Population-Based Local Search

2/1/17

Last time: Local search methods

Key ideas:

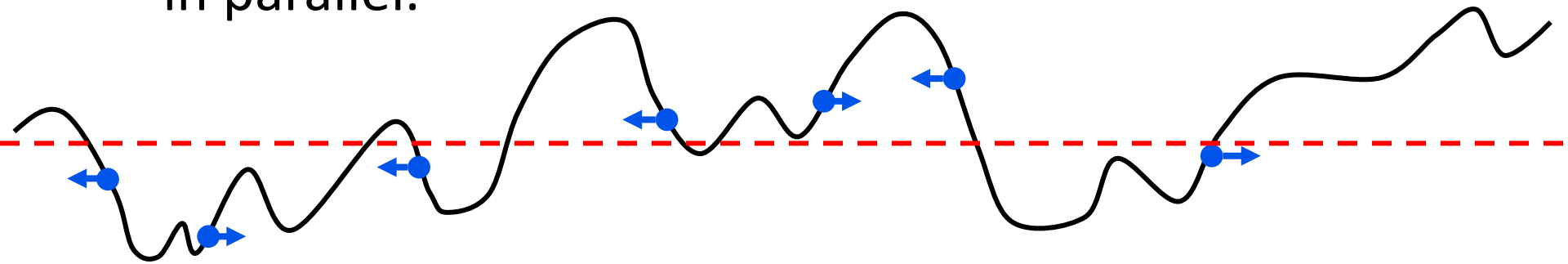
- Generate candidates
- Explore neighbors
- Move toward higher value (lower cost)

Algorithms:

- Hill Climbing
  - Best neighbor or random improving neighbor
  - Random moves / random restarts
- Simulated Annealing
  - Initial temperature
  - Temperature decay

Ideas for improving hill climbing

- Instead of random restarts, run a bunch of searches in parallel.

