Exercise -1

3.16 A basic wooden railway set contains the pieces shown in Figure 3.32. The task is to connect these pieces into a railway that has no overlapping tracks and no loose ends where a train could run off onto the floor.

a. Suppose that the pieces fit together *exactly* with no slack. Give a precise formulation of

the task as a search problem.

b. Identify a suitable uninformed search algorithm for this task and explain your choice.

c. Explain why removing any one of the "fork" pieces makes the problem unsolvable.

d. Give an upper bound on the total size of the state space defined by your formulation.

(*Hint*: think about the maximum branching factor for the construction process and the

maximum depth, ignoring the problem of overlapping pieces and loose ends. Begin by

pretending that every piece is unique.)

Exercise -1

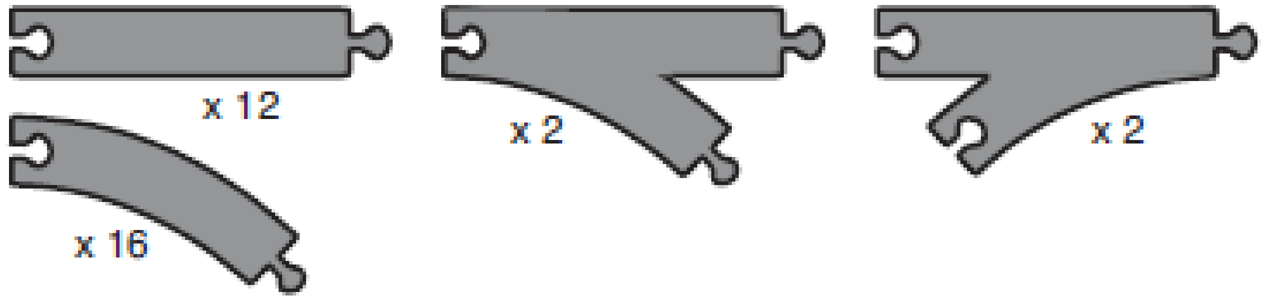


Figure 3.32 The track pieces in a wooden railway set; each is labeled with the number of copies in the set. Note that curved pieces and "fork" pieces ("switches" or "points") can be flipped over so they can curve in either direction. Each curve subtends 45 degrees.

Exercise - 2

3.27 n vehicles occupy squares $(1, 1)$ through $(n, 1)$ (i.e., the bottom row) of an $n \times n$ grid. The vehicles must be moved to the top row but in reverse order; so the vehicle i that starts in $(i, 1)$ must end up in $(n-i+1, n)$. On each time step, every one of the n vehicles can move one square up, down, left, or right, or stay put; but if a vehicle stays put, one other adjacent vehicle (but not more than one) can hop over it. Two vehicles cannot occupy the same square.

a. Calculate the size of the state space as a function of n.

b. Calculate the branching factor as a function of n.

c. Suppose that vehicle i is at (x_i, y_i); write a nontrivial admissible heuristic h_i for the number of moves it will require to get to its goal location $(n - i + 1, n)$, assuming no other vehicles are on the grid.

d. Which of the following heuristics are admissible for the problem of moving all n vehicles to their destinations? Explain.

(i) $\sum_{i=1}^{n} h_i$.

(ii) $\max\{h_1, \ldots, h_n\}$.

(iii) $\min\{h_1, \ldots, h_n\}$.

Local Search

Classical search:

- assumptions: observable, deterministic, known environment
- solution: a sequence of actions
- Optimization problems/constraint satisfaction problems – path is irrelevant

ex: 8 queens problem, IC design, factory floor layout, job shop scheduling, telecom network optimization, portfolio management, …

Local Search Algorithms(LSA)

- use a single current node (rather than multiple paths)
- generally move only to neighbours of that node
- paths followed by search not retained
- use little memory
- not systematic
- often finds reasonable solutions in large/infinite state spaces

Local Search

Local search for

finding goals

solving optimization problems: find the best state according to an objective function

