Seminar: Search and Optimization

2. Basic Search Algorithms & Project Organization

Martin Wehrle

Universität Basel

September 24, 2015

Today's Session

Topics for today

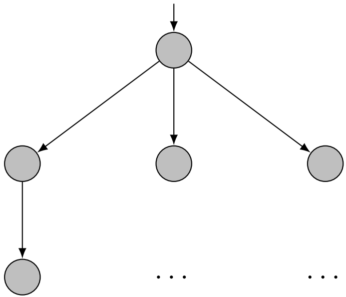
- Introduction to basic search algorithms
- Topic assignment for the seminar
- Organization of the project

Basic Search Algorithms

Search Algorithms

Search algorithms and state spaces

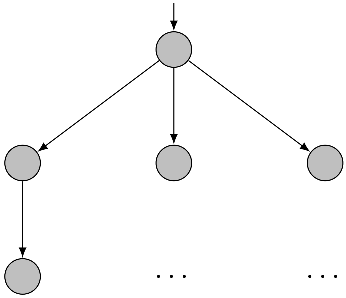
- Search algorithms work in state spaces.
- State spaces consist of states and state transitions, as well as an initial state and (potentially many) goal states.
- Objective of search algorithms: find a path from the initial to a goal state in the state space.



Search Algorithms

Working principle of search algorithms

- Start with initial state. In every step, expand a state through generating its successors.
- Open List: Set of states that are candidates for expansion
- Closed List: Set of states that are already expanded



Blind Search Algorithms

Blind (or uninformed) search algorithms

Use no additional information about the state space beyond the problem definition. In the following:

- Breadth-first search
- Uniform-cost search

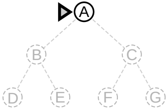
In contrast to

heuristic search algorithms ($\rightsquigarrow$ introduced later)

Breadth-First Search

Breadth-first search

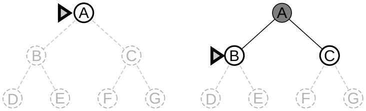
States are expanded in the order they have been generated (FIFO).



Breadth-First Search

Breadth-first search

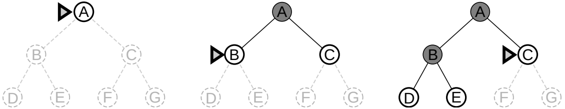
States are expanded in the order they have been generated (FIFO).



Breadth-First Search

Breadth-first search

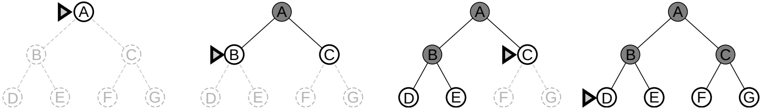
States are expanded in the order they have been generated (FIFO).



Breadth-First Search

Breadth-first search

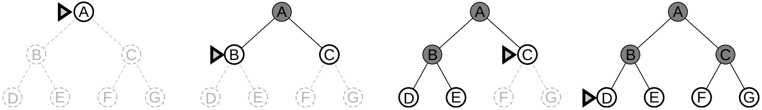
States are expanded in the order they have been generated (FIFO).



Breadth-First Search

Breadth-first search

States are expanded in the order they have been generated (FIFO).



- searches the state space layer by layer
- always finds a solution if a solution exists
- always finds a shallowest goal state first
- optimal in case all actions have the same costs

Breadth-First Search: Pseudo-Code

Breadth-first search: pseudo-code

```
s0 := initial state
if is-goal(s0):
    return extract-solution(s0)
open := new FIFO queue with s0 as the only element
closed := ∅

loop do
    if open.empty():
        return none
    s = open.pop-front()
    closed.insert(s)
    for each successor state s′ of s:
        if s′ ∉ open ∪ closed:
            if is-goal(s′):
                return extract-solution(s′)
            open.push-back(s′)
```

Uniform-Cost Search

Uniform-cost search

- Breadth-first search not optimal if actions have different costs
- Solution: always expand states with minimal path costs $g(s)$
- Implementation: priority queue as open list

$\rightsquigarrow$ uniform-cost search (also known as Dijkstra's algorithm)

Uniform-Cost Search

Uniform-cost search

s_0 := initial state
open := **new** priority queue, ordered by g
open.insert(s_0)
closed := $\emptyset$
while not *open*.empty():
    s = *open*.pop-min()
    if $s \notin closed$:
        closed := *closed* $\cup \{s\}$
        if is-goal(s):
            return extract-solution(s)
        for each successor state s' of s:
            open.insert(s')
return unsolvable

Heuristic Search Algorithms

Drawback of blind search algorithms: Limited scalability

Idea

- Find criteria to estimate which states are "good" and which states are "bad" $\rightsquigarrow$ prefer good states

State evaluation

- Use a heuristic function $h(s)$ to estimate the quality of states s
- Based on h, compute evaluation function f
- Evaluate every state s with f (i. e., compute $f(s)$)
- Expand state with minimal f value next

