Advanced Search Algorithms

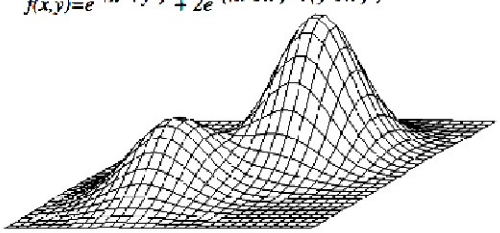
CSE 415: Introduction to Artificial Intelligence
University of Washington
Spring, 2017

Outline

Hill Climbing
Simulated Annealing
Genetic Search
Case-Based Reasoning

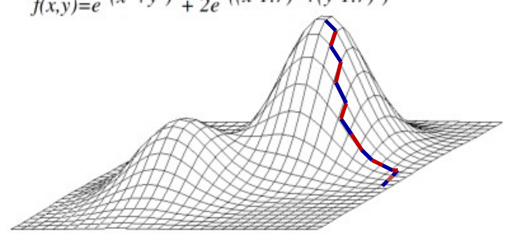
Hill Climbing

$$f(x,y)=e^{-(x^2+y^2)}+2e^{-((x-1.7)^2+(y-1.7)^2)}$$



Hill Climbing (cont.)

$$f(x,y)=e^{-(x^2+y^2)}+2e^{-((x-1.7)^2+(y-1.7)^2)}$$



Hill Climbing (cont.)

Assumptions:
Each state maps to a well-defined "height."

Method:
At each step, choose the move that results in the state having the greatest height.

Similar to:
Greedy algorithms.
Gradient ascent or descent or steepest ascent or descent (in continuous state spaces)

Hill Climbing (cont.)

Major limitation:

Can get stuck in a local optimum (e.g., a lesser peak).

Simulated Annealing

Like probabilistic hill climbing.
Allows for the possibility of escaping from local optima.

Optimum means “lowest potential energy” state.

S.A. is based on an analogy to a metallurgical process called annealing.

S.A. (cont.)

Problem structure for simulated annealing:

We have an energy function

$$E: S \rightarrow \Re^+$$

that assigns to each state a nonnegative real number.

S.A. – The Method

In state s having energy z, randomly select an operator whose precondition is satisfied.
Apply it to create a state s' having energy z'. If $z' < z$, then accept s' as the new current state.
But if $z' > z$, randomly choose to accept s' with probability p, where

$$p = e^{-(z'-z)/kT}$$

T is the “temperature” which starts high and gradually is reduced to 0.

S.A. Typical Problem Structure

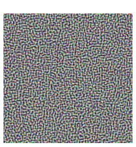
Problem structure for simulated annealing:

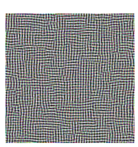
Often: the state space is a cartesian product of many subspaces each of which corresponds to a state variable.

S.A. Example

“Crystallization” in a digital image.
Each pixel corresponds to a separate state variable.
Start with a random image. Use an energy function that gives low energy to similar pixels being adjacent and low energy when pixels at a slight distance are different.

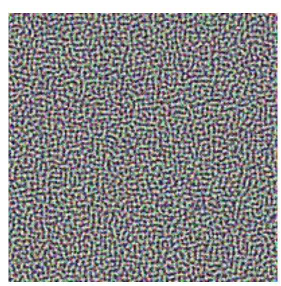
http://en.wikipedia.org/wiki/Simulated_annealing
Results using fast and slow cooling schedules:



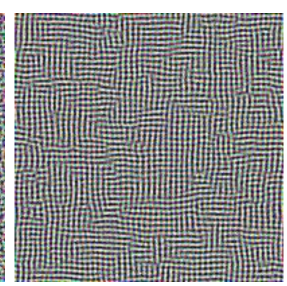


S.A. Example (enlargements)

Fast



Slow

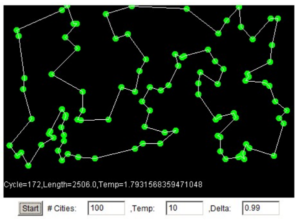


Ss
State-space Search

S. A. Example 2

Travelling Salesman Problem
Demonstration applet at:

http://www.heatonresearch.com/articles/64/page1.html



Ss
State-space Search

Genetic Search

Employ probabilistic state changes to help escape from local minima or maxima.

Combine strengths from multiple candidate solutions.