A complete LSA finds a goal if one exits

An optimal LSA always finds a global max/min

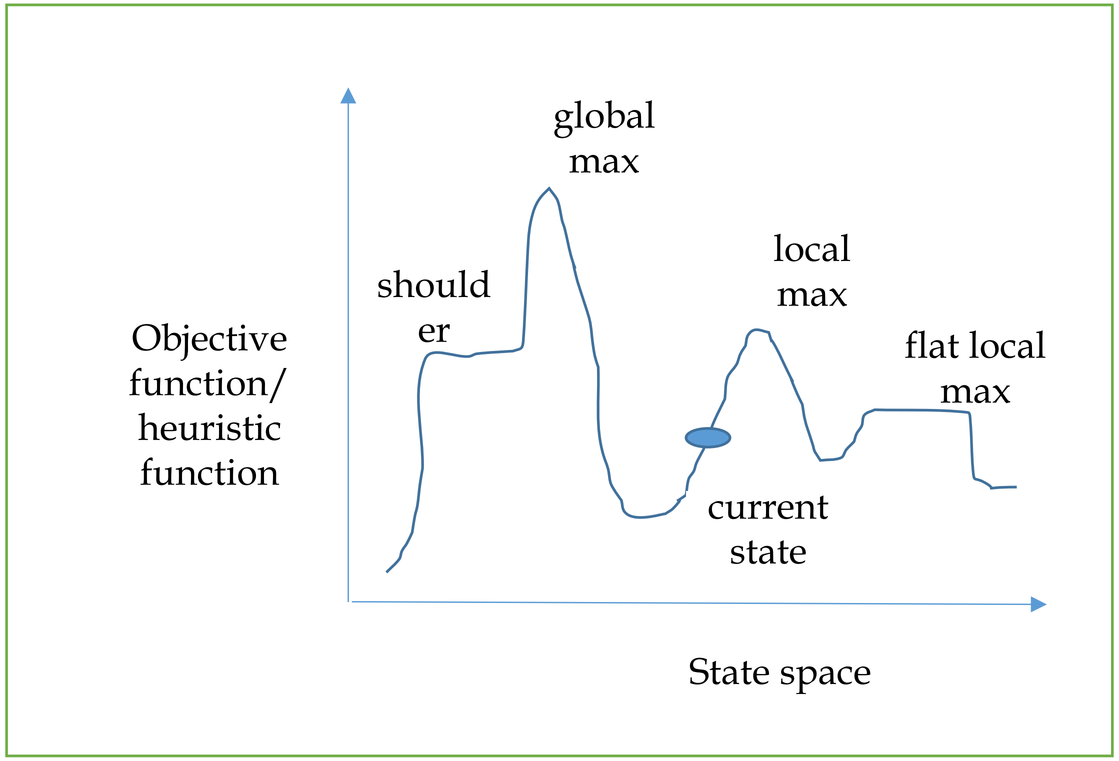
Hill climbing (steepest ascent)

Moves in the direction of increasing value of objective function

Local maxima : peak that is higher than its neighbouring states but lower than the global maximum

Ridges: sequence of local maxima

Plateaux: flat area

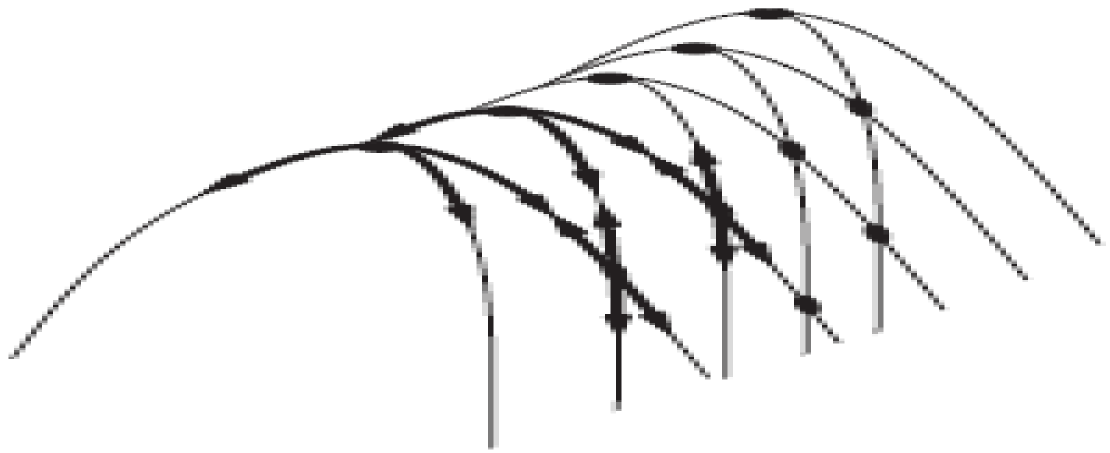


Hill Climbing - Ridge

Ex: why ridges cause difficulties for hill climbing.

