# Advanced Programming in Artificial Intelligence

**Local Search**

Grau en Enginyeria Informàtica
Universitat de Lleida

Josep Argelich

# Local Search

- Introduction
- Preliminaries
- Neighborhood Search
  - General schema
  - Variations
  - Examples
- Simulated Annealing
- Tabu Search
- Genetic Algorithms

# Advanced Programming in Artificial Intelligence

## Local Search

Introduction

# Introduction

- Fields of interest
  - Job shop scheduling
  - Time tabling
  - Crew scheduling, nurse rostering
  - Biotechnology
  - Design (VLSI)
  - Diagnosis
  - Propositional reasoning (SAT)
  - Probabilistic reasoning (Bayesian networks)

# Introduction

- Research Areas
  - Artificial Intelligence
  - Operation Research
  - Discrete Applied Mathematics
  - Algorithms
  - ...

artificial intelligence research group

# Algorithm Classification

- Systematic Algorithms (complete)
  - Always find the solution
    - If they have enough resources (time)
  - For decision problems
    - Solid (cannot prove wrong things, is correct) and complete (tells you if there is solution or not)
  - For optimization problems
    - Solid and complete (gives you the optimum)

# Algorithm Classification

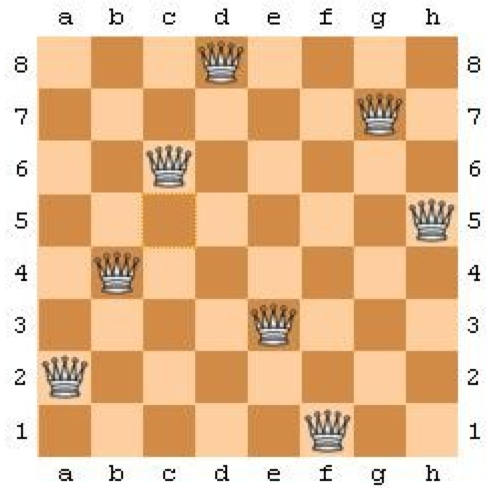
- No Systematic Algorithms (incomplete)
  - Do not always find the solution
    - Even with a lot of time resources
  - For decision problems
    - Solid, but not complete (cannot finish when there is no solution)
  - For optimization problems
    - Solid, but not complete (gives you the sub-optimum)

# Systematic vs. No systematic

- Systematic Algorithms
  - Need a lot of time to prove optimality
- No Systematic Algorithms
  - “Good” solutions in less time

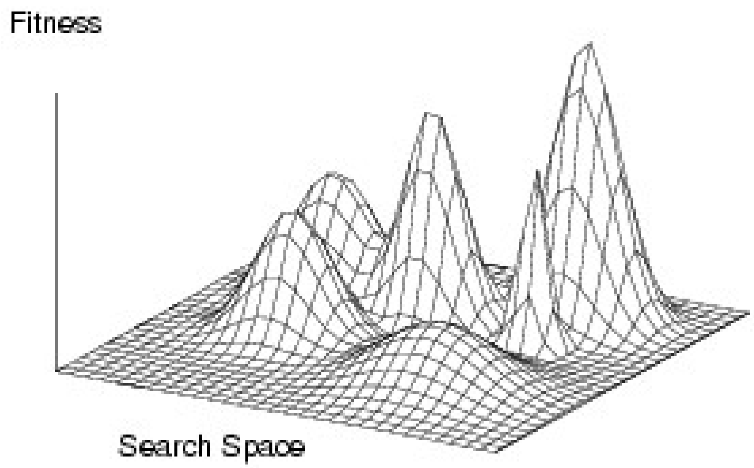
# Local Search Algorithms

- Family of Non Systematic Algorithms
- Do not check all the space of possible solutions
  - N queens problem
    - Space of possible solutions = 4,426,165,368
    - Number of solutions = 92



# Local Search Algorithms

- Family of Non Systematic Algorithms
- Can compute lower/upper bounds of the solution



