22c:145 Artificial Intelligence

Local Search

Readings Textbook: Chapter 4:1 and 6:4

1

So far:
methods that systematically explore the search space, possibly using principled pruning (e.g., A*)

Current best such algorithm can handle search spaces of up to 10^{100} states / around 500 binary variables ("ballpark" number only!)

What if we have much larger search spaces?

Search spaces for some real-world problems may be much larger e.g. $10^{30,000}$ states as in certain reasoning and planning tasks.

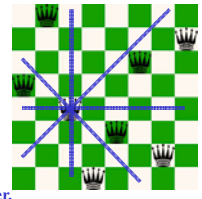
A completely different kind of method is called:

Local search
(sometimes called: Iterative Improvement Methods)

2

Intro example: N-queens

Problem: Place N queens on an NxN chess board so that no queen attacks another.



Example solution for N = 8.

How hard is it to find such solutions? What if N gets larger?

Can be formulated as a search problem.
Start with empty board. [Ops? How many?]
Operators: place queen on location (i,j). [N^2. Goal?]
Goal state: N queens on board. No-one attacks another.

N=8, branching 64. Solution at what depth?
N. Search: (N^2)^N Informed search? Ideas for a heuristic?

Issues: (1) We don't know much about the goal state. That's what we are looking for!
(2) Also, we don't care about path to solution!

What algorithm would you write to solve this?

3

Local Search: General Principle

Key idea (surprisingly simple):

1) Select (random) initial state (initial guess at solution)
e.g. guess random placement of N queens

2) Make local modification to improve current state
e.g. move queen under attack to "less attacked" square

3) Repeat Step 2 until goal state found (or out of time)
cycle can be done billions of times

Unsolvable if out of time?

Requirements:
- generate an initial guess (often random; probably-not-optimal or even valid)
- evaluate quality of guess
- move to other state (well-defined neighborhood function)

Not necessarily!
Method is incomplete.

. . . and do these operations quickly
. . . and don't save paths followed

4

Local Search

1) Hill-climbing search or greedy local search
2) Simulated annealing
3) Local beam search
4) Tabu search

5

Hill-climbing search

"Like climbing Everest in thick fog with amnesia"

Keep trying to move to a better "neighbor", using some quantity to optimize.