The grid of states(dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima that are not directly connected to each other.

From each local maximum, all the available actions point downhill.



Hill climbing search

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ←MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ←a highest-valued successor of current
        If neighbor.VALUE ≤ current.VALUE then return current.STATE
        current←neighbour
```

hill-climbing search algorithm,   most basic local search technique.

At each step the current node is replaced by the best neighbor - the neighbor with the highest VALUE,

(if a heuristic cost estimate h is used, we would find the neighbor with the lowest h)

Hill climbing search (steepest Ascent version):

- does not maintain a search tree
- data structure for the current node – records the state and the current value of the objective function
- does not look ahead beyond the immediate neighbors of the current state

The loop continuously moves uphill – in the direction of increasing value. Terminates when it reaches peak, where no neighbor has a higher value.

Also called greedy local search as it grabs a good neighbor state without thinking ahead about where to go next.

Hill Climbing

Hill Climbing is NOT complete.

Hill Climbing is NOT optimal.

- Why use local search?

Low memory requirements –usually constant

Effective –Can often find good solutions in extremely large state spaces

Randomized variants of hill climbing can solve many of the drawbacks in practice.

- Many variants of hill climbing have been invented.

Ex: 8 queens problem

use of complete state formulation –

each state has 8 queens on the board, one per column

successors of a state are all possible states generated by moving a single queen to another square in the same column

each state has 8 x 7 successors (N(N-1) successors)

heuristic cost function h1: number of pairs of queens that are attacking each other

global minimum of h1 = 0, occurs only at perfect solutions

objective function for optimization = (N(N-1)/2) – L (N(N-1)/2: number of number of pairs of queens, L: number of pairs of queens that are attacking each other. We need to maximize the Objective function

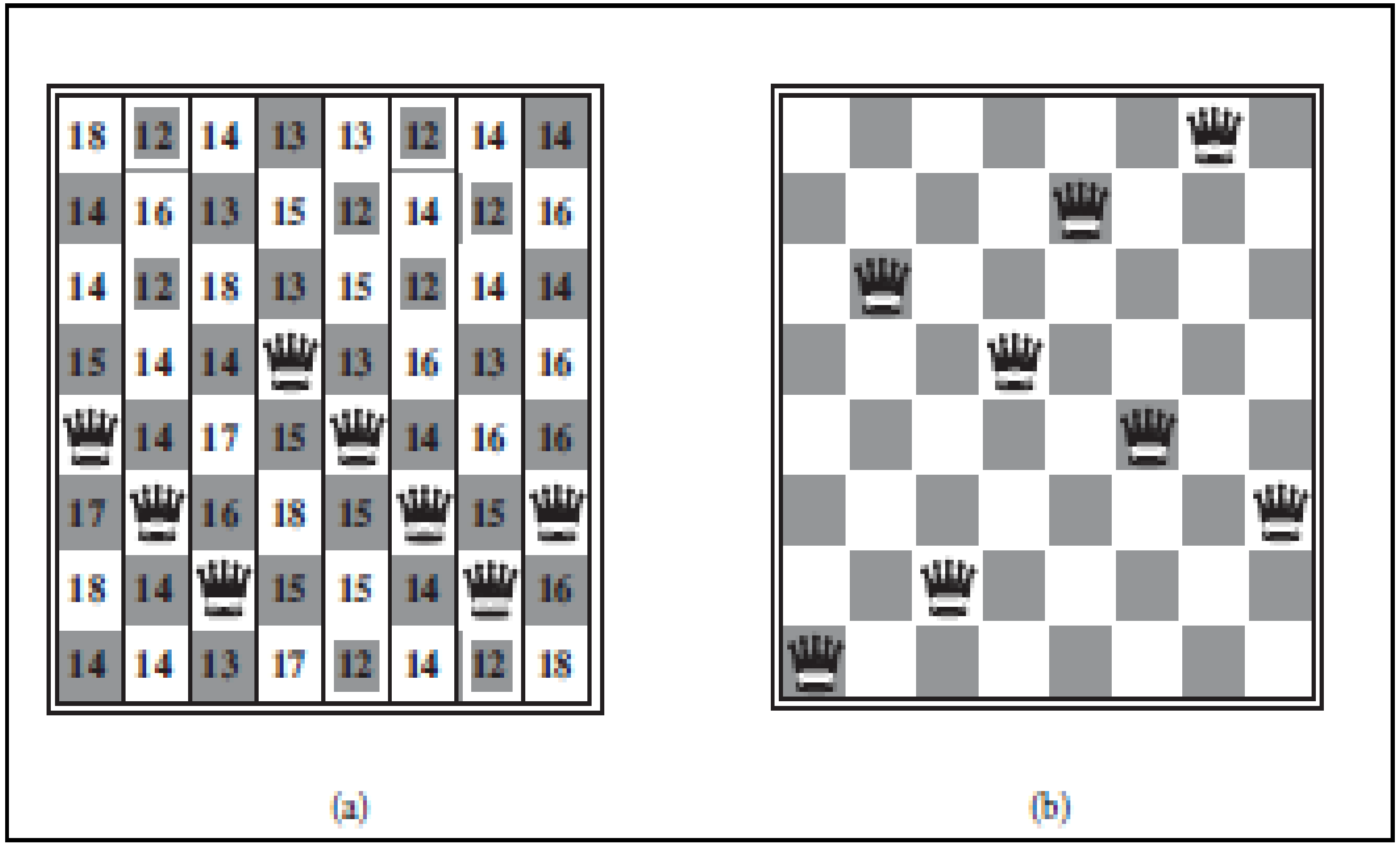
Starting from a randomly generated 8-queens initial state -

With hill climbing (steepest ascent) 8 queens gets stuck 86% of the time, solving only 14% of the instances

But works quickly (4 steps on average when it succeeds and 3 steps when it gets stuck)

8 Queens Problem

- (a) An 8-queens state with heuristic cost estimate h=17,
- the value of h for each possible successor obtained by moving a queen within its column is shown
- The best moves are marked.
- (b) A local minimum in the 8-queens state space; the state has h=1
- but every successor has a higher cost.



(a) (b)

Stochastic Hill Climbing

Stochastic hill climbing chooses at random from among the uphill moves;

