Soundness and Completeness of State Space Search

Sertac Karaman
16.410-13
Sept 20th, 2010

1

Assignments

- Remember:
  Problem Set #2: Uninformed search
  Out last Wednesday,
  Due this Wednesday, September 22nd
- Reading:
  - Today: Proofs and Induction: Lecture 2 and 3 Notes of 6.042.
  - Wednesday: [AIMA] Ch. 6.1; 24.3-5 Constraint Satisfaction.
    - To learn more: Constraint Processing, by Rina Dechter
      - Chapter 2: Constraint Networks
      - Chapter 3: Consistency Enforcing and Propagation

2

Autonomous Systems:

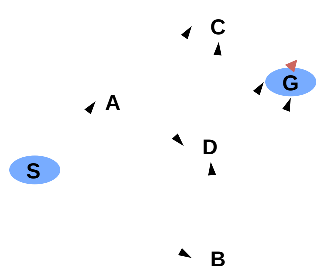
- Plan complex sequences of actions
- Schedule tight resources
- Monitor and diagnose behavior
- Repair or reconfigure hardware.

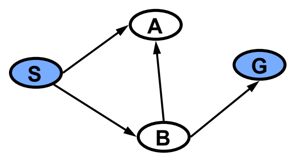
⇨formulate as state space search.

Formalizing Graph Search

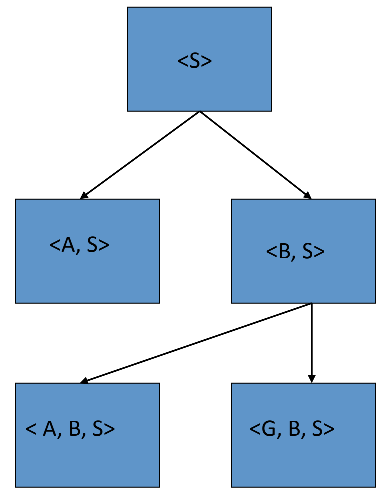
Input: A search problem SP = <g, S, G> where
- **graph g = <V, E>,**
- **start vertex S in V, and**
- **goal vertex G in V.**

Output: A simple path P = <S, v2, … G> in g from S to G.
(i.e., $<v_i,v_{i+1}> \in E$, and $v_i \neq v_j$ if $i \neq j$).

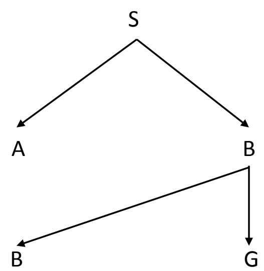




Graph Search is a Kind of State Space Search

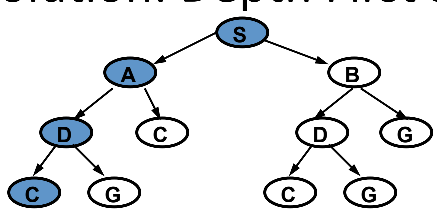


Graph Search is a Kind Of Tree Search

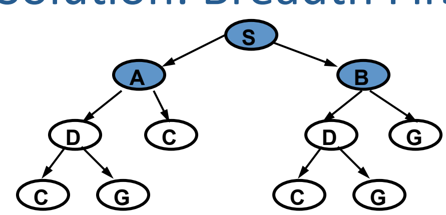


Brian Williams, Fall 10 5

Solution: Depth First Search (DFS)

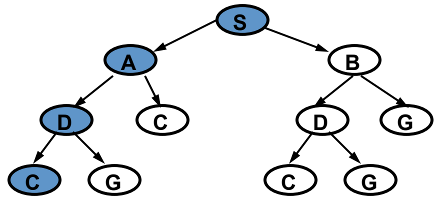


Solution: Breadth First Search (BFS)



Brian Williams, Fall 10 6

Solution: Depth First Search (DFS)

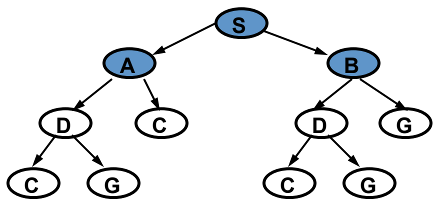


Depth-first:

Add path extensions to **front** of Q

Pick first element of Q

Solution: Breadth First Search (BFS)



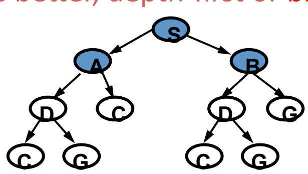
Breadth-first:

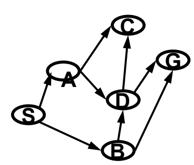
Add path extensions to **back** of Q

Pick first element of Q

The Worst of The Worst

Which is better, depth-first or breadth-first?





- Assume d = m in the worst case, and call both m.
- Best-first can't expand to level m+1, just m.

Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first	$\sim b^m$	b^m	Yes unit lngth	Yes

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Elements of Algorithm Design

Description: (last Monday)
- Problem statement.
- Stylized pseudo code, sufficient to analyze and implement the algorithm.
- Implementation (last Wednesday).

Analysis: (last Wednesday)
- Performance:
  - Time complexity:
    - how long does it take to find a solution?
  - Space complexity:
    - how much memory does it need to perform search?
- Correctness: (today)
  - Soundness:
    - when a solution is returned, is it guaranteed to be correct?
  - Completeness:
    - is the algorithm guaranteed to find a solution when there is one?

Outline

- Review
- Proof techniques and the axiomatic method
- Proofs of soundness and completeness of search algorithms
- Limits of axiomatic method

Envelope game

Probabilities do not work!

- I put an amount \$N and \$2N into two different envelopes *(you do not know N)*.
- I open one of them, it has \$X.
- Would you pick the open one or the other?
- ***Reasoning 1***: (*I pick one at random)*
  – seeing inside an envelope does not matter…
- ***Reasoning 2:*** *(I pick the second one)*
  – If I get this envelope, I get \$X.
  – If I get the other envelope, I get, on average: (1/2) X/2 + (1/2) 2X = (5/4) X

Unexpected hanging paradox

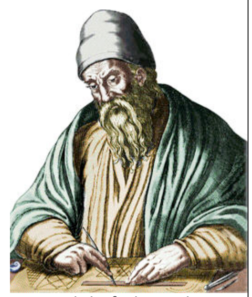
Induction does not work!

- A judge tells a criminal that *"the criminal will be hanged on a weekday at noon next week, he will not know when he will be hanged, it will be a total surprise".*
- Criminal's reasoning:
  – He can not be hanged on a Friday (by Thursday afternoon, he will know – it won't be a surprise).
  – Then, he can not be hanged on Thursday either.
  – Then, he can not be hanged at all… So he feels safe.

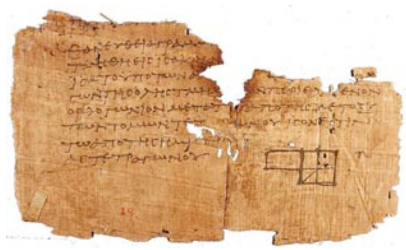
  (He was hanged on Wed. at noon – it was a total surprise…)
- What went wrong with criminal's deduction?

The axiomatic method

- Invented by Euclid around 300BC (in Alexandria, Egypt).
- 5 ***axioms*** of geometry mentioned in his work *Elements.*
- Starting from these ***axioms***, Euclid established many ***"propositions"*** by providing ***"proofs".***



Euclid of Alexandria



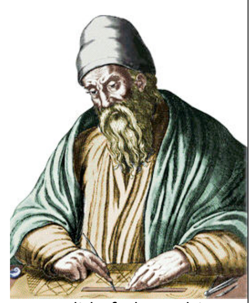
Oldest surviving copy of the *Elements* on a papyrus found in *Oxyrhynchus, Egypt*.

Euclid statute in Oxford

Images are in the public domain.

The axiomatic method

- A definition of a ***"proof"***:
  - Any sequence of *logical deductions* from *axioms* and previously proven *propositions/ statements* that concludes with the proposition in question.
- There are many types of ***"propositions"***:
  - Theorem: Important results, main results
  - Lemma: a preliminary proposition for proving later results
  - Corollary: An easy (but important) conclusion, an afterthought of a theorem.