# Advanced Programming in Artificial Intelligence

## Local Search

Preliminaries

# Local Search

- Problem (X, D, F)
  - $X = (x_1, x_2, \ldots, x_n)$ are the variables of the problem
  - $D = d_1 \times d_2 \times \ldots \times d_n$ space of possible solutions
    - $d_i$ is the domain of variable *i*
  - $f : D \rightarrow C$ is the function cost
    - For each possible solution returns the cost of that solution
    - The goal can be maximize or minimize this cost

# Advanced Programming in Artificial Intelligence

## Local Search

Neighborhood Search

# Local Search

- Idea of the algorithm
  - Starting point $\rightarrow X_0$
    - Initial assignment of values of the domain to the variables
  - Jump to different possible solutions using a local criteria
    - $X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow \ldots$
    - Can visit the same point several times
    - Can be points that are not visited
  - Finish
    - Decision: Solution found or maximum jumps reached
    - Optimization: target cost or maximum jumps reached

# Local Search

- Idea of the algorithm
  - Neighborhood function N(X)
    - N : D → #D
    - N(X) neighbors of point X
    - Example: N(X) = points that only differ in one value of one variable of X
    - Property: N(X) is accessible if I can go from any point to another going trough neighbors
    - Neighborhood → Concept of locality
  - Idea: $X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow$ ... (going from neighbor to neighbor)

artificial intelligence research group

# Local Search

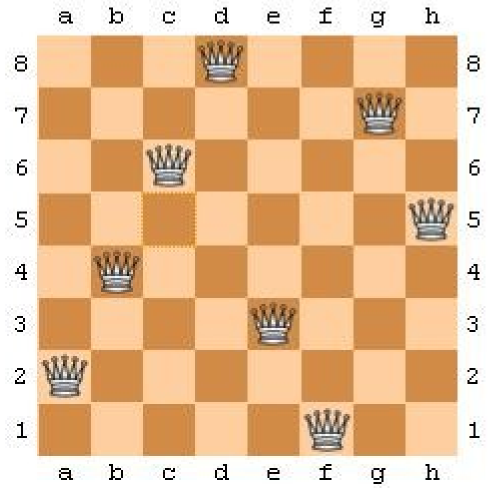
- Neighborhood function
  - The performance of the algorithm can depend a lot on the function N(X)
  - Usual function: $N_i(X)$ = points that differ from X in, at most, *i* variables
  - If *i* is big, $|N_i(X)|$ can be huge. The algorithm is more powerful, but each iteration is more expensive
    - Find the equilibrium (i = n?)
  - In practice $i \leq 2$

# Local Search

- Approaches using Neighborhood functions that we will see
  - Neighborhood Search (basic algorithm)
  - Simulated Annealing
  - Tabu Search
  - Genetic Algorithms

# Local Search

- But first, an example of random search
  - Problem: N queen
    - Space of possible solutions = 4,426,165,368
    - Number of solutions = 92



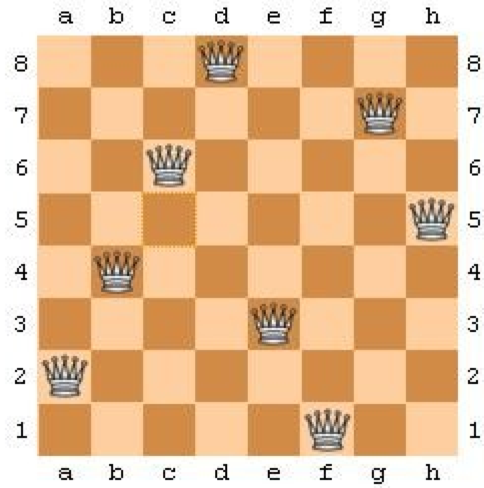
# Local Search

- But first, an example of random search
  - Problem: N queen (minimize menaced queens)

```
i := 0
X := random_point(X)
best_solution := X
best_cost := f(X)
while i < max_tries do
    X := random_point(X)
    if f(X) < best_cost then
        best_solution := X
        best_cost := f(X)
    end if
    i++
end while
return best_solution
```

# Neighborhood Search

- Basic algorithm
  - Problem: N queen
    - Space of possible solutions = 4,426,165,368
    - Number of solutions = 92



