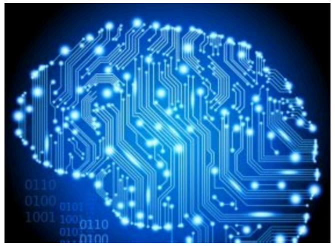
Lecture 5: Search 4

http://cs.nju.edu.cn/yuy/course_ai16.ashx



Previously...

Path-based search

- Uninformed search
  - Depth-first, breadth first, uniform-cost search
- Informed search
  - Best-first, **A* search**

Adversarial search

- Alpha-Beta search

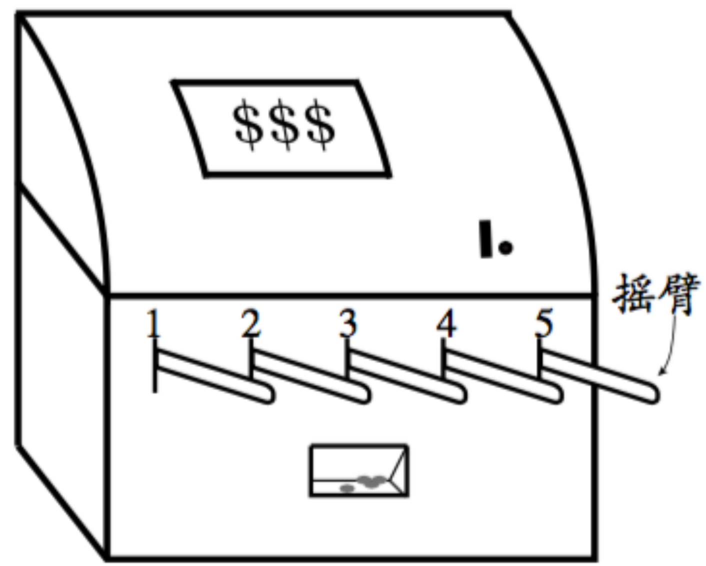
Beyond classical search

Bandit search

Tree search: Monte-Carlo Tree Search

General search:
- Gradient decent
- Metaheuristic search

Bandit



Multiple arms

Each arm has an expected reward,
  but unknown, with an unknown distribution

Maximize your award in fixed trials

Simplest strategies

Two simplest strategies

Exploration-only:
for T trails and K arms, try each arm T/K times

problem?

Simplest strategies

Two simplest strategies

Exploration-only:
for T trails and K arms, try each arm T/K times

problem? waste on suboptimal arms

Simplest strategies

Two simplest strategies

Exploration-only:
for T trails and K arms, try each arm T/K times

problem? waste on suboptimal arms

Exploitation-only:

Simplest strategies

Two simplest strategies

Exploration-only:
for T trails and K arms, try each arm T/K times

problem? waste on suboptimal arms

Exploitation-only:
1. try each arm once
2. try the observed best arm T-K times

Simplest strategies

Two simplest strategies

Exploration-only:

for T trails and K arms, try each arm T/K times

problem? waste on suboptimal arms

Exploitation-only:

1. try each arm once
2. try the observed best arm T-K times

problem?

Simplest strategies

Two simplest strategies

Exploration-only:

for T trails and K arms, try each arm T/K times

problem? waste on suboptimal arms

Exploitation-only:

1. try each arm once
2. try the observed best arm T-K times

problem? risk of wrong best arm

ε-greedy

Balance the exploration and exploitation:

with ε probability, try a random arm
with 1-ε probability, try the best arm

ε controls the balance

输入: 摇臂数 K;
奖赏函数 R;
尝试次数 T;
探索概率 ϵ.

过程:

$r = 0$;
$\forall i = 1, 2, \ldots K: \ Q(i) = 0, \ \mathrm{count}(i) = 0$;
for $t = 1, 2, \ldots, T$ **do**
  if rand()$< \epsilon$ **then**
    $k =$ 从 $1, 2, \ldots, K$ 中以均匀分布随机选取
  else
    $k = \arg\max_i Q(i)$
  end if
  $v = R(k)$;
  $r = r + v$;
  $Q(k) = \frac{Q(k) \times \mathrm{count}(k) + v}{\mathrm{count}(k) + 1}$;
  $\mathrm{count}(k) = \mathrm{count}(k) + 1$;
end for

输出: 累积奖赏 r

Softmax

Balance the exploration and exploitation:

Choose arm with probability

$$P(k) = \frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^{K} e^{\frac{Q(i)}{\tau}}}, \qquad (16.4)$$

τ controls the balance

输入: 摇臂数 K;
奖赏函数 R;
尝试次数 T;
温度参数 τ.

过程:

$r = 0$;
$\forall i = 1, 2, \ldots K: \ Q(i) = 0, \ \mathrm{count}(i) = 0$;
for $t = 1, 2, \ldots, T$ **do**
  $k =$ 从$1, 2, \ldots, K$ 中根据式(16.4)随机选取
  $v = R(k)$;
  $r = r + v$;
  $Q(k) = \frac{Q(k) \times \mathrm{count}(k) + v}{\mathrm{count}(k) + 1}$;
  $\mathrm{count}(k) = \mathrm{count}(k) + 1$;
end for

输出: 累积奖赏 r

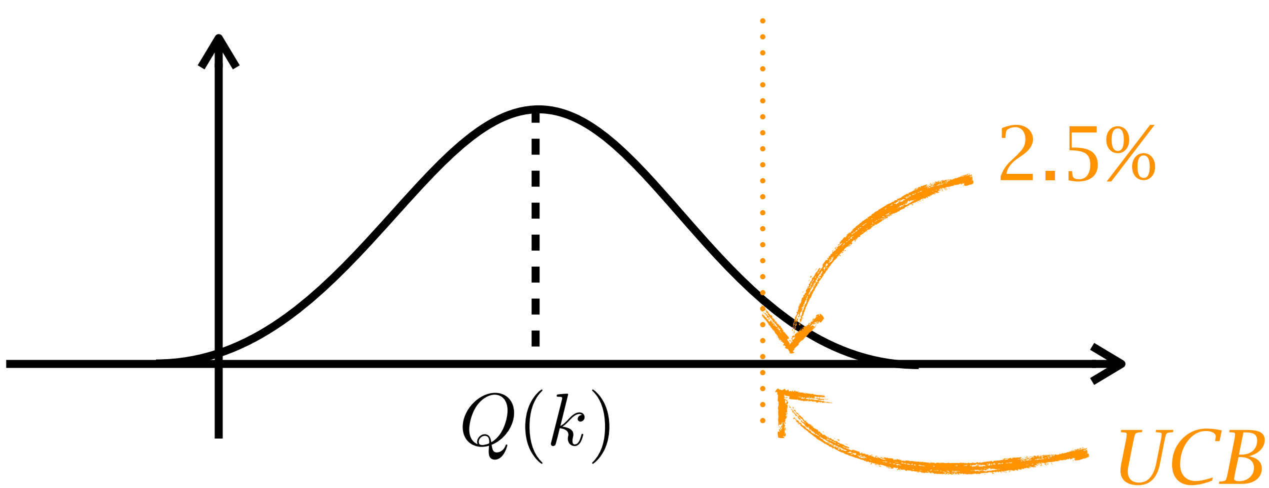
Upper-confidence bound

Balance the exploration and exploitation:

Choose arm with the largest value of

average reward + upper confidence bound

$$Q(k) + \sqrt{\frac{2\ln n}{n_k}},$$

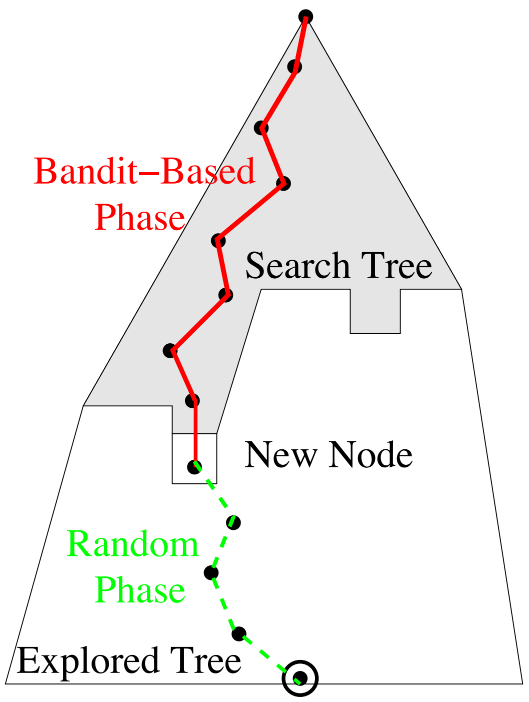


Monte-Carlo Tree Search

Kocsis Szepesvári, 06

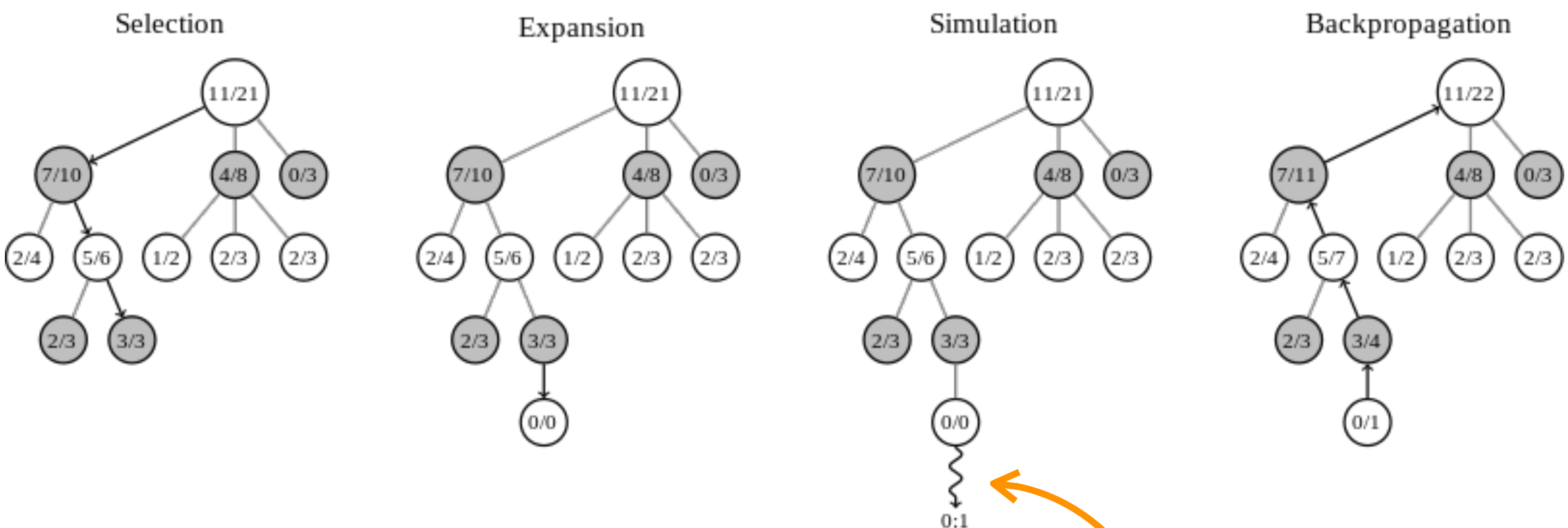
Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - Select next action bis
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate
- Returned solution:
  - Path visited most often



Monte-Carlo Tree Search

Example:



Pic from https://en.wikipedia.org/wiki/Monte_Carlo_tree_search#cite_note-Kocsis-Szepesvari-5

rollout

How to select the leave? As bandit

Monte-Carlo Tree Search

```java
public class TreeNode {
    static Random r = new Random();
    static int nActions = 5;
    static double epsilon = 1e-6;

    TreeNode[] children;
    double nVisits, totValue;

    public void selectAction() {
        List<TreeNode> visited = new LinkedList<TreeNode>();
        TreeNode cur = this;
        visited.add(this);
        while (!cur.isLeaf()) {
            cur = cur.select();
            visited.add(cur);
        }
        cur.expand();
        TreeNode newNode = cur.select();
        visited.add(newNode);
        double value = rollOut(newNode);
        for (TreeNode node : visited) {
            // would need extra logic for n-player game
            node.updateStats(value);
        }
    }
```

```java
public void expand() {
    children = new TreeNode[nActions];
    for (int i=0; i<nActions; i++) {
        children[i] = new TreeNode();
    }
}
```

```java
public void updateStats(double value) {
    nVisits++;
    totValue += value;
}
```

codes from http://mcts.ai/code/java.html

Monte-Carlo Tree Search

```
private TreeNode select() {
    TreeNode selected = null;
    double bestValue = Double.MIN_VALUE;
    for (TreeNode c : children) {
        double uctValue = c.totValue / (c.nVisits + epsilon) +
                Math.sqrt(Math.log(nVisits+1) / (c.nVisits + epsilon)) +
                    r.nextDouble() * epsilon;
        // small random number to break ties randomly in unexpanded nodes
        if (uctValue > bestValue) {
            selected = c;
            bestValue = uctValue;
        }
    }
    return selected;
}
```

```
public
    st
    st
    st

    Tr
    do

    pu

            cur = cur.select();
            visited.add(cur);
        }
        cur.expand();
        TreeNode newNode = cur.select();
        visited.add(newNode);
        double value = rollOut(newNode);
        for (TreeNode node : visited) {
            // would need extra logic for n-player game
            node.updateStats(value);
        }
    }
```

```
;
    totValue += value;
}
```

codes from http://mcts.ai/code/java.html

Monte-Carlo Tree Search

optimal? Yes, after infinite tries

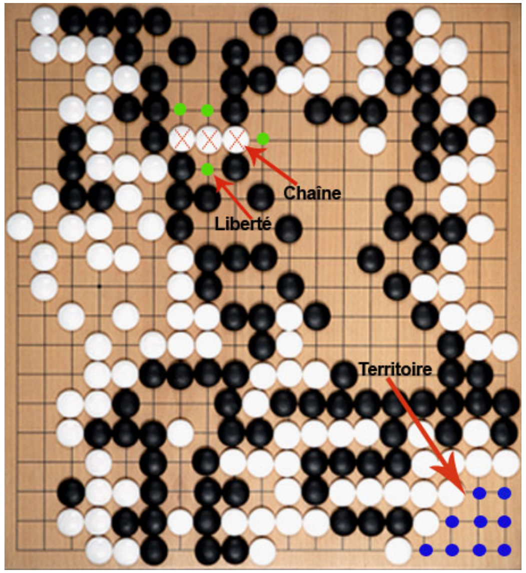
compare with alpha-beta pruning
no need of heuristic function

Monte-Carlo Tree Search

Improving random rollout

Monte-Carlo-based — Brügman 93

1. Until the goban is filled,
   add a stone (black or white in turn)
   at a uniformly selected empty position
2. Compute $r = \text{Win(black)}$
3. The outcome of the tree-walk is r



Improvements ?