Euclid of Alexandria

Euclid statute in Oxford

The axiomatic method

- Euclid's axiom-proof approach is now fundamental to mathematics!
- Amazingly, essentially all mathematics can be derived from just a handful of axioms…
- How to even start a proof?
  - There are many "templates" (outlines, or techniques)
  - The details differ…

Euclid of Alexandria

Euclid statute in Oxford

Proving an ***implication***

- Several mathematical claims are of the form:
  - ***"If P, then Q"***, or equivalently ***"P implies Q".***
  - **Quadratics:**
    - If $ax^2 + bx + c = 0$ and $a \neq 0$, then $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
  - **Inequalities:**
    - If $0 \leq x \leq 2$, then $-x^3 + 4x + 1 > 0$
  - **Goldbach's Conjecture**:
    - If *n* > 2, then *n* is a sum of two primes.

Proving implications:
Simplest proof technique

- To prove "P implies Q",
  - "Assume P" and show that Q logically follows.

- Theorem:
  - If $0 \leq x \leq 2$, then $-x^3 + 4x + 1 > 0$
- Proof:
  - Assume $0 \leq x \leq 2$ (P)
  - Then, x, $2 - x$, and $x + 2$ are all non-negative
  - Then, $x(2-x)(2+x)$ is non-negative
  - Then, $x(2-x)(2+x) + 1$ is non-negative
  - Then, multiplying out the left side gives
    - $-x^3 + 4x + 1 > 0$ (Q)

Proof by Contradiction

- To prove that a statement P is `True`.
  - Assume that it is not.
  - Show that some absurd (clearly false) statement follows.

- ***Formalized***: In order to prove a statement P is `True`
  - Assume that P is `False`,
  - Deduce a logical contradiction (negation of a previous statement or an axiom).

Proof by Contradiction

- ***Theorem***: $\sqrt{2}$ is an irrational number.
- ***Proof***:
  - Assume that $\sqrt{2}$ is not irrational.
  - Then, $\sqrt{2}$ is a rational number and can be written as $\sqrt{2} = a/b$ where a and b are integers and *fraction is in lowest terms*
  - Then, squaring both sides and rearranging gives $2 = a^2/b^2$
  - Then, a must be even
  - Then, a^2 must be a multiple of 4
  - Then, $2b^2$ must also be a multiple of 4
  - Then, b is also even
  - Then, the fraction is **not** in the lowest terms (since a and b are both even)

Proof by Induction

Pick parameter N to define problem size.

- Number of edges in the solution.
- Number of nodes in graph.

Base Step: N = 0 (or small N)

- Prove property for N = 0

Induction Step:

- Assume property holds for N
- Prove property for N+1

- **Conclusion**: property holds for all problem sizes N.

20

Proof by Induction *formalized*

- Let P(*i*) be a statement with parameter *i.*
- Proof by induction states the following implication:
  - "P(0) is `True`" (1) and "P(*i*) implies P(*i+1*)" (2)
  - (1) and (2) *implies* "P(*i*) is `True` for all i".
- Induction is one of the *core principles* of mathematics.
- It is generally taken as an axiom, or the axioms are designed so that induction principle can be proven.

21

An induction example

- ***Theorem***: $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$ for all *n.*
- ***Proof***:
  - *Base case*: P(0) is `True`.
    - Because, 0 = 0.
  - Induction step: P(*n*) implies P(*n*+1)
    - Assume that the hypothesis holds for *n.*
    - For n + 1:

$$\begin{aligned} 1 + 2 + \cdots + n + (n+1) &= \frac{n(n+1)}{2} + n + 1 \\ &= \frac{(n+1)(n+2)}{2} \end{aligned}$$

A faulty induction

- ***Theorem[!]***: All horses are the same color.
- ***Proof[!]***:
  - *Base case*: P(1) is `True`.
    - because, there is only one horse.
  - *Induction step*: P(*i*) implies P(*i*+1).
    - Assume that P(i) is `True`.
    - By the *induction hypothesis* first *i* horses are the same color, and the last *i* horses are also the same color.

$$h_1, h_2, \ldots, h_i, h_{i+1} \qquad h_1, h_2, \ldots, h_i, h_{i+1}$$

    - So all the *i*+1 horses must be the same color.
    - Hence, P(*i*+1) is also `True`.