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node
  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
```

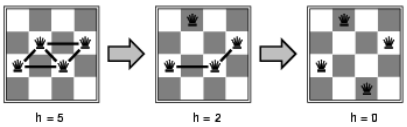
Note: (1) "successor" normally called neighbor.
(2) minimization, isomorphic.
(3) stops when no improvement but often better to just "keep going", especially if improvement = 0

6

4-Queens

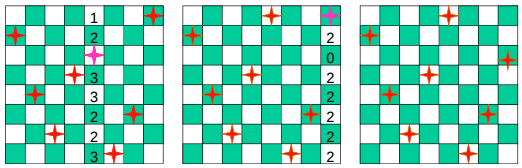
States: 4 queens in 4 columns (256 states)
Neighborhood Operators: move queen in column
Evaluation / Optimization function: h(n) = number of attacks / "conflicts"
Goal test: no attacks, i.e., h(G) = 0

Initial state (guess).



Local search: Because we only consider local changes to the state at each step. We generally make sure that series of local changes can reach all possible states.

8-Queens



Representation: 8 integer variables giving positions of 8 queens in columns (e.g. <2, 5, 7, 4, 3, 8, 6, 1>)

Section 6.4 Textbook ("hill-climbing with min-conflict heuristics")
Pick initial complete assignment (at random)
Repeat
- **Pick a conflicted variable var (at random)**
- **Set the new value of var to minimize the number of conflicts**
- **If the new assignment is not conflicting then return it**

(Min-conflicts heuristics)

Remarks

Local search with min-conflict heuristic works extremely well for N-queen problems. Can do millions and up in seconds. Similarly, for many other problems (planning, scheduling, circuit layout etc.)

Why?

Commonly given: Solns. are densely distributed in the $O(n^n)$ space; on average a solution is a few steps away from a randomly picked assignment. But, solutions still exponentially rare!

In fact, density of solutions not very relevant. Even problems with a single solution can be "easy" for local search!

It all depends on the structure of the search space and the guidance for the local moves provided by the optimization criterion.

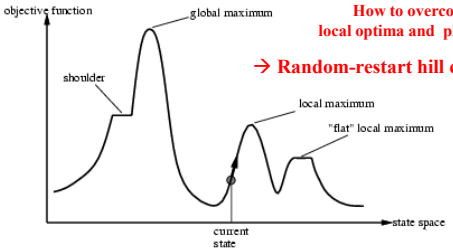
For N-queens, consider h(n) = k, if k queens are attacked. Does this still give a valid solution? Does it work as well?

What happens if h(n) = 0 if no queen under attack; h(n) = 1 otherwise? Does this still give a valid solution? Does it work as well?

"Blind" search! No gradient in optimization criterion!

Issues for hill-climbing search

Problem: depending on initial state, can get stuck in local optimum (here maximum)



How to overcome local optima and plateaus ?

→ Random-restart hill climbing

But, 1D figure is deceptive. True local optima are surprisingly rare in high-dimensional spaces! There often is an escape to a better state.

Potential Issues with Hill Climbing / Greedy Local Search

Local Optima: No neighbor is better, but not at global optimum.
- **May have to move away from goal to find (best) solution.**
- **But again, true local optima are rare in many high-dimensional spaces.**

Plateaus: All neighbors look the same.
- **8-puzzle: perhaps no action will change # of tiles out of place.**
- **Soln. just keep moving around! (will often find some improving move eventually)**

Ridges: sequence of local maxima

May not know global optimum: Am I done?

Improvements to Greedy / Hill-climbing Search

Issue:
- **How to move more quickly to successively better plateaus?**
- **Avoid "getting stuck" / local maxima?**

Idea: Introduce "noise:"
downhill (uphill) moves to escape from plateaus or local maxima (mimima)
E.g., make a move that increases the number of attacking pairs.

Noise strategies:

Simulated Annealing
- **Kirkpatrick et al. 1982; Metropolis et al. 1953**

Simulated Annealing

Idea:
Use conventional hill-climbing style techniques, but occasionally take a step in a direction other than that in which there is improvement (downhill moves; away from solution).

As time passes, the probability that a down-hill step is taken is gradually reduced and the size of any down-hill step taken is decreased.

13

Simulated annealing search (one of the most widely used optimization methods)

What's the probability when: T → inf?

What's the probability when: T → 0?

What's the probability when: ΔE=0? (sideways / plateau move)

What's the probability when: ΔE→-∞?

Idea: escape local maxima by allowing some "bad" moves but gradually decrease frequency of such moves.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

Similar to hill climbing, but a random move instead of best move

