Outline

- Introduction
- Heuristic Search: properties of A*
- Bidirectional Search and the quest for Bi-A*
- LCS*
  - principles, the algorithm, and how it performs
- Main Conclusions
- Pathological cases and open questions
- Application in grids and routing (mainly A*)
  - 6 ideas that we find significant...
Shortest Path Problem

In a locally finite graph $G=(V,E)$, find the shortest path between two designated nodes source $s$ and target $t$ - smallest total cost.

Shortest Path Search

Starting at $s$, the method builds a search tree by repetitive application of the successor operator. A node is expanded when we apply the successor operation on it (the node becomes closed). A node is generated when it is returned by the successor operator (the node becomes open).
**Breadth-First Search (BFS)**

Expand first nodes that are closer to the source, or at the same time nodes that are at the same distance (FIFO).

First Shortest Path Search Methods

- **BFS - Moore 1957** - Goes through all graph nodes, starting with the ones closer to the source. Therefore, it has: unit costs, FIFO structure, complexity $O(n+|E|)$, for $n$ nodes.

- **Shortest-path Search Algorithm - Dijkstra, 1959** - The graph has arc costs and the algorithm seeks the path from a source node to a target node. Complexity $O(n^2)$ or $O(n \log n + |E|)$ with a *priority queue*.

- **Maze Router - Lee, 1961** - An adaptation of BFS, searching in a grid that represents a metal layer, by using numbers from a sequence. After reaching the target, all possible paths with the smallest cost can be identified during the retrace phase. $O(P)$ for a circuit with side $l$. 
The Maze Router

Bidirectional Heuristic Shortest Path Search - Marcelo Johann - 2009
Visit to Intel: Slide 7

The Maze Router

Bidirectional Heuristic Shortest Path Search - Marcelo Johann - 2009
Visit to Intel: Slide 8
The Maze Router

```
X X X X X X X X X X X X X X X X
X 9 8 7 6 5 6 7 8 9 10 11 12  X
X 8 7 6 5 4 5 6 7 8 9 10 11 12 X
X 7 6 5 4 3 4 5 6 7 8 9 10 11 12 X
X 6 5 4 3 2 3 4 5 6 7 8 9 10 11 12 X
X 5 4 3 2 1 2 3 4 5 6 7 8 9 10 11 12 X
X 4 3 2 1 2 3 4 5 6 7 8 9 10 11 12 X
X 3 2 1 2 3 4 5 6 7 8 9 10 11 12 X
X 2 1 2 3 4 5 6 7 8 9 10 11 12 X
X 1 2 1 2 3 4 5 6 7 8 9 10 11 12 X
```

Bidirectional Heuristic Shortest Path Search - Marcelo Johann - 2009
Maze Router's expansion

```cpp
#include <iostream>
#include <queue>
using namespace std;
define SIDE 20
define PLACE(x,y) ((y)*SIDE+(x))
define WEST(n) (n-1)
define EAST(n) (n+1)
define NORTH(n) (n-SIDE)
define SOUTH(n) (n+SIDE)
char Space[SIDE*SIDE];

void init (void)
{
    for (int i=0; i<SIDE*SIDE; ++i)
        Space[i] = ' ';  
    for (int i=0; i<SIDE; ++i)
    {
        Space[PLACE(i,0)] = 'X';
        Space[PLACE(i,SIDE-1)] = 'X';
        Space[PLACE(0,i)] = 'X';
        Space[PLACE(SIDE-1,i)] = 'X';
    }
    for (int i=3; i<SIDE-3; ++i)
    {
        Space[PLACE(i,10)] = 'X';
        Space[PLACE(i,10)] = 'X';
    }
}

void print (void)
{
    for (int i=0; i<SIDE*SIDE; ++i)
        Space[i] = ' ';  
    for (int i=0; i<SIDE; ++i)
    {
        Space[PLACE(i,0)] = 'X';
        Space[PLACE(i,SIDE-1)] = 'X';
        Space[PLACE(0,i)] = 'X';
        Space[PLACE(SIDE-1,i)] = 'X';
    }
    for (int i=3; i<SIDE-3; ++i)
    {
        Space[PLACE(i,10)] = 'X';
        Space[PLACE(i,10)] = 'X';
    }
}

int bfs (int source, int target)
{
    queue<int> q;

    q.push (source);
    while (!q.empty())
    {
        int n = q.front();
        q.pop();
        if (n==target)
            return 1;

        if (Space[n] != 'X')
        {
            Space[n] = 'X';
            print();
            getchar();
            q.push(WEST(n));
            q.push(EAST(n));
            q.push(NORTH(n));
            q.push(SOUTH(n));
        }
    }
    return 0;
}
```