- What went wrong here?

Proof by Invariance

A common technique in algorithm analysis

- Show that a certain property holds throughout in an algorithm.
- Assume that the property holds initially.
- Show that in any step that the algorithm takes, the property still holds.
- Then, property holds forever.
- It is a simple application of induction. Why?

Proving statements about algorithms

Handle with care!

- Correctness of simplest algorithms may be very hard to prove…

- Collatz conjecture:
  - Algorithm (**H**alf **O**r **T**riple **P**lus **O**ne - HOTPO):
    - Given an integer *n.*
    1. If *n* is even, then *n* = *n*/2
    2. If *n* is odd, then *n* = 3*n* + 1
    3. If n = 1, then terminate, else go to step 1.
  - *Conjecture*: For any n, the algorithm always terminates (with n = 1).

Proving statements about algorithms

Handle with care!

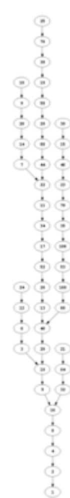
Collatz conjecture:

- First proposed in **1937**.

- It is ***not*** known whether the conjecture is true or false.

Paul Erdős *(1913-1996)*
- famous number theorist –

"Mathematics is not yet ready for such problems", 1985.



First 100 numbers

First 1000 numbers

Soundness and Completeness of Search Algorithms

- **Today**:

- prove statements about the search algorithms we have studied in the class.

- study whether the algorithm returns a correct solution.
- study whether the algorithm returns a solution at all when one exists.

Soundness and Completeness

Given a *problem* PR, an *algorithm* that attempts to solve this problem may have the following properties:

Soundness:
- The solution returned by the algorithm is correct.

Completeness:
- The algorithm always returns a solution, if one exists.
- If there is no solution, the algorithm reports failure.

Also, **Optimality**:
- The algorithm returns the optimal solution, if it returns one.

Some Other Notions of Soundness and Completeness

Probabilistic Completeness:
- The algorithm returns a solution, if one exists, with probability approaching to one as the number of iterations increases.
- If there is no solution, it may run for forever.

Probabilistic Soundness:
- The probability that the "solution" reported solves the problem approaches one, as the number of iterations increases.

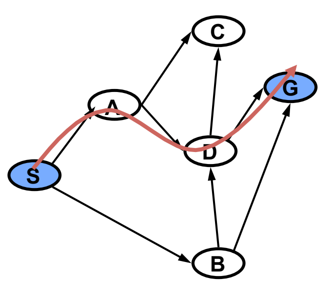
Asymptotic Optimality:
- The algorithm does not necessarily return an *optimal* solution, but the cost of the solution reported approaches the optimal as the number of iterations increases.

29

Problem: State Space Search

Input: A search problem S = <g, S, G> where
- **graph g = <V, E>,**
- **start vertex S in V, and**
- **goal vertex G in V.**

Output: A simple path P = <S, v2, … G> in g from S to G.



Brian Williams, Fall 10

30

Pseudo Code For Simple Search

Let g be a Graph   G be the Goal vertex of g.
S be the start vertex of g   Q be a list of simple partial paths in GR,

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N;   (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

31

Soundness and Completeness Theorems

We would like to prove the following two theorems:

Theorem 1 (Soundness):
Simple search algorithm is sound.

Theorem 2 (Completeness):
Simple search algorithm is complete.

We will use a blend of proof techniques for proving them.

32

Soundness and Completeness Theorems

Theorem 1 (Soundness):
Simple search algorithm is sound.

Let us prove 3 lemmas before proving this theorem.

33

A lemma towards the proof

- **Lemma 1**: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then v_k = S.
- **Proof**: (by invariance)
  - *Base case*: Initially, there is only <S> in the queue. Hence, the *invariant holds*.
  - *Induction step*: Let's check that the *invariant* continues to hold in every step of the algorithm.