The **probability of selection** can vary with the ***steepness of the uphill move***

Stochastic hill climbing usually converges more slowly than steepest ascent, but in some state landscapes, it finds better solutions.

Stochastic hill climbing is NOT complete, but it may be less likely to get stuck.

Idea: Put random walk into greedy hill-climbing

At each step do one of the two

Greedy: With probability, p move to the neighbor with largest value

Random: With probability, 1-p move to a random neighbor

First-Choice Hill Climbing

First-choice hill climbing implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state.

Good strategy when a state has many of successors.

First-choice hill climbing is also NOT complete

Random-Restart Hill Climbing

Random-Restart Hill Climbing

conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

Trivially complete if infinite (or sufficiently many tries) are allowed –

will eventually generate a goal as initial state

If each hill-climbing search has a probability p of success,

then the expected number of restarts required is 1/p.

The success of hill climbing depends on the shape of the state-space landscape

If there are few local maxima and plateau, a good solution is found very quickly.

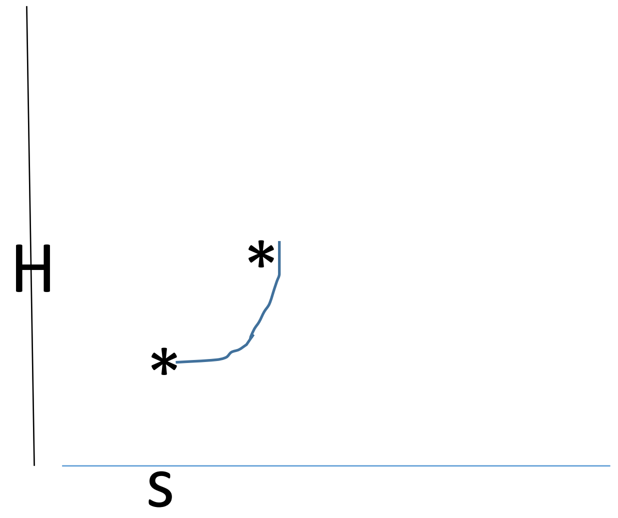
NP-hard problems typically have an exponential number of local maxima to get stuck on.

To avoid getting stuck in local minima

- Random-walk hill-climbing
- Random-restart hill-climbing
- Hill-climbing with both

At each step do one of the three
–Greedy: move to the neighbor with largest value
–Random Walk: move to a random neighbor
–Random Restart: Resample a new current state

- H = 0 goal



- 8 x 7

8 Queens Example

For 8-queens, random-restart hill climbing is very effective indeed.

Even for three million queens, the approach can find solutions in under a minute.

- For 8-queens instances with no sideways moves allowed, $p \approx 0.14$,

  so we need roughly 7 iterations to find a goal (6 failures and 1 success).

  No. of steps 22 ($4 + (.86/0.14) \times 3 \sim 22$)

When side ways move is allowed up to 100 moves, algorithm solves 94% instances.

  But takes roughly 21 steps for each successful instance and 64 for each failure

Expected number of steps = cost of one successful iteration + $(1-p)/p$ times cost of failure.

$= 1 \times 21 + (0.06/0.94) \times 64$

~ 25

Complexity Analysis – 8 Queens

time complexity for this algorithm can be divided into three parts:

1.Calculating Objective – The calculation of objective involves iterating through all queens on board and checking the no. of attacking queens, which is done by our calculate Objective function in $O(N^2)$ time.

2.Neighbor Selection and Number of neighbors – The description of neighbors in our problem gives a total of N(N-1) neighbors for the current state. The selection procedure is best fit and therefore requires iterating through all neighbors, which is again $O(N^2)$.