# Neighborhood Search

- Basic algorithm

```
i := 0
X := random_point(X)
best_solution := X
best_cost := f(X)
while i < max_tries do
    X := selection(N(X))
    if f(X) < best_cost then
        best_solution := X
        best_cost := f(X)
    end if
    i++
end while
return best_solution
```

# Neighborhood Search

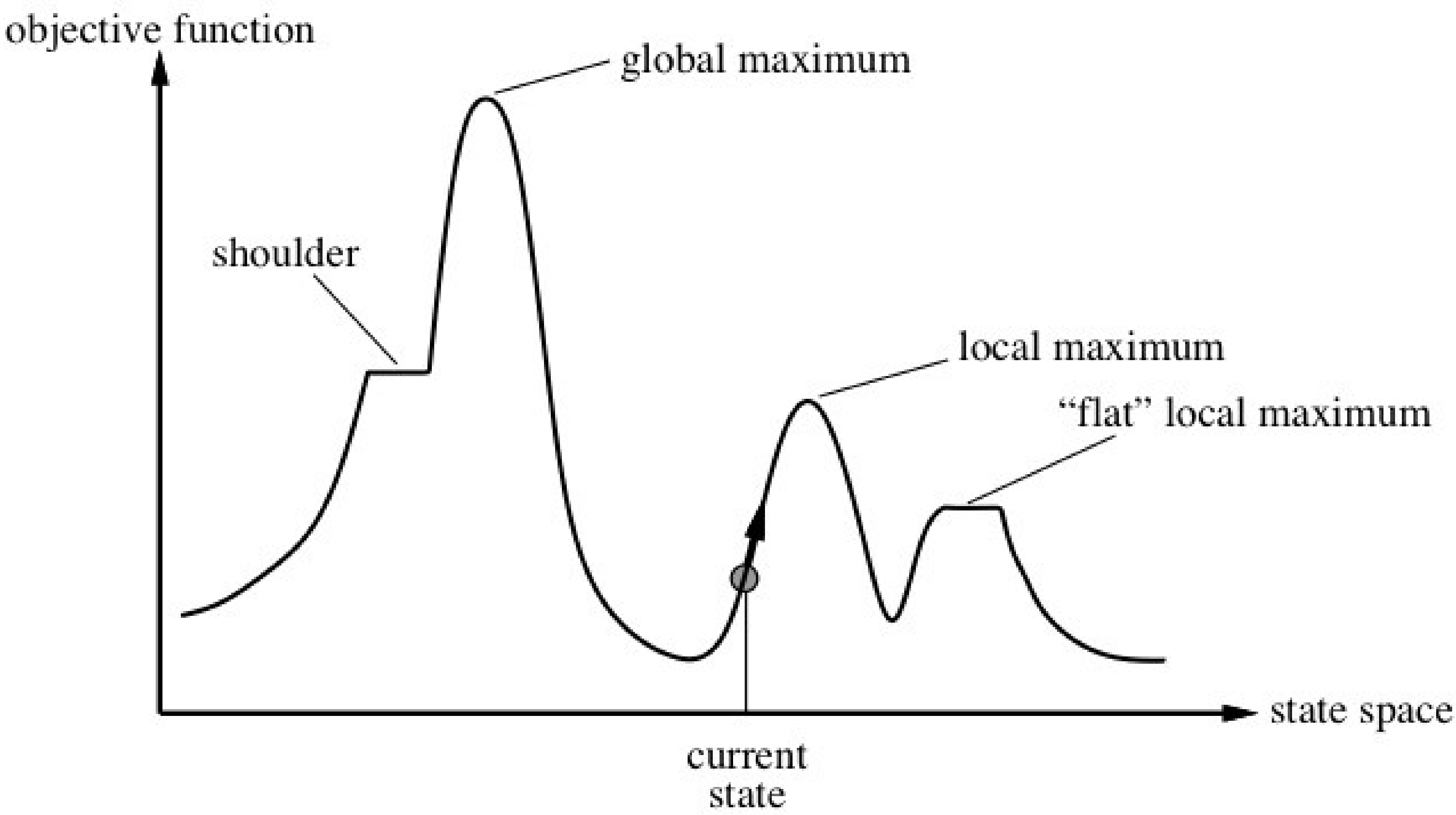
- Optimization version vs. Decision version
  - Homework: Modify previous algorithm from optimization to decision
- Variations in the selection function
  - Hill climbing
  - Steepest ascent
  - Random walk
  - Restarts
  - ...