$\rightsquigarrow$ prominent example: A^* search algorithm

A^* Search Algorithm

A^* search algorithm

- Evaluation function $f(s) := g(s) + h(s)$
- Balance path costs $g(s)$ and estimated proximity $h(s)$ to goal
- Intuition: $f(s)$ estimates costs of cheapest solution from initial state through s to goal

A* Search: Pseudo-Code

A* search (no re-opening)

```
s_0 := initial state
open := new priority queue, ordered by f
if h(s_0) < ∞:
    open.insert(s_0)
closed := ∅
while not open.empty():
    s = open.pop-min()
    if s ∉ closed:
        closed := closed ∪ {s}
        if is-goal(s):
            return extract-solution(s)
        for each successor state s' of s:
            if h(s') < ∞:
                open.insert(s')
return unsolvable
```

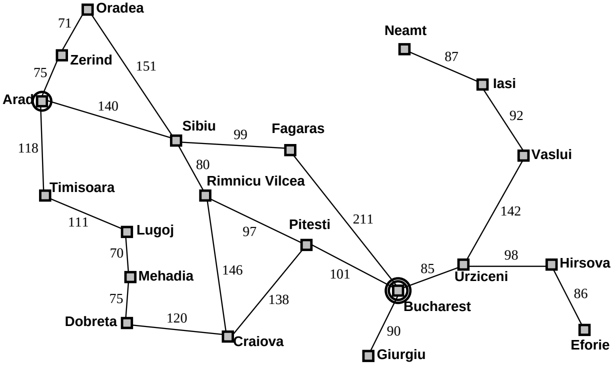
A^* Search Algorithm

Most important property

- A^* is optimal if the applied heuristic is admissible.

Example: A* for Route Planning

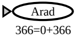
Example heuristic: straight-line distance to Bucharest



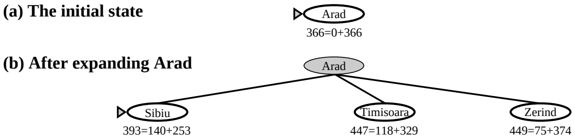
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example: A^* for Route Planning

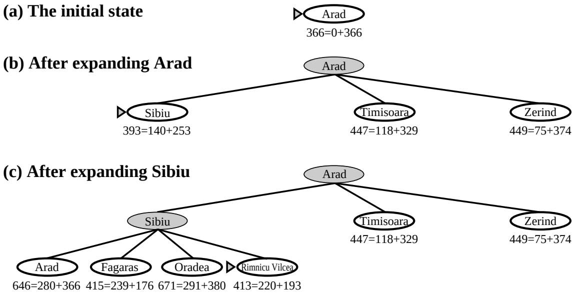
(a) The initial state



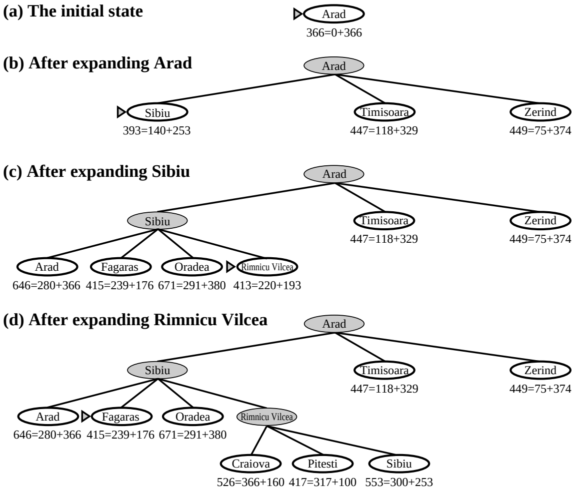
Example: A* for Route Planning



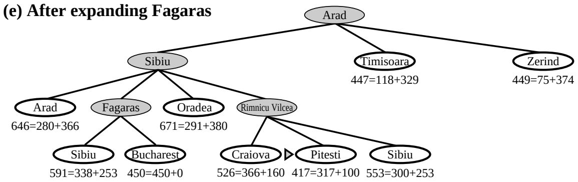
Example: A* for Route Planning



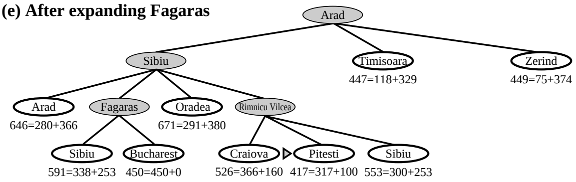
Example: A* for Route Planning

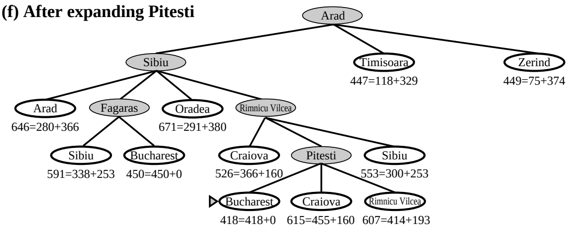


Example: A* for Route Planning



Example: A* for Route Planning





Seminar Topic Assignment

Seminar Schedule

- 13 registered participants
- Every participant has been assigned a topic marked with Yes