3.Search Space – Search space of our problem consists of a total of NN states, corresponding to all possible configurations of the N Queens on board. Note that this is after taking into account the additional constraint of one queen per column.

Therefore, the worst-case time complexity of our algorithm is O(NN). But, this worst-case occurs rarely in practice and thus we can safely consider it to be as good as any other algorithm there is for the N Queen problem. Hence, the effective time complexity consists of only calculating the objective for all neighbors up to a certain depth(no of jumps the search makes), which does not depend on N. Therefore, if the depth of search is d then the time complexity is $O(N^2 * N^2 * d)$, which is $O(d*N^4)$.

Simulated Annealing

A hill climbing algorithm that never makes down hill moves towards states with lower value is guaranteed to be incomplete

A purely random walk - moving to a successor chosen uniformly at random from the set of successors is complete, but inefficient.

Combine- Simulated annealing does this hill climbing with random walk in some way

Annealing – process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low energy crystalline state.

Instead of picking up best move, picks up random move.

If the move improves the situation, it is always accepted

Else, the move is accepted with some probability less than 1.

The probability decreases exponentially with the badness of the move–

the amount by which the evaluation is worsened

The probability also decreases as the temperature goes down

(bad moves ae more likely to be allowed at the start, when T is high, and they become more unlikely as as T decreases)

If the schedule lowers T slow enough, the algorithm will find a global optimum with probability approaching 1

Used to solve VLSI layout problems, factory scheduling

Simulated Annealing

Simulated annealing incorporates sideways and downhill moves.

Downhill moves are accepted with a probability based on the size of the change in the objective function

with the worst moves becoming least likely.

The fraction of accepted local changes is controlled by a formal parameter, temperature

At high temperature almost any possible local move is accepted

When lowering the temperature parameter, fewer moves are accepted that worsen the objective value

the search will start resembling pure greedy strategy

The temperature starts high and is slowly lowered during the search process

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
        schedule , a mapping from time to "temperature"
    current ←MAKE-NODE(problem.INITIAL-STATE)
    for t = 1 to ∞ do
        T ←schedule(t )
        if T = 0 then return current
        next←a randomly selected successor of current
        $\Delta E$ ←next.VALUE – current.VALUE
        if $\Delta E > 0$ then current ←next
        else current ←next only with probability $e^{\Delta E/T}$
```