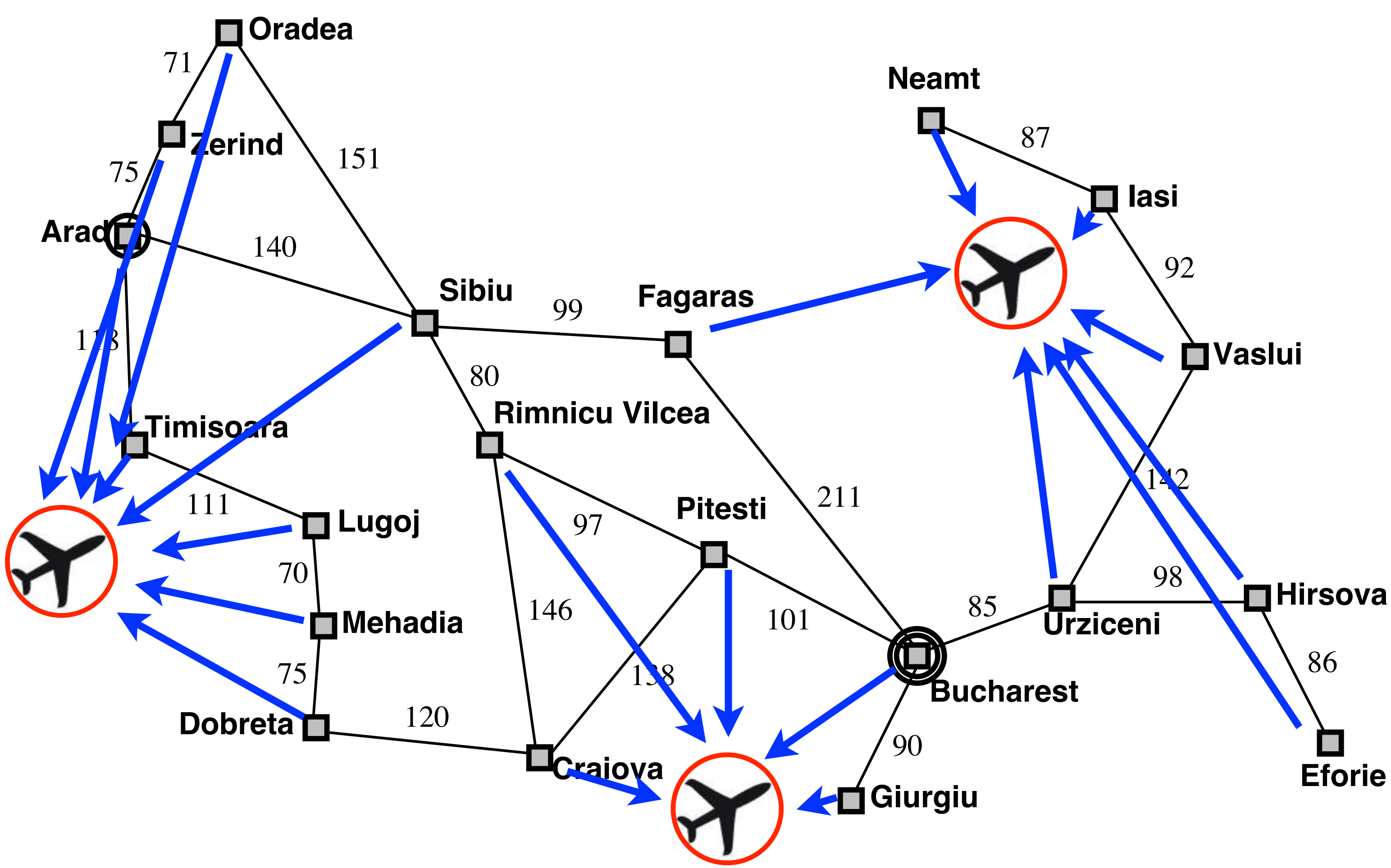
- Put stones randomly in the neighborhood of a previous stone
- Put stones matching patterns — prior knowledge
- Put stones optimizing a value function — Silver et al. 07

General search

Greedy idea in continuous space

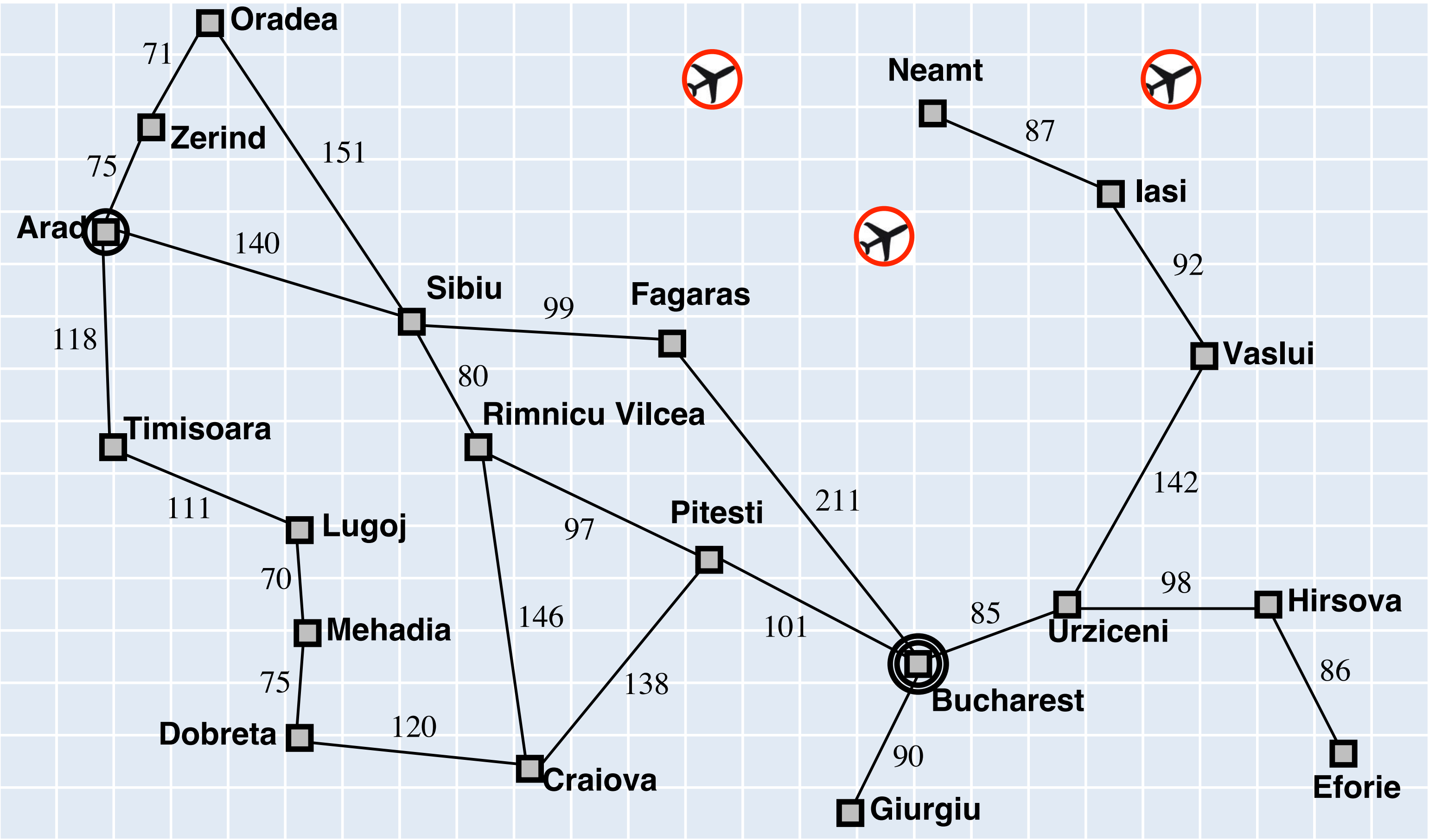
Suppose we want to site three airports in Romania:

- 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
- objective function $f(x_1, y_2, x_2, y_2, x_3, y_3) =$
  sum of squared distances from each city to nearest airport



Greedy idea in continuous space

discretize and use hill climbing



Greedy idea in continuous space

gradient decent

- 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
- objective function $f(x_1, y_2, x_2, y_2, x_3, y_3) =$
  sum of squared distances from each city to nearest airport

Gradient methods compute

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

to increase/reduce f, e.g., by $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

1-order method

Greedy idea in continuous space

gradient decent

– 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
– objective function $f(x_1, y_2, x_2, y_2, x_3, y_3) =$
sum of squared distances from each city to nearest airport

Sometimes can solve for $\nabla f(\mathbf{x}) = 0$ exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x})\nabla f(\mathbf{x})$
to solve $\nabla f(\mathbf{x}) = 0$, where $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$