Outline

- **Introduction**
- **Heuristic Search: properties of A**
- Bidirectional Search and the quest for Bi-A*
- **LCS**
  - principles, the algorithm, and how it performs
- Main Conclusions
- Pathological cases and open questions
- Application in grids and routing (mainly A*)
  - 6 ideas that we find significant...
Heuristic Search (A*) [Hart et al. 1968]

Expand first more promising nodes, according to:
\[ f(n) = g(n) + h(n) \]

Also known as Best-First Search

Behaviour of \( f, g \) and \( h \)

Open List: 7
Behaviour of $f$, $g$ and $h$

Open List:
8, 1, 13, 6

Behaviour of $f$, $g$ and $h$

Open List:
1, 13, 6, 9, 2, 14
Behaviour of $f$, $g$ and $h$

Open List: 1, 13, 6, 9, 2, 14

**But not critical as $f^* = 6$**
The A* Algorithm

h = estimation function
The A* Algorithm

Min $f = g + h$     Max $g$
The A* Algorithm

Try all possible straight paths
The A* Algorithm

Maze router-like with obstacles

The A* Algorithm

Maze router-like with obstacles
The A* Algorithm

Some Notation...

- $s, t$ - source and target nodes
- $k(n1,n2)$ - estimate to go from $n1$ to $n2$
- $k^*(n1,n2)$ - actual cost of the shortest path from $n1$ to $n2$
- $h(n) = k(n,t)$
- $h^*(n) = k^*(n,t)$
- $g^*(n) = k^*(s,n)$
- $P_{a-b}$ = path from $a$ to $b$
- $P_{a-b}^*$ = optimal path from $a$ to $b$
- $f^*(n) = \text{cost of } P_{s-n}^* \cup P_{n-t}^* \text{ (going through } n)$
Properties of the Heuristic Estimator

- **Admissibility (\(*\)**): cost from \(n\) to \(t\) \(\geq h(n)\)
  
  ensures the shortest path

- **Consistency**: \(k(n_1,n_2) + k(n_2,n_3) \geq k(n_1,n_3)\)
  
  Every expanded node has optimal cost known
  
  \(g(n) = g^*(n)\)

Properties of the A* Algorithm

- **Continuity** - the search has always an open node \(\in P_{s-t}^*\).
- **Completeness** - A* terminates.
- **Admissibility** - A* finds the optimal path.
- **sufficient condition for expansion** - \(f(n) < P_{s-t}^*\).
- **Necessary condition for expansion** - \(f(n) \leq P_{s-t}^*\).
- **Indeterminism on critical ties** - biasing.
- **Monotonicity** - if \(n_2\) is expanded after \(n_1\) then
  
  \(f(n_2) \geq f(n_1)\).
- **Dominance** - A more informed A* expands less nodes.
- **Excellence** - A* is optimal among all unidirectional algorithms that are no better informed, with a consistent heuristic and no critical ties.
Outline

- Introduction
- Heuristic Search: properties of A*
- Bidirectional Search and the quest for Bi-A*
- LCS*
  - principles, the algorithm, and how it performs
- Main Conclusions
- Pathological cases and open questions
- Application in grids and routing (mainly A*)
  - 6 ideas that we find significant...

Bidirectional Search

[Stålenes & Fekete 1997]

Two simultaneous search fronts
- Meeting node: recognized by both fronts
- Terminating condition: $f(n) > min[f(m)]$
- Leads to overlapping
Bidirectional Search

Two simultaneous search fronts
- Meeting node: recognized by both fronts
- Terminating condition: \( f(n) > min[f(m)] \)
- Leads to overlapping

Bidirectional Search

Two simultaneous search fronts
- Meeting node: recognized by both fronts
- Terminating condition: \( f(n) > min[f(m)] \)
- Leads to overlapping
Two simultaneous search fronts
- Meeting node: recognized by both fronts
- Terminating condition: \( f(n) > \min[f(m)] \)
- Leads to overlapping

First Meeting
Heuristic Bidirectional Search

**Goal**: to combine the advantages of both

**The challenge**:
- Terminating condition
- Overlapping
- The missing fronts problem

[Wave-Shapping]

One front looks at the other

Calculates distance to each node in the opposite front

\[ f(n) = g_s(n) + \min [k(n,p_i) + g_t(p_i)] \]

Requires quadratic time (or space)
Two separate searches following one another
The first BFS and the second A*, usually

It demonstrates the potential of bidirectional heuristic

Outline

- Introduction
- Heuristic Search: properties of A*
- Bidirectional Search and the quest for Bi-A*
- LCS*
  - principles, the algorithm, and how it performs
- Main Conclusions
- Pathological cases and open questions
- Application in grids and routing (mainly A*)
  - 6 ideas that we find significant...
The LCS* Algorithm

**Lowerbound Cooperative Search** [ISA/CI2000]

**Cooperation**
- overall code is like BS* [Kwa 89]
- dynamic estimation (resistances e penalties)
- Function $h$ of one front will be improved by the other’s $g$ values
- estimated values stored in references

Completeness and admissibility proved [SBIA2000]

---

Conditions for expansion...