The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The schedule input determines the value of the temperature T as a function of time.

Local Beam Search

Keeping only one node in memory is an extreme reaction to memory problems.

Keep track of k states instead of one

- Initially: k randomly selected states
- Next: determine all successors of k states
- If any of successors is goal □ finished
- Else select k best from successors and repeat

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

Idea: Stochastic beam search

Choose k successors randomly, biased towards good ones

Genetic Algorithms

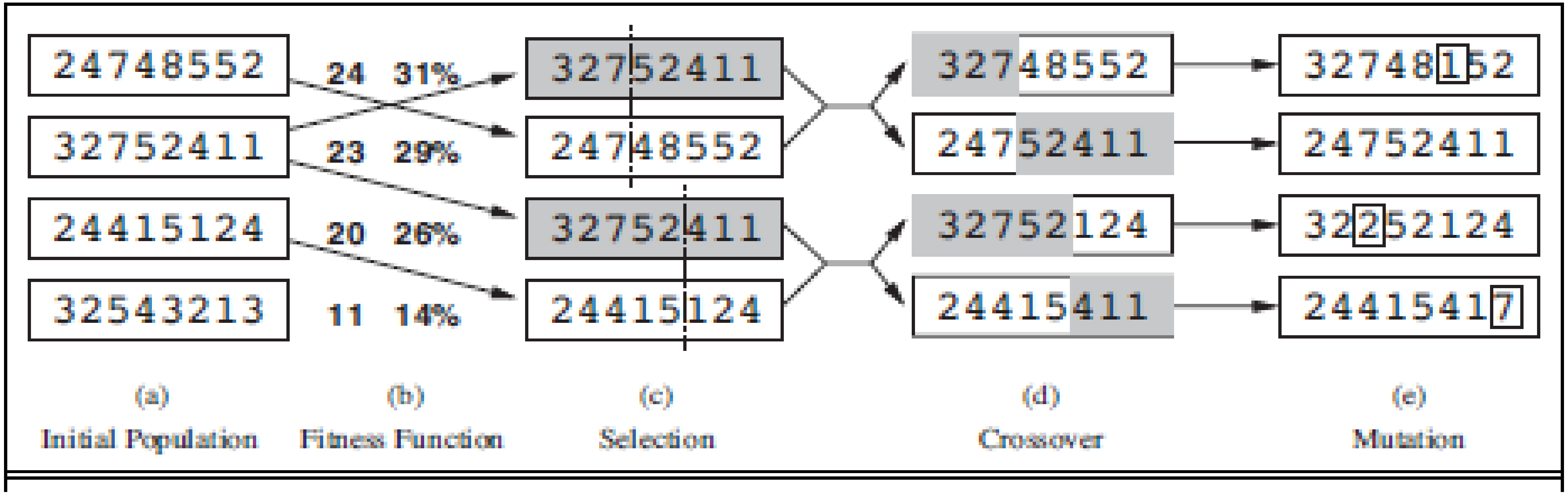
Evolution based Node Selection strategy in Local Search: successor is generated by combining two parent states

A state is represented as a string over a finite alphabet (e.g. binary)

