Vegas and Monte carlo Optimization

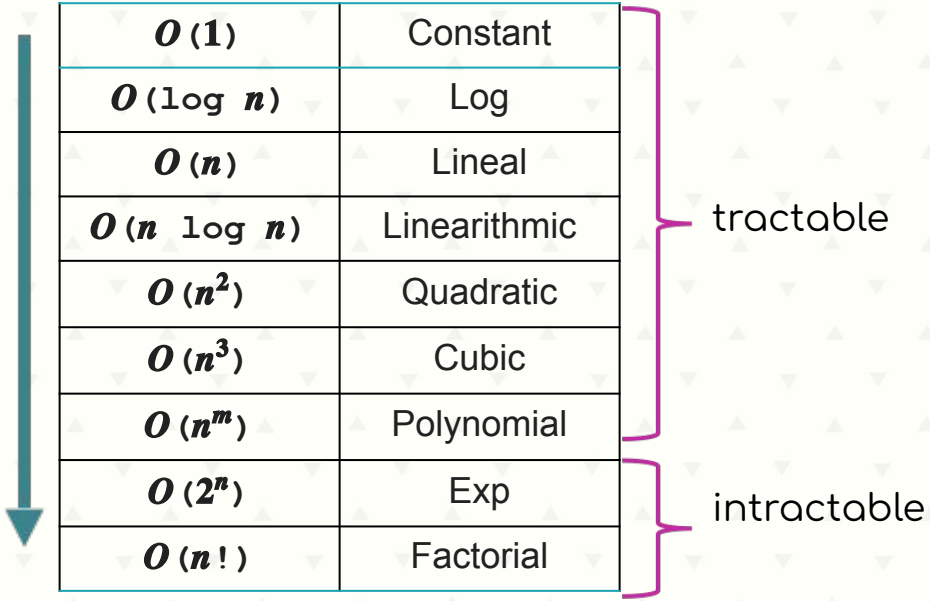
Analysis and Design of Advance Algorithms

Ing. Luis Salomon Flores Ugalde
Luis_SFU@tec.mx

Translation of
Dra. Valentina Navárez Terán
presentation

Complexity theory: P vs NP

The difficulty of a problem depends on the complexity of the most efficient algorithm that can solve it.



For intractable problems, it is believed that algorithms of less than exponential (exp-time) complexity cannot exist.

Class NP: a (magical and impossible) non-deterministic Turing machine would solve them in poly-time... but we believe that a deterministic one cannot

In other words, we believe that P != NP

Notes:

Deterministic: Always follows the same exact steps for a given input.

Non-deterministic: Hypothetical model that can instantly choose correct branches.

P: Problems solvable in polynomial time by deterministic algorithms.

NP: Problems whose solutions can be verified in polynomial time.

P ≠ NP: Widely believed; no polynomial-time solutions for NP-Complete problems.

Tractable: Solvable efficiently (poly-time).

Intractable: Requires exponential or factorial time.

Problems

Satisfiability: Core NP-Complete problem; truth assignment search.

Optimization: Find best value, not just a valid one.

Decision	Regarding verifying that a solution is the "correct" one, or whether a solution that meets certain characteristics exists	
Satisfaction	On finding an assignment of boolean variables that results in true	SAT
Optimization - Combinatorics - Numerical	On finding an assignment of variables (discrete / continuous) that results in minimum or maximum values	TSP Graph Coloring Knapsack

Why are they hard?

In general, because they have so many possible solutions, and we can't know which one is the best without having to review them all. And doing so would take millions of years...

Optimization Problems

You can understand them as generalizations of decision problems.

Highly relevant due to their practical applications:

- Routes and schedules
- Resource management
- Infrastructure construction
- Protein sequencing
- Epidemic studies
- Calibration of AI models (parameters, weights in neural networks, etc.)

PO and NPO Problems:

Similar to P and NP, but for optimization problems instead of decision problems.

NP-Hard Problems:

Remember that NP-Complete problems are decision problems in NP that are all equivalent to each other? NP-Hard problems are in NPO and are equivalent to each other.

NPO: Optimization analog of NP; solutions verifiable in polynomial time.
NP-Hard: At least as hard as NP-Complete; often harder.
PO: Optimization problems solvable efficiently.