Pseudo Code For Simple Search

***Invariant*: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then v_k = S.**

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

35

Pseudo Code For Simple Search

***Invariant*: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then v_k = S.**

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

***Before this line:* assume that invariant holds.**
***After this line:* show that invariant is still true.**
In this case no new path is added to the queue.

36

Pseudo Code For Simple Search

***Invariant*: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then v_k = S.**

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

37

Pseudo Code For Simple Search

***Invariant*: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then v_k = S.**

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue (a path is removed) 38

Pseudo Code For Simple Search

Invariant: **If $\langle v_1, v_2, \ldots, v_k \rangle$ is a path in the queue at any given time, then v_k = S.**

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
Several paths added, each satisfy the invariant since N satisfies it.

39

Pseudo Code For Simple Search

Invariant: **If $\langle v_1, v_2, \ldots, v_k \rangle$ is a path in the queue at any given time, then v_k = S.**

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

40

Pseudo Code For Simple Search

***Invariant*: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then v_k = S.**

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

***Before this line:* assume that invariant holds.**
***After this line:* show that invariant is still true.**
In this case no new path is added to the queue.

41

Another lemma towards the proof

- **Definition:** A path <v_0, v_1, …, v_k> is valid if
  $(v_{i-1}, v_i) \in E$ for all $i \in \{1, 2, \ldots, k\}$
- **Lemma 2**: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is valid.
- **Proof**: (by invariance)
  - *Base case*: Initially there is only one path <S>, which is valid. Hence, the *invariant* holds.
  - *Induction step*: Let's check that the *invariant* continues to hold in every step of the algorithm.

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is valid

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

43

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is valid

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

44

Pseudo Code For Simple Search

Invariant: If $<v_1, v_2, \ldots, v_k>$ is a path in the queue at any given time, then it is valid

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

45

Pseudo Code For Simple Search

Invariant: If $<v_1, v_2, \ldots, v_k>$ is a path in the queue at any given time, then it is valid

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

46

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is valid

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

47

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is valid

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
Note that validity holds for all newly added path (from Line 4.b)

48

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is valid

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

49

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is valid

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

50

Yet another lemma towards the proof

- **Lemma 3**: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path (contains no cycles).
- **Proof**: (by invariance)
  - *Base case*: Initially, there is only <S> in the queue. Hence, the *invariant holds*.
  - *Induction step*: Let's check that the *invariant* continues to hold in every step of the algorithm.

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

52

Pseudo Code For Simple Search

***Invariant*:** If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

53

Pseudo Code For Simple Search

***Invariant*:** If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

54

Pseudo Code For Simple Search

***Invariant*:** If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

55

Pseudo Code For Simple Search

***Invariant*:** If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

56

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

We would like to show that each newly added path is simple assuming N is simple.
Proof: (by contradiction) *Assume one path is not simple. Then, a children of head(N) appears in N. But, this is contradicts Line 4.b*

57

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

58

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

Before this line: **assume that invariant holds.**
After this line: **show that invariant is still true.**
In this case no new path is added to the queue.

59

Soundness and Completeness Theorems

Theorem 1 (Soundness):
Simple search algorithm is sound.

Proof: by contradiction…

60

Proof of Soundness

Assume that the search algorithm is ***not* sound**:
Let the returned path be $< v_0, v_1, \ldots, v_k >$
Then, one of the following must be `True`:

- 1. Returned path does not start with S:
  $v_k \neq S$
- 2. Returned path does not contain G at head:
  $v_0 \neq G$
- 3. Some transition in the returned path is not valid:
  $(v_{i-1}, v_i) \notin E$ for some $i \in \{1, 2, \ldots, v_k\}$
- 4. Returned path is not simple:
  $v_i = v_j$ for some $i, j \in \{0, 1, \ldots, k\}$ with $i \neq j$

61

Proof of Soundness

- 1. Returned path does not start with S: $v_k \neq S$
- But, this contradicts Lemma 1!
- **Lemma 1**: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then v_k = S.

