Artificial Intelligence

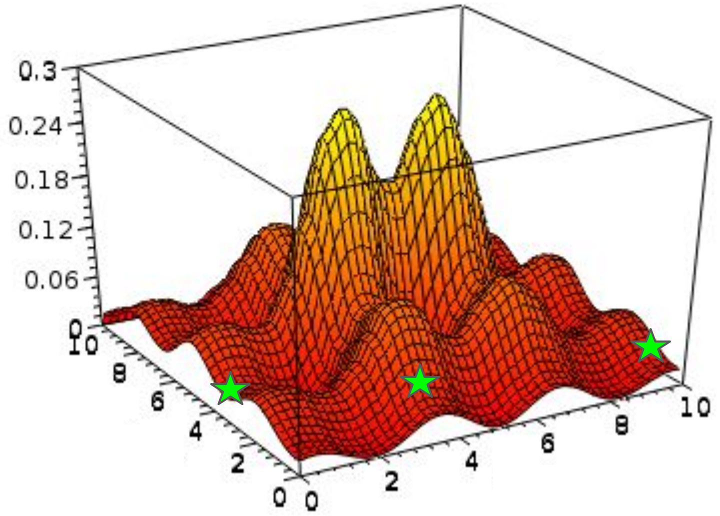
Lec 8: Local Search (contd.)

Pratik Mazumder

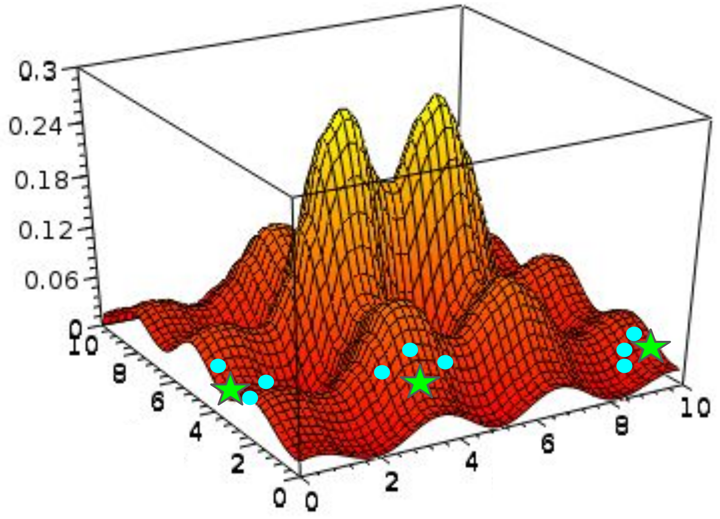
Local beam search

- Idea: Keeping only one node in memory is an extreme reaction to memory problems
- Keep track of k states instead of one [Beam of size k]
    - Initially: k randomly selected states
    - Next: determine all successors of k states
    - If any of the successors is a goal then finish
    - Else select k best from successors and repeat

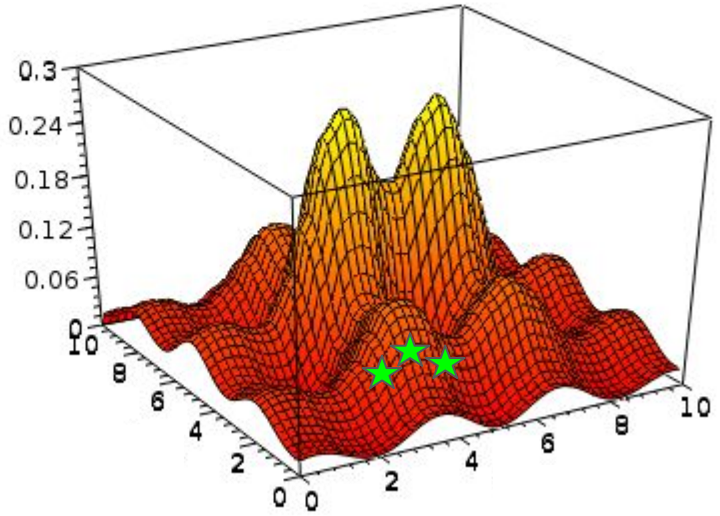
Local beam search



Local beam search



Local beam search



Local beam search

- Not the same as k random-start 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: Stochastic beam search

    - Choose k successors randomly, biased towards good ones
        - better valued states have higher probability of being selected
        - lesser valued states may also get selected but with a lower probability

    - Improves local beam search by introducing a degree of randomness into the selection process, maintaining diversity among candidate solutions, and reducing the risk of premature convergence to local optima.

- Observe the close analogy to natural selection!