# Selection Function

- Hill Climbing
  - Selection(N(X)): returns a neighbor that improves the cost function
    - Greedy algorithm
    - Good for concave and convex functions
  - Problems
    - The initial point is very important
    - Local maximum, shoulder (ridges and alleys), “flat” local maximum

# Selection Function

- State space landscape



# Selection Function

- Steepest ascent Hill Climbing
  - Selection(N(X)): returns the neighbor that improves more the cost function
    - Problems: like Hill Climbing
- Random walk
  - Selection(N(X)): returns a random neighbor
    - Problems: depends on luck
- Steepest ascent HC + Restarts
  - Restarts the search several times starting from random initial points
    - Minimize the influence of the starting point

# Selection Function

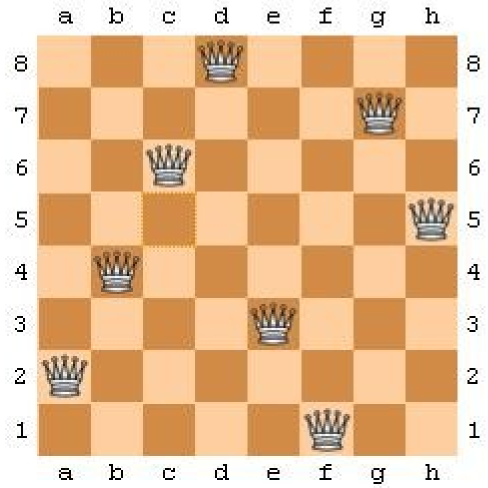
- Steepest ascent HC + mildest descent
  - Selection(N(X)): returns the neighbor that improves more the cost function and, if all of them do not improve, returns the one that worsens less
    - Problems: cycles
- Steepest asc. HC + mildest des. + Random walk
  - Selection(N(X)): previous + occasionally performs a random walk
    - Random walk: break cycles
    - Can be improved adding restarts

# Neighborhood Search

- Breakout
  - Dynamically modifies the cost function
    - To escape from local minimum
- Local beam search
  - Keep *k* states instead of one
  - For each state, generate the neighbors of each one
  - Select the *k* best and repeat

# Neighborhood Search

- Basic algorithm + Random walk + Restarts
  - Problem: N queen
    - – Steepest asc. HC + mildest des. + Random walk + Restarts



# Local Search

## Variants of neighborhood Search

# Simulated Annealing

- Inspiration come from annealing in metallurgy
  - Heating and controlled cooling of materials
- Slow cooling in Simulated Annealing (SA) Algorithm
  - Slow decrease in probability of accepting worse solutions
- Acceptance probability function
  - High probability at the beginning
  - Decreasing probability as the algorithm goes on

# Simulated Annealing

```
s ← s0; e ← E(s)                    // Initial state, energy
sbest ← s; ebest ← e                // Initial "best" solution
k ← 0                               // Energy evaluation count
while k < kmax and e > emax         // While time left & not good enough
    T ← temperature(1 – k / kmax)   // Temperature calculation
    snew ← neighbor(s)              // Pick some neighbor
    enew ← E(snew)                  // Compute its energy
    if P(e, enew, T) > random()     // Should we move to it?
        s ← snew; e ← enew          // Yes, change state
    endif
    if enew < ebest                 // Is this a new best?
        sbest ← snew; ebest ← enew  // Save 'new neighbour' to 'best found'
    endif
    k ← k + 1                       // One more evaluation done
endwhile
return sbest                        // Return the best solution found
```

artificial intelligence research group

# Simulated Annealing

- Acceptance probability function P(e, e', T)
  - Decreases over time in the range [1, 0]
    - Starts at 1
  - Can depend on current energy and the energy of the new state
    - Difference between energies
  - Example
    - P(T) = T (does not take into account the energy of the states)

# Simulated Annealing

- Examples
  - N Queens
  - Wikipedia
  - TSP Visualization

Troubleshooting: In case the applet does not work, add the URLs to the Security tab in the Java Control Panel