Ex: 8-queens

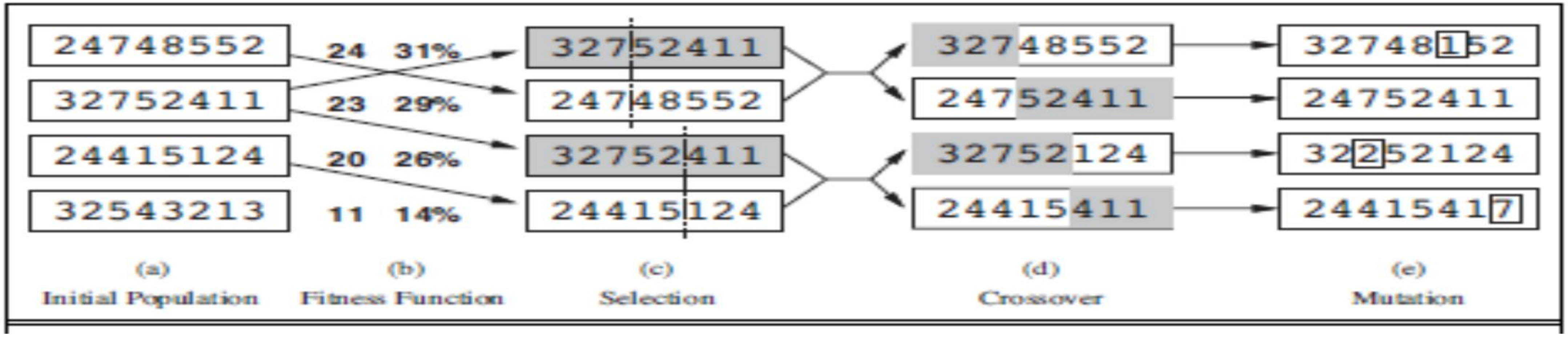
- State = position of 8 queens each in a column
- Start with k randomly generated states (population)
- Evaluation function (fitness function):
  - Higher values for better states.
  - Opposite to heuristic function, e.g., # non-attacking pairs in 8-queens (28 for solution)
- Produce the next generation of states by "simulated evolution"
  - Random selection
    (the probability of being chosen for reproducing is directly proportional to the fitness score)
  - Crossover
  - Random mutation
    (with a small independent probability. One digit was mutated in the first, third, and fourth offspring.
    In the 8-queens problem, this corresponds to choosing a queen at random and moving it to a random square in its column.)

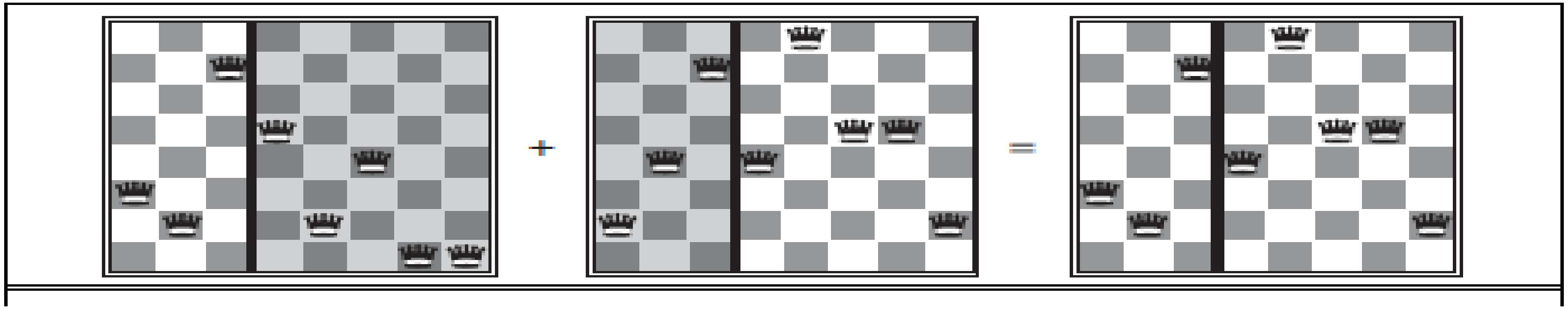
Genetic Algorithms



The genetic algorithm, illustrated for digit strings representing 8-queens states.
The initial population in (a)
is ranked by the fitness function in (b),
resulting in pairs for mating in (c).
They produce offspring in (d), which are subject to mutation in (e).