2-order method

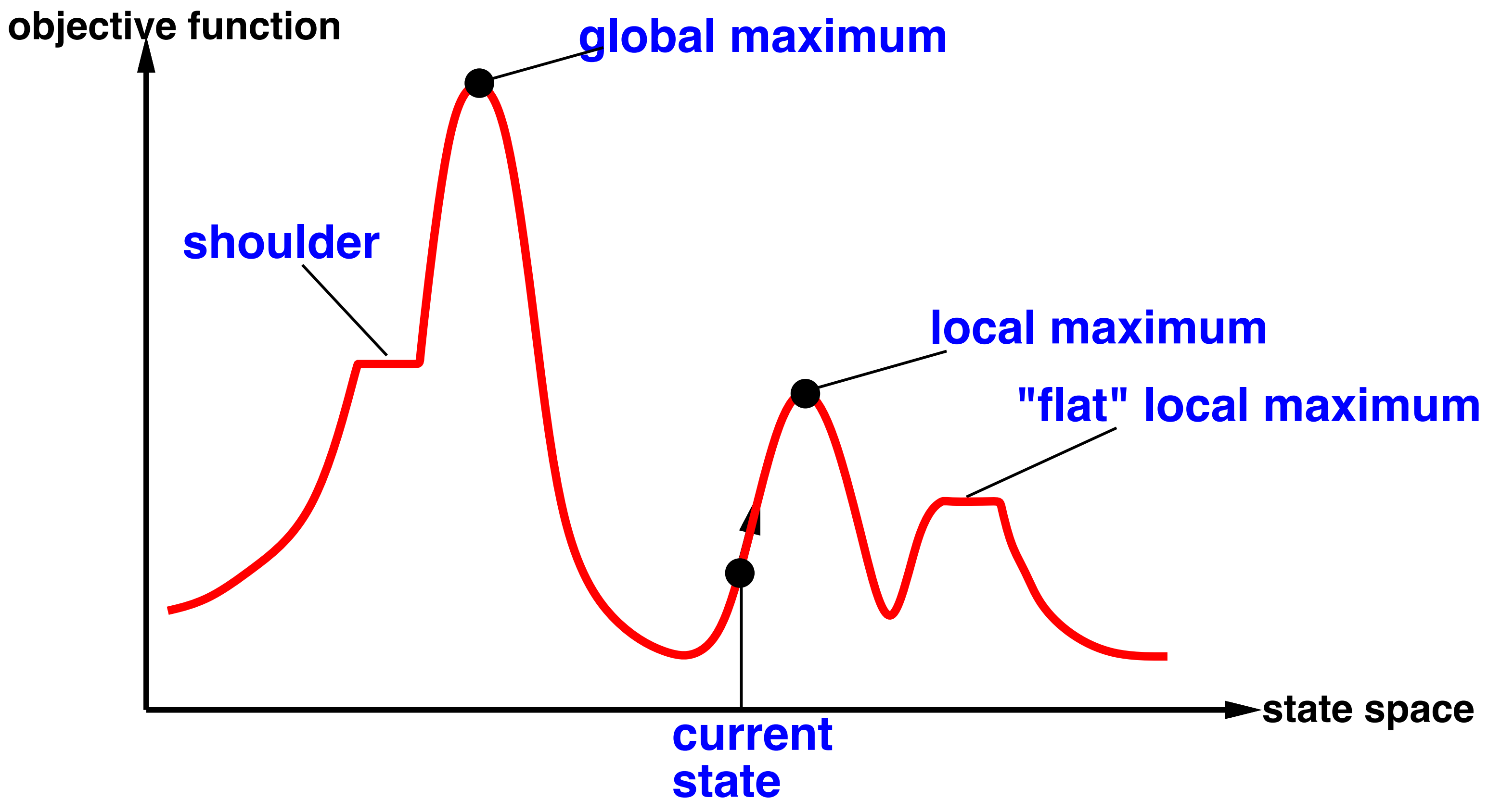
Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2} f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Greedy idea

1st and 2nd order methods may not find global optimal solutions

they work for convex functions



Meta-heuristics

"problem independent
"black-box
"zeroth-order method
...

and usually inspired from nature phenomenon

Simulated annealing

temperature from high to low

when high temperature, form the shape
when low temperature, polish the detail

Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

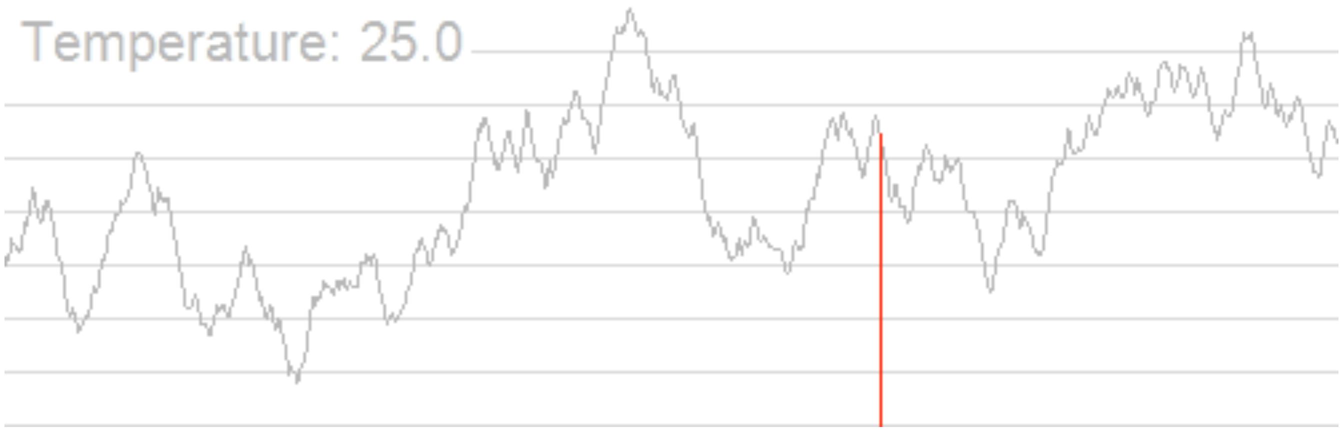
```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward 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^(Δ E/T)
```

the neighborhood range shrinks with T

the probability of accepting a bad solution decreases with T

Simulated annealing

a demo



graphic from http://en.wikipedia.org/wiki/Simulated_annealing

Local beam search

Idea: keep k states instead of 1; choose top k of all their successors

Not the same as k searches run in parallel!
Searches that find good states recruit other searches to join them

Problem: quite often, all k states end up on same local hill

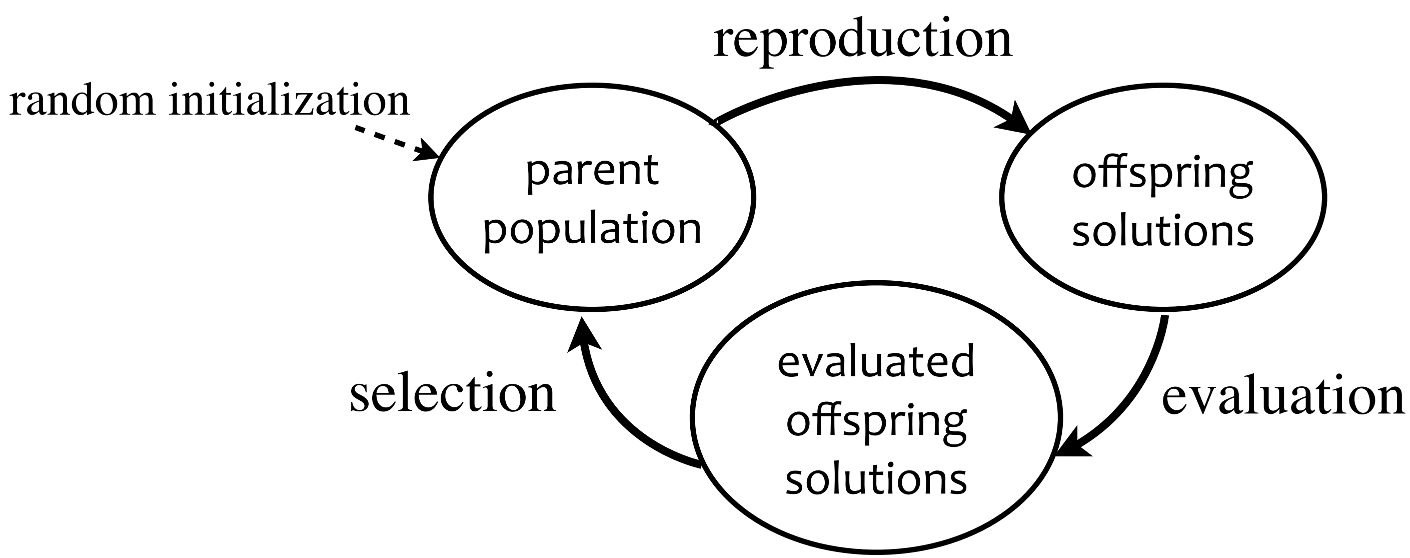
Idea: choose k successors randomly, biased towards good ones

Observe the close analogy to natural selection!