Terminology: Objective Function

$f(x)$ Is the objective Function

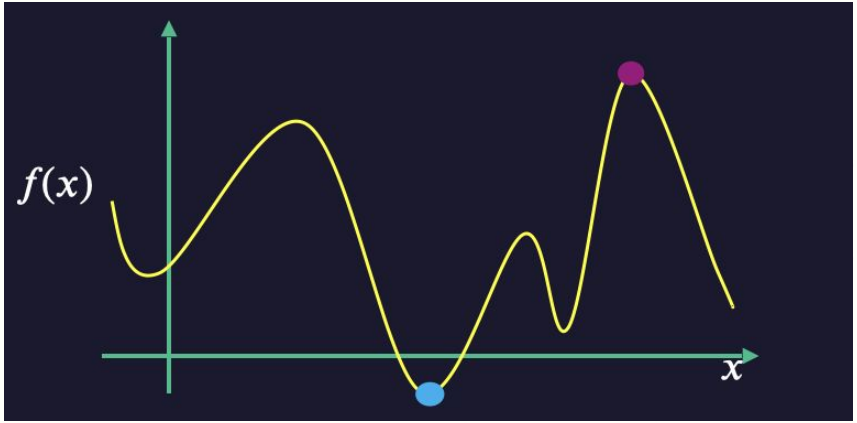
Optimization problems are approached as functions of the variables involved.

Optimizing means finding values for the variables that result in the minimum (or maximum) of the objective function.

Maximization problems: the optimum is the minimum. Examples include TSP.

Minimization problems: the optimum is the maximum. Examples include knapsack.

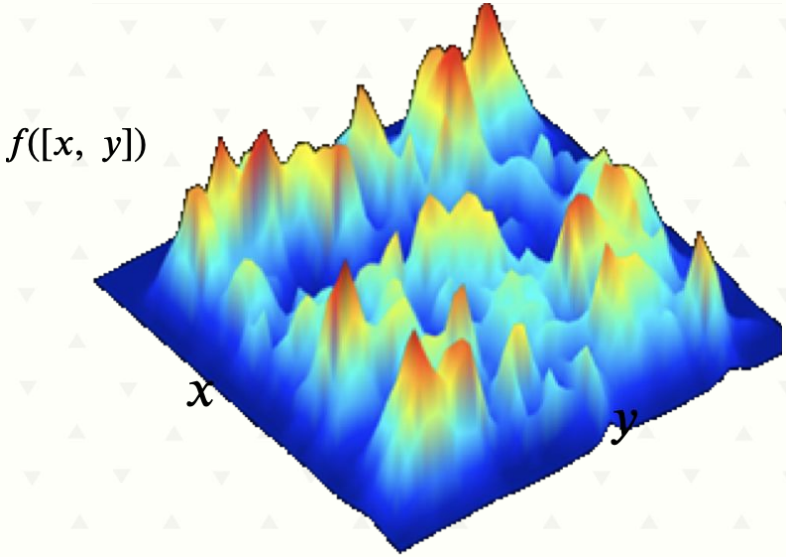
"Minimize the loss (lost value) by not choosing items."



Each value of **x** is a solution **(candidate)** to the problem.

If the objective function involves only one variable, we can visualize it. But interesting problems involve multiple variables.

Why is optimization difficult?



Esto es una visualización del fitness landscape: las soluciones, su distribución y calidad dada la función objetivo

- What happens if the problem involves more than one variable?
- How difficult is it to find optima?
- How can we be sure that a solution is optimal?

Many optimization problems are NP-hard.

Verifying the optimality of a solution cannot be done in polynomial time (if we assume that NP != P).

Optimization is the field of computer science concerned with how to design algorithms to find optima (and why settle for algorithms that can't ☹).

Objective function Knapsack

Given n objects, where:

- v_i represents the value of object i, with 1<= i <=n
- w_i represents the weight

One solution X is represented as:

$x = [x_1, x_2, x_3, \ldots . x_n]$

Subject to $x_i \in \{0,1\}$ 1<=i<=n

If a variable can only take one value within a set of options, it is specified as follows. Here, x_i can be either 0 or 1

The ranges go from 1 to n, but from 0 to a n-1 (inclusive) when programmed in code

v_i is the i-th element of a vector v

w_i is the i-th element of a vector w.