Ss
State-space Search

Motivation

1. Use nature as a guide. (Mutation plus mitotic reproduction, and Darwin's principle of natural selection.)

2. Take advantage of parallel processing. (Allow an entire population -- maybe millions or billions -- to evolve.)

3. Use randomness to escape from local maxima of the fitness function.

Ss
State-space Search

The Evolutionary Model

Goal: To produce an individual having certain characteristics. (A "most fit" individual.)

Method: Create a diverse population of individuals. Provide a means for adding to the population. Provide a means to weed out less fit individuals. Let the population evolve.

Mutation: Random small change to the genetic blueprint for an individual.

Crossover: Formation of a genetic blueprint for a new individual by splicing together a piece from individual A with a piece from individual B.

Fitness function: A function that maps each individual to a scalar fitness value.

Ss
State-space Search

Mutation and Crossover

Mutation: Makes a move or a perhaps a sequence of moves in the state space. If the direction of moving is random, there is a good chance to escape from local maxima.
Type 1: A random modification of one atomic unit -- one gene.
Type 2: A reordering of genes, .e.g, a transposition of two genes.

Crossover: Formation of a genetic blueprint for a new individual by splicing together a piece from individual A with a piece from individual B.

If A and B are each very fit, but in different ways, perhaps crossover will produce an even more fit individual.

Ss
State-space Search

G.S. Example: A Travelling Salesman Problem

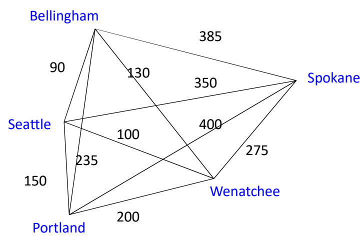
Find a shortest tour for the cities:
Seattle, Bellingham, Spokane, Wenatchee, Portland.

Given: A matrix of intercity distances for these cities.

Tour: a Hamiltonian circuit -- a closed path that starts and ends at one city and visits every other city exactly once.

Our assumption: It's possible to go from any city A to another city B without stopping (visiting) any other city C.

The Map



Problem Representation

Each individual: A list of 5 cities, repetitions permitted.
e.g., (SEATTLE, SEATTLE, SPOKANE, SPOKANE, SPOKANE)

Fitness function: $f(x) = \dfrac{10000}{2\, f_1(x) + 50\, f_2(x)}$

Path cost: $f_1(x) = \sum_{i=0}^{4} \mathrm{dist}(x[i], x[i+1 \bmod 5])$

Non-tour penalty:
$f_2(x) = 100\, (\,|\,\mathrm{Cities} - \mathrm{cities}(x)\,| + |\,\mathrm{cities}(x) - \mathrm{Cities}\,|\,)$
where "-" denotes set difference and |s| denotes cardinality.

Representation for Individuals

[*citySequence*, *strengthValue*]

An initial-state individual:

```
[['Seattle', 'Seattle', 'Seattle', 'Seattle', 'Seattle'], 0.5]
```

A goal-state individual:

```
[['Seattle', 'Bellingham', 'Spokane', 'Wenatchee', 'Portland'],
4.8]
```


MUTATE1

```
def mutate1(individual):
    where = choice(range(NCITIES))
    newCity = choice(CITIES)
    newIndiv = [individual[0][:], individual[1]]
    newIndiv[0][where]=newCity
    return newIndiv
```


MUTATE2

```
def mutate2(individual):
    where1 = choice(range(NCITIES))
    where2 = choice(range(NCITIES))
    city1 = individual[0][where1]
    city2 = individual[0][where2]
    newIndiv = [individual[0][:], individual[1]]
    newIndiv[0][where1]=city2
    newIndiv[0][where2]=city1
    return newIndiv
```


CROSSOVER

```
# In this function, individual1 and individual2
# are the paths, without the strength values.
# In theory there are two new individual produced,
# but we are only returning one of them in this
# implementation.

def crossover(individual1, individual2):
    where = choice(range(NCITIES))
    newIndiv1 = individual1[:where] +\
                individual2[where:]
    #newIndiv2 = individual2[:where] +\
    #           individual1[where:]
    return newIndiv1
```

EVOLVE

```
def evolve(ngenerations, nmutations, ncrossovers):
    global POPULATION; global INITIAL_POPULATION;
    POPULATION = INITIAL_POPULATION
    for i in range(ngenerations):
        for j in range(nmutations):
            mutant = mutate(choice(POPULATION))
            print 'mutant is ' + str(mutant)
            addIndividual(mutant)
        for j in range(ncrossovers):
            theCross = crossover(choice(POPULATION),\
                                 choice(POPULATION))
            print 'theCross is ' + str(theCross)
            addIndividual(theCross)
        print 'In generation ' + str(i) +\
          ', the population is: '
        print str(POPULATION)
```


Design Issues

Individual needs to be represented as a linear string or list to be amenable to crossovers.