# Tabu Search

- Uses memory structures that describe the visited solutions
  - Previously visited solutions within a certain short-term period (simplest form)
- Short-term: List of solutions recently considered
- Intermediate-term: List of rules to bias the search
- Long-term: Rules that promote diversity

# Tabu Search

- Examples
  - Intermediate-term: Prohibits solutions that contain certain attributes (solutions to a TSP which include undesirable arcs)
  - Long-term: Restarts when the search becomes stuck
- Element expiration from tabu list
  - Usually in the same order they are added
- Effectiveness
  - Only for discrete spaces
    - Tabu list with discrete elements

# Tabu Search

## Wikipedia article (old)

```
s ← s0
sBest ← s
tabuList ← null
while (not stoppingCondition())
  candidateList ← null
  for(sCandidate in sNeighborhood)
    if(not containsTabuElements(sCandidate, tabuList))
      candidateList ← candidateList + sCandidate
    endif
  endfor
  sCandidate ← LocateBestCandidate(candidateList)
  if(fitness(sCandidate) < fitness(sBest))
    tabuList ← featureDifferences(sCandidate, sBest)
    sBest ← sCandidate
    while(size(tabuList) > maxTabuListSize)
      ExpireFeatures(tabuList)
    endwhile
  endif
endwhile
return(sBest)
```

# Tabu Search

```
s ← s0
sBest ← s
tabuList ← null
while (not stoppingCondition())
  candidateList ← null
  for(sCandidate in sNeighborhood)
    if(not containsTabuElements(sCandidate, tabuList))
      candidateList ← candidateList + sCandidate
    endif
  endfor
  sCandidate ← LocateBestCandidate(candidateList)
  if(fitness(sCandidate) < fitness(sBest))
    sBest ← sCandidate
  endif
  tabuList ← featureDifferences(sCandidate, sBest)
  while(size(tabuList) > maxTabuListSize)
    ExpireFeatures(tabuList)
  endwhile
endwhile
return(sBest)
```

# Tabu Search

- Examples
  - Tabu Search Applet
  - Visualization of metaheuristics

# Genetic Algorithms

- Inspiration in the natural evolution (Darwin's theory of evolution)
  - Only the strongest (more adapted) will survive
- Techniques inspired by natural evolution
  - Inheritance: From parent to descendant
  - Mutation: To maintain genetic diversity
  - Selection: The more adapted (fitness function)
  - Crossover: Genetic operator

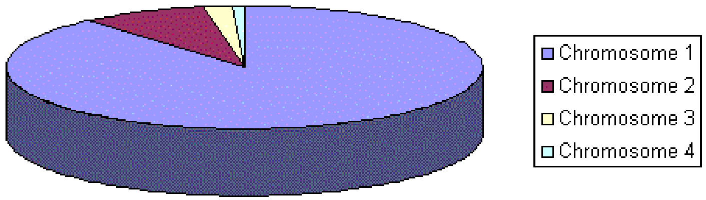
# Genetic Algorithms

- Initialization
  - Population size
  - Many individual solutions are randomly generated
- Selection
  - Strength (or adaptation) of an individual
  - Some members of the population are selected
    - Using a fitness function
  - Some members can be removed

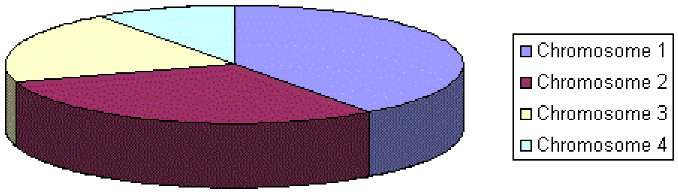
artificial intelligence research group

# Genetic Algorithms

- Crossover
  - Genetic operator that generates the next generation
  - From a pair (or more) of *parent* solutions and producing a *child* solution
  - Several methods to choose individuals
    - Roulette wheel: Fitness proportionate selection
    - Tournament: Several tournaments based on fitness
    - Rank: Sort by fitness and *total/rank* is your new fitness



