяется в основном в сторону цели, но ограничивает свои ставки, чтобы обеспечить оптимальность



Свойства A-алгоритма

Временная сложность **A***-алгоритма по прежнему остается **экспоненциальной**, т.е кол-во раскрываемых узлов в пределах целевого контура пространства состояний все еще зависит экспоненциально от длины решения. По этой причине на практике стремление находить оптимальное решение не всегда оправдано. Иногда вместо этого целесообразно использовать варианты **A**-поиска, позволяющие быстро находить квазиоптимальные решения, или разрабатывать более точные эвристические функции, но не строго допустимые.

Так как при **A***-поиске хранятся все раскрытые узлы, фактические **ресурсы пространства** исчерпаются гораздо раньше, чем временные ресурсы. С этой целью применяют ***A*-алгоритм с итеративным углублением (IDA*).*** Применяемым условием остановки здесь служит **f**-стоимость, а не глубина. Также имеются более современные алгоритмы: ***RBFS и SMA****

RBFS и SMA*

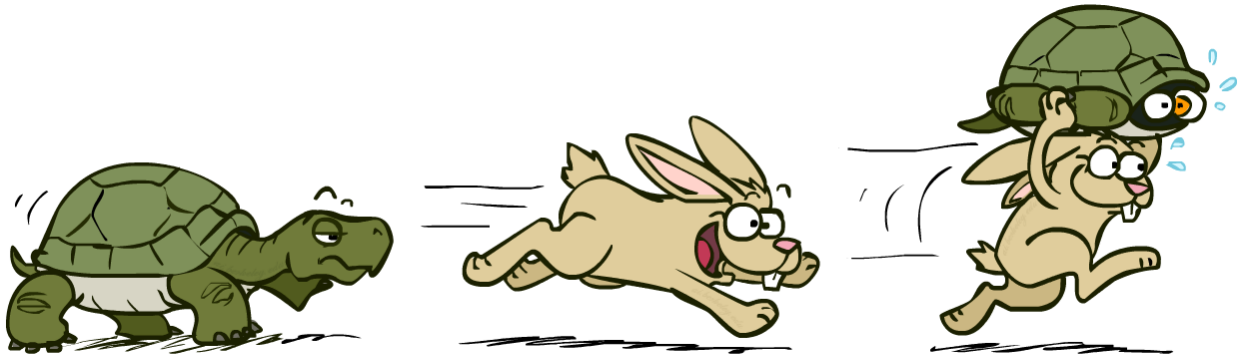
RBFS – ***recursive best first search*** (рекурсивный поиск по первому наилучшему совпадению). Он имеет структуру обычного рекурсивного поиска в глубину, но вместо бесконечного следования вниз вдоль одной ветви контролирует f-значение наилучшего альтернативного пути для текущего узла. Его пространственная сложность равна ***O(bd)***. И **IDA*** и **RBFS** не в состоянии обнаруживать повторяющиеся состояния, поэтому при поиске на графах они склонны к экспоненциальному увеличению временной сложности.

SMA* – ***simplified memory bounded A**** (упрощенный поиск **A*** с ограничением памяти). **SMA*** работает аналогично **A*,** пока не будет исчерпана доступная память. С этого момента он не может добавить новый узел, не уничтожив старый. Уничтожается наихудший узел (с большим значением **f**). Информация о забытом узле запоминается в родительском, что при необходимости позволяет восстановить прерванный поиск.

A*: ВЫВОДЫ

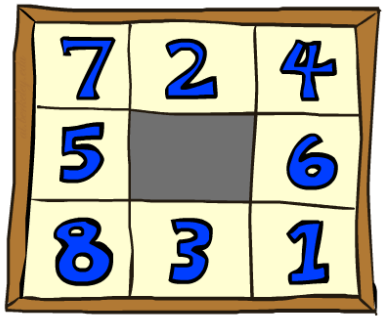
A*: ВЫВОДЫ

- A* использует как прямую стоимость пути к вершине, так и прогнозную стоимость оставшегося пути к цели
- A* оптимальный, если эвристика допустимая и монотонная
- Ключ для проектирования эвристической функции – использование упрощенной задачи

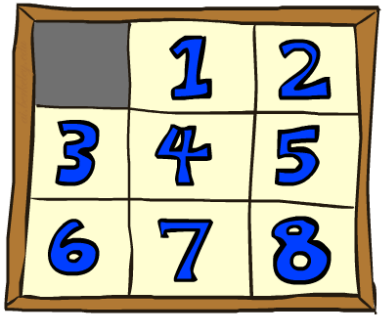


Головоломка 8

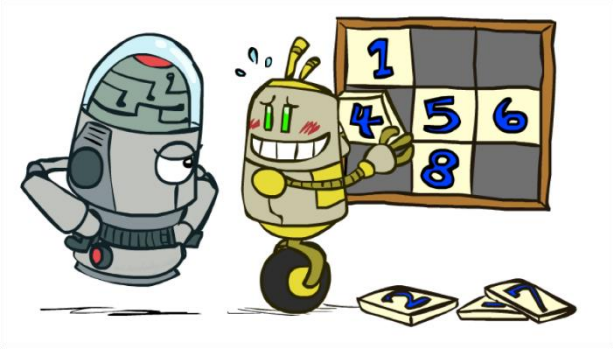
- Эвристика: Число фишек не на своём месте
- *h1*(start) = 8
- Почему она допустима?
- *h1 является допустимой эвристикой, поскольку очевидно, что любая плитка, которая находится не на своем месте, должна быть сдвинута по крайней мере один раз.*



Начальное состояние



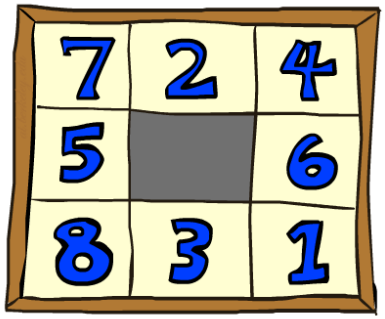
Целевое состояние



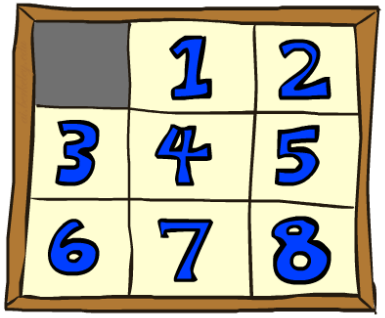
	Среднее число раскрытых вершин, когда длина пути …		
	4 шага	8 шагов	12 шагов
UCS	112	6,300	3.6×10^6
ФИШКИ	13	39	227

Головоломка 8

- А что, если бы мы допустили упрощение головоломки из 8 фишек, в которой любая фишка могла бы скользить в любом направлении, игнорируя другие фишки?
- *h2* =суммарное Манхэттенское расстояние (сумма расстояний плиток от их целевых позиций)
- *h2*(start) = 3+1+2+…=18
- Почему она допустима?



Начальное состояние



Целевое состояние

	Среднее число раскрытых вершин, когда длина пути …		
	4 шага	8 шагов	12 шагов
ФИШКИ	13	39	227
МАНХЭТТЕН	12	25	73

В.Бондарев

Поиск и модели

- Алгоритмы поиска оперируют с моделями мира (среды)
  - Агент в действительности не проверяет все планы в реальном мире (среде)!
  - Планирование происходит путем симулирования
  - Ваш поиск будет настолько хорош насколько хороши модели …