In code, this means v[i], and w[i]

Objective function Knapsack

Given n objects, where:

- v_i represents the value of object i, with 1<= i <=n
- w_i represents the weight

One solution X is represented as:

$x = [x_1, x_2, x_3, \ldots . x_n]$

Subject to $x_i \in \{0,1\}$ 1<=i<=n

The knapsack problem consists of maximizing:

$$f(X) = \sum_{i=1}^{n} v_i \, x_i$$

Subject to

$$\sum_{i=1}^{n} w_i \, x_i \leq W$$

That is, if x_i it is 0, its multiplication with v_i and w_i do not contribute to the summations

Random Algorithms Las Vegas and Monte Carlo

These are algorithms that produce solutions by making random decisions.

Las Vegas: Given enough attempts, they eventually produce an optimal/correct solution.

The execution time is random because it continues making attempts until it produces the correct solution. To know when to stop, they must have a way to recognize the optimal/correct solution.

Monte Carlo: It has limited attempts, and its solution can be suboptimal/partially correct. The execution time is fixed, and the quality of the solution is random.

Imagine you have a d20 (a 20-sided die).

With enough tries, you can roll a 20 (the best possible result).

But with only 10 tries, you might not be able to.

Random Algorithms Las Vegas and Monte Carlo

You already know an algorithm of this type: quicksort.

Where is the randomness in quicksort?

In the choice of the pivot (random with respect to the unknown order of the input values).

With enough iterations($O(n^2)$), you can get the correct solution (the sorted elements).

With the right input values, it might only take a few iterations. O(nlogn)

In the sorting problem, it's easy to recognize when the elements are already sorted.

What sort of random algorithm is Quicksort?

Random Algorithms Las Vegas and Monte Carlo

You already know an algorithm of this type: quicksort.

Where is the randomness in quicksort?

In the choice of the pivot (random with respect to the unknown order of the input values).

With enough iterations($O(n^2)$), you can get the correct solution (the sorted elements).

With the right input values, it might only take a few iterations. O(nlogn)

In the sorting problem, it's easy to recognize when the elements are already sorted.

What sort of random algorithm is Quicksort?

	Las Vegas	Monte Carlo
Solution Quality	Óptima	Random
Stopping criterion	Having produced the correct solution	Fixed number of iterations
Execution Time	Random	Fixed

Decision and satisfiability problems

It is easy to recognize when a solution of cost k has been produced, or a solution that satisfies the constraints, respectively. For example, it is easy to recognize a valid solution for n-Queens.

Optimization problems

Recognizing the optimum is not easy (generally).

Although in some cases, there are mathematical formulas to determine the cost of the optimum (without generating it).

Random Algorithms Las Vegas and Monte Carlo

Why create random algorithms?

For large instances of NP-hard problems, which would not be possible to solve exactly.

Advanced search and optimization techniques use Monte Carlo processes.

- Local searches
- Evolutionary computation (genetic algorithms and similar)
- Simulated annealing
- Swarm intelligence
- and many others

Relationship with heuristics:

A **heuristic** is **an intuitive and logical strategy for creating a solution to a problem.**

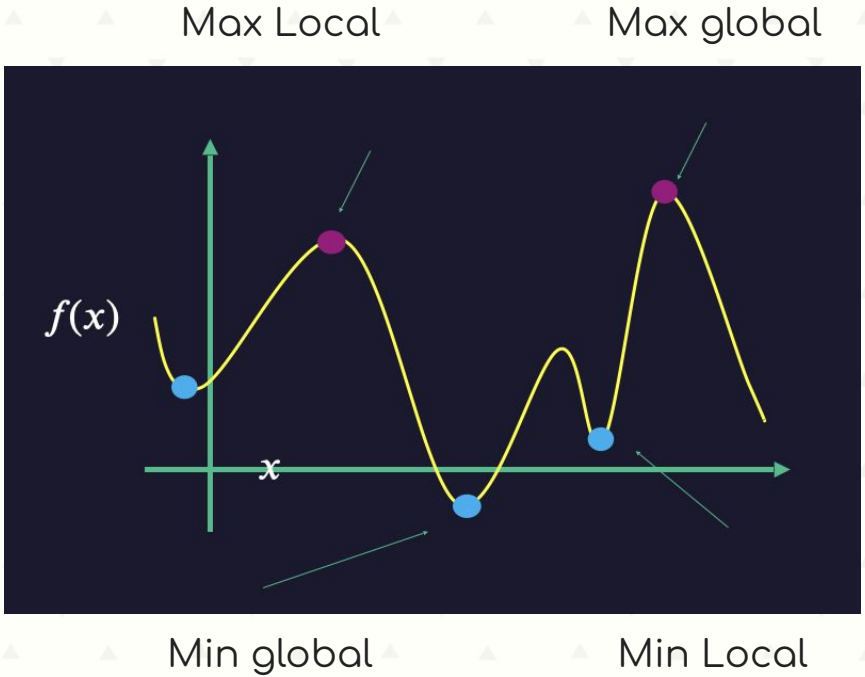
Creating random solutions is a form of heuristic. For example, choosing a random set of objects in Knapsack.

