Introduction to Artificial Intelligence

LECTURE 2:

Complete Search VS. Heuristic Search

Overview

- Evaluation Functions
- Complete Search
- Heuristic Search
- Local Search
- Greediness
- Taboo Search

Hard Problems?

- Very large search spaces
- Hard-to-find/Ill-defined goals
- Complex, Hard/Soft (e.g. timetabling) constraints

- We need evaluation functions to help guide the search
  - Generally, eval(Si)$\approx$ distance(Si,G) (the distance from where we are to the nearest goal state)

Evaluation functions

- Sometimes, **eval** functions are obvious, because there's a clear reward/cost to operations:
  - When driving, mileage to goal
  - When shopping, euros to goal
- Sometimes, less so:
  - E.g., you need to get to a job interview in Grenoble within 2 hours; the direct route through Voiron is jammed up; you have about enough petrol for 130 kms; the speed limit is 110km/h (rain). What do you do?
- Given an *eval* function we can *guide* search
  - E.g., **greedy search**: apply that operator next which minimizes the *eval* function.

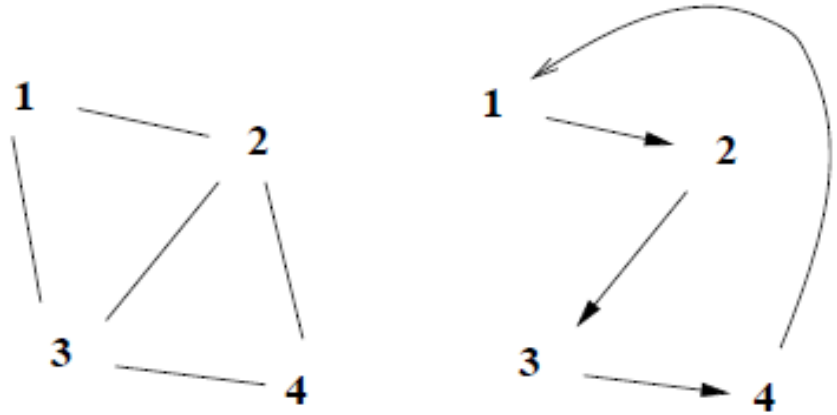
Traveling Salesman Problem

- Given a set of cities, and a list of roads connecting them, find a traversal which visits each city exactly once and which minimizes distance
  - Model?
  - Evaluation function?
  - State & Search space complexities?

Traveling Salesman Problem

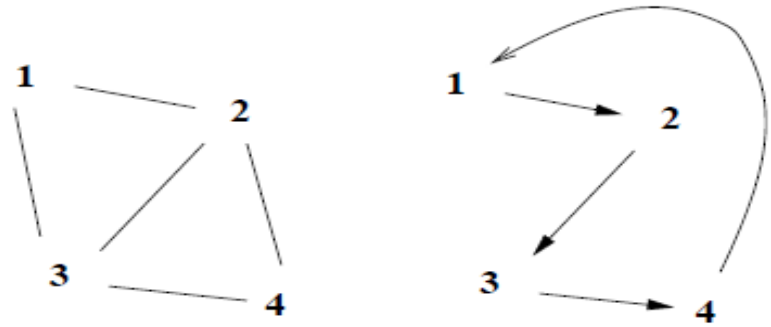
One (rather dumb) representation:

1. A DAG (chain) through all the cities
2. Complete by adding a link from last to first (hence, changing the DAG to cyclic digraph)



Implementation: (a) search through DAG space, eliminating all those DAGs which don't satisfy (1); (b) add last link to remaining digraphs; (c) minimize length among remaining rep's.

Traveling Salesman Problem



- Let states (possible answers) be permutations of cities.
  - Answer on right: (1 2 3 4), *meaning* the traversal 1-2-3-4-1.
  - NB: not all permutations are valid! (e.g., if there is no link between 2 and 3, 1-2-3-4 is invalid)

    # permutations: $n!$
- Edges are symmetric
  - E.g., (1 2 3 4) $\equiv$ (1 4 3 2)

    So, we get $n!/2$
- Starting city is irrelevant; there are n of them, so divide by n

This yields: $|\mathcal{S}| = n!/2n = (n-1)!/2$

This is still exponential, but *much* smaller than the dag search space!

Problem Formulation (TSP)

Let us consider a problem of size 4 (i.e. 4 cities)

- **Representation**: Permutation of natural numbers 1, 2, 3,4 where each number corresponds to a city to be visited in sequence
- **Search Space**: Permutations of all cities. Symmetric TSP, circuit the same regardless of the starting city: (n-1)!/2
- **Goal**: Minimize the total distance traversed, visiting each city once, and returning to the starting city.
- **Evaluation Function**: Map each tour to its corresponding total distance

K-SAT Problem

K-SAT = satisfiability problem with K Boolean variables.