Genetic algorithm

a simulation of Darwin's evolutionary theory

over-reproduction with diversity
nature selection

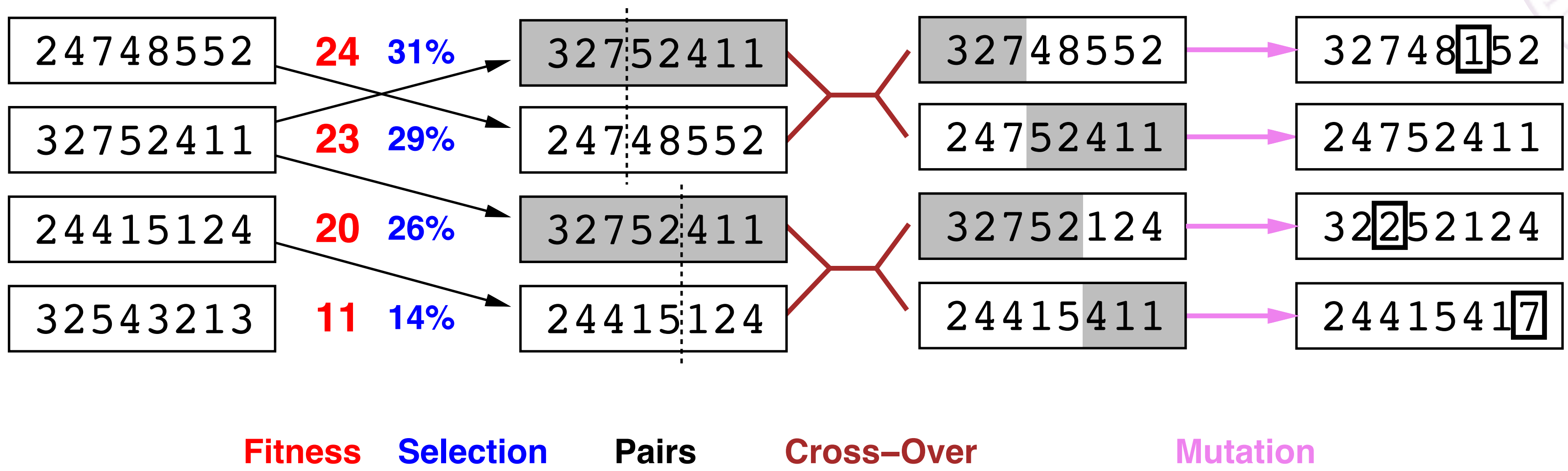


Genetic algorithm

Encode a solution as a vector,

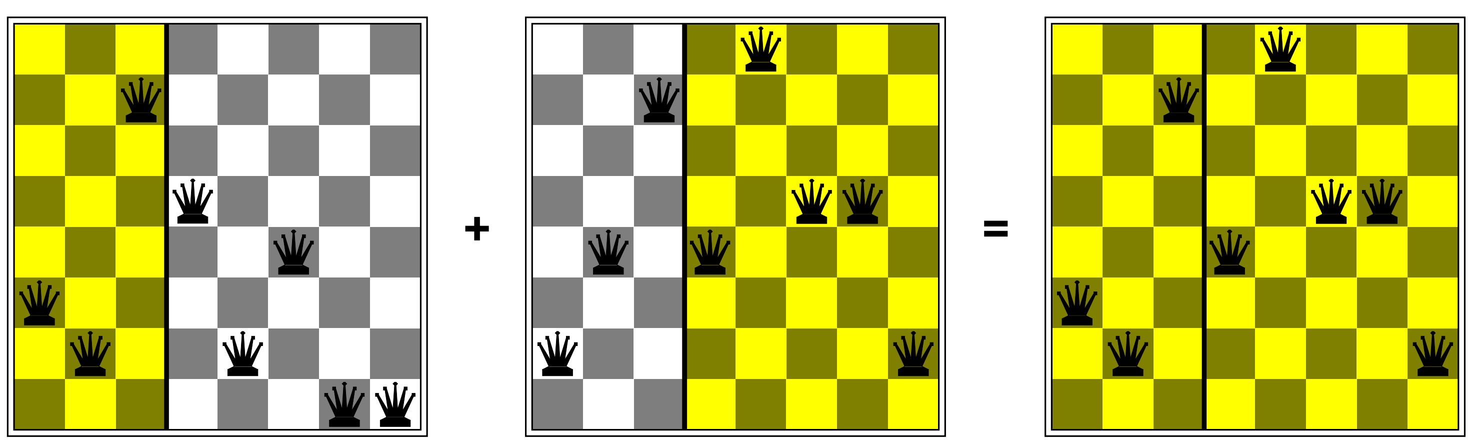
$Pop \leftarrow n$ randomly drawn solutions from $\mathcal{X}$
for t=1,2,... **do**
    $Pop^m \leftarrow \{mutate(s) \mid \forall s \in Pop\}$, the mutated solutions
    $Pop^c \leftarrow \{crossover(s_1, s_2) \mid \exists s_1, s_2 \in Pop^m\}$, the recombined solutions
    evaluate every solution in Pop^c by $f(s) (\forall s \in Pop^c)$
    $Pop^s \leftarrow$ selected solutions from Pop and Pop^c
    $Pop \leftarrow Pop^s$
    terminate if meets a stopping criterion
end for

Genetic algorithm



GAs require states encoded as strings (GPs use programs)

Crossover helps **iff substrings are meaningful components**



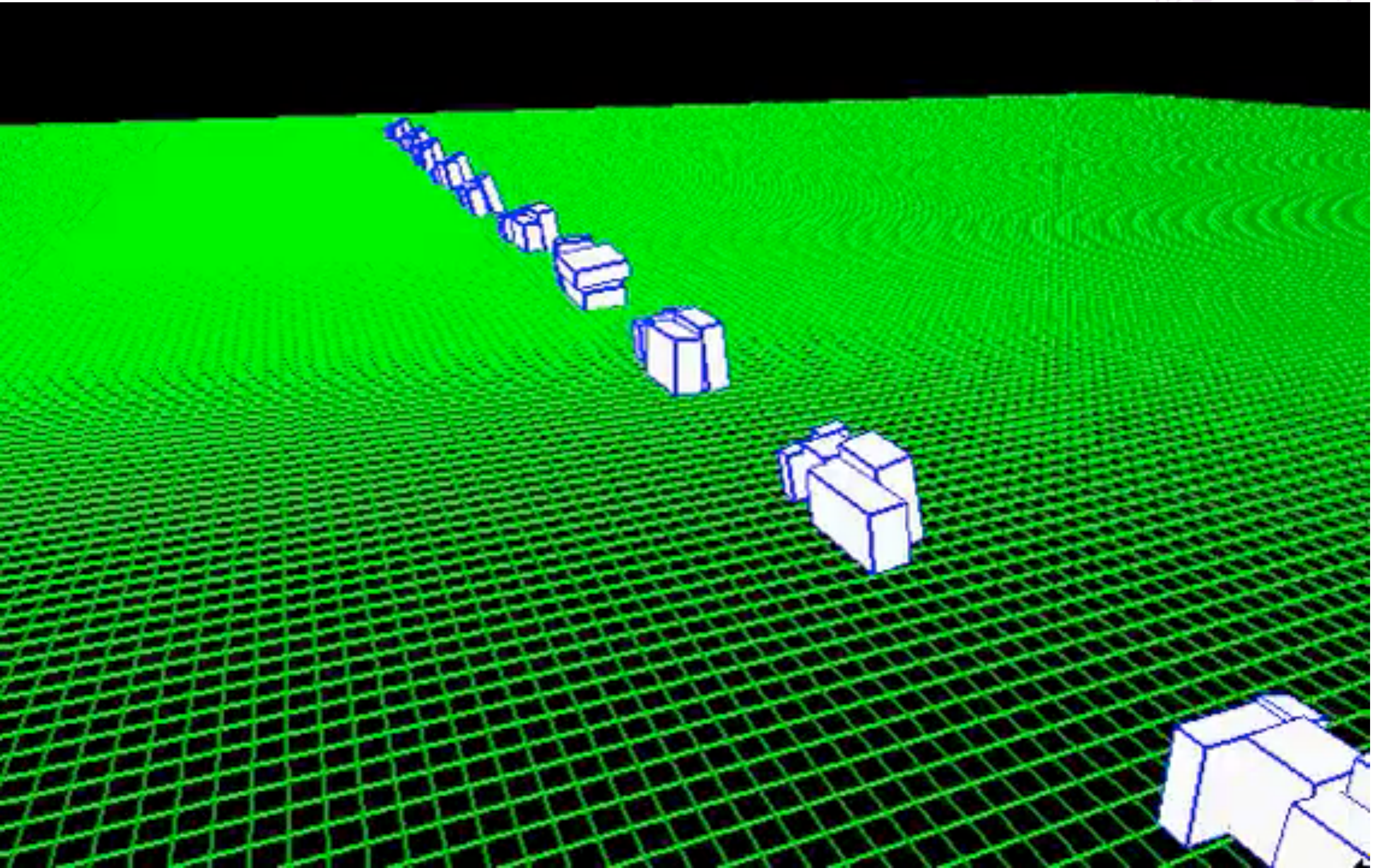
Example

Encode a solution as a vector with length n
each element of the vector can be chosen from $\{1,\ldots,V\}$
parameters: mutation probability p_m, crossover probability p_c