Optima vs Local Optima

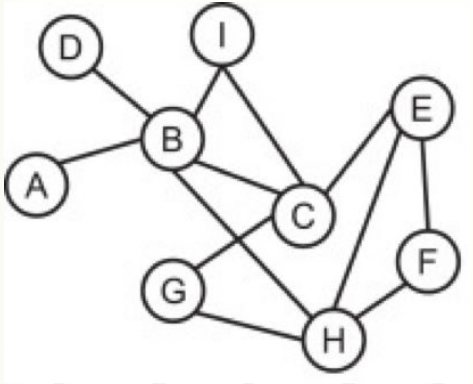
But that doesn't guarantee optima...

No, but somewhat more sophisticated heuristics can guarantee local optima.

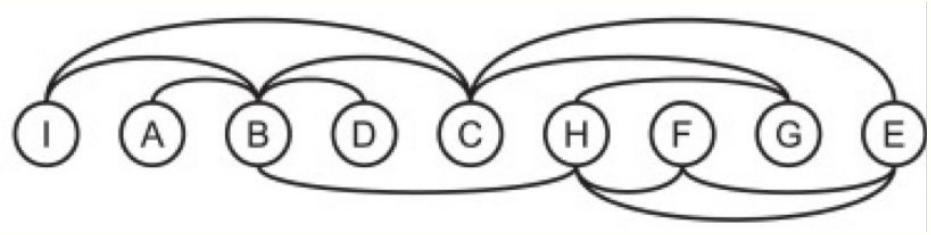
A local optimum is the best solution in its neighborhood, made up of other solutions, in its immediate vicinity.



Bandwidth Minimization



In this problem, you have an undirected, unweighted graph. Imagine you rearrange its nodes linearly.



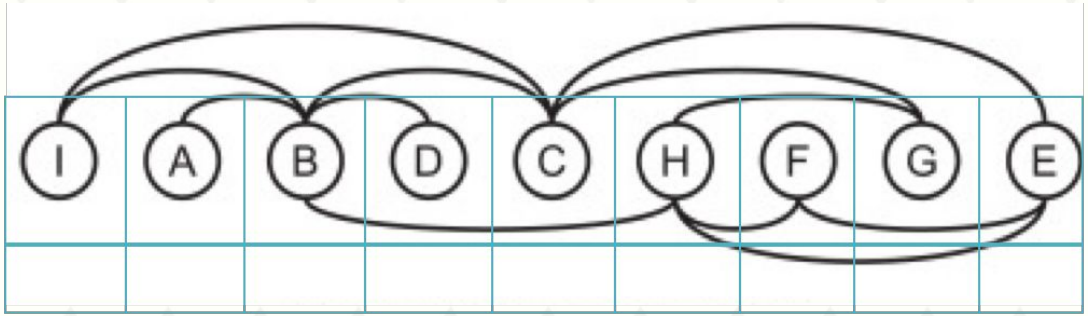
edge	distance
A,B	
B,C	
B,D	
B,H	3
B,I	
C,E	
C,G	
E,F	
E,H	
F,H	
G,H	

The bandwidth (cost) of this solution is the maximum distance between adjacent nodes in the original graph.

The problem is to determine how to achieve the lowest possible bandwidth. In other words, the best way to rearrange the graph.

Nodes

Labels



To calculate distances, we can assign each node a unique label, from **0 to n-1.**

Then, the possible solutions are different labels.