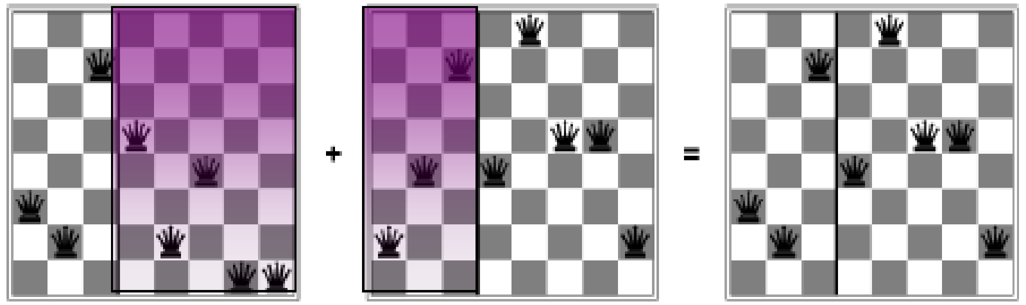
Genetic Algorithms





The 8-queens states corresponding to the first two parents in (c) and the first offspring in (d).
The shaded columns are lost in the crossover step and the unshaded columns are retained.

Genetic Algorithms



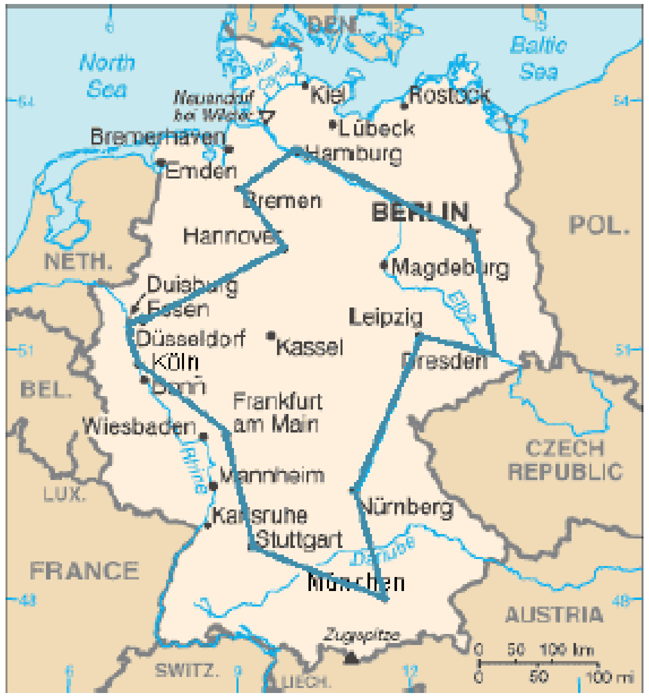
Has the effect of “jumping” to a completely different new part of the search space (quite non-local)

GA can be viewed as a strategy for running a large number of local searches in parallel .

Aside from local modifications, the states are also modified by a crossover,

in which the states from different local searches are combined to provide a better starting point for a new local search

Genetic Algorithms



Genetic Algorithm for TSP

Representation [9 3 4 0 1 2 5 7 6 8]

Mutation

- random swap (weak)
- random greedy swap: swap only when cost reduces
- exhaustive greedy swap

Genetic Algorithms

Greedy Crossover

```
select the first city of one parent,
compare the cities leaving that city in both parents,
    choose the closer one to extend the tour.
if one city has already appeared in the tour,
    choose the other city.
if both cities have already appeared,
    we randomly select a non-selected city.
```

Genetic Algorithms

Genetic algorithm is a variant of “stochastic beam search”

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
  - Difficult to replicate performance from one problem to another
- Lack of good empirical studies comparing to simpler methods
  - Useful on some (small?) set of problems but no convincing evidence that GAs are better than hill-climbing w/random restarts in general

Genetic Algorithms

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
    inputs: population, a set of individuals
        FITNESS-FN, a function that measures the fitness of an individual
    repeat
        new population ←empty set
        for i = 1 to SIZE(population) do
            x ←RANDOM-SELECTION(population, FITNESS-FN)
            y ←RANDOM-SELECTION(population, FITNESS-FN)
            child ←REPRODUCE(x , y)
            if (small random probability) then child ←MUTATE(child )
            add child to new population
        population ←new population
    until some individual is fit enough, or enough time has elapsed
    return the best individual in population, according to FITNESS-FN

function REPRODUCE(x , y) returns an individual
    inputs: x , y, parent individuals
    n←LENGTH(x ); c←random number from 1 to n
    return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```