Given $\mathcal{B}(A_1, \ldots, A_k)$, is there an assignment of truth values to the Boolean variables $A_1, \ldots, A_k$ s.t. $\mathcal{B}(A_1, \ldots, A_k)$ is true?

- Example: $A_1 \wedge A_2$ is satisfiable
- Example: $A_1 \rightarrow A_2$ is satisfiable
- Non-example: $A_1 \wedge \neg A_1$ is not satisfiable

5-SAT

- Given set of variables (A,B,C,D,E), find assignments that satisfy all clauses (or as many as possible)
  - AV¬BVC
  - ¬AVCVD
  - BVDV¬E
  - ¬CV¬DV¬E

- Model?
- Evaluation function?
- State & Search space complexities?

Problem Formulation (5-SAT)

- Representation
  - 1 True, 0 False, Binary String of length 5
- Search Space
  - 2 choices for each variable, taken over 5 variables, generates 2^5 possibilities
- Objective
  - To find the vector of bits such that the compound Boolean statement is satisfied (made true)
- Evaluation Function?
  - Not easy: the expression is either true or false; what could it mean to be near or far from the truth when the expression is just false?

Complete Search

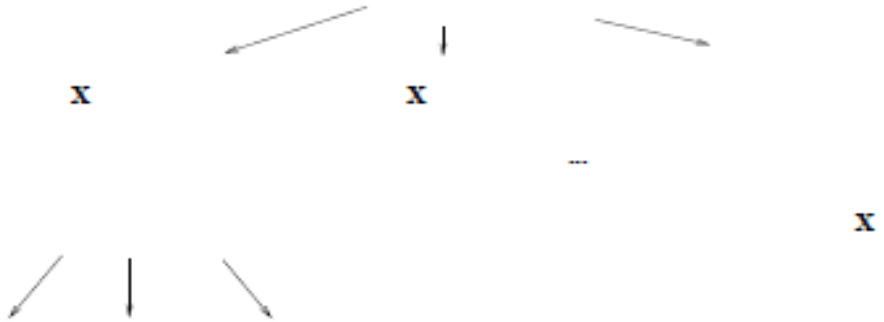
- **Exhaustive Search**
  - *Examines every state of the search space, for the optimum (given* cost*) or the goal state.*
- Exhaustive methods are worth considering because
  - Not all search spaces are as bad as TSP, etc.
  - They're simple and provide a basis for understanding improved techniques
- Some exhaustive search methods:
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Any other enumeration of the state space

Complete Search

- **Enumeration**
  - *Enumerating a set* X *means finding a 1-1 mapping from* X *onto* N
  - I.e., counting everything in X. This is equivalent to exhaustively identifying each element in X and, if X = S, exhaustive search
- Basic question: how to generate every possible state?

Enumeration for Tic-Tac-Toe

We can iteratively (recursively) generate each possible move and counter-move:



Each node in the search tree is a possible solution.

What are the branching factors at the first two levels?

Exercise *Determine the size of this search tree (in nodes).*

Enumerating TSP

- Given a starting city (we can dub it “1”), how to enumerate all the (n – 1)! potential traversals through the remainder? (Ignoring the symmetry issue.)
- A simple recursion TSPenum(l):
  - (i) IF length(l)=1 RETURN l;
  - (ii) FOR i =1 TO length(l)
  - (iii) RETURN append(l_i, TSPenum(l\l_i));
- E.g., if the input is l = (2 3), the output will be (2 3)(3 2). To these we prepend “1” to get two possible solutions, when n = 3.

Enumerating TSP

- This procedure will enumerate invalid solutions (i.e., two cities adjacent in the traversal, but not adjacent in the graph), as well as symmetrical duplicates. We can deal with that two ways:
    1. Ignore it; infeasible and duplicate possible solutions won't change the answer (but will slow down finding it!)
    2. Incorporate a call to two new functions, checking for infeasible and duplicate answers.
- The real problem with this is just the size of the search space! Enumeration is possible, but infeasible!

Enumerating SAT

Given n Boolean variables, there are 2^n possible truth-value assignments to them

(the number of bit strings of length n)

The most straightforward way of enumerating this search space is to generate a truth table for n variables. E.g.,

A	B	C	$\mathcal{B}(A, B, C)$
1	1	1	?
1	1	0	?
	...		...
0	0	0	?