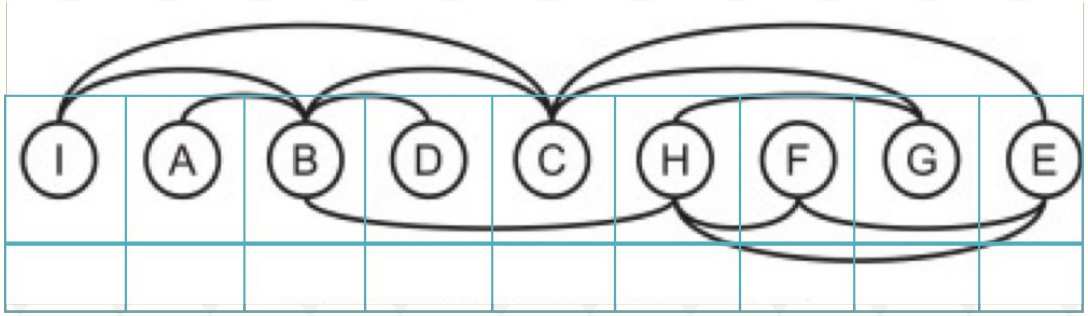
The distance between two nodes is the absolute difference between the values of their labels.

edge	distance
A,B	
B,C	
B,D	
B,H	3
B,I	
C,E	
C,G	
E,F	
E,H	
F,H	
G,H	

Activity 9.1

Nodes

Labels



edge	distance
A,B	
B,C	
B,D	
B,H	
B,I	
C,E	
C,G	
E,F	
E,H	
F,H	
G,H	

Program: Create an algorithm that produces 100 random solutions to this problem and evaluate its cost. Report the best and worst solutions.

Answer:

- How could this algorithm be improved to produce better solutions with the same number of iterations?
- How many possible solutions are there?
- What is the difficulty of the problem? (not of your algorithm)

Bandwidth minimization Search Space

Search space:

All possible solutions, distributed in an n-dimensional space

The size of the search space is the number of possible candidate solutions

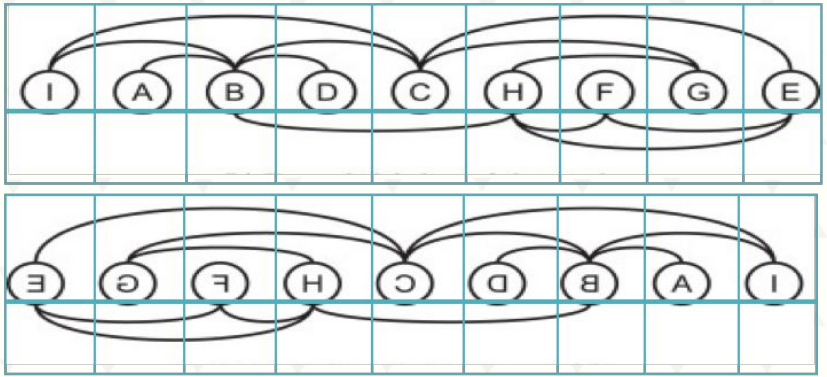
What is the size of the search space for the bandwidth problem?

By representing the solutions as **permutations** and considering the **isomorphism**, where each solution is equal to its inverse

n!/2

Isomorphic solutions: distinct solutions that are equivalent in all the components that determine their cost.

For example, for both solutions, **all distances** between pairs of adjacent nodes are equal.



Random sampler vs random walk

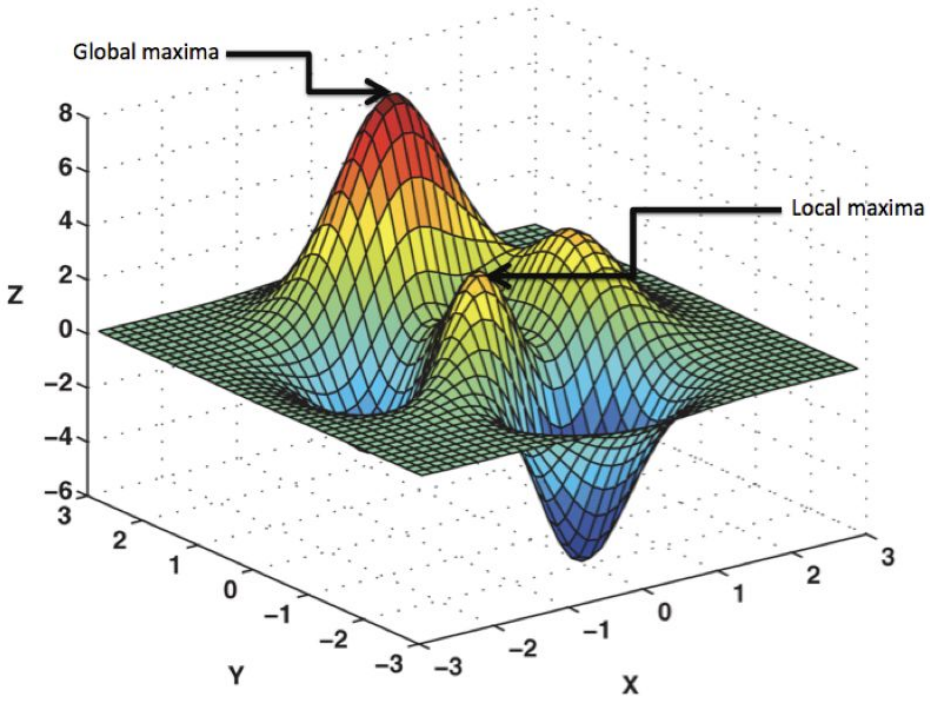
Random sampler: an algorithm that generates independent random solutions (task 5.1)