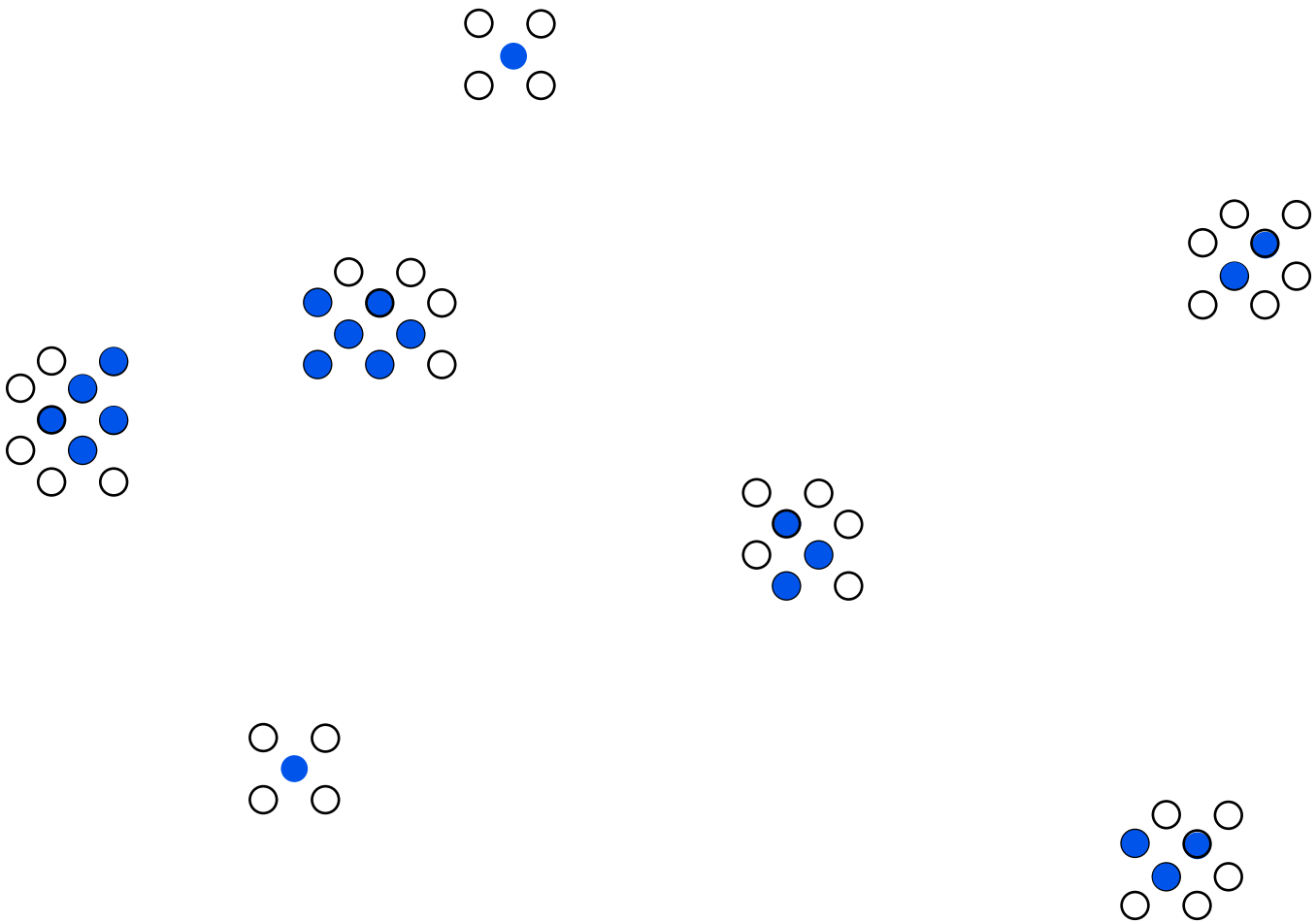


- Focus more effort on the searches that seem more promising.

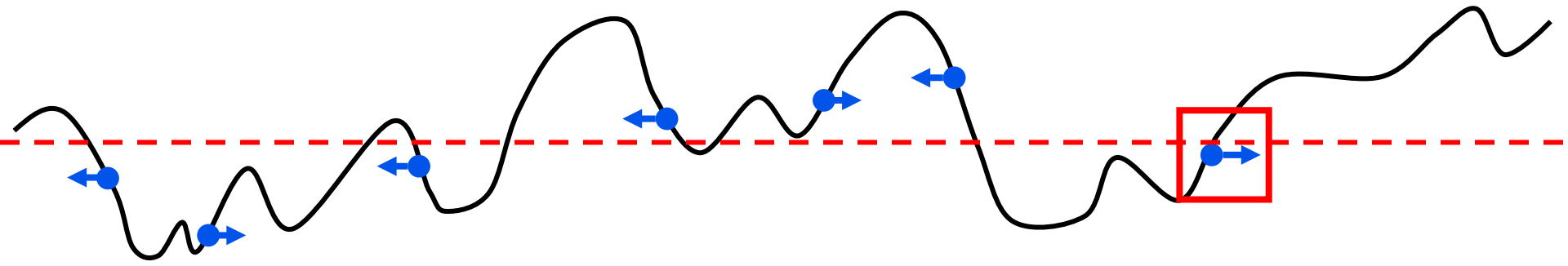
Beam Search Pseudocode

```
population = [POP_SIZE random states]
temp = INIT_TEMP
for i = 1:MAX_ITERS:
  candidates = []
  for individual in population:
    add neighbors of individual to candidates
  population = [best POP_SIZE candidates]
return best state encountered
```

Beam Search Illustrated



What's the problem here?



```
population = [best POP_SIZE candidates]
```

This may throw away some good candidates.

We don't want to focus the search too much too quickly.

What method do we already know to deal with this?

- Simulated annealing temperature

Stochastic Beam Search Pseudocode

```
population = [POP_SIZE random states]
temp = INIT_TEMP
for i = 1:MAX_ITERS:
  candidates = []
  for individual in population:
    add neighbors of individual to candidates
  population = sample(candidates, temp, POP_SIZE)
  temp *= DECAY
return best state encountered
```

Sampling the next population

Key idea: Gibbs sampling

Recall from Simulated Annealing:

```
acceptance probability = e^(delta / temp)
```

Choose new population by sampling:

```
probability ~ e^(-cost / temp)
```

Pseudocode:

```
weights = [e^(-cost(n)/temp) for each candidate n]
distribution = [weights[n] / sum(weights)]
population = [POP_SIZE draws from distribution]
```

Exercise: construct the distribution

```
INIT_TEMP = 20; DECAY = .95; round = 4
```

Suppose we have the following candidates:

```
cost(Boston, DC, Philly, NY) = 13.8
cost(Philly, Boston, DC, NY) = 16.6
cost(NY, Boston, Philly, DC) = 13.8
cost(DC, Philly, Boston, NY) = 13.8
cost(DC, NY, Philly, Boston) = 16.6
```

What is the probability distribution from which the next population will be drawn?

Genetic Algorithms

- Inspired by natural selection and genetic recombination.
  - Individuals with higher fitness are more likely to reproduce.
  - In sexual reproduction, the genomes of the parents get combined.

Key idea:

- Different candidates may have good partial solutions.
- Combining candidates may put those partial solutions together.

Genetic algorithm pseudocode

```
population = [POP_SIZE random states]
for i = 1:MAX_ITERS:
  new_population = []
  for j = 1:POP_SIZE:
    parent1, parent2 = select(population)
    child = reproduce(parent1, parent2)
    add child to new_population
return best state encountered
```

Selecting the parents

- There are many ways this could be done, but a good one is Gibbs sampling.
- Call the `select` function from stochastic beam search with `POP_SIZE = 2`.
- This means we need to add a temperature.