**Sufficient Condition for Expansion**
- $A^*$ expands all nodes with $f(n) < P_{s\cdot t}^*$

**Necessary Condition for Expansion**
- $A^*$ only expands nodes with $f(n) \leq P_{s\cdot t}^*$

**Missing fronts?**
- No, this is not the main problem! [Kaindl 95,97]
The Pruning Power!

- The power of a heuristic admissible algorithm is not in how fast it finds the goal but in how efficiently it computes higher $f(n)$ values for the generated nodes.

![Diagram](a) Needed Bi-A* search effort  
![Diagram](b) Minimum effort desired in bidirectional search

Dynamic Estimation

- **Resistance** *(min idea [Kaindl 96])*  
  \[ R_t = \text{Min}[g_t(p_i) - k(p_i,t)] \]  
  \[ F(n) = f(n) + R_t \]

- A good idea for BFS, but with A* the minimum values are always at the opposite side of the search
Dynamic Estimation

- **Penalty** *(max idea [Kaindl 96]*)
  \[ P_t = \text{Min}[g_r(p) + k(p,s)] \]
  \[ F(n) = g_s(n) + P_t - h_t(n) \]

- LCS* uses both, as penalties can be smaller than resistances in some cases

Cooperation and Visibility

Function *h* of one front will be improved by the other’s *g* values

But *h* is under-estimated while *g* is over-estimated

**In previous works:**

- *g* and *h* were associated with the nodes themselves

**Visibility:**

- estimated values stored in references
- single public closed set for both fronts allows cooperation
**Admissibility**

- There are always references to nodes in the optimal path in both search fronts;
- The algorithm terminates only after they meet;

**Prunning Power**

- Less nodes are expanded by increasing $f(n)$ values
- Normal values are used in the open lists, but single quantities $\Omega$ (resistance) and $P$ (penalty) are subtracted from $L_{\text{min}}$ (best cost so far)

---

**LCS* is Admissible**

**LCS*’s code:**

Too many details!
Ask me later on if you want...
Results: LCS* versus A* [Johann 2000]

Expanded nodes and actual runtime: typical

200 by 200 grids with random costs, mean 100, minimum 50, maximum 300

Ratio LCS*/A* vs. range of random costs

Nodes and runtime comparison.

Specification of random costs

Highest and lowest values with mean.

Range, minimum, maximum, AC costs, and DC cost specification.
\[ \epsilon \text{-Admissibility in A* and LCS*} \]

- It is to find a path which is within \( \epsilon \) from the shortest one.
- In heuristic search this is accomplished by simply multiplying the original \( h \) function by \( 1 + \epsilon \).
- But in LCS*, you can also check how far each meeting node is from a possible optimal by comparing it to the smallest open \( f \).
Testing $\varepsilon$-admissibility

LCS*/A* in 2D grids, regular cost regime

200 by 200 grids with random costs, mean 100, minimum 50, maximum 300

ratio LCS*/A*

- 100%
- 96%

range of random costs

LCS* wins on harder mazes

100x100 random BFS, DFS and multi-DFS mazes

<table>
<thead>
<tr>
<th>Maze type</th>
<th>Time</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>0.698</td>
<td>0.661</td>
</tr>
<tr>
<td>DFS</td>
<td>1.035</td>
<td>0.975</td>
</tr>
<tr>
<td>Multi-DFS</td>
<td>0.612</td>
<td>0.595</td>
</tr>
</tbody>
</table>
LCS* wins on random graphs

70 nodes, all 70x70 path problems

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>random</th>
<th>solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>average A*</td>
<td>126156</td>
<td>78306</td>
</tr>
<tr>
<td>best A*</td>
<td>71576</td>
<td>61016</td>
</tr>
<tr>
<td>worst A*</td>
<td>18036</td>
<td>95596</td>
</tr>
<tr>
<td>LCS*</td>
<td>77710</td>
<td>59869</td>
</tr>
</tbody>
</table>

A* almost optimal in geometric graphs

70 nodes, all 70x70 path problems

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>geometric</th>
<th>plus one arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>average A*</td>
<td>39095</td>
<td>28406</td>
</tr>
<tr>
<td>best A*</td>
<td>24438</td>
<td>21428</td>
</tr>
<tr>
<td>worst A*</td>
<td>53752</td>
<td>35384</td>
</tr>
<tr>
<td>LCS*</td>
<td>38662</td>
<td>34130</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Heuristic Search: properties of A*
- Bidirectional Search and the quest for Bi-A*
- LCS*
  - principles, the algorithm, and how it performs
- Main Conclusions
- Pathological cases and open questions
- Application in grids and routing (mainly A*)
  - 6 ideas that we find significant...