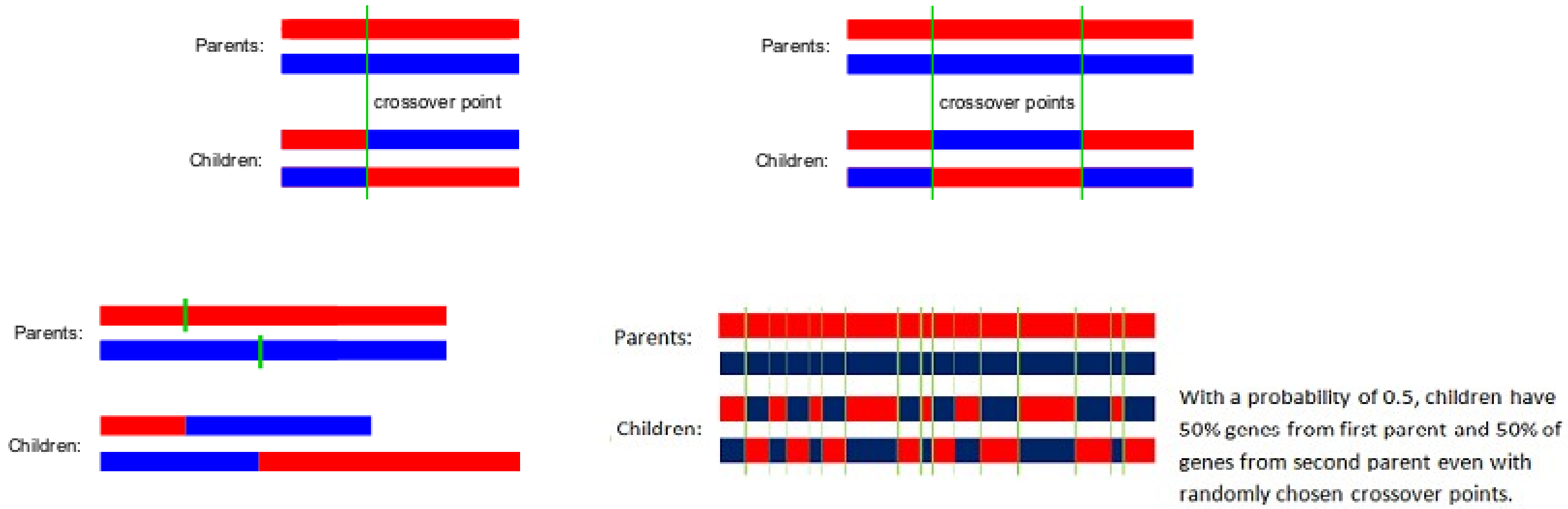

Roulette vs.




Rank

# Genetic Algorithms

- Crossover
  - Genetic operator that generates the next generation
  - From a pair (or more) of *parent* solutions and producing a *child* solution



# Genetic Algorithms

- Mutation
  - Genetic operator to maintain genetic diversity
  - Different mutation types
    - Bit string mutation: bit flip at random position
    - Flip bit: inverts the bits
    - ...
  - Other *mutations*
    - Select a random member of the population for the crossover

# Genetic Algorithms

```
pop ← random_population(n * k)
fit_pop ← fitness(pop)
best ← best_individual(pop, fit_pop)
while (not stopping_condition())
  best_pop ← selection(n, pop, fit_pop)
  pop ← crossover_mutation(best_pop)
  fit_pop ← fitness(pop)
  new_best ← best_individual(pop, fit_pop)
  if best > new_best
    best ← new_best
  endif
endwhile
return best

// n = Population size
// k = Constant to compute initial population size
```

# Genetic Algorithms

- Examples
  - BoxCar 2D
  - Genetic Walkers
  - Mar I/O Machine learning (audio)

# Local Search

- Introduction
- Preliminaries
- Neighborhood Search
  - General schema
  - Variations
  - Examples
- Simulated Annealing
- Tabu Search
- Genetic Algorithms

# Bibliography


- “Artificial Intelligence: A Modern Approach”. S. Russell and P. Norvig. Prentice Hall.
- Wikipedia and several Internet sources.


artificial intelligence research group

# Advanced Programming in Artificial Intelligence

**Local Search**

Grau en Enginyeria Informàtica
Universitat de Lleida

Josep Argelich