Pop = randomly generate n solutions from $\{1,\ldots,V\}^n$
for t=1,2, ... do
    Pop^m=emptyset, Pop^c=emptyset
    for $i = 1$ to n
        let x be the i-th solution in Pop
        for $j = 1$ to n: with probability p_m, change x_j by a random value from $\{1,\ldots,V\}$
        add x into Pop^m
    end for
    for $i = 1$ to n
        let x be the i-th solution in Pop^m
        let x' be a randomly selected solution from Pop^m
        with probability p_c, exchange a random part of x with x'
        add x into Pop^c
    end for
    evaluate solutions in Pop^c, select the best n solutions from Pop and Pop^c to Pop
    terminal if a good solution is found
end for

An evolutionary of virtual life

An evolutionary of virtual life



Properties of meta-heuristics

zeroth order

do not need differentiable functions

convergence

will find an optimal solution if $P(x^* \mid x) > 0$
or $P(x \to x_1 \to \ldots \to x_k \to x^*) > 0$

a missing link

observation → simulation

Properties of meta-heuristics

zeroth order

do not need differentiable functions

convergence

will find an optimal solution if $P(x^* \mid x) > 0$
or $P(x \rightarrow x_1 \rightarrow \ldots \rightarrow x_k \rightarrow x^*) > 0$

a missing link

observation → simulation

observation → **principle** → application

Properties of meta-heuristics

fireworks algorithm
grey wolf optimizer
brainstorm algorithm
2010
bat algorithm
gravitational search algorithm
intelligent water drops algorithm
river formation dynamics
artificial bee colony algorithms

2000
differential evolution
particle swarm optimization algorithms
ant colony optimization algorithms
1990
memetic algorithms
artificial immune systems
cultural algorithms
tabu search
simulated annealing
1980

1970
evolutionary strategies
evolutionary programming
genetic algorithms
1960

Example

hard to apply traditional optimization methods
but easy to test a given solution

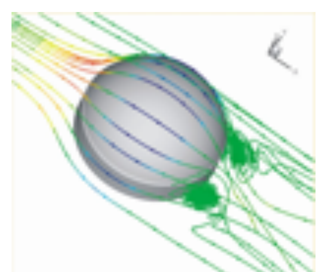
Representation:

parameterize

represented as a vector of parameters

Fitness:

x_i



$f(x_i)$

test by simulation/experiment

Example

Series 700

Series N700

Technological overview of the next generation Shinkansen high-speed train Series N700

M. Ueno[1], S. Usui[1], H. Tanaka[1], A. Watanabe[2]

[1]*Central Japan Railway Company, Tokyo, Japan,* [2]*West Japan Railway Company, Osaka, Japan*

Abstract

In March 2005, Central Japan Railway Company (JR Central) has completed prototype

waves and other issues related to environmental compatibility such as external noise. To combat this, an aero double-wing-type has been adopted for nose shape (Fig. 3). This nose shape, which boasts the most appropriate aerodynamic performance, has been newly developed for railway rolling stock using the latest analytical technique (i.e. genetic algorithms) used to develop the main wings of airplanes. The shape resembles a bird in flight, suggesting a feeling of boldness and speed

On the Tokaido Shinkansen line, Series N700 cars save 19% energy than Series 700 cars, despite a 30% increase in the output of their traction equipment for higher-speed operation (Fig. 4).

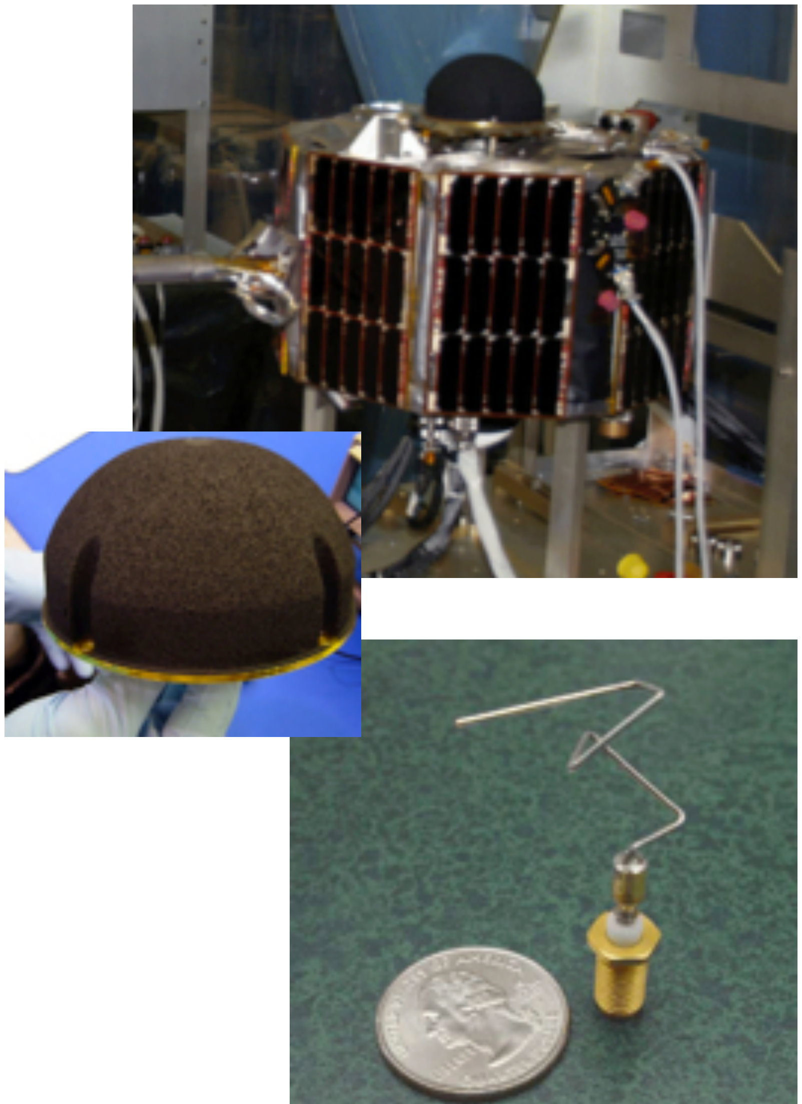
This is a result of adopting the aerodynamically excellent nose shape, reduced running resistance thanks to the drastically smoothened car body and under-floor equipment, effective

this nose ... has been newly developed ... using the latest analytical technique (i.e. **genetic algorithms**)

N700 cars save **19%** energy ... **30%** increase in the output... This is a result of adopting the ... nose shape