Analogy to Natural Selection

- Stochastic Beam Search:
  - Maintains a population (or beam) of candidate solutions at each iteration. The candidates are evaluated, and a subset is chosen to generate the next generation of candidates.
  - From the current set of candidates, a subset is selected based on their quality or fitness, often with some randomness (stochasticity) involved.
  - The stochastic element helps maintain diversity in the candidate population, reducing the risk of premature convergence to suboptimal solutions.

- Natural Selection:
  - In nature, a population of organisms exists, each with varying traits. The population evolves over generations as individuals with favorable traits are more likely to survive and reproduce.
  - In biological evolution, individuals with traits that confer a survival or reproductive advantage are more likely to pass on their genes to the next generation. However, survival is also influenced by chance, introducing a stochastic element.
  - The stochastic nature of survival and reproduction ensures that not only the fittest but also less optimal individuals can contribute to the gene pool, maintaining genetic diversity and allowing the population to explore a broader range of traits.

Evolutionary Computation

- Theory of Evolution [Charles Darwin, Alfred Russel Wallace]:
  - The central idea is that given a population, **variations** occur in reproduction and will be **preserved** in successive generations approximately in **proportion** to their effect on reproductive **fitness**.
- Essentials of Darwinian Evolution
  - Organisms **reproduce** in **proportion** to their **fitness** in the environment.
  - Offspring inherit traits from their **parents**.
  - Traits are **inherited** with some variation via **mutation** and **recombination**.
- Evolutionary Computation: A collection of **computational methods inspired** by biological **evolution**
  - A **population of candidate** solutions **evolves** over time, with the **fittest** in each generation **contributing** the **most** offspring to the next generation.
  - Offspring are produced via **crossover** between parents, along with random **mutations** and other "genetic" operations.

Genetic algorithms

- **Variant** of **stochastic beam search** in which **successor states** are generated by **combining two parent** states rather than by modifying a single state.

- Twist on Local Search: successor is generated by combining two parent states

- Components of a GA:

  - Population of candidate solutions (nodes/states in local search space) to a given problem

  - Fitness function that assigns fitness to each candidate solutions (nodes/states in local search space) in the population

  - Selection procedure that selects individuals to mate/reproduce

  - Genetic operators that take existing candidate solutions and produce offspring with variation (e.g., mutation, crossover)

Genetic algorithms

Idea

1. Start out with a randomly generated population of candidate solutions (states in local search space).
2. Calculate the **fitness** of each state in the population.
3. Select **pairs of parents** with probability as a function of fitness rank in the population.
4. Create a new population: **Cross over** parents, **mutate** offspring, place in the new population.
5. Go to step 2.

Genetic algorithms

- States don’t have a set of genes that you can combine/mutate.

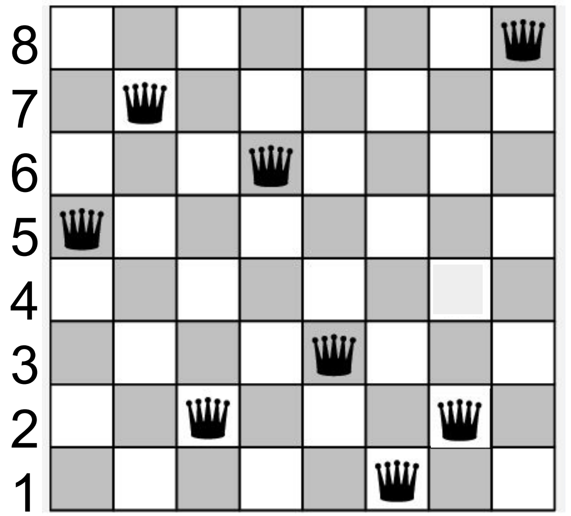
- A state is represented as a string over a finite collection of symbols (alphabet), e.g., binary.
  - e.g. in 8-queens the state must represent the position of 8 queens [shown in the next slide]
    - may restrict one to each column for simplicity and smaller search space, just as shown in an earlier lecture.

- Start with k randomly generated states (population).

- Evaluation function (fitness function):
  - Higher values for better states.