Fitness function must permit some diversity in the population in order to avoid getting stuck in local maxima.

One could allow the mutation mechanisms and fitness functions to change during a run so that greater diversity is permitted near the beginning, but the search can "tighten up" later. (compare to simulated annealing).

Randomness vs deterministic choices.

Can genetic search be forced to systematically cover the entire state space?

Case-Based Reasoning

Outline:

Motivation.
System diagram
Brief description of the menu planning program

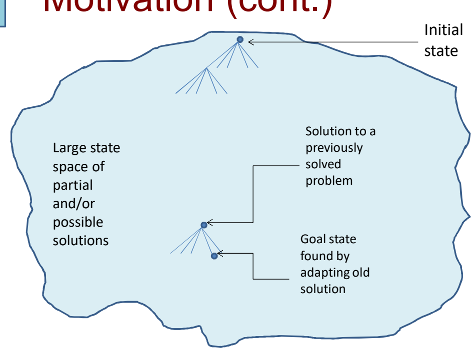
Motivation for CBR

Problem solving knowledge is often organized around a collection of previously solved problems ("cases")

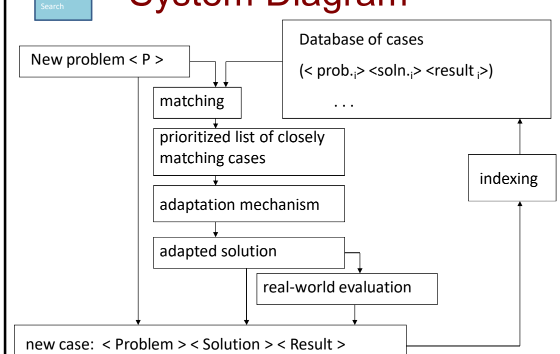
Certain types of problem solving lend themselves naturally to a case-based approach:
e.g., law, medicine, business administration

In a large and complicated state space, it would help if we can begin the search near a goal state.

Motivation (cont.)



System Diagram



Brief Introduction to the Menu Planning Program Using CBR

Problem representation:
Style of cuisine;
Number of diners;
list of dietary restrictions;
Desired price per person

Solution representation:
Main dish;
Appetizer
Dessert

Result representation:
Success (S) or failure (F)

Problem Representation: Details

```
['example',
 ['problem',[
    ['cuisine',  'mexican'],
    ['ndiners', 6],
    ['diet-restrictions', 'no-gluten'],
    ['cost-per-person', 3]]],
 ['solution',[
    ['main-course', 'tacos'],
    ['appetizer', 'chips-n-salsa'],
    ['dessert', 'fruit']]],
 ['result', 's']
]
```


Representing Dietary Information

['food-fact', 'gravlax', 'no-meat']
['food-fact', 'mussels', 'no-meat']
['food-fact', 'mussels', 'no-sugar']
['food-fact', 'chips-n-salsa', 'no-meat']
. . .

Matching a Pair of Problems

Distance metric $d: P \times P \rightarrow R$

$d(x, y) \geq 0$

$d(x, x) = 0$
$d(x, y) = d(y, x)$
$d(x, z) \leq d(x, y) + d(y, z)$

Adaptation Functions

Operators intended to transform a solution of one problem into a solution to another problem.

E.g., replace meat by beans (to satisfy a no-meat restriction).
Replace seafood by artichokes (to satisfy a no-seafood restriction).
Multiply all ingredient quantities by n_2/n_1 (to adapt a recipe to the desired number of servings).

Summary of Strategies

State-space search is simple in principle but often difficult in practice, because of large state spaces and local optima.
Hill climbing is a good strategy for unimodal (convex or concave) objective functions.
Simulated annealing adapts the hill-climbing strategy for more complex objective functions.
Genetic search maintains a whole population of "current states" and can use crossovers as well as mutations as operators.
Case-based reasoning jump-starts the search with a new starting state relatively close to the goal.