Note: "Exhaustive search" need not always result in exhaustion: while generating this table, one would quit as soon as (if) $\mathcal{B}(A, B, C) = 1$, of course:)

Heuristic algorithm(s)

- So, we move on to cheaper methods.
- From Wikipedia:
  - *Heuristic is the art and science of discovery.*
- Comes from the same Greek root as “eureka”: “I find”.
- In CS the contrast is with *algorithmic* – trading accuracy for speed. In particular, the
  - invention of a heuristic evaluation function (that *approximates* the *cost* function)
  - its use in discovery/search

Heuristics

- Some heuristics for people:
  - If you are having difficulty understanding a problem, try drawing a picture.
  - If you can't solve a problem, try assuming you have a solution and seeing what you can derive from that, working backwards.
  - If the problem is abstract, try examining a concrete example…

Heuristic Search

Definition 1 (Heuristic Search) *Heuristic search uses a heuristic evaluation function $h(S)$ to speed up expected search time.*

Good eval functions:

- Are **admissible**; i.e., $\forall S\ h(S) \leq d(S, G)$ where $d(S, G)$ is the distance from S to the nearest goal state
- E.g.,
  - Minimizing path length: let h be crow-flight distance
  - Tic-Tac-Toe: let h be 3 - length of longest chain

  Why are these admissible?
- Ideal if $h(S) = d(S, G)$ for all S. Then h is an **oracle**! Usually, however, this can only be done in a complex way
  - E.g., by building exhaustive search into h, defeating the purpose of heuristic search!

Local Search

- Definition

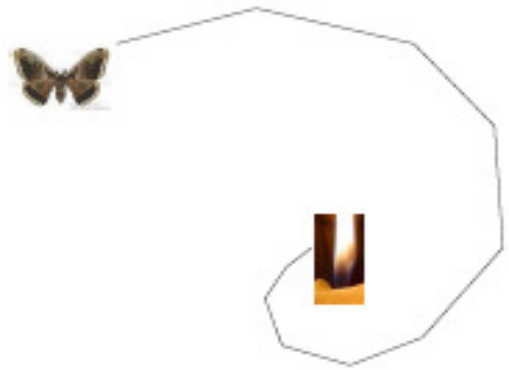
  Local search begins from an arbitrary state in the search space and looks for an improvement in the neighborhood of that state, until no improvement can be found. (This is a kind of iterative improvement*.)*
- Typically memoryless
- Improvement is defined in terms of *eval* - so, this is a kind of heuristic search
- Neighborhood can be defined in various ways

Local VS Global Optima

- Global solution is difficult to find
  - Consider a small subset of the search space
- Neighborhood
  - $N(S_i)$ is a set of all states that are **close to** S_i
- How to define closeness?
  - $N(S_i)=\{S_j: dist(S_i,S_j)\leq\varepsilon\}$ (dist is e.g. Euclidean)
  - Mapping on the search space
- Example for the SAT problem
  - 1-flip mapping
  - Consider the binary string 011001, neighbors?

Greediness

- Informed search VS uninformed search
- A possible greedy principle:
  - Begin with a random candidate
  - Incrementally find the best single improvement, one at a time.
- Greedy search is one which follows a local gradient



Greedy search

Here is a basic greedy search with h:

---

(1) $new \leftarrow randomstate$;

(2) LOOP: $current \leftarrow new$;

(3) FOR $i \leftarrow 1$ TO $maxtries$

(4) [$new \leftarrow min\ h(op(current))$;

(5) IF $h(new) < h(current)$

(6) THEN GOTO LOOP;]

(7) RETURN $current$;

GSAT

First convert to conjunctive normal forma (CNF):

$(A_1 \vee \neg A_4 \vee \ldots \vee A_k) \wedge \ldots \wedge (\neg A_m \vee \ldots)$

- The disjunctions $(A_1 \vee \neg A_4 \vee \ldots \vee A_k)$ are called clauses
- The disjuncts $A_1, \neg A_4$ are literals

GSAT

The GSAT algorithm:

---

(1) FOR $i \leftarrow 1$ TO $maxtries$

(2) [$T \leftarrow random.assignment$;

(3) FOR $j \leftarrow 1$ TO $maxflips$

(4) [IF $SAT(T, \mathcal{B})$ RETURN T

(5) ELSE $T \leftarrow bestflip(T)$;]]

---

bestflip chooses a single flip which results in the most clauses being satisfied.