Seminar Schedule

17.09. Organization, schedule & seminar topics
24.09. Basic search algorithms & project organization
01.10. no meeting
08.10. no meeting
15.10. Viacheslav Sharunov
22.10. Andreas Thüring + project milestone 1
29.10. Samuel Bader + Ziba Tavassoli
05.11. Dorde Relic + Marko Obradovic
12.11. no meeting
19.11. Daniel Federau + project milestone 2
26.11. Oleksandr Dombrovskyi + Kadir Özgür
03.12. Maurus Dähler + Mirko Riesterer
10.12. Patrick Buder
17.12. Wrap-up and final project presentation

Topic Assignment

Pathfinding

- 15.10.: Near Optimal Hierarchical Path-Finding
  Viacheslav Sharunov (supervisor: Jendrik Seipp)
- 22.10.: Subgoal Graphs for Fast Optimal Pathfinding
  Andreas Thüring (supervisor: Martin Wehrle)
- 29.10.: Improved heuristics for optimal path-finding on game maps
  Samuel Bader (supervisor: Martin Wehrle)
- 29.10.: TRANSIT Routing on Video Game Maps
  Ziba Tavassoli (supervisor: Gabi Röger)

Topic Assignment

Real-time strategy games

- 5.11.: UCT for tactical assault planning in Real-Time Strategy Games
  Dorde Relic (supervisor: Silvan Sievers)
- 5.11.: Game-Tree Search over High-Level Game States in RTS Games
  Marko Obradovic (supervisor: Manuel Heusner)
- 19.11.: Build order optimization in StarCraft
  Daniel Federau (supervisor: Silvan Sievers)

Topic Assignment

Content generation & playing games

- 26.11.: Procedural Content Generation
  Oleksandr Dombrovskyi (supervisor: Florian Pommerening)
- 26.11.: Techniques for AI-Driven Experience Management in Interactive Narratives
  Kadir Özgür (supervisor: Florian Pommerening)
- 3.12.: Towards Automatic Personalized Content Generation for Platform Games
  Maurus Dähler (supervisor: Salomé Simon)
- 3.12.: Answer Set Programming for Procedural Content Generation: A Design Space Approach
  Mirko Riesterer (supervisor: Salomé Simon)
- 10.12.: Learning to Win by Reading Manuals in a Monte-Carlo Framework
  Patrick Buder (supervisor: Thomas Keller)

Organization: Update

Update on seminar organization

- Due to the large number of seminar talks, everyone is only supposed to read one paper per session (instead of all papers).
- The paper to read is announced one week in advance (by mail)
- LaTeX template for seminar papers will be available on website

Project Organization

Project Organization

Project topic

- Grid-based pathfinding competition
- Implementation of pathfinding algorithms on grids

Programming framework

- API based on the Grid-Based Path Planning Competition (`http://movingai.com/GPPC/`)
- Provides infrastructure (like parsing, basic search algorithm) for implementations of pathfinding algorithms
- Adapted infrastructure for project hosted at `bitbucket.org`
- Programming language: C++

Project Organization

Benchmarks

- "Real-world" game maps (Dragon Age, Starcraft, . . .)
- Encoding of maps containing obstacles (trees, water, . . .)
- Benchmark set publicly available at `http://www.movingai.com/benchmarks/`
- File format described at `http://www.movingai.com/benchmarks/formats.html`

Benchmark format

Benchmarks consist of two files

- `.map`: encoding of the map to search on
- `.map.scen`: the scenario (e. g., start and goal locations)

Project Organization

Organization

- Teams of at most 2 persons
- No fixed supervisor
- If you have questions or want to meet: contact us directly

Workflow

1. Create account at `bitbucket.org` and tell us your name
2. We will grant access to our repo on project infrastructure
3. Project work is done on (forked) bitbucket repository

For all project milestones

- Out of existing files, changes allowed only to `Entry.h/cpp`

Project Organization

Milestone 1

- Familiarize yourself with API and benchmark format
- Proof-of-concept implementation of uniform-cost search
- Deadline: October 22

Performance target for milestone 1

Solve `AcrossTheCape` with uniform-cost search in ≤ 1 minute.

Project Organization

Milestone 2

- Implementation of at least one additional optimal algorithm
- Deadline: November 19

Project Organization

Milestone 3

- Open (also suboptimal algorithms)
- In particular: Optimize for efficiency
- Deadline: December 17

Project Organization

Grading

- Performance (time, solution quality) on benchmark selection
- Quality of code
- Milestone presentations