Example

NASA ST5 satellite

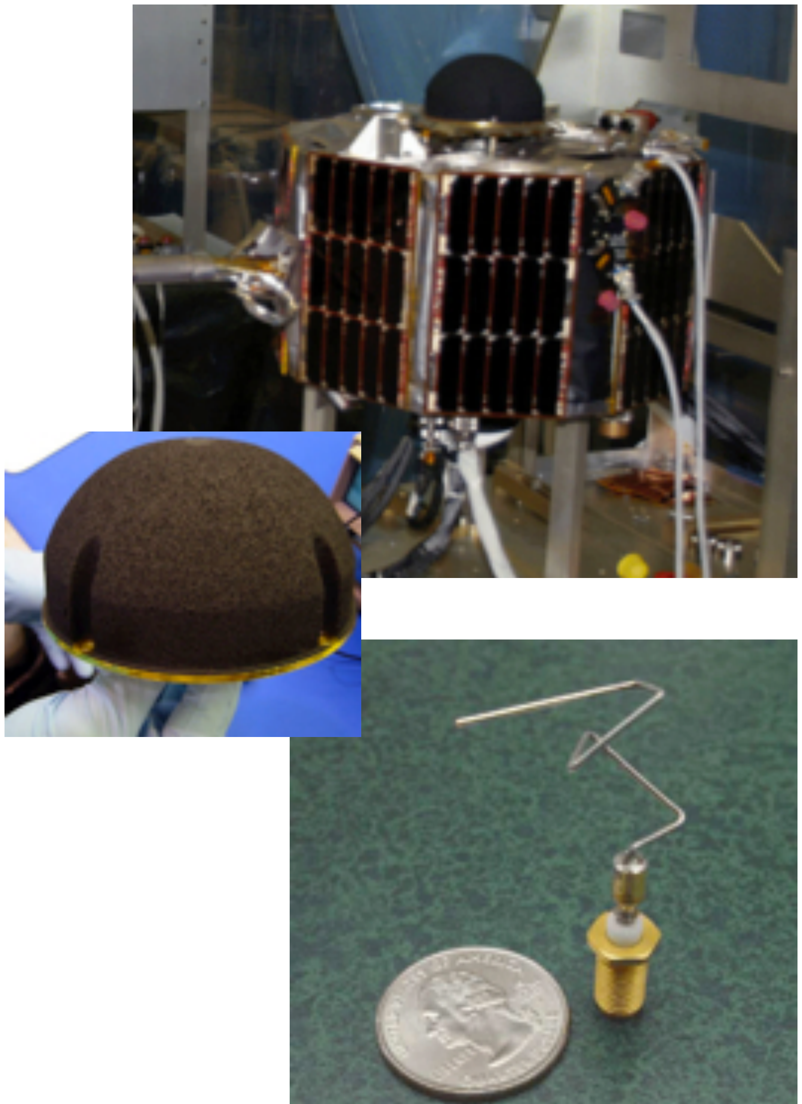


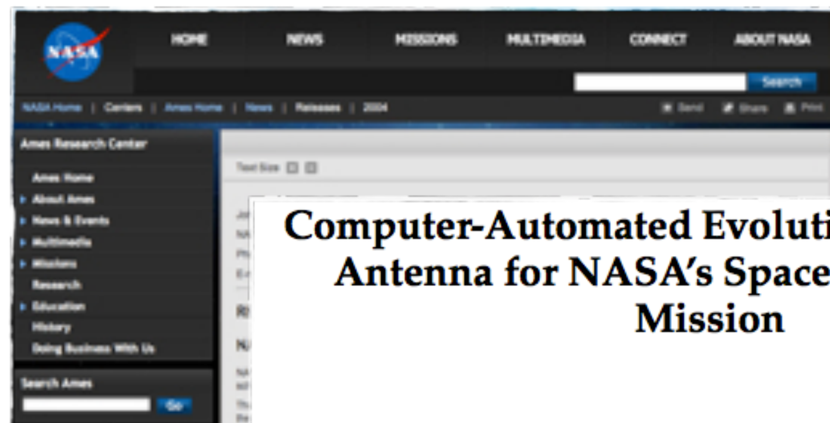
hard to apply traditional optimization methods but easy to test a given solution

Example

NASA ST5 satellite

hard to apply traditional optimization methods
but easy to test a given solution





Computer-Automated Evolution of an X-Band Antenna for NASA's Space Technology 5 Mission

Gregory. S. Hornby Gregory.S.Hornby@nasa.gov
Mail Stop 269-3, University Affiliated Research Center, UC Santa Cruz, Moffett Field, CA, 94035, USA

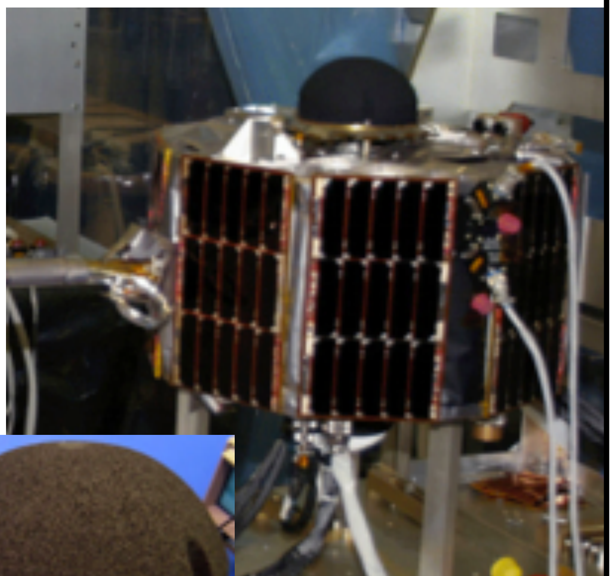
Jason D. Lohn Jason.Lohn@west.cmu.edu
Carnegie Mellon University, Mail Stop 23-11, Moffett Field, CA 94035, USA

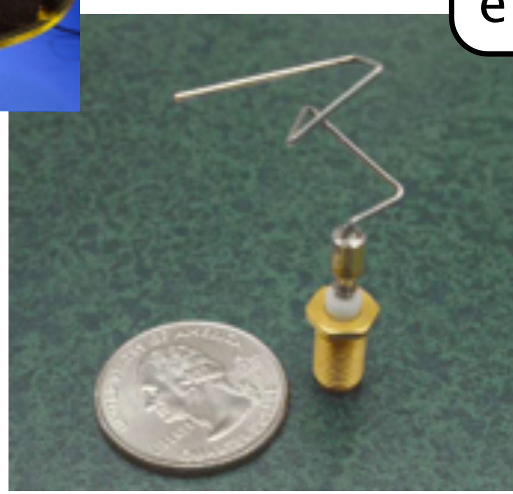
Derek S. Linden dlinden@jemengineering.com
JEM Engineering, 8683 Cherry Lane, Laurel, MD 20707, USA Moffett Field, CA 94035, USA

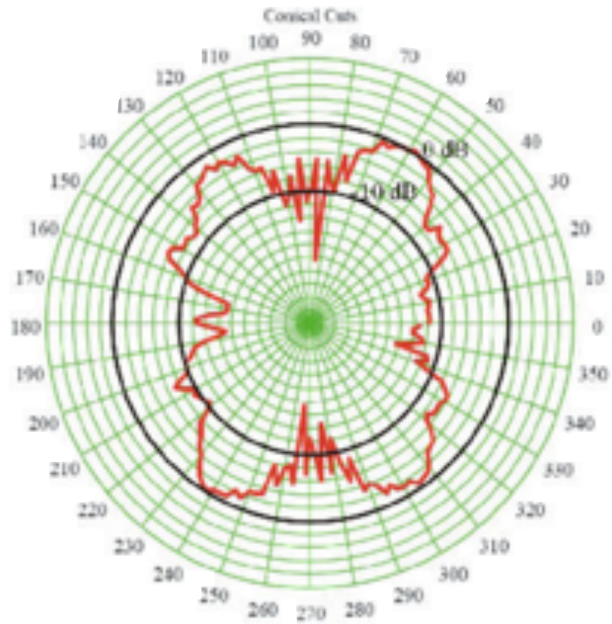
Since there are two antennas on each spacecraft, and not just one, it is important to measure the overall gain pattern with two antennas mounted on the spacecraft. For this, different combinations of the two evolved antennas and the QHA were tried on the the ST5 mock-up and measured in an anechoic chamber. With two QHAs 38% efficiency was achieved, using a QHA with an evolved antenna resulted in 80% efficiency, and using two evolved antennas resulted in 93% efficiency. Here "efficiency" means how much power is being radiated versus how much power is being eaten up in resistance, with greater efficiency resulting in a stronger signal and greater range. Figure 11