62

Proof of Soundness

- 2. Returned path does not contain G at head:

  $v_0 \neq G$

  - But clearly, the returned path has the property that Head(N)= G
  - Recall the pseudo code:

63

Pseudo Code For Simple Search

Invariant: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path.

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

64

Proof of Soundness

- 3. Some transition in the returned path is not valid:
  $(v_{i-1}, v_i) \notin E$ for some $i \in \{1, 2, \ldots, v_k\}$
- Contradicts Lemma 2!
- **Lemma 2**: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is valid.

65

Proof of Soundness

- 4. Returned path is not simple:
  $v_i = v_j$ for some $i, j \in \{0, 1, \ldots, k\}$ with $i \neq j$
- Contradicts Lemma 3!
- **Lemma 3**: If <v_1, v_2, …, v_k> is a path in the queue at any given time, then it is a simple path (contains no cycles).

66

Proof of Soundness

Assume that the search algorithm is ***not*** **sound**:

Let the returned path be $< v_0, v_1, \ldots, v_k >$

Then, one of the following must be `True`:

- 1. Returned path does not start with S:

  $v_k \neq S$
- 2. Returned path does not contain G at head:

  $v_0 \neq G$
- 3. Some transition in the returned path is not valid:

  $(v_{i-1}, v_i) \notin E$ for some $i \in \{1, 2, \ldots, v_k\}$
- 4. Returned path is not simple:

  $v_i = v_j$ for some $i, j \in \{0, 1, \ldots, k\}$ with $i \neq j$

67

Proof of Soundness

Assume that the search algorithm is ***not*** **sound**:

We reach a contradiction in all cases.

Hence, the simple search algorithm is ***sound***.

Proof of Completeness

Theorem 2 (Completeness):
Simple search algorithm is complete.

Need to prove:

- If there is a path to reach from S to G, then the algorithm returns one path that does so.

69

Pseudo Code For Simple Search

Let g be a Graph
S be the start vertex of g
G be the Goal vertex of g.
Q be a list of simple partial paths in GR,

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

70

A common technique in analysis of algorithms

- Let's slightly modify the algorithm
- We will analyze the modified algorithm.
- Then, "project" our results to the original algorithm.

Pseudo Code For Simple Search

Let g be a Graph G be the Goal vertex of g.
S be the start vertex of g Q be a list of simple partial paths in GR,

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. *// If head(N) = G, return N; (goal reached!)*
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

72

Proof of Completeness

- The modified algorithm terminates when the queue is empty.
- Let us prove a few lemmas regarding the behavior of the modified algorithm

Proof of Completeness

- **Lemma 1**: A path that is taken out of the queue is not placed into the queue again at a later step.
- **Proof**: *(using logical deduction)*
  - *Another way to state this*: "If p =$<v_0, v_1, \ldots, v_k>$ is a path that is taken out of the queue, then p = $<v_0, v_1, \ldots, v_k>$ is not placed in to the queue at a later step."
  - Assume that p = $<v_0, v_1, \ldots, v_k>$ is taken out of the queue.
  - Then, p must be placed in to the queue at an earlier step.
  - Then, v_0 must be in the visited list at this step.
  - Then, p = $<v_0, v_1, \ldots, v_k>$ can not placed in to the queue at a later step, since v_0 is in the visited list.

Proof of Completeness

- **Definition**: A vertex v is *reachable* from S, if there exists a path $<v_0, v_1, \ldots, v_k>$ that starts from S and ends at v, i.e., v_k = S and v_0 = v.
- **Lemma 2**: If a vertex v is reachable from S, then v is placed in to the visited list after a finite number of steps.