case of improvement, make the move

Otherwise, choose the move with probability that decreases exponentially with the "badness" of the move.

14

Notes

Noise model based on statistical mechanics
- . . . introduced as analogue to physical process of steel annealing.

Convergence:
1. With exponential schedule, will provably converge to global optimum
   One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
2. Few more precise convergence rate.
   (Recent work on rapidly mixing Markov chains.
   Surprisingly deep foundations.)

Key aspect: downwards / sideways moves
- Expensive, but (if have enough time) can be best

Hundreds of papers / year; original paper one of most cited papers in CS!
- Many applications: VLSI layout, factory scheduling, protein folding. . .

15

Simulated Annealing (SA) --- Foundations

Superficially: SA is local search with some noise added. Noise starts high and is slowly decreased.

True story is much more principled:

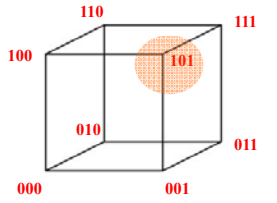
SA is a general sampling strategy to sample from a combinatorial space according to a well-defined probability distribution.

Sampling strategy models the way physical systems, such as gases, sample from their statistical equilibrium distributions. Order 10^23 particles. Studied in the field of statistical physics.

We will sketch the core idea.

16

Example: 3D Hypercube space



States		Value f(s)
s1	000	2
s2	001	4.25
s3	010	4
s4	011	3
s5	100	2.5
s6	101	4.5
s7	110	3
s8	111	3.5

N dimensional "hypercube" space G = (V, E), where V = { N-bit binary numbers}, E = { (x, y) | x and y differ by 1 bit }. N =3. 2^3 = 8 states total.

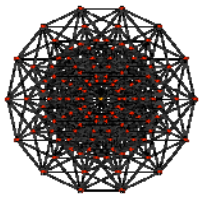
Goal: Optimize f(s), the value function. Maximum value 4.5 in s6.

Use local search: Each state / node has N = 3 neighbors (out of 2^N total).

17

Of course, real interest in large N…

Spaces with 2^N states and each state with N neighbors.



7D hypercube; 128 states.
Every node, connected to 7 others.
Max distance between two nodes: 7.

9D hypercube; 512 states.

Practical reasoning problem: N = 1,000,000. 2^N = 10^300,000

18

SA node sampling strategy

Consider the following "random walker" in hypercube space:

1) **Start at a random node S (the "current node").**
   (How do we generate such a node?)

2) **Select, at random, one of the N neighbors of S, call it S'**

3) **If (f(S') – f(S)) > 0, move to S', i.e. set S := S'**
   (i.e., jump to node with better value)
   else with probability e^(f(S')-f(S))/T move to S', i.e., set S := S'

4) **Go back to 2)**

Note: Walker keeps going and going. Does not get stuck in any one node.

19

Central Claim --- Equilibrium Distribution:

After "a while," we will find the walker in state S with probability

Prob(S) = e^(f(S)/T) / Z

where Z is a normalization constant (function of T) to make sure the probabilities over all states add up to 1.

Z is called the "partition function" and is given by

$$Z = \sum e^{(f(x))/T}$$

where the sum is over all 2^N states x. So, an exponential sum! Very hard to compute but we generally don't have to do!

20

Prob(s) = e^(f(s))/T / Z For our example space

States	Value f(s)	T=1.0	Prob(s)	T=0.5	Prob(s)	T=0.25	Prob(s)
s1 000	2	7.4	0.02	55	2981	0.00	30.000
s2 001	4.25	70.1	0.23	2,4	154,952	0.27	0.24
s3 010	4	54.6	0.18	2,9	86,111	0.17	0.09
s4 011	3	20.1	0.07	40	2,755	0.02	0.001
s5 100	2.5	12.2	0.04	1,4	2,026	0.00	80.008
s6 101	4.5	90.0	0.29	68,1	59,969	0.45	0.65
s7 110	3	20.1	0.07	40	2,755	0.02	0.001
s8 111	3.5	33.1	0.11	1,2	02,604	0.06	0.06
		sum Z = 307.9		sum Z = 18	0,054,153		

So, at T = 1.0, walker will spend roughly 29% of its time in the best state.
T = 0.5, roughly 45% of its time in the best state.
T = 0.25, roughly 65% of its time in the best state.
And, remaining time mostly in s2 (2nd best)!

21

So, when T gets lowered, the probability distribution starts to "concentrate" on the maximum (and close to maximum) value states.

The lower T, the stronger the effect!

What about T high? What is Z and Prob(S)?

2^N and 1/(2^N) because e^0 =1 in each row

At low T, we can just output the current state. It will quite likely be a maximum value (or close to it) state. In practice: Keep track of best state seen during the SA search.

SA is an example of so-called Markov Chain Monte Carlo or MCMC sampling.