Example

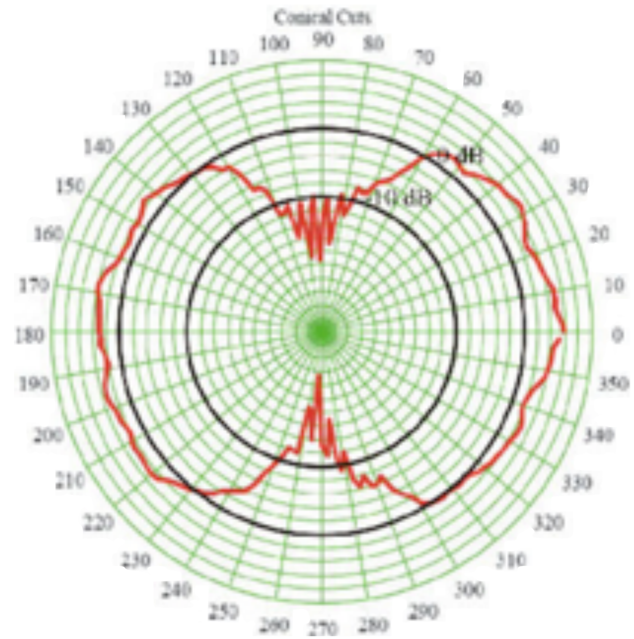
NASA ST5 satellite







QHAs(□ □) 38% efficiency



evolved antennas resulted in 93% efficiency

Jason D. Lohn Ja Lohn@west.cmu.edu
Carnegie Mellon University, Mail Stop 23-11, Moffett Field, C 035, USA
Derek S. Linden dlinde @jemengineering.com
JEM Engineering, 8683 Cherry Lane, Laurel, MD 20707, USA N offett Field, CA 94035, USA

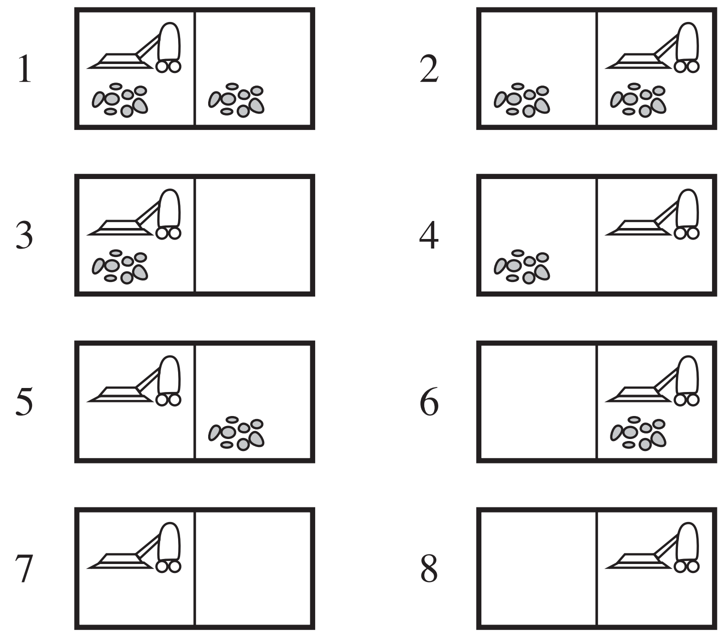
Since there are two antennas on each spacecraft, and not just one, it is important to measure the overall gain pattern with two antennas mounted on the spacecraft. For this, different combinations of the two evolved antennas and the QHA were tried on the the ST5 mock-up and measured in an anechoic chamber. With two QHAs 38% efficiency was achieved, using a QHA with an evolved antenna resulted in 80% efficiency, and using two evolved antennas resulted in 93% efficiency. Here "efficiency" means how much power is being radiated versus how much power is being eaten up in resistance, with greater efficiency resulting in a stronger signal and greater range. Figure 11

Different Environment Properties

Nondeterministic actions

In the **erratic vacuum world**, the *Suck* action works as follows:

- When applied to a dirty square the action cleans the square and sometimes cleans up dirt in an adjacent square, too.
- When applied to a clean square the action sometimes deposits dirt on the carpet.[9]

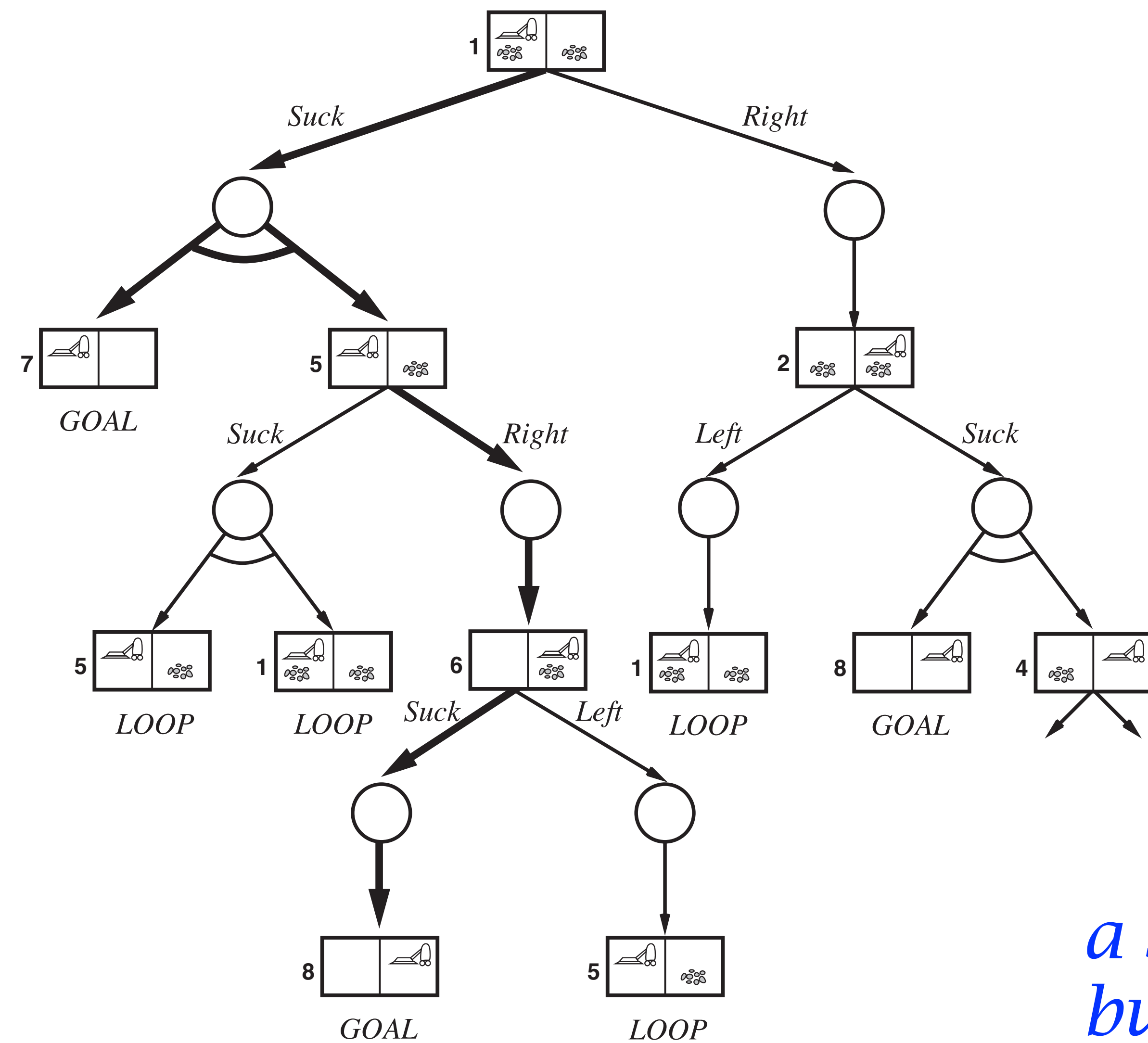


almost all real-world problems are nondeterministic

how do you solve this problem?

AND-OR tree search

OR node: different actions (as usual)
AND node: different transitions



a solution is not a path but a tree

Depth-first AND-OR tree search

function AND-OR-GRAPH-SEARCH($problem$) **returns** *a conditional plan, or failure*
  OR-SEARCH($problem$.INITIAL-STATE, $problem$, [ ])

---

function OR-SEARCH($state, problem, path$) **returns** *a conditional plan, or failure*
  if $problem$.GOAL-TEST($state$) **then return** the empty plan
  if $state$ is on $path$ **then return** $failure$
  for each $action$ **in** $problem$.ACTIONS($state$) **do**
    $plan \leftarrow$ AND-SEARCH(RESULTS($state, action$), $problem$, $[state \mid path]$)
    if $plan \neq failure$ **then return** $[action \mid plan]$
  return $failure$

---

function AND-SEARCH($states, problem, path$) **returns** *a conditional plan, or failure*
  for each s_i **in** $states$ **do**
    $plan_i \leftarrow$ OR-SEARCH($s_i, problem, path$)
    if $plan_i = failure$ **then return** $failure$
  return [**if** s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else** $\ldots$ **if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

Search with no observations

search in **belief (in agent's mind)**