GSAT

Example: $\mathcal{B}(A, B, C) = (A \rightarrow B) \rightarrow C$

Convert to CNF:

1. $\neg(A \rightarrow B) \vee C$
2. $\neg(\neg A \vee B) \vee C$
3. $(A \wedge \neg B) \vee C$
4. $(A \vee C) \wedge (\neg B \vee C)$

If the first assignment tried is $T = \langle 010 \rangle$, then *bestflip* will immediately return $T = \langle 011 \rangle$, since this satisfies both clauses.

> Nevertheless, this is a strange mix of random and local search; if the inner loop fails, a completely random restart is made.

2-opt Heuristic for TSP

"2-opt" is a simple means of solving TSP by local search:

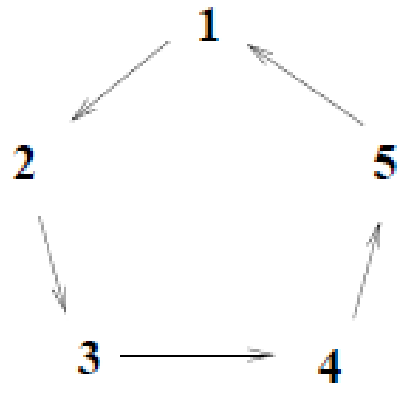
1. $T \leftarrow$random permutation of cities
2. for each $T' \in$ neighborhood(T), check whether $cost(T') < cost(T)$
   (a) if so, then $T \leftarrow T'$ and restart 2
3. return T

Neighborhood of T = permutations obtained by deleting any two edges and replacing with two others

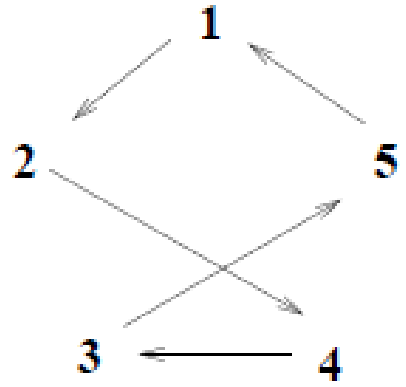
k-opt: any k edges get replaced

2-opt Heuristic for TSP

Example



(12345)



(12435)

(Note also the the arc $3 \rightarrow 4$ had to be flipped since this is a circuit of the cities.)

Partial Solutions

- Thus far search has meant:
  - Start somewhere in the space of candidate solutions.
  - Move through that space until done.
- A different approach to search is to begin with a *partial* solution and then complete it.
- Examples of Full vs. Partial solutions
  - 3-SAT: B(A,B,C)
    - <101>
    - <1**> (where * means ‘whatever’)
  - TSP
    - (12345)
    - 2-3, 5-1

Greediness revisited

- **A Different Greedy Principle:**
  - Begin with a null solution
  - Incrementally find the best single improvement, one *dimension* at a time
- Locality in the principle above is defined in terms of dimensions. Previously, locality was defined in terms of distance through n dimensions.

Greedy k-SAT

Assume $\mathcal{B}$ is in CNF

$$\text{clause}_1 \wedge \ldots \wedge \text{clause}_k$$

Let T represent the current partial instantiation. In each case it is initialized to $\langle * \ldots * \rangle$.

Greedy Search 1

- Set the next bit which (alone) maximizes the number of satisfied clauses.

Problem case:

$$\overline{x}_1 \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3)$$

- Setting $x_1 \leftarrow 1$ satisfies two clauses
- and makes $\mathcal{B}$ impossible to satisfy

Greedy k-SAT

Greedy Search 2

- Sort variables by frequency of occurence, least to most
- In that order, set the value which satisfies the most clauses

Example:

$$\overline{x}_1 \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3)$$

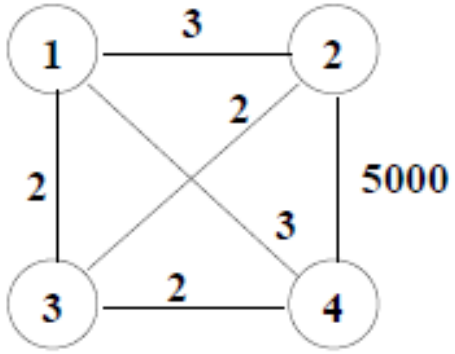
- Order: 231 (or, 321)
- Possible search:
  $\langle * * * \rangle$ $\langle *1* \rangle$ $\langle *11 \rangle$ $\langle 011 \rangle$

Greedy TSP Search

Greedy Search 1

1. current ←city 1
2. Loop: choose the cheapest link from current to any unconnected city. (Use random tiebreaks.)
3. Add the final link back to city 1.