- Produce the next generation of states by “simulated evolution”
  - Random selection
  - Crossover
  - Random mutation

Genetic algorithms for 8-queens



String representation
57263128

Genetic algorithms

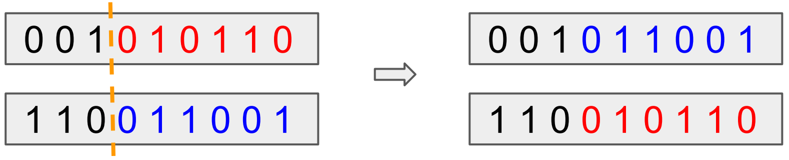
Each Iteration

1. Initial Population: A small subset of candidate solutions/states.
    - Based on the fitness function value of each state, compute a probability of selection for each state that is directly proportional to the fitness function value.
2. Random selection: Based on the above probability of selection, multiple pairs of states (parents) are selected at random for reproduction.
    - Based on the probability of selection, some states may be part of multiple pairs [fitter], and some states may not be part of any pairs.

Genetic algorithms

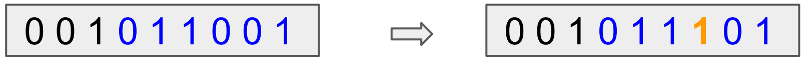
Genetic operators

3. Crossover: exchange subparts of two candidate solutions in each selected pair
    - Need to randomly select a crossover point.
    - Swap selected regions between the two parents.

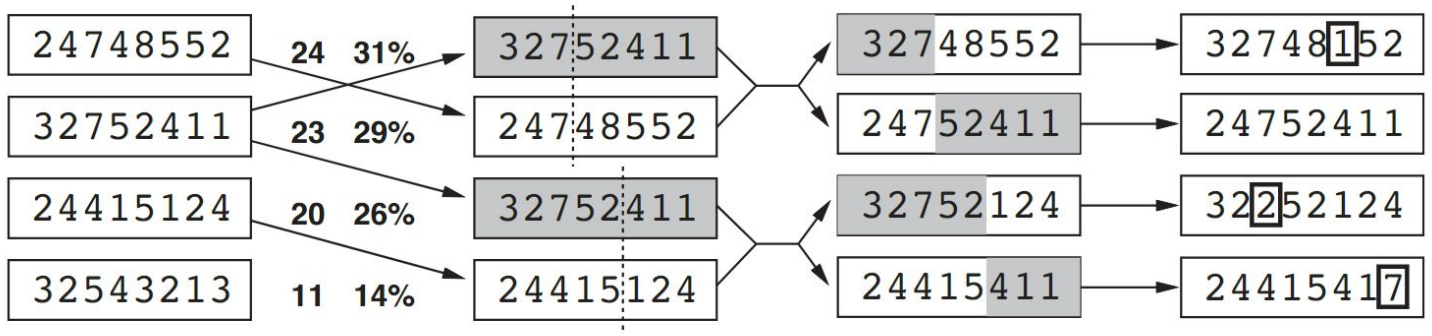


    - When two parent states are quite different, the crossover operation can produce a state that is a long way from either parent state.
    - It is often the case that the population is quite diverse early on in the process, so crossover (like simulated annealing) frequently takes large steps in the state space early in the search process and smaller steps later on when most individuals are quite similar.

4. Mutation: randomly change some location/attribute of the state with a small independent probability



Genetic algorithms for 8-queens



4 states for 8-queens problem

2 pairs of 2 states randomly selected based on fitness. Random crossover points selected

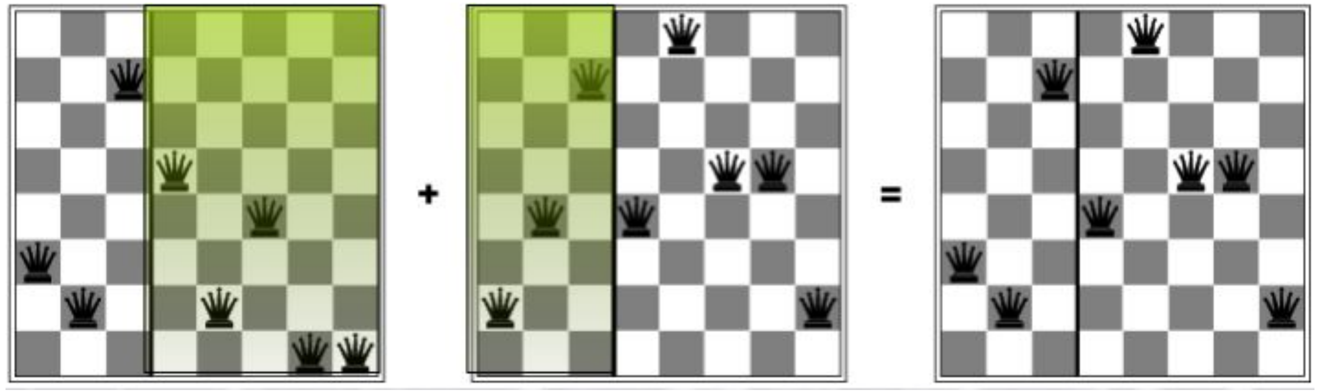
New states after crossover

Random mutation applied

Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)

- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms for 8-queens



Has the effect of “jumping” to a completely different new part of the search space

Genetic algorithms

- Genetic algorithm is a variant of “stochastic beam search” -
    - Like stochastic beam search, genetic algorithms combine an uphill tendency with random exploration and exchange of information among parallel search threads

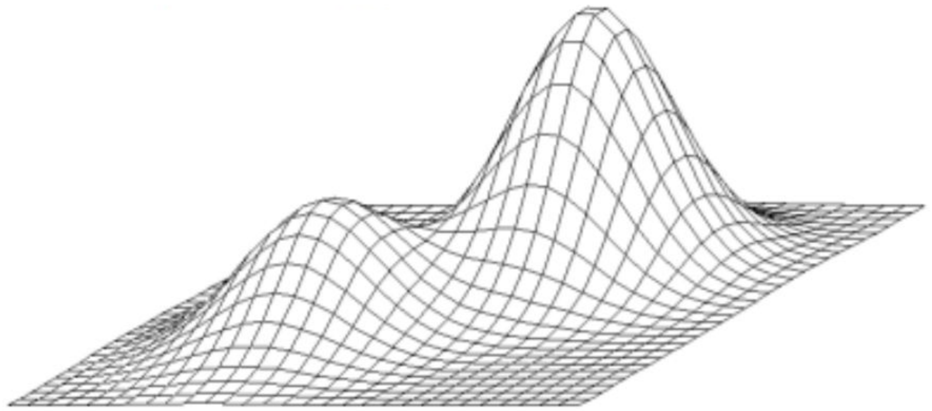
- Positive points
    - Random exploration can find solutions that local search can’t
        - (via crossover primarily)

    - Appealing connection to human evolution
        - “neural” networks, and “genetic” algorithms are metaphors!

- Negative points
    - Large number of “tunable” parameters - may not restrict to 2 parents, point of cross over, what size of population should we start with
        - Difficult to replicate performance from one problem to another

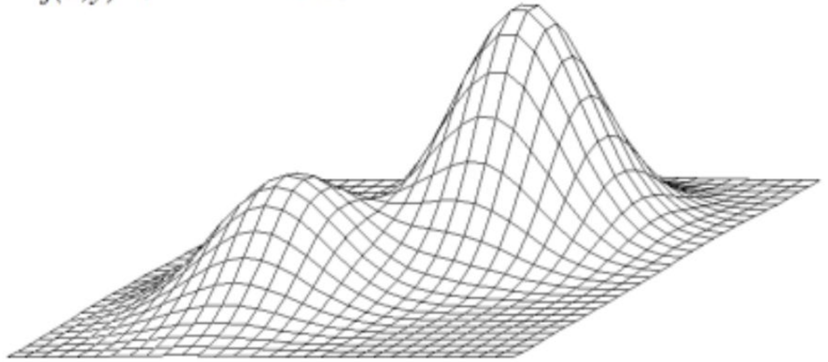
Optimization of Continuous Functions

- Upto now, we were finding an optimal state among a set of states that solves a problem.
- Can we extrapolate to any setting where you want to optimize a set of continuous variables?
- Discretization
    - use hill-climbing
- Gradient ascent/descent
    - make a move in the direction/opposite direction of the gradient.



Optimization of Continuous Functions

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



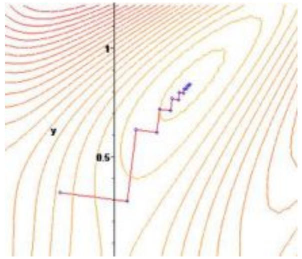
Gradient Descent

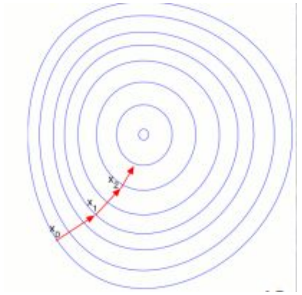
Assume we have a continuous function: $f(x_1,x_2,...,x_N)$ and we want to minimize over continuous variables $X_1,X_2,...,X_n$

1. Compute the gradients for all i: $f(x_1,x_2,\ldots,x_N)/\partial x_i$

2. Take a small downhill step in the opposite direction of the gradient [Greedy Step]:
   $x_i \leftarrow x_i - \lambda f(x_1,x_2,\ldots,x_N)/\partial x_i$

3. Repeat.





Optimization process doing some form of Local Search in the continuous space.