Main Conclusions

- A* is way easier to implement. Always the first choice.
- LCS* will usually lose when the area is empty, as A* expands the smallest number of nodes. (but it’s on easy instances)
- LCS* can get you a small constant advantage, and can be significantly better when the heuristic is bad, like in congested routing, or when one of the pins (A* target) is blocked.
Pathological cases

- When one of the pins is in a very congested area, A* starting from this pin has an advantage, for when the search leaves the congested area, it goes direct to the target. LCS* will always duplicate its efforts.
- Similar case happens with a barrier in the middle. A* wins with 50% probability.

Open question

**Direction choosing criteria:**
- Alternating
- Smallest f
- Cardinality
- Inertial Cardinality
- Expanding from the most difficult first
  - But this contradicts others
Outline

- Introduction
- Heuristic Search: properties of A*
- Bidirectional Search and the quest for Bi-A*
- LCS*
  - principles, the algorithm, and how it performs
- Main Conclusions
- Pathological cases and open questions
- Application in grids and routing (mainly A*)
  - 6 ideas that we find significant...

Application in grids and routing

1. Do we need admissibility?
2. Degrees of freedom
3. Cost functions: add some processing
4. Optimizing data structures
5. How to avoid blocking pins
6. Grid properties and some ideas
1. Admissibility: do we need it?
- LCS* is a nice proved admissible algorithm
- But who cares?
- In most cases we don’t need full admissibility
- The advantage of LCS is that it can compute $\varepsilon$-admissibility in two ways.
- The disadvantage is that if you don't need to guarantee $\varepsilon$, then forget LCS* and make a simple Bi-A* that terminates in the first meeting.

2. Degrees of freedom

During expansion
- On ties, use biasing, like in AMAZE
  [Hentschke ISPD 2007]

And during retrace?
- Need to record multiple pointers:
Bidirectional Heuristic Shortest Path Search - Marcelo Johann - 2009

**Uniform costs**

Open List: 13

**Critical Ties**

Open List: 14, 7

\[ f(14) = f(7) = f^* \]
Can choose what you want

Nodes close to the target first
Nodes close to the target first

Open List: 15, 7, 2, 3

Only four nodes expanded

But pay attention to:

Open List: 15, 7, 2, 3

Degree of freedom
But pay attention to:

Open List: 15,7,2,3

Degree of freedom
But pay attention to:

Open List: 15, 7, 2, 3

The A* Algorithm

h = estimation function
The A* Algorithm

Min $f = g + h$ \hspace{1cm} \text{Max } g
The A* Algorithm

Straight path from S to T

The A* Algorithm

Biasing already considered

Degrees of freedom
The A* Algorithm

But all this area will be expanded anyway...

After it escapes, A* again
3. Cost Functions

- This is a delicate question for routing, but let's just consider we may have different scenarios.
- **So, things we can do:**
- You can introduce noise (dither) in the cost function to better randomize the paths. But then you loose degrees of freedom.
- On the contrary, you can round the costs down to low resolution, as to increase the number of ties.
3. Cost Functions

Also, costs represent conflicting factors:

1. resource usage (how much metal this takes)
2. Path quality, = delay
3. Congestion

So they can be weighted for net, factor, layer
It helps taking better paths with less effort
4. Optimizing Data Structures

A* expands nodes with increasing values of f. If the maximum cost per arc is known, we have a limited range for f values at the same time in the open list. Therefore, we can implement it as an array of lists. Also, the ordering of each list can be calculated only when a new f value is needed.
5. To avoid blocking pins

Typically the number of alternate paths is proportional to the distance from a given pin.
So the probability of getting blocked is higher close to the pins.
We can minimize blocking by reserving some escapes for a pin.
But fixed reservations compromise freedom, and should be used only when there is a single way out.
For multiple possible accesses, a technique of alternate reservations was proposed by Johann/Reis in 1995:

- Place an alternate reservation in one position for each pin, maintaining a list of other options to it. When drawing a connection during retrace, if this connection passes on an alternate reservation, eliminate the reservation and check for the other options. If there are more than one still available, chose one of them to hold an alternate reservation. On the other hand, if there is only one way out remaining, make it a fixed reservation. In both cases, no change is made to the path being recorded in the current retrace.

6. Grid Properties

Let us look at this again...
6. Grid Properties

It's clear that this area could not reach t with cost f=10.

This is still an open problem...
Thank you!

Marcelo Johann
johann@inf.ufrgs.br
www.inf.ufrgs.br/~johann