This may work, but it's easy to construct counterexamples. E.g.,



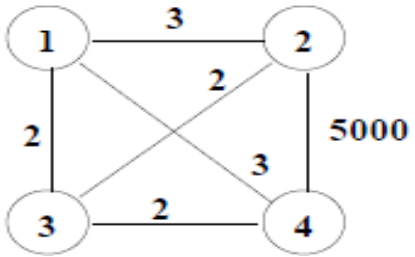
Greedy TSP Search

Greedy Search 2

1. Begin with the null solution.
2. Until each city has two edges selected, select the cheapest edge that connects two cities with (so far) one or no edges. (Use random tie-breaks.)

   (There is a special problem with this algorithm: if you get a circuit through a proper subset of cities, you are stuck. The algorithm needs a check for being stuck and an unsticking technique – simple and "dumb" would be to re-start.)

This may or may not work on the first example:



Greedy Search

- Advantages: simple, fast
- Drawback: getting stuck in local optima which can very quickly get you the wrong answer
- When could greedy search be guaranteed free of this problem?
  - K-step lookahead search: embed a 2^{nd}, 3^{rd} … k^{th} iterations, minimizing h(op(…(op(current_state))…)), before evaluating new
  - Group/parallel search:
    - States are sampled from space with equal probabilities
    - Explore the search space thoroughly
    - But foregoes exploiting promising regions
  - Stochastic techniques: simulated annealing, evolutionary algorithms…

Locality Tradeoff

With very small neighborhoods:

- Neighborhood can be rapidly searched for improvements
- Local search will rapidly get stuck in local optima

With very large neighborhoods:

- "Local" search will avoid getting stuck in local optima
- "Neighborhoods" will take a long time to search

A natural solution suggesting itself is. . . ?

Taboo Search

- Uses temporary memory to forbid (re-)examining parts of the search space (they are ‘taboo’)
  - Random starting state
  - Thereafter deterministic: best state in allowed neighbourhood is accepted
  - Not exhaustive
- SAT example
  - Suppose there are 8 Boolean variables in CNF form of $B(X_i)$: <0,1,1,1,0,0,0,0>
  - We search locally: look at single bit flips and accept that flip which maximises the number of satised clauses
  - This gives us a local neighbourhood of size eight (each element of which corresponds to a 1-bit flip in the above, earlier, assignment)
    - <1,1,1,1,0,0,0,0> …
    - <0,1,1,1,0,0,0,1>
  - But, we refuse to flip any bit for 5 steps after it has last been flipped.
    - This reduces the effective neighbourhood size and directs the search
    - Use of a memory vector storing the number of steps since a bit was last flipped

Taboo Search Example

Suppose the search proceeds by flipping bits: 3, 7, 1, 6, 4.

Then we have:

1. $\langle 0,1,1,1,0,0,0,0\rangle$; $M = \langle 0,0,0,0,0,0,0,0\rangle$

   Start flips

2. $\langle 0,1,0,1,0,0,0,0\rangle$; $M = \langle 0,0,5,0,0,0,0,0\rangle$
3. $\langle 0,1,0,1,0,0,1,0\rangle$; $M = \langle 0,0,4,0,0,0,5,0\rangle$
4. $\langle 1,1,0,1,0,0,1,0\rangle$; $M = \langle 5,0,3,0,0,0,4,0\rangle$
5. $\langle 1,1,0,1,0,1,1,0\rangle$; $M = \langle 4,0,2,0,0,5,3,0\rangle$
6. $\langle 1,1,0,0,0,1,1,0\rangle$; $M = \langle 3,0,1,5,0,4,2,0\rangle$

From here onwards, the available neighbourhood is of size three (because the most recently flipped 5 bits can not be flipped on the subsequent step):

- $\langle 1,0,0,0,0,1,1,0\rangle$
- $\langle 1,1,0,0,1,1,1,0\rangle$
- $\langle 1,1,0,0,0,1,1,1\rangle$

Taboo Acceptance

- The selection of the next flip from the available neighbourhood is greedy
  - By taking the truth assignment that maximises clause satisfactions
- But taboo search itself is not greedy: no comparison with current value
  - So, current value, even if locally optimal, can be left behind
  - Further, we won't return immediately because of the taboo memory
- Idea: Relaxing Taboo Acceptance
  - **Aspiration Criterion:** If some of the excluded local neighbourhood satises our aspirations, abandon the taboos
  - E.g., if the taboo is put upon the global optimum - such as an instantiation that satisfies B - then this would not be good and the taboo would be best abandoned.