It's very general technique to sample from complex probability distributions by making local moves only. For optimization, we chose a clever probability distribution that *concentrates on the optimum states for low T.* (Kirkpatrick et al. 1984)

22

Some final notes on SA:

1) **"Claim Equilibrium Distribution" needs proof. Not too difficult but takes a bit of background about Markov Chains. It's beautiful and useful theory.**
2) **How long should we run at each T? Technically, till the process reaches the stationary distribution. Here's the catch: may take exponential time in the worst case. ☹**
3) **How quickly should we "cool down"? Various schedules in literature.**
4) **To get (near-)optimum, you generally can run much shorter than needed for full stationary distribution.**
5) **Keep track of best solution seen so far.**
6) **A few formal convergence rate results exists, including some poly-time results ("rapidly mixing Markov chains").**
7) **Many variations on basic SA exist, useful for different applications.**

23

Local beam search

- **Start with *k* randomly generated states**
- **Keep track of *k* states rather than just one**
- **At each iteration, all the successors of all *k* states are generated**
- **If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.**

Equivalent to k random-restart hill-climbing?

No: Different since information is shared between k search points:

Some search points may contribute none to best successors: one search point may contribute all k successors "Come over here, the grass is greener" (Textbook)

24

Tabu Search

Tabu – socially or culturally proscribed: forbidden to be used, mentioned, or approached because of social or cultural rather than legal prohibitions. (http://encarta.msn.com/dictionary_1861698691/taboo.html)

Glover, F. 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*. Vol. 13, pp. 533-549.

25

Overview of Tabu Search

Tabu search is based on introducing ***flexible memory*** structures in conjunction with ***strategic restrictions*** and ***aspiration levels*** as a means for exploiting search spaces in local search.

Meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimum by use of a ***Tabu list***.

26

Overview of Tabu Search

Used to solve combinatorial (finite solution set) optimization problems

A dynamic neighborhood search method

Use of a flexible memory to restrict the next solution choice to some subset of neighborhood of current solution

27

Tabu Search Strategy

3 main strategies:

- **Forbidding strategy**: control what enters the tabu list
- **Freeing strategy**: control what exits the tabu list and when
- **Short-term strategy**: manage interplay between the forbidding strategy and freeing strategy to select trial solutions

28

Parameters of Tabu Search

Local search procedure
Neighborhood structure
Aspiration conditions
Form of tabu moves
Addition of a tabu move
Maximum size of tabu list
Stopping rule

29

Basic Ingredients of Tabu Search

A chief way to exploit memory in tabu search is to classify a subset of the moves in a neighborhood as forbidden (or ***tabu***).

A ***neighborhood*** is constructed to identify adjacent solutions that can be reached from current solution.

The classification depends on the history of the search, and particularly on the recency or frequency that certain move or solution components, called ***attributes***, have participated in generating past solutions.

A ***tabu list*** records forbidden moves, which are referred to as ***tabu moves***.

Tabu restrictions are subject to an important exception. When a tabu move has a sufficiently attractive evaluation where it would result in a solution better than any visited so far, then its tabu classification may be overridden. A condition that allows such an override to occur is called an ***aspiration criterion***.

30

Basic Tabu Search Algorithm

Step 1: Choose an initial solution i in S. Set i* = i and k=0.

Step 2: Set k=k+1 and generate a subset V of solution in N(i,k) such that none of the Tabu conditions is violated and at least one of the aspiration conditions holds.

Step 3: Choose a best j in V and set i = j.

Step 4: If f(i) < f(i*) then set i* = i.

Step 5: Update Tabu List and aspiration conditions.

Step 6: If a stopping condition is met then stop. Else go to Step 2.

Tabu Search Stopping Conditions

Some immediate stopping conditions could be the following:

1. N(i, K+1) = 0. (no feasible solution in the neighborhood of solution i)
2. K is larger than the maximum number of iterations allowed.
3. The number of iterations since the last improvement of i* is larger than a specified number.
4. Evidence can be given than an optimum solution has been obtained.

Stopping criterion can, for example, use a fixed number of iterations, a fixed amount of CPU time, or a fixed number of consecutive iterations without an improvement in the best objective function value. Also stop at any iteration where there are no feasible moves into the local neighborhood of the current trial solution.

Use of Short-Tem Memory

- To avoid reversal of moves and cycling
- The most common implementation of the STM is based on move attributes and the recentness of the moves
- Variable tabu list size
  - For a constant size tabu list
    - Too long: deteriorate the search results
    - Too short: cannot effectively prevent from cycling
- Intensification of the search
  - Decrease the tabu list size
- Diversification of the search
  - Increase the tabu list size
  - Penalize the frequent move or unsatisfied constraints

Example 1

After a move that changes the value of x_i from 0 to 1, we would like to prevent x_i from taking the value of 0 in the next iterations

- Attribute to record: *i*
- Tabu activation rule: move ($x_i \leftarrow 0$) is tabu if *i* is tabu-active

Example 2

After a move that exchanges the positions of element *i* and *j* in a sequence, we would like to prevent elements *i* and *j* from exchanging positions in the next iterations

- Attributes to record: *i* and *j*
- Tabu activation rule: move ($i \leftrightarrow j$) is tabu if both *i* and *j* are tabu-active

Example 3

After a move that drops element *i* from and adds element *j* to the current solution, we would like to prevent element *i* from being added to the solution and prevent element *j* from being dropped from the solution in the next iterations

- Attributes to record: *i* and *j*
- Tabu activation rules:
  - move (Add *i*) is tabu if *i* is tabu-active
  - move (Drop *j*) is tabu if *j* is tabu-active

Tabu or not Tabu

For effective use of memory, only moves can be tabu. Attributes are never tabu, they can only be tabu-active

A move may be tabu if it contains one or more tabu-active attributes

The classification of a move (as tabu or not tabu) is determined by the tabu-activation rules which are problem-specific.

37

Aspiration criteria

The criteria for overruling the tabu constraints and differentiating the preference of among the neighbors. E.g.,

By Objective

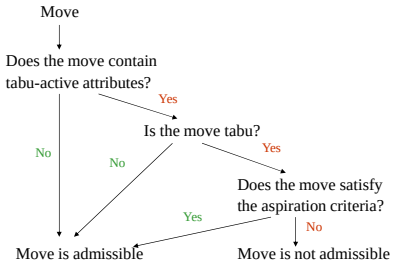
A tabu move becomes admissible if it yields a solution that is better than an aspiration value

By Search Direction

A tabu move becomes admissible if the direction of the search (improving or non-improving) does not change

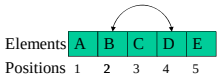
38

Tabu Decision Tree

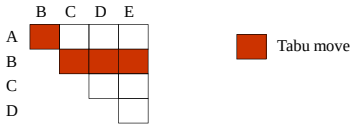


39

Example 4

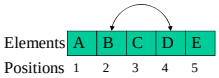


Tabu activation rule: move (B ↔ *) is tabu

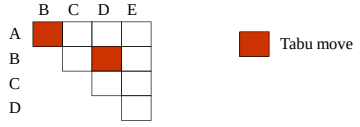


40

Example 5

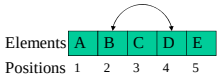


Tabu activation rule: move (B ↔ *) is tabu if B moves a distance of 2 or less

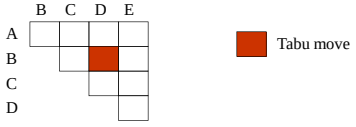


41

Example 6



Tabu activation rule: move (B ↔ D) is tabu



42

Pros and Cons

Pros:

- Allows non-improving solution to be accepted in order to escape from a local optimum
- The use of Tabu list
- Can be applied to both discrete and continuous solution spaces
- For larger and more difficult problems (scheduling, quadratic assignment and vehicle routing), tabu search obtains solutions that rival and often surpass the best solutions previously found by other approaches.

Cons:

- Too many parameters to be determined
- Number of iterations could be very large
- Global optimum may not be found, depends on parameter settings

43

Summary

Local search algorithms

- **Hill-climbing search**
- **Local beam search**
- **Simulated annealing search**
- **Tabu search**

44

1) Surprisingly efficient search technique
2) Often the only feasible approach
3) Wide range of applications
4) Formal properties / guarantees still difficult to obtain
5) Intuitive explanation:

- **Search spaces are too large for systematic search anyway. . .**

6) Area will most likely continue to thrive

45