Proof of Completeness

- **Lemma 2**: If a vertex v is reachable from S, then v is placed in to the visited list after a finite number of steps.
- **Proof**: *(by contradiction)*
  - Assume v is reachable from S, *but* it is never placed on the visited list.
  - Since v is reachable from S, there exists a path that is of the form $<v_0, v_1, \ldots, v_k>$, where v_0 = v and v_k = S.
  - Let v_i be the first node (starting from v_k) in the chain that is never added to the visited list.
  - (1) Note that v_i was not in the visited list before this step.
  - (2) Note also that (v_{i+1}, v_i) is in *E*.
  - Since v_{i+1} was in the visited list, the queue included a path $<v_{i+1}, \ldots, v_k>$ (not necessarily the same as above), where v_k = S.
  - This path must have been popped from the queue, since there are only finitely many different partial paths and no path is added twice (by Lemma 1)and v_i was not in the visited list (see statement 1 above).
  - Since it is popped from the queue, then $<v_{i+1}, v_i, \ldots, v_k>$ must be placed in to queue (see statement 2 above) and v_i placed in to the visited list
  - Red statements contradict!

Proof of Completeness

- **Lemma 2**: If a vertex v is reachable from S, then v is placed in to the visited list after a finite number of steps.
- **Corollary**: In the modified algorithm, G is placed into the visited queue.
- *"Project" back to the original algorithm:*
  - This is exactly when the original algorithm terminates

Proof of Completeness

Theorem 2 (Completeness):
Simple search algorithm is complete.

- **Proof**: Follows from Lemma 2 evaluated in the original algorithm.

78

Pseudo Code For Simple Search

Let g be a Graph G be the Goal vertex of g.
S be the start vertex of g Q be a list of simple partial paths in GR,

1. Initialize Q with partial path (S) as only entry; set Visited = ( );
2. If Q is empty, fail. Else, pick some partial path N from Q;
3. If head(N) = G, return N; (goal reached!)
4. Else
   a) Remove N from Q;
   b) Find all children of head(N) (its neighbors in g) not in Visited and create a one-step extension of N to each child;
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited;
   e) Go to step 2.

79

Summarize Completeness and Soundness

- Hence, we have proven two theorems:

Theorem 1 (Soundness):
Simple search algorithm is sound.

Theorem 2 (Completeness):
Simple search algorithm is complete.

- Soundness and completeness is a requirement for most algorithms, although we will their relaxations quite often

Back to the Axiomatic Method

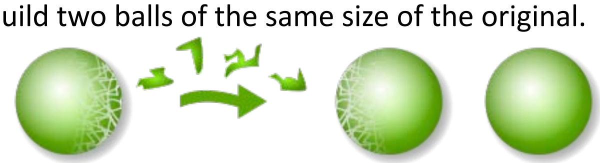
Does it really work?

- Essentially all of what we know in mathematics today can be derived from a handful of axioms called the **Z**armelo-**F**rankel set theory with the axiom of **C**hoice (ZFC).
- These axioms were made up by Zarmelo (they did not exist *a priori*, unlike physical phenomena).
- We do **not** know whether these axioms are *logically consistent*!
  - Sounds crazy! But, happened before… *Around 1900, B. Russell discovered that the axioms of that time were logically inconsistent, i.e., one could prove a contradiction.*

Back to the Axiomatic Method

Does it really work?

- ZFC axioms gives one what she/he wants:
  - **Theorem**: 5 + 5 = 10.
- However, absurd statements can also be driven:
  - **Theorem** (Banach-Tarski): A ball can be cut into a finite number of pieces and then the pieces can be rearranged to build two balls of the same size of the original.



Clearly, this contradicts our geometric intuition!

Back to the Axiomatic Method

Does it really work?

Images of Godel, Turing, and Einstein removed due to copyright restrictions.

Back to the Axiomatic Method

Does it really work?

On the fundamental limits of *mathematics*

- Godel showed in 1930 that there are some propositions that are true, but do not logically follow from the axioms.
- The axioms are not enough!
- But, Godel also showed that simply adding more axioms does not eliminate this problem. Any set of axioms that is not contradictory will have the same problem!
- Godel's results are directly related to computation. These results were later used by **Alan Turing** in 1950s to invent a revolutionary idea: ***computer***…

What you should know

- The definitions of a proposition, proof, theorem, lemma, and corollary.
- Proof techniques such as proof by contradiction, induction, invariance proofs.
- Notions of soundness and completeness.
- *Proving soundness and completeness of search algorithms*.