Producing a child

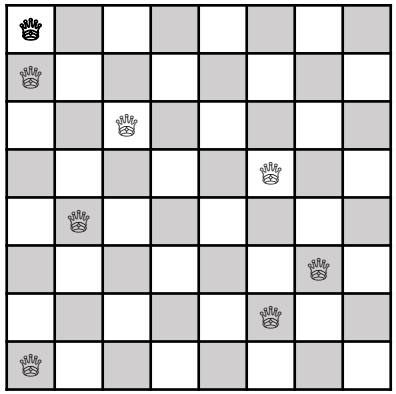
Inspired by biology:

- Crossover (combine the parents)
- Mutation (make random changes)

Pseudocode:

```
cross = random point in state representation
child = parent1[:cross] + parent2[cross:]
for each variable in child:
  if (small mutation probability):
    give it a random new value
  return child
```

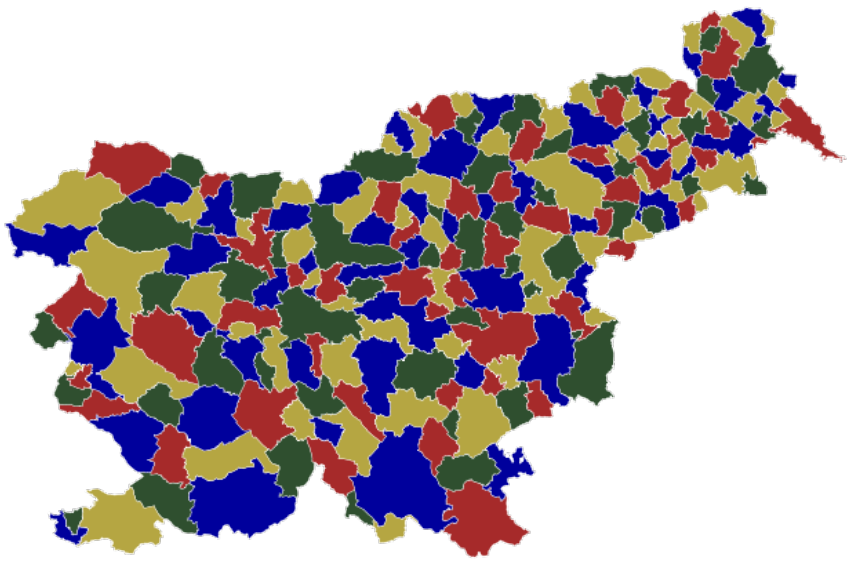
Reproduction in N-Queens

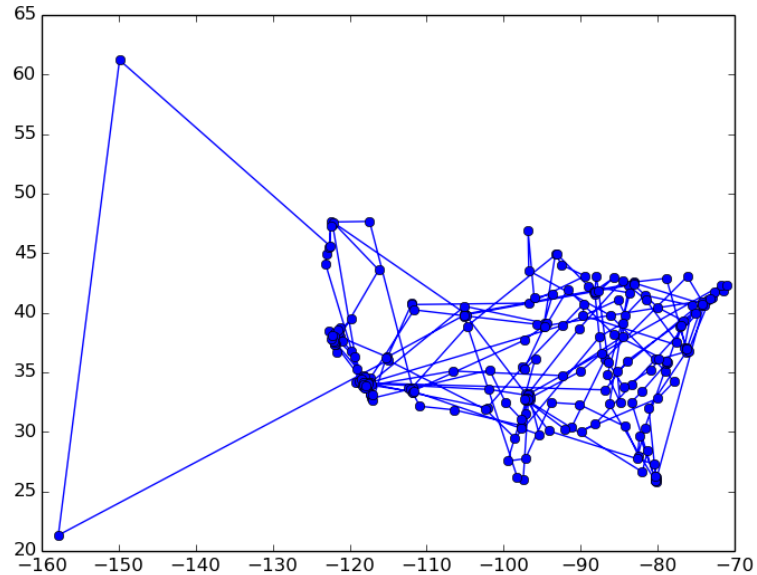


```
parent1 = (0,4,2,5,6,7,2,3)
parent2 = (2,6,4,7,1,6,5,0)
cross = random integer in [1,7]
# suppose it's 4
offspring after crossover = (0,4,2,5,1,6,5,0)
each element mutated with probability .02
# suppose 1 is mutated
mutated elements get random new values
# suppose it's 0
offspring returned = (0,0,2,5,1,6,5,0)
```

Discussion: state representation for GAs

What effect does crossover have on the state representation for these problems?





Is there an alternative state representation that would work better?