How can we create better algorithms than the random sampler?

What if, instead of trying to obtain decent solutions by chance, we try to "walk" from the first solution produced?

Problem: Where to move?

The simplest way to decide is randomly. This is a random walk.



Random Walk

What does "walking" consist of? Applying an operator to an existing solution, obtaining a new solution that replaces it, repeatedly.

What is an operator?

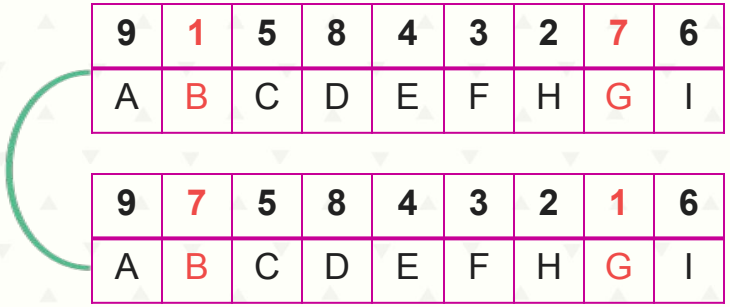
It's the method for obtaining a new solution by randomly modifying one of its elements.

It depends on the problem and how the solutions are represented.

```
S = initial random solution

S_best = S

for i = 1 to steps

    S′ = operator(S)

    S <- S′

    if S′ is better than S_best

      S_best <- S′

Return S_best
```

Operators

Random 2-swap: swaps 2 random elements of a base solution.
The result is a solution that at distance 2 from the original.



9	1	5	8	4	3	2	7	6
A	B	C	D	E	F	H	G	I

9	7	5	8	4	3	2	1	6
A	B	C	D	E	F	H	G	I

Distance 2: They differ in 2 elements

Note that swapping different elements produces different solutions.

In total, there are n(n-1)/2 possible swaps, resulting in n(n-1)/2 neighboring solutions, each at a distance of 2 from the original solution.

These solutions constitute the neighborhood of the original solution.

Operators

There are many other operators.

Which one to use?
It depends on the solution representation.
The representation of solutions is called encoding.

What optimization problem typically uses Boolean encoding?

For example, in solutions represented as Boolean strings, the bit flipping operator can be applied.}

Bit flipping operator: changes a Boolean variable from 1 to 0 , from 0 to 1.

0	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**
0	1	2	3	4	5	6	7	8

Chosen Element

0	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**
0	1	2	3	4	5	6	7	8

Distance 1: They differ in 1 bit

Partial Evaluations

To make the evaluation of new solutions more efficient, it is advisable to evaluate incrementally.

That is, considering only the elements that were affected by the operator.

For example, in knapsack, if an item is added to (or removed from) the selected set, it simply adds (or subtracts) its value and weight from the cost of the base solution.

0	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**
0	1	2	3	4	5	6	7	8

0	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**
0	1	2	3	4	5	6	7	8

Activity 9.2

Create an algorithm that performs a random walk for the following problems:

- knapsack
- bandwidth sum (like bandwidth, but aiming to minimize the sum of distances)
- anti-bandwidth (like bandwidth, but aiming to maximize the minimum distance)

Run your program 50 times, with 100,000 iterations for each run.
Report the best solution and its cost, saving a file with all 50 solutions.